# Manual Scilab

# Manual Scilab

# Índice

I. Scilab ..... 1
abort ..... 2
add_demo ..... 3
ans ..... 4
argn ..... 5
backslash (\) ..... 6
banner ..... 7
boolean ..... 8
brackets ..... 9
break ..... 10
case ..... 11
clear ..... 12
clearfun ..... 13
clearglobal ..... 14
colon ..... 15
comma ..... 16
comments ..... 17
comp ..... 18
comparison ..... 19
continue ..... 21
debug ..... 22
delbpt ..... 23
dispbpt ..... 24
do ..... 25
dot ..... 26
edit ..... 27
else ..... 28
elseif ..... 29
empty ..... 30
end ..... 31
equal ..... 32
errcatch ..... 33
errclear ..... 34
error ..... 35
error_table ..... 36
evstr ..... 45
exists ..... 46
exit ..... 47
external ..... 48
extraction ..... 49
for ..... 52
format ..... 53
funcprot ..... 55
funptr ..... 56
getdebuginfo ..... 57
getmd5 ..... 58
getmemory ..... 59
getmodules ..... 60
getos ..... 61
getscilabmode ..... 62
getshell ..... 63
getvariablesonstack ..... 64
getversion ..... 65
global ..... 66
gstacksize ..... 67

hat ..... 68
ieee ..... 69
if then else ..... 70
insertion ..... 71
intppty ..... 75
inv_coeff ..... 76
iserror ..... 77
isglobal ..... 78
lasterror ..... 79
left ..... 80
less ..... 81
macr2lst ..... 82
macr2tree ..... 84
matrices ..... 85
matrix ..... 86
mode ..... 87
mtlb_mode ..... 88
names ..... 89
newfun ..... 90
null ..... 91
parents ..... 92
pause ..... 94
percent ..... 95
perl ..... 96
plus ..... 97
poly ..... 98
power ..... 99
predef ..... 100
quit ..... 101
quote ..... 102
rational ..... 103
readgateway ..... 104
resume ..... 105
return ..... 106
sciargs ..... 107
scilab ..... 108
select ..... 110
semicolon (;) ..... 111
setbpt ..... 112
sethomedirectory ..... 113
slash ..... 114
stacksize ..... 115
star ..... 116
startup ..... 117
symbols ..... 118
testmatrix ..... 119
then ..... 120
tilda ..... 121
try ..... 122
type ..... 123
typename ..... 124
user ..... 125
varn ..... 126
ver ..... 127
warning ..... 128
what ..... 129
where ..... 130
whereami ..... 131

while ..... 132
who ..... 133
who_user ..... 134
whos ..... 135
with_atlas ..... 136
with_gtk ..... 137
with_javasci ..... 138
with_macros_source ..... 139
with_module ..... 140
with_pvm ..... 141
with_texmacs ..... 142
with_tk ..... 143
II. Equações Diferenciais ..... 144
dae ..... 145
daeoptions ..... 150
dasrt ..... 152
dassl ..... 156
feval ..... 161
impl ..... 162
int2d ..... 164
int3d ..... 166
intc ..... 169
integrate ..... 170
intg ..... 172
intl ..... 174
ode ..... 175
ode_discrete ..... 180
ode_optional_output ..... 181
ode_root ..... 183
odedc ..... 185
odeoptions ..... 188
bvode ..... 189
III. Funções Elementares ..... 203
abs ..... 204
acos ..... 205
acosd ..... 206
acosh ..... 207
acoshm ..... 208
acosm ..... 209
acot ..... 210
acotd ..... 211
acoth ..... 212
acsc ..... 213
acscd ..... 214
acsch ..... 215
adj2sp ..... 216
amell ..... 217
and ..... 218
asec ..... 219
asecd ..... 220
asech ..... 221
asin ..... 222
asind ..... 223
asinh ..... 224
asinhm ..... 225
asinm ..... 226
atan ..... 227
atand ..... 229

atanh ..... 230
atanhm ..... 231
atanm ..... 232
base2dec ..... 233
bin2dec ..... 234
binomial ..... 235
bitand ..... 237
bitor ..... 238
bloc2exp ..... 239
bloc2ss ..... 242
cat ..... 245
ceil ..... 246
cell2mat ..... 247
cellstr ..... 248
char ..... 249
conj ..... 251
cos ..... 252
cosd ..... 253
cosh ..... 254
coshm ..... 255
cosm ..... 256
cotd ..... 257
cotg ..... 258
coth ..... 259
cothm ..... 260
csc ..... 261
cscd ..... 262
csch ..... 263
cumprod ..... 264
cumsum ..... 265
dec2bin ..... 266
dec2hex ..... 267
dec2oct ..... 268
delip ..... 269
diag ..... 270
diff ..... 271
double ..... 272
dsearch ..... 273
eval ..... 275
exp ..... 276
eye ..... 277
factor ..... 278
fix ..... 279
flipdim ..... 280
floor ..... 281
frexp ..... 282
gsort ..... 283
hex2dec ..... 285
imag ..... 286
imult ..... 287
ind2sub ..... 288
int ..... 289
intersect ..... 290
inttrap ..... 292
isdef ..... 293
isempty ..... 294
isequal ..... 295
isequalbitwise ..... 296

isinf ..... 297
isnan ..... 298
isreal ..... 299
kron ..... 300
lex_sort ..... 301
linspace ..... 302
log ..... 303
log10 ..... 304
log1p ..... 305
log2 ..... 306
logm ..... 307
logspace ..... 308
lstsize ..... 309
max ..... 310
maxi ..... 311
meshgrid ..... 312
min ..... 313
mini ..... 314
minus ..... 315
modulo ..... 316
ndgrid ..... 317
ndims ..... 319
nearfloat ..... 320
nextpow2 ..... 321
norm ..... 322
not ..... 323
number_properties ..... 324
oct2dec ..... 326
ones ..... 327
or ..... 328
pen2ea ..... 329
perms ..... 330
permute ..... 331
pertrans ..... 332
primes ..... 333
prod ..... 334
rand ..... 335
rat ..... 337
real ..... 338
resize_matrix ..... 339
round ..... 340
sec ..... 341
secd ..... 342
sech ..... 343
setdiff ..... 344
sign ..... 345
signm ..... 346
sin ..... 347
sinc ..... 348
sind ..... 349
sinh ..... 350
sinhm ..... 351
sinm ..... 352
size ..... 353
solve ..... 355
sort ..... 356
sp2adj ..... 358
speye ..... 359

spones ..... 360
sprand ..... 361
spzeros ..... 362
sqrt ..... 363
sqrtm ..... 364
squarewave ..... 365
ssrand ..... 366
sub2ind ..... 368
sum ..... 369
sysconv ..... 370
sysdiag ..... 372
syslin ..... 373
tan ..... 375
tand ..... 376
tanh ..... 377
tanhm ..... 378
tanm ..... 379
toeplitz ..... 380
trfmod ..... 381
trianfml ..... 382
tril ..... 383
trisolve ..... 384
triu ..... 385
typeof ..... 386
union ..... 388
unique ..... 389
vectorfind ..... 391
zeros ..... 392
& ..... 393
csgn ..... 394
isvector ..... 395
| ..... 396
IV. Funções ..... 397
add_profiling ..... 398
bytecode ..... 399
bytecodewalk ..... 400
deff ..... 401
exec ..... 402
execstr ..... 404
fun2string ..... 406
function ..... 407
functions ..... 409
genlib ..... 411
get_function_path ..... 412
getd ..... 413
head_comments ..... 414
lib ..... 415
librarieslist ..... 417
library ..... 418
libraryinfo ..... 419
listfunctions ..... 420
macro ..... 421
macrovar ..... 422
plotprofile ..... 423
profile ..... 424
recompilefunction ..... 425
remove_profiling ..... 426
reset_profiling ..... 427

showprofile ..... 428
varargin ..... 429
varargout ..... 430
whereis ..... 431
getf ..... 432
V. Arquivos : funções de Entrada/Saída ..... 434
chdir ..... 435
fileinfo ..... 436
get_absolute_file_path ..... 438
getrelativefilename ..... 439
newest ..... 440
sscanf ..... 441
basename ..... 442
copyfile ..... 443
createdir ..... 444
deletefile ..... 445
dir ..... 446
dirname ..... 447
dispfiles ..... 448
fileext ..... 449
fileparts ..... 450
filesep ..... 451
findfiles ..... 452
fprintf ..... 453
fprintfMat ..... 454
fscanf ..... 455
fscanfMat ..... 456
fullfile ..... 457
fullpath ..... 458
getdrives ..... 459
getlongpathname ..... 460
getshortpathname ..... 461
%io ..... 462
isdir ..... 463
isfile ..... 464
listfiles ..... 465
listvarinfile ..... 466
ls ..... 467
maxfiles ..... 468
mclearerr ..... 469
mclose ..... 470
mdelete ..... 471
meof ..... 472
merror ..... 473
mfprintf ..... 474
mscanf ..... 475
mget ..... 478
mgetl ..... 480
mgetstr ..... 481
mkdir ..... 482
mopen ..... 483
movefile ..... 486
mput ..... 487
mputl ..... 489
mputstr ..... 490
mseek ..... 491
mtell ..... 492
pathconvert ..... 493

pathsep ..... 494
pwd ..... 495
removedir ..... 496
rmdir ..... 497
save_format ..... 498
scanf ..... 501
scanf_conversion ..... 502
VI. Biblioteca de Gráficos ..... 504
GlobalProperty ..... 505
Graphics ..... 509
LineSpec ..... 515
Matplot ..... 517
Matplot1 ..... 520
Matplot_properties ..... 522
Sfgrayplot ..... 524
Sgrayplot ..... 526
addcolor ..... 529
alufunctions ..... 530
arc_properties ..... 531
autumncolormap ..... 533
axes_properties ..... 534
axis_properties ..... 542
bar ..... 545
barh ..... 547
barhomogenize ..... 549
bonecolormap ..... 551
captions ..... 552
champ ..... 554
champ1 ..... 556
champ_properties ..... 558
clear_pixmap ..... 560
clf ..... 561
color ..... 563
color_list ..... 564
colorbar ..... 585
colordef ..... 587
colormap ..... 588
Compound_properties ..... 589
contour ..... 590
contour2d ..... 593
contour2di ..... 595
contourf ..... 597
coolcolormap ..... 599
coppercolormap ..... 600
copy ..... 601
delete ..... 602
dragrect ..... 603
draw ..... 604
drawaxis ..... 605
drawlater ..... 607
drawnow ..... 609
edit_curv ..... 610
errbar ..... 611
eval3d ..... 612
eval3dp ..... 613
event handler functions ..... 614
fac3d ..... 617
fchamp ..... 618

fcontour ..... 620
fcontour2d ..... 621
fec ..... 622
fec_properties ..... 625
fgrayplot ..... 627
figure_properties ..... 628
fplot2d ..... 631
fplot3d ..... 632
fplot3d1 ..... 633
gca ..... 634
gce ..... 635
gcf ..... 636
gda ..... 637
gdf ..... 639
ged ..... 640
genfac3d ..... 641
geom3d ..... 642
get ..... 643
get_figure_handle ..... 645
getcolor ..... 646
getfont ..... 647
getlinestyle ..... 648
getmark ..... 649
getsymbol ..... 650
glue ..... 651
graduate ..... 652
graphics_entities ..... 653
graphics_fonts ..... 656
graycolormap ..... 658
grayplot ..... 659
grayplot_properties ..... 660
graypolarplot ..... 662
havewindow ..... 664
hist3d ..... 665
histplot ..... 666
hotcolormap ..... 668
hsv2rgb ..... 669
hsvcolormap ..... 671
is_handle_valid ..... 672
isoview ..... 673
jetcolormap ..... 674
label_properties ..... 675
legend ..... 677
legend_properties ..... 679
legends ..... 682
locate ..... 684
mesh ..... 685
milk_drop ..... 687
move ..... 688
name2rgb ..... 689
newaxes ..... 690
nf3d ..... 691
object_editor ..... 692
oceancolormap ..... 697
oldplot ..... 698
param3d ..... 699
param3d1 ..... 701
param3d_properties ..... 704

paramfplot2d ..... 706
pie ..... 707
pinkcolormap ..... 708
plot ..... 709
plot2d ..... 714
plot2d1 ..... 719
plot2d2 ..... 721
plot2d3 ..... 722
plot2d4 ..... 723
plot2d_old_version ..... 724
plot3d ..... 728
plot3d1 ..... 737
plot3d2 ..... 740
plot3d3 ..... 743
plotframe ..... 746
plzr ..... 748
polarplot ..... 749
polyline_properties ..... 751
rainbowcolormap ..... 755
rectangle_properties ..... 756
relocate_handle ..... 759
replot ..... 760
rgb2name ..... 761
rotate ..... 762
rotate_axes ..... 763
rubberbox ..... 764
sca ..... 765
scaling ..... 766
scf ..... 767
sd2sci ..... 768
sda ..... 769
sdf ..... 770
secto3d ..... 771
segs_properties ..... 772
set ..... 775
seteventhandler ..... 777
show_pixmap ..... 778
show_window ..... 779
springcolormap ..... 780
square ..... 781
stringbox ..... 782
subplot ..... 784
summercolormap ..... 785
surf ..... 786
surface_properties ..... 790
swap_handles ..... 794
text_properties ..... 796
title ..... 799
titlepage ..... 801
twinkle ..... 802
unglue ..... 803
unzoom ..... 804
whitecolormap ..... 805
winsid ..... 806
wintercolormap ..... 807
xarc ..... 808
xarcs ..... 809
xarrows ..... 810

xbasc ... 811
xbasr ... 812
xchange ... 813
xclear ... 814
xclick ... 815
xdel ... 817
xfarc ... 818
xfarcs ... 819
xfpoly ... 820
xfpolys ... 821
xfrect ... 823
xget ... 824
xgetech ... 826
xgetmouse ... 828
xgraduate ... 830
xgrid ... 831
xinfo ... 832
xlfont ... 833
xload ... 835
xname ... 836
xnumb ... 837
xpause ... 838
xpoly ... 839
xpolys ... 840
xrect ... 841
xrects ... 842
xrpoly ... 843
xsave ... 844
xsegs ... 845
xselect ... 846
xset ... 847
xsetech ... 850
xsetm ... 852
xstring ... 853
xstringb ... 854
xstringl ... 855
xtitle ... 856
zoom_rect ... 857
Math rendering in Scilab graphics ... 859
VII. Gráficos : exportando e imprimindo ... 861
driver ... 862
xend ... 863
xinit ... 864
xs2bmp ... 865
xs2emf ... 866
xs2eps ... 867
xs2fig ... 868
xs2gif ... 869
xs2jpg ... 870
xs2pdf ... 871
xs2png ... 872
xs2ppm ... 873
xs2ps ... 874
xs2svg ... 875
VIII. Booleanos ... 876
bool2s ... 877
find ... 878
IX. CACSD ... 879

black ..... 880
bode ..... 882
chart ..... 884
evans ..... 886
gainplot ..... 887
m_circle ..... 888
nyquist ..... 889
routh_t ..... 891
sgrid ..... 892
zgrid ..... 893
abcd ..... 894
abinv ..... 895
arhnk ..... 899
arl2 ..... 900
arma ..... 902
arma2p ..... 904
armac ..... 905
armax ..... 906
armax1 ..... 908
arsimul ..... 910
augment ..... 911
balreal ..... 913
bilin ..... 914
bstap ..... 915
cainv ..... 916
calfrq ..... 918
canon ..... 919
ccontrg ..... 921
cls2dls ..... 922
colinout ..... 923
colregul ..... 924
cont_frm ..... 925
cont_mat ..... 926
contr ..... 927
contrss ..... 929
copfac ..... 930
csim ..... 931
ctr_gram ..... 933
dbphi ..... 934
dcf ..... 935
ddp ..... 936
des2ss ..... 938
des2tf ..... 939
dhinf ..... 940
dhnorm ..... 943
dscr ..... 944
dsimul ..... 945
dt_ility ..... 946
dtsi ..... 947
equil ..... 948
equil1 ..... 949
feedback ..... 950
findABCD ..... 951
findAC ..... 954
findBD ..... 956
findBDK ..... 960
findR ..... 963
findx0BD ..... 966

flts ..... 969
fourplan ..... 972
frep2tf ..... 973
freq ..... 975
freson ..... 976
fspecg ..... 977
fstabst ..... 978
g_margin ..... 980
gamitg ..... 981
gcare ..... 982
gfare ..... 983
gfrancis ..... 984
gtild ..... 986
h2norm ..... 988
h_cl ..... 989
h_inf ..... 990
h_inf_st ..... 991
h_norm ..... 992
hankelsv ..... 993
hinf ..... 994
imrep2ss ..... 997
inistate ..... 998
invsyslin ..... 1000
kpure ..... 1001
krac2 ..... 1002
lcf ..... 1003
leqr ..... 1004
lft ..... 1006
lin ..... 1008
linf ..... 1010
linfn ..... 1011
linmeq ..... 1012
lqe ..... 1016
lqg ..... 1018
lqg2stan ..... 1019
lqg_ltr ..... 1021
lqr ..... 1022
ltitr ..... 1024
macglov ..... 1025
markp2ss ..... 1026
minreal ..... 1027
minss ..... 1028
mucomp ..... 1029
narsimul ..... 1030
nehari ..... 1031
noisegen ..... 1032
obs_gram ..... 1033
obscont ..... 1034
observer ..... 1036
obsv_mat ..... 1038
obsvss ..... 1039
p_margin ..... 1040
parrot ..... 1041
pfss ..... 1042
phasemag ..... 1043
ppol ..... 1044
prbs_a ..... 1045
projsl ..... 1046

reglin ..... 1047
repfreq ..... 1048
ric_desc ..... 1050
ricc ..... 1052
riccati ..... 1054
rowinout ..... 1055
rowregul ..... 1056
rtitr ..... 1057
sensi ..... 1060
show_margins ..... 1061
sident ..... 1062
sm2des ..... 1066
sm2ss ..... 1067
sorder ..... 1068
specfact ..... 1072
ss2des ..... 1073
ss2ss ..... 1074
ss2tf ..... 1076
st_ility ..... 1077
stabil ..... 1079
svplot ..... 1080
sysfact ..... 1082
syssize ..... 1083
tf2des ..... 1084
tf2ss ..... 1085
time_id ..... 1086
trzeros ..... 1088
ui_observer ..... 1090
unobs ..... 1092
zeropen ..... 1093
X. Estruturas de Dados ..... 1094
cell ..... 1095
definedfields ..... 1097
getfield ..... 1098
hypermat ..... 1099
hypermatrices ..... 1100
iscell ..... 1101
iscellstr ..... 1102
isstruct ..... 1103
list ..... 1104
lsslist ..... 1106
lstcat ..... 1107
mlist ..... 1108
rlist ..... 1110
setfield ..... 1111
struct ..... 1112
tlist ..... 1113
fieldnames ..... 1114
isfield ..... 1115
XI. Shell ..... 1116
clc ..... 1117
lines ..... 1118
prompt ..... 1119
tohome ..... 1120
XII. Console ..... 1121
console ..... 1122
XIII. Complementação ..... 1123
completion ..... 1124

XIV. Gerenciador de histórico ..... 1126
addhistory ..... 1127
displayhistory ..... 1128
gethistory ..... 1129
gethistoryfile ..... 1130
historymanager ..... 1131
historysize ..... 1132
loadhistory ..... 1133
removelinehistory ..... 1134
resethistory ..... 1135
saveafterncommands ..... 1136
saveconsecutivecommands ..... 1137
savehistory ..... 1138
sethistoryfile ..... 1139
XV. IGU ..... 1140
1. Tree ..... 1142
uiCreateNode ..... 1143
uiCreateTree ..... 1144
uiDisplayTree ..... 1145
uiDumpTree ..... 1146
about ..... 1147
addmenu ..... 1148
clipboard ..... 1150
close ..... 1151
delmenu ..... 1152
exportUI ..... 1153
figure ..... 1154
findobj ..... 1156
gcbo ..... 1157
getcallbackobject ..... 1158
getinstalledlookandfeels ..... 1159
getlookandfeel ..... 1160
getvalue ..... 1161
messagebox ..... 1163
printfigure ..... 1165
printsetupbox ..... 1166
progressionbar ..... 1167
root_properties ..... 1168
setlookandfeel ..... 1169
setmenu ..... 1171
toolbar ..... 1172
toprint ..... 1173
uicontrol ..... 1175
uigetcolor ..... 1180
uigetdir ..... 1182
uigetfile ..... 1183
uigetfont ..... 1185
uimenu ..... 1186
uiputfile ..... 1188
unsetmenu ..... 1189
usecanvas ..... 1190
waitbar ..... 1193
x_choices ..... 1194
x_choose ..... 1195
x_choose_modeless ..... 1196
x_dialog ..... 1197
x_matrix ..... 1198
x_mdialog ..... 1199

x_message_modeless ..... 1200
XVI. Link Dinâmico/incremental ..... 1201
call ..... 1202
G_make ..... 1205
VCtoLCCLib ..... 1206
addinter ..... 1207
c_link ..... 1210
chooselcccompiler ..... 1211
configure_lcc ..... 1212
configure_ifort ..... 1213
configure_msvc ..... 1214
dllinfo ..... 1215
findlcccompiler ..... 1216
findmsifortcompiler ..... 1217
findmsvccompiler ..... 1218
fort ..... 1219
getdynlibext ..... 1221
haveacompiler ..... 1222
ilib_build ..... 1223
ilib_compile ..... 1231
ilib_for_link ..... 1232
ilib_gen_Make ..... 1234
ilib_gen_cleaner ..... 1236
ilib_gen_gateway ..... 1237
ilib_gen_loader ..... 1239
ilib_mex_build ..... 1240
ilib_verbose ..... 1241
link ..... 1243
ulink ..... 1245
with_lcc ..... 1246
XVII. Inteiros ..... 1247
iconvert ..... 1248
int8 ..... 1249
inttype ..... 1250
XVIII. Interpolação ..... 1251
bsplin3val ..... 1252
cshep2d ..... 1254
eval_cshep2d ..... 1256
interp ..... 1258
interp1 ..... 1261
interp2d ..... 1263
interp3d ..... 1265
interpln ..... 1267
intsplin ..... 1268
linear_interpn ..... 1269
lsq_splin ..... 1273
smooth ..... 1275
splin ..... 1276
splin2d ..... 1279
splin3d ..... 1282
XIX. Funções de Entrada/Saída ..... 1284
file ..... 1285
getenv ..... 1288
getio ..... 1289
getpid ..... 1290
getscilabkeywords ..... 1291
halt ..... 1292
host ..... 1293

input ..... 1294
load ..... 1295
oldload ..... 1296
oldsave ..... 1297
read ..... 1298
read4b ..... 1300
readb ..... 1301
readc_ ..... 1302
save ..... 1303
setenv ..... 1305
unix ..... 1306
unix_g ..... 1307
unix_s ..... 1308
unix_w ..... 1309
unix_x ..... 1310
writb ..... 1311
write ..... 1312
write4b ..... 1313
XX. Funções de Saída ..... 1314
disp ..... 1315
print ..... 1316
printf ..... 1317
printf_conversion ..... 1318
sprintf ..... 1321
ssprint ..... 1322
diary ..... 1323
mprintf ..... 1326
msprintf ..... 1327
prettyprint ..... 1328
XXI. Intersci ..... 1330
intersci ..... 1331
XXII. JVM ..... 1332
javaclasspath ..... 1333
javalibrarypath ..... 1334
jre_path ..... 1335
system_getproperty ..... 1336
system_setproperty ..... 1339
with_embedded_jre ..... 1340
XXIII. Álgebra Linear ..... 1341
aff2ab ..... 1342
balanc ..... 1344
bdiag ..... 1345
chfact ..... 1346
chol ..... 1347
chsolve ..... 1348
classmarkov ..... 1349
cmb_lin ..... 1350
coff ..... 1351
colcomp ..... 1352
companion ..... 1353
cond ..... 1354
det ..... 1355
eigenmarkov ..... 1356
ereduc ..... 1357
expm ..... 1358
fstair ..... 1359
fullrf ..... 1361
fullrfk ..... 1362

genmarkov ..... 1363
givens ..... 1364
glever ..... 1365
gschur ..... 1366
gspec ..... 1367
hess ..... 1368
householder ..... 1369
im_inv ..... 1370
inv ..... 1371
kernel ..... 1372
kroneck ..... 1373
linsolve ..... 1375
lsq ..... 1377
lu ..... 1379
lyap ..... 1380
nlev ..... 1381
orth ..... 1382
pbig ..... 1383
pencan ..... 1384
penlaur ..... 1385
pinv ..... 1386
polar ..... 1387
proj ..... 1388
projspec ..... 1389
psmall ..... 1390
qr ..... 1391
quaskro ..... 1393
randpencil ..... 1395
range ..... 1396
rank ..... 1397
rankqr ..... 1398
rcond ..... 1400
rowcomp ..... 1401
rowshuff ..... 1403
rref ..... 1404
schur ..... 1405
spaninter ..... 1409
spanplus ..... 1410
spantwo ..... 1411
spec ..... 1413
sqroot ..... 1416
squeeze ..... 1417
sva ..... 1418
svd ..... 1419
sylv ..... 1420
trace ..... 1421
XXIV. Localização ..... 1422
dgettext ..... 1423
getdefaultlanguage ..... 1424
getlanguage ..... 1425
gettext ..... 1426
LANGUAGE ..... 1427
setdefaultlanguage ..... 1428
setlanguage ..... 1429
XXV. Otimização e Simulação ..... 1430
2. Neldermead ..... 1432
fminsearch ..... 1433
neldermead ..... 1442

overview ... 1463
nmplot ... 1467
optimget ... 1478
optimplotfunccount ... 1480
optimplotfval ... 1481
optimplotx ... 1482
optimset ... 1483
3. Optimization base ... 1487
optimbase ... 1488
4. Optimization simplex ... 1502
optimsimplex ... 1503
NDcost ... 1519
aplat ... 1522
datafit ... 1523
derivative ... 1526
fit_dat ... 1530
fsolve ... 1532
karmarkar ... 1535
leastsq ... 1536
list2vec ... 1542
lmisolver ... 1543
lmitool ... 1545
lsqrsolve ... 1546
numdiff ... 1549
optim ... 1551
qld ... 1565
qp_solve ... 1568
qpsolve ... 1571
quapro ... 1574
readmps ... 1575
recons ... 1579
semidef ... 1580
vec2list ... 1583
XXVI. Sobrecarga ... 1584
overloading ... 1585
XXVII. Polinômios ... 1587
bezout ... 1588
clean ... 1589
cmndred ... 1590
coeff ... 1591
coffg ... 1592
colcompr ... 1593
degree ... 1594
denom ... 1595
derivat ... 1596
determ ... 1597
detr ... 1598
diophant ... 1599
factors ... 1600
gcd ... 1602
hermit ... 1603
horner ... 1604
hrmt ... 1605
htrianr ... 1606
invr ... 1607
lcm ... 1608
lcmdiag ... 1609
ldiv ... 1610

numer ..... 1611
pdiv ..... 1612
pol2des ..... 1613
pol2str ..... 1614
polfact ..... 1615
residu ..... 1616
roots ..... 1617
rowcompr ..... 1618
sfact ..... 1619
simp ..... 1620
simp_mode ..... 1621
sylm ..... 1622
systmat ..... 1623
XXVIII. Processamento de Sinais ..... 1624
5. How to ..... 1626
How to design an elliptic filter ..... 1627
Signal ..... 1635
analpf ..... 1639
bilt ..... 1641
buttmag ..... 1643
casc ..... 1644
cepstrum ..... 1645
cheb1mag ..... 1646
cheb2mag ..... 1647
chepol ..... 1648
convol ..... 1649
corr ..... 1651
cspect ..... 1654
czt ..... 1657
detrend ..... 1659
dft ..... 1661
ell1mag ..... 1662
eqfir ..... 1663
eqiir ..... 1664
faurre ..... 1665
ffilt ..... 1666
fft ..... 1667
fft2 ..... 1669
fftshift ..... 1671
filt_sinc ..... 1673
filter ..... 1674
find_freq ..... 1675
findm ..... 1676
frfit ..... 1677
frmag ..... 1678
fsfirlin ..... 1679
group ..... 1681
hank ..... 1682
hilb ..... 1683
hilbert ..... 1684
iir ..... 1685
iirgroup ..... 1686
iirlp ..... 1687
intdec ..... 1688
jmat ..... 1689
kalm ..... 1690
lattn ..... 1691
lattp ..... 1692

lev ..... 1693
levin ..... 1694
lindquist ..... 1697
mese ..... 1698
mfft ..... 1699
mrfit ..... 1700
%asn ..... 1701
%k ..... 1702
%sn ..... 1703
phc ..... 1704
pspect ..... 1706
remez ..... 1709
remezb ..... 1711
rpem ..... 1713
sincd ..... 1715
srfaur ..... 1716
srkf ..... 1718
sskf ..... 1719
syredi ..... 1720
system ..... 1722
trans ..... 1723
wfir ..... 1725
wiener ..... 1726
wigner ..... 1727
window ..... 1728
yulewalk ..... 1730
zpbutt ..... 1731
zpch1 ..... 1732
zpch2 ..... 1733
zpell ..... 1734
XXIX. Matrizes Esparsas ..... 1735
full ..... 1736
ludel ..... 1737
lufact ..... 1738
luget ..... 1740
lusolve ..... 1741
mtlb_sparse ..... 1742
nnz ..... 1743
sparse ..... 1744
spchol ..... 1745
spcompack ..... 1746
spget ..... 1748
gmres ..... 1749
pcg ..... 1751
qmr ..... 1755
XXX. Funções Especiais ..... 1757
besseli ..... 1758
beta ..... 1761
calerf ..... 1763
dlgamma ..... 1764
erf ..... 1765
erfc ..... 1766
erfcx ..... 1767
erfinv ..... 1768
gamma ..... 1769
gammaln ..... 1770
legendre ..... 1771
oldbesseli ..... 1774

XXXI. Cadeias de Caracteres (Strings) ..... 1777
ascii ..... 1778
blanks ..... 1779
code2str ..... 1780
convstr ..... 1781
emptystr ..... 1782
grep ..... 1783
isalphanum ..... 1784
isascii ..... 1785
isdigit ..... 1786
isletter ..... 1787
isnum ..... 1788
justify ..... 1789
length ..... 1790
part ..... 1791
regexp ..... 1792
sci2exp ..... 1793
str2code ..... 1794
strcat ..... 1795
strchr ..... 1796
strcmp ..... 1797
strcmpi ..... 1798
strcspn ..... 1799
strindex ..... 1800
string ..... 1802
strings ..... 1803
stripblanks ..... 1804
strncpy ..... 1805
strrchr ..... 1806
strrev ..... 1807
strsplit ..... 1808
strspn ..... 1809
strstr ..... 1810
strsubst ..... 1811
strtod ..... 1812
strtok ..... 1813
tokenpos ..... 1814
tokens ..... 1815
tree2code ..... 1816
XXXII. Cálculos Formais ..... 1817
addf ..... 1818
ldivf ..... 1819
mulf ..... 1820
rdivf ..... 1821
subf ..... 1822
XXXIII. Data e Hora ..... 1823
date ..... 1824
etime ..... 1825
getdate ..... 1826
tic ..... 1828
toc ..... 1829
calendar ..... 1830
clock ..... 1831
datenum ..... 1832
datevec ..... 1834
eomday ..... 1835
now ..... 1836
realtimeinit ..... 1837

sleep ..... 1838
timer ..... 1839
weekday ..... 1840
XXXIV. Estatística ..... 1841
cdfbet ..... 1842
cdfbin ..... 1843
cdfchi ..... 1844
cdfchn ..... 1845
cdff ..... 1846
cdffnc ..... 1847
cdfgam ..... 1848
cdfnbn ..... 1849
cdfnor ..... 1850
cdfpoi ..... 1851
cdft ..... 1852
center ..... 1853
wcenter ..... 1854
cmoment ..... 1855
correl ..... 1856
covar ..... 1857
ftest ..... 1858
ftuneq ..... 1859
geomean ..... 1860
harmean ..... 1861
iqr ..... 1862
labostat ..... 1863
mad ..... 1864
mean ..... 1865
meanf ..... 1866
median ..... 1867
moment ..... 1868
msd ..... 1869
mvvacov ..... 1870
nancumsum ..... 1871
nand2mean ..... 1872
nanmax ..... 1873
nanmean ..... 1874
nanmeanf ..... 1875
nanmedian ..... 1876
nanmin ..... 1877
nanstdev ..... 1878
nansum ..... 1879
nfreq ..... 1880
pca ..... 1881
perctl ..... 1882
princomp ..... 1883
quart ..... 1885
regress ..... 1886
sample ..... 1887
samplef ..... 1888
samwr ..... 1890
show_pca ..... 1891
st_deviation ..... 1892
stdevf ..... 1893
strange ..... 1894
tabul ..... 1895
thrownan ..... 1897
trimmean ..... 1898

variance ..... 1900
variancef ..... 1901
XXXV. ARnoldi PACKage ..... 1902
dnaupd ..... 1903
dneupd ..... 1911
dsaupd ..... 1918
dsaupd ..... 1926
znaupd ..... 1931
zneupd ..... 1939
XXXVI. Funções de Compatibilidade ..... 1945
asciimat ..... 1946
firstnonsingleton ..... 1947
makecell ..... 1948
mstr2sci ..... 1949
mtlb_0 ..... 1950
mtlb_a ..... 1951
mtlb_all ..... 1952
mtlb_any ..... 1953
mtlb_axis ..... 1954
mtlb_beta ..... 1955
mtlb_box ..... 1956
mtlb_close ..... 1957
mtlb_colordef ..... 1958
mtlb_conv ..... 1959
mtlb_cumprod ..... 1960
mtlb_cumsum ..... 1961
mtlb_dec2hex ..... 1962
mtlb_delete ..... 1963
mtlb_diag ..... 1964
mtlb_diff ..... 1965
mtlb_dir ..... 1966
mtlb_double ..... 1967
mtlb_e ..... 1968
mtlb_echo ..... 1969
mtlb_eval ..... 1970
mtlb_exist ..... 1971
mtlb_eye ..... 1972
mtlb_false ..... 1973
mtlb_fft ..... 1974
mtlb_fftshift ..... 1975
mtlb_find ..... 1976
mtlb_findstr ..... 1977
mtlb_fliplr ..... 1978
mtlb_fopen ..... 1979
mtlb_format ..... 1980
mtlb_fprintf ..... 1981
mtlb_fread ..... 1982
mtlb_fscanf ..... 1983
mtlb_full ..... 1984
mtlb_fwrite ..... 1985
mtlb_grid ..... 1986
mtlb_hold ..... 1987
mtlb_i ..... 1988
mtlb_ifft ..... 1989
mtlb_imp ..... 1990
mtlb_int16 ..... 1991
mtlb_int32 ..... 1992
mtlb_int8 ..... 1993

mtlb_is ..... 1994
mtlb_isa ..... 1995
mtlb_isfield ..... 1996
mtlb_isletter ..... 1997
mtlb_isspace ..... 1998
mtlb_l ..... 1999
mtlb_legendre ..... 2000
mtlb_linspace ..... 2001
mtlb_logic ..... 2002
mtlb_logical ..... 2003
mtlb_lower ..... 2004
mtlb_max ..... 2005
mtlb_min ..... 2006
mtlb_more ..... 2007
mtlb_num2str ..... 2008
mtlb_ones ..... 2009
mtlb_plot ..... 2010
mtlb_prod ..... 2011
mtlb_rand ..... 2012
mtlb_randn ..... 2013
mtlb_rcond ..... 2014
mtlb_realmax ..... 2015
mtlb_realmin ..... 2016
mtlb_repmat ..... 2017
mtlb_s ..... 2018
mtlb_setstr ..... 2019
mtlb_size ..... 2020
mtlb_sort ..... 2021
mtlb_strcmp ..... 2022
mtlb_strcmpi ..... 2023
mtlb_strfind ..... 2024
mtlb_strrep ..... 2025
mtlb_sum ..... 2026
mtlb_t ..... 2027
mtlb_toeplitz ..... 2028
mtlb_tril ..... 2029
mtlb_triu ..... 2030
mtlb_true ..... 2031
mtlb_uint16 ..... 2032
mtlb_uint32 ..... 2033
mtlb_uint8 ..... 2034
mtlb_upper ..... 2035
mtlb_var ..... 2036
mtlb_zeros ..... 2038
XXXVII. Interface Java ..... 2039
javasci.SciBoolean ..... 2040
javasci.SciBooleanArray ..... 2042
javasci.SciComplex ..... 2044
javasci.SciComplexArray ..... 2046
javasci.SciDouble ..... 2049
javasci.SciDoubleArray ..... 2051
javasci.SciInteger ..... 2053
javasci.SciIntegerArray ..... 2054
javasci.SciString ..... 2055
javasci.SciStringArray ..... 2057
javasci.Scilab ..... 2059
Compile and run with javasci ..... 2061
javasci ..... 2063

javasci FAQ ..... 2064
XXXVIII. Interface Maple ..... 2065
sci2map ..... 2066
XXXIX. Dicas de Conversão de Matlab para Scilab ..... 2067
About_M2SCI_tools ..... 2068
Contents ..... 2070
Cste ..... 2071
Equal ..... 2072
Funcall ..... 2073
Infer ..... 2074
Matlab-Scilab_character_strings ..... 2075
Operation ..... 2076
Type ..... 2077
Variable ..... 2078
get_contents_infer ..... 2079
m2scideclare ..... 2080
matfile2sci ..... 2082
mfile2sci ..... 2083
sci_files ..... 2086
translatepaths ..... 2088
XL. Interfaces com Tcl/Tk ..... 2089
ScilabEval ..... 2090
TCL_CreateSlave ..... 2092
TCL_DeleteInterp ..... 2093
TCL_EvalFile ..... 2094
TCL_EvalStr ..... 2096
TCL_ExistArray ..... 2098
TCL_ExistInterp ..... 2099
TCL_ExistVar ..... 2100
TCL_GetVar ..... 2101
TCL_GetVersion ..... 2103
TCL_SetVar ..... 2104
TCL_UnsetVar ..... 2106
TCL_UpVar ..... 2107
browsevar ..... 2108
config ..... 2109
editvar ..... 2110
tk_getdir ..... 2111
tk_savefile ..... 2112
winclose ..... 2113
winlist ..... 2114
XLI. Texmacs ..... 2115
pol2tex ..... 2116
texprint ..... 2117
XLII. Manipulação de Arquivos de Som ..... 2118
analyze ..... 2119
auread ..... 2120
auwrite ..... 2121
beep ..... 2122
lin2mu ..... 2123
loadwave ..... 2124
mapsound ..... 2125
mu2lin ..... 2126
playsnd ..... 2127
savewave ..... 2128
sound ..... 2129
soundsec ..... 2130
wavread ..... 2131

wavwrite ..... 2132
XLIII. Randlib ..... 2133
grand ..... 2134
XLIV. Ferramentas de Desenvolvimento ..... 2140
bench_run ..... 2141
tbx_build_cleaner ..... 2143
tbx_build_gateway ..... 2144
tbx_build_gateway_clean ..... 2145
tbx_build_gateway_loader ..... 2146
tbx_build_help ..... 2147
tbx_build_help_loader ..... 2148
tbx_build_loader ..... 2149
tbx_build_macros ..... 2150
tbx_build_src ..... 2151
tbx_builder_gateway ..... 2153
tbx_builder_gateway_lang ..... 2154
tbx_builder_help ..... 2155
tbx_builder_help_lang ..... 2156
tbx_builder_macros ..... 2157
tbx_builder_src ..... 2158
tbx_builder_src_lang ..... 2159
test_run ..... 2160
XLV. Ferramentas de Demonstração ..... 2164
demo_begin ..... 2165
demo_choose ..... 2166
demo_compiler ..... 2167
demo_end ..... 2168
demo_file_choice ..... 2169
demo_function_choice ..... 2170
demo_mdialog ..... 2171
demo_message ..... 2172
demo_run ..... 2173
XLVI. Planilhas ..... 2174
readxls ..... 2175
xls_open ..... 2176
xls_read ..... 2178
excel2sci ..... 2180
read_csv ..... 2181
write_csv ..... 2182
XLVII. call_scilab API ..... 2183
Boolean management ..... 2184
Complex management ..... 2186
DisableInteractiveMode ..... 2188
Double management ..... 2189
GetLastJob ..... 2191
ScilabHaveAGraph ..... 2192
SendScilabJob ..... 2193
SendScilabJobs ..... 2194
StartScilab ..... 2195
String management ..... 2196
TerminateScilab ..... 2198
call_scilab ..... 2199
Compile and run with Call Scilab ..... 2201
creadbmat (obsolete) ..... 2204
creadchain (obsolete) ..... 2206
creadcmat (obsolete) ..... 2208
creadmat (obsolete) ..... 2210
cwritebmat (obsolete) ..... 2212

cwritechain (obsolete) ..... 2214
cwritecmat (obsolete) ..... 2216
cwritemat (obsolete) ..... 2218
fromc ..... 2220
fromjava ..... 2221
XLVIII. FFTW ..... 2222
fftw ..... 2223
fftw_flags ..... 2225
fftw_forget_wisdom ..... 2227
get_fftw_wisdom ..... 2228
set_fftw_wisdom ..... 2229
XLIX. Interfaces com UMFPACK ..... 2230
PlotSparse ..... 2231
ReadHBSparse ..... 2232
cond2sp ..... 2234
condestsp ..... 2236
rafiter ..... 2238
res_with_prec ..... 2240
taucs_chdel ..... 2241
taucs_chfact ..... 2242
taucs_chget ..... 2244
taucs_chinfo ..... 2246
taucs_chsolve ..... 2247
taucs_license ..... 2248
umf_license ..... 2249
umf_ludel ..... 2250
umf_lufact ..... 2251
umf_luget ..... 2253
umf_luinfo ..... 2255
umf_lusolve ..... 2257
umfpack ..... 2258
L. Algoritmos Genéticos ..... 2260
coding_ga_binary ..... 2261
coding_ga_identity ..... 2262
crossover_ga_binary ..... 2263
crossover_ga_default ..... 2264
init_ga_default ..... 2266
mutation_ga_binary ..... 2267
mutation_ga_default ..... 2268
optim_ga ..... 2269
optim_moga ..... 2271
optim_nsga ..... 2273
optim_nsga2 ..... 2275
pareto_filter ..... 2277
selection_ga_elitist ..... 2278
selection_ga_random ..... 2280
LI. Arrefecimento Simulado ..... 2282
compute_initial_temp ..... 2283
neigh_func_csa ..... 2284
neigh_func_default ..... 2285
neigh_func_fsa ..... 2286
neigh_func_vfsa ..... 2287
optim_sa ..... 2288
temp_law_csa ..... 2290
temp_law_default ..... 2292
temp_law_fsa ..... 2293
temp_law_huang ..... 2295
temp_law_vfsa ..... 2297

LII. Parâmetros ..... 2299
add_param ..... 2300
get_param ..... 2301
init_param ..... 2302
is_param ..... 2303
list_param ..... 2304
remove_param ..... 2305
set_param ..... 2306
LIII. Atoms ..... 2307
Getting started ..... 2308
Functions Summary ..... 2310
atomsAutoloadAdd ..... 2311
atomsAutoloadDel ..... 2313
atomsAutoloadList ..... 2315
atomsDepTreeShow ..... 2316
atomsGetInstalled ..... 2317
atomsGetLoaded ..... 2319
atomsInstall ..... 2320
atomsIsInstalled ..... 2322
atomsIsLoaded ..... 2324
atomsList ..... 2326
atomsLoad ..... 2327
atomsRemove ..... 2328
atomsRepositoryAdd ..... 2330
atomsRepositoryDel ..... 2331
atomsRepositoryList ..... 2332
atomsSearch ..... 2333
atomsSetConfig ..... 2334
atomsShow ..... 2336
atomsSystemUpdate ..... 2337
atomsUpdate ..... 2338
LIV. Entrada/Saída de Arquivos Matlab Binários ..... 2340
loadmatfile ..... 2341
matfile_close ..... 2342
matfile_listvar ..... 2343
matfile_open ..... 2344
matfile_varreadnext ..... 2345
matfile_varwrite ..... 2346
savematfile ..... 2347
LV. xcos ..... 2349
6. Batch functions ..... 2351
lincos ..... 2352
scicos ..... 2353
scicos_simulate ..... 2354
scicosim ..... 2356
steadycos ..... 2358
7. palettes ..... 2360
1. Annotations palette ..... 2360
2. Commonly used blocks palette ..... 2363
3. Continuous time systems palette ..... 2371
4. Demonstrations blocks palette ..... 2396
5. Discontinuities palette ..... 2413
6. Discrete time systems palette ..... 2426
7. Electrical palette ..... 2443
8. Event handling palette ..... 2501
9. Implicit palette ..... 2547
10. Integer palette ..... 2552
11. Lookup tables palette ..... 2583

12. Math operations palette ..... 2590
13. Matrix operation palette ..... 2637
14. Port & Subsystem palette ..... 2700
15. Signal processing palette ..... 2707
16. Signal routing palette ..... 2712
17. Sinks palette ..... 2748
18. Sources palette ..... 2803
19. Thermohydraulics palette ..... 2856
20. User defined functions palette ..... 2870
21. Zero crossing detection palette ..... 2896
8. Programming xcos Blocks ..... 2906
1. C Computational Functions ..... 2906
2. Scilab Computational Functions ..... 2920
3. Utilities Functions ..... 2924
9. Scilab Data Structures ..... 2935
1. Blocks ..... 2935
2. Compilation/Simulation ..... 2946
3. Diagram ..... 2954
4. Links ..... 2959
10. Scilab Utilities Functions ..... 2962
buildouttb ..... 2963
create_palette ..... 2964
get_scicos_version ..... 2965
scicos_debug ..... 2966
var2vec ..... 2967
vec2var ..... 2968
xcos ..... 2970
Menu_entries ..... 2971
LVI. scilab editor ..... 2981
edit_error ..... 2982
Editor ..... 2983
LVII. API Scilab ..... 2987
11. Scilab Gateway API ..... 2989
1. How to ..... 2989
CheckColumn ..... 3011
CheckDimProp ..... 3012
CheckDims ..... 3013
CheckLength ..... 3014
CheckLhs ..... 3015
CheckRhs ..... 3016
CheckRow ..... 3017
CheckSameDims ..... 3018
CheckScalar ..... 3020
CheckSquare ..... 3021
CheckVector ..... 3022
CreateListVarFrom ..... 3023
CreateListVarFromPtr ..... 3026
CreateVar ..... 3029
FindOpt ..... 3031
FirstOpt ..... 3033
GetListRhsVar ..... 3035
GetRhsVar ..... 3037
GetType ..... 3039
IsOpt ..... 3040
Lhs ..... 3042
LhsVar ..... 3043
NumOpt ..... 3045
OverLoad ..... 3047

Rhs ... 3049
Scierror ... 3050
Scilab C Types ... 3051
get_optionals ... 3053
istk ... 3055
sci_types ... 3056
sciprint ... 3058
stk ... 3059
12. list_management ... 3060
Boolean reading (Scilab gateway) ... 3061
Boolean writing (Scilab gateway) ... 3073
Boolean sparse reading (Scilab gateway) ... 3077
Boolean sparse writing (Scilab gateway) ... 3089
Create List (Scilab gateway) ... 3093
Double reading (Scilab gateway) ... 3097
Double writing (Scilab gateway) ... 3109
Get child item (Scilab gateway) ... 3113
Item Number (Scilab gateway) ... 3116
Integer reading (Scilab gateway) ... 3119
Integer writing (Scilab gateway) ... 3131
Pointer reading (Scilab gateway) ... 3136
Pointer writing (Scilab gateway) ... 3148
Polynomial reading (Scilab gateway) ... 3152
Polynomial writing (Scilab gateway) ... 3164
Sparse reading (Scilab gateway) ... 3168
Sparse writing (Scilab gateway) ... 3180
String reading (Scilab gateway) ... 3184
String writing (Scilab gateway) ... 3196
Boolean reading (Scilab gateway) ... 3200
Boolean writing (Scilab gateway) ... 3202
Boolean sparse reading (Scilab gateway) ... 3205
Boolean sparse writing (Scilab gateway) ... 3208
Variable Reference (Scilab gateway) ... 3211
Variable dimension (Scilab gateway) ... 3215
Variable Type (Scilab gateway) ... 3219
Variable Complexity (Scilab gateway) ... 3223
Matrix Type (Scilab gateway) ... 3227
Double reading (Scilab gateway) ... 3231
Double writing (Scilab gateway) ... 3234
Integer Precision (Scilab gateway) ... 3238
Integer reading (Scilab gateway) ... 3242
Integer writing (Scilab gateway) ... 3249
Pointer reading (Scilab gateway) ... 3256
Pointer writing (Scilab gateway) ... 3258
Polynomial Symbolic Variable (Scilab gateway) ... 3261
Polynomial reading (Scilab gateway) ... 3265
Polynomial writing (Scilab gateway) ... 3270
Sparse matrix reading (Scilab gateway) ... 3272
Sparse writing (Scilab gateway) ... 3275
String reading (Scilab gateway) ... 3278
String writing (Scilab gateway) ... 3281
LVIII. Gerenciamento de ajuda online ... 3283
add_help_chapter ... 3284
apropos ... 3285
foo ... 3286
help ... 3287
help_from_sci ... 3288
help_skeleton ... 3290

make_index ..... 3291
man ..... 3292
manedit ..... 3296
%helps ..... 3297
xmltohtml ..... 3298
xmltojar ..... 3300
xmltopdf ..... 3302
xmltops ..... 3304
del_help_chapter ..... 3306
xmltochm ..... 3307

# Parte I. Scilab

# Name

abort — Interrupção de avaliação

# Descrição

`abort` interrompe a avaliação corrente e retrona ao prompt de comando. Dentro de um nível de `pause` , `abort` retorna ao prompt de nível 0.

# Ver Também

quit, pause, break

# Name

add_demo — adiciona uma entrada na lista de demonstrações

```
add_demo(title,path)
```

# Parâmetros

title
: string, o título da demonstração

path
: string, o endereço do script scilab associado à demonstração

# Descrição

Esta função adiciona uma nova entrada na lista de demonstrações. A demonstração deve ser executada por um arquivo script do Scilab. Se o dado título já existir `title` na lista de demonstrações associada ao mesmo arquivo, nada é feito. Esta função checa se o arquivo existe.

# Exemplos

```
//criando um simples script de demonstração
path=TMPDIR+'/foo.sce';
mputl('disp Hello',path)
add_demo('My first demo',path)
//a demonstração pode agora ser executada através do menu "Demos".
```

# Ver também

add_help_chapter

# Autor

Serge Steer , INRIA

# Name

ans — resposta

# Descrição

`ans` significa "resposta". A variável `ans` é criada automaticamente quando expressões não são atribuídas. `ans` contém a última expressão não-atribuída.

# Name

argn — número de argumentos na chamada de uma função

```
[lhs [,rhs] ]=argn()
lhs=argn(1)
rhs=argn(2)
```

# Descrição

Esta função é usada dentro da definição de uma função. Ela fornece os números de parâmetros de entrada `rhs` e saída `lhs` passados à função quando esta é chamada. Geralmente é usada em definições de funções para lidar com parâmetros opcionais.

# Ver Também

function, varargin

# Name

backslash (\) — divisão matricial direita-esquerda

```
x=A\b
```

# Descrição

(\) denota a divisão matricial direita-esquerda. `x=A\b` é a solução para `A*x=b`.

Se `A` é quadrada e não-singular, `x=A\b` (unicamente definida) é equivalente a `x=inv(A)*b` (mas as computações são menos custosas).

Se `A` não é quadrada, `x` é uma solução de mínimo quadrado, i.e., `norm(A*x-b)` é mínima (norma euclidiana). Se `A` é de posto coluna cheio, a solução de mínimo quadrado, `x=A\b` , é unicamente definida (há um único `x` que minimiza `norm(A*x-b)`). Se `A`não é de posto coluna cheio, então a solução de mínimo quadrado não é única e `x=A\b` , em geral, não é a solução com norma mínima (a solução com norma mínima é `x=pinv(A)*b`).

`A.\B` é a matriz com entrada `(i,j)` igual a `A(i,j)\B(i,j)`. Se `A` (ou `B`é um escalar, `A.\B` é equivalente a `A*ones(B).\B` (ou `A.\(B*ones(A))`

`A\.B` é um operador sem significado predefinido. Pode ser usado para definir um novo operador (ver overloading) com a mesma precedência que * ou /.

# Exemplos

```
A=rand(3,2);b=[1;1;1]; x=A\b; y=pinv(A)*b;  x-y
A=rand(2,3);b=[1;1]; x=A\b; y=pinv(A)*b; x-y, A*x-b, A*y-b
A=rand(3,1)*rand(1,2); b=[1;1;1]; x=A\b; y=pinv(A)*b; A*x-b, A*y-b
A=rand(2,1)*rand(1,3); b=[1;1]; x=A\b; y=pinv(A)*b; A*x-b, A*y-b

// A benchmark of several linear solvers

[A,descr,ref,mtype] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa"

b = 0*ones(size(A,1),1);

tic();
res = umfpack(A,'\',b);
printf('\ntime needed to solve the system with umfpack: %.3f\n',toc());

tic();
res = linsolve(A,b);
printf('\ntime needed to solve the system with linsolve: %.3f\n',toc());

tic();
res = A\b;
printf('\ntime needed to solve the system with the backslash operator: %.3f\n',
```

# Ver Também

slash, inv, pinv, percent, ieee, linsolve, umfpack

# Name

banner — exibe banner do Scilab (Windows)

```
banner()
```

# Descrição

Exibe banner do Scilab

# Exemplos

```
clc();banner()
```

# Autor

Allan CORNET

# Name

boolean — objetos Scilab: variáveis booleanas e operadores '&', '|' e '~'

# Descrição

Uma variável booleana é `%T` (para "true", "verdadeiro") ou `%F` (for "false"). (para "false","falso"). Estas variáveis podem ser usadas para definir matrizes de valores booleanos, com a sintaxe usual. Matrizes de valores booleanos podem ser manipuladas como matrizes ordinárias para extração/inserção de elementos e concatenação. Note que outras operações usuais ( +, *, -, ^, etc.) não são definidas para matrizes de valores booleanos. Três operadores especiais são definidos para estas matrizes:

~b
: é a negação elemento a elemento de `b` (matriz).

b1&b2
: é o `and` ("e") lógico elemento a elemento de `b1` e `b2` (matrizes).

b1|b2
: é o `or` ("ou") lógico elemento a elemento de `b1` e `b2` (matrizes).

Variáveis booleanas podem ser usadas para se indexar vetores ou matrizes.
: Por exemplo `a([%T,%F,%T],:)` retorna a submatriz feita das linhas `1` e `3` de `a`. Matrizes esparsas de valores booleanos são suportadas.

# Exemplos

```
[1,2]==[1,3]
[1,2]==1
a=1:5; a(a&gt;2)
```

# Ver Também

matrices, or, and, not

# Name

brackets — ([,]) colchetes esquerdo e direito

```
[a11,a12,...;a21,a22,...;...]
[s1,s2,...]=func(...)
```

# Parâmetros

a11,a12,...
: qualquer matriz (de reais, polinômios, razões de polinômios, lista `syslin`...) com dimensões apropriadas

s1,s2,...
: qualquer nome de variável possível

# Descrição

Colchetes esquerdo e direito são usados para notar concatenação de vetores e matrizes. Estes símbolos também são usados para denotar lado esquerdo múltiplo para a chamada de uma função.

Dentro de colchetes de concatenação, espaços vazios ou vírgulas significam "concatenação de colunas" e pontos-e-vírgulas ou retronos de carro significa "concatenação de linhas".

Nota : para evitar confusões, é melhor utilizar vírgulas, ao invés de espaços vazios para separar colunas.

Dentro de lados esquerdos múltiplos, os nomes das variáveis de vem ser separados por vírgulas.

# Exemplos

```
[6.9,9.64; sqrt(-1) 0]
[1 +%i  2 -%i  3]
[]
['this is';'a string';'vector']
s=poly(0,'s');[1/s,2/s]
[tf2ss(1/s),tf2ss(2/s)]

[u,s]=schur(rand(3,3))
```

# Ver Também

comma, semicolon

## Name

break — palavra-chave para interromper um laço ("loop") (significa "interromper (neste ponto)")

## Descrição

Dentro de um laço `for` ou `while` loop, o comando `break` força o fim do laço.

## Exemplos

```
k=0; while 1==1, k=k+1; if k &gt; 100 then  break,end; end
```

## Ver Também

while, if, for, abort, return

# Name

case — palavra-chave usada na estrutura select (significa "caso...")

# Descrição

Palavra chave usada na estrutura `select ... case`

Use-a do seguinte modo:

```
select expressão0,
  case expressão1 then instruções1,
  case expressão2 then instruções2,
  ...
  case expressãon then instruçõesn,
  [else instruções],
end
```

# Ver Também

select, while, end, for

# Name

clear — cancela variáveis

```
clear a
```

# Descrição

Este comando cancela variáveis que não estão protegidas. Ele remove as variáveis nomeadas do ambiente. Por si só, `clear` cancela todas as variáveis, exceto as protegidas por `predef`. Logo, os dois comandos `predef(0)` e `clear` removem todas as variáveis.

Normalmente, variáveis protegidas são bibliotecas padrões e variáveis com o prefixo '%'

Note a sintaxe particular `clear a` e não `clear(a)`. Note também que `a=[]` não cancela `a` , mas define `a` como uma matriz vazia.

# Ver Também

predef, who

# Name

clearfun — remove função primitiva

```
ret=clearfun('name')
```

# Descrição

`clearfun('name')` remove a primitiva `'name'` do conjunto de primitivas (funções built-in). clearfun retorna %t ou %f. Esta função permite renomear uma primitiva : uma primitiva Scilab pode ser substituída por uma função definida pelo usuário. Para usuários experientes...

# Ver Também

newfun, funptr

# Name

clearglobal — cancela variáveis globais

```
clearglobal()
clearglobal nam1 .. namn
clearglobal('nam1', ..,'namn')
```

# Parâmetros

nam1,..., namn
: nomes de variáveis válidos

# Descrição

`clearglobal()` cancela todas as variáveis globais.

`clearglobal nam1 .. namn` cancela as variáveis globais dadas por seus nomes.

Note que `clearglobal()` limpa apenas as variáveis globais; as cópias locais destas variáveis não são destruídas.

# Exemplos

```
global a b c
a=1;b=2;c=3;
who('global')
clearglobal b
who('global')
```

# Ver Também

global, who

# Name

colon — (:) operador dois pontos

# Descrição

Colon symbol `:` can be used to form implicit vectors. (see also linspace, logspace)

j:k
: é o vetor `[j, j+1,...,k]` (vazio se `j>k`).

j:d:k
: é o vetor `[j, j+d, ..., j+m*d]`

A notação de dois pontos também pode ser usada para retirar linhas, colunas e elementos selecionados de vetores e matrizes (ver também extraction,insertion)

A(:)
: é o vetor de todos os elementos de `A` considerados como uma única coluna.

A(:,j)
: é a j-ésima coluna de `A`

A(j:k)
: é `[A(j),A(j+1),...,A(k)]`

A(:,j:k)
: é `[A(:,j),A(:,j+1),...,A(:,k)]`

A(:)=w
: preenche a matriz `A` com entradas de `w` (tomados coluna a coluna se `w` é uma matriz).

# Ver Também

matrix, for, linspace, logspace

# Name

comma — (,) separador de colunas, instruções e argumentos

# Descrição

Vírgulas são usadas para separar parâmetros em funções, ou para separar entradas em vetores linhas.

Espaços vazios podem ser usados para separar entradas em um vetor linha, mas é preferível que se use vírgulas.

Também é usada para separar instruções do Scilab. (Use ; para que o resultado não seja exibido na tela).

# Exemplos

```
a=[1,2,3;4,5,6];
a=1,b=1;c=2
```

# Ver Também

semicolon, brackets

# Name

comments — comentários

# Descrição

Uma seqüência de duas barras `//` fora de uma definição de string marca o início de um comentário. As barras, tanto quanto o restante dos caracteres até o fim das linhas não são interpretados.

Dentro de uma função, as primeiras linhas de comentário até a primeira instrução ou linha vazia podem ser usadas para fornecer o conteúdo padrão para a documentação de ajuda da função ("help" da função).

# Exemplos

```
g=9.81// a gravidade

text='a//b'

function y=myfunction(x)
//  myfunction computa y=x^2+1
// x deve ser um vetor ou uma matriz
  y=x^2+1
endfunction

help myfunction
```

# Name

comp — compilação de função Scilab

```
comp(function [,opt])
```

# Parâmetros

function
: uma função scilab não-compilada (tipo 11)

opt
: flag com valor 0 (padrão), 1 ou 2.

# Descrição

`comp(function)` compila a função `function`. Funções compiladas e interpretadas são equivalentes, mas geralmente as compiladas são mais rápidas. As funções fornecidas nas bibliotecas padrões são compiladas.

A definição online bem como a sintaxe curta dos comandos `exec` e `deff` geram funções compiladas. Então comp tem que ser usado apenas em casos bem particulares. Para se produzir uma função não compilada, deve-se usar >exec ou deff com a opção `"n"`.

O valor opt==2 faz com que a função seja compilada "para definição de perfil". Note que agora é possível adicionar a instrução de definição de perfil após a compilação utilizando a função add_profiling .

A opção obsoleta opt==1 era específica para propósitos de análise de códigos e agora é ignorada, i.e tratada como opt==0.

Note: na compilação a função original é modificada e nenhum objeto novo é criado.

# Ver Também

type, deff, exec, function, add_profiling, profile

# Name

comparison — comparação, operadores relacionais

```
a==b
a~=b ou a<>b
a<b
a<=b
a>b
a>=b
```

# Parâmetros

a

qualquer tipo de variável para comparações de igualdade `a==b`, `a~=b` `a<>b` e restrita a arrays de pontos flutuantes reais e inteiros para comparações relacionadas à ordem `a<b`, `a<=b`, `a>b`,`a>=b`.

b

qualquer tipo de variável para comparações de igualdade `a==b`, `a~=b` `a< > b` e restrita a arrays de ponto flutuantes reais e inteiros para comparações relacionadas à ordem `a<b`, `a<=b`, `a>b`,`a>=b`.

# Descrição

Duas classes de operadores devem ser distinguidas:

Comparações de igualdade e desigualdade:

`a==b`, `a~=b` (ou de modo equivalente `a<>b`). Estes operadores se aplicam a quaisquer tipos de operandos.

Comparações de ordem:

`a<b`, `a<=b`, `a>b`,`a>=b`. Estes operadores se aplicam apenas a arrays de pontos flutuantes reais e inteiros.

A semântica dos operadores de comparação também depende dos tipos de operandos:

Com variáveis arrays

como arrays de pontos flutuantes ou inteiros, arrays lógicos, arrays de strings, arrays de polinômios ou razões de polinômios, arrays de manipuladores, listas,... o seguinte se aplica:

- Se `a` e `b` são avaliados como arrays do mesmo tipo e dimensões idênticas, a comparação é realizada elemento a elemento e o resultado é um array de booleanos da comparação.
- Se `a` e `b` são avaliados como arrays do mesmo tipo, mas `a` ou `b` é um array 1 por 1, o elemento é comparado com cada elemento do outro array. O resultado é um array de booleanos com o tamanho do operando não-escalar.
- Em outros casos, o resultado é o booleano `%f`
- Se os tipos de dados dos operandos são diferentes, mas "compatíveis", como pontos flutuantes e inteiros, uma conversão de tipo é realizada antes da comparação.

Com outros tipos de operandos

como `function`, `libraries`, o resultado é `%t` se os objetos são idênticos e `%f` em outros casos.

Comparação de igualdade entre operandos com tipos de dados incompatíveis retorna `%f`.

# Exemplos

```
//comparações elemento a elemento
(1:5)==3
(1:5)<=4
(1:5)<=[1 4 2 3 0]
1<[]
list(1,2,3)~=list(1,3,3)

//comparações objeto a objeto
(1:10)==[4,3]
'foo'==3
1==[]
list(1,2,3)==1

isequal(list(1,2,3),1)
isequal(1:10,1)

//comparação com conversão de tipo
int32(1)==1
int32(1)<1.5
int32(1:5)<int8(3)
p=poly(0,'s','c')
p==0
p/poly(1,'s','c')==0
```

# Ver Também

less, boolean, isequal

# Name

continue — Palavra-chave para passar o controle para a próxima iteração de um laço ("loop") (significa "continuar")

# Descrição

Dentro de um laço `for` ou `while`, o comando `continue` passa o contole para a próxima iteração do laço no qual aparece, pulando quaisquer sentenças restantes entre esta instrução e a instrução `end` do laço.

# Exemplos

```
for k=1:10,K=k;if k&gt;2&amp;k&lt;=8 then continue,disp('hello'),end,k,end

for j=1:2
  x=[];
  for k=1:10,if k&gt;j+1&amp;k&lt;=8 then continue,end,x=[x,k];end
  x
end
```

# Ver Também

while, for, break, return

# Autor

Serge Steer, INRIA

# Name

debug — nivel de depuração ("debugging") debugging level

```
debug(level-int)
level-int=debug()
```

# Parâmetros

level-int
: inteiro (de 0 a 4)

# Descrição

Para os valores 0, 1, 2, 3 ou 4 de `level-int` , define vários níveis de depuração. Isto é voltado para o analisador sintático, não para scripts Scilab. Para usuários mais experientes do Scilab.

# Name

delbpt — deleta pontos de parada ("breakpoints")

```
delbpt([macroname [,linenumb]])
```

# Parâmetros

macroname
: string

linenumb
: escalar inteiro ou vetor de inteiros

# Descrição

Deleta o ponto de parada na linha `linenumb` na função `macroname`.

`linenumb` pode ser um vetor linha ou coluna de inteiros, ou um único escalar.

Se `linenumb` for omitido, todos os pontos de parada da função `macroname` são deletados.

Se ambos `macroname` e `linenumb` são omitidos, todos os pontos de parada em todas as funções são deletados.

# Exemplos

```
setbpt('foo',1),setbpt('foo',10),delbpt('foo',10),dispbpt()
delbpt('foo',1),dispbpt()

setbpt('foo1',4),setbpt('foo1',9),setbpt('foo2',6),setbpt('foo3',8),dispbpt()
delbpt('foo2'),dispbpt()
delbpt(),dispbpt()

delbpt('foo',[1,2,5,6]),dispbpt()
```

# Ver Também

setbpt, dispbpt, pause, resume

# Name

dispbpt — exibe breakpoints (pontos de parada)

```
dispbpt()
```

# Descrição

`dispbpt()` exibe todos os breakpoints ativos inseridos nas funções.

# Ver Também

setbpt, delbpt, pause, resume

# Name

do — palavra-chave de linguagem para laços ("loops") (significa "faça...")

# Descrição

Pode ser usado dentro de instruções `for` ou `while` para separar a definição de variável do laço e o conjunto de instruções do mesmo.

# Ver Também

for, while

# Name

dot — (.) símbolo de ponto

```
123.33
a.*b
[123,..
456]
```

# Descrição

.

É usado para marcar pontos (vírgulas) decimais em números: 3.25 e 0.001

.<op>

usado em conjunto com outros símbolos de operadores ( *, /, \, ^, ' ), forma outros operadores. Operações multiplicativas elemento a elemento são obtidas utilizando-se .* , .^ , ./ , .\ ou .' . Por exemplo, C = A ./ B é a matriz com elementos c(i,j) = a(i,j)/b(i,j). O produto de Kronecker é notado por.*. . Note que quando o ponto segue um número, é sempre parte deste 2.*x é avaliado como 2.0*x e 2 .*x (há um espaço entre 2 e .) é avaliado como (2).*x

..

Marca de continuação. Dois ou mais pontos ao fim de uma linha (ou seguidos por um comentário) indicam que a próxima linha será uma continuação.

Linhas de continuação são manipuladas por um processador que constrói uma linha lógica longa de uma dada seqüência de linhas de continuação. Então, marcas de continuação podem ser usadas para cortar uma linha em qualquer ponto.

# Exemplos

```
//ponto decimal
1.345

//usado como parte de um operador
x=[1 2 3];x.^2 .*x // um espaço é requerido entre 2 e o ponto

// usado como marcas de continuação
T=[123,..//primeiro elemento
   456] //segundo elemento

a=  "aqui, eu começo um  grande string... //mas não quero continuar aqui
     - e aqui eu continuo."
y=12..
45
```

# Ver Também

star, hat, slash, backslash

# Name

edit — edição de funções

```
edit(functionname)
```

# Parameters

functionname
: string

# Descrição

Se functionname for o nome de uma função Scilab definida `edit(functionname)` tente abrir o arquivo associado `functionname.sci`.

Se functionname for o nome de uma função Scilab indefinida, `edit` cria o arquivo functionname.sci no diretório TMPDIR.

# Exemplos

```
edit('edit')  //abre o editor com o texto dessa função
edit('myfunction')  //abre o editor para uma nova função
```

# Ver Também

manedit

# Name

else — palavra-chave usada na estrutura "if-then-else" (significa "senão...")

# Descrição

Usado com `if`.

# Ver Também

if

# Name

elseif — palavra-chave usada na estrutura "if-then-else" (significa "senão, se...")

# Descrição

Ver `if,then,else` .

# Name

empty — ([]) matriz vazia

# Descrição

`[]` denota a matriz vazia. É definida de modo único e possui 0 linha e 0 coluna, i.e. `size([]) =[0,0]` . As seguintes convenções convenientes foram feitas:

`[] * A = A * [] = []`

`[] + A = A + [] = A`

[ [], A] = [A, []] = A `inv([]) =[]`

`det([])=cond([])=rcond([])=1, rank([])=0`

Funções de matrizes retornam `[]` ou produzem uma mensagem de erro quando não há resposta óbvia. Sistemas lineares vazios (listas `syslin`) podem ter várias linhas ou colunas.

# Exemplos

```
s=poly(0,'s'); A = [s, s+1];
A+[], A*[]
A=rand(2,2); AA=A([],1), size(AA)
svd([])
w=ssrand(2,2,2); wr=[]*w; size(wr), w1=ss2tf(wr), size(w1)
```

# Ver Também

matrices, poly, string, boolean, rational, syslin

# Name

end — Palavra-chave end (significa "fim")

# Descrição

Usado no fim de estruturas de laço ou condicionais. `for`, `while`, `if`, `select` devem ser terminados por `end` .

# Ver Também

for, while, if, select

# Name

equal — (=) atribuição ou comparação, sinal de igualdade

# Descrição

Atribuição:

o sinal de igualdade ( = ) é usado para denotar a atribuição de valor(es) a variável(eis). A sintaxe pode ser :

- `a=expr` onde `a` é um nome de variável e `expr` uma expressão do Scilab cuja avaliação conduz a um único resultado.
- `[a,b,...]=expr` onde `a`,`b`,`...` são nomes de variáveis e `expr` uma expressão do Scilab cuja avaliação conduz a tantos resultados quanto o número de variáveis dadas.

Comparação:

o sinal de igualdade ( = ) também é usado em operadores de comparação:

- `a==b`, denota comparação de igualdade entre os valores das expressões `a` e `b`.
- `a~=b`, denota comparação de desigualdade entre os valores das expressões `a` e `b`:
- `a<=b` e `a>=b` denotam comparações de ordem entre os valores de `a` e `b`:

Ver comparison para detalhes semânticos.

# Exemplos

```
a = sin(3.2)
M = [2.1,3.3,8.5;7.6,6.7,6.9;0,6.3,8.8];
[u,s] = schur(M)
[1:10] == 4
1~=2
```

# Ver Também

less, great, boolean, isequal, comparison

# Name

errcatch — "Armadilha" para erros

```
errcatch(n [,'action'] [,'option'])
errcatch()
```

# Parâmetros

n
: inteiro

action, option
: strings

# Descrição

`errcatch` fornece uma "ação" (manipulador de erros) quando um erro do tipo `n` ocorre. `n` tem o seguinte significado:

Se `n>0`, `n` é o número do erro a ser apanhado

Se `n<0` todos os erros devem ser apanhados

`action` é um dos seguintes strings:

"pause"
: uma pausa é executada quando se apanha o erro. Esta opção é útil para propósitos de depuramento. Use `whereami()` para obter informações sobre o contexto corrente.

"continue"
: a próxima instrução na função ou em arquivos executáveis é executada, a instrução corrente é ignorada. É possível verificar se um erro ocorreu utilizando a função `iserror` function . Não se esqueça de apagar o erro utilizando a função `errclear` o quanto antes. Esta função é útil para recuperação de erros. Em muitos casos, o uso de `errcatch(n,"continue",..)` pode ser substituído pelo uso da função execstr ou da estrutura de controle try control structure.

"kill"
: modo padrão, todas as funções intermediárias são canceladas e o Scilab retorna ao prompt de nível 0.

"stop"
: interrompe a sessão do Scilab corrente (útil quando o Scilab é chamado por um programa externo).

`option` é o string `'nomessage'` para cancelar mensagens de erro.

Para retornar ao modo padrão, entre `errcatch(-1,"kill")` ou, de modo semelhante, `errcatch(-1)`.`errcatch()` é um equivalente obsoleto a `errcatch(-1)`.

As ações de `errcatch` se aplicm ao contexto corrente de avaliação (function, exec, pause ) e a todos os subníveis. Uma segunda chamada a `errcatch` em um sub-nível esconde a inicial para este sub-nível. Se uma segunda chamada a `errcatch` é feita no mesmo nível o efeito da primeira é removido.

Quando chamado no contexto de uma função do Scilab ou em exec o `errcatch` é automaticamente reiniciado quando a função retorna

# Ver Também

try, errclear, iserror, whereami, execstr

# Name

errclear — limpeza de erros

```
errclear([n])
```

# Descrição

Limpa a ação (manipulador de erros) conectada ao erro de tipo n.

Se n é positivo, é o número do erro limpo; caso contrário, todos os erros são limpos (caso padrão).

# Ver Também

errcatch, iserror, lasterror

# Name

error — mensagens de erro

```
error(message [,n])
error(n)
error(n,pos)
```

# Parâmetros

message
: um string. A mensagem de erro a ser exibida

n
: um inteiro. O número associado à mensagem de erro

pos
: um inteiro. Um parâmetro para a mensagem de erro

# Descrição

`error` permite escrever uma mensagem de erro e manipular o erro. Por padrão, `error` interrompe a execução corrente e retorna ao nível de prompt. Este padrão pode ser modificado utilizando-se as funções `errcatch` ou `execstr(...,'errcatch')`.

`error(message)` imprime o string contido em `message`. O número associado à mensagem de erro é 10000.

`error(message,n)` imprime o string contido em `message`. O número associado à mensagem de erro é dado por n . Este número deve ser maior que 10000.

`error(n)` imprime a mensagem de erro predefinda associada ao número de erro `n`.

Algumas mensagens de erro predefinidas requerem um parâmetro (ver error_table). Neste caso, o argumento `pos` deve ser usado `error(n,pos)` para fornecer o valor do parâmetro. Em outros casos, o parâmetro é ignorado.

Ver error_table para uma lista das mensagens de erro e seus números associados.

# Exemplos

```
error('my error message')
error(43)
error(52,3)
```

# Ver Também

warning, errcatch, execstr, lasterror

# Name

error_table — tabela de mensagens de erros

# Description

Esta página fornece a tabela e mensagens de erros predefinidas e seus números associados. Algumas dessas mensagens de erro são utilizadas pelo próprio Scilab para erros de "parser" ou erros de "builtins" específicos. Algumas outras são de uso mais geral e podem ser utilizadas por funções do Scilab. As com asteríscos são aquelas para as quais a sintaxe `error(n,pos)` é manipulada.

1 "Incorrect assignment" (atribuição incorreta)

2 "Invalid factor" (fator inválido)

3 "Waiting for right parenthesis" (esperava parêntese direito)

4 "Undefined variable : %s" (variável indefinida: %s)

5 "Inconsistent column/row dimensions" (dimensões de coluna/linha inconsistentes)

6 "Inconsistent row/column dimensions" (dimensões de linha/coluna inconsistentes)

7 "Dot cannot be used as modifier for this operator" (ponto não pode ser usado como modificador para ste operador)

8 "Inconsistent addition" (adição inconsistente)

9 "Inconsistent subtraction" (subtração inconsistente)

10 "Inconsistent multiplication" (multiplicação inconsistente)

11 "Inconsistent right division " (divisão esquerda-direita inconsistente)

12 "Inconsistent left division" (divisão direita-esquerda inconsistente)

13 "Redefining permanent variable" (redefinição de variável permanente)

14 "Eye variable undefined in this context" (variável eye (identidade) indefinida neste contexto)

15 "Submatrix incorrectly defined" (submatriz definida incorretamente)

16 "Incorrect command!" (comando incorreto)

17 "Stack size exceeded! (Use stacksize function to increase it)" (tamanho de pilha excedido! (Use a função stacksize para aumentá-lo))

18 "Too many variables!" (muitas variáveis!)

19 "Problem is singular" (o problema é singular)

* 20 "Wrong type for argument %d: Square matrix expected." (tipo errado para o argumento %d: esperava matriz quadrada)

21 "Invalid index" (índice inválido)

22 "Recursion problems. Sorry...." (problemas de recursão. Perdão...)

23 "Matrix norms available are 1, 2, inf, and fro" (normas de matriz disponíveis são 1,2,inf e fro)

24 "Convergence problem..." (problema de convergência...)

25 "Bad call to primitive :%s" (chamada ruim à primitiva: %s)

26 "Too complex recursion! (recursion tables are full)" (recursão muito complexa! as tabelas de recursão estão cheias)

27 "Division by zero..." (divisão por zero...)

28 "Empty function..." (função vazia...)

29 "Matrix is not positive definite" (a matriz não é positiva definida)

30 "Invalid exponent" (expoente inválido)

31 "Incorrect string" (string incorreto)

32 "Singularity of log or tan function" (singularidade da função log ou tan)

33 "Too many "":"""" (muitos ':')

34 "Incorrect control intruction syntax" (sintaxe incorreta de instrução de controle)

35 "Syntax error in an if,a while or a select instruction" (erro de sintaxe em uma instrução if, while ou select)

* 36 "Wrong input argument %d." (argumento de entrada%d incorreto)

* 37 "Incorrect function at line %d" (função incorreta na linha %d)

38 "Wrong file name." (nome de arquivo incorreto)

39 "Incorrect number of arguments" (número incorreto de argumentos)

40 "Waiting for end of command" (esperava fim do comando)

41 "Incompatible output argument." (argumento de saída incompatível)

42 "Incompatible input argument." (argumento de entrada incompatível)

43 "Not implemented in scilab...." (não implementado no Scilab...)

* 44 "Wrong argument %d." (argumento %d incorreto)

* 45 "Null matrix (argument # %d)" (matriz nula (argumento número %d))

46 "Incorrect syntax" (sintaxe incorreta)

47 "End or else is missing..." (faltando end ou else)

* 48 "Input line longer than buffer size %d" (linha de entrada maior que tamanho de buffer %d)

49 "Incorrect file or format" (arquivo ou formato incorreto)

50 "Subroutine not found : %s" (subrotina não encontrada: %s)

* 52 "Wrong type for argument %d: Real matrix expected." (tipo errado para o argumento %d: esperava matriz de reais)

* 53 "Wrong type for input argument %d: Real or complex matrix expected." (tipo errado para o argumento %d: esperava matriz de reais ou de complexos)

* 54 "Wrong type for input argument %d: Polynomial expected." (tipo errado para o argumento %d: esperava polinômio)

* 55 "Wrong type for argument %d: String expected." (tipo errado para o argumento %d: esperava string)

* 56 "Wrong type for argument %d: List expected." (tipo errado para o argumento %d: esperava lista)

57 "Problem with comparison symbol..." (problema com símbolo de comparação)

58 "Function has no input argument..." (a função não tem argumento de entrada)

59 "Function has no output." (a função não tem saída)

60 "Wrong size for argument: Incompatible dimensions." (tamanho errado para o argumento: dimensões incompatíveis)

61 "Direct access file : give format" (arquivo de acesso direto: fornecer formato)

* 62 "End of file at line %d" (fim do arquivo na linha %d)

* 63 "%d graphic terminal?" (%d terminais gráficos?)

64 "Integration fails" (falha de integração)

* 65 "%d: logical unit already used" (%d: unidade lógica já utilizada)

66 "No more logical units available!" (não há mais unidades lógicas disponíveis)

67 "Unknown file format " (formato de arquivo desconhecido)

68 "Fatal error!!! Your variables have been saved in the file : %s"

69 "Floating point exception" (exceção de ponto flutuante)

70 "Too many arguments in fort (max 30)" (muitos argumentos em fort (máximo de 30))

71 "This variable is not valid in fort" (esta variável não é válida em fort)

72 "%s is not valid in this context" (%s não é válido neste contexto)

73 "Error while linking" (erro no linking)

74 "Leading coefficient is zero" (coeficiente regente é 0)

75 "Too high degree (max 100)" (grau muito alto (no máximo 100))

* 76 "For x=val with type(val)="%d is not implemented in Scilab" (para x=val com type(val)=%d não há implementação no Scilab)

77 "%s : wrong number of rhs arguments" (%s: número incorreto de argumentos do lado direito)

78 "%s : wrong number of lhs arguments" (%s: número incorreto de argumentos do lado esquerdo)

79 "Indexing not allowed for output arguments of resume." (indexação não permitida para argumentos de saída de resume)

80 "Incorrect function (argument n:%s)" (função incorreta (argumento n: %s))

81 "%s: Wrong type for argument %d: Real or complex matrix expected." (%s: tipo errado para o argumento %d: esperava matriz de reais ou de complexos)

82 "%s: Wrong type for argument %d: Real matrix expected." (%s: tipo errado para o argumento %d: esperava matriz de reais)

83 "%s: Wrong type for argument %d: Real vector expected." (%s: tipo errado para o argumento %d: esperava vetor de reais)

84 "%s: Wrong type for argument %d: Scalar expected." (%s: tipo errado para o argumento %d: esperava escalar)

85 "Host does not answer..." (host não responde)

86 "Uncontrollable system" (sistema incontrolável)

87 "Unobservable system" (sistema inobservável)

88 "sfact : singular or assymetric problem" (sfact: problema singular ou assimétrico)

* 89 "Wrong size for argument %d." (tamanho errado para o argumento %d)

* 90 "Wrong type for argument %d: Transfer matrix expected." (tipo errado para o argumento %d: esperava matriz de transferência)

* 91 "Wrong type for argument %d: In state space form expected." (tipo errado para o argumento %d: esperava em forma de espaço de estados)

* 92 "Wrong type for argument %d: Rational matrix expected." (tipo errado para o argumento %d: esperava matriz de razões de polinômios)

* 93 "Wrong type for argument %d: In continuous time expected." (tipo errado para o argumento %d: esperava em tempo contínuo)

* 94 "Wrong type for argument %d: In discrete time expected." (tipo errado para o argumento %d: esperava em tempo discreto)

* 95 "Wrong type for argument %d: SISO expected." (tipo errado para o argumento %d: esperava SISO

* 96 "Time domain of %dth argument is not defined" (domínio de tempo do %d-ésimo argumento não está definido)

* 97 "Wrong type for argument %d: A system in state space or transfer matrix form expected." (tipo errado para o argumento %d: esperava um sistema em espaço de estados ou uma matriz de transferência)

98 "Variable returned by scilab argument function is incorrect" (variável retornada pela função argumento do Scilab está incorreta)

* 99 "Elements of %dth argument must be in increasing order!" (elementos do %d-ésimo argumento devem estar em ordem crescente!)

* 100 "The elements of %dth argument are not in (strictly) decreasing order" (os elementos do %d-ésimo argumento não estão em ordem (estritamente) crescente)

* 101 "Last element of %dth argument <> first" (o último elemento do %d-ésimo argumento não é igual ao primeiro)

102 "Variable or function %s is not in file %s" (variável ou função %s não está no arquivo %s)

103 "Variable %s is not a valid rational function " (variável %s não é uma função racional válida)

104 "Variable %s is not a valid state space representation" (variável %s não é uma representação de espaço de estados válida)

105 "Undefined fonction" (função indefinida)

106 "Function name already used" (nome de função já utilizado)

* 107 "Too many functions are defined (maximum #:%d)" (muitas fuñções estão definidas (número máximo: %d))

108 "Too complex for scilab, may be a too long control instruction" (muito complexo para o Scilab, pode ser uma instrução de controle muito longa)

109 "Too large, can't be displayed" (muito grande, não pode ser exibido)

110 "%s was a function when compiled but is now a primitive!" (%s era uma função quando compilada, mas agora é uma primitiva)

111 "Trying to re-define function %s " (tentando redefinir a função %s)

112 "Cannot allocate more memory" (não pode alocar mais memória)

113 "Too large string" (string muito grande)

114 "Too many linked entry points" (muitos pontos de entradas "linkados")

115 "Stack problem detected within a loop" (problema de empilhamento detectado dentro de um laço)

* 116 "Wrong value for argument %d." (valor errado para o argumento %d)

* 117 "List element number %d is Undefined" (elemento número %d da lista é indefinido)

* 118 "Wrong type for argument %d: Named variable not an expression expected." (tipo errado para o argumento %d: esperava variável nomeada, não uma expressão)

119 "Indices for non-zero elements must be given by a 2 column matrix" (índices para elementos não-nulos devem ser fornecidos por uma matriz de duas colunas)

121 "Incompatible indices for non-zero elements " (índices incompatíveis para elementos não-nulas)

* 122 "Logical unit number should be larger than %d" (número de unidade lógica deve ser maior que %d)

123 "Function not bounded from below" (função não limitada por baixo)

124 "Problem may be unbounded :too high distance between two consecutive iterations" (o problema pode ser ilimitado: distância muito alta entre duas iterações consecutivas)

126 "Inconsistent constraints" (restrições inconsistentes)

127 "No feasible solution" (não há solução viável)

128 "Degenerate starting point" (ponto de partida degenerado)

129 "No feasible point has been found" (nenhum ponto viável foi encontrado)

130 "Optimization fails: back to initial point" (a otimização falhou, de volta ao ponto inicial)

131 "optim: stop requested by simulator (ind= 0)" (optim: parada requerida pelo simulador (ind=0))

132 "optim: incorrect input parameters" (optim: parâmetros de entrada incorretos)

133 "Too small memory" (memória muito pequena)

134 "optim: problem with initial constants in simul " (optim: problema com constantes iniciais em simul)

135 "optim : bounds and initial guess are incompatible" (optim: limites e palpite inicial são incompatíveis)

136 "optim : this method is NOT implemented " (optim: este método NÃO é implementado)

137 "NO hot restart available in this method" (NÃO há recomeço rápido disponível neste método)

138 "optim : incorrect stopping parameters" (optim: parâmetros de parada incorretos)

139 "optim : incorrect bounds" (optim: limites incorretos)

140 "Variable : %s must be a list" (variável : s% deve ser uma lista)

* 141 "Incorrect function (argument n:%d)" (função incorreta (argumento n: %d))

* 142 "Hot restart : dimension of working table (argument n:%d)" (recomeço rápido: dimensão da mesa de trabalho (argumento n:%d))

143 "optim : df0 must be positive !" (optim: df0 deve ser positivo!)

144 "Undefined operation for the given operands." (operação indefinida para os dados operandos)

201 "%s: Wrong type for argument %d: Real or complex matrix expected." (%s tipo errado para o argumento %d: esperava matriz de reais ou complexos)

202 "%s: Wrong type for argument %d: Real matrix expected." (%s tipo errado para o argumento %d: esperava matriz de reais)

203 "%s: Wrong type for argument %d: Real vector expected." (%s tipo errado para o argumento %d: esperava vetor de reais)

* 204 "%s: Wrong type for argument %d: Scalar expected." (%s tipo errado para o argumento %d: esperava escalar)

205 "%s: Wrong size for argument %d: (%d,%d) expected." (%s tamanho errado para o argumento %d: esperava (%d,%d))

206 "%s: Wrong size for argument %d: %d expected." (%s tamanho errado para o argumento %d: esperava %d)

207 "%s: Wrong type for argument %d: Matrix of strings expected." (%s tipo errado para o argumento %d: esperava matriz de strings)

208 "%s: Wrong type for argument %d: Boolean matrix expected." (%s tipo errado para o argumento %d: esperava matriz de booleanos)

209 "%s: Wrong type for argument %d: Matrix expected." (%s tipo errado para o argumento %d: esperava matriz)

210 "%s: Wrong type for argument %d: List expected." (%s tipo errado para o argumento %d: esperava lista)

211 "%s: Wrong type for argument %d: Function or string (external function) expected." (%s tipo errado para o argumento %d: esperava função ou string (função externa))

212 "%s: Wrong type for argument %d: Polynomial expected." (%s tipo errado para o argumento %d: esperava polinômio)

213 "%s: Wrong type for argument %d: Working integer matrix expected." (%s tipo errado para o argumento %d: esperava matriz de inteiros operante)

214 "Argument %d of %s: wrong type argument, expecting a vector" (Argumento %d de %s: tipo de argumento errado, esperava um vetor)

* 215 "%dth argument type must be boolean" (%d-ésimo argumento deve ser booleano)

* 216 "Wrong type for argument %d: Boolean or scalar expected." (%s tipo errado para o argumento %d: esperava booleano ou escalar)

* 217 "Wrong type for argument %d: Sparse matrix of scalars expected." (%s tipo errado para o argumento %d: esperava matriz esparsa de escalares)

* 218 "Wrong type for argument %d: Handle to sparse lu factors expected." (%s tipo errado para o argumento %d: esperava manipulador para fatores lu esparsos)

* 219 "Wrong type argument %d: Sparse or full scalar matrix expected." (%s tipo errado para o argumento %d: esperava matriz de escalares esparsa ou completa)

220 "Null variable cannot be used here" (variável nula não pode ser usada aqui)

221 "A sparse matrix entry is defined with two differents values" (uma entrada da matriz esparsa é definida com dois valores diferentes)

222 "%s not yet implemented for full input parameter." (%s ainda não foi implementado para parâmetro de entrada completa)

223 "It is not possible to redefine the %s primitive this way (see clearfun)." (não é possível redefinir a primitiva %s desta forma (ver clearfun))

224 "Type data base is full" (banco de dados de tipos está cheio)

225 "This data type is already defined" (este tipo de dado já foi definido)

226 "Inequality comparison with empty matrix" (comparação de desigualdade com matriz vazia)

227 "Missing index" (faltando índice)

228 "Reference to the cleared global variable %s" (referência à variável global limpa %s)

229 "Operands of / and \\ operations must not contain NaN of Inf." (operandos de / e \\ não devem conter NaN ou Inf)

230 "semidef fails" (semidef falhou)

231 "Wrong type for first input argument: Single string expected." (tipo errado para o primeiro argumento de entrada: esperava string único)

232 "Entry name not found" (nome de entrada não encontrado)

233 "Maximum number of dynamic interfaces reached" (número máximo de interfaces dinâmicas alcançado)

234 "link: expecting more than one argument" (link: experava mais que um argumento)

235 "link: problem with one of the entry point" (link: problema com um dos pontos de entrada)

236 "link: the shared archive was not loaded" (link: o arquivo compartilhado não foi carregado)

237 "link: Only one entry point allowed On this operating system" (link: apenas um ponto de entrada permitido neste sistema operacional)

238 "link: First argument cannot be a number" (primeiro argumento não pode ser um número)

239 "You cannot link more functions, maxentry reached" (você não pode mais linkar funções, número máximo de entradas alcançado)

240 "File %s already exists or directory write access denied " (arquivo %s já existe ou o acesso à escrita de diretório foi negado)

241 "File %s does not exist or read access denied " (arquivo %s não existe ou o acesso à leitura foi negado)

242 "Binary direct access files must be opened by 'file'" (arquivos binários de acesso direto devem ser abertos por "file")

243 "C file logical unit not allowed here" (unidade lógica do arquivo C não permitida aqui)

244 "Fortran file logical unit not allowed here" (unidade lógica do arquivo FORTRAN não permitida aqui)

* 245 "No input file associated to logical unit %d" (nenhum arquivo de entrada associado à unidade lógica %d)

246 "Function not defined for given argument type(s)" (função não definida para o(s) tipo(s) de argumento(s) fornecido(s))

247 "Wrong value for argument %d: the lu handle is no more valid." (valor errado para o argumento %d: o manipulador lu não é mais válido)

* 248 "Wrong value for argument %d: Valid variable name expected." (valor errado para o argumento %d: esperava nome de variável válido)

* 249 "Wrong value for argument %d: Empty string expected." (valor errado para o argumento %d: esperava string vazio)

250 "Recursive extraction is not valid in this context" (extração recursiva não é válida neste contexto)

251 "bvode: ipar dimensioned at least 11" (bvode: ipar dimensionado em pelo menos 11)

252 "bvode: ltol must be of size ipar(4)" (bvode: ltol deve ser de tamanho ipar(4))

253 "bvode: fixpnt must be of size ipar(11)" (bvode: fixpnt deve ser de tamanho ipar(11))

254 "bvode: ncomp must be less than 20" (bvode: ncomp deve ser menor do que 20)

255 "bvode: m must be of size ncomp" (bvode: m deve ser de tamanho ncomp)

256 "bvode: sum(m) must be less than 40" (bvode: sum(m) deve ser menor do que 40)

257 "bvode: sum(m) must be less than 40" (bvode: sum(m) deve ser menor do que 40)

258 "bvode: input data error" (bvode: erro de dado de entrada)

259 "bvode: no. of subintervals exceeds storage" (bvode: número de subintervalos excede o armazenamento)

260 "bvode: Th colocation matrix is singular" (bvode: a matriz de colocação é singular)

261 "Interface property table is full" (tabela de propriedades de interface está cheia)

* 262 "Too many global variables!,max number is %d" (muitas variáveis globais! O máximo é %d)

263 "Error while writing in file,(disk full or deleted file)" (erro ao escrever em um arquivo (disco cheio ou arquivo deletado))

* 264 "Wrong value for argument %d: Must not contain NaN or Inf." (valor errado para o argumento %d: não deve conter NaN ou Inf)

265 "A and B must have equal number of rows" (A e B devem ter o mesmo número de linhas)

266 "A and B must have equal number of columns" (A e B devem ter o mesmo número de colunas)

267 "A and B must have equal dimensions" (A e B devem ter dimensões iguais)

* 268 "Invalid return value for function passed in arg%d" (valor de retorno inválido para função passado no argumento %d)

* 269 "Wrong value for argument %d: eigenvalues must have negative real parts." (valor errado para o argumento %d: autovalores devem ter partes reais negativas)

* 270 "Wrong value for argument %d: eigenvalues modulus must be less than one." (valor errado para o argumento %d: módulos dos autovalores não devem ser menores que 1)

* 271 "Size varying argument aeye(), (arg %d) not allowed here" (argumento de variação de tamanho aeye(), (arg %d) não permitido aqui)

272 "endfunction is missing" (está faltando endfunction)

273 "Instruction left hand side: waiting for a dot or a left parenthesis" (lado esquerdo da instrução: esperava por um ponto ou parêntese esquerdo)

274 "Instruction left hand side: waiting for a name" (lado esquerdo da instrução: esperava um nome)

275 "varargout keyword cannot be used here" (palavra-chave varargout não pode ser usada aqui)

276 "Missing operator, comma, or semicolon" (faltando operador, vírgula ou ponto-e-vírgula)

277 "Too many commands defined" (muitos comandos definidos)

278 "%s: Input arguments should have the same formal variable name." (%s: argumentos de entrada devem ter o mesmo nome de variável formal)

# Ver Também

warning, errcatch, execstr, lasterror

# Name

evstr — avaliação de expressões

```
H=evstr(Z)
[H,ierr]=evstr(Z)
```

# Parâmetros

Z
: matriz de strings `M` ou `list(M,Subexp)`

  M
  : matriz de strings

  Subexp
  : vetor de strings

H
: matriz

ierr
: inteiro, indicador de erro

# Descrição

Retorna o resultado da avaliação da matriz de strings `M`. Cada elemento da matriz deve definir uma expressão Scilab válida.

Se a avaliação de `M` levar a um erro, a versão de valor de único retorno, `H=evstr(M)`, levanta erro de modo usual. A versão de dois valores de retorno, `[H,ierr]=evstr(M)` , por outro lado, não produz erro, mas retorna o número de erro em `ierr`.

Se `Z` é uma lista, `Subexp` é um vetor de strings, que define subexpressões que são avaliadas antes da avaliação de `M`. Estas subexpressões devem ser referidas como `%(k)` em `M`, onde `k` é o índice da subexpressão em `Subexp`.

evstr('a=1') não é válido (use `execstr` ao invés).

# Exemplos

```
a=1; b=2; Z=['a','b'] ; evstr(Z)

a=1; b=2; Z=list(['%(1)','%(1)-%(2)'],['a+1','b+1']);
evstr(Z)
```

# Ver Também

execstr

# Name

exists — verifica existência de variáveis

```
exists(name [,where])
```

# Parâmetros

name
: string

where
: um caractere opcional com possíveis valores: `'l'` (local), `'n'` (nolocal) and `'a'` (all). O valor padrão é `'all'`.

# Descrição

`exists(name)` retorna `1` se a variável chamada `name` existe e `0`, em caso contrário.

Aviso: uma função que utiliza `exists` pode retornar um resultado que depende do ambiente!

`exists(name,'local')` retorna `1` se a variável chamada `name` existe no ambiente local da função corrente e `0` , em caso contrário.

`exists(name,'nolocal')` retorna `1` se a variável chamada `name` existe em qualquer nível do ambiente de chamamento (incluindo o nível principal do shell do Scilab) da função corrente e `0` , em caso contrário.

Aviso: a função exists não verifica se a variável existe no espaço de nomes global.

# Exemplos

```
deff('foo(x)',..
['disp([exists(''a12''),exists(''a12'',''local'')])'
 'disp([exists(''x''),exists(''x'',''local'')])'])
foo(1)
a12=[];foo(1)

function level1()
  function level2()
    disp(exists("a","all"));
    disp(exists("a","local"));
    disp(exists("a","nolocal"));
  endfunction
  level2()
endfunction
function go()
  a=1;
  level1()
endfunction
go()
```

# Ver Também

isdef, isglobal, whereis, type, typeof, macrovar

# Name

exit — termina a sessão Scilab corrente

# Descrição

Termina a sessão Scilab corrente.

# Ver Também

quit, abort, break, return, resume

# Name

external — objeto Scilab, função ou rotina externa

# Descrição

Função ou rotina externa para uso com comandos específicos.

Uma "external" é uma função ou rotina que é usada como argumento em algumas primitivas de alto nível (tais como `ode`, `optim`, `schur`...).

A seqüência de chamamento da "external" (função ou rotina) é imposta pela primitiva de alto nível que configura os argumentos da external.

Por exemplo, a função externa costfunc é um argumento da primitiva optim . Sua seqüência de chamamento deve ser: `[f,g,ind]=costfunc(x,ind)` e `optim` (a primitiva de otimização de alto nível) é invocada como segue:

```
optim(costfunc,...)
```

Aqui `costfunc` (a função de custo a ser minimizada pela primitiva `optim`) avalia `f=f(x)` e `g=` gradiente de `f` em `x` (`ind` é um inteiro. Seu uso é esclarecido na ajuda de `optim`).

Se a função externa necessita de outros valores, estas variáveis podem ser definidas em seu ambiente. Elas também podem ser colocadas em uma lista. Por exemplo, a função externa

```
[f,g,ind]=costfunc(x,ind,a,b,c)
```

é válida para `optim` se o external (função externa) é `list(costfunc,a,b,c)` e a chamada a `optim` é, então:

```
optim(list(costfunc,a1,b1,c1),....
```

Um external também pode ser uma rotina FORTRAN ou C: isto é conveniente para acelerar computações.

TO nome da rotina é fornecido para a primitiva de alto nível como um string. A seqüência de chamamento da rotina também é imposta. Exemplos são dados no diretório `routines/default` (ver o arquivo `README`).

Rotinas FORTRAN ou C podem ser dinamicamente linkadas ("ligadas, conectadas") (ver `link`)

# Ver Também

ode, optim, impl, dassl, intg, schur, gschur

# Name

extraction — extração de entradas de matrizes e listas

```
x(i)
x(i,j)
x(i,j,k,..)
[...]=l(i)
[...]=l(k1)...(kn)(i) ou [...]=l(list(k1,...,kn,i))
l(k1)...(kn)(i,j) or l(list(k1,...,kn,list(i,j))
```

# Parâmetros

x
: matriz de qualquer tipo possível

l
: variável do tipo lista

i,j, k
: índices

k1,...kn
: índices

# Descrição

CASO DE MATRIZES
: `i`, `j`, `k`,.. podem ser:

escalares reais ou vetores ou matrizes com elementos positivos.

- `r=x(i,j)` constrói a matriz `r` tal que `r(l,k)=x(int(i(l)),int(j(k)))` para `l` de 1 a `size(i,'*')` e `k` de 1 a `size(j,'*')`. O valor máximo de `i` (`j`) deve ser menor do que ou igual a `size(x,1)` (`size(x,2)`).
- `r=x(i)` com `x` uma matriz 1x1 contrói a matriz `r` tal que `r(l,k)=x(int(i(l)),int(i(k)))` para `l` de 1 a `size(i,1)` e `k` para 1 a `size(i,2)`.

  Note que, nesse caso, o índice `i` é válido apenas se todas as suas entradas são iguais a 1.
- `r=x(i)` com `x` um vetor linha constrói o vetor linha `r` tal que `r(l)=x(int(i(l)))` para `l` de 1 a `size(i,'*')i` deve ter valor máximo menor do que ou igual a `size(x,'*')`.
- `r=x(i)` com `x` uma matriz com uma ou mais colunas constrói o vetor coluna `r` tal que `r(l)` (`l` de 1 a `size(i,'*')`) contém a entrada `int(i(l))` do vetor coluna formado pela concatenação das colunas de `x`.

  `i` deve ter valor máximo menor do que ou igual a `size(x,'*')`.

O símbolo `':'`
: significa "todos os elementos".

- `r=x(i,:)` constrói uma matriz `r` tal que `r(l,k)=x(int(i(l)),k))` para `l` de 1 a `size(i,'*')` and `k` from 1 to `size(x,2)`

- `r=x(:,j)` constrói a matriz `r` tal que `r(l,k)=x(l,int(j(k)))` para `l` de 1 a `size(r,1)` e `k` de 1 a `size(j,'*')`.
- `r=x(:)` constrói o vetor coluna `r` formado pelas concatenações das colunas de `x`. É equivalente a `matrix(x,size(x,'*'),1)`.

vetor de booleanos
: Se um índice (`i` ou `j` ) é um vetor de booleanos, é interpretado como `find(i)` ou respectivamente `find(j)`

um polinômio
: Se um índice (`i` ou `j` ) é um vetor de polinômios ou um vetor de polinômios implícito, é interpretado como `horner(i,m)` ou respectivamente `horner(j,n)` onde `m` e `n` são as dimensões associadas a `x`. Mesmo se este recurso funcionar para todos os polinômios, é recomendado utilizar polinômios em `$` para legibilidade.

Para matrizes com mais de duas dimensoes (ver:hypermatrices), a dimensionalidade é automaticamente reduzida quando as dimensões mais a direita são 1.

CASO DE LISTAS OU TLISTS
: Se estiverem presentes, os `ki` fornecem o endereço para uma entrada de sub-lista da estrutura de dados `l` Eles permitem uma extração recursiva sem cópias intermediárias. As instruções

```
[...]=l(k1)...(kn)(i)
```

e

```
[...]=l(list(k1,...,kn,i))
```

são interpretadas como:

`lk1 = l(k1) .. = .. lkn = lkn-1(kn)` `[...] = lkn(i)` e as instruções `l(k1)...(kn)(i,j)` e

`l(list(k1,...,kn,list(i,j))` são interpretadas como: `lk1 = l(k1) .. = .. lkn = lkn-1(kn)` `lkn(i,j)` `i` e `j`, podendo ser: quando pontos de endereço sobre mais de um componente da lista, a instrução deve ter tantos argumentos do lado esquerdo quanto os componentes selecionados. Mas se a sintaxe de extração é usada dentro da seqüência de chamamento de entrada de uma função, cada componente da lista retornado é adicionado à seqüência de chamamento da função.

Note que, `l(list())` é o mesmo que `l`.

i e j podem ser :

escalares reais ou vetores ou matrizes de elementos positivos
: `[r1,...rn]=l(i)` extrai os elementos `i(k)` da lista l e armazena-os em variáveis `rk` para `k` de 1 a `size(i,'*')`

O símbolo `:`
: significa "todos os elementos".

um vetor de booleanos
: Se `i` é um vetor de booleanos, é interpretado como `find(i)`.

um polinômio
: Se `i` é um vetor de polinômios ou um vetor de polinômios implícito, é interpretado como `horner(i,m)` onde `m=size(l)`. Mesmo que este recurso funcione para todos os polinômios, é recomendado utilizar polinômios em `$` para legibilidade.

k1,..kn podem ser :

escalares reais posistivos

um polinômio
: interpretado como `horner(ki,m)` onde `m` é o tamanho da sub-lista correspondente.

string
: associado ao nome da entrada da sub-lista .

# Observações

Para tipos de matrizes "soft-coded" como funções racionais e sistemas lineares de espaços de estados, a sintaxe `x(i)` não pode ser usada para extração de elementos de vetores devido a confusões com extração de elementos de listas. A sintaxe `x(1,j)` ou `x(i,1)` deve ser usada.

# Exemplos

```
// CASO DE MATRIZES
a=[1 2 3;4 5 6]
a(1,2)
a([1 1],2)
a(:,1)
a(:,3:-1:1)
a(1)
a(6)
a(:)
a([%t %f %f %t])
a([%t %f],[2 3])
a(1:2,$-1)
a($:-1:1,2)
a($)
//
x='teste'
x([1 1;1 1;1 1])
//
b=[1/%s,(%s+1)/(%s-1)]
b(1,1)
b(1,$)
b(2) // o numerador
// CASO DE LISTAS OU TLISTS
l=list(1,'qwerw',%s)
l(1)
[a,b]=l([3 2])
l($)
x=tlist(l(2:3)) //forma um tlist com os últimos dois componentes de l
//
dts=list(1,tlist(['x';'a';'b'],10,[2 3]));
dts(2)('a')
dts(2)('b')(1,2)
[a,b]=dts(2)(['a','b'])
```

# Ver Também

find, horner, parents

# Name

for — palavra-chave de linguagem para laços ("loops") (significa "para...")

# Descrição

Usado para definir laços ("loops"). sua sintaxe é: `for variável=expressão ,instrução, ... ,instrução,end`

`for variável= expressão do instrução, ... ,instrução,end`

Se `expressão` é uma matriz ou um vetor linha, `variável` toma como valores os valores de cada coluna da matriz.

Um caso particular utiliza o operador colon para criar vetores linhas regularmente espaçados, e remonta a formas de laço "for" tradicionais : `for variable=n1:step:n2, ...,end`

Se `expressão` é uma lista, `variável` toma como valores as entradas sucessivas da lista.

Aviso: o número de caracteres usados para definir o corpo de qualquer instrução condicional (if, while, for ou select/case) deve ser limitado a 16k.

# Exemplos

```
// laços "for" tradicionais
n=5;
for i = 1:n, for j = 1:n, a(i,j) = 1/(i+j-1);end;end
for j = 2:n-1, a(j,j) = j; end; a
for j= 4:-1:1, disp(j),end // laço decrescente

//laço em colunas de matrizes
for  e=eye(3,3),e,end
for v=a, write(6,v),end
for j=1:n,v=a(:,j), write(6,v),end

//laço em listas de entradas
for l=list(1,2,'example'); l,end
```

# Ver Também

while, end, do

# Name

format — impressão de números e formato de exibição

```
format([type],[long])
v = format()
format(m)
```

# Parâmetros

type
: string

long
: inteiro ( número máximo de digitos (padrão 10))

v
: um vetor para o formato corrente onde v(1) é o de tipo de formato : 0 para 'e' e 1 para 'v' v(2) número de dígitos

m
: um vetor para ajuste de novo formato

  m(1) número de digitos

  m(2) tipo de formato : 0 para 'e' e 1 para 'v'

# Descrição

Ajusta o formato de impressão corrente através do parâmetro `type` ; é um dos seguintes :

"v"
: para um formato variável (default)

"e"
: para o formato "e" (notação científica)

`long` define o número máximo de dígitos (padrão 10). `format()` retorna o vetor que indica o fomato corrente: primeiro componente é o tipo do formato (1 se `v` ; 0 se `e` ); o segundo componente é o número de digitos.

No "formato variável", entradas de vetores que são menores que %eps vezes o valor absoluto máximo das entradas era exibido como "0" em versões anteriores do Scilab. Não é mais o caso, a função de clean pode ser usada para ajustar as entradas negligenciáveis para zeros.

# Exemplos

```
x=rand(1,5);
format('v',10);x
format(20);x
format('e',10);x
format(20);x

x=[100 %eps];
format('e',10);x
format('v',10);x
```

```
format("v")
```

## Ver Também

write, disp, print

# Name

funcprot — alterna o modo de proteção de funções Scilab

```
prot=funcprot()
funcprot(prot)
```

# Parâmetros

prot
: inteiro com valores possíveis 0,1 ou 2

# Descrição

Funções Scilab são variáveis, funcprot permite especificar o que o Scilab faz quando tais variáveis são redefinidas.

- Se prot==0 nada especial é feito
- Se prot==1 o scilab exibe uma mensagem de aviso quando a função é redefinida (modo padrão)
- Se prot==2 o scilab exibe uma mensagem de erro quando a função é redefinida

# Exemplos

```
funcprot(1)
deff('[x]=foo(a)','x=a')
deff('[x]=foo(a)','x=a+1')
foo=33
funcprot(0)
deff('[x]=foo(a)','x=a')
deff('[x]=foo(a)','x=a+1')
foo=33
```

# Name

funptr — codificação de primitivas

```
[numptr] = funptr(name)
```

# Parâmetros

name
: string, nome de uma primitiva

numptr
: o número de rotina interno da primitiva

# Descrição

Fução utilitária (para usuários mais experientes apenas) para retornar o número de rotina interno `numptr` da primitiva `'name'`. `numptr` é formado a partir do número de interface `fun` e do número de rotina `fin` da primitiva em sua interface por `numptr = 100*fun + fin` (fin < 100). De `numptr` pode-se obter o número de interface `fun = floor(numptr/100)` que pode ser útil para ligar uma interface dinâmica com seus argumentos passados por referência (ver seção de exemplos).

# Exemplos

```
// Suponha que você queira carregar alguns códigos através
// das facilidades de carregamento dinâmico oferecidas por addinter. Por padrão
// os argumentos são passados por valores, mas se você quiser
// passá-los por referência, você pode fazer o seguinte
// (nome sendo o nome scilab de uma das rotinas com
// interface) :
//
// addinter(files,spnames,fcts)  // argumentos passados por valores
// num_interface = floor(funptr(nome)/100)
// intppty(num_interface)  // argumentos agora passados por referência
//
// Note que se você digitar o seguinte
//
// intppty()
//
// você verá todas as interfaces funcionando por referência
```

# Ver Também

clearfun, newfun, intppty, addinter

# Name

getdebuginfo — retorna informações sobre o Scilab para o debug

```
getdebuginfo()
dynamic_info = getdebuginfo();
[dynamic_info,static_info] = getdebuginfo();
```

# Descrição

getdebuginfo retorna informações sobre o Scilab para o debug.

dynamic_info = getdebuginfo(); retorna informações sobre o seu sistema operacional.

[dynamic_info,static_info] = getdebuginfo(); retorna informações sobre o seu sistema operacional e sobre o Scilab.

# Ver Também

getversion

# Autor

A.C

# Name

getmd5 — retorna soma de verificação md5

```
res=getmd5(filename)
res=getmd5(ParamString,'string')
```

# Parâmetros

res
: resultado md5 (string)

filename
: nome do arquivo (string ou matrizes de strings)

ParamString
: string ou matrizes de strings

# Descrição

`getmd5(...)` retorna a soma de verificação md5 de um arquivo ou um string.

# Exemplos

```
getmd5('ola mundo','string')
getmd5(['ola' 'mundo'],'string')
getmd5(['ola' ; 'mundo'],'string')

getmd5( SCI+'/modules/core/etc/core.start' )
getmd5( SCI+'/modules/core/etc/'+['core.start' 'core.quit'])
```

# Autor

A.C.

# Name

getmemory — retorna as memórias livre e total do sistema

```
[free, total]=getmemory()
```

# Descrição

`getmemory()` retorna a memória livre do sistema (em kilo-octetos).

`[free, total]=getmemory()` retorna as memórias livre e total do sistema (em kilo-octetos).

# Autor

A.C

# Name

getmodules — retorna a lista de módulos instalados no Scilab

```
res=getmodules()
```

# Parâmetros

res
: uma matriz de strings

# Descrição

Retorna a lista de módulos instalados no Scilab.

# Ver Também

with_module

# Autor

A.c

# Name

getos — retorna o nome e a versão do Sistema Operacional

```
OS=getos()
[OS,Version]=getos()
```

# Descrição

`getos` retorna o nome e a versão do Sistema Operacional

# Exemplos

```
OS=getos()
[OS,version]=getos()
```

# Autor

A.C

# Name

getscilabmode — retorna o modo do Scilab

```
mode = getscilabmode()
```

# Descrição

Retorna o modo do Scilab. Quatro são os possíveis modos : STD , API , NW , NWNI .

`API` o Scilab é lançado como um API.

`STD` o Scilab padrão (gui, plot ...)

`NW` Scilab em linha de comando com esboços.

`NWNI` Scilab em linha de comando sem quaisquer gráficos.

# Exemplos

```
getscilabmode()
```

# Ver Também

scilab

# Name

getshell — retorna o interpretador de comando corrente

```
getshell()
```

# Descrição

`getshell` retorna o interpretador de comando corrente

# Exemplos

```
getshell()
```

# Autor

Allan CORNET

# Name

getvariablesonstack — retorna nomes de variáveis em na pilha Scilab

```
s=getvariablesonstack()
s=getvariablesonstack('local')
s=getvariablesonstack('global')
```

# Parâmetros

s
: uma matriz de strings

# Descrição

Retorna na variável `s` nomes de variáveis na pilha do Scilab.

getvariablesonstack('local') retorna o nome das variáveis locais na pilha.

getvariablesonstack('global') retorna o nome das variáveis globais na pilha.

As variáveis são organizadas na ordem alfabética.

# Exemplos

```
getvariablesonstack()
getvariablesonstack('local')
getvariablesonstack('global')
```

# Ver Também

who

# Name

getversion — retorna o nome da versão do Scilab

```
version=getversion()
[version,opts]=getversion()
ver=getversion('scilab')
versioninfo=getversion('scilab','string_info')
ver=getversion('<module>')
versioninfo=getversion('<module>','string_info')
```

# Parâmetros

version
: um string

versioninfo
: um string sobre a versão

ver
: um vetor de inteiros

  ver(1): versão maior

  ver(2): versão menor

  ver(3): versão de manutençãp

  ver(4): timestamp GIT

opts
: um vetor de strings de 7 entradas: `[compiler, pvm, tk, modelicac, release_mode, release_date, release_time]`

# Descrição

Retorna em `version` o nome da versão do Scilab e em `opts` build options ("opções de construção") que podem ser usadas para determinar se o Scilab foi construído em pvm, tk ou modelicac.

# Exemplos

```
getversion()
[version,opts]=getversion()
ver=getversion('scilab')
verstr=getversion('scilab','string_info')
ver=getversion('overloading')
verstr=getversion('overloading','string_info')
```

# Ver Também

getmodules

# Name

global — define variável global

```
global('nam1',...,'namn')
global nam1 ... namn
```

# Parâmetros

nam1,..., namn
: nomes de variáveis válidos

# Descrição

De um modo simples, cada função do Scilab possui suas próprias variáveis locais e pode "ler" todas as variáveis criadas no espaço de trabalho base ou pelas funções de chamamento. A palavra-chave `global` permite realizar leituras ou escritas de variáveis através de funções. Qualquer atribuição àquela variável, em qualquer função, está disponível para todas as outras funções que a declaram como variável `global`.

Se a variável global não existe (não possui valor) na primeira vez em que você escrever a declara como `global` , será inicializada como matriz nula.

# Exemplos

```
//primeiro: o ambiente de chamamento e uma função compartilham uma variável
global a
a=1
deff('y=f1(x)','global a,a=x^2,y=a^2')
f1(2)
a
//segundo: três funções compartilham variáveis
deff('initdata()','global A C ;A=10,C=30')
deff('letsgo()','global A C ;disp(A) ;C=70')
deff('letsgo1()','global C ;disp(C)')
initdata()
letsgo()
letsgo1()
```

# Ver Também

who, isglobal, clearglobal, gstacksize, resume

# Name

gstacksize — Ajusta/retorna a pilha global do Scilab

```
gstacksize(n)
gstacksize('max')
gstacksize('min')
sz=gstacksize()
```

# Parâmetros

n
: inteiro, o tamanho da pilha global requerido dado em número das palavras de dupla precisão

sz
: vetor de duas entradas [total used]

# Descrição

Scilab armazena variáveis globais em uma pilha.

`gstacksize(n)` permite ao usuário aumentar ou diminuir o tamanho da pilha. O tamanho máximo permitido depende da quantidade de memória livre e do espaço de troca ("swap space") disponível no momento. Note que o Scilab pode aumentar automaticamente o tamanho da pilha global quando necessário.

`sz=gstacksize()` retorna um vetor de duas entradas que contém os tamanhos de pilha global total e usado correntes.

`gstacksize('max')` permite que o usuário aumente o tamanho desta pilha global ao máximo.

`gstacksize('min')` permite que o usuário diminua o tamanho desta pilha global para máximo.

# Ver Também

who, stacksize

# Name

hat — (^) exponentiation

```
A^b
```

# Descrição

Exponenciação de matrizes ou vetores por um vetor constante.

Se `A` é um vetor ou uma matriz retangular, a exponenciação é feita elemento a elemento no sentido usual.

Para matrizes quadradas, a exponenciação é feita no sentido matricial.

Para matrizes de booleanos, polinômios ou razões de polinômios, o expoente deve ser um inteiro

## Observações

`123.^b` é interpretado como `(123).^b`. Em tais casos, o ponto é parte do operador, não do número.

Para dois números reais ou complexos `x1` e `x2` , o valor de `x1^x2` é o "valor principal" determinado por `x1^x2 = exp(x2*log(x1))`.

# Exemplos

```
2^4
(-0.5)^(1/3)
[1 2;2 4]^(1+%i)
s=poly(0,"s");
[1 2 s]^4
[s 1;1  s]^(-1)
```

# Ver Também

exp, inv

# Name

ieee — ajusta o modo de exceção de ponto flutuante

```
mod=ieee()
ieee(mod)
```

# Parâmetros

mod
: escalar inteiro com possíveis valores 0,1,ou 2

# Descrição

`ieee()` retorna o modo de exceção de ponto flutuante corrente

0
: exceção de ponto flutuante produz um erro;

1
: exceção de ponto flutuante produz um aviso;

2
: exceção de ponto flutuante produz um Inf ou NaN ("infinito" ou "não é número").

`ieee(mod)` ajusta o modo corrente de excessão de ponto flutuante.

O valor de modo incial é 0.

# Observações

Exceções de ponto flutuante emergentes em alguns algoritmos de bibliotecas ainda não são manipuladas pelos modos `ieee`.

# Exemplos

```
ieee(1);1/0
ieee(2);1/0,log(0)
```

# Ver Também

errcatch

# Name

if then else — Execução condicional (significa "se então senão")

```
    if expressão1 then sentenças
    elseif expressãoi then sentenças
    ....
    else sentenças
    end
```

# Descrição

A sentença `if` avalia sentenças lógicas e executa um grupo de sentenças se a expressão é verdadeira.

As `expressõesi` são expressões com valores numéricos ou booleanos. Se as `expressõesi` são matrizes, a condição é verdadeira somente se todas as entradas da matriz são "true" (verdadeiro) ou diferentes de 0.

Os opcionais `elseif` e `else` propiciam execução de grupos de sentenças alternativas. A palavra-chave `end` keyword, que corresponde ao `if`termina o último grupo de sentenças. A estrutura de linhas dada acima não é importante, a única restrição é que cada palavra-chave `then`deve estar na mesma linha de sua palavra-chave `if` ou `elseif` correspondente.

A palavra-chave `then` pode ser substituída por um retorno de carro ou uma vírgula.

Aviso: o número de caracteres utilizados para definir o corpo de qualquer instrução condicional (if while for or select/case) deve ser limitado a 16k.

# Exemplos

```
i=2
for j = 1:3,
   if i == j then
     a(i,j) = 2;
   elseif abs(i-j) == 1 then
     a(i,j) = -1;
   else a(i,j) = 0;
   end,
end
```

# Ver Também

try, while, select, boolean, end, then, else

# Name

insertion — atribuição ou modificação parcial de variáveis
assignation — atribuição parcial de variáveis

```
x(i,j)=a
x(i)=a
l(i)=a
l(k1)...(kn)(i)=a or l(list(k1,...,kn,i))=a
l(k1)...(kn)(i,j)=a or l(list(k1,...,kn,list(i,j))=a
```

# Parâmetros

x
: matriz de qualquer tipo

l
: lista

i,j
: índices

k1,...kn
: índices com valores inteiros

a
: novo valor de entrada

# Descrição

CASO DE MATRIZES
: Se `x` é uma matriz, os índices `i` e `j`, podem ser:

Escalares reais, vetores ou matrizes
: neste caso, os valores fornecidos como índices devem ser inteiros e apenas suas partes inteiras são levadas em consideração.

- Se `a` é uma matriz com dimensões `(size(i,'*'),size(j,'*'))`, `x(i,j)=a` retorna uma nova matriz `x` tal que `x(int(i(l)),int(j(k)))=a(l,k)` para `l` de 1 a `size(i,'*')` e `k` de 1 a `size(j,'*')`, as outras entradas iniciais de `x` não são modificadas.
- Se `a` é um escalar, `x(i,j)=a` retorna uma nova matriz `x` tal que `x(int(i(l)),int(j(k)))=a` para `l` de 1 a `size(i,'*')` e `k` de 1 a `size(j,'*')`, as outras entradas iniciais de `x` não são modificadas.
- Se o valor máximo de `i` ou `j` exceder a dimensão correspondente da matriz `x` , o array `x` é previamente estendido para as dimensões requeridas com entradas 0 para matrizes padrões, strings de comprimento 0 para matrizes de strings e valores falsos para matrizes de booleanos.
- `x(i,j)=[]` cancela linhas especificadas por `i` se `j` corresponde a todas as colunas de `x` ou cancela colunas especificadas por `j` se `i` corresponde a todas as linhas de `x`. Em outros casos `x(i,j)=[]` produz um erro.
- `x(i)=a` com `a` um vetor retorna a nova matriz `x` tal que `x(int(i(l)))=a(l)` para `l` de 1 a `size(i,'*')` , outras entradas iniciais de `x` não são modificadas.
- `x(i)=a` com `a` um escalar retorna uma nova matriz `x` tal que `x(int(i(l)))=a` para `l` de 1 a `size(i,'*')` , outras entradas iniciais de `x` não são modificadas.

Se o valor máximo de `i` exceder `size(x,1)`, `x` é previamente estendida para a dimensão requerida com entradas 0 para matrizes padrões, strings de comprimento 0 para matrizes de strings e valores falsos para matrizes de booleanos.

Se
: `x` é uma matriz 1x1

  `a` pode ser um vetor linha (respectivamente coluna) com dimensão `size(i,'*')`. A matriz `x` resultante é um vetor linha (respectivamente coluna).

Se
: `x` é um vetor linha

  `a` deve ser um vetor linha de dimensão `size(i,'*')`

Se
: `x` é um vetor coluna

  `a` deve ser um vetor coluna de dimensão `size(i,'*')`

Se
: `x` é uma matriz geral

  `a` deve ser um vetor linha ou coluna de dimensão `size(i,'*')` e o valor máximo de `i` não pode exceder `size(x,'*')`,

- `x(i)=[]` cancela entradas especificadas por `i`.

O símbolo :
: O símbolo `':'` significa "todos os elementos".

- `x(i,:)=a` é interpretado como `x(i,1:size(x,2))=a`
- `x(:,j)=a` é interpretado como `x(1:size(x,1),j)=a`
- `x(:)=a` retorna em `x` a matriz `a` de forma redefinida de acordo com as dimensões de x. `size(x,'*')` deve ser igual a `size(a,'*')`

Vetores de booleanos
: se um índice (`i` ou `j` ) é um vetor de booleanos, é interpretado como `find(i)` ou, respectivamente, `find(j)`

Polinômios
: se um índice (`i` ou `j` ) é um vetor de polinômios ou um vetor de polinômios implícito, é interpretado como `horner(i,m)` ou respectivamente `horner(j,n)` onde `m` e `n` são as dimensões associadas de `x`. Mesmo se esse recurso funcionar para todos os polinômios, é recomendado utilizar polinômios em `$` para legibilidade.

CASO DE LISTAS OU TLISTS

- Se estiverem presentes, os `ki` fornecem o endereço para uma entrada de sub-lista da estrutura de dados `l` data structure. Eles permitem uma extração recursiva sem cópias intermediárias. As instruções `l(k1)...(kn)(i)=a` e `l(list(k1,...,kn,i)=a)` são interpretadas como:

```
lk1 = l(k1)  .. = ..

lkn = lkn-1(kn)  lkn(i) = a

lkn-1(kn) = lkn  .. = ..  l(k1) = lk1
```

E as instruções `l(k1)...(kn)(i,j)=a` e `l(list(k1,...,kn,list(i,j))=a` são interpretadas como:

```
lk1 = l(k1)  .. = ..

lkn = lkn-1(kn) lkn(i,j) = a

lkn-1(kn) = lkn  .. = ..  l(k1)= lk1
```

- `i` pode ser :
  - Um escalar real não-negativo. `l(0)=a` adiciona uma entrada à "esquerda" da lista, `l(i)=a` configura a entrada `i` da lista `l` como `a`. Se `i>size(l)`, `l` é previamente estendido com entradas de comprimento 0 (indefinidas). `l(i)=null()` suprime a `i`-ésima entrada da lista.
  - Um polinômio. Se `i` é um polinômio, é interpretado como `horner(i,m)` onde `m=size(l)`. Mesmo se este recurso funcionar para todos os polinômios, é recomendado utilizar polinômios em `$` para legibilidade.
- `k1,..kn` podem ser :
  - Escalares reais positivos.
  - Polinômios, interpretados como `horner(ki,m)` onde `m` é o tamanho da sub-lista correspondente.
  - Strings associados a nomes de entradas de uma sub-lista.

# Observações

Para tipos de matrizes "soft-coded" como funções racionais e sistemas lineares de espaços de estados, a sintaxe `x(i)` não pode ser usada para inserção de entradas em vetores devido a confusões com inserção de entradas em listas. A sintaxe `x(1,j)` ou `x(i,1)` deve ser usada.

# Exemplos

```
// CASO DE MATRIZES
a=[1 2 3;4 5 6]
a(1,2)=10
a([1 1],2)=[-1;-2]
a(:,1)=[8;5]
a(1,3:-1:1)=[77 44 99]
a(1)=%s
a(6)=%s+1
a(:)=1:6
a([%t %f],1)=33
a(1:2,$-1)=[2;4]
a($:-1:1,1)=[8;7]
a($)=123
//
x='teste'
x([4 5])=['4','5']
//
b=[1/%s,(%s+1)/(%s-1)]
b(1,1)=0
b(1,$)=b(1,$)+1
```

```
b(2)=[1 2] // o numerador
// CASO DE LISTAS OU TLISTS
l=list(1,'qwerw',%s)
l(1)='Modificado'
l(0)='Adicionado'
l(6)=['mais um';'adicionado']
//
//
dts=list(1,tlist(['x';'a';'b'],10,[2 3]));
dts(2).a=33
dts(2)('b')(1,2)=-100
```

# Ver Também

find, horner, parents, extraction

# Name

intppty — ajusta propriedades de passagem de argumentos de interface

```
funs=intppty()
intppty(fun)
```

# Parâmetros

fun
: inteiro, um número de interface (ver funptr)

funs
: vetor de inteiros, vetor de números de interface (ver funptr)

# Descrição

Os programas de interface pode ser escritos de duas formas diferentes para o modo de passagem de argumento de funções.

No primeiro modo, que é o padrão, os argumentos são passados por valor. A sintaxe é a seguinte:

```
foo(A,1+2)
```

o argumento associado a `A` será passado por valores (uma cópia de `A` é feita antes de `foo` i ser chamada, e o argumento associado a `1+2` será passado por valor.

No segundo modo, os argumentos podem ser passados por referência, se houver "argumentos nomeados" (nenhuma cópia do valor da variável é feita). `intppty(fun)` com `fun>0` diz ao Scilab que a interface com o número `fun` pode manipular argumentos passados por referência. A sintaxe é a seguinte:

```
foo(A,1+2)
```

O argumento associado a `A` será passado por referência e o argumento associado a `1+2` será passado por valor.

Aviso: a declaração de interface com número `fun` pode manipular argumentos passados por referência se não for o caso de produzir resultados imprevisíveis.

`intppty(fun)` com `fun<0` suprime esta propriedade para para a interface `-fun`.

`intppty()` retorna o vetor de interfaces que manipulam argumentos passados por referência.

Esta função pode ser útil para interfaces dinamicamente carregadas (ver addinter).

# Ver Também

funptr, addinter

# Name

inv_coeff — constrói uma matriz de polinômios a partir de seus coeficientes

```
[P]=inv_coeff(C,[,d,[name])
```

# Parâmetros

C
: matriz grande de coeficientes

d
: grau da matriz de polinômios. Parâmetro opcional com valor padrão `d=-1+size(C,'c')/size(C,'r')`

name
: string fornecendo o nome da variável polinomial (valor padrão `'x'`).

# Descrição

`P=inv_coeff(Mp,k)`, quando `k` é compatível com o tamanho de `Mp` size, retorna uma matriz de polinômios de grau k. `C=[C0,C1,...,Ck]` and `P= C0 + C1*x +... +Ck*x^k`.

# Exemplos

```
A=int(10*rand(2,6))
// Construindo uma matriz de polinômios de grau 1
P=inv_coeff(A,1)
norm(coeff(P)-A)
// Usando valor padrão para o grau
P1=inv_coeff(A)
norm(coeff(P1)-A)
```

# Ver Também

poly, degree, coeff

# Name

iserror — teste de ocorrência de erros

```
iserror([n])
```

# Descrição

Testa se um número de erro `n` ocorreu (após uma chamada a `errcatch`). `iserror` retorna `1` se o erro ocorreu e `0` , caso contrário.

`n>0` é o número do erro ; todos os erros são testados com `n<0`.

# Ver Também

error, errcatch

# Name

isglobal — verifica se uma variável é global

```
t=isglobal(x)
```

# Parâmetros

x
: qualquer variável

t
: um valor booleano

# Descrição

`isglobal(x)` retorna "true" ("verdadeiro"), se x tiver sido declarada como variável global e "false" ("falso"), em caso contrário.

# Exemplos

```
isglobal(1)
global a
isglobal(a)
```

# Ver Também

global, clearglobal, who

# Name

lasterror — retorna a última mensagem de erro registrada

```
str=lasterror( [opt] )
[str,n]=lasterror([opt])
[str,n,line,func]=lasterror([opt])
```

# Parâmetros

str
: vetor de strings ou matriz vazia: a última mensagem de erro registrada

n
: inteiro, 0 ou o último número de erro registrado

line
: inteiro, 0 ou o último número de linha da função registrado

func
: string, o último nome da função registrado

opt
: booleano, se %t for registrado, a mensagem é limpa. O padrão é %t

# Descrição

Cada vez que um erro ocorre, o manipulador de erros do Scilab o registra em tabelas ("tables") internas (apenas o último erro é retido). A função `lasterror` permite obter a mensagem de erro, o número de erro, a função corrente (se houver) e a linha corrente da função corrente destas tabelas.

O número de linha reportado é o número da linha *física* onde o último erro ocorreu. Note que versões Scilab anteriores à 5.0 reportavam o número da linha *lógica* do último erro. A diferença realmente importa apenas se a função em erro inclui linhas continuadas antes do ponto onde o erro ocorreu.

Esta função é útil quando usada em conjunto com `errcatch` ou `execstr`.

A mensagem de erro pode ser retida para uma futura chamada a `lasterror` usando `lasterror(%f)`.

# Exemplos

```
ierr=execstr('a=zzzzzzz','errcatch')
if ierr>0 then disp(lasterror()),end
```

# Ver Também

errcatch, execstr, error, errclear, edit_error

# Name

left — ([) colchete esquerdo

```
[a11,a12,...;a21,a22,...;...]
[s1,s2,...]=func(...)
```

# Parâmetros

a11,a12,...
: matriz de qualquer tipo compatível com dimensões compatíveis. s1,s2,... : qualquer nome de variável possível

# Descrição

Colchetes esquerdo e direito são usados para concatenação de vetores e matrizes. Estes símbolos também são utilizados para denotar um lado esquerdo múltiplo para uma chamada de função.

Dentro de colchetes de concatenação, espaços em branco ou vírgulas significam "concatenação de colunas"; pontos-e-vírgulas e retornos de carro significam "concatenação de linhas".

Nota : para evitar confusões, é preferível que se use vírgula no lugar de espaços em branco para separar colunas.

Dentro de colchetes de lado esquerdo múltiplo, nomes de variáveis devem ser separados por vírgula.

# Exemplos

```
[6.9,9.64; sqrt(-1) 0]
[1 +%i  2 -%i  3]
[]
['this is';'a string';'vector']
s=poly(0,'s');[1/s,2/s]
[tf2ss(1/s),tf2ss(2/s)]

[u,s]=schur(rand(3,3))
```

# Ver Também

comma, semicolon

# Name

less — (<) comparação "menor do que"
great — (>) comparação "maior do que"

# Descrição

Símbolo de comparação lógica

<>
: significa "diferente" (é o mesmo que ~= )

<
: significa "menor do que".

>
: significa "maior do que".

<=
: significa "maior do que".

>=
: significa "maior do que ou igual a".

# Ver Também

if, comparison, equal

# Name

macr2lst — conversão de função para lista

```
[txt]=macr2lst(function-name)
```

# Descrição

Esta primitiva converte uma versão compilada do Scilab `function-name` em uma lista que codifica a representação interna da função (notação polonesa inversa).

A primeira entrada da lista é o nome da função, as segunda e terceira são, respectivamente, os vetores das variáveis de lado esquerdo e direito (lhs e rhs). As entradas seguintes são registros de operação básica ou listas que contém as estruturas de controle hierárquicas como if , for, ...

Registros de operação básica são descritos por um vetor de strings cujo primeiro elemento representa código do operador (op code).

| op codes | significado | parâmetros |
|---|---|---|
| "0" | opcode ignorado | nenhum |
| "1" | não mais usado | |
| "2" | referência à função ou variável | nome da variável, #rhs, #lhs |
| "3" | coloca um string na pilha | um string |
| "4" | coloca uma matriz vazia na pilha | nenhum |
| "5" | aplica uma operação | código de operação, #rhs,#lhs |
| "6" | coloca um número na pilha | o número |
| "12" | comando pause | nenhum |
| "13" | comando break | nenhum |
| "14" | comando abort | nenhum |
| "15" | marca de fim da linha | nenhum |
| "17" | comando quit | nenhum |
| "18" | variável nomeada | nome da variável |
| "19" | cria estrutura de índice recursiva | comprimento do endereço, número de índices finais |
| "20" | chamada a função | nome da função, #rhs, #lhs |
| "23" | cria variável a partir do nome | nome da variável |
| "24" | coloca uma variável de tipo 0 na pilha | nenhum |
| "25" | registro de perfil | número da chamada, gasto de tempo |
| "26" | coloca um vetor de strings na pilha | #linhas, #colunas, seqüência de elementos |
| "27" | coloca uma referência a um builtin na pilha | número da interface, posição da interface, nome da função |
| "28" | comando continue | nenhum |
| "29" | atribuição | #lhs, modo de exibição, (nome da variável, #rhs)* |
| "30" | curto circuito de expressão lógica | tipo, tamanho do salto (jump) |

| | | |
|---|---|---|
| "31" | comentário | o comentário |
| "99" | comando return | nenhum |
| > "100" | chamada a um builtin (obsoleto) | 100*fun, #rhs, #lhs, fin |

A função fun2string pode ser usada para se gerar o código inicial

# Exemplos

```
//EXIBIÇÃO
function y=foo(x,flag)
  if flag then
    y=sin(x)
  else
    y=cos(x)
  end
endfunction
L=macr2lst(foo)
fun2string(L)
```

# Ver Também

macrovar, fun2string, macr2tree, tree2code

# Name

macr2tree — conversão de função para árvore

```
t=macr2tree(function-name)
```

# Parâmetros

function-name
: macro do Scilab

t
: uma "árvore" do Scilab

# Descrição

Esta primitiva converte uma função Scilab compilada `function-name` em uma árvore (tlistas imbricadas) que codifica a representação interna da função. Deve ser usada com `tree2code`.

# Exemplos

```
tree=macr2tree(cosh);
txt=tree2code(tree,%T);
write(%io(2),txt,'(a)');
```

# Ver Também

tree2code

# Autor

V.C.

# Name

matrices — objeto do Scilab: matrizes

# Descrição

Matrizes são objetos básicos definidos no Scilab. Elas podem ser definidas como segue:

```
E=[e11,e12,...,e1n;
   e21,e22,...,e2n;
        ....
   em1,em2,...,emn];
```

As entradas `eij` podem ser números reais, complexos, polinômios, razões de polinômios, strings ou booleanos.

Vetores são vistos como matrizes de uma linha ou uma coluna.

# Exemplos

```
E=[1,2;3,4]
E=[%T,%F;1==1,1~=1]
s=poly(0,'s');E=[s,s^2;1,1+s]
E=[1/s,0;s,1/(s+1)]
E=['A11','A12';'A21','A22']
```

# Ver Também

poly, string, boolean, rational, empty, hypermatrices

# Name

matrix — Muda a forma de vetores ou matrizes

```
y=matrix(v,n,m)
y=matrix(v,[sizes])
```

# Parâmetros

v
: vetor, matriz ou hipermatriz

n,m
: inteiros

sizes
: vetor de inteiros

y
: vetor, matriz ou hipermatriz

# Descrição

Para um vetor ou matriz com `n` x `m` entradas, o comando `y=matrix(v,n,m)` ou, de modo similar, `y=matrix(v,[n,m])` transforma o vetor (ou matriz) `v` em uma matriz `nxm` empilhando as entradas coluna a coluna de `v`.

Se uma das dimensões m ou n é igual a -1, ela é automaticamente atribuída ao quociente de tamanho (v,'*') pela outra dimensão,

Para uma hipermatriz tal como `prod(size(v))==prod(sizes)`, o comando `y=matrix(v,sizes)` (ou equivalentemente `y=matrix(v,n1,n2,...nm)`) transforma `v` em uma matriz ou hipermatriz empilhando "coluna a coluna" as entradas de `v` (a primeira dimensão variando primeiro). `y=matrix(v,sizes)` resulta numa matriz regular se os tamanhos são escalares ou um vetor de duas entradas.

# Exemplos

```
a=[1 2 3;4 5 6]
matrix(a,1,6)
matrix(a,1,-1)
matrix(a,3,2)
```

# Ver Também

matrices, hypermatrices, ones, zeros, rand, poly, empty

# Name

mode — seleciona um modo em um arquivo executável

```
mode(k)
k=mode()
```

# Descrição

Usado exclusivamente dentro de um arquivo executável ou uma função do Scilab, `mode(k)` permite mudar as informações exibidas durante a execução, dependendo do valor de `k`:

k=0
: os novos valores de variáveis são exibidos, se requeridos (ver ajuda em semi ou comma).

k=-1
: o arquivo executável ou a função Scilab roda "silenciosamente". (este é o valor padrão para funções do Scilab)

k=1 or k=3
: cada linha de instruções é ecoada precedida do prompt (se possível). Os novos valores de variáveis são exibidos, se requerido. Este é o padrão para arquivos executáveis.

k=7
: Os novos valores de variáveis são exibidos, se requerido. Cada linha de instruções é ecoada (se possível) e um prompt (>>) aparece após cada linha esperando por um retorno de carro.

  Se o retorno de carro segue o caractere "p", a execução é pausada (ver pause).

A exibição de linha é desabilitada para funções do Scilab compiladas (ver comp). Por padrão, funções do Scilab são executadas no modo "silencioso" ("-1").

# Ver Também

exec, semicolon, comma

# Name

mtlb_mode — alterna para modo de operações do Matlab

```
mmode=mtlb_mode()
mtlb_mode(mmode)
```

# Parâmetros

mmode
: booleano

# Descrição

Adições e subtrações do Scilab e do Matlab funcionam de modo diferente quando usadas com matrizes vazias:

Scilab

```
a+[] -->a
a-[] -->a
[]+a -->a
[]-a -->-a
```

Matlab

```
a+[] -->[]
a-[] -->[]
[]+a -->[]
[]-a -->[]
```

`mtlb_mode(%t)` alterna para o modo de operações para adição e subtração do Matlab. `mtlb_mode(%f)` volta para o modo de operações do Scilab.

`mtlb_mode()` retorna o valor 'mmode' corrente.

# Ver Também

empty

# Name

names — scilab names syntax

# Descrição

Nomes de variáveis ou funções no Scilab devem começar com uma letra, ou com um dos seguintes caracteres especiais: '%', '_', '#', '!', '$', '?'.

Os próximos caracteres podem ser letras ou um dos seguintes caracteres especiais: '_', '#', '!', '$', '?'

Nomes podem ser tão longos quanto se queira, mas apenas os primeiros 24 caracteres serão levados em consideração. Letras maiúsculas e minúsculas são diferentes.

# Exemplos

```
//Nomes válidos
%eps
A1=123
#Color=8
My_Special_Color_Table=rand(10,3)
//Nomes inválidos
//1A , b%, .C
```

# Name

newfun — adiciona um nome à tabela de funções

```
newfun("function-name",nameptr)
```

# Descrição

Função utilitária (apenas para peritos). Adiciona o nome `"function-name"` à tabela de funções conhecidas pelo interpretador. `"nameptr"` é um inteiro `100*fun+fin` onde `fun` e `fin` é a codficação interna da primitiva `"function-name"`. Esta função é útil para associar uma primitiva a uma rotina com interface em `"matusr.f"` (`fun=14`). Usado com `funptr` e `clearfun` pode-se redefinir uma primitiva por uma função com o mesmo nome.

# Ver Também

clearfun

# Name

null — deleta um elemento em uma lista

```
l(i)=null()
```

# Descrição

Deleta objetos dentro de uma lista.

# Exemplos

```
l=list(1,2,3);
l(2)=null() // retorna list(1,3)
```

# Ver Também

list, clear

# Name

parents — ( ) parênteses esquerdo e direito

```
(expressão)
[...]=func(e1,e2,...)
[x1,x2,...]=(e1,e2,...)
x(i,j)
v(i)
[...]=l(i)
```

# Parâmetros

x
: matriz de qualquer tipo possível

v
: vetor linha ou coluna de qualquer tipo possível

l
: variável tipo lista

func
: qualquer nome de função

e1,e2,...
: qualquer tipo possível de expressão

# Descrição

Parênteses esquerdo e direito são usados para

* Especificar ordem de avaliação dentro de expressões,

* Formar listas de argumentos do lado direito de uma função. Dentro de lados direitos múltiplos, os argumentos devem ser separados por vírgula.

* Selecionar elementos dentro de vetores, matrizes e listas.Ver ajuda em insertion e extraction para maiores informações.

* [x1,x2,...]=(e1,e2,...) é equivalente a realizar primeiro %t_1 = e1, %t_2 = e2, ..., e então x1 = %t_1, x2 = %t_2, ..., onde as variáveis %t_i, i = 1, 2, ... são visíveis ao usuário.

# Exemplos

```
3^(-1)
x=poly(0,"x");
//
(x+10)/2
i3=eye(3,3)
//
a=[1 2 3;4 5 6;7 8 9],a(1,3),a([1 3],:),a(:,3)
a(:,3)=[]
a(1,$)=33
a(2,[$ $-1])
a(:,$+1)=[10;11;12]
//
```

```
w=ssrand(2,2,2);ssprint(w)
ssprint(w(:,1))
ss2tf(w(:,1))
//
l=list(1,2,3,4)
[a,b,c,d]=l(:)
l($+1)='novo'
//
v=%t([1 1 1 1 1])
//
[x,y,z]=(1,2,3)
```

# Ver Também

colon, comma, brackets, list, extraction, insertion

# Name

pause — modo de pausa, invoca teclado

# Descrição

Alterna para o modo `pause` ; inserido no código de uma função, `pause` interrompe a execução da mesma: recebe-se um símbolo de prompt para indicar o nível de `pause` (exemplo: `-1->`). O usuário está agora num novo espaço de trabalho em que todas as variáveis de nível infeiror (em particular todas as variáveis da função) estão disponíveis. Para retornar ao espaço de trabalho de chamamento, entre `"return"`

Neste modo, `[...]=return(...)` retorna as variáveis dos argumentos `(...)` ao espaço de trabalho de chamamento com nomes na saída `[...]`. Em caso contrário, as variáveis de nível infeiror são protegidas e não podem ser modificadas.

`pause` é extremamente útil para propósitos de depuramento.

Este modo pode ser cancelado pelo comando `"abort"`.

# Ver Também

halt, return, abort, quit, whereami, where, sleep

# Name

percent — caractere especial (%)

# Descrição

Alguns nomes de variáveis predefinidas começam com '%', tais como `%i` (para `sqrt(-1)`), `%inf` (para `Infinity`), `%pi` (para `3.14...`), `%T` (para a constante booleana`"true"`),...

Ainda, funções cujo nome começa com % são especiais : elas são usadas para primitivas e overloading ("sobrecarga") de operadores (ver `overloading`).

Por exemplo, a função `%rmr` realiza a operação de multiplicação (`m`) `x*y` para `x` e `y` matrizes de razões de polinômios (`r`). As convenções de codificação são fornecidas pelo arquivo "leia-me" no diretório `SCIDIR/macros/percent`.

# Exemplos

```
x1=tlist('x',1,2);
x2=tlist('x',2,3);
deff('x=%xmx(x1,x2)','x=list(''x'',x1(2)*x2(2),x2(3)*x2(3))');
x1*x2
```

# Ver Também

overloading

# Name

perl — chama script Perl utilizando um executável apropriado do sistema operacional

```
perl('perlfile')
perl('perlfile',arg1,arg2,...)
result = perl(...)
```

# Descrição

perl('perlfile') chama o script Perl 'perlfile', utilizando o executável Perl apropriado do sistema operacional.

perl('perlfile',arg1,arg2,...) chama o script Perl 'perlfile', utilizando o executável Perl apropriado do sistema operacional, e passa os argumentos arg1, arg2, e assim por diante, a 'perlfile'.

result = perl(...) retorna os resultados de uma chamada a Perl tentada para 'result'.

# Ver Também

unix

# Autor

A.C

# Name

plus — (+) operador de adição

```
X+Y
str1+str2
```

# Parâmetros

X,Y
: escalar, vetor ou matriz de números, polinômios ou razões de polinômios. Também pode ser uma lista `syslin`

str1,str2
: um string, um vetor ou uma matriz de strings

# Descrição

Adição.

Para operandos numéricos, é a adição em sentido usual. Se um dos operandos é uma matriz e o outro um escalar, o escalar é adicionado a cada uma das entradas da matriz. Se um dos operandos é a matriz vazia, um o outro operando é retornado (este comportamento padrão pode ser modificado pela função `mtlb_mode`).

Para strings ' + ' significa concatenação.

A adição também pode ser definida para outros tipos de dados através de operações "soft-coded" ("codificação suave") (ver `overloading`).

# Ver Também

```
[1,2]+1
[]+2
s=poly(0,"s");
s+2
1/s+2
"cat"+"enate"
```

# See Also

addf, mtlb_mode, overloading

# Name

poly — definição de polinômios

```
p=poly(a,x, ["flag"])
```

# Parâmetros

a
: matriz ou número real

x
: String, o nome da variável simbólica. Se o string tiver mais de 4 caracteres, apenas os quatro primeiros serão levados em conta.

"flag"
: string ("roots", "coeff"), valor padrão é "roots".

# Descrição

Se a é uma matriz,
: `p` é o polinômio característico, i.e., `determinant(x*eye()-a)`, `x` sendo a variável simbólica.

Se v é um vetor,

- `poly(v,"x",["roots"])` é o polinômio com entradas de `v` como raízes e `"x"` como variável formal (neste caso, `roots` e `poly` são funções inversas). Note que infinitas raízes fornecem zero coeficientes de grau mais elevado.
- `poly(v,"x","coeff")` cria o polinômio com símbolo `"x"` e com entradas de `v` como coeficientes (v(1) é o termo constante do polinômio). (aqui `poly` e `coeff` são funções inversas).

`s=poly(0,"s")` é a fonte para definir polinômios com símbolo `"s"`.

# Exemplos

```
s=poly(0,"s");p=1+s+2*s^2;
A=rand(2,2);poly(A,"x")
//frações racionais
h=(1+2*%s)/poly(1:4,'s','c')
```

# See Also

coeff, roots, varn, horner, derivat, matrices, rational

# Name

power — operação de potenciação(^,.^)

```
t=A^b
t=A**b
t=A.^b
```

# Parâmetros

A,t
: matriz de escalares, polinômios ou razões de polinômios.

b
: um escalar ou um vetor ou matriz de escalares.

# Descrição

- `"(A:square)^(b:scalar)"`Se `A` é uma matriz quadrada e `b` é um escalar, então `A^b` é a matriz `A` elevada à potência `b`.
- `"(A:matrix).^(b:scalar)"`Se `b` é um escalar e `A` uma matriz, então `A.^b` é formada pelos elementos de `A` elevados à potência `b` (potenciação elemento a elemento). Se `A` é um vetor e `b` é um escalar, então `A^b` e `A.^b` realizam a mesma operação (i.e., potenciação elemento a elemento).
- `"(A:scalar).^(b:matrix)"` Se `A` é um escalar e `b` é uma matriz (ou vetor) então `A^b` e `A.^b` são as matrizes (ou vetores) formados por `a^(b(i,j))`.
- `"(A:matrix).^(b:matrix)"` Se `A` e `b` são vetores (matrizes) de mesmo tamanho `A.^b` é o vetor `A(i)^b(i)` (matriz `A(i,j)^b(i,j)`).

Notas:

- Para matrizes quadradas `A^p` é computada através de sucessivas multiplicações de matrizes se `p` is é um número inteiro positivo e por diagonalização se não for.

- Os operadores ** e ^ são sinônimos.

# Exemplos

```
A=[1 2;3 4];
A^2.5,
A.^2.5
(1:10)^2
(1:10).^2

s=poly(0,'s')
s^(1:10)
```

# Ver Também

exp

# Name

predef — proteção de variáveis

```
n=predef()
oldnew=predef(n)
oldnew=predef('all')
oldnew=predef('clear')
```

# Descrição

Função utilitária para definir as variáveis "mais antigas" como "protegidas". Variáveis protegidas não podem ser cancelada. Elas não são salvas pelo comando `'save'` . As variáveis "mais antigas" são aquelas que aparecem por último em `who('get')`.

`predef()` retorna o número de variáveis protegidas.

`predef('a[ll]')` ajusta todas as variáveis como protegidas, também retorna o número antigo e o novo de variáveis protegidas.

`predef('c[lear]')` desprotege todas as variáveis, exceto as 7 últimas e também retorna o número antigo e o novo de variáveis protegidas.

`predef(n)` ajusta as `max(n,7)` últimas variáveis definidas como protegidas, também retorna o número antigo e o novo de variáveis protegidas.

# Observação

Um número de variáveis protegidas está configurado no arquivo de "start-up" (inicialização) `SCI/etc/scilab.start`. O usuário pode, em particular, configurar suas próprias variáveis predefinidas nos arquivos de inicialização do usuário `SCIHOME/.scilab` e `SCIHOME/scilab.ini`

Definição SCIHOME: no Windows : C:/Documents and Settings/<User>/Scilab/<Scilab-Version> No Linux/Unix : /home/<User>/.Scilab/<Scilab-Version>

# Ver Também

clear, save

# Name

quit — Decresce o nível de pausa ou termina o Scilab

```
quit
```

# Descrição

O comando `quit` tem dois significados diferentes dependendo do contexto de chamamento:

Se não houver `pause` ativo,
: Então o comando `quit` faz o Scilab terminar, mesmo que o comando seja chamado dentro de uma função.

Se houver `pause` ativo,
: Então o comando `quit` aborta instruções iniciadas neste nível de pausa e termina o nível de pausa corrente.

# Exemplos

```
// Saindo do Scilab
function foo(x),if x then quit,end,endfunction
foo(%t) //quits scilab

//terminando instrução iniciada num contexto de pausa
function foo(x),if x then quit,end,endfunction
pause
foo(%t) //retorna ao nível de prompt principal

function foo1(x),
  mprintf('P1\n')
  if x then pause, mprintf('P2\n'),end,
  mprintf('P3\n')
endfunction

foo1(%t) //enter com quit no prompt seguinte
```

# Ver Também

pause, break, abort, exit

# Name

quote — (') transpose operator, string delimiter

# Descrição

quote (aspas) ( ' ) é usado para a transposta conjugada de uma matriz.

quote ( .' ) é usado para a transposta não-conjugada de uma matriz.

Aspas simples (') ou duplas (") também são usadas para definir strings (strings são definidos por aspas nas extremidades). Aspas simples dentro de um string são denotadas por duas aspas simples e aspas duplas são denotadas por uma aspas simples e uma dupla.

# Exemplos

```
[1+%i, 2]'
[1+%i, 2].'
x='Isto é um string'
'Ele disse: ''Bom'''
```

# Name

rational — objeto do Scilab, razão de polinômios

# Descrição

Um razão de polinômios `r` é um quociente entre dois polinômios `r=num/den`. A representação interna de uma razão de polinômios é uma lista. `r=tlist('['r','num','den','dt'],num,den,[])` é o mesmo que `r=num/den`. Uma matriz de razões de polinômios pode ser definida pela sintaxe usual, por exemplo: `[r11,r12;r21,r22]` ié uma matriz 2x2 onde `rij` são razões 1x1. Uma matriz de razões de polinômios também pode ser definida como acima como uma lista `tlist(['r','num','den','dt'],num,den,[])` com `num` e `den` matrizes de polinômios.

# Exemplos

```
s=poly(0,'s');
W=[1/s,1/(s+1)]
W'*W
Num=[s,s+2;1,s];Den=[s*s,s;s,s*s];
tlist(['r','num','den','dt'],Num,Den,[])
H=Num./Den
syslin('c',Num,Den)
syslin('c',H)
[Num1,Den1]=simp(Num,Den)
```

# Ver Também

poly, syslin, simp

# Name

readgateway — retorna a lista de primitivas de um módulo

```
readgateway(module_name)
primitives = readgateway(module_name);
[primitives,primitivesID] = readgateway(module_name);
[primitives,primitivesID,gatewayID] = readgateway(module_name);
```

# Descrição

Retorna a lista de primitivas de um módulo.

Primitives : lista de primitivas de um módulo.

primitivesID : lista de identificadores das primitivas.

gatewayID : lista de identificadores de portas de ligaçãos associados ao módulo.

# Exemplos

```
[primitives,primitivesID,gatewayID] = readgateway('core');
primitives(1) // primitiva 'debug' primitive
primitivesID(1) // 1 é o ID de 'debug' na porta de ligação de 'core'
gatewayID(1) // 13 é o ID da porta de ligação de 'core' no Scilab
```

# Ver Também

getmodules

# Name

resume — retorna ou pára a execução de uma função e copia algumas de suas variáveis locais

```
resume
[x1,..,xn]=resume(a1,..,an)
```

# Parâmetros

x

...

# Descrição

Em uma função, `resume` pára a execução da mesma, `[..]=resume(..)` pára a execução de uma função e põe as variáveis locais `ai` no ambiente de chamamento sob os nomes de `xi`.

No modo `pause` , permite retornar a um nível inferior `[..]=resume(..)` retorna ao nível inferior e põe as variáveis locais `ai` no ambiente de chamamento sob os nomes de `xi`.

Em um `execstr` chamado por uma função, `[..]=resume(..)` pára a execução da função e põe as variáveis locais `ai` no ambiente de chamamento sob os nomes de `xi`.

`resume` é equivalente a `return`.

# Ver Também

abort, break

# Name

return — retorna ou pára a execução de uma função e copia algumas de suas variáveis locais

```
return
[x1,..,xn]=return(a1,..,an)
```

# Parâmetros

x

...

# Descrição

Em uma função, `return` pára a execução da mesma, `[..]=return(..)` pára a execução de uma função e põe as variáveis locais `ai` no ambiente de chamamento sob os nomes de `xi`.

No modo `pause` , permite retornar a um nível inferior `[..]=return(..)` retorna ao nível inferior e põe as variáveis locais `ai` no ambiente de chamamento sob os nomes de `xi`.

Em um `execstr` chamado por uma função, `[..]=return(..)` pára a execução da função e põe as variáveis locais `ai` no ambiente de chamamento sob os nomes de `xi`.

`resume` é equivalente a `return`.

# Ver Também

abort, break

# Name

sciargs — scilab command line arguments

```
args=sciargs()
```

# Descrição

Esta função retorna um vetor de strings contendo os argumentos da linha de comando do Scilab. As primeiras entradas de `args` contém o endereço do arquivo executável rodado.

Esta função corresponde a `getarg` na linguagem C.

# Ver Também

getenv

# Name

scilab — principal script unix para executar o Scilab e ferramentas de miscelânea

```
scilab <Options>
```

# Descrição

-args Arguments

Se esta opção estiver presente, os argumentos são passados ao Scilab. Eles podem ser recebidos pela função sciargs. para passagem de múltiplos argumentos use, seqüências de palavras separadas por espaços entre aspas simples: `scilab -args 'foo1 foo2'`

-display Display

Para uso apenas em sistemas Xwindow para ajustar um display de servidor X. O display padrão é unix:0.0

`-display` pode ser abreviado por `-d`

-debug

Inicia o Scilab no debugger gdb (apenas para Unix/linux).

-e Instrução

Se esta opção estiver presente, então a instrução Scilab `Instrução` é executada primeiro (logo após a execução do arquivo de inicialização) no Scilab. As opções `-e` e `-f` são mutuamente exclusivas

-f arquivo

Se esta opção estiver presente, então o script Scilab `arquivo` é executado primeiro (logo após a execução do arquivo de inicialização) no Scilab. As opções `-e` e `-f` são mutuamente exclusivas.

-l idioma

Se esta opção estiver presente, ela fica o idioma do usuário. Os possíveis valores para `idioma` são `'fr'` para francês, `'en'` para inglês e 'br' para português brasileiro. O idioma padrão é inglês. O valor padrão é fixado no arquivo `scilab.start`.

-mem N

Ajusta o tamanho de pilha inicial, para uso com a opção -ns. Sem a opção -ns, a pilha inicial é ajustada pelo script scilab.start.

-nb

Se esta opção está presente, então o cartão de boas-vindas não é impresso.

-ns

Se esta opção está presente, o arquivo de inicialização `SCI/etc/scilab.start` e os arquivos de inicialização do usuário `SCIHOME/.scilab`, `SCIHOME/scilab.ini` não são executados.

-nouserstartup

Se esta opção estiver presente, os arquivos de inicialização do usuário `SCIHOME/.scilab`, `SCIHOME/scilab.ini` não são executados.

-nw

Se esta opção estiver presente, então o Scilab não é executado em uma janela específica.

-nwni

Se esta opção estiver presente, então o Scilab não é executado em uma janela específica e não aceita interação do usuário. Esta opção pode ser utilizada com as opções -f ou -e .

--texmacs
Esta opção é reservada para TexMacs.

-version
Esta opção imprime a versão do produto e sai.

# Name

select — Palavra-chave da estrutura select (significa "selecionar (de acordo com a variável)")

# Descrição

```
select expressão,
  case expressão1 then instruções1,
  case expressão2 then instruções2,
  ...
  case expressãon then instruçõesn,
  [else instruções],
end
```

Notas:

- A única restrição é de que a palavra-chave "then" deve estar na mesma linha que a palavra-chave "case" correspondente.
- A palavra-chave "then" pode ser substituída por uma vírgula ou um retorno de carro. `instruçõesi` são executadas se `expressãoi=expressão`.

Aviso: o número de caracteres usados para definir o corpo de qualquer instrução condicional (if while for ou select/case) deve ser limitado a 16k.

# Exemplos

```
while %t do
  n=round(10*rand(1,1))
  select n
  case 0 then
    disp(0)
  case 1 then
    disp(1)
  else
    break
  end
end
```

# Ver Também

if, while, for

# Name

semicolon (;) — (;) fim de expressão e separador de linhas

# Descrição

Pontos-e-vírgulas são usados para separar linhas em uma definição de matrizes.

Pontos-e-vírgulas também podem ser usados para indicar o fim de uma instrução. Neste caso, indica que o(s) resultado(s) não serão exibidos. Se desejar exibição, use vírgula (,).

# Exemplos

```
a=[1,2,3;4,5,6];
a=1;b=1,c=2
```

# Ver Também

comma, brackets

# Name

setbpt — ajusta pontos de parada

```
setbpt(macroname [,linenumb])
```

# Parâmetros

macroname
: string

linenumb
: escalar inteiro ou vetor de escalares

# Descrição

`setbpt` insere interativamente um ponto de parada na linha de número `linenumb` (valor padrão é 1) da função `macroname`

`linenumb` pode ser um vetor linha ou coluna dos números das linhas, ou um único número de linha escalar.

Quando alcança o ponto de parada, o Scilab avalia a linha especificada, imprime o número da linha e da função. Se a função não for compilada (see `comp`) , a linha é exibida na tela. Então, o Scilab entra em modo `pause` no qual o usuário pode verificar os valores correntes. O `pause` é cancelado com `resume` ou `abort`. Redefinir a função não limpa pontos de parada, o usuário deve deletar os pontos de parada explicitamente usando `delbpt`. O número máximo de funções com pontos de parada habilitados deve ser menor que 100 e o número máximo de pontos de parada é 1000.

# Exemplos

```
setbpt('foo'),setbpt('foo',10),dispbpt()
delbpt()

setbpt('foo',[1,2,5,6]),dispbpt()
```

# Ver Também

delbpt, dispbpt, pause, resume

# Name

sethomedirectory — ajusta o diretório home do Scilab

```
[home,scilabhome] = sethomedirectory()
```

# Descrição

Ajusta o endereço hom do Scilab : variável "SCIHOME".

No Windows 2k e XP , C:\Documents and Settings\<User>\Scilab\<Scilab-Version>

No Windows Vista , C:\Users\<User>\Scilab\<Scilab-Version>

No Unix, /home/<User>/.Scilab/<Scilab-Version>

# Autor

Allan CORNET

# Name

slash — (/) divisão esquerda-direita e feedback ("resposta")

# Descrição

Divisão esquerda-direita. `x=A / b` é a solução de `x*b=A`.

`b/a = (a' \ b')'`.

`a ./ b` é a matriz com entradas `a(i,j)/ b(i,j)`. Se `b` é escalar (matriz 1x1) esta operação é o mesmo que `a./b*ones(a)`. (mesma convenção se `a` é um escalar).

Observe que `123./b` é interpretado como `(123.)/b`. Nestes casos, o ponto é parte do número, não do operador.

(\) significa divisão direita-esquerda.

Feedback de sistema. `S=G/.K` avalia `S=G*(eye()+K*G)^(-1)`. Este operador evita o problema de simplificação.

Observe que `G/.5` é interpretado como `G/(.5)`. Em tais casos, o ponto é parte do número, não do operador.

Comentário. `//` comenta uma linha, i.e. linhas que começam por `//` são ignoradas pelo interpretador.

# Ver Também

inv, percent, backslash, ieee

# Name

stacksize — ajusta tamanho da pilha do Scilab

```
stacksize(n)
stacksize('max')
stacksize('min')
sz=stacksize()
```

# Parâmetros

n
: inteiro, o tamanho da pilha requerido dado em número de palavras de dupla precisão

sz
: vetor de duas entradas [total used]

# Descrição

O Scilab armazena variáveis "usuais" em uma pilha `stk` (para variáveis globais, ver `gstacksize`).

`stacksize(n)` permite aumentar ou diminuir o tamanho desta pilha. O máximo permitido depende da quantidade de memória livre e do espaço de troca ("swap space") disponível no momento.

`stacksize('max')` permite ao usuário aumentar o tamanho dessa pilha ao máximo.

`stacksize('min')` permite ao usuário diminuir o tamanho dessa pilha ao mínimo.

Esta função com o argumento `n` pode ser usada em qualquer lugar.

`sz=stacksize()` retorna um vetor de duas entradas que contém os tamanhos correntes de pilha total e usado.

# Ver Também

who, gstacksize

# Name

star — (*) operador de multiplicação

# Descrição

Multiplicação no sentido usual. Válido para matrizes de contrantes, valores booleanos, polinômios ou razões de polinômios e para listas `syslin` (o significado é de conexão de série).

A multiplicação elemento a elemento é denotada por `x.*y`. Se `x` ou `y` é escalar (matriz 1x1) `.*` é o mesmo que `*`.

O produto de Kronecker é denotado por `x.*.y`

`A*.B` é um operador sem significado predefinido. Pode ser usado para definir um novo operador (ver "overloading") com a mesma precedência que `*` ou /.

# Ver Também

slash, backslash, syslin

# Name

startup — arquivo de inicialização

# Descrição

O arquivo de inicialização `SCIHOME/.scilab` e `SCIHOME/scilab.ini` são automaticamente executados (se estiverem presentes) quando o Scilab é chamado, junto com o arquivo `scilab.star` no diretório Scilab (SCI).

# Observações

A última linha de um arquivo de inicialização deve ser terminada por uma nova linha a ser levada em conta.

Definição de SCIHOME : no Windows : C:/Documents e Settings/<User>/Scilab/<Scilab-Version>

ou no Vista : C:/<User>/AppData/Roaming/Scilab/<Scilab-Version>

No Linux/Unix : /home/<User>/.Scilab/<Scilab-Version>

# Ver Também

scilab

# Name

symbols — nomes dos operadores Scilab

# Descrição

Use os nomes seguintes para obter ajuda em um símbolo específico

| operator | nome na ajuda do Scilab |
|---|---|
| ',", .' | quote (aspas) |
| + | plus (sinal de mais) |
| - | minus (sinal de menos) |
| *,.* | star (estrela, asterísco) |
| /, ./, /. | slash (barra) |
| \,.\ ,\. | backslash (barra invertida) |
| . | dot (ponto) |
| =, == | equal (igual) |
| <,>,>=,<=,<> | less (desigualdades ) |
| ~ | tilda (til) |
| [ | left (colchete esquerdo) |
| ] | right (colchete direito) |
| () | parents (parênteses) |
| % | percent |
| : | colon (dois-pontos) |
| , | comma (vírgula) |
| ; | semi (ponto-e-vírgula) |
| ^ | hat (chapéu, circunflexo) |
| .^ | power (potência) |
| \| | or (barra vertical, "ou" lógico) |
| & | and ("e" comercial, "e" lógico) |
| .*., ./., .\. | kron (produto de Kronecker) |

# Observações

Por razões históricas, símbolos diferentes podem representar o mesmo operador:

`{` tem o mesmo significado que `[`

`}` tem o mesmo significado que `]`

`@` tem o mesmo significado que `~`

` ` ` tem o mesmo significado que <

É altamente recomendável que não se use esses recursos, pois eles serão removidos futuramente

# Ver Também

overloading

# Name

testmatrix — gera algumas matrizes particulares

```
[y]=testmatrix(name,n)
```

# Parâmetros

name
: um string

n
: inteiros , tamanho da matriz

y
: matriz `n` x `m`

# Descrição

Cria algumas matrizes particulares

testmatrix('magi',n)
: retorna um quadrado mágico de tamanho `n` .

testmatrix('frk',n)
: retorna a matriz de Franck :

testmatrix('hilb',n)
: é a matriz inversa da matriz n x n de Hilbert `(Hij= 1/(i+j-1))`.

## Name

then — Palavra-chave usada na estrutura "if-then-else" (significa "então...")

## Descrição

Usado com `if`.

## Ver Também

if

# Name

tilda — (~) não lógico

```
~m
```

# Parâmetros

m
: matriz de valores booleanos

# Descrição

`~m` é a negação de `m`.

# Name

try — início de um bloco try numa instrução de controle try-catch (significa "tentar...")
catch — início de um bloco catch numa instrução de controle try-catch (significa "prender...")

```
try
statements
catch
statements
end
```

# Descrição

A instrução de controle `try-catch`pode ser usada para gerenciar códigos que possivelmente poderiam gerar erros.

Quando uma instrução de controle `try-catch` é executada, normalmente apenas as sentenças entre as palavras-chaves `try` e `catch`são executadas. Contudo, se um erro ocorre durante a execução de uma dessas sentenças, o erro é registrado, as sentenças restantes até `catch`são puladas e as sentenças entre as palavras-chaves `catch` e `end` são executadas usando-se o modo de manipulação de erros padrão (ver: errcatch).

A mensagem de erro registrada pode ser recuperada pela função lasterror .

As sentenças `catch` , tanto quanto a palavra-chave `catch` podem ser omitidas se nenhuma sentença alternativa for fornecida.

Note que também é possível utilizar a função execstr com o argumento `'errcatch'` para manipulação de erros. Isto pode ser particularmente útil para manipulação de erros sintáticos.

# Exemplos

```
file_path=TMPDIR+'/wrong
try
  u=mopen(file_path,'r')
  x=mget(10,'c',u)
catch
  disp(['o arquivo '+file_path+ ' não pode ser lido',
        'usando valores padrões para x'])
  x=1:10
end
[error_message,error_number]=lasterror(%t)
```

# Ver Também

error, execstr, if, lasterror, errcatch

# Autor

Serge Steer, INRIA

# Name

type — tipo de variável

```
[i]=type(x)
```

# Parâmetros

x
: objeto Scilab

i
: inteiro

# Descrição

`type(x)` retorna um inteiro que indica o tipo de `x` como segue:

1 : uma matriz constante de reais ou complexos;

2 : uma matriz de polinômios;

4 : uma matriz de valores booleanos;

5 : uma matriz esparsa;

6 : uma matriz de valores booleanos esparsa;

7 : uma matriz esparsa do Matlab.

8 : uma matriz de inteiros armazenados em 1, 2 ou 4 bytes;

9 : uma matriz de manipuladores gráficos;

10 : uma matriz de strings;

11 : uma função não-compilada (código Scilab);

13 : uma função compilada (código Scilab);

14 : uma biblioteca de funções;

15 : uma lista;

16 : uma lista com tipo ("typed list" ou "tlist");

17 : uma lista com tipo orientada matricialmente (mlist);

128 : um ponteiro (ver lufact).

129 : polinômio de tamanho implícito, usado para indexação.

130 : intrínseco do Scilab (código C ou Fortran).

# Ver Também

typeof

# Name

typename — associa um nome ao tipo de variável

```
[types [ [,names]]=typename()
typename(name,type)
```

# Parameters

types
: vetor coluna de inteiros: os códigos dos tipos de cada tipo de dado definido.

names
: vetor coluna de strings: os nomes associados aos códigos de tipos.

type
: inteiro: o código de tipo do novo tipo de dado.

name
: string: o nome associado ao código de tipo

# Descrição

O overloading de funções e operadores faz uso de nomes formais associados aos tipos de dados para formar o nome da função de overloading (ver overloading). `typename` pode ser usado para manipular esses nomes formais para tipos de dados "hard-coded" (os nomes formais de tipos de dados codificados de `tlist` ou `mlist` são definidos de outra forma, ver `overloading`).

Chamado sem argumento do lado direito, `typename` retorna informações sobre tipos de dados definidos.

Chamado com argumento do lado direito, `typename` associa um nome a um código de tipo de dado.

`typename('',type)` suprime o tipo de dado fornecido pelo seu código `type` de tabelas de tipos de dados conhecidos.

Number max. of defined types is 50.

# Ver Também

type, typeof, overloading, tlist, mlist

# Name

user — interface para rotinas FORTRAN ou C

```
[s_1,s_2,...,s_lhs]=user(e_1,e_2,...,e_rhs)
```

# Descrição

Com este comando, é possível utilizar um programa externo como um comando do Scilab onde `(s_1,s_2,...,s_lhs)` são as variáveis de saída e `(e_1,e_2,...,e_rhs)` asão as variáveis de entrada. Para inserir este comando no Scilab, é necessário escrever algumas linhas na subrotina FORTRAN `user` do Scilab. Ver `intersci` ou a documentação do Scilab para mais .

# Ver Também

fort, link

# Name

varn — variável simbólica de um polinômio

```
[symb]=varn(p)
[pm]=varn(x,var)
```

# Parâmetros

p
: polinômio (matriz de polinômios)

symb
: string

x
: polinômio ou matriz de polinômios

var
: variável simbólica (string)

pm
: matriz ou matriz de polinômios

# Descrição

`symb=varn(p)` retorna em `symb` a variável simbólica de `p` (i.e. `varn(poly(0,'x'))` is `'x'`).

`varn(x,'s')` retorna uma matriz de polinômios com os mesmos coeficientes que `x` mas com `'s'` como variável simbólica (mudança do nome de variável).

# Exemplos

```
s=poly(0,'s');p=[s^2+1,s];
varn(p)
varn(p,'x')
```

# Ver Também

horner, poly

# Name

ver — informação de versão sobre Scilab

```
r = ver()
```

# Parâmetros

r
: uma matriz de strings

# Descrição

Informação de versão sobre o Scilab.

Retorna uma matriz de strings com informações de versão sobre o Scilab.

# Exemplos

```
ver
```

# Autor

A.C

# Name

warning — mensagens de aviso

```
warning('string')
warning('off')
warning('on')
mode = warning('query')
```

# Descrição

Imprime o string `'string'` em uma mensagem de aviso

`'on'` habilita mensagens de aviso.

`'off'` desabilita mensagens de aviso.

`'query'` retorna o estado 'on' ou 'off'.

# Exemplos

```
warning('on')
warning('este é um aviso')
warning('off')
warning('este é um aviso')
warning('query')
warning('on')
```

# Ver Também

error

# Name

what — lista de primitivas do Scilab

```
what()
[primitives,commands]=what();
```

# Descrição

Lista de comandos e primitivas de nível baixo.

# Autor

A.C

# Name

where — retorna a árvore de chamamento de instruções corrente

```
[linenum,mac]=where()
```

# Parâmetros

linenum
: vetor coluna de inteiros

mac
: vetor coluna de strings

# Descrição

Retorna `linenum` e `mac` tais que a instrução corrente foi chamada pela linha linenum(1) da função mac(1), mac(1) foi chamada pela linha linenum(2) da função mac(2), e assim por diante.

mac(i) é, em geral o nome de uma função, mas também pode ser "exec" ou "execstr" se a instrução reside em um arquivo executável ou em uma instrução execstr.

# Ver Também

whereami, pause

# Name

whereami — exibe a árvore de chamamento de instruções corrente

```
whereami()
```

# Descrição

Exibe a árvore de chamamento para a instrução que contém whereami(). Pode ser usado dentro de níveis de pause (pausa).

# Exemplos

```
deff('y=test(a)',['y=sin(a)+1';
                  'y=t1(y)';
                  'y=y+1'])
deff('y=t1(y)',['y=y^2';'whereami()'])
test(1)
```

# Ver Também

where, pause, errcatch

# Name

while — palavra-chave da estrutura while (significa "enquanto...")

# Descrição

`while` . Deve ser terminado por `"end"`

`while expressão ,instruções1,...[,else instruções], end`

`while expressão do instruções1,...[,else instruções], end`

`while expressão then instruções1,...[,else instruções], end`

Notas:

- A única restrição é que cada palavra-chave "then" ou "do" deve estar na mesma linha que a palavra-chave "while".
- As palavras-chaves "then" ou "do" podem ser substituídas por um retorno de carro ou uma vírgula. Para compatibilidade com o Matlab, também é possível, mas não recomendado, por um espaço entre o fim da expressão e o início da primeira instrução.
- A construção opcional `,else instruções` permite fornecer instruções que são executadas quando a expressão expressão torna-se falsa.

Aviso: o número de caracateres usados para definir o corpo de qualquer estrutura condicional (if while for ou select/case) deve ser limitado a 16k.

# Exemplos

```
e=1; a=1; k=1;
while norm(a-(a+e),1) > %eps, e=e/2; k=k+1; end
e,k
```

# Ver Também

for, select, break, return, pause

# Name

who — listagem de variáveis

```
who
who()
names=who('local')
[names,mem]=who('local')
names=who('global')
[names,mem]=who('global')
who('sorted')
names=who('local','sorted')
[names,mem]=who('local','sorted')
names=who('global','sorted')
[names,mem]=who('global','sorted')
```

# Descrição

`who` exibe os nome de variáveis correntes.

`who('local')` ou `who('get')` retorna os nomes de variáveis correntes e a memória utilizada em palavras de dupla precisão.

`who('global')` retorna os nomes de variáveis globais e a memória utilizada em palavras de dupla precisão.

`who('sorted')` exibe todas as variáveis em ordem alfabética.

# Ver Também

whos, who_user

# Name

who_user — listagem das variáveis do usuário

```
who_user()
```

# Descrição

`who_user` exibe os nomes das variáveis do usuário.

# Ver Também

whos, who

# Name

whos — listing of variables in long form

```
whos()
whos -type typ
whos -name nam
```

# Parâmetros

typ
: nome do tipo da variável selecionada (ver typeof)

nam
: os primeiros caracteres dos nomes selecionados

# Descrição

`whos()` exibe todos os nomes de variáveis correntes, bem como seus tipos e memória utilizada.

`whos -type typ` exibe todas as variáveis correntes com o tipo especificado.

`whos -name nam` exibe todas as variáveis cujos nomes começam com `nam`.

Nota : se uma variável é global, um "*" aparece em frente ao seu tipo.

# Exemplos

```
lines(0)
whos()
whos -type boolean
whos -name %
```

# Ver Também

who, typeof

# Name

with_atlas — Checa se o Scilab foi construído com a biblioteca "Atlas"

```
r=with_atlas()
```

# Parâmetros

r
: um booleano

# Descrição

Retorna `%t` se o Scilab tiver sido construído com a biblioteca "Atlas", ou `%f` senão.

# Name

with_gtk — Checa se o Scilab foi contstruído com a biblioteca "GIMP Toolkit"

```
r=with_gtk()
```

# Parameters

r
: um booleano

# Description

Sempre retorna `%f` , a biblioteca gtk não é mais suportada pelas versões 5 em diante.

# Name

with_javasci — Checa se o Scilab foi construído com a interface java

```
r=with_javasci()
```

# Parâmetros

r
: um booleano

# Descrição

Retorna `%t` se o Scilab tiver sido construído com a interface java, ou `%f` senão.

# Name

with_macros_source — Verifica se a fonte de macros está instalada

```
r=with_macros_source()
```

# Parâmetros

r
: um booleano

# Descrição

Retorna `%t` se a fonte de macros está instalada, ou `%f` senão.

# Name

with_module — checa se um módulo Scilab foi instalado

```
r=with_module(module_name)
```

# Parâmetros

r
: um booleano

module_name
: um string. Exemplo : 'core'

# Descrição

Retorna `%t` se o módulo Scilab tiver sido instalado.

# Ver Também

getmodules

# Autor

A.C

# Name

with_pvm — Checa se o Scilab foi construído com a interface "Parallel Virtual Machine"

```
r=with_pvm()
```

# Parâmetros

r
: um booleano

# Descrição

Retorna `%t` se o Scilab tiver sido construído com a interface "Parallel Virtual Machine", ou `%f` senão.

# Name

with_texmacs — checa se o Scilab foi chamado por TeXmacs

```
r=with_texmacs()
```

# Parâmetros

r
: um booleano

# Descrição

Retorna `%t` se o Scilab tiver sido chamado por TeXmacs

# Name

with_tk — checa se o Scilab foi construído com TCL/TK

```
r=with_tk()
```

# Parâmetros

r
: um booleano

# Descrição

Retorna `%t` se o Scilab tiver sido construído com a interface TCL/TK ou `%f` se não.

# Parte II. Equações Diferenciais

# Name

dae — Solucionador de equações diferenciais algébricas

```
 y=dae(initial,t0,t,res)
[y [,hd]]=dae(initial,t0,t [,rtol, [atol]],res [,jac] [,hd])
[y,rd]=dae("root",initial,t0,t,res,ng,surface)
[y ,rd [,hd]]=dae("root",initial,t0,t [,rtol, [atol]],res [,jac], ng, surface [
```

# Parâmetros

initial
: um vetor coluna. Pode ser igual a `x0` or `[x0;xdot0]`, onde `x0` é o valor de estado no tempo inicial `t0` e `ydot0` é o valor da derivada do estado inicial ou uma estimativa dela (ver abaixo).

t0
: número real, o tempo inicial

t
: escalar real ou vetor. Fornece instantes para os quais você deseja uma solução. Note que você pode obter soluções para cada ponto de passo de dae fazendo `%DAEOPTIONS(2)=1`.

rtol
: escalar real ou vetor coluna com o mesmo tamanho que `x0`. A tolerância do erro relativo da solução. Se `rtol` for um vetor, as tolerâncias são especificadas para cada componente do estado.

atol
: escalar real ou vetor coluna com o mesmo tamanho que `x0`. A tolerância do erro absoluto da solução. Se `atol` for um vetor, as tolerâncias são especificadas para cada componente do estado.

res
: uma função externa (external). Computa o valor de `g(t,y,ydot)`. Pode ser

  Uma função do Scilab
  : Neste caso, a sua seqüência de chamamento pode ser `[r,ires]=res(t,x,xdot)` e `res` deve retornar o resíduo `r=g(t,x,xdot)` e o indicador de erro `ires`. `ires = 0` se `res` obtiver sucesso ao computar `r`, `=-1` se o resíduo é indefinido localmente para `(t,x,xdot)`, `=-2` se os parâmetros estão fora do intervalo admissível.

  Uma lista
  : Esta forma é utilizada para passar parâmetros à função. Deve ser como segue:

    ```
    list(res,p1,p2,...)
    ```

    Onde a seqüência de chamamento da função `res` é agora

    ```
    r=res(t,y,ydot,p1,p2,...)
    ```

    `res` ainda retorna o valor residual como uma função de `(t,x,xdot,x1,x2,...)`, e p1,p2,... são parâmetros da função.

  Um string
  : Deve se referir ao nome subrotina C ou Fortran. Supondo que <r_name> ié o nome dado.

    - `A seqüência de chamamento em Fortran deve ser`

```
<r_name>(t,x,xdot,res,ires,rpar,ipar)

double precision t,x(*),xdot(*),res(*),rpar(*)

integer ires,ipar(*)
```

- A seqüência de chamamento em C deve ser

  C2F(<r_name>)(double *t, double *x, double *xdot, double *res, integer *ires, double *rpar, integer *ipar)

onde

- `t` é o valor de tempo corrente
- `x` é o array de estados
- `xdot` é o array das derivadas dos estados
- res é o array de resíduos
- `ires` é o indicador de execução
- `rpar` é o array de valores de parâmetros em ponto flutuante, necessário, mas não pode ser definido pela função `dae`
- `ipar` é o array de valores de parâmetros inteiros, necessário, mas não pode ser definido pela função `dae`

jac

uma função externa (external). Computa o valor de `dg/dx+cj*dg/dxdot` para um dado valor do parâmetro `cj. Pode ser`

Uma função do Scilab

Sua seqüência de chamamento deve ser `r=jac(t,x,xdot,cj)` e a função `jac` deve retornar `r=dg(t,x,xdot)/dy+cj*dg(t,x,xdot)/dxdot` onde `cj` é um escalar real.

Uma lista

Esta forma é utilizada para passar parâmetros à função. Deve ser como segue:

```
list(jac,p1,p2,...)
```

Onde a seqüência de chamamento da função `jac` é agora

```
r=jac(t,x,xdot,p1,p2,...)
```

`jac` ainda retorna `dg/dx+cj*dg/dxdot` como uma função de `(t,x,xdot,cj,p1,p2,...)`.

Um string

Deve se referir ao nome de uma subrotina C ou Fortran. Supondo que <j_name> é o nome dado,

- `A seqüência de chamamento em Fortran deve ser`

  `<j_name>(t, x, xdot, r, cj, ires, rpar, ipar)`

```
double precision t, x(*), xdot(*), r(*), ci, rpar(*)
integer ires, ipar(*)
```

- A seqüência de chamamento dem C deve ser

  C2F(<j_name>)(double *t, double *x, double *xdot, double *r, double *cj,

  integer *ires, double *rpar, integer *ipar)

onde `t`, x, xdot, ires, rpar, ipar são definidas semelhantemente como acima, r é o array de resultados.

surface
: uma função externa (external). Computa o valor de cada vetor coluna `surface(t,x)` como componentes `ng`. Cada componente define uma superfície.

Uma função do Scilab
: Sua seqüência de chamamento deve ser `r=surface(t,x)`, esta função deve retornar um vetor com `ng` elementos.

Uma lista
: Esta forma é utilizada para passar parâmetros à função. Deve ser como segue:

```
list(surface,p1,p2,...)
```

Onde a seqüência de chamamento da função `surface` é agora

```
r=surface(t,x,p1,p2,...)
```

String
: Deve se referir ao nome de uma rotina C ou Fortran. Supondo que <s_name> é o nom dado,

- `A seqüência de chamamento em Fortran deve ser`

  `<r_name>(nx, t, x, ng, r, rpar, ipar)`

  `double precision t, x(*), r(*), rpar(*)`

  `integer nx, ng,ipar(*)`

- A seqüência de chamamento em C deve ser

  C2F(<r_name>)(double *t, double *x, double *xdot, double *r, double *cj,

  integer *ires, double *rpar, integer *ipar)

onde `t`, x, rpar, ipar são definidas semelhantemente como acima, `ng` é o número de superfícies, `nx` é a dimensão do estado e r é o array de resultados.

rd
: um vetor com duas entradas `[times num]` `times` é o valor do tempo no qual a superfície é cruzada, `num` é o número da superfície cruzada.

hd
: um vetor de reais que permite armazenar o contexto de `dae`. Pode ser utilizado como argumento de entrada para retomar a integração (recomeço rápido).

y
: matriz de reais. Se `%DAEOPTIONS(2)=1`, cada coluna é o vetor `[t;x(t);xdot(t)]` onde `t` é o índice do tempo para o qual a solução foi computada. De outro modo, `y` é o vetor `[x(t);xdot(t)]`.

# Descrição

A função `dae` é uma porta construída sobre as funções dassl e dasrt designada para equações diferenciais implícitas.

```
g(t,x,xdot)=0
x(t0)=x0  e   xdot(t0)=xdot0
```

Se `xdot0` não for fornecido no argumento `initial`, a função dae tenta computá-lo resolvendo g(t,x0,xdot0)=0,

Se `xdot0` for fornecido no argumento `initial`, pode ser tanto uma derivada compatível satisfazendo g(t,x0,xdot0)=0 ou um valor aproximado . No último caso `%DAEOPTIONS(7) deve ser ajustado para 1.`

Exemplos detalhados utilizando funções externas codificadas em C e Fortran são dados em `modules/differential_equations/tests/unit_tests/dassldasrt.tst`

# Exemplos

```
//Exemplo com um código Scilab
function [r,ires]=chemres(t,y,yd)
  r(1) = -0.04*y(1) + 1d4*y(2)*y(3) - yd(1);
  r(2) =  0.04*y(1) - 1d4*y(2)*y(3) - 3d7*y(2)*y(2) - yd(2);
  r(3) =       y(1) +     y(2)      + y(3)-1;
  ires =  0;
endfunction
function pd=chemjac(x,y,yd,cj)
  pd=[-0.04-cj , 1d4*y(3)               , 1d4*y(2);
       0.04    ,-1d4*y(3)-2*3d7*y(2)-cj ,-1d4*y(2);
       1       , 1                      , 1       ]
endfunction

x0=[1; 0; 0];
xd0=[-0.04; 0.04; 0];
t=[1.d-5:0.02:.4, 0.41:.1:4, 40, 400, 4000, 40000, 4d5, 4d6, 4d7, 4d8, 4d9, 4d1

y=dae([x0,xd0],0,t,chemres);// retorna os pontos de observação requisitados

%DAEOPTIONS=list([],1,[],[],[],0,0); // pede que a malha de dae seja retornada
y=dae([x0,xd0],0,4d10,chemres); // sem jacobiano
y=dae([x0,xd0],0,4d10,chemres,chemjac); // com jacobiano

//exemplo com um código C (requer-se um compilador C) -------------------------
//-1- criando os códigos C em TMPDIR - equação de Vanderpol, forma implícita
code=['#include &lt;math.h&gt;'
      'void res22(double *t,double *y,double *yd,double *res,int *ires,double *
      '{res[0] = yd[0] - y[1];'
      ' res[1] = yd[1] - (100.0*(1.0 - y[0]*y[0])*y[1] - y[0]);}'
      ' '
```

```
      'void jac22(double *t,double *y,double *yd,double *pd,double *cj,double *
      '{pd[0]=*cj - 0.0;'
      ' pd[1]=    - (-200.0*y[0]*y[1] - 1.0);'
      ' pd[2]=    - 1.0;'
      ' pd[3]=*cj - (100.0*(1.0 - y[0]*y[0]));}'
      ' '
      'void gr22(int *neq, double *t, double *y, int *ng, double *groot, double
      '{ groot[0] = y[0];}']
mputl(code,TMPDIR+'/t22.c')

//-2- compilando e carregando
ilib_for_link(['res22' 'jac22' 'gr22'],'t22.c',[],'c',TMPDIR+'/Makefile',TMPDIR
exec(TMPDIR+'/t22loader.sce')

//-3- executando
rtol=[1.d-6;1.d-6];atol=[1.d-6;1.d-4];
t0=0;y0=[2;0];y0d=[0;-2];t=[20:20:200];ng=1;

//simulação simples
t=0:0.003:300;
yy=dae([y0,y0d],t0,t,atol,rtol,'res22','jac22');
clf();plot(yy(1,:),yy(2,:))

//achando o primeiro ponto onde yy(1)=0
[yy,nn,hotd]=dae("root",[y0,y0d],t0,300,atol,rtol,'res22','jac22',ng,'gr22');
plot(yy(1,1),yy(2,1),'r+')
xstring(yy(1,1)+0.1,yy(2,1),string(nn(1)))

//recomeço rápido para o próximo ponto
t01=nn(1);[pp,qq]=size(yy);y01=yy(2:3,qq);y0d1=yy(3:4,qq);
[yy,nn,hotd]=dae("root",[y01,y0d1],t01,300,atol,rtol,'res22','jac22',ng,'gr22',
plot(yy(1,1),yy(2,1),'r+')
xstring(yy(1,1)+0.1,yy(2,1),string(nn(1)))
```

# Ver Também

ode, daeoptions, dassl, impl, fort, link, external

# Name

daeoptions — ajusta opções para o solucionador de equações diferenciais algébricas

```
daeoptions()
```

# Descrição

Se no contexto do chamamento da função dae existe a variável `%DAEOPTIONS`, a função dae a utiliza para ajustar suas opções.

A função daeoptions exibe interativamente um comando que deveria ser executado para se ajustar diversas opções do solucionador dae.

CUIDADO: a função `dae` verifica se esta variável existe e, neste caso, a utiliza. Para utilizar valores padrões, você deve limpar esta variável. Note que `daeoptions` não cria esta variável. Para criá-la, você deve executar a linha de comando exibida por `daeoptions`.

A variável `%DAEOPTIONS` é uma lista list com os seguintes elementos:

```
list(tstop,imode,band,maxstep,stepin,nonneg,isest)
```

O valor padrão é:

```
list([],0,[],[],[],0,0)
```

Os siginificados destes elementos são listados abaixo.

tstop
: um escalar real ou uma matriz, fornece o tempo máximo para o qual se permite a avaliação de `g`. Uma matriz vazia significa que "não há limites" impostos pelo tempo.

imode
: Se for 0, dae retorna apenas os valores de pontos do tempo especificado pelo usuário. Se for 1, dae retorna seus valores intermediários computados.

band
: um vetor de dois componentes que fornece a definição `[ml,mu]` da matriz de bandas computada por `jac` ;

  `r(i - j + ml + mu + 1,j)` = `dg(i)/dy(j)+cj*dg(i)/dydot(j)` . Se `jac` retorna uma matriz completa ajustado `band=[]`

maxstep
: um escalar ou matriz vazia, o tamanho máximo do passo. Uma matriz vazia indica ausência de limitiação.

stepin
: um escalar ou matriz vazia, o tamanho mínimo do passo. Uma matriz vazia indica "não especificado".

nonneg
: um escalar, deve ser ajustado para 0 se `a solução se conhece ser não negativa.` E caso contrário, deve ser ajustado para 1.

isest

A scalar, deve ser ajustado para 0 se a dada condição inicial é compatível: `g(t0,x0,xdot0)=0`. Deve ser ajustado para 1 se `xdot0` é apenas uma estimativa.

# Ver Também

dae

# Name

dasrt — Solucionador de equações diferenciais algébricas com cruzamento de zeros

```
[r,nn,[,hd]]=dasrt(x0,t0,t [,atol,[rtol]],res [,jac],ng, surf [,info] [,hd])
```

# Parâmetros

x0
: pode ser tanto `y0` (`ydot0` é estimado por `dassl` com a primeira estimativa sendo 0) ou a matriz `[y0 ydot0]`. `g(t,y0,ydot0)` deve ser igual a zero. Se você conhece apenas uma estimativa de `ydot0` faça `info(7)=1`

  y0
  : vetor coluna de reais de condições iniciais

  ydot0
  : vetor coluna de reais da derivada do tempo de `y` em `t0` (pode ser uma estimativa).

t0
: número real, é o instante inicial

t
: escalar real ou vetor. Fornece instantes para os quais você deseja uma solução. Note que você pode obter soluções para cada ponto de passo de dassl fazendo `info(2)=1`.

nn
: um vetor com duas entradas `[times num]` `times` é o valor do tempo no qual a superfície é cruzada, `num` é o número da superfície cruzada

atol,rtol
: escalares reais ou vetores colunas com o mesmo tamanho que `y`. `atol,rtol` fornecem respectivamente as tolerâncias de erros absolutos e relativos da solução. Se forem vetores, as tolerâncias são especificadas para cada elemento de `y`.

res
: função, lista ou string externos. Computa o valor de `g(t,y,ydot)`.Pode ser :

  - Uma função do Scilab

    Sua seqüência de chamamento pode ser `[r,ires]=res(t,y,ydot)` e `res` e deve retornar o resíduo `r=g(t,y,ydot)` e o indicador de erro `ires`. `ires = 0` se `res` obtiver sucesso ao computar `r`, `=-1` se o resíduo é indefinido localmente para `(t,y,ydot)`, `=-2` se os parâmetros estão fora do intervalo admissível.

  - Uma lista

    Esta forma permite passar outros parâmetros além de t,y,ydot à função. Deve ser como segue:

    ```
    list(res,x1,x2,...)
    ```

    Onde a seqüência de chamamento da função `res` é agora

    ```
    r=res(t,y,ydot,x1,x2,...)
    ```

    `res` ainda retorna `r=g(t,y,ydot)` como função de `(t,y,ydot,x1,x2,...)`.

    Warning: this form must not be used if there is no extra argument to pass to `the function.`

- Um string

  Deve se referir ao nome subrotina C ou Fortran ligada ao Scilab.

  Em C a seqüência de chamamento:

```
void res(double *t, double y[], double yd[], double r[],
         int *ires, double rpar[], int ipar[])
```

  Em Fortran deve ser:

```
subroutine res(t,y,yd,r,ires,rpar,ipar)
double precision t, y(*),yd(*),r(*),rpar(*)
integer ires,ipar(*)
```

  Os arrays rpar e ipar devem estar presentes, mas não podem ser utilizados.

jac

função, lista ou string externos. Computa o valor de `dg/dy+cj*dg/dydot` para um dado valor do parâmetro `cj`

- Uma função do Scilab

  Sua seqüência de chamamento deve ser `r=jac(t,y,ydot,cj)` e a função `jac` deve retornar `r=dg(t,y,ydot)/dy+cj*dg(t,y,ydot)/dydot` onde `cj` é um escalar real.

- Uma lista

  Deve ser como segue

```
list(jac,x1,x2,...)
```

  Onde a seqüência de chamamento da função `jac` é agora

```
r=jac(t,y,ydot,cj,x1,x2,...)
```

  `jac` ainda retorna `dg/dy+cj*dg/dydot` como uma função de `(t,y,ydot,cj,x1,x2,...)`.

- Um string

  Deve se referir ao nome de uma subrotina C ou Fortran ligada ao Scilab

  Em C, a seqüência de chamamento deve ser

```
void jac(double *t, double y[], double yd[],
         double pd[], double *cj, double rpar[], int ipar[])
```

  Em Fortran deve ser

```
subroutine jac(t,y,yd,pd,cj,rpar,ipar)
double precision t, y(*),yd(*),pd(*),cj,rpar(*)
```

```
integer ipar(*)
```

surf
: função, lista ou string externo. Computa o valor do vetor coluna `surf(t,y)` com componentes `ng`. Cada componente define uma superfície. Pode ser definido por:

- Uma função Scilab

  Sua seqüência de chamamento deve ser `surf(t,y)`

- Uma lista

  Deve ser como segue:

  ```
  list(surf,x1,x2,...)
  ```

  onde a seqüência de chamamento `surf` é agora

  ```
  r=surf(t,y,x1,x2,...)
  ```

- Um string

  Deve se referir ao nome de uma subrotina C ou Fortran ligada ao Scilab

  Em C a seqüência de chamamento deve ser:

  ```
  void surf(int *ny, double *t, double y[], int *ng, double gout[])
  ```

  Em Fortran deve ser

  ```
  subroutine surf(ny,t,y,ng,gout)
  double precision t, y(*),gout(*)
  integer ny,ng
  ```

info
: lista que contém `7` elementos, o valor padrão é list([],0,[],[],[],0,0)

info(1)
: escalar real que fornece o tempo máximo para o qual se permite avaliar `g`, ou uma matriz vazia `[]` se não houver limites impostos sobre o tempo.

info(2)
: indica se `dassl` retorna seus valores intermediários computados (`flag=1`) ou apenas os valores de pontos tempo especificados pelo usuário (`flag=0`).

info(3)
: vetor de dois componentes que fornece a definição `[ml,mu]` de matriz de bandas computada por `jac`; `r(i - j + ml + mu + 1,j) = "dg(i)/dy(j)+cj*dg(i)/dydot(j)"`. Se `jac` retorna uma matriz cheia, faça `info(3)=[]`.

info(4)
: escalar real que fornece o tamanho de passo máximo. Faça `info(4)=[]` se não houver limitação.

info(5)
: escalar real que fornece o passo inicial. Faça `info(5)=[]` se não for especificado.

info(6)
: faça `info(6)=1` caso se saiba que a soulução não é negativa, em outro caso, ajuste. `info(6)=0`.

info(7)
: faça `info(7)=1` se `ydot0` é apenas uma estimativa, `info(7)=0` se `g(t0,y0,ydot0)=0`.

hd
: vetor de reais que permite armazenar o contexto de `dassl` e retomar integração

r
: matriz de reais . Cada coluna é o vetor [t;x(t);xdot(t)] onde t é o índice do tempo para o qual a solução foi computada

# Descrição

Solução da equação diferencial implícita

```
g(t,y,ydot)=0
y(t0)=y0  e   ydot(t0)=ydot0
```

Retorna os instantes de cruzamento da superfície e o número da superfície alcançada em `nn`.

Exemplos detalhados podem ser encontrados em SCIDIR/tests/dassldasrt.tst

# Exemplos

```
//dy/dt = ((2*log(y)+8)/t -5)*y,  y(1) = 1,  1&lt;=t&lt;=6
//g1 = ((2*log(y)+8)/t - 5)*y
//g2 = log(y) - 2.2491
y0=1;t=2:6;t0=1;y0d=3;
atol=1.d-6;rtol=0;ng=2;

deff('[delta,ires]=res1(t,y,ydot)','ires=0;delta=ydot-((2*log(y)+8)/t-5)*y')
deff('[rts]=gr1(t,y)','rts=[((2*log(y)+8)/t-5)*y;log(y)-2.2491]')

[yy,nn]=dasrt([y0,y0d],t0,t,atol,rtol,res1,ng,gr1);
//(Deveria retornar nn=[2.4698972 2])
```

# Ver Também

ode, dassl, impl, fort, link, external

# Name

dassl — Equação diferencial algébrica

```
[r [,hd]]=dassl(x0,t0,t [,atol,[rtol]],res [,jac] [,info] [,hd])
```

# Parâmetros

x0

pode ser tanto `y0` (`ydot0` é estimado por `dassl` com a primeira estimativa sendo 0) ou a matriz `[y0 ydot0]`. `g(t,y0,ydot0)` deve ser igual a zero. Se você conhece apenas uma estimativa de `ydot0` faça `info(7)=1`

y0

vetor coluna de reais de condições iniciais

ydot0

vetor coluna de reais da derivada do tempo de `y` em `t0` (pode ser uma estimativa).

t0

número real, é o instante inicial

t

escalar real ou vetor. Fornece instantes para os quais você deseja uma solução. Note que você pode obter soluções para cada ponto de passo de dassl fazendo `info(2)=1`.

atol,rtol

escalares reais ou vetores colunas com o mesmo tamanho que `y`. `atol,rtol` fornecem respectivamente as tolerâncias de erros absolutos e relativos da solução. Se forem vetores, as tolerâncias são especificadas para cada elemento de `y`.

res

função, lista ou string externos. Computa o valor de `g(t,y,ydot)`. Pode ser :

- Uma função do Scilab

  Sua seqüência de chamamento pode ser `[r,ires]=res(t,y,ydot)` e `res` deve retornar o resíduo `r=g(t,y,ydot)` e o indicador de erro `ires`. `ires = 0` se `res` obtiver sucesso ao computar `r`, `=-1` se o resíduo é indefinido localmente para `(t,y,ydot)`, `=-2` se os parâmetros estão fora do intervalo admissível.

- Uma lista

  Esta forma permite passar outros parâmetros além de t,y,ydot à função. Deve ser como segue:

  ```
  list(res,x1,x2,...)
  ```

  Onde a seqüência de chamamento da função `res` é agora

  ```
  r=res(t,y,ydot,x1,x2,...)
  ```

  `res` ainda retorna `r=g(t,y,ydot)` como função de `(t,y,ydot,x1,x2,...)`.

- Um string

  Deve se referir ao nome subrotina C ou Fortran ligada ao Scilab.

Em C a seqüência de chamamento:

```
void res(double *t, double y[], double yd[], double r[],
         int *ires, double rpar[], int ipar[])
```

Em Fortran deve ser:

```
subroutine res(t,y,yd,r,ires,rpar,ipar)
double precision t, y(*),yd(*),r(*),rpar(*)
integer ires,ipar(*)
```

Os arrays rpar e ipar devem estar presentes, mas não podem ser utilizados.

jac

função, lista ou string externos. Computa o valor de `dg/dy+cj*dg/dydot` para um dado valor do parâmetro `cj`

- Uma função do Scilab

  Sua seqüência de chamamento deve ser `r=jac(t,y,ydot,cj)` e a função `jac` deve retornar `r=dg(t,y,ydot)/dy+cj*dg(t,y,ydot)/dydot` onde `cj` é um escalar real.

- Uma lista

  Deve ser como segue

  ```
  list(jac,x1,x2,...)
  ```

  Onde a seqüência de chamamento da função `jac` é agora

  ```
  r=jac(t,y,ydot,cj,x1,x2,...)
  ```

  `jac` ainda retorna `dg/dy+cj*dg/dydot` como uma função de `(t,y,ydot,cj,x1,x2,...)`.

- Um string

  Deve se referir ao nome de uma subrotina C ou Fortran ligada ao Scilab

  Em C, a seqüência de chamamento deve ser

  ```
  void jac(double *t, double y[], double yd[],
           double pd[], double *cj, double rpar[], int ipar[])
  ```

  Em Fortran deve ser

  ```
  subroutine jac(t,y,yd,pd,cj,rpar,ipar)
  double precision t, y(*),yd(*),pd(*),cj,rpar(*)
  integer ipar(*)
  ```

info
: lista opcional que contém `7` elementos. O valor padrão é list([],0,[],[],[],0,0);

info(1)
: escalar real que fornece o tempo máximo para o qual se permite avaliar `g` , ou uma matriz vazia `[]` se não houver limites impostos sobre o tempo.

info(2)
: indica se `dassl` retorna seus valores intermediários computados (`flag=1`) ou apenas os valores de pontos tempo especificados pelo usuário (`flag=0`).

info(3)
: vetor de dois componentes que fornece a definição `[ml,mu]` de matriz de bandas computada por `jac`; `r(i - j + ml + mu + 1,j) = "dg(i)/dy(j)+cj*dg(i)/dydot(j)"`. Se `jac` retorna uma matriz cheia, faça `info(3)=[]`.

info(4)
: escalar real que fornece o tamanho de passo máximo. Faça `info(4)=[]` se não houver limitação.

info(5)
: escalar real que fornece o passo inicial. Faça `info(5)=[]` se não for especificado.

info(6)
: faça `info(6)=1` caso se saiba que a soulução não é negativa, em outro caso, ajuste `info(6)=0`.

info(7)
: faça `info(7)=1` se `ydot0` é apenas uma estimativa, `info(7)=0` se `g(t0,y0,ydot0)=0`.

hd
: vetor de reais que permite armazenar o contexto de `dassl` e retomar integração

r
: matriz de reais . Cada coluna é o vetor [t;x(t);xdot(t)] onde t é o índice do tempo para o qual a solução foi computada

# Descrição

A função dassl integra a equação alegébrica-diferencial e retorna a evolução de `y` em pontos de tempo dados.

```
g(t,y,ydot)=0
y(t0)=y0  e   ydot(t0)=ydot0
```

# Exemplos

```
function [r,ires]=chemres(t,y,yd)
  r=[-0.04*y(1)+1d4*y(2)*y(3)-yd(1)
      0.04*y(1)-1d4*y(2)*y(3)-3d7*y(2)*y(2)-yd(2)
      y(1)+y(2)+y(3)-1];
  ires=0
endfunction
function pd=chemjac(x,y,yd,cj)
  pd=[-0.04-cj , 1d4*y(3)                , 1d4*y(2);
       0.04    ,-1d4*y(3)-2*3d7*y(2)-cj ,-1d4*y(2);
       1       , 1                      , 1       ]
```

```
endfunction

y0=[1;0;0];
yd0=[-0.04;0.04;0];
t=[1.d-5:0.02:.4,0.41:.1:4,40,400,4000,40000,4d5,4d6,4d7,4d8,4d9,4d10];

y=dassl([y0,yd0],0,t,chemres);

info=list([],0,[],[],[],0,0);
info(2)=1;
y=dassl([y0,yd0],0,4d10,chemres,info);
y=dassl([y0,yd0],0,4d10,chemres,chemjac,info);

//utilizando um argumento extra para parâmetros
//-----------------------------------
function [r,ires]=chemres(t,y,yd ,a,b,c)
   r=[-a*y(1)+b*y(2)*y(3)-yd(1)
       a*y(1)-b*y(2)*y(3)-c*y(2)*y(2)-yd(2)
       y(1)+y(2)+y(3)-1];
   ires=0
endfunction
function pd=chemjac(x,y,yd,cj, a,b,c)
    pd=[-a-cj , b*y(3)             , b*y(2);
         a    ,-b*y(3)-2*c*y(2)-cj ,-b*y(2);
         1    , 1                  , 1       ]
endfunction
y=dassl([y0,yd0],0,t,list(chemres,0.04,1d4,3d7),list(chemjac,0.04,1d4,3d7));

//utilizando código c
//------------
// - criando o código C
rescode=['void chemres(double *t, double y[], double yd[], double r[], int *ire
         ' {'
         '   r[0] = -0.04*y[0]+1.0e4*y[1]*y[2]                -yd[0];'
         '   r[1] =  0.04*y[0]-1.0e4*y[1]*y[2]-3.0e7*y[1]*y[1]-yd[1];'
         '   r[2] =       y[0]+y[1]+y[2]-1;'
         '   *ires = 0;'
         ' }'];

jaccode=['void chemjac(double *t, double y[], double yd[], double pd[], double
         ' {'
         '   /* primeira coluna*/'
         '   pd[0] = -0.04-*cj;'
         '   pd[1] =  0.04;'
         '   pd[2] =  1.0;'
         '    /* segunda coluna*/'
         '   pd[3] =  1.0e4*y[2];'
         '   pd[4] = -1.0e4*y[2]-2*3.0e7*y[1]-*cj;'
         '   pd[5] =  1.0;'
         '    /* terceira coluna*/'
         '   pd[6] =  1.0e4*y[1];'
         '   pd[7] = -1.0e4*y[1];'
         '   pd[8] =  1.0;'
         ' }'];
mputl([rescode;jaccode],TMPDIR+'/mycode.c') //create the C file

// - compilando-o
ilib_for_link(['chemres','chemjac'],'mycode.c',[],'c',TMPDIR+'/Makefile',TMPDIR
```

```
// - ligando-o com o Scilab
exec(TMPDIR+'/loader.sce') //linking incremental

// - chamada a dassl
y=dassl([y0,yd0],0,t,'chemres','chemjac');
```

# Ver Também

ode, dasrt, impl, fort, link, external

# Name

feval — múltipla avaliação

```
[z]=feval(x,y,f)
[z]=feval(x,f)
```

# Parâmetros

x,y
: dois vetores

f
: função ou string (para chamada FORTRAN ou C)

# Descrição

Múltipla avaliação de uma função para um ou dois argumentos do tipo vetor:

z=feval(x,f)
: retorna o vetor `z` definido por `z(i)=f(x(i))`

z=feval(x,y,f)
: retorna a matriz `z` tal que `z(i,j)=f(x(i),y(j))`

`f` é uma external (função ou rotina) que aceita dois ou mais argumentos supostos reais. O resultado retornado por `f` pode ser real ou complexo. No caso de uma chamada FORTRAN, a função `'f'` deve ser definida na subrotina `ffeval.f` (no diretório `SCIDIR/routines/default`)

# Exemplos

```
deff('[z]=f(x,y)','z=x^2+y^2');
feval(1:10,1:5,f)
deff('[z]=f(x,y)','z=x+%i*y');
feval(1:10,1:5,f)
feval(1:10,1:5,'parab')   //See ffeval.f file
feval(1:10,'parab')
// Para link dinâmico (ver exemplo ftest em ffeval.f)
// você pode usar o comando link (os parâmetros dependem da máquina):
// unix('make ftest.o');link('ftest.f','ftest); feval(1:10,1:5,'ftest')
```

# Ver Também

evstr, horner, execstr, external, link

# Name

impl — equações diferenciais algébricas

```
y=impl([type],y0,ydot0,t0,t [,atol, [rtol]],res,adda [,jac])
```

# Parâmetros

y0,ydot0
: vetor de reais ou matriz (condições iniciais)

t0
: escalar real (tempo inicial).

t
: vetor de reais (tempos nos quais a solução é computada).

res,adda
: funções externas (função, strting ou lista).

type
: string `'adams'` ou `'stiff'`

atol,rtol
: escalar real ou vetor de reais com as mesmas dimensões que `y`.

jac
: função externa (função, string ou lista).

# Descrição

Solução da equação diferencial linear implicita

A(t,y) dy/dt=g(t,y), y(t0)=y0

`t0` é o instante inicial, `y0` é o vetor de condições iniciais. O vetor `ydot0` da derivada em relação ao tempo `y` em `t0` também deve ser dado. A entrada `res` é uma função externa i.e. uma função com sintaxe especificada, ou o nome da subrotina FORTRAN ou da função C (string) com seqüência de chamamento especificada, ou uma lista.

Se `res` for uma função, a sintaxe deve ser como segue:

```
r = res(t,y,ydot)
```

onde `t` é um escalar real (tempo) e `y` e `ydot` são vetores de reais (estado e derivada do estado). Esta função deve retornar `r=g(t,y)-A(t,y)*ydot`.

Se `res` for um string, ele se refere a uma subrotina FORTRAN ou uma função C. Ver `SCIDIR/routines/default/Ex-impl.f` para um exemplo.

`res` também pode ser uma lista, ver a ajuda de `ode`.

A entrada `adda` também é uma função externa

Se `adda` for uma função, a sintaxe deve ser como segue:

```
r = adda(t,y,p)
```

e deve retornar `r=A(t,y)+p` onde `p` é uma matriz a ser adicionada a `A(t,y)`.

Se `adda` for um string, ele se refere a uma subrotina FORTRAN ou uma função C. Ver `SCIDIR/routines/default/Ex-impl.f` para um exemplo.

`adda` também pode ser uma lista, ver a ajuda de `ode`.

A entrada `jac` também é uma função externa

Se `jac` for uma função, a sintaxe deve ser como segue:

```
j = jac(t,y,ydot)
```

e deve retornar o Jacobiano de `r=g(t,y)-A(t,y)*ydot` em relação a `y`.

Se `jac` for um string, ele se refere a uma subrotina FORTRAN ou uma função C. Ver `SCIDIR/routines/default/Ex-impl.f` para um exemplo.

`jac` também pode ser uma lista, ver a ajuda de `ode`.

# Exemplos

```
y=impl([1;0;0],[-0.04;0.04;0],0,0.4,'resid','aplusp');
// utilizando reinicialização rápida
//[x1,w,iw]=impl([1;0;0],[-0.04;0.04;0],0,0.2,'resid','aplusp');
// inicialização rápida da chamada anterior
//[x1]=impl([1;0;0],[-0.04;0.04;0],0.2,0.4,'resid','aplusp',w,iw);
//maxi(abs(x1-x))
```

# Ver Também

dassl, ode, external

# Name

int2d — integral definida 2d por quadratura e cubatura

```
[I,err]=int2d(X,Y,f [,params])
```

# Parâmetros

X
: um array 3 por N contendo as abscissas dos vertices dos N triângulos.

Y
: um array 3 por N contendo as ordenadas dos vertices dos N triângulos.

f
: função externa (função, string ou lista) definindo o integrando `f(u,v)`;

params
: vetor de reais `[tol, iclose, maxtri, mevals, iflag]`. O valor padrão é `[1.d-10, 1, 50, 4000, 1]`.

  tol
  : o limite desejado do erro. Se `iflag=0`, `tol` é interpretado como um limite de erro relativo; se `iflag=`1, o limite é de erro absoluto.

  iclose
  : um inteiro que determina a seleção dos métodos LQM0 ou LQM. Se `iclose=1` , então LQM1 é utilizado. Qualquer outro valor de `iclose` faz com que LQM0 seja usado. LQM0 utiliza valores da função apenas em pontos interiores ao triângulo. LQM1 geralmente é mais preciso que LQM0 mas envolve a avaliação do integrando em mais pontos, incluindo em alguns pontos da fronteira do triângulo. Geralmente é melhor utilizar LQM1 a não ser que o integrando possuia singularidades nas bordas do triângulo.

  maxtri
  : o número máximo de triângulos na triangularização final da região

  mevals
  : o número máximo de avaliações da função permitido. Este número terá efeito na limitação da computação se for menor que 94*`maxtri` quando LQM1 é especificado ou 56*`maxtri` quando LQM0 é especificado.

  iflag

I
: o valor da integral

err
: o erro estimado

# Descrição

`int2d` computa a integral bidimensional de uma função `f` sobre uma região que consiste de `n` triângulos. Um estimativa de erro total é obtida e comparada a - `tol` - que é fornecida como entrada para a subrotina. A tolerância de erro é tratada como relativa ou absoluta dependendo do valor de entrada de `iflag`. Um "módulo de quadratura local" ("Local Quadrature Module") é aplicado para cada triângulo de entrada e estimativas da integral total e do erro total são computadas. O módulo de quadratura local é a subrotina LQM0 ou a subrotina LQM1 e a escolha entre elas é determinada pelo valor da variável `iclose`.

Se a estimativa de erro total excede a tolerância, o triângulo com maior erro absoluto é dividio em dois outro triângulos traçando-se uma mediana por seu maior lado. O módulo de quadratura local é então aplicado a cada um dos subtriângulos para se obter novas estimativas da integral e do erro. Este processo é repetido até que um dos seguintes (1) a tolerância é satisfeita, (2) o número de triângulos gerados excede o parâmetro `maxtri`, (3) o número de avaliações do integrando excede o parâmetro `mevals`, ou (4) a função sente que um erro de arredondamento está começando a contaminar o resultado.

# Exemplos

```
X=[0,0;1,1;1,0];
Y=[0,0;0,1;1,1];
deff('z=f(x,y)','z=cos(x+y)')
[I,e]=int2d(X,Y,f)
// computa o integrando sobre o quadrado [0 1]x[0 1]
```

# Ver Também

intc, intl, int3d, intg, mesh

# Autores

Autores da rotina FORTRAN twodq: Kahaner,D.K.,N.B.S., Rechard,O.W.,N.B.S.,; Barnhill,Robert,Univ. de UTAH

# Name

int3d — integral definida 3d pelo método da quadratura e cubatura

```
[result,err]=int3d(X,Y,Z,f [,nf[,params]])
```

# Parâmetros

X
: um array 4 por `NUMTET` contendo as abscissas dos vértices dos `NUMTET` tetraedros.

Y
: um array 4 por `NUMTET` contendo as ordenadas dos vértices dos `NUMTET` tetraedros.

Z
: um array 4 por `NUMTET` contendo as terceiras coordenadas dos vértices dos `NUMTET` tetraedros.

f
: função externa (function, string ou lista) definindo o integrando `f(xyz,nf)`, onde `xyz` é o vetor das coordenadas de um ponto e nf os números das funções

nf
: o número da função a ser integrada (o padrão é 1)

params
: vetor de reais `[minpts, maxpts, epsabs, epsrel]`. O valor padrão é `[0, 1000, 0.0, 1.d-5]`.

  epsabs
  : limite do erro absoluto desejado

  epsrel
  : limite do erro relativo desejado

  minpts
  : número mínimo de avaliações da função

  maxpts
  : número máximo de avaliações da função. O número de avaliações da função sobre cada subregião é 43

result
: o valor da integral, ou vetor de valores da integral.

err
: estimativas do erro absoluto

# Descrição

A função calcula uma aproximação a um dado vetor de integrais definidas

```
I  I  I (F ,F ,...,F )      dx(3)dx(2)dx(1),
          1  2      numfun
```

onde a região de integração são os NUMTET tetraedros e onde

```
F = F (X(1),X(2),X(3)), J = 1,2,...,NUMFUN.
 J   J
```

uma estratégia globalmente adaptativa é aplicada para se computar aproximações `result(k)` esperando-se que satisfaça, para cada componente de I, à seguinte precisão: `ABS(I(K)-RESULT(K))<=MAX(EPSABS,EPSREL*ABS(I(K)))`

`int3d` repetidamente subdivide os tetraedros com maiores erros estimados e estima as integrais e os erros sobre os novos tetraedros até que a exigência de erro seja encontrada ou `MAXPTS` avaliações da função tenham sido feitas.

Uma regra de 43 pontos de integração com todos os pontos de avaliação dentro dos tetraedros é aplicada. A regra tem grau polinomial 8.

Se os valores dos parâmetros de entrada `EPSABS` ou `EPSREL` são suficientemente grandes, uma regra de integração é aplicada sobre cada tetraedro e os valores são aproximados para se fornecer as aproximações `RESULT(K)`. Nenhuma subdivisão posterior dos tetraedros será feita.

Quando `int3d` computa estimativas a um vetor de integrais, é dado tratamento igual a todos os componentes do vetor. Isto é, `I(Fj)` e `I(Fk)` para `j` diferente de `k`, são estimados com a mesma subdivisão da região de integração. Para integrais suficientemente semelhantes, podemos economizar tempo aplicando `int3d` a todos os integrando em uma chamada. Para integrais que variam continuamente em função de um parâmetro, as estimativas produzidas por `int3d` também irão variar continuamente quando a subdivisão é aplicada a todos os componentes. Este geralmente não será o caso quando componentes diferentes são tratados separadamente.

Por outro lado, este recurso deve ser utilizado com cautela quando os componentes diferentes da integral requerem claramente subdivisões diferentes.

# Referências

Rotina FORTRAN dcutet.f

# Exemplos

```
X=[0;1;0;0];
Y=[0;0;1;0];
Z=[0;0;0;1];
[RESULT,ERROR]=int3d(X,Y,Z,'int3dex')
// computa o integrando exp(x*x+y*y+z*z) sobre o
//tetraedro (0.,0.,0.),(1.,0.,0.),(0.,1.,0.),(0.,0.,1.)

//integração sobre um cubo  -1&lt;=x&lt;=1;-1&lt;=y&lt;=1;-1&lt;=z&lt;=1

//  fundo  -topo-    direita -esquerda- frente  -traseira-
X=[ 0, 0,    0, 0,    0, 0,    0, 0,    0, 0,    0, 0;
   -1,-1,   -1,-1,    1, 1,   -1,-1,   -1,-1,   -1,-1;
    1,-1,    1,-1,    1, 1,   -1,-1,    1,-1,    1,-1;
    1, 1,    1, 1,    1, 1,   -1,-1,    1, 1,    1, 1];
Y=[ 0, 0,    0, 0,    0, 0,    0, 0,    0, 0,    0, 0;
   -1,-1,   -1,-1,   -1, 1,   -1, 1,   -1,-1,    1, 1;
   -1, 1,   -1, 1,    1, 1,    1, 1,   -1,-1,    1, 1;
    1, 1,    1, 1,   -1,-1,   -1,-1,   -1,-1,    1, 1];
Z=[ 0, 0,    0, 0,    0, 0,    0, 0,    0, 0,    0, 0;
   -1,-1,    1, 1,   -1, 1,   -1, 1,   -1,-1,   -1,-1;
   -1,-1,    1, 1,   -1,-1,   -1,-1,   -1, 1,   -1, 1;
```

```
   -1,-1,    1, 1,    1, 1,    1, 1,    1, 1,    1, 1];

function v=f(xyz,numfun),v=exp(xyz'*xyz),endfunction
[result,err]=int3d(X,Y,Z,f,1,[0,100000,1.d-5,1.d-7])

function v=f(xyz,numfun),v=1,endfunction
[result,err]=int3d(X,Y,Z,f,1,[0,100000,1.d-5,1.d-7])
```

# Ver Também

intc, intl, int2d

# Autores

Jarle Berntsen

The Computing Centre, University of Bergen, Thormohlens gt. 55, N-5008 Bergen, Noruega Fone.. 47-5-544055 Email.. jarle@eik.ii.uib.no,

Ronald Cools

Dept. of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3030 Heverlee, Belgium Fone.. 32-16-201015 (3562) Email.. ronald@cs.kuleuven.ac.be,

Terje O. Espelid

Department of Informatics, University of Bergen, Thormohlens gt. 55, N-5008 Bergen, Norway Fone.. 47-5-544180 Email.. terje@eik.ii.uib.no

# Name

intc — integral de Cauchy

```
[y]=intc(a,b,f)
```

# Parâmetros

a,b
: dois números complexos

f
: função externa

# Descrição

Se `f` é uma função com valores complexos, `intc(a,b,f)` computa o valor da integral de `a` a `b` de `f(z)dz` ao longo da reta `a b` do plano complexo.

# Ver Também

intg, intl

# Autor

F. D.

# Name

integrate — integração pela quadratura

```
x=integrate(expr,v,x0,x1 [,atol [,rtol]])
```

# Parâmetros

expr
: string definindo uma expressão Scilab

v
: string (variável da integração)

x0
: número real (limite inferior de integração)

x1
: vetor de números reais, limites superiores de integração

atol
: número real (erro absoluto). Valor padrão: 1.-8

rtol
: número real (erro relativo). Valor padrão: 1e-14

x
: vetor de números reais, o valor da integral para cada `x1(i)`.

# Descrição

Computa : $x(i) = \int_{x_0}^{x_1(i)} f(v)\,dv$ para `i=1:size(x1,'*')`

Onde $f(v)$ é dado pela expressão `expr`.

A avaliação procura satisfazer à seguinte precisão: `abs(I-x)<= max(atol,rtol*abs(I))` onde I é o valor exato da integral.

# Restrição

A expressão dada não deve utilizar nomes de variáveis começando por %.

# Exemplos

```
x0=0;x1=0:0.1:2*%pi;
X=integrate('sin(x)','x',x0,x1);
norm(cos(x1)-(1-X))

x1=-10:0.1:10;
X=integrate(['if x==0 then 1,';
  'else sin(x)/x,end'],'x',0,x1)
```

# Ver Também

intg, inttrap, intsplin, ode

# Name

intg — integral definida

```
[v,err]=intg(a,b,f [,ea [,er])
```

# Parâmetros

a,b
: números reais

f
: função externa (função, lista ou string)

ea, er
: números reais

ea
: erro absoluto requerido no resultado. Valor padrão: 1.d-14

er
: erro relativo requerido no resultado. Valor padrão: 1.d-8

err
: erro absoluto estimado no resultado

# Descrição

`intg(a,b,f)` avalia a integral definida de `a` a `b` de `f(t)dt`. A função `f(t)` deve ser contínua.

Espera-se que a avaliação satisfaça à seguinte precisão: `abs(I-v)<= max(ea,er*abs(I))` onde I é o valor exato da integral.

`f` é uma função externa :

Se `f` é uma função, sua definição deve ser como segue: `y = f(t)`

Se `f` é uma lista, a lista deve ser como segue: `list(f,x1,x2,...)` onde `f` é uma função com seqüência de chamamento `f(t,x1,x2,...)`.

Se `f` é um string, ele se refere ao nome de uma função FORTRAN ou procedure C com dada seqüência de chamamento:

No caso FORTRAN, a seqüência de chamamento deve ser `double precision function f(x)` onde `x` tabém é um número de dupla precisão.

No caso C, a seqüência de chamamento é `double f(double *x)`.

# Exemplos

```
//caso de função Scilab
function y=f(x),y=x*sin(30*x)/sqrt(1-((x/(2*%pi))^2)),endfunction
exact=-2.5432596188;
I=intg(0,2*%pi,f)
abs(exact-I)

//caso de função Scilab com parâmetros
function y=f1(x,w),y=x*sin(w*x)/sqrt(1-((x/(2*%pi))^2)),endfunction
```

```
I=intg(0,2*%pi,list(f1,30))
abs(exact-I)

// caso de código FORTRAN (compilador FORTRAN requerido)
// escreve o código FORTRAN
F=['      double precision function ffun(x)'
   '      double precision x,pi'
   '      pi=3.14159265358979312d+0'
   '      ffun=x*sin(30.0d+0*x)/sqrt(1.0d+0-(x/(2.0d+0*pi))**2)'
   '      return'
   '      end'];
mputl(F,TMPDIR+'/ffun.f')

// compile o código FORTRAN
l=ilib_for_link('ffun','ffun.f',[],'f',TMPDIR+'/Makefile');

// linking incremental
link(l,'ffun','f')

//integrando a função
I=intg(0,2*%pi,'ffun')
abs(exact-I)

// caso de código C (compilador C requerido)
// escreva o código C
C=['#include <math.h>'
   'double cfun(double *x)'
   '{'
   '  double y,pi=3.14159265358979312;'
   '  y=*x/(2.0e0*pi);'
   '  return *x*sin(30.0e0**x)/sqrt(1.0e0-y*y);'
   '}'];
mputl(C,TMPDIR+'/cfun.c')

// compile o código C
l=ilib_for_link('cfun','cfun.c',[],'c',TMPDIR+'/Makefile');

// linking incremental
link(l,'cfun','c')

//integrando a função
I=intg(0,2*%pi,'cfun')
abs(exact-I)
```

# Ver Também

intc, intl, inttrap, intsplin, ode

# Funções Utilizadas

As rotinas associadas podem ser encontradas no diretório routines/integ :

dqag0.f e dqags.f de quadpack

# Name

intl — integral de Cauchy

```
[y]=intl(a,b,z0,r,f)
```

# Parâmetros

z0
: número complexo

a,b
: dois número reais

r
: número real positivo

f
: função externa

# Descrição

Se `f` é uma função externa com valores complexos, `intl(a,b,z0,r,f)` computa a integral de `f(z)dz` ao longo da curva no plano complexo definida por `z0 + r.exp(%i*t)` para `a<=t<=b` (parte do círculo com centro `z0` e raio `r` com fase entre `a` e `b`).

# Ver Também

intc

# Autor

F. D.

# Name

ode — Solucionador de equações diferenciais ordinárias

```
y=ode(y0,t0,t,f)
[y,w,iw]=ode([type],y0,t0,t [,rtol [,atol]],f [,jac] [,w,iw])
[y,rd,w,iw]=ode("root",y0,t0,t [,rtol [,atol]],f [,jac],ng,g [,w,iw])
y=ode("discrete",y0,k0,kvect,f)
```

# Parâmetros

y0
: matriz ou vetor de reais (condições iniciais).

t0
: escalar real (tempo inicial).

t
: vetor de reais (tempos nos quais a solução é computada).

f
: função externa (função, lista ou string).

type
: um dos strings seguintes: `"adams"` `"stiff"` `"rk"` `"rkf"` `"fix"` `"discrete"` `"roots"`

rtol,atol
: constantes reais ou vetores com o mesmo tamanho que `y`.

jac
: função externa (função, lista ou string).

w,iw
: vetores de reais

ng
: inteiro

g
: função externa (função, lista ou string).

k0
: inteiro (tempo inicial). kvect : vetor de inteiros

# Descrição

`ode` é a função padrão para se resolver sistemas de EDO explícitos definidos por: dy/dt=f(t,y) , y(t0)=y0. É uma interface entre vários solucionadores, em particular a ODEPACK. O tipo de problema resolvido e o método a ser utilizado dependem do primeiro argumento opcional `type` que pode ser um dos strings seguintes:

<not given>:
: O `lsoda`, solucionador do pacote ODEPACK é chamado por padrão. Ele escolhe automaticamente entre o método preditor-corretor não-rígido de Adams e a Fórmula de Diferenciação Retroativa (FDR) rígida. Ele utiliza o método não rígido inicialmente e monitora os dados para decidir qual método utilizar.

"adams":
Este é para problemas não rígidos. O solucionador `lsode` do pacote ODEPACK é chamado e utiliza o método de Adams.

"stiff":
Este é para problemas rígidos. O solucionador `lsode` do pacote ODEPACK é chamado e é utilizado o método FDR.

"rk":
Método adaptativo de Runge-Kutta de ordem 4 (RK4).

"rkf":
O programa de Shampine e Watts baseado no par Runge-Kutta de Fehlberg de ordem 4 e 5 (RKF45) é utilizado. Este é utilizado para problemas não-rígidos e mediamente rígidos quando as computações de derivação não são custosas. Este método não é recomendado ser utilizado quando o usuário requer uma maior precisão.

"fix":
Mesmo solucionador que "rkf", mas a interface do usuário é bem simples, i.e. apenas os parâmetros `rtol` e `atol` podem ser passados ao solucionador. Este é o método mais simples a se tentar.

"root":
Solucionador de EDOs com capacidade de encontrar raízes. O solucionador `lsodar` do pacote ODEPACK é utilizado. É uma variante do solucionador `lsoda` onde se acha raízes de uma dada função vetorial. Ver ajuda em ode_root para mais detalhes.

"discrete":
Simulação de tempo discreto. Ver ajuda em ode_discrete para mais detalhes.

Nesta ajuda podemos apenas descrever o uso de `ode` para sistemas padrões explícitos EDOs .

- A chamada mais simples a `ode` é: `y=ode(y0,t0,t,f)` onde `y0` é o vetor de condições iniciais, `t0` é o tempo inicial, `t` é o vetor de tempos onde a solução `y` é computada e `y` é a matriz de vetores soluções `y=[y(t(1)),y(t(2)),...]`.

  O argumento de entrada `f` define o lado direito da equação diferencial de primeira ordem: dy/dt=f(t,y). É uma função externa, isto é, uma função com sintaxe especificada, ou o nome de uma subrotina Fortran ou uma subfunção C (string) com seqüência de chamamento especificada, ou uma lista:

  - Se `f` for uma função do Scilab, a sua sintaxe deve ser `ydot = f(t,y)`, onde `t` é um escalar real (tempo) e `y` é um vetor de reais (estado). `ydot` é um vetor de reais (dy/dt)

  - Se `f` é um string ele se refere ao nome de uma subrotina Fortran ou uma subfunção C, i.e. Se `ode(y0,t0,t,"fex")` for o comando, então a subrotina `fex` será chamada.

    A rotina Fortran deve ter a seguinte seqüência de chamamento: `fex(n,t,y,ydot)`, com n um inteiro, t um escalar de dupla precisão, y e ydot vetores de dupla precisão.

    A função C deve ter o seguinte protótipo: `fex(int *n,double *t,double *y,double *ydot)`

    `t` é o tempo, `y` o estado e `ydot` a derivada do estado (dy/dt)

    Esta função externa pode ser construída em um SO de modo indpendente através de ilib_for_link e ligada dinamicamente através da função do Scilab link.

  - O argumento `f` também pode ser uma lista com a seguinte estrutura: `lst=list(realf,u1,u2,...un)` onde `realf` é uma função do Scilab com a sintaxe: `ydot = f(t,y,u1,u2,...,un)`

Esta sintaxe permite utilizar parâmetros como argumentos de `realf`.

A função `f` pode retornar uma matriz `p x q` ao invés de um vetor. Com esta notação de matriz, nos resolvemos o sistema `n=p+q` de EDOs `dY/dt=F(t,Y)` onde `Y` é uma matriz `p x q` matrix. Então, as condições iniciais, `Y0`, também devem ser uma matriz `p x q` e o resultado de `ode` é a matriz `p x q(T+1)` `[Y(t_0),Y(t_1),...,Y(t_T)]`.

- Argumentos de entrada opcionais podem ser fornecidos para o erro da solução: `rtol` e `atol` são limiares para os erros relativos e absolutos estimados. O erro estimado em `y(i)` é: `rtol(i)*abs(y(i))+atol(i)`

  e uma intergração é feita enquanto este erro é pequeno para todos os componentes do estado. Se `rtol` e/ou `atol` for uma constante `rtol(i)` e/ou `atol(i)` são ajustados para esta constante. Os valores padrões para `rtol` e `atol` são respectivamente `rtol=1.d-5` e `atol=1.d-7` para a maior parte dos solucionadores `rtol=1.d-3` e `atol=1.d-4` para `"rfk"` e `"fix"`.

- Para problemas rígidos, é melhor fornecer o Jacobiano da função do lado direito da equação como o argumento opcional `jac`. É uma função externa, istoé i.e. uma função com sintaxe especificada, ou o nome de uma subrotina Fortran ou uma subfunção C (string) com seqüência de chamamento especificada, ou uma lista.

  Se `jac` for uma função, a sintaxe deve ser `J=jac(t,y)`

  onde `t` é um escalar real (tempo), `y` é um vetor de reais (estado). A matriz resultante `J` deve fornecer df/dx i.e. `J(k,i) = dfk/dxi` com `fk` = k-ésimo componente de f.

  Se `jac` for um sting, ele se refere a uma subrotina Fortran, ou uma subfunção C, com as seguinte seqüência de chamamento:

  No caso Fortran:

  ```
  subroutine fex(n,t,y,ml,mu,J,nrpd)
  integer n,ml,mu,nrpd
  double precision t,y(*),J(*)
  ```

  No caso C:

  ```
  void fex(int *n,double *t,double *y,int *ml,int *mu,double *J,int *nrpd,)
  ```

  `jac(n,t,y,ml,mu,J,nrpd)`. Na maior parte dos casos, você não tem que se referir a `ml`, `mu` e `nrpd`.

  Se `jac` for uma lista, as mesmas convenções que para `f` se aplicam.

- Os argumentos opcionais `w` e `iw` e vetores para armazenamento de informações podem ser retornados pela rotina de integração (ver ode_optional_output para detalhes). Queando esses vetores são fornecidos no lado direito de `ode`, a integração reinicia com os mesmos parâmetros da parada anteiror.

- Mais opções podem ser fornecidas a solucionadores ODEPACK utilizando-se a variável `%ODEOPTIONS`. Ver odeoptions.

# Exemplos

```

```

```
// ----------  EDO simples de uma dimensão (função externa do Scilab)
// dy/dt=y^2-y sin(t)+cos(t), y(0)=0
function ydot=f(t,y),ydot=y^2-y*sin(t)+cos(t),endfunction
y0=0;t0=0;t=0:0.1:%pi;
y=ode(y0,t0,t,f)
plot(t,y)

// ---------- EDO simples de uma dimensão (função externa codificada em C)
ccode=['#include <math.h>'
       'void myode(int *n,double *t,double *y,double *ydot)'
       '{'
       '  ydot[0]=y[0]*y[0]-y[0]*sin(*t)+cos(*t);'
       '}']
mputl(ccode,TMPDIR+'/myode.c') //criando o arquivo C
ilib_for_link('myode','myode.c',[],'c',TMPDIR+'/Makefile',TMPDIR+'/loader.sce')
exec(TMPDIR+'/loader.sce') //linking incremental
y0=0;t0=0;t=0:0.1:%pi;
y=ode(y0,t0,t,'myode');

// ---------- Simulação de dx/dt = A x(t) + B u(t) com u(t)=sin(omega*t),
// x0=[1;0]
// solução x(t) desejada em t=0.1, 0.2, 0.5 ,1.
// A e a função u são passados para a função do lado direito em uma lista.
// B e omega são passados como variáveis globais
function xdot=linear(t,x,A,u),xdot=A*x+B*u(t),endfunction
function ut=u(t),ut=sin(omega*t),endfunction
A=[1 1;0 2];B=[1;1];omega=5;
ode([1;0],0,[0.1,0.2,0.5,1],list(linear,A,u))

// ---------- Integração com notação matricial da equação diferencial de Ricatti
// Xdot=A'*X + X*A - X'*B*X + C , X(0)=Identity
// Solução em t=[1,2]
function Xdot=ric(t,X),Xdot=A'*X+X*A-X'*B*X+C,endfunction
A=[1,1;0,2]; B=[1,0;0,1]; C=[1,0;0,1];
t0=0;t=0:0.1:%pi;
X=ode(eye(A),0,t,ric)

// ---------- Notação matricial, computação de exp(A)
A=[1,1;0,2];
function xdot=f(t,x),xdot=A*x;,endfunction
ode(eye(A),0,1,f)
ode("adams",eye(A),0,1,f)

// ---------- Notação de matriz, computação de exp(A) com matriz rígida, Jacobia
A=[10,0;0,-1];
function xdot=f(t,x),xdot=A*x,endfunction
function J=Jacobian(t,y),J=A,endfunction
ode("stiff",[0;1],0,1,f,Jacobian)
```

# Ver Também

ode_discrete, ode_root, dassl, impl, odedc, odeoptions, csim, ltitr, rtitr

# Autor

Alan C. Hindmarsh
, mathematics and statistics division, l-316 livermore, ca 94550.19

# Bibliografia

Alan C. Hindmarsh, lsode and lsodi, two new initial value ordinary differential equation solvers, acm-signum newsletter, vol. 15, no. 4 (1980), pp. 10-11.

# Funções Utilizadas

As rotinas associadas podem ser encontradas no diretório routines/integ:

lsode.f lsoda.f lsodar.f

# Name

ode_discrete — solucionador de equações diferenciais ordinárias, simulação de tempo discreto

```
y=ode("discrete",y0,k0,kvect,f)
```

# Parâmetros

y0
: vetor de reais ou matriz (condições iniciais).

t0
: escalar real (tempo inicial).

f
: função externa i.e. função, string ou lista.

k0
: inteiro (tempo inicial).

kvect
: vetor de inteiros.

# Descrição

Com esta sintaxe (primeiro argumento igual a `"discrete"`) `ode`, computa recursivamente `y(k+1)=f(k,y(k))` de um estado inicial `y(k0)` e retorna `y(k)` para `k` em `kvect`. `kvect(1)` deve ser maior do que ou igual a `k0`.

Outros argumentos e opções são os mesmos que para `ode`, veja a página de ajuda de ode.

# Exemplos

```
y1=[1;2;3]; deff("yp=a_function(k,y)","yp=A*y+B*u(k)")
A=diag([0.2,0.5,0.9]); B=[1;1;1];u=1:10;n=5;
y=ode("discrete",y1,1,1:n,a_function);
y(:,2)-(A*y1+B*u(1))

// Agora, y é avaliado em [y3,y5,y7,y9]
y=ode("discrete",y1,1,3:2:9,a_function)
```

# Ver Também

ode

# Name

ode_optional_output — descrição de saídas opcionais de solucionadores de EDOs

# Descrição

Esta página descreve os valores mais importantes retornados nos argumento opcionais do lado esquerdo da função ode `w` e `iw`. São válidos apenas para os solucionadores de EDOs `lsode`, `lsoda` e `lsodar`. Para mais detalhes, pode se consultar os comentários dos códigos das subrotinas FORTRAN `routines/integ/lsod*.f`.

w(11)
: último tamanho do passo em t utilizado (sucessivamente).

w(12)
: o tamanho do passo a ser tentado no próximo passo

w(13)
: o valor corrente da variável independente alcançado pelo solucionador, i.e. o ponto de malha interno t corrente. Ná saída, tcur sempre estará pelo menos tão longe quanto o argumento t, mas pode estar mais longe (se foi realizada interpolação).

w(14)
: um fator de escala de tolerância, maior que 1.0, computado quando se detecta requerer muita precisão (istate = -3 se detectado no início do problema, istate = -2, em caso contrário). Se itol for inalterado, mas rtol e atol forem uniformemente escalados por um fator de `tolsf=w(14)` para a próxima chamada, emtão supõe-se que o solucionador pode continuar. (o usuário também pode ignorar `tolsf` e alterar os parâmetros de tolerância de qualquer outro modo apropriado.)

w(15)
: o valor de t no tempo da última alternação de método, se houver. Este valor não é significante com o solucionador `lsode`.

iw(10)
: o número de avaliações de `g` para problema até então. Este valor é significante apenas para o solucionador `lsodar`.

iw(11)
: o número de passos tomados para o problema até então

iw(12)
: o número de avaliações de f até então

iw(13)
: o número de avaliações do jacobiano (e de decomposições lu da matriz) para o problema até então.

iw(14)
: a última ordem do método utilizada (sucessivamente).

iw(15)
: a ordem a ser tentada no próximo passo

iw(16)
: o índice do componente de maior magnitude no vetor de erro local ponderado ( e(i)/ewt(i) ), sobre um erro, retorna istate = -4 or -5.

iw(17)
: o comprimento de `w` realmente requerido, supondo que o comprimento de rwork deve ser fixado para o resto do problema, e que a alternância pode ocorrer. Isto é definido sobre retornos normais e sobre um retorno de entrada ilegal para armazenamento insuficiente.

iw(18)
: o comprimento de `iw` realmente requerido, supondo que o comprimento de `iw` deve ser fixado pelo resto do problema, e que a alternância pode ocorrer. Isto é definido sobre retornos normais e sobre um retorno de entrada ilegal para armazenamento insuficiente.

iw(19)
: o indicador do método para o último passo realizado com sucesso. 1 significa adams (nonstiff), 2 significa bdf (stiff). Este valor não é significante com o solucionador `lsode`.

iw(20)
: o indicador do método corrente. 1 significa adams (nonstiff), 2 significa bdf (stiff). Este é o método a ser tentado no próximo passo. Logo, difere de `iw(19)` apenas se uma alternância de métodos foi realizada. Este valor não é significante com o solucionador `lsode`.

# Name

ode_root — solucionador de EDOs com busca de raízes

```
y,rd[,w,iw]=ode("root",y0,t0,t [,rtol [,atol]],f [,jac],ng,g [,w,iw])
```

# Parâmemtros

y0
: vetor de reais ou matriz (condições iniciais)

t0
: escalar real (tempo inicial)

t
: vetor de reais (tempos nos quais a solução é computada).

f
: função externa i.e. função, string ou lista.

rtol,atol
: constantes reais ou vetores reais com o mesmo tamanho que `y`.

jac
: função externa i.e. função, string ou lista.

w,iw
: vetores de reais

ng
: inteiros.

g
: função externa i.e. função, string ou lista.

# Descrição

Com esta sintaxe (primeiro argumento igual a `"root"`) `ode` computa a solução da EDO `dy/dt=f(t,y)` até que o estado `y(t)` cruze a superfície `g(t,y)=0`.

`g` deve fornecer a equação da superfície. É uma função externa, i.e. uma função com sintaxe especificada, ou o nome de uma subrotina FORTRAN ou função C (string) com seqüência de chamamento especificada ou uma lista.

Se `g` é uma função, a sintaxe deve ser como segue:

```
z=g(t,y)
```

onde `t` é um escalar real (tempo) e `y` um vetor de reais (estado). Ela retorna um vetor de tamanho `ng` que corresponde às `ng` restrições. Se `g` é um string, ele se refere a uma subrotina FORTRAN ou uma função C, com a seguinte seqüência de chamamento: `g(n,t,y,ng,gout)` onde `ng` é o número de restrições e `gout` é o valor de `g` (saída do programa). Se `g` é uma lista, as mesmas convenções para `f` se aplicam (ver ajuda de ode).

A saída `rd` é um vetor `1 x k` . A primeira entrada contém o tempo de parada. Outras entradas indicam que componentes de `g` possuem sinal trocado. `k` maior que 2 indica que mais de uma superfície (`(k-1)` superfícies) foram simultaneamente atravessadas.

Outros argumentos e opções são os mesmos que para `ode`, ver a página de ajuda de ode.

# Exemplos

```
// Integração da equação diferencial
// dy/dt=y , y(0)=1, e acha o tempo mínimo t tal que y(t)=2
deff("[ydot]=f(t,y)","ydot=y")
deff("[z]=g(t,y)","z=y-2")
y0=1;ng=1;
[y,rd]=ode("roots",y0,0,2,f,ng,g)

deff("[z]=g(t,y)","z=y-[2;2;33]")
[y,rd]=ode("roots",1,0,2,f,3,g)
```

# Ver Também

dasrt, ode

# Name

odedc — solucionador de EDOs contínuas/discretas

```
yt=odedc(y0,nd,stdel,t0,t,f)
```

# Parâmetros

y0
: vetor coluna de reais (condições iniciais), `y0=[y0c;y0d]` onde `y0d` tem `nd` componentes.

nd
: inteiro, dimensão de `y0d`

stdel
: vetor de reais com uma ou duas entradas, `stdel=[h, delta]` (com `delta=0` como valor padrão).

t0
: escalar real (tempo inicial).

t
: vetor (linha de reais), instantes onde `yt` é calculado .

f
: função externa i.e. função, string ou lista com seqüência de chamamento: `yp=f(t,yc,yd,flag)`.

# Descrição

`y=odedc([y0c;y0d],nd,[h,delta],t0,t,f)` computa a solução de um sistema misto contínuo/discreto. O estado de sistema discreto `yd_k` é embutido em uma função de tempo seccional constante `yd(t)` como segue:

```
yd(t)=yd_k para t em
[t_k=atraso+k*h,t_(k+1)=delay+(k+1)*h[ (com atraso=h*delta).
```

As equações simuladas são agora:

```
dyc/dt=f(t,yc(t),yd(t),0),  para t em [t_k,t_(k+1)[
yc(t0)=y0c
```

e nos instantes `t_k` a variável discreta `yd` é atualizada por:

```
yd(t_k+)=f(yc(t_k-),yd(t_k-),1)
```

Note que, utilizando a definição de `yd(t)`, a última equação fornece

```
yd_k = f (t_k,yc(t_k-),yd(t_(k-1)),1)  (yc é de tempo contínuo: yc(t_k-)=yc(tk)
```

Os parâmetros de chamada de `f` são fixados: `ycd=f(t,yc,yd,flag)`; esta função deve retornar a derivada do vetor `yc` se `flag=0` ou a atualização de `yd` se `flag=1`.

`ycd=dot(yc)` deve ser um vetor de mesma dimensão que `yc` se `flag=0` e `ycd=update(yd)` deve ser um vetor com a mesma dimensão que `yd` se `flag=1`.

`t` é um vetor de constantes onde a solução `y` é computada.

`y` é o vetor `y=[y(t(1)),y(t(2)),...]`. Esta função pode ser chamada com os mesmo parâmetro opcionais da função `ode` (desde que `nd` e `stdel` sejam dados na seqüência de chamamento como segundo e terceiro parâmetros). Em particular, flags de integração e tolerâncias podem ser ajustadas. Parâmetros opcionais podem ser ajustados pela função `odeoptions`.

Um exemplo para chamada de rotina externa é encontrado em `SCIDIR/default/fydot2.f`

Rotinas externas podem ser ligadas dinamicamente (ver `link`).

# Exemplos

```
//sistema linear com entrada alternante
deff('xdu=phis(t,x,u,flag)','if flag==0 then xdu=A*x+B*u; else xdu=1-u;end');
x0=[1;1];A=[-1,2;-2,-1];B=[1;2];u=0;nu=1;stdel=[1,0];u0=0;t=0:0.05:10;
xu=odedc([x0;u0],nu,stdel,0,t,phis);x=xu(1:2,:);u=xu(3,:);
nx=2;
plot2d1('onn',t',x',[1:nx],'161');
plot2d2('onn',t',u',[nx+1:nx+nu],'000');

//função externa FORTRAN( ver fydot2.f):
norm(xu-odedc([x0;u0],nu,stdel,0,t,'phis'),1)

//feedback (resposta) amostrado
//
//        |     xcdot=fc(t,xc,u)
//  (sistema)   |
//        |     y=hc(t,xc)
//
//
//        |     xd+=fd(xd,y)
//  (feedback) |
//        |     u=hd(t,xd)
//
deff('xcd=f(t,xc,xd,iflag)',...
  ['if iflag==0 then '
   '  xcd=fc(t,xc,e(t)-hd(t,xd));'
   'else '
   '  xcd=fd(xd,hc(t,xc));'
   'end']);
A=[-10,2,3;4,-10,6;7,8,-10];B=[1;1;1];C=[1,1,1];
Ad=[1/2,1;0,1/20];Bd=[1;1];Cd=[1,1];
deff('st=e(t)','st=sin(3*t)')
deff('xdot=fc(t,x,u)','xdot=A*x+B*u')
deff('y=hc(t,x)','y=C*x')
deff('xp=fd(x,y)','xp=Ad*x + Bd*y')
deff('u=hd(t,x)','u=Cd*x')
h=0.1;t0=0;t=0:0.1:2;
x0c=[0;0;0];x0d=[0;0];nd=2;
xcd=odedc([x0c;x0d],nd,h,t0,t,f);
norm(xcd-odedc([x0c;x0d],nd,h,t0,t,'fcd1')) // cálculo rápido (ver fydot2.f)
plot2d([t',t',t'],xcd(1:3,:)');
xset("window",2);plot2d2("gnn",[t',t'],xcd(4:5,:)');
```

```
xset("window",0);
```

## Ver Também

ode, odeoptions, csim, external

# Name

odeoptions — ajusta opções para solucionadores de EDO

```
odeoptions()
```

# Descrição

Esta função exibe interativamente um comando que deveria ser executado para se ajustar diversas opções para solucionadores de EDOs. A variável global `%ODEOPTIONS` ajusta as opções.

CUIDADO: a função `ode` verifica se esta variável existe e, neste caso, a utiliza. Para se utilizar valores padrões, deve-se limpar esta variável. Note que `odeoptions` não cria esta variável. Para criá-la, você deve executar a linha de comando exibida por `odeoptions`.

A variável `%ODEOPTIONS` é um vetor com os seguintes elementos:

```
[itask,tcrit,h0,hmax,hmin,jactyp,mxstep,maxordn,maxords,ixpr,ml,mu]
```

O valor padrão é:

```
[1,0,0,%inf,0,2,500,12,5,0,-1,-1]
```

O significado de cada elemento é descrito abaixo

`itask` 1 : computação normal nos tempos especificados 2 : computação nos tempos de malha (dados na primeira linha da saída de `ode`) 3 : um passo em um ponto interno da malha e retorna 4 : computação normal sem overshooting ("desvalorização excessiva") `tcrit` 5 : um passo, sem passagem de `tcrit`, e retorna

`tcrit` assume `itask` igual a 4 ou 5, descrito acima

`h0` primeiro passo tentado

`hmax` tamanho máximo do passo

`hmin` tamanho mínimo do passo

`jactype` 0 : iterações funcionais, nenhum jacobiano usado (`"adams"` ou `"stiff"` apenas) 1 : jacobiano completo fornecido pelo usuário 2 : jacobiano completo internamente gerado 3 : jacobiano diagonal internamente gerado (`"adams"` ou `"stiff"` apenas) 4 : jacobiano com bandas fornecido pelo usuário (ver `ml` e `mu` abaixo) 5 : jacobiano com bandas internamente gerado (`ver` ml e `mu` abaixo)

`maxordn` máxima ordem não-stiff permitida, no máximo 12

`maxords` máxima ordem stiff permitida, no máximo 5

`ixpr` nível de impressão, 0 ou 1

`ml`,`mu` se `jactype` é igual a 4 ou 5, `ml` e `mu` são as semi-bandas inferior e superior do jacobiano com bandas: a banda são os i,j's com i-ml <= j <= ny-1. Se `jactype` é igual a 4 a função jacobiana deve retornar uma matriz J que é ml+mu+1 x ny (onde ny=dim de y em ydot=f(t,y)) tal que a coluna 1 de J é feita de mu zeros seguidos por df1/dy1, df2/dy1, df3/dy1, ... (1+ml entradas não-nulas possivelmente) a coluna 2 é feita de mu-1 zeros seguidos por df1/dx2, df2/dx2, etc

# Ver Também

ode

# Name

bvode — boundary value problems for ODE using collocation method
bvodeS — Simplified call to bvode

```
zu=bvode(xpoints,N,m,x_low,x_up,zeta,ipar,ltol,tol,fixpnt,fsub,dfsub,gsub,dgsub
zu=bvodeS(xpoints,m,N,x_low,x_up,fsub,gsub,zeta, <optional_args>)
```

# Parameters

zu
: a column vector of size M. The solution of the ode evaluated on the mesh given by points. It contains z(u(x)) for each requested points.

xpoints
: an array which gives the points for which we want to observe the solution.

N
: a scalar with integer value,number of differential equations (N <= 20).

m
: a vector of size `N` with integer elements. `It is the vector of order of each differential equation: m(i)` gives the order of the i-th differential equation. In the following, `M` will represent the sum of the elements of `m`.

x_low
: a scalar: left end of interval

x_up
: a scalar: right end of interval

zeta
: `a vector of size M,zeta(j)` gives j-th side condition point (boundary point). One must have x_low<=zeta(j) <= zeta(j+1)`<=x_up`

  All side condition points must be mesh points in all meshes used, see description of `ipar(11)` and `fixpnt` below.

ipar
: an array with 11 integer elements:

  `[nonlin, collpnt, subint, ntol, ndimf, ndimi, iprint, iread, iguess, rstart,nfxpnt]`

  nonlin: ipar(1)
  : 0 if the problem is linear, 1 if the problem is nonlinear

  collpnt: ipar(2)
  : Gives the number of collocation points per subinterval where max(m(j)) <= collpnt <= 7

    if `ipar(2)=0` then `collpnt` is set to max ( max(m(j))+1 , 5-max(m(j)) )

  subint: ipar(3)
  : Gives the number of subintervals in the initial mesh. if `ipar(3) = 0` then bvode arbitrarily sets `subint = 5`.

  ntol: ipar(4)
  : Gives the number of solution and derivative tolerances. We require 0 < ntol <= M. `ipar(4)` must be set to the dimension of the `tol` argument or to `0`. In the latter case the actual value will automatically be set to `size(tol,'*')`.

ndimf: ipar(5)
Gives the dimension of `fspace` (a real work array). its value provides a constraint on nmax the maximum number of subintervals.

The ipar(5) value must respect the constraint ipar(5)>=nmax*nsizef where

`nsizef=4+3*M+(5+collpnt*N)*(collpnt*N+M)+(2*M-nrec)*2*M` (nrec is the number of right end boundary conditions).

ndimi: ipar(6)
Gives the dimension of ispace (an integer work array). its value provides a constraint on nmax, the maximum number of subintervals.

The ipar(6) value must respect the constraint ipar(6)>=nmax*nsizei where

`nsizei=3 + collpnt*N+M`.

iprint: ipar(7)
output control, make take the following values:

-1
for full diagnostic printout

0
for selected printout

1
for no printout

iread: ipar(8)

= 0
causes bvode to generate a uniform initial mesh.

= xx
Other values are not implemented yet in Scilab

= 1
if the initial mesh is provided by the user. it is defined in fspace as follows: the mesh

will occupy `fspace(1), ..., fspace(n+1)`. the user needs to supply only the interior mesh points `fspace(j) = x(j), j = 2, ..., n.`

= 2 if the initial mesh is supplied by the user
as with `ipar(8)=1`, and in addition no adaptive mesh selection is to be done.

iguess: ipar(9)

= 0
if no initial guess for the solution is provided.

= 1
if an initial guess is provided by the user trought the argument `guess`.

= 2
if an initial mesh and approximate solution coefficients are provided by the user in fspace. (the former and new mesh are the same).

= 3
if a former mesh and approximate solution coefficients are provided by the user in fspace, and the new mesh is to be taken twice as coarse; i.e.,every second point from the former mesh.

= 4
if in addition to a former initial mesh and approximate solution coefficients, a new mesh is provided in fspace as well. (see description of output for further details on iguess = 2, 3, and 4.)

ireg: ipar(10)

= 0
if the problem is regular

= 1
if the first relax factor is equal to ireg, and the nonlinear iteration does not rely on past covergence (use for an extra sensitive nonlinear problem only).

= 2
if we are to return immediately upon (a) two successive nonconvergences, or (b) after obtaining error estimate for the first time.

nfxpnt: ipar(11)
Gives the number of fixed points in the mesh other than `x_low` and `x_up` (the dimension of `fixpnt`). `ipar(11)` must be set to the dimension of the `fixpnt` argument or to `0`. In the latter case the actual value will automatically be set to `size(fixpnt,'*')`.

ltol
an array of dimension `ntol`=ipar(4). `ltol(j) = l` specifies that the j-th tolerance in the tol array controls the error in the `l`-th component of $Z$. It is also required that:

```
1 <= ltol(1) < ltol(2) < ... < ltol(ntol) <= M
```

tol
an array of dimension `ntol=ipar(4)`.

`tol(j)` is the error tolerance on the `ltol(j)`-th component of $Z$. Thus, the code attempts to satisfy $\left((z(v)-z(u))_{\mathrm{ltol}(j)}\right) \leq \mathrm{tol}(j)\cdot\left((z(u))_{\mathrm{ltol}(j)}\right)+\mathrm{tol}(j), \mathrm{for}\, j=1:\mathrm{ntol}$ on each subinterval

if $v$ is the approximate solution vector an u the exact solution (unknown).

fixpnt
an array of dimension `nfxpnt=ipar(11)`. it contains the points, other than `x_low` and `x_up`, which are to be included in every mesh. The code requires that all side condition points other than `x_low` and `x_up` (see description of zeta ) be included as fixed points in `fixpnt`.

fsub
an external used to evaluate the column vector `f=` $\left[f_1(x,z(u(x))); f_2(x,z(u(x))), \cdots, f_N(x,z(u(x)))\right]$ for any x such as x_low <= x <= x_up and for any z=z(u(x)) (see description below)

The external must have the headings:

- In Fortran the calling sequence must be:

```
subroutine fsub(x,zu,f)
double precision zu(*), f(*),x
```

```
```

- In C the function prototype must be

```
void fsub(double *x, double *zu, double *f)
```

- And in Scilab

```
function f=fsub(x,zu,parameters)
```

dfsub

an external used to evaluate the Jacobian of `f(x,z(u))` at a point x. Where `z(u(x))` is defined as for `fsub` and the (`N`) by (`M`) array df should be filled by the partial derivatives of f:

$$\mathrm{df}(i,j) = \frac{\partial f_i}{\partial z_j} \text{for} \begin{cases} i = 1{:}N \\ j = 1{:}M \end{cases}$$

The external must have the headings:

- In Fortran the calling sequence must be:

```
subroutine dfsub(x,zu,df)
double precision zu(*), df(*),x
```

- In C the function prototype must be

```
void dfsub(double *x, double *zu, double *df)
```

- And in Scilab

```
function df=dfsub(x,zu,parameters)
```

gsub

an external used to evaluate $g_i(\zeta_i, z(u(\zeta_i)))$ given z= $z(u(\zeta_i))$ `z = zeta(i)` for `1<=i<=M.`

The external must have the headings:

- In Fortran the calling sequence must be:

```
subroutine gsub(i,zu,g)
double precision zu(*), g(*)
integer i
```

- In C the function prototype must be

```
void gsub(int *i, double *zu, double *g)
```

- And in Scilab

```
function g=gsub(i,zu,parameters)
```

Note that in contrast to `f` in `fsub`, here only one value per call is returned in `g`.

dgsub
: an external used to evaluate the i-th row of the Jacobian of `g(x,u(x))`. Where `z(u)` is as for fsub, i as for gsub and the M-vector `dg` should be filled with the partial derivatives of g, viz, for a particular call one calculates

$$\mathrm{dg}(i,j) = \frac{\partial g_i}{\partial z_j} \text{ for} \begin{cases} i = 1{:}M \\ j = 1{:}M \end{cases}$$

The external must have the headings:

- In Fortran the calling sequence must be:

```
subroutine dgsub(i,zu,dg)
double precision zu(*), dg(*)
```

- In C the function prototype must be

```
void dgsub(int *i, double *zu, double *dg)
```

- And in Scilab

```
function dg=dgsub(i,zu,parameters)
```

guess
: An external used to evaluate the initial approximation for `z(u(x))` and `dmval(u(x))` the vector of the mj-th derivatives of `u(x)`. Note that this subroutine is used only if `ipar(9) = 1`, and then all `M` components of zu and N components of dmval should be computed for any x such as x_low <= x <= x_up.

The external must have the headings:

- In Fortran the calling sequence must be:

```
subroutine guess(x,zu,dmval)
double precision x,z(*), dmval(*)
```

- In C the function prototype must be

```
void fsub(double *x, double *zu, double *dmval)
```

- And in Scilab

```
function [dmval,zu]=fsub(x,parameters)
```

<optional_args>
It should be either:

- any left part of the ordered sequence of values: `guess, dfsub, dgsub, fixpnt, ndimf, ndimi, ltol, tol, ntol,nonlin, collpnt, subint, iprint, ireg, ifail`
- or any sequence of `arg_name=argvalue` with `arg_name` in: `guess`, `dfsub`, `dgsub`, `fixpnt`, `ndimf`, `ndimi`, `ltol`, `tol`, `ntol`, `nonlin`, `collpnt`, `subint`, `iprint`, `ireg`, `ifail`

Where all these arguments excepted `ifail` are described above. `ifail` can be used to display the bvode call corresonding to the selected optional arguments. If `guess` is given `iguess` is set to 1

# Description

These functions solves a multi-point boundary value problem for a mixed order system of ode-s given by

$$\begin{cases} u_i^{(m_i)} = f_i(x, z(u(x))) \begin{cases} i = 1{:}N \\ x \in (x_l, x_u) \end{cases} \\ g_j(\zeta_j, z(u(\zeta_j))) = 0, j = 1{:}M \end{cases}$$

Where

$$\begin{cases} M = \sum_{i=1}^{N} m_i \\ z(u(x)) = \left[u_1(x); u_1^1(x); \cdots u_1^{(m_1-1)}(x); \cdots; u_N(x); u_N^1(x); \cdots; u_N^{(m_N-1)}(x)\right] \\ x_l \leq \zeta_1 \leq \cdots \leq \zeta_M \leq x_u \end{cases}$$

The argument `zu` used by the external functions and returned by `bvode` is the column vector formed by the components of z(u(x)) for a given `x`.

The method used to approximate the solution u is collocation at gaussian points, requiring m(i)-1 continuous derivatives in the i-th component, i = 1:N. here, k is the number of collocation points (stages) per subinterval and is chosen such that k .ge. max m(i). a runge-kutta-monomial solution representation is utilized.

# Examples

The first two problems below are taken from the paper [1] of the Bibliography.

- **The problem 1** describes a uniformy loaded beam of variable stifness, simply supported at both end.

  It may be defined as follow :

  Solve the fourth order differential equation:

$$\frac{d^4}{dx^4}u(x) = \frac{1-6.x^2.\frac{d^3}{dx^3}u(x)-6.x.\frac{d^2}{dx^2}u(x)}{x^3}$$

  Subjected to the boundary conditions:

$$\begin{cases} u(1) = 0 \\ \frac{d^2}{dx^2}u(x)(1) = 0 \\ u(2) = 0 \\ \frac{d^2}{dx^2}u(x)(2) = 0 \end{cases}$$

  The exact solution of this problem is known to be:

$$u(x) = \frac{1}{4}(10\log(2) - 3)(1 - x) + \frac{1}{2}\left[\frac{1}{x} + (3 + x)\log(x) - x\right]$$

```
N=1;// just one differential equation
m=4;//a fourth order  differential equation
M=sum(m);

x_low=1;x_up=2; // the x limits
zeta=[x_low,x_low,x_up,x_up]; //two constraints (on the value of u and its sec

//The external functions
//Theses functions are called by the solver with zu=[u(x);u'(x);u''(x);u'''(x

// - The function which computes the right hand side of the differential equat
function f=fsub(x,zu),f=(1-6*x^2*zu(4)-6*x*zu(3))/x^3,endfunction

// - The function which computes the derivative of fsub with respect to zu
function df=dfsub(x,zu),df=[0,0,-6/x^2,-6/x],endfunction

// - The function which computes the ith constraint for a given i
function g=gsub(i,zu),
  select i
  case 1 then  //x=zeta(1)=1
    g=zu(1) //u(1)=0
  case 2 then //x=zeta(2)=1
    g=zu(3) //u''(1)=0
  case 3 then //x=zeta(3)=2
    g=zu(1) //u(2)=0
  case 4 then  //x=zeta(4)=2
    g=zu(3) //u''(2)=0
  end
```

```
endfunction

// - The function which computes the derivative of gsub with respect to z
function dg=dgsub(i,z)
  select i
  case 1 then  //x=zeta(1)=1
    dg=[1,0,0,0]
  case 2 then //x=zeta(2)=1
    dg=[0,0,1,0]
  case 3 then //x=zeta(3)=2
     dg=[1,0,0,0]
  case 4 then  //x=zeta(4)=2
    dg=[0,0,1,0]
  end
endfunction

// - The function which computes the initial guess, unused here
function [zu,mpar]=guess(x),zu=0;mpar=0,endfunction

 //define the function which computes the exact value of u for a given x ( for
function zu=trusol(x)
  zu=0*ones(4,1)
  zu(1) =  0.25*(10*log(2)-3)*(1-x) + 0.5 *( 1/x   + (3+x)*log(x) - x)
  zu(2) = -0.25*(10*log(2)-3)       + 0.5 *(-1/x^2 + (3+x)/x      + log(x) - 1
  zu(3) = 0.5*( 2/x^3 + 1/x   - 3/x^2)
  zu(4) = 0.5*(-6/x^4 - 1/x/x + 6/x^3)
endfunction

fixpnt=[ ];//All boundary conditions are located at x_low and x_up

//    nonlin  collpnt n ntol ndimf  ndimi iprint iread iguess rstart nfxpnt
ipar=[0       0       1 2    2000   200   1      0     0      0      0     ]

ltol=[1,3];//set tolerance control on zu(1) and zu(3)
tol=[1.e-11,1.e-11];//set tolreance values for these two controls
xpoints=x_low:0.01:x_up;

zu=bvode(xpoints,N,m,x_low,x_up,zeta,ipar,ltol,tol,fixpnt,...
        fsub,dfsub,gsub,dgsub,guess)
//check the constraints
zu([1,3],[1 $]) //should be zero
plot(xpoints,zu(1,:)) // the evolution of the solution u
zu1=[];for x=xpoints,zu1=[zu1,trusol(x)]; end;
norm(zu-zu1)
```

- Same problem using `bvodeS` and an initial guess

```
function [z,lhS]=zstart(x)
  z=zeros(5,1);z(5)=1;
  lhS=[0;1];
endfunction
zu=bvode(xpoints,N,m,x_low,x_up,zeta,ltol=ltol,tol=tol,guess=zstart)
```

- **The problem 2** describes the small finite deformation of a thin shallow spherical cap of constant thickness subject to a quadratically varying axisymmetric external pressure distribution. Here phi

is the meridian angle change of the deformed shell and psi is a stress function. For eps=mu=1e-3 two different solutions may found depending on the starting point

$$\begin{cases} \frac{\epsilon^4}{\mu}\left(\varphi'' + \frac{\varphi'}{x} - \frac{\varphi}{x^2}\right) + \psi\left(1 - \frac{\varphi}{x}\right) - \varphi + \gamma x\left(1 - \frac{x^2}{2}\right) = 0 \\ \mu\left(\psi'' + \frac{\psi'}{x} - \frac{\psi}{x^2}\right) - \varphi\left(1 - \frac{\varphi}{2x}\right) = 0 \\ 0 < x < 1 \end{cases}$$

Subject to the boundary conditions

$$\begin{cases} \varphi = 0 \\ x\psi' - 0.3\psi + 0.7x = 0 \end{cases}$$

for `x=0` and `x=1`

```
N=2;// two differential equations
m=[2 2];//each differential equation is of second  order
M=sum(m);

x_low=0;x_up=1; // the x limits
zeta=[x_low,x_low, x_up x_up]; //two  constraints on each bound.

//The external functions
//Theses functions are called by the solver with zu=[u1(x);u1'(x);u2(x);u2'(x

// - The function which computes the right hand side of the differential equat
function f=fsub2(x,zu,eps,dmu,eps4mu,gam,xt),
   f=[zu(1)/x^2-zu(2)/x+(zu(1)-zu(3)*(1-zu(1)/x)-gam*x*(1-x^2/2))/eps4mu //phi
      zu(3)/x^2-zu(4)/x+zu(1)*(1-zu(1)/(2*x))/dmu];//psi''
endfunction

// - The function which computes the derivative of fsub with respect to zu
function df=dfsub2(x,zu,eps,dmu,eps4mu,gam,xt),
  df=[1/x^2+(1+zu(3)/x)/eps4mu, -1/x, -(1-zu(1)/x)/eps4mu, 0
      (1-zu(1)/x)/dmu             0    1/x^2              -1/x];
endfunction

// - The function which computes the ith constraint for a given i
function g=gsub2(i,zu),
  select i
  case 1 then  //x=zeta(1)=0
    g=zu(1) //u(0)=0
  case 2 then //x=zeta(2)=0
    g=-0.3*zu(3) //x*psi'-0.3*psi+0.7x=0
  case 3 then //x=zeta(3)=1
    g=zu(1) //u(1)=0
  case 4 then  //x=zeta(4)=1
    g=1*zu(4)-0.3*zu(3)+0.7*1 //x*psi'-0.3*psi+0.7x=0
  end
endfunction

// - The function which computes the derivative of gsub with respect to z
function dg=dgsub2(i,z)
  select i
```

```
  case 1 then  //x=zeta(1)=1
    dg=[1,0,0,0]
  case 2 then //x=zeta(2)=1
    dg=[0,0,-0.3,0]
  case 3 then //x=zeta(3)=2
     dg=[1,0,0,0]
  case 4 then  //x=zeta(4)=2
    dg=[0,0,-0.3,1]
  end
endfunction

gam=1.1
eps=1d-3
dmu=eps
eps4mu=eps^4/dmu
xt=sqrt(2*(gam-1)/gam)

fixpnt=[ ];//All boundary conditions are located at x_low and x_up
collpnt=4;
nsizef=4+3*M+(5+collpnt*N)*(collpnt*N+M)+(2*M-2)*2*M ;
nsizei=3 + collpnt*N+M;;
nmax=200;
//    nonlin  collpnt n  ntol ndimf        ndimi        iprint iread iguess rs
ipar=[1       k       10 4    nmax*nsizef  nmax*nsizei   -1      0     0

ltol=1:4;//set tolerance control on zu(1) zu(2 ) zu(3) and zu(4)
tol=[1.e-5,1.e-5,1.e-5,1.e-5];//set tolreance values for these four controls
xpoints=x_low:0.01:x_up;

zu=bvode(xpoints,N,m,x_low,x_up,zeta,ipar,ltol,tol,fixpnt,...
        fsub2,dfsub2,gsub2,dgsub2,guess2);
scf(1);clf();plot(xpoints,zu([1 3],:)) // the evolution of the solution phi an

//using an initial guess
// - The function which computes the initial guess, unused here
function [zu,dmval]=guess2(x,gam),
   cons=gam*x*(1-x^2/2)
   dcons=gam*(1-3*x^2/2)
   d2cons=-3*gam*x
   dmval=zeros(2,1)
   if x>xt then
     zu=[0 0 -cons -dcons]
     dmval(2)=-d2cons
   else
     zu=[2*x;2;-2*x+cons;-2*dcons]
     dmval(2)=d2cons
   end
endfunction
ipar(9)=1;//iguess

zu2=bvode(xpoints,N,m,x_low,x_up,zeta,ipar,ltol,tol,fixpnt,...
        fsub2,dfsub2,gsub2,dgsub2,guess2);
scf(2);clf();plot(xpoints,zu2([1 3],:)) // the evolution of the solution phi a
```

- An eigenvalue problem:

```
// y''(x)=-la*y(x)
// BV: y(0)=y'(0); y(1)=0
// Eigenfunctions and eigenvalues are y(x,n)=sin(s(n)*(1-x)), la(n)=s(n)^2,
// where s(n) are the zeros of f(s,n)=s+atan(s)-(n+1)*pi, n=0,1,2,...
// To get a third boundary condition, we choose y(0)=1
// (With y(x) also c*y(x) is a solution for each constant c.)
// We solve the following ode system:
// y''=-la*y
// la'=0
// BV: y(0)=y'(0), y(0)=1; y(1)=0
// z=[y(x) ; y'(x) ; la]

function rhs=fsub(x,z)
  rhs=[-z(3)*z(1);0]
endfunction

function g=gsub(i,z)
  g=[z(1)-z(2) z(1)-1 z(1)]
  g=g(i)
endfunction

// The following start function is good for the first 8 eigenfunctions.
function [z,lhs]=ystart(x,z,la0)
  z=[1;0;la0]
  lhs=[0;0]
endfunction

a=0;b=1;
m=[2;1];
n=2;
zeta=[a a b];
N=101;
x=linspace(a,b,N)';

// We have s(n)-(n+1/2)*pi -> 0 for n to infinity.
la0=input('n-th eigenvalue: n= ?');la0=(%pi/2+la0*%pi)^2;

z=bvodeS(x,m,n,a,b,fsub,gsub,zeta,ystart=list(ystart,la0));

clf()
plot(x,[z(1,:)' z(2,:)'])
xtitle(['Startvalue =  '+string(la0);'Eigenvalue = '+string(z(3,1))],'x',' ')
legend(['y(x)';'y''(x)'])
```

- A boundary value problem with more than one solution.

```
// DE: y''(x)=-exp(y(x))
// BV: y(0)=0; y(1)=0
// This boundary value problem has more than one solution.
// It is demonstrated how to find two of them with the help of
// some preinformation of the solutions y(x) to build the function ystart.
// z=[y(x);y'(x)]

a=0;b=1;m=2;n=1;
zeta=[a b];
N=101;
```

```
tol=1e-8*[1 1];
x=linspace(a,b,N);

function rhs=fsub(x,z),rhs=-exp(z(1));endfunction

function g=gsub(i,z)
  g=[z(1) z(1)]
  g=g(i)
endfunction

function [z,lhs]=ystart(x,z,M)
  //z=[4*x*(1-x)*M ; 4*(1-2*x)*M]
  z=[M;0]
  //lhs=[-exp(4*x*(1-x)*M)]
  lhs=0
endfunction

for M=[1 4]
   if M==1
      z=bvodeS(x,m,n,a,b,fsub,gsub,zeta,ystart=list(ystart,M),tol=tol);
   else
      z1=bvodeS(x,m,n,a,b,fsub,gsub,zeta,ystart=list(ystart,M),tol=tol);
   end
end

// Integrating the ode yield e.g. the two solutions yex and yex1.

function y=f(c),y=c.*(1-tanh(sqrt(c)/4).^2)-2;endfunction
c=fsolve(2,f);

function y=yex(x,c)
  y=log(c/2*(1-tanh(sqrt(c)*(1/4-x/2)).^2))
endfunction

function y=f1(c1), y=2*c1^2+tanh(1/4/c1)^2-1;endfunction
c1=fsolve(0.1,f1);

function y=yex1(x,c1)
  y=log((1-tanh((2*x-1)/4/c1).^2)/2/c1/c1)
endfunction

disp(norm(z(1,:)-yex(x)),'norm(yex(x)-z(1,:))= ')
disp(norm(z1(1,:)-yex1(x)),'norm(yex1(x)-z1(1,:))= ')
clf();
subplot(2,1,1)
plot2d(x,z(1,:),style=[5])
xtitle('Two different solutions','x',' ')
subplot(2,1,2)
plot2d(x,z1(1,:),style=[5])
xtitle(' ','x',' ')
```

- A multi-point boundary value problem.

```
// DE y'''(x)=1
// z=[y(x);y'(x);y''(x)]
// BV: y(-1)=2 y(1)=2
```

```
// Side condition: y(0)=1

a=-1;b=1;c=0;
// The side condition point c must be included in the array fixpnt.
n=1;
m=[3];

function rhs=fsub(x,z)
  rhs=1
endfunction

function g=gsub(i,z)
  g=[z(1)-2 z(1)-1 z(1)-2]
  g=g(i)
endfunction

N=10;
zeta=[a c b];
x=linspace(a,b,N);

z=bvodeS(x,m,n,a,b,fsub,gsub,zeta,fixpnt=c);

function y=yex(x)
y=x.^3/6+x.^2-x./6+1
endfunction

disp(norm(yex(x)-z(1,:)),'norm(yex(x)-z(1,:))= ')
```

# See Also

link, external, ode, dassl

# Used Functions

This function is based on the Fortran routine `colnew` developped by

U. Ascher, Department of Computer Science, University of British Columbia, Vancouver, B.C. V6T 1W5, Canada

G. Bader, institut f. Angewandte mathematik university of Heidelberg; im Neuenheimer feld 294d-6900 Heidelberg 1

# Authors

This help is based on the original colnew.f comments, adapted to Scilab by J.P Chancelier ENPC, on the bvodeS help page due to Rainer von Seggern, both merged and revised by S. Steer INRIA

# Bibliography


1. U. Ascher, J. Christiansen and R.D. Russell, collocation software for boundary-value ODEs, acm trans. math software 7 (1981), 209-222. this paper contains EXAMPLES where use of the code is demonstrated.

2. G. Bader and U. Ascher, a new basis implementation for a mixed order boundary value ode solver, siam j. scient. stat. comput. (1987).

3. U. Ascher, J. Christiansen and R.D. Russell, a collocation solver for mixed order systems of boundary value problems, math. comp. 33 (1979), 659-679.

4. U. Ascher, J. Christiansen and R.D. russell, colsys - a collocation code for boundary value problems, lecture notes comp.sc. 76, springer verlag, b. childs et. al. (eds.) (1979), 164-185.

5. C. Deboor and R. Weiss, solveblok: a package for solving almost block diagonal linear systems, acm trans. math. software 6 (1980), 80-87.

# Parte III. Funções Elementares

# Name

abs — valor absoluto, magnitude

```
t=abs(x)
```

# Parâmetros

x
: matriz ou vetor de reais ou complexos

t
: matriz ou vetor de reais

# Descrição

`abs(x)` é o valor absoluto dos elementos de `x`. Quando `x` é de complexos, `abs(x)` é o módulo complexo (magnitude) dos elementos de `x`.

# Exemplos

```
abs([1,%i,-1,-%i,1+%i])
```

# Name

acos — arco-cosseno em relação aos elementos

```
t=acos(x)
```

# Parâmetros

x
: vetor de reais ou complexos

t
: vetor de reais ou complexos

# Descrição

Os componentes do vetor `t` são os arcos-cossenos das entradas correspondentes do vetor `x`. O domínio de definição é [-1, 1].

`acos` toma valores em :

# Exemplos

```
x=[1,%i,-1,-%i]
cos(acos(x))
```

# Name

acosd — arco-cosseno elemento a elemento com resultado em graus

```
t=acosd(x)
```

# Parâmetros

x
: array de reais ou complexos

t
: array de reais ou complexos

# Descrição

Os componentes do vetor `t` são o arco-cosseno, em graus, das entradas correspondentes do vetor `x` (t=acos(x)*180/%pi). Para dados reais em `[-1, 1]`, os resultados estão em `[0 180];`

# Exemplos

```
x=[-1 0 1 sqrt(2)/2 -sqrt(2)/2 sqrt(3)/2 -sqrt(3)/2];
acosd(x)
```

# Ver Também

acos

# Name

acosh — arco-coseno hiperbólico

```
[t]=acosh(x)
```

# Parâmetros

x
: vetor de reais ou complexos

t
: vetor de reais ou complexos

# Descrição

Os componentes do vetor `t` são os arcos-cosenos hiperbólicos das entradas correspondentes do vetor `x`. O domínio de definição é ]1,+infinito[. A função toma valores em conjuntos.

# Exemplos

```
x=[0,1,%i];
cosh(acosh(x))
```

# Name

acoshm — arco-cosseno hiperbólico da matriz (matriz arco-cosseno hiperbólico)

```
t=acoshm(x)
```

# Parâmetros

x,t
: matriz quadrada de reais ou complexos

# Descrição

acoshm é o arco-cosseno hiperbólico da matriz x (matriz arco-cosseno hiperbólico). Usa a fórmula `t=logm(x+(x+eye())*sqrtm((x-eye())/(x+eye())))` Para matrizes não-simétricas, o resultado pode ser impreciso.

# Exemplos

```
A=[1,2;3,4];
coshm(acoshm(A))
A(1,1)=A(1,1)+%i;
coshm(acoshm(A))
```

# Ver Também

acosh, logm, sqrtm

# Name

acosm — arco-coseno da matriz (matriz arco-cosseno)

```
t=acosm(x)
```

# Parâmetros

x
: matriz quadrada de reais ou complexos

t
: matriz quadrada de reais ou complexos

# Descrição

`t` é o arco-cosseno da matriz `x` (matriz arco-cosseno de x). É usado o método da diagonalização. Para matrizes não-simétricas, o resultado pode ser impreciso. Tem-se `t=-%i*logm(x+%i*sqrtm(eye()-x*x))`

# Exemplos

```
A=[1,2;3,4];
cosm(acosm(A))
```

# Ver Também

acos, sqrtm, logm

# Name

acot — computa o arco-cotangente elemento a elemento do argumento

```
y = acot(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos

# Descrição

computa o arco-cotangente elemento a elemento do argumento Para argumento reais, o resultado é real

As seguintes desigualdades se verificam: `acot(z) = %pi-acot(-z)=%pi/2-atan(z)= %i*acoth(%i*z)+%pi/2*(1-csgn(z+%i))`

# Exemplos

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];
acot(x)/%pi
```

# Ver Também

cotg, acotd

# Referências

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

# Autor

Serge Steer, INRIA

# Name

acotd — computa o arco-cotangente elemento a elemento do argumento com resultado em graus

```
y = acotd(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos

# Descrição

Computa o arco-cotangente elemento a elemento do argumento com resultado em graus. Para argumento real w, o resultado é real.

# Exemplos

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];
acotd(x)
```

# Ver Também

cotd, acot

# Referências

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

# Autor

Serge Steer, INRIA

# Name

acoth — arco-cotangente hiperbólico elemento a elemento

```
y = acoth(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos

# Descrição

Computa o arco-cotangente hiperbólico do argumento elemento a elemento.

# Exemplos

```
x=-30:0.1:30;
x(abs(x)<=1)=%nan;
plot(x,acoth(x))
```

# Ver Também

atanh, coth

# Autor

Serge Steer, INRIA

# Funções Usadas

Esta rotina é baseada na função `atanh`.

# Name

acsc — computa o arco-cossecante elemento a elemento do argumento

```
y = acsc(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos

# Descrição

computa o arco-cossecante elemento a elemento do argumento. Para entradas com valores reais absolutos maiores que 1, o resultado é real.

As seguintes desigualdades se verificam: `acsc(z) = -acsc(-z) = asin(1/z) = %pi/2-asec(z) = %i*acsch(%i*z)`

# Exemplos

```
x=linspace(1,20,200);
x=[-x($:-1:1) %nan x];
plot(x,acsc(x))
```

# Ver Também

csc, acscd

# Referências

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

# Autor

Serge Steer, INRIA

# Name

acscd — computa o arco-cossecante elemento a elemento do argumento com resutado em graus

```
y = acscd(x)
```

# Parâmetros

x
: array de reais

y
: array de reais ou complexos

# Descrição

computa o arco-cossecante elemento a elemento do argumento com resutado em graus. Para entradas reais com valores absolutos maiores que 1, o resultado é real.

# Exemplos

```
x=linspace(1,20,200);
x=[-x($:-1:1) %nan x];
plot(x,acscd(x))
```

# Ver Também

cscd, acsc

# Referências

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

# Autor

Serge Steer, INRIA

# Name

acsch — computa o arco-cossecante hiperbólico elemento a elemento do argumento

```
y = acsch(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos

# Descrição

Computa o arco-cossecante hiperbólico elemento a elemento do argumento. Para argumentos com valores absolutos maiores que 1, o resultado é real.

As seguintes igualdades se verificam: `acsch(z) = -acsch(-z) = asinh(1/z) = csgn(%i+1/z)*asech(-i*z)-%i*%pi/2 = %i*acsc(%i*z)`

# Exemplos

```
x=linspace(1,20,200);
x=[-x($:-1:1) %nan x];
plot(x,acsch(x))
```

# Ver Também

csch

# Referências

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

# Autor

Serge Steer, INRIA

# Name

adj2sp — converte forma de adjacência para matriz esparsa

# Parâmetros

xadj
: vetor de inteiros de comprimento (n+1).

adjncy
: vetor de inteiros de comprimento nz contendo os índices de linha para os elementos correspondentes em anz

anz
: vetor coluna de comprimento nz contendo os elementos não-nulos de A

mn
: vetor linha com duas entradas, `mn=size(A)` (opcional).

A
: matriz quadrada esparsa de reais ou complexos (nz entradas não-nulas)

# Descrição

```
adj2sp converte uma representação em forma de adjacência de uma matriz
em sua representação padrão do Scilab (função utilitária).
xadj, adjncy, anz = representação  de adjacência de A i.e:
```

`xadj(j+1)-xadj(j)` = número de entradas não-nulas na linha j. `adjncy` = índice de coluna das entradas não-nulas nas linha 1, linha 2..., linha n. `anz` = valores das entradas não-nulas nas linha 1, linha 2,..., linha n. `xadj` é um vetor (coluna) de tamanho n+1 e `adjncy` é um vetor (coluna) de inteiros de tamanho `nz=nnz(A)`. `anz` é um vetor de reais de tamanho `nz=nnz(A)`.

# Exemplos

```
A = sprand(100,50,.05);
[xadj,adjncy,anz]= sp2adj(A);
[n,m]=size(A);
p = adj2sp(xadj,adjncy,anz,[n,m]);
A-p,
```

# Ver Também

sp2adj, spcompack

# Name

amell — função "am" de Jacobi

```
[sn]=amell(u,k)
```

# Parâmetros

u
: escalar real ou vetor de reais

k
: escalar

sn
: escalar real ou vetor de reais

# Descrição

Computa a função elíptica de Jacobi `am(u,k)` onde `k` é o parâmetro e `u` é o argumento. Se `u` é um vetor `sn` é o vetor dos valores computados (elemento a elemento) . Usado na função `%sn`.

# Ver Também

delip, %sn, %asn

# Name

and — (&) "E" lógico

```
b=and(A), b=and(A,'*')
b=and(A,'r'), b=and(A,1)
b=and(A,'c'), b=and(A,2)
A&B
```

# Descrição

`and(A)` é o "E" lógico dos elementos da matriz de valores booleanos `A`. `and(A)` retorna `%T` ("true") ("verdadeiro") se, e só se, todas as entradas de `A` são `%T`.

`y=and(A,'r')` (ou, de modo equivalente, `y=and(A,1)`) é o "e" em relação às linhas. Retorna, em cada entrada do vetor linha `y` , o "e" das linhas de `x` (O "e" é realizado no índice de linhas : `y(j)= and(A(i,j),i=1,m)`).

`y=and(A,'c')` (ou, de modo equivalente, `y=and(A,2)`) é o "e" em relação às colunas. Retorna, em cada entrada do vetor coluna `y` , o "e" das colunas de `x` (O "e" é realizado no índice de colunas: `y(i)= and(A(i,j),j=1,n)`)).

`A&B` fornece o `and` elemento a elemento das matrizes de valores booleanos `A` e `B` .`A` e `B` devem ser matrizes com mesmas dimensões ou uma delas deve ser um único booleano.

# Ver Também

not, or

# Name

asec — computa o arco-secante elemento a elemento do argumento

```
y = asec(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos

# Descrição

computa o arco-secante elemento a elemento do argumento. Para valores reais com valores absolutos maiores que 1, o resultado é real

As seguintes igualdades se verificam: `asec(z) = -acsc(-z) = asin(1/z) = %pi/2-asec(x) = %i*acsch(%i*z)`

# Exemplos

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];
asec(x)/%pi
```

# Ver Também

sec, asecd

# Referências

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

# Autor

Serge Steer, INRIA

# Name

asecd — computa o arco-secante elemento a elemento do argumento com resultados em graus

```
y = asecd(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos

# Descrição

computa o arco-secante elemento a elemento do argumento com resultados em graus. Para dados de entrada com valores absolutos maiores que 1, os resultados são reais em `[-90 90]`.

`asecd(x)` is equal to `asec(x)*180/%pi`.

# Exemplos

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];
asecd(x)
```

# Ver Também

asec, secd

# Referências

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

# Autor

Serge Steer, INRIA

# Name

asech — computa o arco-secante hiperbólico elemento a elemento do argumento

```
y = asech(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos

# Descrição

Computa o arco-secante hiperbólico elemento a elemento do argumento Se o argumento é real e tem valor absoluto menor que 1, o resultado é real

As seguintes igualdades se verificam: `asech(x) = acosh(1 ./ x)= %i*csgn(%i*(1#1 ./ x))*asec(x)=csgn(%i*(1 # 1 ./ x))*(%pi/2*(%i+acsch(%i*x)))`

# Exemplos

```
asech(1)
```

# Ver Também

sech

# Referências

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

# Autor

Serge Steer, INRIA

# Name

asin — arco-seno

```
[t]=asin(x)
```

# Parâmetros

x
: vetor ou matriz de reais ou complexos

t
: vetor ou matriz de reais ou complexos

# Descrição

As entradas de `t` são os arcos-senos das entradas correspondentes de `x`. O domínio de deifinição é [-1, 1]. A função toma valores em conjuntos

# Exemplos

```
A=[1,2;3,4]
sin(asin(A))
```

# Ver Também

sin, sinm, asinm

# Name

asind — arco-seno, resultado em graus

```
t=asind(x)
```

# Parâmetros

x
: vetor/matriz de reais. Os elementos devem estar em `[-1 1]`.

t
: vetor/matriz de reais com as mesmas dimensões que `x`

# Descrição

As entradas de `t` são os arcos-inversos das entradas correspondentes de `x`. O domínio de definição é [-1, 1]. Os resultados estão em [-90 90];

# Exemplos

```
x=[-1 0 1 sqrt(2)/2 -sqrt(2)/2 sqrt(3)/2 -sqrt(3)/2];
asind(x)
```

# Ver Também

sin, sind, asinm

# Name

asinh — arco-seno hiperbólico

```
[t]=asinh(x)
```

# Parâmetros

x
: vetor ou matriz de reais ou complexos

t
: vetor ou matriz de reais ou complexos

# Descrição

As entradas de `t` são os arcos-senos hiperbólicos das entradas correspondentes de `x`. O domínio de definição é]-1,infinito[. A função toma valores em conjuntos.

# Exemplos

```
A=[1,2;2,3]
sinh(asinh(A))
```

# Name

asinhm — arco-seno hiperbólico da matriz (matriz arco-seno hiperbólico)

```
t=asinhm(x)
```

# Parâmetros

x,t
: matriz quadrada de reais ou complexos

# Descrição

asinhm é o arco-seno hiperbólico da matriz x (matriz arco-seno hiperbólico de x). Usa a fórmula `t=logm(x+sqrtm(x*x+eye()))`. Os resultados podem ser imprecisos se a matriz é não-simétrica.

# Exemplos

```
A=[1,2;2,3]
sinhm(asinhm(A))
```

# Ver Também

asinh, logm, sqrtm

# Name

asinm — arco-seno da matriz (matriz arco-seno)

```
t=asinm(x)
```

# Parâmetros

x
: matriz quadrada de reais ou complexos

t
: matriz quadrada de reais ou complexos

# Descrição

`t` é o arco-seno da matriz `x` matrix. (matriz arco-seno de x). É usado o método da diagonalização. Para matrizes não-simétricas, o resultado pode ser impreciso.

# Exemplos

```
A=[1,2;3,4]
sinm(asinm(A))
asinm(A)+%i*logm(%i*A+sqrtm(eye()-A*A))
```

# Ver Também

asin, sinm

# Name

atan — arco-tangente 2-quadrantes e 4-quadrantes

```
phi=atan(x)
phi=atan(y,x)
```

# Parâmetros

x
: escalar real ou complexo, ou vetor ou matriz de reais ou complexos

phi
: escalar real ou complexo, ou vetor ou matriz de reais ou complexos

x, y
: escalares real, ou vetores ou matrizes de reais de mesmo tamanho

phi
: escalar real, ou vetor ou matriz de reais

# Descrição

A primeira forma computa o arco-tangente 2-quadrantes, que é o inverso de `tan(phi)`. Para um real `x`, `phi` está no intervalo (-pi/2, pi/2).. Para um complexo `x`, `atan` tem dois pontos de ramificação singulares `+%i`,`-%i` e os ramos escolhidos são as duas semi-retas imaginárias [i, i*oo) e (-i*oo, -i].

A segunda forma computa o arco-tangente 4-quadrantes (atan2 em FORTRAN), isto é, retorna o argumento (ângulo) do número complexo `x+i*y`. A imagem de `atan(y,x)` é (-pi, pi].

Para argumentos reais, ambas as formas produzem valores idênticos se `x>0`.

No caso de os argumentos serem vetores ou matrizes, a avaliação é feita elemento a elemento, de modo que `phi` é um vetor ou matriz com o mesmo tamanho de `phi(i,j)=atan(x(i,j))` ou `phi(i,j)=tan(y(i,j),x(i,j))`.

# Exemplos

```
// exemplos com a segunda forma
x=[1,%i,-1,%i]
phasex=atan(imag(x),real(x))
atan(0,-1)
atan(-%eps,-1)

// ramos
atan(-%eps + 2*%i)
atan(+%eps + 2*%i)
atan(-%eps - 2*%i)
atan(+%eps - 2*%i)

// valores nos pontos de ramificação
ieee(2)
atan(%i)
atan(-%i)
```

# Ver Também

tan, ieee

# Autores

B.P.
L.V.D. (autores da função atan para complexos).

# Name

atand — arcos-tangentes 2-quadrantes e 4-quadrantes elemento a elemento do argumento com resultados em graus

```
phi=atand(x)
phi=atand(y,x)
```

# Parâmetros

x
: escalar, vetor ou matriz de reais

phi
: escalar, vetor ou matriz de reais

x, y
: escalare, vetore ou matrizes de reais com tamanhos iguais

phi
: escalar, vetor ou matriz de reais

# Descrição

A primeira forma computa o arco-tangente 2-quadrantes, que é o inverso de `tand(phi)`. Os elementos de `phi` estão no intervalo [-90, 90].

A primeira forma computa o arco-tangente 4-quadrantes (atan2 em Fortran), isto é, retorna o argumento (ângulo) do número complexo `x+i*y`. A imagem de `atand(y,x)` é [-180,180i].

Ambas as formas são idênticas se `x>0`.

# Exemplos

```
// exemplo com a segunda forma
x=[0,1/sqrt(3),1,sqrt(3),%inf,0]
atand(x)
```

# Ver Também

tan, tand

# Autor

Serge Steer, INRIA

# Name

atanh — arco-tangente hiperbólico

```
t=atanh(x)
```

# Parâmetros

x
: matriz ou vetor de reais ou complexos

t
: matriz ou vetor de reais ou complexos

# Descrição

Os compontente do vetor `t` são os arcos-tangentes hiperbólicos das entradas correspondentes em `x`. O domínio de definição é `[-1,1]` para a função real (ver "Observação").

# Observação

No Scilab (como em alguns outros softwares numéricos), quando se tenta avaliar uma função matemática elementar fora de seu domínio de definição no caso real, uma extensão complexa é usada (com resultado complexo). O exemplo mais famoso é a função sqrt (tente `sqrt(-1)` !). !). Esta aproximação tem algumas desvantagens quando você avalia a função em um ponto singular que pode levar a diferentes resultados quando o ponto é considerado como real ou complexo. Para a função `atanh` isto ocorre para `-1` e `1` porque, nestes pontos, a parte imaginária não converge e então `atanh(1) = +Inf + i NaN` enquanto `atanh(1) = +Inf` para o caso real (como lim x->1- de atanh(x)). Então, quando você avaliar esta função no vetor `[1 2]` , como `2` está fora do domínio de definição, a extensão complexa é usada para todo o vetor e o resultado de `atanh(1) = +Inf + i NaN` enquanto o resultado de `atanh(1) = +Inf` com `[1 0.5]` , por exemplo.

# Exemplos

```
// exemplo 1
x=[0,%i,-%i]
tanh(atanh(x))

// exemplo 2
x = [-%inf -3 -2 -1 0 1 2 3 %inf]
ieee(2)
atanh(tanh(x))

// exemplo 3 (ver "Observação")
ieee(2)
atanh([1 2])
atanh([1 0.5])
```

# Ver Também

tanh, ieee

# Name

atanhm — Arco-tangente hiperbólico da matriz (matriz arco-tangente hiperbólico)

```
t=atanhm(x)
```

# Parâmetros

x
: matriz quadrada de reais ou complexos

t
: matriz quadrada de reais ou complexos

# Descrição

`atanhm(x)` é o arco-tangente hiperbólico da matriz `x`. (matriz arco-tangente hiperbólico). O resultado pode ser impreciso se `x` não é simétrica.

# Exemplos

```
A=[1,2;3,4];
tanhm(atanhm(A))
```

# Ver Também

atanh, tanhm

# Name

atanm — arco-tangente da matriz quadrada (matriz arco-tangente)

```
[t]=atanm(x)
```

# Parâmetros

x
: matriz de reais ou complexos quadrada

t
: matriz de reais ou complexos quadrada

# Descrição

`atanm(x)` é o arco-tangente da matriz x (matrtiz arco-tangente de x). O resultado pode ser impreciso se `x` não é simétrica.

# Exemplos

```
tanm(atanm([1,2;3,4]))
```

# Ver Também

atan

# Name

base2dec — conversão da base b para inteiros

```
d=base2dec(s,b)
```

# Parâmetros

d
: matriz de inteiros

s
: matriz de strings correspondentes às representações em base b

b
: inteiro, a base

# Descrição

`base2dec(x,b)` retorna a matriz de número correspondentes à representação em base b

# Exemplos

```
base2dec(['ABC','0','A'],16)
```

# Ver Também

bin2dec, oct2dec, hex2dec, dec2bin, dec2oct, dec2hex

# Name

bin2dec — conversão de representação binária para inteira

```
y=bin2dec(str)
```

# Parâmetros

str
: um string ou vetor/matriz de strings contendo apenas caracteres '1' e '0'

y
: um escalar ou um vetor/matriz de inteiros positivos

# Descrição

Dado `str` um string binário, está função retorna `y`, o número decimal cuja representação binária é dada por `str` ( `y` e `str` possuem o mesmo tamanho)

# Exemplos

```
// example 1 :
// '1010110' : é a representação binária de 86
str='1010110';
y=bin2dec(str)

// example 2 :
// '1011011' : é a representação binária de 91
// '1010010' : é a representação binária de 82
str=['1011011'; '1010010']
y=bin2dec(str)
```

# Ver Também

base2dec, oct2dec, hex2dec, dec2bin, dec2oct, dec2hex

# Name

binomial — probabilidades de distribuição binomial

```
pr=binomial(p,n)
```

# Parâmetros

pr
: vetor linha com n+1 componentes

p
: número real em [0,1]

n
: um inteiro >= 1

# Descrição

`pr=binomial(p,n)` retorna o vetor de probabilidade binomial, i.e. `pr(k+1)` é a probabilidade de `k` successos em `n` tentativas independentes de Bernoulli com probabilidade de sucesso `p`. Em outras palavras : `pr(k+1) = probability(X=k)` , com X uma variável aleatória, segundo a distribuição B(n,p) , e numericamente :

```
           / n \  k      n-k       / n \       n!
pr(k+1) =  |   | p  (1-p)    com  |   |  = ---------
           \ k /                   \ k /    k! (n-k)!
```

# Exemplos

```
// primeiro exemplo
n=10;p=0.3; xbasc(); plot2d3(0:n,binomial(p,n));

// segundo exemplo
n=50;p=0.4;
mea=n*p; sigma=sqrt(n*p*(1-p));
x=( (0:n)-mea )/sigma;
xbasc()
plot2d(x, sigma*binomial(p,n));
deff('y=Gauss(x)','y=1/sqrt(2*%pi)*exp(-(x.^2)/2)')
plot2d(x, Gauss(x), style=2);

// pela fórmula binomial (cuidado, se n for grande)
function pr=binomial2(p,n)
x=poly(0,'x');pr=coeff((1-p+x)^n).*horner(x^(0:n),p);
endfunction
p=1/3;n=5;
binomial(p,n)-binomial2(p,n)

// pela função gama: gamma(n+1)=n! (cuidado, se n for grande)
p=1/3;n=5;
Cnks=gamma(n+1)./(gamma(1:n+1).*gamma(n+1:-1:1));
x=poly(0,'x');
pr=Cnks.*horner(x.^(0:n).*(1-x)^(n:-1:0),p);
```

```
pr-binomial(p,n)
```

## Ver Também

cdfbin, grand

# Name

bitand — AND ("e") aplicado às representações binárias dos argumentos de entrada

```
[z]=bitand(x,y)
```

# Parâmetros

x
: escalar/vetor/matriz/hipermatriz de inteiros positivos

y
: escalar/vetor/matriz/hipermatriz de inteiros positivos

z
: escalar/vetor/matriz/hipermatriz de inteiros positivos

# Descrição

Dado `x` e `y` dois inteiros positivos, esta função retorna `z` o número decimal cuja forma binária é o AND das representações binárias de `x` e `y`. (`x`, `y` e `z` devem ter o mesmo tamanho). Se a dimensão de `x` (e `y`) é superior a 1, então `z(i)` é igual a `bitand(x(i),y(i))`

# Exemplos

```
// exemplo 1 :
// '1010110' : é a representação binária de 86
// '1011011' : é a representação binária de 91
// '1010010' : é a representação binária para AND aplicado às formas binárias d
// então, o número decimal correspondente a AND aplicado às formas binárias de 
x=86; y=91
z=bitand(x,y)

// exemplo 2 :
x=[12,45],y=[25,49]
z=bitand(x,y)
```

# Ver Também

bitor, bin2dec, dec2bin

# Name

bitor — OR ("ou") aplicado às representações binárias dos argumentos de entrada

```
[z]=bitor(x,y)
```

# Parâmetros

x
: escalar/vetor/matriz/hipermatriz de inteiros positivos

y
: escalar/vetor/matriz/hipermatriz de inteiros positivos

z
: escalar/vetor/matriz/hipermatriz de inteiros positivos

# Descrição

Dado `x` e `y` dois inteiros positivos, esta função retorna `z` o número decimal cuja forma binária é o OR das representações binárias de `x` e `y` (`x`, `y` e `z` devem ter o mesmo tamanho). Se a dimensão de `x` é superior a 1, então `z(i)` é igual a `bitor(x(i),y(i))`

# Exemplos

```
// exemplo 1 :
// '110000' : é a representação binária de 48
// '100101' : é a representação binária de 37
// '110101' : é a representação binária para OR aplicado às formas binárias de 4
// então, o número decimal correspondente a OR aplicado às formas binárias de 48
x=48; y=37
z=bitor(x,y)

// exemplo 2 :
x=[12,45]; y=[25,49]
z=bitor(x,y)
```

# Ver Também

bitand, bin2dec, dec2bin

# Name

bloc2exp — retorna a avaliação simbólica de um diagrama de blocos

```
[str]=bloc2exp(blocd)
[str,names]=bloc2exp(blocd)
```

# Parâmetros

blocd
: lista

str
: string

names
: string

# Descrição

Dada uma representação em diagrama de blocos de um sistema linear `bloc2exp` retorna sua avaliação simbólica. O primeiro elemento da lista `blocd` deve ser o string (cadeia de caracteres) `'blocd'`. Cada outro elemento desta lista `(blocd(2),blocd(3),...)` é, por si mesmo, uma lista de um dos seguintes tipos :

```
list('transfer','nome_do_sistema_linear')
```

```
list('link','nome_da_ligação',
               [número_de_caixa_de_fluxo_superior,porta_da_caixa_de_fluxo_super
               [caixa_de_fluxo_inferior_1,número_de_porta_da_caixa_inferior_1],
               [caixa_de_fluxo_inferior_2,número_de_porta_da_caixa_inferior_2],
               ...)
```

Os strings `'transfer'` e `'links'` são palavras-chaves as quais indicam o tipo de elemento no diagrama de blocos ("transfer"=transferência, "link"= ligação).

Caso 1: o segundo parâmetro da lista é um string que pode se referir (para uma possível avaliação posterior) ao nome Scilab de um sistema linear dado em representação de estado-espaço (lista `sys-lin` ) ou em forma de trasnferência (matriz de razões de polinômios).

A cada bloco de transferência é associado um inteiro. A cada entrada e saída de um bloco de transferência, também é associado seu número, um inteiro (ver exemplos).

Caso 2: O segundo tipo de elemento em uma representação de diagrama de blocos é um link. Um link liga uma saída de um bloco representado pelo par `[número_de_caixa_de_fluxo_superior,porta_da_caixa_de_fluxo_superior]`, a diferentes entradas de outros blocos. Cada tal entrada é representada pelo par [caixa_de_fluxo_inferior_i,número_de_porta_da_caixa_inferior_i] .

Os diferentes elementos de um diagrama de blocos podem ser definidos em ordem arbitrária.

Por exemplo:

[1] `S1*S2` com feedback (resposta) de unidade.

Há 3 transferências `S1` (número `n_s1=2`) , `S2` (número `n_s2=3`) e um adicionador ("adder") (número `n_add=4`) com função de transferência simbólica `['1','1']`.

Há 4 ligações. O primeiro (de nome `'U'`) liga a entrada (porta 0 de bloco fictício -1, omitido) à porta 1 do adicionador. Os segundo e terceiro ligam respectivamente a porta (de saída) 1 do adicionador à porta (de entrada) 1 do sistema `S1`, e a porta (de saída) 1 de `S1` à porta (de entrada) 1 de `S2`. A quarta ligação (de nome `'Y'`) liga a porta (de saída) 1 de `S2` à saída (porta 0 do bloco fictício -1, omitido) e à porta (de entrada) 2 do adicionador.

```
//Inicialização
syst=list('blocd'); l=1;
//
//Sistemas
l=l+1;n_s1=l;syst(l)=list('transfer','S1');  //Sistema 1
l=l+1;n_s2=l;syst(l)=list('transfer','S2');  //Sistema 2
l=l+1;n_adder=l;syst(l)=list('transfer',['1','1']);  //adicionador
//
//Links
// Entradas  -1 --> entrada 1
l=l+1;syst(l)=list('link','U',[-1],[n_adder,1]);
// Internal
l=l+1;syst(l)=list('link',' ',[n_adder,1],[n_s1,1]);
l=l+1;syst(l)=list('link',' ',[n_s1,1],[n_s2,1]);
// Saídas // -1 -> saída 1
l=l+1;syst(l)=list('link','Y',[n_s2,1],[-1],[n_adder,2]);
//Chamada de avaliação
w=bloc2exp(syst);
```

O resultado é o string: `w=-(s2*s1-eye())\s2*s1`.

Perceba que, chamado com dois agumentos de saída, `[str,names]= blocd(syst)` retorna em `names` a lista de nomes simbólicos dos links nomeados. Isto é útil para definir nomes de entradas e saídas.

[2] segundo exemplo

```
//Inicialização
syst=list('blocd'); l=1;

//Sistema (planta de blocos 2x2)
l=l+1;n_s=l;syst(l)=list('transfer',['P11','P12';'P21','P22']);

//Controlador
l=l+1;n_k=l;syst(l)=list('transfer','k');

//Links
l=l+1;syst(l)=list('link','w',[-1],[n_s,1]);
l=l+1;syst(l)=list('link','z',[n_s,1],[-1]);
l=l+1;syst(l)=list('link','u',[n_k,1],[n_s,2]);
l=l+1;syst(l)=list('link','y',[n_s,2],[n_k,1]);

//Chamada de avaliação
w=bloc2exp(syst);
```

Neste caso, o resultado é uma fórmula equivalente à usual:

```
P11+P12*invr(eye()-K*P22)*K*P21;
```

# Ver Também

bloc2ss

# Autores

S. S., F. D. (INRIA)

# Name

bloc2ss — conversão de diagrama de blocos para espaço de estados

```
[sl]=bloc2ss(blocd)
```

# Parâmetros

blocd
: lista

sl
: lista

# Descrição

Dada uma representação em diagrama de blocos de um sistema linear, `bloc2ss` converte esta representação em um sistema linear de espaço de estados. O primeiro elemento da lista `blocd` deve ser o string (cadeia de caracteres) `'blocd'`. Cada outro elemento desta lista é, por si mesmo, uma lista de um dos seguintes tipos :

```
list('transfer','nome_do_sistema_linear')
```

```
list('link','nome_do_link', [número_de_caixa_de_fluxo_superior,porta_da_caixa_d
               [caixa_de_fluxo_inferior_1,número_de_porta_da_caixa_inferior_1],
               [caixa_de_fluxo_inferior_2,número_de_porta_da_caixa_inferior_2],
               ...)
```

Os strings `'transfer'` e `'links'` são palavras-chaves as quais indicam o tipo de elemento no diagrama de blocos ("transfer"=transferência, "link"= ligação).

Caso 1: o segundo parâmetro da lista é um string que pode se referir (para uma possível avaliação posterior) ao nome Scilab de um sistema linear dado em representação de espaço de estados (lista `syslin`) ou em forma de trasnferência (matriz de razões de polinômios).

A cada bloco de transferência é associado um inteiro. A cada entrada e saida de um bloco de transferência, também é associado seu número, um inteiro (ver exemplos).

Caso 2: O segundo tipo de elemento em uma representação de diagrama de blocos é um link. Um link liga uma saída de um bloco representado pelo par `[número_de_caixa_de_fluxo_superior,porta_da_caixa_de_fluxo_superior]`, a diferentes entradas de outros blocos. Cada tal entrada é representada pelo par [caixa_de_fluxo_inferior_i,número_de_porta_da_caixa_inferior_i] .

Os diferentes elementos de um diagrama de blocos podem ser definidos em ordem arbitrária.

Por exmeplo:

[1] `S1*S2` com feedback (resposta) de unidade.

Há 3 transferências `S1` (número `n_s1=2`) , `S2` (número `n_s2=3`) e um adicionador ("adder") (número `n_add=4`) com função de transferência simbólica `['1','1']`.

Há 4 links. O primeiro (de nome `'U'`) liga a entrada (porta 0 de bloco fictício -1, omitido) à porta 1 do adicionador. Os segundo e terceiro ligam respectivamente a porta (de saída) 1 do adicionador à

porta (de entrada) 1 do sistema `S1`, e a porta (de saída) 1 de `S1` à porta (de entrada) 1 de `S2`. O quarto link (de nome `'Y'`) liga a porta (de saída) 1 de `S2` à saída (porta 0 do bloco fictício -1, omitido) e à porta (de entrada) 2 do adicionador.

```
//Inicialização
syst=list('blocd'); l=1;

//Sistemas
l=l+1;n_s1=l;syst(l)=list('transfer','S1');  //Sistema 1
l=l+1;n_s2=l;syst(l)=list('transfer','S2');  //Sistema 2
l=l+1;n_adder=l;syst(l)=list('transfer',['1','1']);  //adicionador

//Links
// Entradas  -1 --> entrada 1
l=l+1;syst(l)=list('link','U1',[-1],[n_adder,1]);

// Internal
l=l+1;syst(l)=list('link',' ',[n_adder,1],[n_s1,1]);
l=l+1;syst(l)=list('link',' ',[n_s1,1],[n_s2,1]);

// Saídas // -1 -> saída 1
l=l+1;syst(l)=list('link','Y',[n_s2,1],[-1],[n_adder,2]);
```

Com `s=poly(0,'s');S1=1/(s+1);S2=1/s;` o resultado da avaliação de chamada `sl=bloc2ss(syst);` é uma representação em estado-espaço para `1/(s^2+s-1)`.

[2] exemplo da transformação linear fracional (TLF e, em inglês LFT):

```
//Inicialização
syst=list('blocd'); l=1;

//Sistema (planta de blocos 2x2)
l=l+1;n_s=l;syst(l)=list('transfer',['P11','P12';'P21','P22']);

//Controlador
l=l+1;n_k=l;syst(l)=list('transfer','k');

//Links
l=l+1;syst(l)=list('link','w',[-1],[n_s,1]);
l=l+1;syst(l)=list('link','z',[n_s,1],[-1]);
l=l+1;syst(l)=list('link','u',[n_k,1],[n_s,2]);
l=l+1;syst(l)=list('link','y',[n_s,2],[n_k,1]);
```

Com

```
P=syslin('c',A,B,C,D);
P11=P(1,1);
P12=P(1,2);
P21=P(2,1);
P22=P(2,2);
K=syslin('c',Ak,Bk,Ck,Dk);
```

`bloc2exp(syst)` retorna a avaliação de TLF de `P` e `K`.

# Ver Também

bloc2exp

# Autores

S. S., F. D. (INRIA)

# Name

cat — concatena vários arrays

```
y=cat(dims,A1,A2,...,An)
```

# Parâmetros

dims
: um escalar real positivo

A1,A2,..An
: escalares, vetores, matrizes, multi-arrays ou cell arrays. A1, A2, ..., An, devem ter o mesmo tamanho (excluindo o número de dimensão dims). size(A1,i)=size(A2,i)=...=size(An,i) para i diferente de dims e size(A1,dims), size(A2,dims),...,size(An,dims) podem ser diferentes

y
: um escalar, vetor, matriz ou multi-array, y tem o mesmo tipo de A1,A2,...,An

# Descrição

`y=cat(dims,A1,A2,...,An) :` y é o resutado da concatenação dos argumentos de entrada A1, A2, ..., An. Se dims= 1, então a concatenação é feita de acordo com as linhas; se dims= 2, então a concatenação é feita de acordo com as colunas dos argumentos de entrada.

Se dims= 1,então a concatenação é feita de acordo com as linhas

A1=[1 2 3 ; 4 5 6]; A2=[7 8 9 ; 10 11 12]; y=cat(1,A1,A2) => y=[1 2 3 ; 4 5 61 ;7 8 9; 10 11 12]

Se dims= 2,então a concatenação é feita de acordo com as colunas dos argumentos de entrada

A1=[1 2 3;4 5 6]; A2=[7 8 9 ;10 11 12]; y=cat(2,A1,A2) => y=[1 2 3 7 8 9 ; 4 5 6 10 11 12]

# Exemplos

```
// primeiro exemplo: concatenação de acordo com as linhas
dims=1; A1=[1 2 3]; A2=[4 5 6 ; 7 8 9]; A3=[10 11 12]; y=cat(dims,A1,A2,A3)

// segundo exemplo: concatenação de acordo com as colunas
dims=2; A1=[1 2 3]'; A2=[4 5;7 8;9 10]; y=cat(dims,A1,A2)

// terceiro exemplo concatenação de acordo com a terceira dimensão
dims=3; A1=matrix(1:12,[2,2,3]); A2=[13 14;15 16]; A3=matrix(21:36,[2,2,4]); y=
```

# Ver Também

permute, matrix

# Autor

Farid Belahcene

# Name

ceil — arredonda o número para o inteiro maior ou igual ao número

```
[y]=ceil(x)
```

# Parâmetros

x
: uma matriz de reais

y
: uma matriz de inteiros

# Descrição

`ceil(x)` retorna uma matriz de inteiros feita de "arredondamentos para cima" dos elementos.

# Exemplos

```
ceil([1.9 -2.5])-[2,-2]
ceil(-%inf)
x=rand()*10^20;ceil(x)-x
```

# Ver Também

round, floor, int

# Name

cell2mat — converte um cell array (array de células) em uma matriz

```
x=cell2mat(c)
```

# Parâmetros

c
: cell, os componentes de c devem ter o mesmo tipo e podem ser escalares ou matrizes

x
: matriz

# Descrição

Retorna uma matriz que é a concatenação de todos os componentes do cell (célula) c.

cell2mat(c)
: todos os componentes de c devem ter o mesmo tipo de dado (string, doubles, inteiros ou booleanos). Para cada linha `i` de `c, cell2mat` concatena todos os componentes da i-ésima linha do cell `c`

Perceba que se os componentes da entrada do cell c são são strings, então `cell2mat(c)` retorna um vetor-coluna de concatenação de strings.

# Exemplos

```
c=makecell([2,2],[1 2 3; 6 7 8],[4 5;9 10],[11 12;16 17],[14 13 15;18 19 20])
cell2mat(c)
```

# Ver Também

cell

# Name

cellstr — converte um vetor (ou matriz) de strings em um cell de strings

```
c=cellstr(s)
```

# Parâmetros

s
: vetor de strings ou matriz de strings

# Descrição

Retorna um cell array de strings

- Se `s` ié um vetor linha de strings, então `cellstr(s)` retorna um cell (um-por-um) que contém um componente (a concatenação de todas as colunas componentes de `s` ).
- Se `s` é um vetor coluna de strings, então `cellstr(s)` converte s em um cell que tem o mesmo tamanho : cell de strings `(size(s,1)-por-um)`.
- Se `s` é uma matriz de strings, então, para cada linha `i` de `s`, `cellstr(s)` concatena todos os componentes da i-ésima linha da matriz `s` (i.e s(i,1), s(i,2), s(i,3),...) e retorna um cell de strings (`size(s,1)-por-um)`.

# Exemplos

```
cellstr("foo")
cellstr(["sci","lab"])
cellstr(["abc","def",'gh';"i","j","klm"])
```

# Ver Também

cell, string

# Name

char — função char

```
y=char( x)
y=char(st1,st2,st3,....)
```

# Parâmetros

x
: um cell de arrays de strings, ou um array de códigos ASCII

st1,st2,st3
: arrays de strings

y:
: vetor (coluna) de strings

# Descrição

`Argumento de uma entrada :`

Dado um `cell de arrays de strings` `x`, esta função retorna um vetor de strings `y` no qual as linhas são componentes do cell de strings.

Dado um `array de códigos ASCII` `x`, esta função retorna um array de strings `y` correspondente aos códigos ASCII. Se dims (`x`)=[n1,n2,n3,n4,....], então o valor retornado tem o mesmo tamanho que o valor de entrada, ao invés do segundo dims, dims(`y`)=[n1,n3,n4,..]

`Argumento de mais de uma entrada :`

Dados `arrays de strings` `st1,st2,st3,...`, esta função retorna um vetor de strings no qual as linhas são os componentes de `st1,st2,st3,...` `No vetor y` o comprimento de todos os strings sti é completado por lacunas, para terem o mesmo comprimento que o comprimento máximo de sti.

# Exemplos

```
//exemplo com uma hipermatriz de códigos ASCII :
x=hypermat([4,2,3],61:84);
y=char(x)
size(x)
size(y)

//Exemplo com mais de um argumento :
st1="zeros";
st2=["one","two"];
st3=["three"];
y=char(st1,st2,st3)
size(y)
//todos os strings são completados por "lacunas" para terem o mesmo comprimento
length(y)
```

# Ver Também

asciimat

# Autor

F.B

# Name

conj — matriz conjugada

```
[y]=conj(x)
```

# Parâmetros

x,y
: matrizes de reais ou complexos

# Descrição

`conj(x)` é a matriz conjugada complexa de `x`.

# Exemplos

```
x=[1+%i,-%i;%i,2*%i];
conj(x)
x'-conj(x)  //x'  é a matriz transposta conjugada
```

# Name

cos — função co-seno

```
[y]=cos(x)
```

# Parâmetros

x
: matriz ou vetor de reais ou complexos

# Descrição

Para uma matriz ou vetor, `cos(x)` é o co-seno de seus elementos . Para o co-seno da matriz (matriz co-seno), use a função `cosm(X)` .

# Exemplos

```
x=[0,1,%i]
acos(cos(x))
```

# Ver Também

cosm

# Name

cosd — função co-seno elemento a elemento, argumento dado em graus

```
y=cosd(x)
```

# Parâmetros

x
: vetor/matriz de reais

# Descrição

Para um vetor ou uma matriz `x` de ângulos dados em graus, `cosd(x)` é o co-seno de seus elementos. Os resultados estão em `[-1 1]`. Para elementos de entrada que são iguais a `n*90` com `n` inteiro e ímpar, o resultado é exatamento zero.

# Exemplos

```
x=[0,30 45 60 90 360];
cosd(x)
```

# Ver Também

cos

# Autor

Serge Steer, INRIA

# Name

cosh — co-seno hiperbólico

```
[t]=cosh(x)
```

# Parâmetros

x,t
: matriz ou vetor de reais ou complexos

# Descrição

Os elementos de `t` são os co-senos hiperbólicos da entradas correspondentes do vetor `x`.

# Exemplos

```
x=[0,1,%i]
acosh(cosh(x))
```

# Ver Também

cos, acosh

# Name

coshm — co-seno hiperbólico da matriz (matriz co-seno hiperbólico)

```
t=coshm(x)
```

# Parâmetros

x,t
: matriz quadrada de reais ou complexos

# Descrição

coshm é o co-seno hiperbólico da matriz x (matriz co-seno hiperbólico de x). `t=(expm(x)+expm(-x))/2`. O resultado pode ser impreciso para matrizes não-simétricas.

# Exemplos

```
A=[1,2;2,4]
acoshm(coshm(A))
```

# Ver Também

cosh, expm

# Name

cosm — co-seno da matriz (matriz co-seno)

```
t=cosm(x)
```

# Parâmetros

x
: matriz quadrada de reais ou complexos

# Descrição

`cosm(x)` é o co-seno da matriz x (matriz co-seno de x). `t=0.5*(expm(%i*x)+expm(-%i*x))`.

# Exemplos

```
A=[1,2;3,4]
cosm(A)-0.5*(expm(%i*A)+expm(-%i*A))
```

# Ver Também

cos, expm

# Name

cotd — cotangent elemento a elemento do argumento dado em graus

```
y=cotd(x)
```

# Parâmetros

x
: array de reais

y
: array de reais com as mesmas dimensões que `x`.

# Descrição

As entradas `y` são as cotangentes das entradas correspondentes de `x` supostamente dadas em graus. `t=cos(x)./sin(x)`. Para entradas iguais a `n*180` com `n` inteiro, os resultados são infinitos, enquanto que com `cotg(n*%pi)` o resultado é grande, pois `%pi` não pode ser representado exatamente. para entradas iguais a `n*90` com `n` inteiro e ímpar, o resultado é exatamente 0.

# Exemplos

```
x=[30 45 60 90];
cotd(x)
```

# Ver Também

cotg

# Autor

Serge Steer, INRIA

# Name

cotg — cotangente

```
[t]=cotg(x)
```

# Parâmetros

x,t
: vetores ou matrizes de reais ou complexos

# Descrição

Os elementos de `t` são as cotangentes das entradas correspondentes de `x`. `t=cos(x)./sin(x)`

# Exemplos

```
x=[1,%i];
cotg(x)-cos(x)./sin(x)
```

# Ver Também

tan

# Name

coth — cotangente hiperbólica

```
[t]=coth(x)
```

# Descrição

Os elementos do vetor `t` são as cotangentes hiperbólicas dos elementos do vetor `x`.

# Exemplos

```
x=[1,2*%i]
t=exp(x);
(t-ones(x)./t).\(t+ones(x)./t)
coth(x)
```

# Ver Também

cotg

# Name

cothm — cotangente hiperbólica da matriz (matriz cotangente hiperbólica)

```
[t]=cothm(x)
```

# Descrição

`cothm(x)` é a (matriz) contangente hiperbólica da matriz quadrada x.

# Exemplos

```
A=[1,2;3,4];
cothm(A)
```

# Ver Também

coth

# Name

csc — computa a cossecante do argumento elemento a elemento

```
y = csc(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos com as mesmas dimensões que `x`.

# Description

Computa a cossecante do argumento elemento a elemento. A função cossecante é uma função periódica definida como 1/sin. Para dados reais, os resultados são reais em `]-%inf -1] U [1 %inf[`.

# Exemplos

```
x=linspace(0.01,%pi-0.01,200)
clf();plot(-x,csc(-x),x,csc(x))
```

# Ver Também

sec, cscd

# Autor

Serge Steer, INRIA

# Name

cscd — computa a cossecante elemento a elemento do argumento em graus

```
x = cscd(x)
```

# Parâmetros

x
: array de reais ou complexos

x
: array de reais ou complexos

# Descrição

As emtradas de `y` são as cossecantes `1/sin` das entradas de `x` dadas em graus. Os resultados são reais em `]-%inf -1] U [1 %inf[`. Para entradas iguais a `n*180` com `n` inteiro, o resultado é infinito (ou um erro dependendo do modo ieee). Para entradas iguais a `n*90` com `n` inteiro e ímpar, o resultado é exatamente 1 ou -1.

# Exemplos

```
csc(%pi/4)
cscd(90)
```

# Ver Também

secd, csc, sind

# Autor

Serge Steer, INRIA

# Name

csch — computa a cossecante hipertbólica do argumento elemento a elemento

```
y = csch(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos com as mesmas dimensões que `x`.

# Descrição

Computa a cossecante hipertbólica do argumento elemento a elemento. Para dados reais, os resultados são reais

# Exemplos

```
x=linspace(0.01,4,200);x=[-x($:-1:1) %nan x];
clf();plot(x,csch(x))
```

# Ver Também

csc, acsch

# Autor

Serge Steer, INRIA

# Name

cumprod — produto cumulativo

```
y=cumprod(x)
y=cumprod(x,'r') ou y=cumprod(x,1)
y=cumprod(x,'c') ou y=cumprod(x,2)
y=cumprod(x,'m')
```

# Parâmetros

x
: vetor ou matriz de reais ou complexos

y
: vetor ou matriz de reais ou complexos

# Descrição

Para um vetor ou matriz `x`, `y=cumprod(x)` retorna em `y` o produto cumulativo de todas as entradas de `x` tomadas coluna a coluna.

`y=cumprod(x,'c')` (ou, equivalentemente, `y=cumprod(x,2)`) retorna em `y` o produto cumulativo dos elementos das colunas de `x`: `y(i,:)=cumprod(x(i,:))`

`y=cumprod(x,'r')` (ou, equivalentemente, `y=cumprod(x,2)`) retorna em `y` o produto cumulativo dos elementos das linhas de `x`: `y(:,i)=cumprod(x(:,i))`.

`y=cumprod(x,'m')` é o produto cumulativo ao longo da primeira dimensão não-singleton de `x` (para compatibilidade com Matlab).

# Exemplos

```
A=[1,2;3,4];
cumprod(A)
cumprod(A,'r')
cumprod(A,'c')
rand('seed',0);
a=rand(3,4);
[m,n]=size(a);
w=zeros(a);
w(1,:)=a(1,:);
for k=2:m;w(k,:)=w(k-1,:).*a(k,:);end;w-cumprod(a,'r')
```

# Ver Também

cumsum, sum, prod

# Name

cumsum — soma cumulativa

```
y=cumsum(x)
y=cumsum(x,'r') or y=cumsum(x,1)
y=cumsum(x,'c') or y=cumsum(x,2)
```

# Parâmetros

x
: vetor ou matrix (de reais ou complexos)

y
: vetor ou matrix (de reais ou complexos)

# Descrição

Para um vetor ou uma matriz `x`, `y=cumsum(x)` retorna em `y` a soma cumulativa de todas as entradas de `x` tomadas coluna a coluna.

`y=cumsum(x,'c')` (ou, equivalentemente, `y=cumsum(x,2)`) retorna em `y` a soma cumulativa das colunas de `x`: `y(i,:)=cumsum(x(i,:))`

`y=cumsum(x,'r')` (ou, equivalentemente, `y=cumsum(x,1)`) retorna em `y` a soma cumulativa das linhas de `x`: `y(:,i)=cumsum(x(:,i))`

`y=cumsum(x,'m')` é a soma cumulativa ao longo da primeira dimensão "não-singleton" de `x` (para compatibilidade com o Matlab).

# Exemplos

```
A=[1,2;3,4];
cumsum(A)
cumsum(A,'r')
cumsum(A,'c')
a=rand(3,4)+%i;
[m,n]=size(a);
w=zeros(a);
w(1,:)=a(1,:);
for k=2:m;w(k,:)=w(k-1,:)+a(k,:);end;w-cumsum(a,'r')
```

# Ver Também

cumprod, sum

# Name

dec2bin — representação binária

```
[str]=dec2bin(x[,n])
```

# Parâmetros

x
: escalar/vetor/matriz/hypermatriz de inteiros positivos

n
: um inteiro positivo

str
: um string ou vetor de strings

# Descrição

Dado `x`, um inteiro (ou um vetor/matriz de inteiros) positivo, esta função retorna um string (ou um vetor coluna de strings) que é a representação binária de `x.` Se a dimensão de `x` é superior a 1, então cada componente do vetor coluna `str` é a representação binária dos componentes de `x` (i.e `str(i)` é a representação binária de `x(i)`). Se o comprimento dos componentes de `str` é menor que `n` ( i.e length str(i) < n ), então, adicione aos componentes de `str` os caracteres '0' à esquerda, para se obter o comprimento dos componentes igual a `n`

# Exemplos

```
// exemplo 1 :
x=86;
str=dec2bin(x)

// exemplo 2 :
// a representação binária de 86 é: '1010110'
// seu comprimento é 7(menor que n), então, temos adicionamos a str,
//8 vezes o caracteres '0'  (à esquerda)
x=86;n=15;
str=dec2bin(x,n)

// exemplo 3 :
x=[12;45;135]
z=dec2bin(x)
```

# Ver Também

base2dec, bin2dec, oct2dec, hex2dec, dec2oct, dec2hex

# Name

dec2hex — representação hexadecimal de inteiros

```
h=dec2hex(d)
```

# Parâmetros

d
: matriz de inteiros não-negativos

h
: matriz de strings

# Descrição

`dec2hex(x)` retorna a representação hexadecimal de uma matriz de inteiros.

# Exemplos

```
dec2hex([2748 10;11 3])
```

# Ver Também

base2dec, bin2dec, oct2dec, hex2dec, dec2bin, dec2oct

# Name

dec2oct — representação octal de inteiros

```
o=dec2oct(d)
```

# Parâmetros

d
: matriz de inteiros não-negativos

o
: matriz de strings

# Descrição

`dec2oct(x)` retorna a representação octal de uma matriz de inteiros.

# Exemplos

```
dec2oct([2748 10;11 3])
```

# Ver Também

base2dec, bin2dec, oct2dec, hex2dec, dec2bin, dec2hex

# Name

delip — Integral elíptica

```
[r]=delip(x,ck)
```

# Parâmetros

x
: vetor real com elementos não-negativos

ck
: número real entre -1 e 1

r
: número real ou complexo ou vetor de reais ou complexos com mesmo tamanho que `x`

# Descrição

A integral elíptica de primeira espécie com parâmetro `ck` definido como segue:

$$\int_0^x \frac{dt}{\sqrt{(1-t^2)(1-c_k^2 t^2)}}$$

Onde `x` é real e positivo e `ck` está em `[-1 1]`.

Se x é menor do que ou igual a 1, o resultado é real.

Quando chamado com `x`, um vetor real `r` é avaliado para cada entrada de `x`.

# Exemplos

```
ck=0.5;
delip([1,2],ck)
deff('y=f(t)','y=1/sqrt((1-t^2)*(1-ck^2*t^2))')
intg(0,1,f)    //OK, desde que a solução seja real!
```

# Ver Também

amell, %asn, %sn

# Name

diag — inclusão ou extração diagonal

```
[y]=diag(vm, [k])
```

# Parâmetros

vm
: vetor ou matriz (armazenamento cheio ou esparso)

k
: inteiro (o valor padrão é 0)

y
: vetor ou matriz

# Descrição

Para um n-vetor (linha ou coluna) `vm` , `diag(vm)` retorna uma matriz diagonal com entradas de `vm` ao longo da diagonal principal.

`diag(vm,k)` é uma matriz `(n+abs(k))x(n+abs(k))` com entradas de `vm` ao longo da k-ésima diagonal. `k=0` é a diagonal principal, `k>0` são as diagonais superiores e `k<0` são as diagonais inferiores.

Para uma matriz `vm`, `diag(vm,k)` é o vetor coluna feito das entradas da k-ésima diagonal de `vm`. `diag(vm)` é a diagonal principal de `vm`. `diag(diag(x))` é a matriz diagonal.

Se `vm` é uma matriz esparsa, `diag(vm,k)` retorna uma matriz esparsa.

Para construir um sistema linear diagonal, use `sysdiag`.

Perceba que `eye(A).*A` retorna uma matriz diagonal feita das entradas diagonais de `A`. Isto é válido para qualquer matriz (constante, de polinômios, de razões de polinômios, sistema linear em espaço de estados,...).

# Exemplos

```
diag([1,2])

A=[1,2;3,4];
diag(A)  // diagonal principal
diag(A,1)

diag(sparse(1:10))  // matriz diagonal esparsa

// forma de uma matriz triadiagonal de tamanho 2*m+1
m=5;diag(-m:m) +  diag(ones(2*m,1),1) +diag(ones(2*m,1),-1)
```

# Ver Também

sysdiag, sparse

# Name

diff — diferença e derivada discreta

```
y=diff(x)
y=diff(x [,n [,dim]])
```

# Parâmetros

x
: vetor ou matriz (de reais, complexos, esparso(a) ou de polinômios)

n
: inteiro, ordem da diferenciação

dim
: inteiro ou cadeia de caracteres com valores "r","c" e "*"

y
: escalar ou vetor

# Descrição

`y=diff(x)` computa a função de diferença y=x(2:$)-x(1:$-1)

`diff(x,n,dim)` é a n-ésima função de diferença ao longo da dimensão `dim`.

`diff(x,n,"*")` é equivalente a diff(x(:),n).

O valor padrão para `n` é 1. O valor padrão para `dim` é `"*"`.

`dim='r'` é equivalente a `dim=1` e `dim='c'` é equivalente a `dim=2`.

# Exemplos

```
v=(1:8)^3;
diff(v)
diff(v,3)

A=[(1:8)^2
   (1:8)^3
   (1:8)^4];

diff(A,3,2)

//diferenciação aproximada
step=0.001
t=0:step:10;
y=sin(t);
dy=diff(sin(t))/step; //diferenciação aproximada da função seno
norm(dy-cos(t(1:$-1)),%inf)
```

# Name

double — conversão de inteiro para representação de dupla precisão

```
y=double(X)
y=int16(X)
y=int32(X)
y=uint8(X)
y=uint16(X)
y=uint32(X)
```

# Parâmetros

X
: matriz de números em ponto flutuante ou inteiros

y
: matriz de números em ponto flutuante

# Descrição

Converte os dados armazenados usando inteiros de 1, 2 ou 4 bytes em representação de ponto flutuante de dupla precisão. Se as entradas de X já são números de ponto flutuante de dupla precisão, nada é feito .

# Exemplos

```
x=int8([0 12 140])
double(x)
```

# Ver Também

int8, inttype, type

# Name

dsearch — Busca binária (também conhecida como busca dicótoma em francês)

```
[ind, occ, info] = dsearch(X, val [, ch ])
```

# Parâmetros

X
: um vetor ou matriz de reais

val
: um vetor (linha ou coluna) de reais com n componentes em ordem estritamente crescente val(1) < val(2) < ... < val(n)

ch
: (opcional) um caractere "c" ou "d" (valor padrão "c")

ind
: um vetor ou matriz de reais com as mesmas dimensões que X

occ
: um vetor de reais com o mesmo formato que val (mas com n-1 componentes no caso em que ch="c")

info
: inteiro

# Descrição

Esta função é útil para encontrar em uma tabela ("table") ordenada e/ou contar o número de componentes de um vetor caindo em algumas classes (sendo uma classe um intervalo ou um valor).

Por padrão ou quando `ch="c"`, temos o caso de intervalos, isto é, para cada X(i), busca em qual dos n-1 intervalos o componente do vetor cai, sendo os intervalos definidos por:

```
I1 = [val(1), val(2)]
Ik = (val(k), val(k+1)] for 1 < k <= n-1 ;
```

e:

ind(i)
: é o número do intervalo de X(i) (0 se X(i) não está em [val(1),val(n)])

occ(k)
: é o número de componentes de X que estão em Ik

info
: é o número de componentes de X que não estão em [val(1),val(n)]

Quando `ch="d"`, temos o caso discreto, isto é, para cada X(i) busca se é igual a um valor val(k) e:

ind(i)
: é igual ao índice do componente de val que corresponde a X(i) (ind(i) = k se X(i)=val(k)) ou 0 se X(i) não está em val.

occ(k)
: é igual ao número de componentes de X iguais a val(k)

info
: é o número de componentes de X que não estão no conjunto {val(1),...,val(n)}

# Exemplos

```
// exemplo #1 (estatística elementar para U(0,1))
m = 50000 ; n = 10;
X = grand(m,1,"def");
val = linspace(0,1,n+1)';
[ind, occ] = dsearch(X, val);
xbasc() ; plot2d2(val, [occ/m;0])  // sem normalização: y deve ser próximo de 1

// exemplo #2 (estatística elementar para B(N,p))
N = 8 ; p = 0.5; m = 50000;
X = grand(m,1,"bin",N,p); val = (0:N)';
[ind, occ] = dsearch(X, val, "d");
Pexp = occ/m; Pexa = binomial(p,N);
xbasc() ; hm = 1.1*max(max(Pexa),max(Pexp));
plot2d3([val val+0.1], [Pexa' Pexp],[1 2],"111",  ...
        "Pexact@Pexp", [-1 0 N+1 hm],[0 N+2 0 6])
xtitle(  "Distribuição binomial B("+string(N)+","+string(p)+") :" ...
        +" probabilidades exatas versus as experimentais")

// exemplo #3 (polinômio de Hermite seccionado)
x = [0 ; 0.2 ; 0.35 ; 0.5 ; 0.65 ; 0.8 ;  1];
y = [0 ; 0.1 ;-0.1  ; 0   ; 0.4  ;-0.1 ;  0];
d = [1 ; 0   ; 0    ; 1   ; 0    ; 0   ; -1];
X = linspace(0, 1, 200)';
ind = dsearch(X, x);
// definindo funções bases hermitianas
deff("y=Ll(t,k,x)","y=(t-x(k+1))./(x(k)-x(k+1))")   // Lagrange esquerdo em Ik
deff("y=Lr(t,k,x)","y=(t-x(k))./(x(k+1)-x(k))")     // Lagrange direito em Ik
deff("y=Hl(t,k,x)","y=(1-2*(t-x(k))./(x(k)-x(k+1))).*Ll(t,k,x).^2")
deff("y=Hr(t,k,x)","y=(1-2*(t-x(k+1))./(x(k+1)-x(k))).*Lr(t,k,x).^2")
deff("y=Kl(t,k,x)","y=(t-x(k)).*Ll(t,k,x).^2")
deff("y=Kr(t,k,x)","y=(t-x(k+1)).*Lr(t,k,x).^2")
// plotando a curva
Y = y(ind).*Hl(X,ind) + y(ind+1).*Hr(X,ind) + d(ind).*Kl(X,ind) + d(ind+1).*Kr(
xbasc(); plot2d(X,Y,2) ; plot2d(x,y,-9,"000")
xtitle("Um polinômio seccionado de hermite")
// NOTE : a verificação pode ser feita adicionando-se :
// YY = interp(X,x,y,d); plot2d(X,YY,3,"000")
```

# Ver Também

find, tabul

# Autor

B.P.

# Name

eval — avaliação de uma matriz de strings

```
[H]= eval(Z)
```

# Descrição

Retorna a avaliação da matriz de strings Z.

# Exemplos

```
a=1; b=2; Z=['a','sin(b)'] ; eval(Z)  //retorna a matriz [1,0.909];
```

# Ver Também

evstr, execstr

# Name

exp — exponencial em relação aos elementos

```
exp(X)
```

# Parâmetros

X

escalar, vetor ou matriz de entradas reais ou complexas.

# Descrição

`exp(X)` é a função exponencial (elemento a elemento) das entradas de`X`.

# Exemplos

```
x=[1,2,3+%i];
log(exp(x))  //elemento a elemento
2^x
exp(x*log(2))
```

# Ver Também

coff, log, expm

# Name

eye — matriz identidade

```
X=eye(m,n)
X=eye(A)
X=eye()
```

# Parâmetros

A,X
: matrizes ou listas `syslin`

m,n
: inteiros

# Descrição

Define uma matriz `mxn` com 1's ao longo de sua diagonal ou uma matriz identidade com as mesmas dimensões de `A` .

Cuidado: `eye(10)` é interpretado como `eye(A)` com `A=10` i.e. `1`. (NÃO é uma matriz identidade 10 x 10!).

Se `A` é um sistema linear representado por uma lista `syslin` , `eye(A)` retorna uma matriz `eye` com dimensões apropriadas: (número de saídas x número de entradas).

`eye()` produz uma matriz identidade com dimensões indefinidas. As dimensões serão definidas quando esta matriz identidade for adicionada a uma matriz com dimensões fixas.

# Exemplos

```
eye(2,3)
A=rand(2,3);eye(A)
s=poly(0,'s');A=[s,1;s,s+1];eye(A)
A=[1/s,1;s,2];eye(A);
A=ssrand(2,2,3);eye(A)
[1 2;3 4]+2*eye()
```

# Ver Também

ones, zeros

# Name

factor — fatoração

```
[y]=factor(x)
```

# Parâmetros

x
: escalar real

y
: vetor de números primos

# Descrição

Dado um real `x`, `factor(x)` retorna em um vetor `y` a decomposição em números primos de `x`. Caso particular: `factor(0)` retorna `0`, e `factor(1)` retorna `1`.

# Exemplos

```
x=620
y=factor(x)
```

# Ver Também

primes

# Name

fix — arredonda em direção a zero

```
[y]=fix(x)
```

# Parâmetros

x
: uma matriz de reais

y
: uma matriz de inteiros

# Descrição

`fix(x)` retorna uma matriz de inteiros feita de "arredondamentos em direção a zero" ,i.e., `y=sign(x).*floor(abs(x))`. É o mesmo que `int`.

# Ver Também

round, floor, ceil

# Name

flipdim — gira os componentes de x ao longo de uma dada dimensão

```
y=flipdim(x,dim)
```

# Parâmetros

x
: um escalar, vetor ou array de reais

dim
: um inteiro positivo

y
: um escalar, vetor ou array de reais

# Descrição

Dado `x`, um escalar/vetor/array de reais e `dim` um inteiro positivo, esta função gira os componentes de x ao longo da dimensão de número `dim` de `x` (`x` e `y` têm o mesmo tamanho)

# Exemplos

```
// exemplo 1: girando os componentes de x ao longo da primeira dimensão
x=[1 2 3 4; 5 6 7 8];
dim=1;
y=flipdim(x,dim)

// exemplo 1: girando os componentes de x ao longo da segunda dimensão
dim=2;
y=flipdim(x,dim)

// exemplo e: girando os componentes de x ao longo da terceira dimensão
x=matrix(1:48,[3 2,4,2]);
dim=3;
y=flipdim(x,dim)
```

# Ver Também

F.Belahcene

# Name

floor — arredondamento para o inteiro menor ou igual ao número

```
[y]=floor(x)
```

# Parâmetros

x
: matriz de reais

y
: matriz de inteiros

# Descrição

`floor(x)` retorna uma matriz de inteiros feita a partir de "arredondamentos para baixo".

# Exemplos

```
floor([1.9 -2.5])-[1,-3]
floor(-%inf)
x=rand()*10^20;floor(x)-x
```

# Ver Também

round, fix, ceil

# Name

frexp — separa um número em ponto flutuante em potência de base 2 e mantissa

```
[f,e]=frexp(x)
```

# Parâmetros

x
: vetor ou matriz de reais

f
: array de valores reais, geralmente no intervalo 0.5 <= abs(f) < 1.

e
: array de inteiros que satisfazem a equação: `x = f.*2.^e`

# Descrição

Esta função corresponde à função ANSI C frexp(). Quaisquer zeros em `x` produzem `f=0` e `e=0`.

# Exemplos

```
[f,e]=frexp([1,%pi,-3,%eps])
```

# Ver Também

log, hat, ieee, log2

# Name

gsort — ordenação decrescente

```
[s, [k]]=gsort(v )
[s, [k]]=gsort(v,flag1)
[s, [k]]=gsort(v,flag1,flag2)
```

# Parâmetros

v,s
: vetor ou matriz de reais, inteiros ou strings ou sparse vector

flag1
: um string `'r'`, `'c'`,`'g'`,`'lr'` ou `'lc'`.

flag2
: um string `'i'` para ordem crescente ou `'d'` para ordem decrescente. k : vetor ou matriz de inteiros

# Descrição

`gsort` é semelhante a `sort` com propriedades adicionais. O terceiro argumento pode ser usado para escolher ordem crescente ou decrescente. O segundo argumento podem ser usado para ordens léxicas.

`[s,k]=gsort(a,'g')` e `[s,k]=gsort(a,'g','d')` são o mesmo que `[s,k]=gsort(a)`. Eles realizam uma ordenação das entradas da matriz `a`, `a` sendo vista como vetor de pilhas `a(:)` (coluna a coluna). `[s,k]=gsort(a,'g','i')` realiza a mesma operação, mas em ordem crescente.

`[s,k]=gsort(a,'lr')` ordena as linhas da matriz `a` em ordem léxica decrescente. `s` é obtida por uma permutação das linhas da matriz `a` dada pelo vetor coluna `k`) de tal modo que as linhas de `s` verificam `s(i,:) > s(j,:)` se `i<j`. `[s,k]=gsort(a,'lr','i')` realiza a mesma operação, mas em ordem léxica crescente.

`[s,k]=gsort(a,'lc')` ordena as colunas da matriz `a` em ordem léxica decrescente. `s` é obtida por uma permutação das colunas da matriz `int(a)` (ou `a`) dada pelo vetor linha `k`) ide tal modo que as colunas de `s` verificam `s(:,i) > s(:,j)` se `i<j`. `[s,k]=gsort(a,'lc','i')` realiza a mesma operação, mas em ordem léxica crescente.

Quando `v` é complexo, os elementos são ordenados pela magnitude, i.e., abs(`v`) . Apenas 'g' como segundo argumento funciona com complexos.

Se `v` tem elementos `%nan` ou `%inf` . gsort coloca esses elementos no início com o argumento `'i'` ou ao fim com o argumento `'d'`.

# Exemplos

```
alr=[1,2,2;
     1,2,1;
     1,1,2;
     1,1,1];
[alr1,k]=gsort(alr,'lr','i')
[alr1,k]=gsort(alr,'lc','i')

v=int32(alr)

gsort(v)
```

```
gsort(v,'lr','i')
gsort(v,'lc','i')

v=['Scilab' '2.6'
   'Scilab' '2.7'
   'Scicos' '2.7'
   'Scilab' '3.1'
   'Scicos' '3.1'
   'Scicos' '4.0'
   'Scilab' '4.0']

gsort(v,'lr','i')
gsort(v,'lc','i')
```

# Ver Também

find

# Bibliografia

Algoritmo Quicksort.

# Name

hex2dec — conversão de representação hexadecimal para inteiros

```
d=hex2dec(h)
```

# Parâmetros

d
: matriz de inteiros

h
: matriz de strings correspondentes às representações hexadecimais

# Descrição

`hex2dec(x)` retorna a matriz de números correspondentes às representações hexadecimais.

# Exemplos

```
hex2dec(['ABC','0','A'])
```

# Ver Também

base2dec, bin2dec, oct2dec, dec2bin, dec2oct, dec2hex

# Name

imag — parte imaginária

```
[y]=imag(x)
```

# Parâmetros

x
: matriz ou vetor de reais ou complexos

y
: matriz ou vetor de reais

# Descrição

`imag(x)` é a parte imaginária de `x`. (ver `%i` para entrar com números complexos).

# Ver Também

real

# Name

imult — multiplicação pela parte imaginária i

```
y=imult(x)
```

# Parâmetros

x
: escalar real ou complexo, vetor ou matriz de reais ou complexos

y
: escalar complexo, vetor ou matriz de complexos

# Descrição

`imult(x)` é um modo mais eficiente de se multiplicar x por i que y = %i*x, sem os problemas que ocorrem quando x inclui números "especiais" em ponto flutuante como %inf e %nan.

# Exemplos

```
z1 = imult(%inf)
z2 = %i * %inf
```

# Autor

B.P.;

# Name

ind2sub — conversão de índice linear para matriz de valores subescritos

```
[i1,i2,..] =ind2sub(dims,I)
Mi = ind2sub(dims,I)
```

# Parâmetros

dims
: vetor com as dimensões da matriz

I
: vetor com os dados índices lineares

i1,i2,..
: valores subescritos (de mesma forma que `I`)

Mi
: matriz cujas colunas contém os valores subescritos.

# Descrição

`ind2sub` é usado para determinar os valores subescritos equivalentes que correspondem a um índice único num "array". `[i1,i2,..] = ind2sub(dims,I)` retorna os "arrays" `i1`, `i2`, ... contendo os subescritos das linha, coluna, ... equivalentes correspondente à matriz de índices `I` para uma matriz de tamanho `dims`. `Mi=ind2sub(dims,I)` retorna uma matriz `Mi` cujas colunas são os "arrays" `i1(:)`, `i2(:)`, ...

# Exemplos

```
ind2sub([2,3,2],1:12)
[i,j,k]=ind2sub([2,3,2],1:12)
```

# Ver Também

sub2ind, extraction, insertion

# Autor

Serge Steer, INRIA

# Name

int — parte inteira

```
[y]=int(X)
```

# Parâmetros

X
: matriz de reais

y
: matriz de inteiros

# Descrição

`int(X)` retorna a parte inteira da matriz de reais `X`. É o mesmo que `fix`.

# Ver Também

round, floor, ceil

# Name

intersect — returns the vector of common values of two vectors

```
[v [,ka,kb]]=intersect(a,b)
[v [,ka,kb]]=intersect(a,b,orient)
```

# Parâmetros

a
: vetor de números ou strings

b
: vetor de números ou strings

orient
: flag com possíveis valores : 1 ou "r", 2 ou "c"

v
: vetor linha de números ou strings

ka
: vetor linha de inteiros

kb
: vetor linha de inteiros

# Descrição

`intersect(a,b)` retorna um vetor-linha ordenado dos valores comuns a dois vetores `a` e `b`.

`[v,ka,kb]=intersect(a,b)` retorna também dois vetores de índices `ka` e `kb` tais que `v=a(ka)` e `v=b(kb)`.

`intersect(a,b,"r")` ou `intersect(a,b,1)`retorna a matriz formada interseção das linhas únicas de `a` e `b` ordenadas em ordem lexicográfica ascendente. Neste caso, `a` e `b` devem ter o mesmo número de colunas.

`[v,ka,kb]=intersect(a,b,"r")` retorna também dois vetores de índices `ka` e `kb` tais que `v=a(ka,:)` e `v=b(kb,:)`.

`intersect(a,b,"c")` ou `intersect(a,b,2)`retorna a matriz formada interseção das colunas únicas de `a` e `b` ordenadas em ordem lexicográfica ascendente. Neste caso, `a` e `b` devem ter o mesmo número de linhas.

`[v,ka,kb]=intersect(a,b,"c")` retorna também dois vetores de índices `ka` e `kb` tais que `v=a(:,ka)` e `v=b(:,kb)`.

# Observação

NaN são considerados diferentes deles mesmos , então são excluídos da interseção em caso de interseção de vetores.

# Exemplos

```
A=round(5*rand(10,1));
B=round(5*rand(7,1));

intersect(A,B)
[N,ka,kb]=intersect(A,B)

intersect('a'+string(A),'a'+string(B))

intersect(int16(A),int16(B))

//com matrizes
 A = [0,0,1,1 1;
      0,1,1,1,1;
      2,0,1,1,1;
      0,2,2,2,2;
      2,0,1,1,1;
      0,0,1,1,%nan];
B = [1,0,1;
     1,0,2;
     1,2,3;
     2,0,4;
     1,2,5;
     %nan,0,6];

[v,ka,kb] = intersect(A,B,'c')
A(:,ka)
```

# Ver Também

unique, gsort, union

# Name

inttrap — integração de dados experimentais por interpolação trapezoidal

```
v = inttrap([x,] s)
```

# Parâmetros

x
: vetor de dados de coordenadas x crescentes. O valor padrão é `1:size(y,'*')`

s
: vetor de dados de coordenadas y

v
: valor da integral

# Descrição

Computa :

Onde f é uma função descrita por um conjuto de valores experimentais:

`s(i)=f(x(i))` e `x0=x(1)`, `x1=x(n)`

Entre os pontos da malha, a função é interpolada linearmente.

# Exemplos

```
t=0:0.1:%pi
inttrap(t,sin(t))
```

# Ver Também

intg, intc, intl, integrate, intsplin, splin

# Name

isdef — verifica existência de variáveis

```
isdef(name [,where])
```

# Parâmetros

name
: um string

where
: um string opcional com valor padrão 'all' (todos os locais)

# Descrição

`isdef(name)` retorna `%T` se a variável `name` existe e `%F` em caso contrário.

Aviso: uma função que utiliza `isdef` pode retornar um resultado que depende do ambiente!

`isdef(name,'local')` retorna `%T` se a variável `name` existe no ambiente local da função corrente e `%F` em caso contrário.

`isdef(name,'nolocal')` retorna `%T` se a variável `name` existe no ambiente de chamada completo (incluindo o nível global) da função corrente e `%F` em caso contrário.

# Exemplos

```
A=1;
isdef('A')
clear A
isdef('A')

function level1()
  function level2()
    disp(isdef("a","all"));
    disp(isdef("a","local"));
    disp(isdef("a","nolocal"));
  endfunction
  level2()
endfunction
function go()
  a=1;
  level1()
endfunction
go()
```

# Ver Também

exists isglobal, whereis, type, typeof, clear

# Name

isempty — verifica se uma variável é uma matriz vazia ou uma lista vazia

```
t=isempty(x)
```

# Parâmetros

x
: vetor, matriz ou lista

t
: um booleano

# Descrição

`isempty(x)` retorna 'true' (verdadeiro) se x é uma matriz vazia ou uma lista vazia.

# Exemplos

```
a=1
isempty(a(2:$))
isempty(find(rand(1:10)==5))
```

# Name

isequal — comparação de objetos

```
t=isequal(a,b)
t=isequal(a,b,..)
```

# Parâmetros

a, b , ...
: variáveis de quaisquer tipos

t
: um booleano

# Descrição

`isequal` compara seus argumentos. Se todos eles forem iguais, a função retorna `%t` e em caso contrário, retorna `%f` .

Na comparação de listas, estruturas, etc., a o processo é feito recursivamente e a ordem dos campos importa.

Dados de ponto flutuante são comparados de acordo com a regra IEEE, i.e., valores NaN ("não é um número") não são iguais. Ver isequalbitwise para comparações bit a bit.

# Exemplos

```
a=[1 2]
isequal(a,[1 2])
isequal(a,1)
```

# Ver Também

isequalbitwise, equal, less

# Name

isequalbitwise — comparação bit a bit de variáveis

```
t=isequalbitwise(a,b)
t=isequalbitwise(a,b,..)
```

# Parâmetros

a, b , ...
: variáveis de quaisquer tipos

t
: um booleano

# Descrição

`isequalbitwise` compara seus argumentos. Se todos eles são iguais, a função retorna `%t` e, em caso contrário, `%f`.

Na comparação de listas, estruturas, etc., a comparação é feita recursivamente, a ordem dos campos importa.

Dados de ponto flutuante são comparados bit a bit, i.e. valores NaN ("não é um número") são iguais, double(1) e int32(1) não são iguais. Ver isequal para comparações IEEE.

# Exemplos

```
a=list(1:5,%s+1,'ABCDEFG');
isequalbitwise(a,a)
```

# Ver Também

isequal

# Name

isinf — verifica se há entradas infinitas

```
r=isinf(x)
```

# Parâmetros

x
: um vetor ou matriz de reais ou complexos. r : um vetor ou matriz de valores booleanos

# Descrição

`isinf(x)` retorna um vetor ou matriz de valores booleanos que contém entradas verdadeiras correspondentes às entradas de `x` infinitas e entradas falsas correspondentes às entradas de `x` finitas.

# Exemplos

```
isinf([1 0.01 -%inf %inf])
```

# Ver Também

isnan

# Name

isnan — verifica entradas NaN ("não é um número")

```
r=isnan(x)
```

# Parâmetros

x
: real or complex vector or matrix r : boolean vector or matrix

# Descrição

`isnan(x)` retorna um vetor ou matriz de valores booleanos que contém entradas verdadeiras correspondentes às entradas NaN de `x` e entradas falsas correspondentes às entradas regulares de `x` .

# Exemplos

```
isnan([1 0.01 -%nan %inf-%inf])
```

# Ver Também

isinf

# Name

isreal — verifica se uma variável é armazenada como real ou complexa.

```
t=isreal(x)
t=isreal(x,eps)
```

# Parâmetros

x
: vetor ou matriz com entradas ou coeficientes de ponto flutuante

t
: um booleano

# Descrição

`isreal(x)` retorna 'true' (verdadeiro) se `x` é armazenado como uma variável real e falso se `x` é armazenado com uma parte imaginária (que pode ser zero 0).

`isreal(x,eps)` retorna 'true' se `x` é armazenado como uma variável real ou se o valor absoluto máximo dos pontos flutuantes imaginários é menor que ou igual a `eps`.

# Exemplos

```
isreal([1 2])
isreal(1+0*%i)
isreal(1+0*%i,0)
isreal(1+%s)
isreal(sprand(3,3,0.1))
```

# Name

kron — produto de Kronecker (.*.)

```
kron(A,B)
A.*.B
```

# Descrição

`kron(A,B)` ou `A.*.B` retorna o produto tensorial de Kronecker entre duas matrizes `A` e `B`. A matriz resultante tem a seguinte forma de blocos:

```
              | A(1,1) B ..... A(1,n) B |
              |   .              .      |
    A .*. B = |   .              .      |
              |   .              .      |
              | A(m,1) B ..... A(m,n) B |
```

Se `A` é uma matriz `m x n` e `B` é uma matriz `p x q`, então `A.*.B` é uma matriz `(m*p) x (n*q)`.

`A` e `B` podem ser matrizes esparsas.

# Exemplos

```
A=[1,2;3,4];
kron(A,A)
A.*.A
sparse(A).*.sparse(A)
A(1,1)=%i;
kron(A,A)
```

# Name

lex_sort — ordenação lexicográfica de linhas de uma matriz

```
[N, [k]]=lex_sort(M [,sel] [,'unique'])
```

# Parâmetros

M
: matriz de reais

N
: matriz de reais

k
: vetor coluna de inteiros

# Descrição

A função `lex_sort` está obsoleta gora. Ela pode ser substituída pelas funções gsort e unique.

`N=lex_sort(M)` ordena as linhas (como um grupo) da matriz `M` em ordem ascendente. Se requerido, o argumento de saída `k` contém a ordem: `[N,k]=lex_sort(M)` retorna `k` tal que `N` é igual a `M(k,:)` .

`N=lex_sort(M,sel [,'unique'])` produz o mesmo resultado que a seguinte seqüência de instruções:

```
[N,k]=lex_sort(M(:,sel) [,'unique']);
N=M(k,:)
```

O flag (indicador) `'unique'` deve ser fornecido caso deseje-se reter apenas linhas únicas no resultado. Perceba que `lex_sort(M,sel,'unique')` retém apenas linhas tais que M(:,sel) são únicas.

# Exemplos

```
M=round(2*rand(20,3));

lex_sort(M)
lex_sort(M,'unique')
[N,k]=lex_sort(M,[1 3],'unique')
```

# Ver Também

gsort, unique

# Name

linspace — vetor linearmente espaçado

```
[v]=linspace(x1,x2 [,n])
```

# Parâmetros

x1,x2
: escalares reais ou complexos

n
: inteiro (número de valores). O valor padrão é 100

v
: vetor linha de reais ou complexos

# Descrição

Vetor linearmente espaçado. `linspace(x1, x2)` era um vetor linha de n (valor padrão = 100) pontos linearmente espaçados de modo igual entre `x1` e `x2`. Se `x1` ou `x2` são complexos, então `linspace(x1,x2)` retorna um vetor-linha de n complexos cujas partes reais (e respectivamente as imaginárias) dos n complexos são linearmente espaçadas de modo igual entre a partes reais (e respectivamente as imaginárias) de `x1` e `x2`

# Exemplos

```
linspace(1,2,10)
linspace(1+%i,2+2*%i,10)
```

# Ver Também

logspace

# Name

log — logaritmo natural

```
y=log(x)
```

# Parâmetros

x
: vetor ou matriz de constantes

# Descrição

`log(x)` é o logaritmo natural elemento a elemento. `y(i,j)=log(x(i,j))`. Para logaritmo da matriz (matriz logaritmo) veja `logm`.

# Exemplos

```
exp(log([1,%i,-1,-%i]))
```

# Ver Também

exp, logm, log10, ieee

# Name

log10 — logaritmo na base 10

```
y=log10(x)
```

# Parâmetros

x
: vetor ou matriz

# Descrição

Logartimo decimal. Se `x` é um vetor `log10(x)=[log10(x1),...,log10(xn)]`.

# Exemplos

```
10.^log10([1,%i,-1,-%i])
```

# Ver Também

log, logm, hat, ieee

# Name

log1p — computa, com precisão, o logaritmo natural de seu argumento acrescido de uma unidade

```
y=log1p(x)
```

# Parâmetros

x
: escalar real ou vetor ou matriz de reais

y
: escalar real ou vetor ou matriz de reais

# Descrição

`logp1(x)` é a função log(1+x) elemento a elemento. `y(i,j)=log(1 + x(i,j))`. Esta função, definida para x > -1, deve ser usada se quisermos computar log(1+x) com precisão para |x| << 1 ("<<" significa "muito menor").

# Exemplos

```
format("e",24)
log(1.001)
log1p(0.001)
log(1 + 1.e-7)
log1p(1.e-7)
log(1 + 1.e-20)
log1p(1.e-20)
format("v") //retoma o formato padrão
```

# Ver Também

log

# Autor

B.P.;

# Name

log2 — logaritmo na base 2

```
y=log2(x)
```

# Parâmetros

x
: vetor ou matriz

# Descrição

Logaritmo na base 2. Se `x` é um vetor `log2(x)=[log2(x1),...,log2(xn)]`.

# Exemplos

```
2.^log2([1,%i,-1,-%i])
```

# Ver Também

log, hat, ieee, log10, frexp

# Name

logm — logaritmo de matriz quadrada (matriz logaritmo)

```
y=logm(x)
```

# Parâmetros

x
: matriz quadrada

# Descrição

`logm(x)` é o logaritmo da matriz `x`. (matriz logaritmo de x). O resultado é complexo se `x` não é positiva ou positiva definida. Se `x` é uma matriz simétrica, o cálculo é feito pela forma de Schur. Em caso contrário, `x` é assumida como sendo diagonalizável. Tem-se `expm(logm(x))=x`

# Exemplos

```
A=[1,2;3,4];
logm(A)
expm(logm(A))
A1=A*A';
logm(A1)
expm(logm(A1))
A1(1,1)=%i;
expm(logm(A1))
```

# Ver Também

expm, log

# Name

logspace — vetor espaçado logaritmicamente

```
logspace(d1,d2, [n])
```

# Parâmetros

d1,d2
: escalar real ou complexo (significado especial para `%pi`)

n
: inteiro (número de valores). O valor padrão é 50

# Descrição

Retorna um vetor linha de `n` pontos espaçados logaritmicamente de maneira igual entre `10^d1` e `10^d2`. Se `d2=%pi` então os pontos estão entre `10^d1` e `pi`.

# Exemplos

```
logspace(1,2,10)
```

# Ver Também

linspace

# Name

lstsize — números de entradas de uma lista, tlist ou mlist

```
n=lstsize(x)
```

# Parâmetros

l
: uma lista, tlist ou mlist

n
: um inteiro, número de entradas

# Descrição

`lstsize(x)` retorna o número de entradas de uma lista, tlist ou mlist. Esta função é mais eficiente que a função `size` e opera de modo semelhante para todos os tipos de listas, enquanto a função `size` fica sobrecarregada para `mlist`'s.

# Exemplos

```
lstsize(list(1,'aqsdf'))
x=ssrand(3,2,4);
[ny,nu]=size(x)
lstsize(x)
```

# Ver Também

length, size, list, tlist, mlist

# Name

max — máximo

```
[m [,k]]=max(A)
[m [,k]]=max(A,'c')
[m [,k]]=max(A,'r')
[m [,k]]=max(A,'m')
[m [,k]]=max(A1,A2,...,An)
[m [,k]]=max(list(A1,A2,...,An))
```

# Parâmetros

A
: vetor ou matriz de reais

A1,...,An
: um conjunto de vetores ou matrizes, todos de tamanhos iguais, ou de escalares

# Descrição

Para `A`, um vetor ou matriz de reais, `max(A)` é o maior elemento de `A`. `[m,k]=max(A)` fornece, ainda, o índice do máximo. Um segundo argumento do tipo string `'r'` ou `'c'` pode ser usado : `'r'` é usado para obter um vetor-linha `m` tal que `m(j)` contém o méximo da `j` -ésima coluna de A (`A(:,j)`), `k(j)` fornece o índice de linha que contém o máximo para a coluna `j`. `'c'` é usado para a operação dual de linhas de `A`. `'m'` é usado para compatibilidade com Matlab.

`m=max(A1,A2,...,An)`, onde todos os `Aj` são matrizes de mesmo tamanho, retorna um vetor ou matriz `m` de tamanho `size(m)=size(A1)` tal que `m(i)= max( Aj(i)), j=1,...,n`. `[m,k]=max(A1,A2,...,An)` fornece, ainda, o vetor ou matriz `k`. Fixado `i`, `k(i)` é o número do primeiro Aj(i) a alcançar o máximo.

`[m,k]=max(list(A1,...,An))` é uma outra sintaxe para `[m,k]=max(A1,A2,...,An)`

# Exemplos

```
[m,n]=max([1,3,1])
[m,n]=max([3,1,1],[1,3,1],[1,1,3])
[m,n]=max([3,-2,1],1)
[m,n]=max(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=max(list(1,3,1))
```

# Ver Também

gsort, find, mini

# Name

maxi — máximo

```
[m [,k]]=maxi(A)
[m [,k]]=maxi(A,'c')
[m [,k]]=maxi(A,'r')
[m [,k]]=maxi(A,'m')
[m [,k]]=maxi(A1,A2,...,An)
[m [,k]]=maxi(list(A1,A2,...,An))
```

# Parâmetros

A
: vetor ou matriz de reais

A1,...,An
: um conjunto de vetores ou matrizes, todos de tamanhos iguais, ou de escalares

# Descrição

Para `A`, um vetor ou matriz de reais, `maxi(A)` é o maior elemento de `A`. `[m,k]=maxi(A)` fornece, ainda, o índice do máximo. Um segundo argumento do tipo string `'r'` ou `'c'` pode ser usado : `'r'` é usado para obter um vetor-linha `m` tal que `m(j)` contém o méximo da `j` -ésima coluna de A (`A(:,j)`), `k(j)` fornece o índice de linha que contém o máximo para a coluna `j`. `'c'` é usado para a operação dual de linhas de `A`. `'m'` é usado para compatibilidade com Matlab.

`m=maxi(A1,A2,...,An)`, onde todos os `Aj` são matrizes de mesmo tamanho, retorna um vetor ou matriz `m` de tamanho `size(m)=size(A1)` tal que `m(i)= maxi( Aj(i)), j=1,...,n`. `[m,k]=maxi(A1,A2,...,An)` fornece, ainda, o vetor ou matriz `k`. Fixado `i`, `k(i)` é o número do primeiro Aj(i) a alcançar o máximo.

`[m,k]=maxi(list(A1,...,An))` é uma outra sintaxe para `[m,k]=maxi(A1,A2,...,An)`

# Exemplos

```
[m,n]=maxi([1,3,1])
[m,n]=maxi([3,1,1],[1,3,1],[1,1,3])
[m,n]=maxi([3,-2,1],1)
[m,n]=maxi(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=maxi(list(1,3,1))
```

# Ver Também

gsort, find, mini

# Name

meshgrid — cria matrizes ou arrays 3-D

```
[X, Y] = meshgrid(x)
[X, Y] = meshgrid(x,y)
[X, Y, Z] = meshgrid(x,y,z)
```

# Parâmetros

x, y, z
: vetores

X, Y, Z
: matrizes, no caso de dois argumentos de entrada, arrays 3-D, no caso de 3 argumentos de entrada.

# Descrição

...

# Exemplos

```
x = -1:0.1:1;
y = -1:0.1:1;

[X,Y] = meshgrid(x,y);

for i=1:size(X,1)
  for j=1:size(X,2)
    Z(i,j) = sinc(2*%pi*X(i,j)*Y(i,j));
  end
end

surf(X,Y,Z)
```

# Ver Também

ndgrid

# Autor

Farid Belahcene

# Name

min — mínimo

```
[m [,k]]=min(A)
[m [,k]]=min(A,'c')
[m [,k]]=min(A,'r')
[m [,k]]=min(A,'m')
[m [,k]]=min(A1,A2,...,An)
[m [,k]]=min(list(A1,A2,...,An))
```

# Parâmetros

A
: vetor ou matriz de reais

A1,...,An
: um conjunto de vetores ou matrizes, todos de tamanhos iguais, ou de escalares

# Descrição

Para `A`, um vetor ou matriz de reais, `min(A)` é o menor elemento de `A`. `[m,k]=min(A)` fornece, ainda, o índice do mínimo. Um segundo argumento do tipo string `'r'` ou `'c'` pode ser usado : `'r'` é usado para obter um vetor linha `m` tal que `m(j)` contém o mínimo da `j` -ésima coluna de A (`A(:,j)`), `k(j)` fornece o índice de linha que contém o mínimo para a coluna `j`. `'c'` é usado para a operação dual de linhas de `A`. `'m'` é usado para compatibilidade com Matlab.

`m=min(A1,A2,...,An)`, onde todos os `Aj` são matrizes de mesmo tamanho, retorna um vetor ou matriz `m` de tamanho `size(m)=size(A1)` tal que `m(i)= min( Aj(i)), j=1,...,n`. `[m,k]=min(A1,A2,...,An)` fornece, ainda, o vetor ou matriz `k`. Fixado um `i`, `k(i)` é o número do primeiro `Aj(i)` a alcançar o mínimo.

`[m,k]=min(list(A1,...,An))` é uma outra sintaxe para `[m,k]=min(A1,A2,...,An)`

# Exemplos

```
[m,n]=min([1,3,1])
[m,n]=min([3,1,1],[1,3,1],[1,1,3])
[m,n]=min([3,-2,1],1)
[m,n]=min(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=min(list(1,3,1))
```

# Ver Também

gsort, find, mini

# Name

mini — mínimo

```
[m [,k]]=mini(A)
[m [,k]]=mini(A,'c')
[m [,k]]=mini(A,'r')
[m [,k]]=mini(A,'m')
[m [,k]]=mini(A1,A2,...,An)
[m [,k]]=mini(list(A1,A2,...,An))
```

# Parâmetros

A
: vetor ou matriz de reais

A1,...,An
: um conjunto de vetores ou matrizes, todos de tamanhos iguais, ou de escalares

# Descrição

Para `A`, um vetor ou matriz de reais, `mini(A)` é o menor elemento de `A`. `[m,k]=mini(A)` fornece, ainda, o índice do mínimo. Um segundo argumento do tipo string `'r'` ou `'c'` pode ser usado : `'r'` é usado para obter um vetor linha `m` tal que `m(j)` contém o mínimo da `j` -ésima coluna de A (`A(:,j)`), `k(j)` fornece o índice de linha que contém o mínimo para a coluna `j`. `'c'` é usado para a operação dual de linhas de `A`. `'m'` é usado para compatibilidade com Matlab.

`m=mini(A1,A2,...,An)`, onde todos os `Aj` são matrizes de mesmo tamanho, retorna um vetor ou matriz `m` de tamanho `size(m)=size(A1)` tal que `m(i)= mini( Aj(i)), j=1,...,n`. `[m,k]=mini(A1,A2,...,An)` fornece, ainda, o vetor ou matriz `k`. Fixado um `i`, `k(i)` é o número do primeiro `Aj(i)` a alcançar o mínimo.

`[m,k]=mini(list(A1,...,An))` é uma outra sintaxe para `[m,k]=mini(A1,A2,...,An)`

# Exemplos

```
[m,n]=mini([1,3,1])
[m,n]=mini([3,1,1],[1,3,1],[1,1,3])
[m,n]=mini([3,-2,1],1)
[m,n]=mini(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=mini(list(1,3,1))
```

# Ver Também

gsort, find, min

# Name

minus — (-) operador de subtração, mudança de sinal

```
X-Y
-Y
```

# Parâmetros

X
: escalar, vetor ou matriz de números, polinômios ou razões de polinômios. Também pode ser uma lista `syslin`

Y
: escalar, vetor ou matriz de números, polinômios ou razões de polinômios. Também pode ser uma lista `syslin`

# Descrição

Subtração

Para operandos numéricos, é a operação de subtração em seu sentido usual. Se um dos operandos é uma matriz e o outro um escalar, a operção é realizada elemento a elemento. Se `Y==[] X` é retornado; Se `X==[] -Y` é retornado.

A subtração também pode ser definida para outros tipos de dados, através de operações "soft-coded".

# Exemplos

```
[1,2]-1
[]-2

%s-2
1/%s-2
"cat"+"enate"
```

# Ver Também

addf, mtlb_mode

# Name

modulo — Resto aritmético simétrico da divisão de n por m
pmodulo — Resto aritmético positivo da divisão de n por m

```
i=modulo(n,m)
i=pmodulo(n,m)
```

# Parâmetros

n,m
: inteiros ou vetores ou matrizes de inteiros de ordens iguais

# Descrição

`modulo` computa `i= n (modulo m)` i.e. resto da divisão de `m` (`n` e `m` inteiros).

i = n - m .* int (n ./ m). Aqui, a resposta pode ser negativa se `n` ou `m` são negativos.

`pmodulo` computa `i = n - m .* floor (n ./ m)`, a resposta é positiva ou zero.

# Exemplos

```
n=[1,2,10,15];m=[2,2,3,5];
modulo(n,m)

modulo(-3,9)
pmodulo(-3,9)
```

# Name

ndgrid — Arrays para avaliação de função multidimensional em grid

```
[X, Y] = ndgrid(x,y)
[X, Y, Z] = ndgrid(x,y,z)
[X, Y, Z, T] = ndgrid(x,y,z,t)
[X1, X2, ..., Xm] = ndgrid(x1,x2,...,xm)
```

# Parâmetros

x, y, z, ...
: vetores

X, Y, Z, ...
: matrizes, no caso de 2 argumentos de entrada, ou hipermatrizes em outro caso

# Descrição

Esta rotina utilitária é útil para criar arrays para a avaliação da função em grids 2, 3, ..., n dimensionais. Por exemplo, em 2d, um grid é definido por dois vetores, `x` e `y` de comprimento nx e ny, e se deseja avaliar uma função (dita f) em todos os pontos do grid, isto é, em todos os pontos de coordenadas (x(i),y(j)) com i=1,..,nx e j=1,..,ny . Neste caso, esta função pode computar as duas matrizes `X,Y` de tamanho nx x ny tais que :

```
X(i,j) = x(i)   para todo i em [1,nx]
Y(i,j) = y(j)       e j em [1,ny]
```

e a avaliação pode ser feita com `Z=f(X,Y)` (sob a condição de que `f` foi codificada para a avaliação em argumentos de vetor, que é feito (em geral) usando os operadores elemento a elemento `.*`, `./` and `.^` no lugar de `*`, `/` e `^`).

No caso 3d, considerando 3 vetores `x,y,z` de comprimentos nx, ny e nz, `X,Y,Z` são 3 hipermatrizes de tamanho nx x ny x nz tais que :

```
X(i,j,k) = x(i)
Y(i,j,k) = y(j)   para todo (i,j,k) in [1,nx]x[1,ny]x[1,nz]
Z(i,j,k) = z(k)
```

No caso geral de m argumentos de entrada `x1, x2, .., xm`, os m argumentos de saída `X1, X2, .., Xm` são hipermatrizes de tamanho *nx1 x nx2 x ... x nxm* e :

```
Xj(i1,i2,...,ij,...,im) = xj(ij)
for all (i1,i2,...,im) in [1,nx1]x[1,nx2]x...x[1,nxm]
```

# Exemplos

```
// criando um grid 2d simples
nx = 40; ny = 40;
x = linspace(-1,1,nx);
y = linspace(-1,1,ny);
```

```
[X,Y] = ndgrid(x,y);
// computando uma função no grid e plotando
//deff("z=f(x,y)","z=128*x.^2 .*(1-x).^2 .*y.^2 .*(1-y).^2");
deff("z=f(x,y)","z=x.^2 + y.^3")
Z = f(X,Y);
xbasc()
plot3d(x,y,Z, flag=[2 6 4]); xselect()

// criando um grid 3d simples
nx = 10; ny = 6; nz = 4;
x = linspace(0,2,nx);
y = linspace(0,1,ny);
z = linspace(0,0.5,nz);
[X,Y,Z] = ndgrid(x,y,z);
// tente exibir este grid 3d...
XF=[]; YF=[]; ZF=[];
for k=1:nz
   [xf,yf,zf] = nf3d(X(:,:,k),Y(:,:,k),Z(:,:,k));
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf];
end
for j=1:ny
   [xf,yf,zf] = nf3d(matrix(X(:,j,:),[nx,nz]),...
                     matrix(Y(:,j,:),[nx,nz]),...
                     matrix(Z(:,j,:),[nx,nz]));
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf];
end
xbasc()
plot3d(XF,YF,ZF, flag=[0 6 3], leg="X@Y@Z")
xtitle("A 3d grid !"); xselect()
```

# Ver Também

kron

# Autor

B. Pincon

# Name

ndims — número de dimensões de um array

# Parâmetros

A
: um array

n
: inteiro, o número de dimensões do array

# Descrição

`n=ndims(A)` retorna o número de dimensões de um array `A`. `n` é maior que ou igual a 2.

# Exemplos

```
A=rand(2,3);
ndims(A)

A=rand(2,3,2);
size(A),ndims(A)

H=[1/%s,1/(%s+1)]
ndims(H)
```

# Ver Também

size

# Autor

S. Steer

# Name

nearfloat — get previous or next floating-point number

```
xnear = nearfloat(dir, x)
```

# Parâmetros

dir
: string ("succ" ou "pred")

x
: escalar real, vetor ou matriz de reais

xnear
: escalar real, vetor ou matriz de reais

# Descrição

Esta função computa, elemento a elemento, os vizinhos correspondentes dos elementos de `x` (no conjunto de pontos flutuantes subjacente, ver number_properties), os sucessores se `dir = "succ"` e e os predecessores se `dir = "pred"`

# Exemplos

```
format("e",22)
nearfloat("succ",1) - 1
1 - nearfloat("pred",1)
format("v") //retoma o formato padrão
```

# Ver Também

number_properties, frexp

# Autor

B.P.

# Name

nextpow2 — próxima potência de 2 mais alta

```
t=nextpow2(x)
```

# Parameters

x
: vetor ou matriz de reais

p
: matriz ou vetor de inteiros

# Descrição

Se `x` é escalar, `nextpow2(x)` retorna o primeiro `p` tal que `2^p >= abs(x)` . Se `x` é um vetor ou uma matriz `nextpow2(x)` se aplica elemento a elemento.

# Exemplos

```
nextpow2(127)
nextpow2(128)
nextpow2(0:10)
```

# Ver Também

frexp

# Name

norm — norma de matrizes

```
[y]=norm(x [,flag])
```

# Parâmetros

x
: vetor ou matriz de reais ou complexos (armazenamento cheio ou esparso)

flag
: string (tipo da norma) (valor padrão = 2)

# Descrição

For matrices

norm(x)
: ou `norm(x,2)` é o maior valor singular de `x` (`max(svd(x))`).

norm(x,1)
: a norma l_1 de `x` (a maior soma coluna a coluna : `maxi(sum(abs(x),'r'))` ).

norm(x,'inf'),norm(x,%inf)
: a norma infinitesimal de `x` (a maior soma linha a linha : `maxi(sum(abs(x),'c'))` ).

norm(x,'fro')
: norma de Frobenius i.e. `sqrt(sum(diag(x'*x)))`

Para vetores

norm(v,p)
: norma l_p (`sum(v(i)^p))^(1/p)` .

norm(v)
: `=norm(v,2)` : norma l_2

norm(v,'inf')
: `max(abs(v(i)))`.

# Exemplos

```
A=[1,2,3];
norm(A,1)
norm(A,'inf')
A=[1,2;3,4]
max(svd(A))-norm(A)

A=sparse([1 0 0 33 -1])
norm(A)
```

# Ver Também

h_norm, dhnorm, h2norm, abs

# Name

not — (~) não lógico

```
~A
```

# Descrição

~A fornece a negação dos elementos da matriz de valores booleanos A elemento a elemento.

# Exemplos

```
~[%t %t %f]
```

# Ver Também

and, or, find

# Name

number_properties — determina parâmetros de ponto flutuante

```
pr = number_properties(prop)
```

# Parâmetros

prop
: string

pr
: escalar real ou booleano

# Descrição

Esta função pode ser usada para receber os números/propriedades característicos do conjunto de pontos flutuantes aqui dentoado por `F(b,p,emin,emax)` ((geralmente o conjunto de "floats" de 64 bits prescritos por IEEE 754). Números de `F` são da forma:

```
sign * m * b^e
```

`e` é o expoente e `m` a mantissa:

```
m = d_1 b^(-1) + d_2 b^(-2) + .... + d_p b^(-p)
```

`d_i` os dígitos estão em `[0, b-1]` e `e` em `[emin, emax]`, o número é dito "normalizado" se `d_1 ~= 0`. Os seguintes podem ser recebidos:

prop = "radix"
: então `pr` é a raiz `b` do conjunto `F`

prop = "digits"
: então `pr` é o número de dígitos de `p`

prop = "huge"
: então `pr` é o maior float positivo de `F`

prop = "tiny"
: então `pr` é o menor float normalizado positivo de `F`

prop = "denorm"
: então `pr` é um booleano (%t se números denormalizados são utilizados)

prop = "tiniest"
: então se denorm = %t, `pr` é o número positivo denormalizado mínimo. Em outro caso, `pr` = tiny

prop = "eps"
: então `pr` é a máquina epsilon ( geralmente (`b^(1-p))/2` ) que é o erro máximo relativo entre um real `x` (tal que `|x|` está em `[tiny, huge]`) e `fl(x)`, sua aproximação em ponto flutuante em `F`

prop = "minexp"
: então `pr` é `emin`

prop = "maxexp"
então `pr` é `emax`

# Observações

Esta função usa uma rotina de LAPACK dlamch para receber os parâmetros máquinas (os nomes (radix, digits, huge, etc...) são aqueles recomendados pelo padrão LIA 1 e são diferentes daqueles correspondentes em LAPACK) ; CUIDADO: às vezes você poderá encontrar a seguinte para a máquina epsilon: `eps = b^(1-p)` , mas nesta função nós usamos a definição tradicional (ver prop = "eps" acima) e então `eps = (b^(1-p))/2` se o arredondamento normal acontecer e `eps = b^(1-p)` se não acontecer.

# Exemplos

```
b = number_properties("radix")
eps = number_properties("eps")
```

# Ver Também

nearfloat, frexp

# Autor

Bruno Pincon

# Name

oct2dec — conversão de octais para inteiros

```
d=oct2dec(o)
```

# Parâmetros

d
: matriz de inteiros

o
: matriz de strings correspondentes às representações octais

# Descrição

`oct2dec(x)` retorna a matriz de número correspondentes à representação octal

# Exemplos

```
oct2dec(["1" "756115"; "0" "23"])
```

# Ver Também

base2dec, bin2dec, hex2dec, dec2bin, dec2oct, dec2hex

# Name

ones — matriz de entradas todas iguais a 1

```
y=ones(m1,m2,...)
y=ones(x)
y=ones()
```

# Parâmetros

x,y
: matrizes

m1, m2,..
: inteiros

# Descrição

Retorna uma matriz cujas entradas são todas iguais a 1.

ones(m1,m2)
: retorna uma matriz `(m1,m2)` cujas entradas são todas 1.

ones(m1,m2,..,mn)
: retorna uma matriz `(m1,m2,..,mn)` cujas entradas são todas 1.

ones(x)
: retorna uma matriz cujas entradas são todas 1 com o mesmo tamanho que `x`.

ones(x)
: também é válido para `x` uma lista `syslin` .

Perceba que `ones(3)` é `ones(a)` com `a=3` , i.e., NÃO é uma matriz 3x3!

ones() é equivalente a ones(1,1).

# Exemplos

```
ones(3)
ones(3,3)
ones(2,3,2)
```

# Ver Também

eye, zeros

# Name

or — (|) ou lógico

```
or(A), or(A,'*')
or(A,'r'), or(A,1)

or(A,'c'), or(A,2)
A|B
```

# Descrição

`or(A)` fornece o `or` dos elementos da matriz de valores booleanos `A`. `or(A)` é verdadeira (`%t`) se, e só se, pelo menos uma entrada de `A` é `%t`.

`y=or(A,'r')` (ou, equivalentemente, `y=or(A,1)`) é o or linha a linha. Retorna em cada entrada do vetor linha `y` o "or" das linhas de `x` (o "or" é realizado no índice de linhas : `y(j)= or(A(i,j),i=1,m)`).

`y=or(A,'c')` (ou, equivalentemente, `y=or(A,2)`) é o or coluna-a-coluna. Retorna em cada entrada do vetor-coluna `y` o "or" das colunas de `x` (o "or" é realizado no índice de colunas : `y(i)= or(A(i,j),j=1,n)`)).

`A|B` fornece o `or` elemento a elemento das matrizes `A` e `B` .`A` e `B` devem ser matrizes de mesmas dimensões ou uma delas deve ser um único booleano.

# Exemplos

```
or([%t %t %f])
[%t %t %f]|[%f %t %t]
[%t %t %f]|%f
```

# Ver Também

and, not, find

# Name

pen2ea — conversão de feixe em matrizes E e A

```
[E,A]=pen2ea(Fs)
```

# Parâmetros

Fs
: feixe de matrizes `s*E-A`

E,A
: duas matrizes tais que `Fs=s*E-A`

# Descrição

Função utilitária. Dado o feixe `Fs=s*E-A`, retorna as matrizes `E` e `A`.

# Exemplos

```
E=[1,0];A=[1,2];s=poly(0,'s');
[E,A]=pen2ea(s*E-A)
```

# Name

perms — retorna todas as permutações dos componentes de um vetor

```
y=perms(x)
```

# Parâmetros

x
: um escalar ou um vetor

y
: um escalar ou uma matriz

# Descrição

Dado um vetor `x` de comprimento `n`, `perms` retorna todas as permutações dos `n` componentes de `x` (i.e `n!` permutações). O tamanho de `y` é `n! x n`

# Exemplos

```
x=[4, 7, 10]
y=perms(x)
x=[1, 5, 2, 5]
y=perms(x)
```

# Name

permute — permuta as dimensões de um array

```
y=permute(x,dims)
```

# Parâmetros

dims
: um escalar ou um vetor de números reais positivos

x
: um escalar, um vetor, uma matriz ou um multi-array

# Descrição

`y=permute(x,dims) :`

# Exemplos

```
//exemplo 1:
x=[1 2 3;4 5 6];
y=permute(x,[2 1]);

//exemplo 2:
x=matrix(1:12,[2,3,2]);
y=permute(x,[3 1 2]);
```

# Ver Também

pertrans, quote, cat

# Autor

Farid Belahcene

# Name

pertrans — matriz pertransposta

```
[Y]=pertrans(X)
```

# Parâmetros

X
: matriz de reais ou complexos

Y
: matriz de reais ou complexos

# Descrição

`Y=pertrans(X)` retorna a matriz pertransposta de `X`, i.e. a matriz simétrica a `X` em relação à segunda diagonal (função utilitária).

# Exemplos

```
A=[1,2;3,4]
pertrans(A)
```

# Name

primes — função dos primos

```
[y]=primes(x)
```

# Parâmetros

x
: escalar real

y
: vetor

# Descrição

Dado um real `x`, `primes(x)` retorna em um vetor `y` todos os números primos entre 1 e `x`. Se `x<2` então, `primes(x)` retorna uma matriz vazia.

# Exemplos

```
x=35
y=primes(x)
```

# Ver Também

factor

# Name

prod — produto

```
y=prod(x)
y=prod(x,'r') ou y=prod(x,1)
y=prod(x,'c') ou y=prod(x,2)
y=prod(x,'m')
```

# Parâmetros

x
: matriz ou vetor de reais ou complexos

y
: matriz ou vetor de reais ou complexos

# Descrição

Para um vetor ou uma matriz `x`, `y=prod(x)` retorna no escalar `y` o produtório de todas as entradas de `x`, , ex. `prod(1:n)` é n!

`y=prod(x,'r')` (ou, equivalentemente, `y=prod(x,1)`) computa os produtórios dos elementos das linhas de `x`. `y` é o vetor linha: `y(1,j)=prod(x(:,j))`.

`y=prod(x,'c')` (ou, equivalentemente, `y=prod(x,2)`) computa os produtórios dos elementos das colunas de `x`. `y` é o vetor coluna: `y(i,1)=prod(x(i,:))`.

`y=prod(x,'m')` é o produtório ao longo da primeira dimensão "não-singleton" de `x` (para compatibilidade com o Matlab).

`prod` não é implementado para matrizes esparsas.

# Exemplos

```
A=[1,2;0,100];
prod(A)
prod(A,'c')
prod(A,'r')
```

# Ver Também

sum, cumprod

# Name

rand — gerador de números randômico

```
rand(m1,m2,.. [,key])
rand(x [, key])
rand()

rand(key)
rand("seed" [,n])
rand("info")
```

# Parâmetros

mi
: inteiros

key
: string com valor `"uniform"`, ou `"normal"`

x
: uma matriz. Apenas suas dimensões são levadas em conta

# Descrição

Gerador de matriz randômico.

Sem argumento "key", as sintaxes abaixo produzem matrizes randômicas com o gerador randômico corrente (o padrão é "uniform")

rand(m1,m2)
: é uma matriz randômica de dimensão `m1` por `m2`.

rand(m1,m2,..,mn)
: é uma matriz randômica de dimensão `m1` por `m2`,.. por `mn`.

rand(a)
: é uma matriz randômica com mesmo tamanho que a. `rand(a)`é de complexos se `a` é uma matriz de complexos.

rand() : sem argumentos, fornece um escalar cujo valor muda
: a cada vez que é referenciado.

Se presente, o argumento "key" permite especificar uma outra distribuição randômica.

rand('uniform')
: o gerador corrente é acertado como gerador randômico uniforme. Números aleatórios são distribuidos uniformemente e pertencem ao intervalo (0,1).

rand('normal')
: o gerador corrente é acertado como gerador de números randômico gaussiano (com média 0 e variância 1) .

str=rand('info')
: retorna o tipo do gerado randômico padrão ('uniform' ou 'normal')

É possível (re-)inicializar o "seed" do gerador randômico:

rand('seed')
: retorna o valor corrente do "seed".

rand('seed',n)
: define o seed como `n`. (o padrão é `n=0` numa primeira chamada).

# Observação

Use a função mais poderosa grand ao invés.

# Exemplos

```
x=rand(10,10,'uniform')
rand('normal')
rand('info')
y=rand(x,'normal');
x=rand(2,2,2)
```

# Ver Também

grand, ssrand

# Name

rat — aproximação racional de ponto-flutuante

```
[N,D]=rat(x [,tol])
y=rat(x [,tol])
```

# Parâmetros

x
: vetor ou matriz de reais

n
: vetor ou matriz de inteiros

d
: vetor ou matriz de inteiros

y
: vetor ou matriz de reais

# Descrição

`[N,D] = rat(x,tol)` retorna duas matrizes de inteiros tais que `N./D` é próxima a `x` no sentido de que `abs(N./D - X) <= tol*abs(x)`. As aproximações racionais são geradas truncando-se expansões de frações contínuas. `tol = 1.e-6*norm(X,1)` é o padrão. `y = rat(x,tol)` retorna o quociente `N./D`

# Exemplos

```
[n,d]=rat(%pi)
[n,d]=rat(%pi,1.d-12)
n/d-%pi
```

# Ver Também

int, round

# Name

real — parte real

```
[y]=real(x)
```

# Parâmetros

x
: vetor ou matriz de reais ou complexos

y
: matriz de reais

# Descrição

`real(x)` é a parte real de `x` (ver `%i` para entrar com números complexos).

# Ver Também

imag

# Name

resize_matrix — cria uma nova matriz com tamanho diferente

```
resMat = resize_matrix(mat,nbRow,nbCol,[typeOfMat])
```

# Parâmetros

mat
: matriz de entrada da qual a matriz redimensionada será criada

nbRow
: número de linhas da matriz redimensionada

nbCol
: número de colunas da matriz redimensionada

typeOfMat
: string, nome do tipo da matriz redimensionada

resMat
: matriz redimensionada

# Descrição

Cria uma matriz de tamanho `nbRow` x `nbCol` e cujos elementos `(i,j)` são `mat(i,j)` se `(i,j)` está ao alcance da matriz de entrada. De outro modo, `(i,j)` são `0` para matrizes de reais ou de inteiros, `%f` para matrizes de booleanos e um string vazio para matrizes de strings.

O tipo da matriz de saída pode ser modificado especificando-se o argumento `typeOfMat`. Neste caso, esteja certo de que o tipo da matriz de entrada é compatível com este.

Por enquanto, apenas matrizes de reais, inteiros, booleanos, e strings são suportadas. Isto significa que `typeOfMat` deve ser escolhido dentre: 'constant', 'boolean', 'string' ou qualquer tipo inteiro ('int8', 'int16',...).

# Exemplos

```
// matriz de números
myMat = 5 * rand( 3, 4 )
myMat = resize_matrix( myMat, 3, 3 ) // reduzindo tamanho da matriz
myMatInteger = resize_matrix( myMat, 4, 4, 'int32' ) // criando uma matriz de
myMatBoolean = resize_matrix( myMat, 2, 2, 'boolean' )
myMatBoolean = resize_matrix( myMatBoolean, 3, 5 )

// Matriz de strings
myMatString = ["Scilab","the";"Open Source","Scientific";"Software","Package"
myMatString = resize_matrix( myMatString, 3, 1 )
```

# Ver Também

matrix, size, typeof

# Autor

Jean-Baptiste Silvy

# Name

round — arredondamento para inteiro mais próximo

```
y=round(x)
```

# Parâmetros

x
: matriz de reais ou complexos

y
: matriz de inteiros ou complexos (com partes reais e imaginárias inteiras )

# Descrição

`round(x)` arredonda os elementos de `x` para os inteiros mais próximos.

# Exemplos

```
round([1.9 -2.5])-[2,-3]
round(1.6+2.1*%i)-(2+2*%i)
round(-%inf)
x=rand()*10^20;round(x)-x
```

# Ver Também

int, floor, ceil

# Name

sec — computa a secante elemento a elemento do argumento.

```
y = sec(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos

# Descrição

Computa a secante elemento a elemento do argumento. A secante é uma função periódica definida como 1/cos. Para dados reais, os resultados são reais em `]-%inf -1] U [1 %inf[`.

# Exemplos

```
x=[0 %pi/3 2*%pi/3 %pi/4 3*%pi/4 %pi/6 5*%pi/6 %pi];
sec(x)
x=linspace(-%pi,%pi,100)
plot(x,sec(x))
```

# Ver Também

cos, secd

# Autor

Serge Steer, INRIA

# Função Usada

Esta função usa a função `cos`.

# Name

secd — computa a secante elemento a elemento do argumento em graus

```
y = secd(x)
```

# Parâmetros

x
: array de reais

y
: array de reais

# Descrição

as entradas de `y` são as secantes `1/cos` das entradas de `x` dadas em graus. Os resultados são reais e estão em `]-%inf -1] U [1 %inf[`. para entradas iguais a `n*180` com `n` inteiro, o resultado é exatamente `-1` ou `+1`. para entradas iguais a `n*90` com `n` inteiro e ímpar, o resultado é infinito (ou um erro dependendo do modo ieee).

# Exemplos

```
secd(90)
sec(%pi/2)
```

# Ver Também

cosd, sec

# Autor

Serge Steer, INRIA

# Name

sech — computa a secante hipertbólica do argumento elemento a elemento

```
y = sech(x)
```

# Parâmetros

x
: array de reais ou complexos

y
: array de reais ou complexos

# Descrição

Computa a secante hipertbólica do argumento elemento a elemento. A secante hiperbólica é definida como 1/cosh. Para dados reais, os resultados são reais em `[0 1]`.

# Exemplos

```
x=linspace(-10,10,1000)
plot(x,sech(x))
```

# Ver Também

cosh, asech

# Autor

Serge Steer, INRIA

# Name

setdiff — retorna componentes de um vetor que não pertecem a outro vetor

```
v=setdiff(a,b)
[v,ka]=setdiff(a,b)
```

# Parâmetros

a
: vetor de números reais ou strings

b
: vetor de números reais ou strings

v
: vetor de números reais ou strings com a mesma orientação de `a`

ka
: vetor linha de inteiros, ka(i) é a localização de v(i) em a

# Descrição

`setdiff(a,b)` retorna um vetor ordenado que retém as entradas de `a` que não estão em `b`

`[v,ka]=setdiff(a,b)` retorna um vetor ordenado que retém as entradas de `a` que não estão em `b` e a localização destas entradas em a.

# Exemplos

```
a = [223;111;2;4;2;2];
b = [2;3;21;223;123;22];
setdiff(a,b)
[v,k]=setdiff(string(a),string(b))
```

# Ver Também

unique, gsort, union

# Name

sign — função de sinal

# Descrição

`X=sign(A)` retorna a matriz feita dos sinais de `A(i,j)`. Para `A`, feita de complexos `sign(A) = A./abs(A)`.

# Exemplos

```
sign(rand(2,3))
sign(1+%i)
```

# Ver Também

abs

# Name

signm — função do sinal da matriz

# Descrição

Para matrizes quadradas e Hemitianas, `X=signm(A)` é a função do sinal da matriz.

# Exemplos

```
A=rand(4,4);B=A+A';X=signm(B);spec(B),spec(X)
```

# Ver Também

sign

# Name

sin — função seno

```
[t]=sin(x)
```

# Parâmetros

x
: vetor ou matriz de reais ou complexos

# Descrição

Para um vetor ou matriz, `sin(x)` é o seno de seus elementos. Para o seno da matriz (matriz seno) use a função `sinm(X)`.

# Exemplos

```
asin(sin([1,0,%i]))
```

# Ver Também

sinm

# Name

sinc — função sinc (seno cardinal)

```
t=sinc(x)
```

# Parâmetros

x
: vetor ou matriz de reais ou complexos

t
: vetor ou matriz de reais ou complexos

# Descrição

Se `x` é um vetor ou uma matriz, `t=sinc(x)` é o vetor ou matriz tal que `t(i)=sin(x(i))/x(i)` if `x(i)~=0` e `t(i)=1` se `x(i)==0`

# Exemplos

```
x=linspace(-10,10,3000);
plot2d(x,sinc(x))
```

# Ver Também

sin, cos

# Name

sind — função seno, argumento em graus

```
t=sind(x)
```

# Parâmetros

x
: array de reais ou complexos

t
: array de reais ou complexos com as mesmas dimensões que `x`

# Descrição

Para um vetor ou uma matriz x , `sind(x)` é o seno de seus elementos que se supõe estar em graus. Os resultados estão em [-1 1]. Para inteiros `n`, `sind(n*180)` é exatamente zero.

# Exemplos

```
x=[0,30 45 60 90 360];
sind(x)
```

# Ver Também

sin

# Autor

Serge Steer, INRIA

# Name

sinh — seno hiperbólico

```
[t]=sinh(x)
```

# Parâmetros

x,t
: vetores ou matrizes de reais ou complexos

# Descrição

Os elementos do vetor `t` são os senos hiperbólicos dos elementos do vetor `x`.

# Exemplos

```
asinh(sinh([0,1,%i]))
```

# Ver Também

asinh

# Name

sinhm — seno hiperbólico da matriz (matriz seno hiperbólico)

```
t=sinhm(x)
```

# Parâmetros

x,t
: matriz quadrada de reais ou complexos

# Descrição

sinhm(x) é o seno hiperbólico da matriz x (matriz seno hiperbólico de x). `t=(expm(x)-expm(-x))/2`

# Exemplos

```
A=[1,2;2,3]
asinhm(sinhm(A))
A(1,1)=%i;sinhm(A)-(expm(A)-expm(-A))/2   //Caso para complexos
```

# Ver Também

sinh

# Name

sinm — matriz seno

```
t=sinm(x)
```

# Parâmetros

x
: matriz quadrada de reais ou complexos

# Descrição

`sinm(x)` é o seno da matriz x (matriz seno de x).

# Exemplos

```
A=[1,2;2,4];
sinm(A)+0.5*%i*(expm(%i*A)-expm(-%i*A))
```

# Ver Também

sin, asinm

# Name

size — tamanho de objetos

```
y=size(x [,sel])
[nr,nc]=size(x)
```

# Parâmetros

x
: matrix (incluindo matriz de transferência), lista, ou sistema linear (`syslin`)

y
: `1x2` vetor de inteiros ou número inteiro

sel
: um escalar ou um string

nr,nc
: dois inteiros

# Descrição

Aplicado a :
: uma matriz (de constantes, de polinômios, de "strings", de booleanos ou de razões de polinômios) `x`, com apenas um argumento LHS ("Left Hand Side" ou "do lado esquerdo") `size` retorna um vetor `1x2` [número de linhas, número de colunas]. Chamado com LHS= 2, retorna `nr,nc` = [número de linhas, número de colunas]. `sel` pode ser usado para especficar a dimensão a ser recebida:

1 ou 'r'
: para receber o número de linhas.

2 ou 'c'
: para receber o número de colunas

'*'
: para receber o produto de linhas e números de colunas

Aplicado a:
: uma lista retorna o número de elementos. Neste caso, apenas a sintaxe `y=size(x)` pode ser usada.

Aplicado a:
: um sistema linear, `y=size(x)` retorna em `y` o vetor (linha) [número de saídas, número de entradas] i.e. a dimensão da matriz de transferência correspondente. A sintaxe `[nr,nc]=size(x)` também é válida (com `(nr,nc)=(y(1),y(2))`. Se `x` é um sistema linear em forma de estado-espaço, então `[nr,nc,nx]=size(x)` retorna ainda a dimensão `nx` da matriz `A` de `x`.

Aplicado a
: uma hipermatriz `y=size(x)` retorna o vetor de dimensões da hipermatriz. `[n1,n2,...nn]=size(x)` retorna as dimensões da hipermatriz. `ni=size(x,i)` retorna a i-ésima dimensão e `size(x,'*')` retorna o produto de dimensões.

# Exemplos

```
[n,m]=size(rand(3,2))
[n,m]=size(['a','b';'c','d'])
x=ssrand(3,2,4);[ny,nu]=size(x)
[ny,nu]=size(ss2tf(x))
[ny,nu,nx]=size(x)
```

# Ver Também

length, syslin

# Name

solve — Solucionador simbólico de sistemas lineares

```
[x]=solve(A,b)
```

# Parâmetros

A,b,x
: matrizes (respectivamente vetores) de "strings"

# Descrição

Resolve `A*x = b` quando `A` é uma matriz triangular superior feita de strings.

# Exemplos

```
A=['1','a';'0','2'];   //Triangular superior
b=['x';'y'];
w=solve(A,b)
a=1;x=2;y=5;
evstr(w)
inv([1,1;0,2])*[2;5]
```

# Ver Também

trianfml

# Name

sort — ordenamento decrescente (DEPRECATED see gsort)

```
[s, [k]]=sort(v)
[s, [k]]=sort(v,'r')
[s, [k]]=sort(v,'c')
```

# Parâmetros

v
: vetor esparso; vetor ou matriz de reais, complexos ou "strings"

s
: vetor esparso; vetor ou matriz de reais, complexos ou "strings"

k
: vetor ou matriz de inteiros

# Descrição

sort will be removed in Scilab 5.3. see gsort

A função `sort` está obsoleta. Ela pode ser substituída por gsort.

`s=sort(v)` põe `v` em ordem decrescente. Se `v` é uma matriz, a ordenação é feita coluna a coluna, `v` sendo visto como o vetor empilhado `v(:)`. Se `v` é um string, a ordenação é crescente. `[s,k]=sort(v)` retorna ainda os índices de entradas de `s` em `v`, i.e. `v(k(:))` é o vetor `s`.

`s=sort(v,'r')` põe as linhas de `v` em ordem decrescente i.e. cada coluna de `s` é obtida de cada coluna de `v` por reordenação decrescente. `[s,k]=sort(v,'r')` retorna ainda em cada coluna de `k` os índices tais que `v(k(:,i),i)=s(:,i)` para cada índice de coluna `i`.

`s=sort(v,'c')` põe as colunas de `v` em ordem decrescente i.e. cada linha de `s` é obtida de cada linha de `v` por reordenação decrescente. `[s,k]=sort(v,'c')` retorna ainda em cada linha de `k` os índices tais que `v(i,k(i,:))=s(i,:)` para cada índice de linha `i`.

Matrizes ou vetores de complexos são ordenados de acordo com suas magnitudes. Ordenação por colunas/linhas não é implementada para matrizes complexas.

`y=sort(A)` é válido quando `A` é um vetor esparso. Ordenação de linhas/colunas não é implementado para matrizes esparsas.

Limitação : se `v` inclui elementos %nan, a matriz não será ordenada. Por favor, use gsort neste caso.

# Exemplos

```
[s,p]=sort(rand(1,10));
//p  é uma permutação aleatória de 1:10
A=[1,2,5;3,4,2];
[Asorted,q]=sort(A);A(q(:))-Asorted(:)
v=1:10;
sort(v)
sort(v')
sort(v,'r')  //Não faz nada para vetores-linha
sort(v,'c')
```

## Ver Também

find, gsort

# Name

sp2adj — converte uma matriz esparsa para forma de adjacência

# Parâmetros

A
: matriz esparsa de reais ou complexos (nz entradas não-nulas)

xadj
: vetor de inteiros de comprimento (n+1).

adjncy
: vetor de inteiros de comprimento nz contendo os índices de linha para os elementos correspondentes em anz

anz
: vetor coluna de comprimento nz contendo os elementos não-nulos de A

# Descrição

```
sp2adj converte uma matriz esparsa para usa forma de adjacência
(função utilitária).
A = matriz esparsa n x m . xadj, adjncy, anz = representação
em adjacência de A, i.e.:
```

`xadj(j+1)-xadj(j)` = número de entradas não-nulas na linha j. `adjncy` = índice de coluna das entradas não-nulas nas linha 1, linha 2,..., linha n. `anz` = valores de entradas não-nulas nas linha 1, linha 2,..., linha n. `xadj` é um vetor (coluna) de tamanho n+1 e `adjncy` é um vetor (coluna) de inteiros de tamanho `nz=nnz(A)`. `anz` é um vetor de reais de tamanho `nz=nnz(A)`.

# Exemplos

```
A = sprand(100,50,.05);
[xadj,adjncy,anz]= sp2adj(A);
[n,m]=size(A);
p = adj2sp(xadj,adjncy,anz,[n,m]);
A-p,
```

# Ver Também

adj2sp, sparse, spcompack, spget

# Name

speye — matriz identidade esparsa

```
Isp=speye(nrows,ncols)
Isp=speye(A)
```

# Parâmetros

nrows
: inteiro (número de linhas)

ncols
: inteiro (número de colunas)

A
: matriz esparsa

sp
: matriz identidade esparsa

# Descrição

`Isp=speye(nrows,ncols)` retorna uma matriz identidade esparsa `Isp` com `nrows` linhas e , `ncols` colunas (matrizes identidades não-quadradas têm um número máximo de algarismos 1 na diagonal principal).

`Isp=speye(A)` retorna uma matriz identidade esparsa com as mesmas dimensões de `A`. Se `[m,n]=size(A)`, `speye(m,n)` e `speye(A)` são equivalentes. Em particular `speye(3)` não é equivalente a `speye(3,3)`.

# Exemplos

```
eye(3,3)-full(speye(3,3))
```

# Ver Também

sparse, full, eye, spzeros, spones

# Name

spones — matriz esparsa

```
sp=spones(A)
```

# Parâmetros

A
: matriz esparsa

sp
: matriz esparsa

# Descrição

`sp=spones(A)` gera uma matriz com a mesma estrutura de espargimento de `A`, mas com 1 em posições não-nulas;

# Exemplos

```
A=sprand(10,12,0.1);
sp=spones(A)
B = A~=0
bool2s(B)
```

# Ver Também

sparse, full, eye, speye, spzeros

# Name

sprand — matriz esparsa randômica

```
sp=sprand(nrows,ncols,fill [,typ])
```

# Parâmetros

nrows
: inteiro (número de linhas)

ncols
: inteiro (número de colunas)

fill
: coeficiente de preenchimento (densidade)

typ
: string (`'uniform'` (padrão) ou `'normal'`)

sp
: matriz esparsa

# Descrição

`sp=sprand(nrows,ncols,fill)` retorna uma matriz esparsa `sp` com `nrows` linhas e `ncols` colunas e aproximadamente `fill*nrows*ncols` entradas não-nulas.

Se `typ='uniform'` valores uniformemente distribuídos em [0,1] são gerados. Se `typ='normal'` valores normalmente distribuídos são gerados (média=0 e desvio padrão=1).

# Exemplos

```
W=sprand(100,1000,0.001);
```

# Ver Também

sparse, full, rand, speye

# Name

spzeros — matriz nula esparsa

```
sp=spzeros(nrows,ncols)
sp=spzeros(A)
```

# Parâmetros

nrows
: inteiro (número de linhas)

ncols
: inteiro (número de colunas)

A
: matriz esparsa

sp
: matriz nula esparsa

# Descrição

`sp=spzeros(nrows,ncols)` retorna uma matriz nula esparsa `sp` com `nrows` linhas e, `ncols` colunas. (Equivalente a `sparse([],[],[nrow,ncols])`)

`sp=spzeros(A)` retorna uma matriz nula esparsa com as mesmas dimensões que `A`. Se `[m,n]=size(A)`, `spzeros(m,n)` e `spzeros(A)` são equivalentes. Em particular `spzeros(3)` não é equivalente a `spzeros(3,3)`.

# Exemplos

```
sum(spzeros(1000,1000))
```

# Ver Também

sparse, full, eye, speye, spones

# Name

sqrt — raiz quadrada

```
y=sqrt(x)
```

# Parâmetros

x
: escalar real ou complexo ou vetor de reais ou complexos

# Descrição

`sqrt(x)` é o vetor de raízes quadradas dos elementos de `x`. O resultado é complexo se `x` é negativo.

# Exemplos

```
sqrt([2,4])
sqrt(-1)
```

# Ver Também

hat, sqrtm

# Name

sqrtm — raiz quadrada da matriz (matriz raiz quadrada)

```
y=sqrtm(x)
```

# Parâmetros

x
: matriz quadrada de reais ou complexos

# Descrição

`y=sqrt(x)` é a raiz quadrada da matriz `x` (matriz raiz quadrada de x) (`x=y^2`) . O resultado pode ser inacurado se `x` não é simétrica.

# Exemplos

```
x=[0 1;2 4]
w=sqrtm(x);
norm(w*w-x)
x(1,2)=%i;
w=sqrtm(x);norm(w*w-x,1)
```

# Ver Também

expm, sqroot

# Name

squarewave — gera uma onda quadrada de período 2*%pi

```
x=squarewave(t [,percent])
```

# Parâmetros

t
: vetor de reais, discretização do tempo

x
: vetor de reais, o valor da onda em cada ponto do tempo no conjunto (-1,+1)

percent
: escalar real positivo, a porcentagem do período no qual o sinal é positivo. O valor padrão é 50

# Descrição

`squarewave(t)` gera o vetor dos valores da onda quadrada de período 2*%pi em cada data dada no vetor `t` .

`squarewave(t,%)` gera uma onda quadrada tal que % é a porcentagem do período no qual o sinal é positivo.

# Exemplos

```
t=(0:0.1:5*%pi)';
plot2d1('onn',t,[2*sin(t),1.5*squarewave(t),squarewave(t,10)])
```

# Ver Também

sin, cos

# Name

ssrand — gerador de sistema randômico

```
sl=ssrand(nout,nin,nstate)
[sl,U]=ssrand(nout,nin,nstate,flag)
```

# Parâmetros

nout
: inteiro (número de saídas)

nin
: inteiro (número de entradas)

nstate
: inteiro (dimensão de espaço de estados)

flag
: lista feita de um string ou vários inteiros

sl
: lista (lista `syslin` )

U
: matriz quadrada não-singular `(nstate x nstate)`

# Descrição

`sl=ssrand(nout,nin,nstate)` retorna um sistema em espaço de estado randômico estritamente próprio (`D=0`) de tamanho `[nout,nint]` representado por uma lista `syslin` e com `nstate` variáveis de estado.

`[sl,U]=ssrand(nout,nin,nstate,flag)` retorna um sistema linear teste com dadas propriedades especificadas por `flag`. `flag` pode ser um dos seguintes:

```
flag=list('co',dim_cont_subs)
flag=list('uo',dim_unobs_subs)
flag=list('ncno',dim_cno,dim_ncno,dim_co,dim_nco)
flag=list('st',dim_cont_subs,dim_stab_subs,dim_stab0)
flag=list('dt',dim_inst_unob,dim_instb0,dim_unobs)
flag=list('on',nr,ng,ng0,nv,rk)
flag=list('ui',nw,nwu,nwui,nwuis,rk)
```

A completa descrição dos `Sys` é dada no código da função `ssrand` (em `SCIDIR/macros/util`). Por exemplo, com `flag=list('co',dim_cont_subs)` um sistema não-controlável é retornado e `dim_cont_subs` é a dimensão do subespaço controlável de `Sys`. Os strings `'co','uo','ncno','st','dt','on','ui'` significam "controlável", "inobservável", "não-controlável-não-observável", "estabilizável", "detectável", "anulador-de-saída", "saída-desconhecida".

# Exemplos

```
//flag=list('st',dim_cont_subs,dim_stab_subs,dim_stab0)
```

```
//dim_cont_subs<=dim_stab_subs<=dim_stab0
//par (A,B) U-similar a:
//    [*,*,*,*;     [*;
//    [0,s,*,*;     [0;
//A=  [0,0,i,*;   B=[0;
//    [0,0,0,u]     [0]
//
// (A11,B1) controlável  s=matriz estável i=matriz neutra u=matriz instável
[Sl,U]=ssrand(2,3,8,list('st',2,5,5));
w=ss2ss(Sl,inv(U)); //desfaz a mudança aleatória de base=> forma como acima
[n,nc,u,sl]=st_ility(Sl);n,nc
```

# Ver Também

syslin

# Name

sub2ind — converte matriz de valores subescrtos para índice linear

```
I = sub2ind(dims,i1,i2,...)
J = sub2ind(dims,Mi)
```

# Parâmetros

dims
: vetor, as dimensões da matriz

i1,i2,...
: os arrays de valores subescritos (de mesma forma que a matriz `I`)

Mi
: matriz cujas colunas contém os valores subescritos.

I
: o array de índice linear

# Descrição

`sub2ind` é usado para determinar o índice único equivalente que corresponde ao dado conjunto de valores subescritos. `I = sub2ind(dims,i1,i2,..)` retorna o índice linear equivalente aos subescritos das linha, coluna, ... nos arrays `i1`, `i2`,.. para uma matriz de tamanho `dims`. Neste caso, `i1`, `i2`,.. devem ter o mesmo formato e o resultado `I` tem a mesma forma da matriz. `I = sub2ind(dims,Mi)` retorna o índice linear equivalente aos subescritos nas colunas da matriz `Mi` para uma matriz com tamanho `dims`. Neste caso `I` é um vetor coluna.

# Exemplos

```
i=[1 2 1 1 2 1 1];
j=[1 2 3 1 2 3 3];
k=[1 2 1 2 1 2 1];
sub2ind([2,3,2],i,j,k)

sub2ind([2,3,2],[i',j',k'])
```

# Ver Também

ind2sub, extraction, insertion

# Autor

Serge Steer, INRIA

# Name

sum — soma (soma linha, soma coluna) de entradas de um vetor ou matriz

```
y=sum(x)
y=sum(x,'r') ou y=sum(x,1)

y=sum(x,'c') ou y=sum(x,2)

y=sum(x,'m')
```

# Parâmetros

x
: vetor ou matriz (de reais, complexos, esparso(a) ou de polinômios)

y
: escalar ou vetor

# Descrição

Para um vetor ou matriz `x`, `y=sum(x)` retorna no escalar `y` a soma de todas as entradas de `x`.

`y=sum(x,'r')` (ou, equivalentemente, `y=sum(x,1)`) é a soma linha a linha:: `y(j)= sum(x(:,j))`. `y` é um vetor linha.

`y=sum(x,'c')` (ou, equivalentemente, `y=sum(x,2)`) é a soma coluna a coluna. Retorna em cada entrada do vetor coluna `y` a soma : `y(i)= sum(x(i,:))`)).

`y=sum(x,'m')` é a soma ao longo da primeira dimensão "não-singleton" de `x` (para compatibilidade com Matlab).

# Exemplos

```
A=[1,2;3,4];
trace(A)-sum(diag(A))
sum(A,'c')-A*ones(2,1)
sum(A+%i)
A=sparse(A);sum(A,'c')-A*ones(2,1)
s=poly(0,'s');
M=[s,%i+s;s^2,1];
sum(M),sum(M,2)
```

# Ver Também

cumsum, prod

# Name

sysconv — conversão de sistema

```
[s1,s2]=sysconv(s1,s2)
```

# Parâmetros

s1,s2
: lista (sistemas `syslin` lineares)

# Descrição

Converte `s1` e `s2` para representação comum a fim de que operações de interconexão de sistemas possam ser aplicadas. Função utilitária para peritos. A regra de conversão é dada na seguinte tabela.

"c"
: sistema de tempo contínuo

"d"
: sistema de tempo discreto

n
: sistema amostrado com período de amostragem n

[]
: sistema com domínio de tempo indefinido. Para sistemas mistos, s1 e s2 são postos em representação de estado-espaço.

```
s1\s2  |   "c"     |     "d"     |      n2        |     []     |
-----------------------------------------------------------------
"c"   | nada       |incompatível | c2e(s1,n2)     |  c(s2)     |
-----------------------------------------------------------------
"d"   |incompatível| nada        | e(s1,n2)       |  d(s2)     |
-----------------------------------------------------------------
n1    | c2e(s2,n1) | e(s2,n1)    | n1<>n2 incomp  |  e(s2,n1)  |
      |            |             | n1=n2  nada    |            |
-----------------------------------------------------------------
[]    | c(s1)      | d(s1)       | e(s1,n2)       |  nada      |
-----------------------------------------------------------------
```

Com o seguinte significado:

n1,n2
: período de amostragem

c2e(s,n)
: o sistema de tempo contínuo s é transformado em um sistema amostrado com período de amostragem n.

c(s)
: conversão para tempo contínuo (domínio de tempo é `"c"`)

d(s)
: conversão para tempo discreto (domínio de tempo é `"d"`)

e(s,n)
: conversão para sistema amostrado com período `n`

# Exemplos

```
s1=ssrand(1,1,2);
s2=ss2tf(s1);
[s1,s2]=sysconv(s1,s2);
```

# Ver Também

syslin, ss2tf, tf2ss

# Name

sysdiag — conexão de sistemas diagonais em blocos

```
r=sysdiag(a1,a2,...,an)
```

# Descrição

Retorna um sistema diagonal em blocos feito de subsistemas postos na diagonal principal.

ai
: subsistemas (i.e. ganhos, ou sistema linear em forma de espaço de estados ou de transferência)

Usado em particular para interconexões de sistemas.

# Observação

No máximo 17 argumentos.

# Exemplos

```
s=poly(0,'s')
sysdiag(rand(2,2),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])
sysdiag(tf2ss(1/s),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])
```

# Ver Também

brackets, insertion, feedback

# Name

syslin — definição de sistemas lineares

```
[sl]=syslin(dom,A,B,C [,D [,x0] ])
[sl]=syslin(dom,N,D)
[sl]=syslin(dom,H)
```

# Parâmetros

dom
: string (`'c'`, `'d'`), ou `[]` ou um escalar.

A,B,C,D
: matrizes em representação de espaço de estados (`D` opcional com valor padrão matriz nula). para sistemas impróprios, `D` é uma matriz de polinômios.

x0
: vetor (estado inicial; valor padrão é `0`)

N, D
: matrizes de polinômios

H
: matriz de razões de polinômios ou representação de espaço de estados linear

sl
: tlist (lista "`syslin`") representando o sistema linear

# Descrição

`syslin` define um sistema linear como uma lista e verifica a consistência dos dados.

`dom` especifica o domínio de tempo do sistema e pode ter um dos seguintes valores:

`dom='c'` para um sistema de tempo contínuo, `dom='d'` para um sistema de tempo discreto, `n` para um sistema amostrado com período de amostragem `n` (em segundos).

`dom=[]` se o domínio de tempo é indefinido.

Representação em espaço de estados:

```
sl=syslin(dom,A,B,C [,D [,x0] ])
```

Representa o sistema :

```
      s x  = A*x + B*u
        y  = C*x + D*u
      x(0) = x0
```

A saída de `syslin` é uma lista da seguinte forma: `sl=tlist(['lss','A','B','C','D','X0','dt'],A,B,C,D,x0,dom)` . Note que `D` pode ser uma matriz de polinômios (sistemas impróprios).

Representação de matriz de transferência:

```
sl=syslin(dom,N,D)
sl=syslin(dom,H)
```

A saída de `syslin` é uma lista da seguinte forma: `sl=tlist(['r','num','den','dt'],N,D,dom)` ou `sl=tlist(['r','num','den','dt'],H(2),H(3),dom)`.

Sistemas lineares definidos como `syslin` podem ser manipulados como matrizes usuais (concatenação, extração, transposição, multiplicação, etc.) ambos em estado-espaço ou representação de transferência.

A maior parte das funções de controle de estado-espaço recebem uma lista `syslin` como entrada, ao invés de quatro matrizes definindo o sistema.

# Exemplos

```
A=[0,1;0,0];B=[1;1];C=[1,1];
S1=syslin('c',A,B,C)   //Definição de sistema linear
S1("A")    //Exibição da matriz A
S1("X0"), S1("dt") // Exibição de X0 e domínio de tempo
s=poly(0,'s');
D=s;
S2=syslin('c',A,B,C,D)
H1=(1+2*s)/s^2, S1bis=syslin('c',H1)
H2=(1+2*s+s^3)/s^2, S2bis=syslin('c',H2)
S1+S2
[S1,S2]
ss2tf(S1)-S1bis
S1bis+S2bis
S1*S2bis
size(S1)
```

# Ver Também

tlist, lsslist, rlist, ssrand, ss2tf, tf2ss, dscr, abcd

# Name

tan — tangente

```
[t]=tan(x)
```

# Parâmetros

x
: vetor ou matriz

t
: vetor ou matriz

# Descrição

Os elementos de `t` são as tangentes dos elementos de `x`.

# Exemplos

```
x=[1,%i,-1,-%i]
tan(x)
sin(x)./cos(x)
```

# Ver Também

atan, tanm

# Name

tand — tangente com o argumento em graus

```
t=tand(x)
```

# Parâmetros

x
: vetor ou matriz de reais

t
: vetor ou matriz de reais

# Descrição

Os elementos de `t` são as tangentes dos elementos de `x`

# Exemplos

```
mod=ieee();ieee(2);
x=[0,30 45 60 90 360];
tand(x)
ieee(mod)
```

# Ver Também

atand, tan

# Name

tanh — tangente hiperbólica

```
t=tanh(x)
```

# Descrição

Os elementos de `t` são as tangentes hiperbólicas dos elementos de `x`

# Exemplos

```
x=[1,%i,-1,-%i]
tanh(x)
sinh(x)./cosh(x)
```

# Ver Também

atanh, tan, tanhm

# Name

tanhm — tangente hiperbólica da matriz (matriz tangente hiperbólica)

```
t=tanhm(x)
```

# Parâmetros

x,t
: matrizes quadradas de reais ou complexos

# Descrição

tanhm é a tangente hiperbólica da matriz x (matriz tangente hiperbólica de x).

# Exemplos

```
A=[1,2;3,4];
tanhm(A)
```

# Ver Também

tan, tanm, expm, sinm, cosm, atanhm

# Name

tanm — tangente da matriz (matriz tangente)

```
[t]=tanm(x)
```

# Parâmetros

x
: matriz quadrada de reais ou complexos

t
: matriz quadrada

# Descrição

`tanm(x)` é a tangente da matriz x (matriz tangente de x).

# Exemplos

```
A=[1,2;3,4];
tanm(A)
```

# Ver Também

tan, expm, sinm, atanm

# Name

toeplitz — matriz de Toeplitz

```
A=toeplitz(c [,r])
```

# Parâmetros

a,c,r
: matrizes de constantes, polinômios ou strings

# Descrição

Retorna a matriz de Toeplitz cuja primeira linha é `r` e a primeira coluna é `c`. `c(1)` deve ser igual a `r(1)`. `toeplitz(c)` retorna a matriz de Toeplitz simétrica.

# Exemplos

```
A=toeplitz(1:5);

T=toeplitz(1:5,1:2:7);T1=[1 3 5 7;2 1 3 5;3 2 1 3;4 3 2 1;5 4 3 2];
T-T1

s=poly(0,'s');
t=toeplitz([s,s+1,s^2,1-s]);
t1=[s,1+s,s*s,1-s;1+s,s,1+s,s*s;s*s,1+s,s,1+s;1-s,s*s,1+s,s]
t-t1

t=toeplitz(['1','2','3','4']);
t1=['1','2','3','4';'2','1','2','3';'3','2','1','2';'4','3','2','1']
```

# Ver Também

matrix

# Name

trfmod — exibição de zeros e pólos

```
[hm]=trfmod(h [,job])
```

# Descrição

Visualiza a estrutura pólo-zero de uma função de transferência SISO `h` .

job='p'
: visualização de polinômios (padrão)

job='f'
: visualização de freqüências naturais e amortecimento

Simplificação interativa de `h`. `trfmod` abre uma janela de diálogo.

# Ver Também

poly, simp

# Name

trianfml — triangularização simbólica

```
[f [,sexp]]=trianfml(f [,sexp])
```

# Descrição

Triangularização simbólica da matriz `f` ; a triangularização é executada por transformações elementares de linha; `sexp` é um conjunto de expressões comuns aramazenadas pelo algoritmo.

# Exemplos

```
A=['1','2';'a','b']
W=trianfml([A,string(eye(2,2))])
U=W(:,3:4)
a=5;b=6;
A=evstr(A)
U=evstr(U)
U*A
evstr(W(:,1:2))
```

# Ver Também

addf, mulf, solve, trisolve

# Name

tril — parte triangular inferior de uma matriz

```
tril(x [,k])
```

# Parâmetros

x
: matriz (de reais, complexos, polinômios ou razões de polinômios)

k
: inteiro (valor padrão 0)

# Descrição

Parte triangular inferior de uma matriz. `tril(x,k)` é formada pelas entradas abaixo da k-ésima diagonal : k>0 (acima da diagonal superior) e k<0 (diagonais abaixo da diagonal principal).

# Exemplos

```
s=poly(0,'s');
tril([s,s;s,1])
tril([1/s,1/s;1/s,1])
```

# Ver Também

triu, ones, eye, diag

# Name

trisolve — solucionador simbólico de sistemas lineares

```
[x [,sexp]] = trisolve(A,b [,sexp])
```

# Parâmetros

A,b
: matrizes de strings

# Descrição

Resolve simbolicamente `A*x =b` , `A` sendo assumida como triangular superior.

`sexp` é um vetor de subexpressões comuns em `A, b, x`.

# Exemplos

```
A=['x','y';'0','z'];b=['0';'1'];
w=trisolve(A,b)
x=5;y=2;z=4;
evstr(w)
inv(evstr(A))*evstr(b)
```

# Ver Também

trianfml, solve

# Autores

F.D, S.S

# Name

triu — triângulo superior da matriz

# Descrição

Triângulo superior da matriz. Ver tril.

# Exemplos

```
s=poly(0,'s');
triu([s,s;s,1])
triu([1/s,1/s;1/s,1])
```

# Ver Também

tril, ones, eye, diag

# Name

typeof — tipo do objeto

```
[t]=typeof(object)
```

# Parâmetros

object
: objeto Scilab

t
: string

# Descrição

`t=typeof(object)` retorna um dos seguintes strings:

"constant"
: se o objeto é uma matriz constante de reais ou complexos

"polynomial"
: se o objeto é uma matriz de polinômios

"function"
: se o objeto é uma função (código Scilab).

"handle"
: se o objeto é um manipulador ("alça")

"string"
: se o objeto é uma matriz de strings

"boolean"
: se o objeto é uma matriz de valores booleanos

"list"
: se o objeto é uma lista

"rational"
: se o objeto é a é uma martriz de razões de polinômios (matriz de transferência)

"state-space"
: se o objeto é um modelo de espaço de estados (ver syslin).

"sparse"
: se o objeto é uma matriz (de reais) esparsa.

"boolean sparse"
: se o objeto é uma matriz de valores booleanos esparsa.

"hypermat"
: se o objeto é uma hipermatriz (array N-dimensional N >=3).

"st"
: se o objeto é um array de estrutura

"ce"
: se o objeto é um array de células

the first string in the first list entry
: se o objeto é um tlist ou mlist

"fptr"
: se o objeto é intrínseco ao Scilab (código C ou Fortran).

"pointer"
: se o objeto é um ponteiro (ver lufact).

"size implicit"
: se o objeto é um polinômio de tamanho implícito para indexação.

# Exemplos

```
typeof(1)
typeof(poly(0,'x'))

typeof(1/poly(0,'x'))
typeof(%t)

w=sprand(100,100,0.001);
typeof(w)
typeof(w==w)

deff('y=f(x)','y=2*x');
typeof(f)

L=tlist(['V','a','b'],18,'Scilab');
typeof(L)
```

# Ver Também

type, strings, syslin, poly

# Name

union — extrai componentes da união de um vetor

```
[v [,ka, kb] ] = union(a,b)
[v [,ka, kb] ] = union(a,b,orient)
```

# Parâmetros

a
: vetor ou matriz de números ou strings

b
: vetor ou matriz de números ou strings

orient
: flag com valores possíveis : 1 ou "r", 2 ou "c".

v
: vetor linha ou matriz de números ou strings

ka
: vetor linha de inteiros

kb
: vetor linha de inteiros

# Descrição

`union(a,b)` retorna um vetor linha ordenado que retém as entradas únicas de `[a(:);b(:)]`.

`union(a,b,"r")` ou `union(a,b,1)`retorna a matriz formada pela união das linhas únicas de `a` e `b` em ordem lexicográfica ascendente. Neste caso, as matrizes `a` e `b` devem ter o mesmo número de colunas.

`union(a,b,"c")` ou `union(a,b,2)`retorna a matriz formada pela união das colunas únicas de `a` e `b` em ordem lexicográfica ascendente. Neste caso, as matrizes `a` e `b` devem ter o mesmo número de colunas.

`[v,ka,kb]=union(a,b)` também retorna vetores de índices `ka` e `kb` tais que `v` é uma combinação ordenada das entradas `a(ka)` e `b(kb)`.

# Exemplos

```
A=round(5*rand(10,1));
B=round(5*rand(7,1));

union(A,B)
[N,ka,kb]=union(A,B)

union('a'+string(A),'b'+string(B))
```

# Ver Também

unique, gsort

# Name

unique — extrai componentes únicos de um vetor ou de matrizes

```
[N [,k]]=unique(M)
[N [,k]]=unique(M ,orient)
```

# Parâmetros

M
: vetor ou matriz de números ou strings

orient
: flag com valores possíveis : 1 ou "r", 2 ou "c"

N
: vetor ou matriz de números ou strings

k
: vetor de inteiros

# Descrição

`unique(M)` retorna um vetor que retém as entradas únicas de `M` em ordem ascendente.

`unique(M,"r")` ou `unique(M,1)`retorna as linhas únicas de `M` em ordem lexicográfica ascendente.

`unique(M,"c")` ou `unique(M,2)`retorna as linhas únicas de `M` em ordem lexicográfica ascendente.

Se requerido, o argumento de saída, `k` contém a posição das primeiras entradas únicas encontradas.

# Exemplos

```
M=round(2*rand(20,1));

unique(M)
[N,k]=unique(M)

unique(string(M))
[N,k]=unique(string(M))

A = [0,0,1,1;
     0,1,1,1;
     2,0,1,1;
     0,2,2,2;
     2,0,1,1;
     0,0,1,1];
T='x'+string(A);

//linhas únicas

[m,k]=unique(A,'r')
unique(T,'r')
```

```
//colunas únicas
[m,k]=unique(T,'c')
unique(A,'c')
```

# Ver Também

union, intersect, gsort, lex_sort

# Name

vectorfind — acha, em uma matriz, linhas ou colunas que coincidem com um vetor

```
ind = vectorfind(m,v,job)
```

# Parâmetros

m
: uma matriz de qualquer tipo

v
: um vetor de qualquer tipo

job
: um string indicador com valores possíveis "r" para procura de linhas coincidentes ou "c" para procura de colunas coincidentes

ind
: vetor-linha contendo os índices das linhas ou colunas coincidentes

# Descrição

Acha, em uma matriz, as linhas ou colunas que coincidem com um dado vetor.

# Exemplos

```
alr=[1,2,2;
     1,2,1;
     1,1,2;
     1,1,1;
     1,2,1];
ind = vectorfind(alr,[1,2,1],'r')
ind = vectorfind(string(alr),string([1,2,1]),'r')
```

# Ver Também

find, gsort

# Autores

R. Nikoukhah, S. Steer INRIA

# Name

zeros — matriz feita de zeros

```
y=zeros()
y=zeros(x)
y=zeros(m1,m2,..)
```

# Parâmetros

x,y
: matrizes

m1, m2,..
: inteiros

# Descrição

Cria uma matriz de zeros (é o mesmo que `0*ones`).

zeros(m1,m2)
: para uma matriz `(m1,m2)` .

zeros(m1,m2,..,mn)
: cria uma matriz `(m1,m2,..,mn)` preenhida com zeros

zeros(A)
: para uma matriz de mesmo tamanho que `A`.

zeros(3)
: é `zeros(a)` com `a=3` é uma matriz 3x3!

zeros()
: retorna um único zero

Se `x` é uma lista `syslin` (sistema linear em forma de espaço de estados ou transferência), `zeros(x)` também é válido e retorna uma matriz nula.

# Exemplos

```
zeros(3)
zeros(3,3)
zeros(2,3,2)
```

# Ver Também

eye, ones, spzeros

# Name

& — logical and operator

```
A&B
```

# Description

A&B gives the element-wise logical and of the booleans matrices A and B .A and B must be matrices with the same dimensions or one from them must be a single boolean.

# See Also

not, and, or operator (|)

# Nom

csgn — Returns the sign of a vector of real of complex values.

```
s = csgn(z)
```

# Parameters

z

The vector of values on which we want to compute the sign.

s

If the real part is not equal to zero:

- +1 if the real part of an element if positive
- -1 if the real part of an element if negative

If the real part is equal to zero:

- +1 if the imaginary part of an element if positive
- -1 if the imaginary part of an element if negative

if the element is equal to zero, then returns %nan

# Description

Returns the sign of a vector of real of complex values.

# Examples

```
A = [1 1+%i 0 -1 1-%i -1-%i];
disp(csgn(A))
```

# Authors

Y. Collette

# Name

isvector — check if a variable is a vector

```
t=isvector(x)
```

# Parameters

x
: vector or matrix

t
: a boolean

# Description

`isvector(x)` returns true if `x` is a vector (one of its dimension is different from 1).

# Examples

```
isvector(ones(10,1))
isvector(1)
```

# Name

| — logical or operator

```
A|B
```

# Description

`A|B` gives the element-wise logical `or` of the booleans matrices `A` and `B` .`A` and `B` must be matrices with the same dimensions or one from them must be a single boolean.

# Examples

```
[%t %t %f] | [%f %t %t]
[%t %t %f] | %f
```

# See Also

or, and, and operator (&), not, find

# Parte IV. Funções

# Nom

add_profiling — adiciona instruções de profiling (análise de performance) a uma função

```
add_profiling(funname)
```

# Parâmetros

funname
: string, o nome da função

# Descrição

`add_profiling(funname)` adiciona instruções de profiling a uma função de nome `funname`. Então, quando esta função é chamada, o número de chamadas e o tempo gasto é armazenado para cada linha da função.

# Exemplos

```
function x=foo(a,n)
 x=0;
 for i=1:n
   if x<10 then
     x=x+a
   else
     x=x+1
   end
 end
 x=x^2+1
endfunction

add_profiling("foo")
foo(0.1,100) //executando a função
profile(foo) //extraindo informação de profile
```

# Ver Também

profile, plotprofile, remove_profiling, reset_profiling

# Autor

Serge Steer, INRIA

# Funções Utilizadas

Esta função utiliza as funções Scilab `bytecode` e `walkbytecode`

# Nom

bytecode — dada uma função, retorna "bytecode" (código de bytes) da função em um array Scilab e vice-versa.

```
x = bytecode(f)
f= bytecode(X)
```

# Parâmetros

f
: função Scilab

x
: vetor linha int32

# Descrição

`x = bytecode(f)` retorna o "bytecode" da função `f` no array Scilab de inteiros `x`.

`f = bytecode(x)` retorna em `f` a função associada ao "bytecode" dado no array Scilab de inteiros `x`. AVISO: a validade de `x` não é verificada.

# Observação

O bytecode da função Scilab evoluirá drasticamente no futuro, então o uso dessa função deve se restringir à manipulação de instruções de profiling.

# Exemplos

```
function a=foo(),a=sin(3),endfunction
bytecode(foo)
```

# Ver Também

add_profiling, bytecodewalk, macr2lst, macr2tree

# Autor

Serge Steer, INRIA

# Nom

bytecodewalk — caminha no bytecode (código de bytes) da função aplicando transformação

```
c1 = bytecodewalk(code,query,job)
```

# Parâmetros

code
: vetor int32: array de bytecode de entrada

query
: inteiro, o opcode (código de operação) a ser procurado

job
: a operação a ser realizada, requerida para

c1
: vetor int32: array de bytecode de saída

# Descrição

caminha no bytecode da função aplicando transformação

# Ver Também

bytecode

# Autor

Serge Steer INRIA

# Name

deff — definição on-line de função

```
deff('[s1,s2,...]=newfunction(e1,e2,....)',text [,opt])
```

# Parâmetros

e1,e2,...,
: variáveis de entrada

s1,s2,...,
: variáveis de saída

text
: matriz de strings

opt
: string opcional

  'c'
  : a função é "compilada" para ser eficiente (padrão)

  'p'
  : a função é compilada e preparada para profiling (ver profile)

  'n'
  : a função não é "compilada"

# Descrição

`deff` pode ser usada para definir funções de seqüências de instruções escritas em strings de textos. Objeto função resultante tem as mesmas propriedades que qualquer outra função definida em um arquivo de texto e carregada através de `exec` ou `exec`.

Aspas em instruções (delimitando strings ou indicando transposição de matrizes) devem ser dobradas para serem interpretadas corretamente (ver quote). Isto pode tornar a escrita um tanto estranha. Uma alternativa em tais casos é definir funções em arquivos, como de uso, para carregá-las no Scilab através de `exec` (com a opção `'n'`) e utilizar `sci2exp` para uma impressão das instruções `deff` correspondentes.

# Exemplos

```
deff('[x]=myplus(y,z)','x=y+z')

deff('[x]=mymacro(y,z)',['a=3*y+1'; 'x=a*z+y'])
```

# Ver Também

comp, exec, function, profile

# Name

exec — execução de arquivo script ("script" significa roteiro)

```
exec(path [,mode])
exec(fun [,mode])
ierr=exec(path,'errcatch' [,mode])
ierr=exec(fun,'errcatch' [,mode])
```

# Parâmetros

path
: string, o endereço do arquivo script

mode
: escalar inteiro, o modo de execução (ver abaixo)

fun
: uma função do Scilab

ierr
: inteiro, 0 ou número de erro

# Descrição

`exec(path [,mode])` executa seqüencialmente as instruções contidas no arquivo fornecido por `path` com um modo de execução opcional `mode` .

Os casos diferentes para `mode` são :

0 : o valor padrão

-1 : nada é impresso

1 : eco de cada linha de comando

2 : prompt --> é impresso

3 : ecos + prompts

4 : pára antes de cada prompt. A execução retorna após cada retorno de carro.

7 : paradas + prompts + ecos : útil para demos.

`exec(fun [,mode])` executa a função `fun` como um script: sem argumentos de entrada ou saída nem ambientes de variáveis específicos. Esta forma é mais eficiente, porque o código script pode se pré-compilado (ver comp). Este método para avaliação de scripts permite armazenar scripts como funções em bibliotecas.

Se um erro é encontrado durante a execução, se o flag 'errcatch' estiver presente `exec` não imprime mensagem de erro, aborta execução de instruções e retorna com `ierr` e igual ao número de erro. Se o flag 'errcatch' não estiver presente, a manipulação de erros padrão é utilizada.

# Observação

Arquivos executáveis (exec files) podem agora ser usados para definir funções "inline" utilizando a sintaxe de definição (ver function).

# Exemplos

```
// criando um arquivo script
mputl('a=1;b=2',TMPDIR+'/meuscript')
// executando-o
exec(TMPDIR+'/meuscript')
whos -name "a "

// criando uma função
deff('y=foo(x)','a=x+1;y=a^2')
clear a b
// chamando a função
foo(1)
// a é uma variável criada no ambiente da função foo
//    ela é destruída quando foo retorna
whos -name "a "

x=1 //criando x para torná-la conhecida pelo script foo
exec(foo)
// a e y são criadas no ambiente corrente
whos -name "a "
```

# Ver Também

exec, execstr, evstr, comp, mode, chdir, pwd

# Name

execstr — executa código Scilab em strings

```
execstr(instr)
ierr=execstr(instr,'errcatch' [,msg])
```

# Parâmetros

instr
: vetor de strings, instrução Scilab a ser executada.

ierr
: inteiro, 0 ou número de erro.

msg
: string com valores `'m'` ou `'n'`. O padrão é `'n'`.

# Descrição

Executa as instruções Scilab fornecidas pelo argumento `instr`.

Note que instr não deve fazer uso de marcas de continuação(..)

Se o flag 'errcatch' não estiver presente, a manipulação de erros ocorre de maneira usual.

Se o flag 'errcatch' for ajustado, e um erro for encontrado enquanto são executadas as instruções definidas em `instr`, `execstr` não imprime uma mensagem de erro, mas aborta a execução das instruções `instr` (no ponto onde o erro ocorreu) e retorna com `ierr` igual ao número de erro. Neste caso a exibição da mensagem de erro é controlada pela opção `msg` :

"m"
: a mensagem é exibida e registrada.

"n"
: nenhuma mensagem de erro é exibida, mas a mensagem de erro é registrada (ver `lasterror`). Este é o padrão.

ierr= execstr(instr,'errcatch') pode manipular erros sintáticos. Isto é útil para a avaliação de uma instrução obtida por uma pegunta ao usuário.

# Exemplos

```
execstr('a=1') // ajusta a=1.
execstr('1+1') // faz nada (enquanto evstr('1+1') retorna 2)

execstr(['if %t then';
         '  a=1';
         '  b=a+1';
         'else'
         ' b=0'
         'end'])

execstr('a=zzzzzzz','errcatch')
execstr('a=zzzzzzz','errcatch','m')
```

```
//erros de sintaxe
execstr('a=1?02','errcatch')
lasterror(%t)

execstr('a=[1 2 3)','errcatch')
lasterror(%t)
```

# Ver Também

evstr, lasterror, error, try

# Name

fun2string — gera definição ASCII de uma função Scilab

```
txt=fun2string(fun,name)
```

# Parâmetros

fun
: variável do tipo função

name
: string, o nome da função gerada

txt
: vetor coluna de strings, o texto fornecendo as instruções Scilab

# Descrição

Dado um pseudo-código de função Scilab carregada fun2string permite gerar novamente o código. O código gerado é identado e embelezado.

O mecanismo é similar, mas mais simples que o de mfile2sci. Também pode ser adaptado para traduções de sintaxe.

# Exemplos

```
txt=fun2string(asinh,'foo');
write(%io(2),txt,'(a)')
```

# Ver Também

exec, edit, macrovar

# Name

function — abre definição de função
endfunction — encerra definição de função

# Descrição

```
function <lhs_arguments>=<function_name><rhs_arguments>
<statements>
endfunction
```

Onde

<function_name>
é o nome da função

<rhs_arguments>
é a lista de argumentos de entrada. Pode ser:

- uma seqüência separada por vírgula de nomes de variáveis encerrada por parênteses, como `(x1,...,xm)`. O último nome de variável pode ser a palavra-chave `varargin` (ver varargin)
- a seqüência `()` ou nada, se a função não possui argumentos de entrada.

<lhs_arguments>
é a lista de argumentos de saída. Pode ser:

- uma seqüência separada por vírgula de nomes de variáveis encerrada por colchetes, como `[y1,...,yn]`. O último nome de variável pode ser a palavra-chave `varargout` (ver varargout)
- a seqüência `[]` ,se a função não possui argumentos de saída. neste caso, a sintaxe também, pode ser: `function <function_name><rhs_arguments>`

<statements>
conjunto de instruções Scilab (declarações). Esta sintaxe pode ser usada para definir funções (ver functions) inline ou em arquivo script (ver exec). Para compatibilidade com versões Scilab antigas, as funções podem ser definidas em um arquivo script contendo apenas definições de funções podem ser "carregadas" no Scilab através da função `exec`.

A seqüência `function <lhs_arguments>=<function_name><rhs_arguments>` não pode ser separada em várias linhas. Esta seqüência pode ser seguida por declarações na mesma linha, se uma vírgula ou ponto-e-vírgula for adicionado ao seu fim.

Definições de funções podem ser aninhadas.

# Exemplos

```
//definição inline (ver functions)
function [x,y]=myfct(a,b)
x=a+b
y=a-b
endfunction

[x,y]=myfct(3,2)
```

```
//uma definição de função de uma linha
function y=sq(x),y=x^2,endfunction

sq(3)

//definição de função aninhada
function y=foo(x)
a=sin(x)
function y=sq(x), y=x^2,endfunction
y=sq(a)+1
endfunction

foo(%pi/3)

// definição em um arquivo script (ver exec)
exec SCI/modules/elementary_functions/macros/asinh.sci;
```

# Ver Também

functions, exec, exec

# Name

functions — procedimentos Scilab e objetos Scilab

# Descrição

Funções são procedimentos Scilab ("macro", "função" e "procedure" possuem o mesmo significado).

# Definição de Função

Geralmente, elas são definidas em arquivos com um edito e carregadas no Scilab através da função exec ou através de uma biblioteca (ver lib ou genlib). Mas também podem ser definidas on-line (ver deff ou function. Uma função é definida por dois componentes:

- uma parte "definição de sintaxe" como segue:

```
function [y1,...,yn]=foo(x1,...,xm)
function [y1,...,yn,varargout]=foo(x1,...,xm,varargin)
```

- uma seqüência de instruções Scilab.

A linha de "definição de sintaxe" fornece a sintaxe de chamamento "completa" para esta função. As variáveis `yi` são variáveis de saída calculadas em função das variáveis de entrada `xi` e variáveis existentes no Scilab quando a função é executada.

# Chamada de Função

- A sintaxe de chamamento de função usual é `[y1,...,yn]=foo(x1,...,xm)`. Listas de argumentos de entrada ou saída mais curtos que as da definição podem ser usadas. Em tais casos, apenas as primeiras variáveis da esquerda para direita são usadas ou definidas.

  A função argn pode ser utilizada para se o número total real de argumentos de chamada.

- É possível definir funções com número máximo indeterminado de argumentos de entrada ou saída. Isto pode ser feito utilizando-se as palavras-chave varargin e varargout keywords. Veja os links para detalhes.

- Também é possível utilizar "argumentos nomeados" para especificar argumentos de entrada: suponha que a função `fun1 seja` definida como `function y1=fun1(x1,x2,x3)` , então pode ser chamada com uma sintaxe como `y=fun1(x1=33,x3=[1 2 3])`. Dentro de `fun1` x2 será indefinida.

  Também pode ser chamada com sintaxe como `y=fun1(x1=33,y='foo')`. Em tal caso, a variável `y` estará disponível no contexto da função `fun1`. Note que o número máximo de argumentos deve ser menor que ou igual ao número de argumentos de entrada fornais utilizados na parte de sintaxe da função.

  É possível buscar por variáveis definidas através da função exists.

- Quando uma função não possui argumento de lado esquerdo e é chamada apenas com argumentos strings, a sintaxe de chamamento pode ser simplificada:

```
fun('a','toto','a string')
```

  is equivalent to:

```
fun a toto 'a string'
```

# Miscelânea

Funções Scilab são objetos (com números de tipo 13 ou 11). Elas podem ser manipuladas (construídas, salvas, carregadas, passadas como argumentos,..) como outros tipos de variáveis.

Coleções de funções podem ser reunidas em bibliotecas. Funções que começam pelo sinal `%` (ex: `%foo`) são geralmente utilizadas para operações de sobrecarga (ver overloading ) ou são funções para novos tipos de dados.

# Exemplos

```
//definição inline (ver function)
function [x,y]=myfct(a,b)
  x=a+b
  y=a-b
endfunction

[x,y]=myfct(3,2)

//definição inline (ver deff)
deff('[x,y]=myfct(a,b)',['x=a+b';
                         'y=a-b'])
// definição em um arquivo ASCI (ver exec)
exec SCI/modules/elementary_functions/macros/asinh.sci;
```

# Ver Também

function, deff, exec, comp, lib, getd, genlib, exists, varargin, varargout

# Name

genlib — constrói biblioteca a partir de funções em um diretório

```
genlib(lib_name [[,dir_name, [ Force [,verb [,Names]]]])
genlib(lib_name [,path=dir_name] [,verbose=verb] [,force=Force] [,names=Names])
```

# Parâmetros

lib_name:
: string. O nome da variável biblioteca a ser (re)criada.

dir_name:
: string. O nome do diretório onde se deve procurar arquivos `.sci`.

Force
: booleano (o padrão é `%f`). Para forçar recompilação de arquivos .sci, ajuste-o para `%t`.

verb
: booleano (o padrão é `%f`). Para obter informações ajuste-o para `%t`.

Names
: vetor de strings, os nomes das funções a serem inclusas na bibilioteca. Por padrão, todos os arquivos sci são levados em conta.

# Descrição

Para cada arquivo `.sci` em `dir_name` (ou apenas para aqueles especificados pelo argumento `Names`), `genlib` executa um `exec` e salva as funções no arquivo `.bin` correspondente. O arquivo `.sci` só pode conter instruções Scilab. Se um arquivo `.bin` for mais novo que o arquivo `.sci` assicuadi, `genlib` não traduz nem salva o arquivo.

O comportamento padrão pode ser modificado se `force` for fornecido e ajustado para `%t`. Neste último caso, a recompilação é sempre feita para cada arquivo `.sci`.

Quando todos os arquivos `.sci` tiverem sido processados, `genlib` cria uma biblioteca nomeada `lib_name` e a salva arquivo `lib` em `dir_name`. Se a variável Scilab `lib_name` não estiver protegida (ver predef), esta variável é atualizada.

Se `verb` estiver ajustado para `%t` informações são exibidas durante o processo de construção.

Seo argumento `dir_name` não for fornecido e se a variável Scilab `lib_name` existe e é uma variável dir_name é tomado como sendo igual a o endereço de bibilioteca `lib_name` (modo de atualização).

# Restrições

Scilab assume implicitamente que `foo.sci` define pelo menos uma função de nome `foo`. Se funções subsidiárias estiverem inlusas, elas são tornadas conhecidas ao Scilab apenas depois que função `foo` tiver sido referencida.

# Ver Também

getd, exec, save, lib

# Name

get_function_path — retorna o endereço do arquivo fonte de uma função de biblioteca

```
path=get_function_path(fun_name)
```

# Parâmetros

fun_name
: string, o nome da função

path
: string, o nome absoluto do endereço do arquivo fonte da função (.sci) ou [].

# Descrição

Dadi o nome de uma uma função, get_function_path o nome absoluto do endereço do arquivo fonte da função se a função estiver definida em uma biblioteca Scilab (ver lib) ou [] se nenhum nome corresponde a uma função de biblioteca.

# Exemplos

```
get_function_path('median')
```

# Ver Também

lib, string

# Name

getd — retorna todas as funções definidas em um diretório

```
getd(path)
```

# Parameters

path
: string Scilab. O endereço do diretório

# Descrição

Carrega todos os arquivos `.sci` (contendo funções Scilab) definidos no diretório `path`.

# Exemplos

```
getd('SCI/modules/cacsd/macros')
```

# Ver Também

exec , lib , pwd , chdir

# Name

head_comments — exibe comentários do cabeçalho da função Scilab

```
head_comments(name)
head_comments(name,%paths)
```

# Parâmetros

name
: string, nome da função

%paths
: vetor de strings, endereços onde se deve procurar o arquivo sci

# Descrição

`comments(name)` exibe comentários do cabeçalho da função (como a ajuda do Matlab). Os comentários são lidos do arquivo sci associado. Se `name` for uma função em uma biblioteca, o endereço do arquivo sci é aquele dado pelo endereço da biblioteca (ver `lib`). Se `name` for uma função que não está na biblioteca, um arquivo de nome `name.sci` é procurado nos outros diretórios fornecidos pela variável `%paths`

AVISO: a maior parte das funções Scilab predefinidas não possui comentários de cabeçalho.

# Exemplos

```
head_comments sinc
```

# Ver Também

help

# Autor

Serge Steer, INRIA

# Name

lib — definição de biblioteca

```
xlib = lib('lib-dir')
```

# Parameters

lib-dir
: string

# Descrição

`lib-dir` é um string definindo um diretório que contém arquivos de funções compiladas do Scilab (`.bin`).

Em adição a esses arquivos `lib-dir` deve conter um arquivo chamado `names`, que contém os nomes das funções definidas em `lib-dir`. Com sucesso, todas as funções em `lib-dir` estarão disponíveis dentro do Scilab. Elas são carregadas em demanda quando são chamadas pela primeira vez.

Arquivos binários podem ser criados de dentro do Scilab com o comando `save`.

As bibliotecas padrões do Scilab são definidas utilizando-se `lib` nos subdiretórios `SCIDIR/macros/*`.

Uma variável biblioteca geralmente é salva para carregamento posterior, tanto on-line quanto do arquivo de inicialização específico do usuário (ver startup).

# Restrições

O Scilab assume tacitamente que cada arquivo xxxx.bin define uma variável chamada `xxxx`.

# Exemplos

```
//define some variables
function z = myplus(x, y), z = x + y,endfunction
function z = yourplus(x, y), x = x - y,endfunction
A=1:10;

//create the *.bin files in libdir
libdir=TMPDIR
save(libdir + '/myplus.bin', myplus);
save(libdir + '/yourplus.bin', yourplus);
save(libdir + '/A.bin', A);

//create the name file
mputl(['myplus';'yourplus';'A'],TMPDIR+'/names');

//build the library containing myplus and yourplus
xlib = lib(libdir+'/')

//erase the variables
clear myplus yourplus A

//Automatic loading and execution
myplus(1,2)
```

```
A
```

## Ver Também

library, genlib, save, deff, exec, whereis

# Name

librarieslist — retorna as bibliotecas do Scilab

```
s=librarieslist()
```

# Parâmetros

s
: uma matriz de strings

# Descrição

Retorna em `s` todas as bibliotecas na pilha Scilab.

# Exemplos

```
librarieslist()
```

# Ver Também

libraryinfo

# Name

library — descrição de tipo de dado bibilioteca

# Descrição

Uma bibilioteca é um tipo de dado com número 14. Contém um nome de endereço e um conjunto de nomes. Permite o carregamento automático de variáveis utilizando o seguinte algoritmo:

Suponha o usuário Scilab referencie a variável de nome `foo`. O Scilab procura primeiro se `foo` é o nome de uma primitiva, ou de uma variável já definida. Se naõ for, procura por `foo` seqüencialmente (a mais nova) em toda a biblioteca definida .

Suponha que `foo` pertence ao conjunto de nomes da biblioteca `xlib`. Então o Scilab tenta carregar o arquivo <xlib-path-name>/foo.bin. <xlib-path-name>/foo.bin deve ter sido criado utilizando-se a função `save`.

Bibliotecas geralmente são utilizadas para coleções de funções, mas também podem ser utilizadas para coleções de variáveis Scilab.

Se uma função estiver definida em mais de uma biblioteca, o algoritmo de busca padrão carrega aquela contida na mais nova. É possível forçar o uso de uma biblioteca específica utilizando a notação de ponto:

`xlib.foo` carrega a variável `foo` contida em `xlib`, if `foo` for uma função e `xlib.foo(args)` executa as funções.

# Exemplos

```
// elemlib é uma biblioteca predefinida
elementary_functionlib //exibindo o conteúdo da biblioteca
A=rand(3,3);
cosm(A) //carregando cosm e executando-o
whos -name cosm // agora, cosm é uma variável
elementary_functionlib.sinm //carregando sinm da biblioteca
elementary_functionlib.cosm(A) //carregando novamente cosm e executando-o
```

# Ver Também

lib, string, load, save

# Name

libraryinfo — retorna macros e endereço de uma biblioteca Scilab

```
macros = libraryinfo(libraryname)
[macros,path] = libraryinfo(libraryname)
```

# Parâmetros

macros
: uma matriz de strings (todas as funções principais da biblioteca)

path
: um string (endereço da biblioteca)

libraryname
: string (nome da biblioteca)

# Descrição

Retorna nomes de funções e o endereço de uma biblioteca Scilab. Os nomes de funções são aqueles que correspondem aos nomes de arquivo associados .sci ou .bin. Os outros são funções subsidiárias.

# Exemplos

```
[m,p]=libraryinfo('corelib')
```

# Ver Também

librarieslist

# Name

listfunctions — propriedades de todas as funções no espaço de trabalho

```
[flist,compiled,profilable,called] = listfunctions([scope])
```

# Parâmetros

scope
: string, "local" (padrão) ou "global"

flist
: array de strings, names nomes de todas as variáveis funções especificadas no espaço de trabalho

compiled
: array de booleanos, verdadeiro se o elemento correspondente de flist for do tipo 13

profilable
: array de booleanos, verdadeiro se o elemento correspondente de flist for do tipo 13, e adicionalmente informações sobre profiling forem encontradas no pseudo-código da função.

called
: array uint32, número de vezes que o elemento correspondente de flist foi chamado (não-zero apenas se a função possuir profiling)

# Descrição

- Esta função verifica todas as variáveis do espaço de trabalho (dadas por who) e coleta aquelas de tipo 11 ou 13; para as últimas, lst=macr2lst(fun) é chamada, de modo a verificar a entrada para magic profiling ao fim da primeira linha de código, i.e. lst(5)(1)=="25".

# Exemplos

```
recompilefunction("asinh","p")
[flist,compiled,profilable,called] = listfunctions();
flist(profilable)
```

# Ver Também

function, exec, deff, comp, fun2string, profile, recompilefunction

# Autor

Enrico Segre

# Bibliografia

http://wiki.scilab.org/Scilab_function_variables%3A_representation%2C_manipulation

# Name

macro — Procedimento Scilab e objeto Scilab

# Descrição

Macros são procedimentos Scilab ("macro", "função" e "procedimento" possuem o mesmo significado). Geralmente, eles são definidos em arquivos com um editor e carregados no Scilab com `exec` ou através de uma biblioteca.

Também podem ser definidos on-line (ver `deff`). Um arquivo que contém uma macro deve começar como segue:

```
function [y1,...,yn]=foo(x1,...,xm)
```

Os `yi` são variáveis de saída calculadas como funções de variáveis de entrada e variáveis existentes no Scilab quando o macro é executado. Um macro pode ser compilado para uma execução mais rápida. Coleções de macros podem ser armazenadas em bibliotecas. Macros que começam pelo símbolo `%` (ex.: `%foo`) e cujos argumentos são listas são usadas para executar operações específicas: por exemplo, `z=%rmr(x,y)` é equivalente a `z=x*y` quando `x` e `z` são racionais (i.e. `x=list('r',n,d,[])` com `n` e `d` polinômios).

# Ver Também

deff, getf, comp, lib

# Name

macrovar — variáveis de uma função

```
vars=macrovar(function)
```

# Parâmetros

vars
: list `list(in,out,globals,called,locals)`

function
: nome de uma função

# Descrição

Retorna em uma lista o conjunto de variáveis utilizadas por uma função. `vars` é uma lista feita de cinco vetores colunas de strings

`in` variáveis de entrada (`vars(1)`)

`out` variáveis de saída (`vars(2)`)

`globals` variáveis globais (`vars(3)`)

`called` nomes de funções chamadas (`vars(4)`)

`locals` variáveis locais (`vars(5)`)

# Exemplos

```
deff('y=f(x1,x2)','loc=1;y=a*x1+x2-loc')
vars=macrovar(f)
```

# Ver Também

string, macr2lst

# Name

plotprofile — extrai e exibe execução de profiles (dossiês) de uma função Scilab

```
plotprofile(fun)
```

# Parâmetros

fun
: função Scilab compilada, ou nome de função (string), ou array de nomes de funções

# Descrição

Para utilizar `plotprofile`, a função Scilab deve ter sido preparada para profiling (análise de desempenho) (ver exec).

Quando tal função é executada, o sistema conta quantas vezes cada linha foi executada e quanto tempo de cpu foi gasto para cada linha. Estes dados são armazenados dentro da estrutura de dados da função. A função `plotprofile` é um comando interativo que exibe os resultados em uma janela gráfica. Quando uma linha é clcada, a fonte da função é exibida com a linha selecionada realçada.

NOTA: você deve clicar no item "Exit" na janela de gráficos para sair de "plotprofile".

O código da função é gerado com `fun2string` e guardado em um arquivo temporário.

# Exemplos

```
//definindo função e preparando-a para profiling
deff('x=foo(n)',['if n==0 then'
                 '  x=[]'
                 'else'
                 '  x=0'
                 '  for k=1:n'
                 '    s=svd(rand(n+10,n+10))'
                 '    x=x+s(1)'
                 '  end'
                 'end'],'p')
//chamando a função
foo(30)
//obtendo profiles de execução
plotprofile(foo) // clique em Exit para sair
```

# Ver Também

profile, showprofile, fun2string

# Name

profile — extrai profiles (dossiês) de execução de uma função do Scilab

```
c=profile(fun)
```

# Parâmetro

fun
: função Scilab

c
: uma matriz nx3 contendo os profiles de execução

# Descrição

Para utilizar `profile` a função Scilab deve ter sido preparada para profiling (análise de desempenho) (ver exec).

Pata tal função, quando tal função é executada, o sistema conta quantas vezes cada linha foi executada e quanto tempo de cpu foi gasto para cada execução da linha. Estes dados são armazenados na estrutura de dados da função. A função profile permite extrair esses dados e retorná-los nas duas primeiras colunas de `c`. A terceira coluna de `c` fornece a medida do esforço do interpretador para a execução da linha correspondente. A i-ésima linha de `c` corresponde à i-ésima linha da função (inclusa primeiro)

Note que, devido à precisão do clock do processador (tipicamente, um microssegundo), algumas linhas executadas aparecem com tempo de execução 0, mesmo que o tempo de execução de cpu total realmente gasto seja grande.

# Exemplos

```
//definindo a função e preparando-a para profiling
deff('x=foo(n)',['if n==0 then'
                 '  x=[]'
                 'else'
                 '  x=0'
                 '  for k=1:n'
                 '    s=svd(rand(n+10,n+10))'
                 '    x=x+s(1)'
                 '  end'
                 'end'],'p')
//chamando a função
foo(10)
//obtendo profiles de execução
profile(foo)
//chamando a função
foo(20)
profile(foo) //a execução de profiles é acumulada
```

# Ver Também

exec, deff, plotprofile, showprofile

# Name

recompilefunction — recompila uma função Scilab modificando o seu tipo

```
recompilefunction(funname [,kind [,force]])
```

# Parâmetros

funname
: string, nome da função a ser recompilada

kind
: string: **"n"** (não compilado, tipo 11), **"c"** (compilado, tipo 13) or **"p"** (compilado, tipo 13, preparado para profiling). Pardrão: "c".

force
: booleano. Se falso, a função é recompilada apenas se seu tipo mudar; se verdadeiro, é recompilada ainda que mantenha o tipo (notavelmente útil para compilar uma função "p" function, para reiniciar as estatísticas de profiling).

# Descrição

- Esta função compila reversamente uma variável função via fun2string, e a recompila para o tipo desejado através de deff.

# Exemplos

```
recompilefunction("asinh","p")
for i=1:100; asinh(rand(100,100)); end
showprofile(asinh)
```

# Ver Também

function, exec, deff, comp, fun2string, profile

# Autor

Enrico Segre

# Bibliografia

http://wiki.scilab.org/Scilab_function_variables%3A_representation%2C_manipulation

# Nom

remove_profiling — remove instruções de profiling (análise de performance) de uma função

```
remove_profiling(funname)
```

# Parâmetros

funname
: string, o nome da função

# Descrição

`remove_profiling(funname)` remove instruções de profiling (se houver alguma) da função de nome `funname`.

# Exemplos

```
function x=foo(a,n)
 x=0;
 for i=1:n
   if x<10 then
     x=x+a
   else
     x=x+1
   end
 end
 x=x^2+1
endfunction

add_profiling("foo")
foo(0.1,100) //executando a função
profile(foo) //extraindo informação de profile
remove_profiling("foo")
```

# Ver Também

profile, plotprofile, remove_profiling, reset_profiling

# Autor

Serge Steer, INRIA

# Funções Utilizadas

Esta função utiliza as funções Scilab `bytecode` e `walkbytecode`

# Nom

reset_profiling — reinicia contadores de profiling de uma função

```
reset_profiling(funname)
```

# Parâmetros

funname
: string, o nome da função

# Descrição

`reset_profiling(funname)` reinicia contadores de profiling (se houver) da função de nome `funname`.

# Exemplos

```
function x=foo(a,n)
 x=0;
 for i=1:n
   if x<10 then
     x=x+a
   else
     x=x+1
   end
 end
 x=x^2+1
endfunction

add_profiling("foo")
foo(0.1,100) //executando a função
profile(foo) //extraindo informação de profile
reset_profiling("foo")
profile(foo) //extraindo informação de profile
```

# Ver Também

profile, plotprofile, add_profiling, reset_profiling, remove_profiling

# Autor

Serge Steer, INRIA

# Funções Utilizadas

Esta função utiliza as funções Scilab `bytecode` e `walkbytecode`

# Name

showprofile — extrai e exibe profiles (dossiês) de execução de uma função Scilab

```
showprofile(fun)
```

# Parâmetros

fun
: função Scilab

# Descrição

Para utilizar `showprofile` a função Scilab deve ter sido preparada para profiling (análise de performance) (ver exec).

Para tal função,Quando tal função é executada, o sistema conta quantas número de vezes que cada linha é executada e quanto tempo de cpu é gasto para execução de cada linha. Estes dados são armazenados dentro da estrutura de dados da função. A função `showprofile` retorna resultados de profiling (ver `profile`) com o texto das linhas da função.

O texto da função é reconstruído com `fun2string`.

# Exemplos

```
//definindo função e preparando-a para profiling
deff('x=foo(n)',['if n==0 then'
                 '  x=[]'
                 'else'
                 '  x=0'
                 '  for k=1:n'
                 '    s=svd(rand(n+10,n+10))'
                 '    x=x+s(1)'
                 '  end'
                 'end'],'p')
//chamada à função
foo(30)
//obtendo profiles
showprofile(foo)
```

# Ver Também

profile, plotprofile, fun2string

# Name

varargin — variável do número de argumentos de saída em uma lista de argumentos de entrada

# Descrição

Uma função cujo último argumento de entrada é `varargin` pode ser chamada com mais argumentos de entrada que os indicados pela lista de argumentos de entrada. Os argumentos de chamada passados da palavra-chave `varargin` em diante podem ser recuperados em uma lista chamada `varargin`.

Suponha que a palavra-chave `varargin` é o n-ésimo argumento da lista de argumentos de entrada formal, então, se a função é chamada com `n-1` argumentos de entrada, a lista `varargin` não é definida. Se a função for chamada com `n-1` argumentos, então `varargin` é uma lista vazia

`y = function ex(varargin)` pode ser chamada com qualquer número de argumentos de entrada. Dentro da função `ex` argumentos de entrada podem ser recuperados em `varargin(i)` `,i=1:length(varargin)`

Se não for o último argumento de entrada da função, `varargin` é um argumento de entrada normal, sem nenhum significado especial.

O número de argumentos de entrada real é dado por argn(2).

# Observações

Sintaxe de argumento nomeada, como `foo(...,key=value)` , é incompatível com o uso de varargin. A razão é que os nomes (i.e. keys) associados aos valores value não são armazenados na lista varargin. Considere por exemplo:

function foo(varargin),disp([varargin(1),varargin(2)]),endfunction

foo(a=1,b=2)

O Scilab responde: 1. 2.

foo(b=1,a=2)

O Scilab responde: 1. 2.

O resultado é o mesmo, mas os argumentos foram invertidos.

# Exemplos

```
deff('exampl(a,varargin)',['[lhs,rhs]=argn(0)'
                           'if rhs>=1 then disp(varargin),end'])
exampl(1)
exampl()
exampl(1,2,3)
l=list('a',%s,%t);
exampl(1,l(2:3))
```

# Ver Também

function, varargout, list

# Name

varargout — variável do número de argumentos de saída em uma lista de argumentos de saída

# Descrição

Uma função cuja lista de argumentos de saída contém `varargout` deve ser chamada com mais argumentos de saída que indicado na lista de argumentos de saída. A chamada de argumentos passados da palavra-chave `varargout` em diante são extraídos da lista `varargout` definida na função.

`varargout= function ex()` pode ser chamado com qualquer número de argumentos de saída. Dentro da função `ex` , argumentos de saída podem ser armazenados em `varargout(i)`.

`[X1,...Xn,varargout]= function ex()` também pode ser usado. Neste caso, as variáveis `Xi` devem estar atribuídas na função tanto quanto `varargout(i)`.

O número total real de argumentos de saída é dado por argn(1)

# Observação

A variável `varargout` deve ser criada dentro de uma função e atribuída a uma list. Se varargout for a única variável de saída formal, a lista deve conter pelo menos uma entrada.

# Exemplos

```
function varargout=exampl()
  varargout=list(1,2,3,4)
endfunction

x=exampl()
[x,y]=exampl()
[x,y,z]=exampl()

function [a,b,varargout]=exampl1()
  a='first'
  b='second'
  varargout=list(1,2,3,4)
endfunction
exampl1()
[a,b]=exampl1()
[a,b,c]=exampl1()
```

# Ver Também

function, varargin, list

# Name

whereis — ome da biblioteca contendo a função

```
[librname]=whereis(function-name)
```

# Descrição

Retorna um string correspondente ao nome da biblioteca onde a função `function-name`. se encontra. O endereço da biblioteca é retornado digitando-se `"librname"`.

# Ver Também

lib

# Name

getf — defining a function from a file

```
getf(file-name [,opt])
```

# Parameters

filename
: Scilab string.

opt
: optional character string

"c"
: loaded functions are "compiled" to be more efficient (default)

"n"
: loaded functions are not "compiled"

"p"
: loaded functions are "compiled" and prepared for profiling (see profile)

# Description

WARNING: this function has been deprecated (see exec as a replacement of getf). getf will be removed in Scilab 5.3

loads one or several functions (see `functions`) defined in the file `'file-name'`. The string `opt='n'` means that the functions are not compiled (pre-interpreted) when loaded. This can be useful for some debugging purpose (see `comp`). By default, functions are compiled when loaded (i.e. `opt='c'` is used).

In the file a function must begin by a "syntax definition" line as follows:

```
function [y1,...,yn]=foo(x1,...,xm)
```

The following lines contain a sequence of scilab instructions.

The "syntax definition" line gives the "full" calling syntax of this function. The `yi` are output variables calculated as functions of input variables `xi` and variables existing in Scilab when the function is executed. Shorter input or output argument list may be used.

Many functions may be written in the same file. A function is terminated by an `endfunction` keyword. For compatibility with previous versions a function may also be terminated by the following `function` keyword or the `EOF` mark. For that reason it is not possible to load function containing nested function definition using the `getf` function.

`getf` is an obsolete way for loading functions into scilab from a file. It is replaced by the function exec. Note that functions in a file should be terminated by an `endfunction` keyword. The `exec` function supposes `opt=='c'`.

To prepare a function for profiling please use the add_profiling function.

# Examples

```
getf('SCI/modules/graphics/macros/plot.sci')

getf SCI/modules/graphics/macros/plot.sci
```

## See Also

functions, function, genlib, getd, exec, edit, comp, add_profiling

# Parte V. Arquivos : funções de Entrada/Saída

# Name

chdir — muda o diretório corrente do Scilab
cd — muda o diretório corrente do Scilab

```
b=chdir(path)
realpath=cd(path)
cd path
```

# Parâmetros

b
: um booleano %t se a operação chdir estiver ok.

path
: um string

realpath
: um string, o nome real do endereço após a conversão do nome do endereço (ver abaixo)

# Descrição

Muda o diretório corrente do Scilab para aquele dado por `path`. Note que a conversão de endereço é realizada e. por exemplo `SCI/modules/core/macros` é um padrão válido tanto pra Unix quanto para Windows. Se path for vazio, muda para diretório "home".

# Exemplos

```
chdir(TMPDIR);
pwd
cd
cd SCI
```

# Ver Também

pwd

# Name

fileinfo — Fornece informações sobre um arquivo

```
[x,ierr]=fileinfo(file)
```

# Parâmetros

file
: string, o endereço do arquivo

x
: um vetor de inteiros de tamanho 13 contendo informações ou uma matriz vazia se o arquivo não existir.

ierr
: indicador de erro, 0, se não for encontrado nenhum erro.

# Descrição

x=fileinfo(file) retorna

x(1)
: O tamanho do arquivo

x(2)
: O modo do arquivo (valor decimal)

x(3)
: O id do usuário

x(4)
: O id do grupo

x(5)
: O número do dispositivo

x(6)
: A data da última modificação

x(7)
: A data da última mudança

x(8)
: A data do último acesso

x(9)
: O tipo de dispositivo (se o dispositivo for inode)

x(10)
: O tamanho de bloco para a entrada/saída do sismtema de arquivos (sempre 0 no Windows)

x(11)
: O número de blocos alocados (sempre 0 no Windows)

x(12)
: O inode

x(13)
: O número de hard links.

# Referência

Esta função é uma interface para a função C stat.

Permissões são tipicamente especificadas como números octais : dec2oct(x(2)) para conversão.

O modo numérico possui de um a quatro dígitos octais (0-7), derivados através da soma de bits com os valores 4, 2, e 1. Quaisquer dígitos omitidos são supostos como zeros à esquerda. O primeiro dígito seleciona o ID de usuário (4) e ajusta o ID do grupo (2) e atributos "sticky" (1). O segundo dígito seleciona permissões para o usuário que possui o arquivo: leitura (4), escrita (2), e execução (1); o terceiro dígito seleciona permissões para os outros usuários no grupo do arquivo, com os mesmos valores; e o quarto seleciona permissões para outros usuários que não estão no grupo do arquivo, com os mesmos valores.

# Exemplos

```
w = fileinfo(SCI+'/etc/scilab.start')
// permiss##o do arquivo
dec2oct(w(2))
// data do arquivo
getdate(w(6))

// Verificando a permiss##o de escrita em um arquivo
w = fileinfo(SCI+'/etc/scilab.start')

S_IWRITE = 128; // mascarando a permiss##o de escrita
S_IEXEC = 64; // mascarando a permiss##o de execu#§##o
S_IREAD = 256; // mascarando a permiss##o de leitura
S_IFCHR = 8192; // mascarando a permiss##o de diret##rio

if ( bitand( w(2), S_IWRITE ) <> 0) then
 disp('PERIMISS##O DE ESCRITA neste arquivo.');
else
 disp('N##O H## PERIMISS##O DE ESCRITA neste arquivo.');
end
```

# Ver Também

getdate, file, dispfiles, newest, isdir

# Autores

S. Steer INRIA

A.C

# Name

get_absolute_file_path — fornece o nome de endereço absoluto de um arquivo aberto no Scilab

```
pathname = get_absolute_file_path(filename)
```

# Parâmetros

filename
: string : nome do arquivo

pathname
: string : o nome de endereço absoluto

# Descrição

Fornece o nome de endereço absoluto de um arquivo já aberto no Scilab.

get_absolute_file_path procura na lista interna de arquivos abertos correntemente do Scilab, filename e retorna seu endereço.

Se o arquivo não for aberto, um erro é retornado.

Aviso : na versão anterior (Scilab 5.0.x) o diretório corrente era retornado se o arquivo não fosse encontrado.

Esta função pode ser utilizada para encontrar de onde (endereço) um script Scilab é executado.

# Exemplos

```
// executando este script

a=mopen(TMPDIR+'test.sce','wt');
disp(get_absolute_file_path('test.sce'));
mclose(a);
```

# Ver Também

getshortpathname, getlongpathname, pwd

# Autores

Allan CORNET

# Name

getrelativefilename — dado um nome de diretório absoluto e um nome de arquivo relativo, retorna um nome de arquivo relativo.

```
rel_file = getrelativefilename(abs_dir,abs_file)
```

# Parâmetros

abs_dir
: string : o diretório absoluto

abs_file
: string : o nome de arquivo absoluto

rel_file
: string : o nome de arquivo relativo

# Descrição

dado um nome de diretório absoluto e um nome de arquivo relativo, retorna um nome de arquivo relativo.

Por exemplo, se o diretório corrente é C:\scilab\bin e o nome de arquivo C:\scilab\modules\helptools\readme.txt é dado, getrelativefilename retornará ..\modules\helptools\readme.txt.

# Exemplos

```
if MSDOS then
  getrelativefilename('C:\program file\scilab-4.0\bin','C:\program file\scilab-
  getrelativefilename('C:\program file\scilab-4.0\bin\','C:\program file\scilab
  getrelativefilename(SCI+'\bin',SCI+'\modules\helptools\help.dtd')
  getrelativefilename(WSCI+'\bin',WSCI+'\modules\helptools\help.dtd')
  getrelativefilename(pwd(),WSCI+'\bin\Wscilex')
else
  getrelativefilename('/usr/local/scilab-4.0/bin','/usr/local/scilab-4.0/module
  getrelativefilename('/usr/local/scilab-4.0/bin/','/usr/local/scilab-4.0/modul
  getrelativefilename(SCI+'/bin',SCI+'/modules/helptools/help.dtd')
  getrelativefilename(pwd(),SCI+'/bin/scilex')
end
```

# Ver Também

getshortpathname, getlongpathname, pwd

# Autor

Pierre MARECHAL

# Name

newest — retorna o arquivo mais novo de um conjunto de arquivos

```
k=newest(paths)
k=newest(path1,path2,...,pathn)
```

# Parâmetros

k
: o índice do arquivo mais novo

paths
: vetor de strings, paths(i) é o endereço do i-ésimo arquivo

pathi
: string, o endereço do i-ésimo arquivo

# Descrição

Dado um conjunto de endereços, newest retorna o índice do mais novo. Arquivos não existentes são supostos como mais antigos.

# Exemplos

```
newest('SCI/modules/graphics/macros/bode.sci','SCI/modules/graphics/macros/bode
newest(['SCI/modules/graphics/macros/bode.sci','SCI/modules/graphics/macros/bod
newest('SCI/modules/graphics/macros/bode.'+['sci','bin'])
```

# Ver Também

fileinfo

# Name

sscanf — converte entrada formatada fornecida por um string

```
[v_1,...v_n]=sscanf (string,format)
```

# Parameters

format
: especifica a conversão de formato

string
: especifica a entrada a ser lida

# Descrição

Esta função está obsoleta, use a função `msscanf` de preferência, que é mais eficiente mais compatível com o procedimento C `sscanf`.

A função sscanf intepreta um string de acordo com um formato, e retorna os resultados convertidos.

O parâmetro de formato contém especificações de conversão utilizadas para interpretar a saída.

O parâmetro de formato pode conter caracteres em branco (espaços, tabulações, novas linhas, quebra de página) os quais, exceto nos dois casos seguintes, lê a entrada até o próximo caractere preenchido. A menos que haja uma correspondência no string de controle, espaços em branco (incluindo caractere de nova linha) vindo por último não são lidos.

- Qualquer caractere exceto % (sinal de porcento), que deve corresponder ao próximo caractere do fluxo de entrada.
- Uma especificação de conversão que direciona a conversão do próximo campo de entrada. Ver scanf_conversion para detalhes.

# Ver Também

mprintf, msscanf, mfscanf, scanf_conversion

# Name

basename — strip directory and suffix from filenames

```
files= basename(files[,flag [,flagexpand]])
```

# Parameters

files
: a string matrix giving a set of file names.

flag,flagexpand
: boolean optional parameters. (default value `%t`).

files
: a string matrix.

# Description

`basename` return the basename of the file entries given in files.

If `flag` is true the files are first converted to the target type given by the `MSDOS` variable. Moreover, if `flagexpand` is true leading strings like `HOME`, `SCI` or ~ are expanded using environment variables.

# Examples

```
files=basename('SCI/modules/fileio/macros/poo.sci')
files=basename('SCI/modules\fileio/macros/poo.sci')
files=basename('SCI/modules\fileio/macros/poo.sci.k')
```

# See Also

listfiles, pathconvert, fileparts

# Name

copyfile — Copy file

```
copyfile('source','destination')
[status,message] = copyfile('source','destination')
```

# Description

copyfile('source','destination') copies the file or directory , source (and subdirectories) to the file or directory, destination.

If source is a directory, destination can not be a file. copyfile replaces existing files without warning.

[status, message] = copyfile('source','destination') copies source to destination, returning the status and a message.

Whatever the operating system, if the copy succeeds, the status is 1 and the message is empty ; if the copy fails, the status is 0 and the message is not empty.

The timestamp given to the destination file is identical to that taken from source file.

# Examples

```
copyfile(SCI+"/etc/scilab.start",TMPDIR+"/scilab.start")
[status,message] = copyfile(SCI+"/etc/scilab.start",TMPDIR);
```

# See Also

movefile, mdelete

# Authors

Allan CORNET

# Name

createdir — Make new directory

```
createdir('dirname')
status = createdir('dirname')
```

# Description

createdir('dirname') creates the directory dirname in the current directory, if dirname is not in the current directory, specify the relative path to the current directory or the full path for dirname.

[status] = createdir('dirname') creates the directory dirname in the existing directory parentdir, returning the status, a message. Here, status is %T for success and %F otherwise.

createdir is used by mkdir.

# Examples

```
createdir(SCIHOME+'/Directory_test')
removedir(SCIHOME+'/Directory_test')
```

# See Also

mkdir , rmdir

# Authors

A.C

# Name

deletefile — delete a file

```
f = deletefile(filename)
```

# Parameters

filename
: a file name existing or not.

f
: %t or %f

# Description

delete a file. if file has been deleted, it will return %t else %f.

# Examples

```
fd = mopen(TMPDIR+'/filetodelete.txt','wt');
mclose(fd);
if (fileinfo(TMPDIR+'/filetodelete.txt') <> []) then deletefile(TMPDIR+'/fileto
deletefile(TMPDIR+'/notexistingfile')
```

# Authors

A.C

# Name

dir — get file list

```
dir path
S=dir([path])
```

# Parameters

path
: a string matrix giving a directory pathname (eventually ended by a pattern built with `*`). Default value is `.`

S
: a `tlist` of type `dir` with fields : `name`, `date` and `isdir`

# Description

`dir` can be used to get the files which match the patterns given by the path argument. Patterns are given to the unix `ls` or to the windows `dir` commands in order to get information on files. Thus in order to write portable Scilab script valid wildcard patterns for both os are to be given. Note that Pathname conversion is performed and for example `SCI/modules/core/macros/*.sci` is a valid pattern for both unix and windows.

The `name` field of the returned variable is the column vector of the file names.

The `date` field of the returned variable is the column vector of integers containing a last modification date coded in second from 1 Jan 1970).

The `isdir` field of the returned variable is the column vector of boolean true if the corresponding name is a directory.

The default display of the returned structure is a column formatted list of files. It can be changed redefining the function `%dir_p`

# Examples

```
dir
dir SCI/modules/core/macros/*.bin
x=dir('SCI/modules/core/macros/*.bin')
dt=getdate(x.date);
mprintf("%s: %04d-%02d-%02d %02d:%02d:%02d\n",x.name,dt(:,[1 2 6 7:9]))
```

# See Also

listfiles , findfiles , ls , fileinfo , date

# Name

dirname — get directory from filenames

```
dirs= dirname(files[,flag [,flagexpand]])
```

# Parameters

files
: a string matrix giving a set of file names.

flag,flagexpand
: boolean optional parameters. (default value `%t`).

files,dir
: string matrices.

# Description

`dirname` return the dirname of the file entries given in files.

If `flag` is true the files are first converted to the target type given by the `MSDOS` variable. Moreover, if `flagexpand` is true leading strings like `HOME`, `SCI` or ~ are expanded using environment variables.

Note that `dirname(files,%f)` can give erroneous results if pathnames given in files do not follow the convention given by the `MSDOS` variable.

# Examples

```
files=dirname('SCI/modules/fileio/macros/poo.sci')
files=dirname('SCI/modules\fileio/macros/poo.sci')
files=dirname('SCI/modules\fileio/macros/poo.sci.k')
```

# See Also

basename , listfiles , pathconvert

# Name

dispfiles — display opened files properties

```
dispfiles([units])
```

# Parameters

units
: a vector of numbers, the file's logical units. By default all opened files.

# Description

dispfiles displays properties of currently opened files.

# Examples

```
dispfiles()
```

# See Also

file , mopen

# Authors

S. Steer

# Name

fileext — returns extension for a file path

```
extension = fileext(fullpath)
```

# Parameters

fullpath
: a character string, the given file path

extension
: a character string, the extension part is any or ''

# Description

`extension=fileext(fullpath)` splits the fullpath character string in the extension part including the dot.

# Examples

```
extension = fileext('SCI/etc/scilab.start')
extension = fileext(['SCI/etc/scilab.start';'SCI/etc/scilab.quit'])
```

# See Also

fileparts

# Authors

Allan CORNET

# Name

fileparts — returns the path, filename and extension for a file path

```
[path,fname,extension]=fileparts(fullpath)
v=fileparts(fullpath,sel)
```

# Parameters

fullpath
: a character string, the given file path

sel
: a optional character string selector, with posible values: 'path' 'fname' or 'extension'

path
: a character string, the path of the directory pointed to by fullpath

fname
: a character string, the filename part is any or ''

extension
: a character string, the extension part is any or ''

value
: a character string, depending on sel value

# Description

`[path,fname,extension]=fileparts(fullpath)` splits the fullpath character string in its three parts: the path of the directory pointed to, the filename part, the extension part including the dot.

# Examples

```
[path,fname,extension]=fileparts('SCI/etc/scilab.start')
fileparts('SCI/etc/scilab.start','extension')
```

# See Also

pathconvert , basename , fullfile

# Authors

Serge Steer, INRIA

# Name

filesep — returns directory separator for current platform

```
s = filesep()
```

# Parameters

s
: a string

# Description

returns directory separator. ( '/' on Linux or '\' on Windows )

# Examples

```
filesep()
```

# Authors

A.C

# Name

findfiles — Finding all files with a given filespec

```
f = findfiles()
f=findfiles(path)
f=findfiles(path,filespec)
```

# Parameters

path
: a path

filespec
: a spec file. example "*.sce"

f
: returns a string matrix of filenames

# Description

Finding all files with a given filespec

# Examples

```
f=findfiles()
f=findfiles(SCI)
f=findfiles(SCI+'/modules/core/macros','*.sci')
```

# See Also

listfiles

# Authors

A.C

# Name

fprintf — Emulator of C language fprintf function

```
fprintf(file,format,value_1,..,value_n)
```

# Parameters

format
: a Scilab string. Specifies a character string combining literal characters with conversion specifications.

value_i
: Specifies the data to be converted according to the format parameter.

str
: column vector of character strings

file
: a Scilab string specifying a file name or a logical unit number (see `file`)

  Note that is file=0, the message will be display on standard error stream (stderr).

# Description

Obsolete function, use preferabily the mfprintf function which is much more compatible with the C `fprintf` functionalities.

The `fprintf` function converts, formats, and writes its `value` parameters, under control of the `format` parameter, to the file specified by its `file` parameter.

The `format` parameter is a character string that contains two types of objects:

Literal characters
: which are copied to the output stream.

Conversion specifications
: each of which causes zero or more items to be fetched from the `value` parameter list. see printf_conversion for details

If any `value`s remain after the entire `format` has been processed, they are ignored.

# Examples

```
u=file('open','results','unknown') //open the result file
t=0:0.1:2*%pi;
for tk=t
  fprintf(u,'time = %6.3f value = %6.3f',tk,sin(tk)) // write a line
end
file('close',u) //close the result file

fprintf(0,'My error which is going to be displayed on the stderr')
```

# See Also

mfprintf , string , print , write , format , disp , file , printf , sprintf , printf_conversion

# Name

fprintfMat — Write a matrix in a file.

```
fprintfMat(file,M [,format,text])
```

# Parameters

fil
: a string, the pathname of the file to be written.

M
: A matrix of real numbers.

format
: a character string, a C like format. This is an optional parameter, the default value is `"%f"`

text
: a string matrix giving non numerical comments stored at the beginning of the file.

# Description

The `fprintfMat` function writes a matrix in a formated file. Each row of the matrix give a line in the file. If `text` is given then the elements of `text` are inserted columnwise at the beginning of the file one element per line.

# Examples

```
n=50;
a=rand(n,n,'u');
fprintfMat(TMPDIR+'/Mat',a,'%5.2f');
a1=fscanfMat(TMPDIR+'/Mat');
```

# See Also

mclose , meof , mfprintf , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

# Name

fscanf — Converts formatted input read on a file

```
[v_1,...v_n]=fscanf (file,format)
```

# Parameters

format
: Specifies the format conversion.

file
: Specifies the input file name or file number.

# Description

The fscanf functions read character data on the file specified by the `file` argument , interpret it according to a format, and returns the converted results.

The format parameter contains conversion specifications used to interpret the input.

The format parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.

- Any character except % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. see scanf_conversion for details.

# See Also

printf , read , scanf , sscanf , mfscanf , scanf_conversion

# Name

fscanfMat — Reads a Matrix from a text file.

```
M=fscanfMat(filename);
[M,text]=fscanfMat(filename);
```

# Parameters

filename
: a character string giving the name of the file to be scanned.

M
: Output variable. A matrix of real numbers.

text
: Output variable. A string matrix.

# Description

The `fscanfMat` function is used to read a scalar matrix from a text file. The first non-numeric lines of the file are returned in `text` if requested and all the remaining lines must have the same number of columns (column separator are assumed to be white spaces or tab characters). The number of columns of the matrix will follow the number of columns found in the file and the number of lines is fetched by detecting eof in the input file. This function can be used to read back numerical data saved with the `fprintfMat`.

# Examples

```
fd=mopen(TMPDIR+'/Mat','w');
mfprintf(fd,'Some text.....\n');
mfprintf(fd,'Some text again\n');
a=rand(6,6);
for i=1:6 ,
  for j=1:6, mfprintf(fd,'%5.2f ',a(i,j));end;
  mfprintf(fd,'\n');
end
mclose(fd);
a1=fscanfMat(TMPDIR+'/Mat')
```

# See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

# Name

fullfile — Build a full filename from parts

```
f = fullfile(varargin)
```

# Parameters

varargin
: all directories and filename used to build the full filename (at least one directory and filename)

f
: full filename

# Description

`f = fullfile(varargin)` builds a full filename taking care of platform on which it is run and handling the cases when the directories begin or end with a directory separator.

# Examples

```
f=fullfile("/home/","\scilab","macros","\util","fullfile.sci")
f=fullfile("C:","\scilab","macros","\util","fullfile.sci")
```

# See Also

pathconvert , fileparts

# Authors

V.C.

# Name

fullpath — Creates an full path name for the specified relative path name.

```
res = fullpath(relative_path)
```

# Parameters

res
: a string

relative_path
: a string

# Description

Creates an full path name for the specified relative path name.

On linux 'relative_path' needs to exist.

# Examples

```
mkdir(TMPDIR+'/niv1');
mkdir(TMPDIR+'/niv1/niv2');
mputl(' ',TMPDIR+'/niv1/test.txt');
cd(TMPDIR+'/niv1/niv2');
fullpath('../test.txt')
```

# Authors

A.C

# Name

getdrives — Get the drive letters of all mounted filesystems on the computer.

```
drives = getdrives()
```

# Parameters

drives
: a matrix of strings

# Description

Get the drive letters of all mounted filesystems on the computer.

returns the roots of all mounted filesystems on the computer as a matrix of strings.

For Linux this list consists solely of the root directory, / .

# Examples

```
getdrives()
```

# Authors

A.C

# Name

getlongpathname — get long path name (Only for Windows)

```
longpath=getlongpathname(shortpath)
[longpath,bOK]=getlongpathname(shortpath)
```

# Parameters

shortpath
: A character string or matrix of strings the short path

longpath
: A character string or matrix of strings the long path

bOK
: A boolean %T or a matrix of boolean if path has been converted else %F

# Description

The getlongpathname primitive converts the specified path to its long form. It no long path is found, this primitive returns the specified name.

# Examples

```
[longpath,bOK]=getlongpathname(SCI)
[longpaths,bOKs]=getlongpathname([SCI;SCI])
```

# See Also

getshortpathname

# Authors

Allan CORNET

# Name

getshortpathname — get short path name (Only for Windows)

```
shortpath=getshortgpathname(longpath)
[shortpath,bOK]=getshortpathname(longpath)
```

# Parameters

longpath
: A character string or matrix of strings the long path

shortpath
: A character string or a matrix of strings the short path

bOK
: A boolean %T or a matrix of boolean if path has been converted else %F

# Description

The getshortpathname primitive converts the specified path to its short form.

# Examples

```
[shortpath,bOK]=getshortpathname(SCI)
[shortpaths,bOKs]=getshortpathname([SCI;SCI])
```

# See Also

getlongpathname

# Authors

Allan CORNET

# Name

%io — variable returns standard input and standard output (file descriptor).

```
%io(1)
%io(2)
```

# Description

%io(1) returns standard input (file descriptor 5).

%io(2) returns standard ouput (file descriptor 6).

file descriptor 0 is the standard error.

# Examples

```
mfprintf(%io(2),'Scilab stdout (C)');
mfprintf(0,'Scilab stderr (C)');
write(%io(2),'Scilab stdout (Fortran)');
write(0,'Scilab stderr (Fortran');
[units,typ,names]=file()
```

# See Also

write, mfprintf, file

# Name

isdir — checks if argument is a directory path

```
r=isdir(path)
```

# Parameters

path
: a character string or a matrix of strings, the file pathname

r
: a boolean, true if `path` is a path to a directory.

# Description

`r=isdir(path)` checks if `path` is a path to a directory.

# Reference

This function is based on the C function stat. The `SCI` and ~ shortcuts for Scilab directory and home directory are handled.

# Examples

```
isdir(TMPDIR)
isdir SCI/etc/scilab.start
```

# See Also

fileinfo

# Authors

S. Steer INRIA

# Name

isfile — checks if argument is a file

```
b = isfile(filenames)
```

# Parameters

files
: a character string or a string matrix.

x
: an matrix of boolean (%t is filename exists).

# Description

x = isfile(filename) checks if filename is existing file or not. (a directory is not a file. see isdir.)

# Examples

```
cd SCI
filenames = ls()
isfile(filenames)

isfile(filenames)

isfile(SCI + '/etc')
isdir(SCI + '/etc')
```

# See Also

fileinfo, isdir

# Author

A.C

# Name

listfiles — list files

```
files= listfiles(paths [,flag,flagexpand])
```

# Parameters

paths
: a string matrix giving a set of pathnames (eventually ended by a pattern built with `*`)

flag,flagexpand
: boolean optional parameters. (default value `%t`).

files
: a string matrix.

# Description

`listfiles` can be used to list the files which match the patterns given by one of the paths entries. Patterns are given to the unix `ls` or to the windows `dir` commands in order to get information on files. Thus in order to write portable Scilab script valid wildcard patterns for both os are to be given. Note that Pathname conversion is performed and for example `SCI/core/macros/*.sci` is a valid pattern for both unix and windows.

if `flag` is true the pathnames given by `paths` are converted according to the MSDOS value (See `pathconvert`). Moreover, if `flagexpand` is true leading strings like `HOME`, `SCI` or `~` are expanded using environment variables.

# Examples

```
files=listfiles(['SCI/modules/core/macros/*.sci';'SCI/modules/core/macros/*.bin
```

# See Also

findfiles , basename , pathconvert

# Name

listvarinfile — list the contents of a saved data file

```
listvarinfile(filename)
[nams,typs,dims,vols]=listvarinfile(filename)
```

# Parameters

filename
: character string, the pathname of the file to be inspected

nams
: character array, names of the variables saved in the file

dims
: list, dimensions of the variables saved in the file

typs
: numeric array, types of the variables saved in the file

vols
: numeric array, size in bytes of the variables saved in the file

# Description

- This utility function lists "a la whos" the variables contained in a Scilab data file produced by save.

  Remark: hypermatrices are reported as plain mlists; rationals and state-spaces are reported as plain tlists; graphic handles are not recognized.

# Examples

```
a=eye(2,2); b=int16(ones(a)); c=rand(2,3,3);
save("vals.dat",a,b,c)
listvarinfile("vals.dat")
```

# See Also

whos , save , load , save_format , type

# Authors

Serge Steer
: 31 Jan 2001; reediting by Enrico Segre

# Name

ls — show files

```
ls path options
files=ls( [path] )
```

# Parameters

path
: a string matrix giving a directory pathname (eventually ended by a pattern built with `*`). Default value is `.`

files
: a string column vector. By default it contains a column formatted output. if one of the option is `'-1'`. `files` contains an entry for each files

# Description

`ls` can be used to list the files which match the patterns given by the path argument. Patterns are given to the unix `ls` or to the windows `dir` commands in order to get information on files. Thus in order to write portable Scilab script valid wildcard patterns for both os are to be given. Note that Pathname conversion is performed and for example `SCI/modules/core/macros/*.sci` is a valid pattern for both unix and windows.

If you want to get a vector of all files matching a pattern use preferabily the listfiles or the dirfunction.

Please note that starting from the version 5.0 of Scilab, the second input argument has been removed (a sequence of strings which can be added under Unix systems: the Unix ls command options). This option has been removed mainly for security and portability reasons.

# Examples

```
ls
ls SCI/modules/core/macros/*.sci
x=ls('SCI/modules/core/macros/*.sci')
```

# See Also

listfiles , findfiles , dir , fileinfo

# Name

maxfiles — sets the limit for the number of files a scilab is allowed to have open simultaneously.

```
r= maxfiles(newnumbermax)
```

# Parameters

newnumbermax
: a integer the new value

r
: effective new value.

# Description

sets the limit for the number of files a scilab is allowed to have open simultaneously.

Minimum : 20

Maximum : 100

Default : 20

# Examples

```
r=maxfiles(50);
```

# See Also

mopen

# Name

mclearerr — reset binary file access errors

```
mclearerr([fd])
```

# Parameters

fd

scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

# Description

The function `clearerr` is used to resets the error indicator and EOF indicator to zero.

# See Also

merror , mclose , mopen , mput , mget , mgetstr , mputstr , meof , mseek , mtell

# Name

mclose — close an opened file

```
err=mclose([fd])
mclose('all')
```

# Parameters

fd

scalar. The `fd` parameter returned by the function `mopen` is used as a file descriptor (it's a positive integer).

err

a scalar. Error indicator : vector

# Description

`mclose` must be used to close a file opened by `mopen`. If fd is omitted `mclose` closes the last opend file.

`mclose('all')` closes all files opened by `file('open',..)` or `mopen`. Be careful with this use of `mclose` because when it is used inside a Scilab script file, it also closes the script and Scilab will not execute commands written after `mclose('all')`.

# See Also

meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetl , mgetstr , mopen , mprintf , mput , mputl , mputstr , mscanf , mseek , mtell , file , mdelete

# Name

mdelete — Delete file(s)

```
mdelete(filename)
```

# Parameters

filename
: a character string. The pathname of the file(s) to delete.

# Description

`mdelete` may be used to delete a file or a set of files if `filename` contains meta-charaters.

Note that `mdelete` does not ask for confirmation when you enter the delete command. To avoid accidentally losing files, make sure that you have accurately specified the items you want deleted.

# See Also

mopen , mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell

# Name

meof — check if end of file has been reached

```
err=meof(fd)
```

# Parameters

fd
: scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

err
: scalar. Error indicator

# Description

The function `meof` will return a non null value if end of file has been reached in a previous call to `mget` or `mgetstr`. The function `clearerr` is used to reset the error flag and EOF flag to zero.

# See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

# Name

merror — tests the file access errors indicator

```
err = merror([fd])

[err,msg] = merror([fd])
```

# Parameters

fd
: scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

err
: scalar. returns the error status number errnum of the most recent file I/O operation associated with the specified file.

  If the most recent I/O operation performed on the specified file was successful, the value of msg is empty and merror returns an err value of 0.

msg
: string. returns the error string message.

# Description

The function `merror` is used to tests the file access errors indicator. returning non-zero if it is set. The error indicator can only be reset by the `mclearerr` function.

A nonzero err indicates that an error occurred in the most recent file I/O operation. The value of message is a string that can contain information about the nature of the error. If the message is not helpful, consult the C run-time library manual for your host operating system for further details.

# Examples

```
fd = mopen(TMPDIR +'/filetxt.txt','wt');
[err,msg] = merror(fd)
if (err <> 0) then
  printf('Problem\n');
end
mclose(fd);
```

# See Also

mclearerr, mclose, mopen, mput, mget, mgetstr, mputstr, meof, mseek, mtell

# Name

mfprintf — converts, formats, and writes data to a file

```
mfprintf(fd,format,a1,...,an);
```

# Parameters

fd
: scalar, file descriptor given by `mopen` (it's a positive integer).

  if fd equals 0 redirection in stderr.

  if fd equals 6 redirection in stdout.

  OBSOLETE :The value `-1` refers to the default file ( i.e the last opened file).

format
: a Scilab string describing the format to use to write the remaining operands. The format operand follows, as close as possible, the C printf format operand syntax.

str
: a character string, string to be scanned.

a1,...,an
: Specifies the data to be converted and printed according to the format parameter.

# Description

The `mfprintf` function is a interface for C-coded version of `fprintf` functions.

The `mfprintf` function writes formatted operands to the file specified by the file desciptor `fd`. The argument operands are formatted under control of the format operand.

this function may be used to output column vectors of numbers and string vectors without an explicit loop on the elements. In that case this function iterates on the rows. The shortest vector gives the number of time the format has to be iterated.

An homogeneous sequence of identical type parameters can be replaced by a matrix

# Examples

```
fd = mopen(TMPDIR+'/text.txt','wt');
mfprintf(fd,'hello %s %d.\n','world',1);
mfprintf(fd,'hello %s %d.\n','scilab',2);
mclose(fd);
if (isdef('editor') | (funptr('editor')<>0)) then
  editor(TMPDIR+'/text.txt')
end
mfprintf(0,'stderr output.\n');
mfprintf(6,'stdout output.\n');
```

# See Also

mclose, meof, fprintfMat, mfscanf, fscanfMat, mget, mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell, mdelete, printf_conversion

# Name

mscanf — reads input from the standard input (interface to the C scanf function)
mfscanf — reads input from the stream pointer stream (interface to the C fscanf function)
msscanf — reads its input from the character string (interface to the C sscanf function)

```
[n,v_1,...v_n]=mfscanf([niter,]fd,format)
L=mfscanf([niter,] fd,format)

[n,v_1,...v_n]=mscanf([niter,] format)
L=mscanf([niter,]format)

[n,v_1,...v_m]=msscanf([niter,]str,format)
L=msscanf([niter,] str,format)
```

# Parameters

format
: a Scilab string describing the format to use to write the remaining operands. The format operand follows, as close as possible, the C printf format operand syntax as described in scanf_conversion.

fd
: The fd parameter returned by the function `mopen` is used as a file descriptor (it's a positive integer). The value -1 refers to the last opened file.

str
: a Scilab string or string vector.

niter
: an integer, the number of times the format as to be used.

n
: an integer, the number of data read or -1 if EOL has been encountered before any datum has been read.

v_i
: Each function reads characters, interprets them according to a format, and stores the results in its output arguments. If more than $n$ output arguments are provided, the last ones `v_n+1,...v_m` are set to empty matrices.

L
: if all data are homogeneous they are stored in a unique vector which is returned, otherwise subsequences of same data type are stored in matrices and an `mlist` (with type `cblock`) containing all the built matrices is returned.

# Description

The `mfscanf` function reads characters from the stream `fd`.

The `mscanf` function reads characters from Scilab window.

The `msscanf` function reads characters from the Scilab string `str`.

The `niter` optional argument specifies how many time the format has to used. One iteration produces one line in the output matrix. If `niter==-1` the function iterates up to the end of file. The `niter` default value is 1.

comments about precision :

mfscanf is based on C function fscanf. If you use '%f', '%g', '%e' as format your datas will be cast to float and returned in a scilab variable.

This scilab variable is a double then you can have some precision errors. In this case, it is better to use '%lg' format.

# Examples

```
//-----------------------------------------------------------
//--    Simple use                                         --
//-----------------------------------------------------------
s='1 1.3'  //a string
[n,a,b]=msscanf(s,"%i %e")
L=msscanf(s,"%i %e")

//-----------------------------------------------------------
//--    Formats samples                                    --
//-----------------------------------------------------------

msscanf(" 12\n",'%c%c%c%c') //scan characters

msscanf('0xabc','%x') //scan with hexadecimal format

msscanf('012345abczoo','%[0-9abc]%s')  //[] notation

// reading float and double
msscanf('4345.988','%g')-4345.988 // scan as float
msscanf('4345.988','%lg')-4345.988 // scan as double

//-----------------------------------------------------------
//--    scanning multi-line data files                     --
//-----------------------------------------------------------
//create a file with data
u=mopen(TMPDIR+'/foo','w');
t=(0:0.1:%pi)';mfprintf(u,"%6.3f %6.3f\n",t,sin(t))
mclose(u);

u=mopen(TMPDIR+'/foo','r'); // open the file for reading
//read the file line by line
[n,a,b]=mfscanf(u,'%e %e') //first line using mutiple LHS syntax
l=mfscanf(u,'%e %e')       //second one using single LHS syntax
//use niter to read 5 more lines
l=mfscanf(5,u,'%e %e')

//use niter=-1 to read up to the end of file
l=mfscanf(-1,u,'%e %e')

mclose(u); //close the file

//-----------------------------------------------------------
//--    scanning multi-line  strings vectors               --
//-----------------------------------------------------------
//use niter to scan a string vector
[n,Names,Ages]=msscanf(-1,["Alain 19";"Pierre 15";"Tom 12"],'%s %d')
D=msscanf(-1,["Alain 19";"Pierre 15";"Tom 12"],'%s %d')
typeof(D)
Names=D(:,1) //strings
```

```
Age=D(:,2)   //numerical values
```

## See Also

mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell, mdelete, scanf_conversion

# Name

mget — reads byte or word in a given binary format and convert to double
mgeti — reads byte or word in a given binary format return an int type

```
x=mget([n,type,fd])
x=mgeti([n,type,fd])
```

# Parameters

n
: a positive scalar: The number of items to be read.

fd
: a scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

type
: a string. Give the binary format used to write all the entries of x.

x
: a vector of floating point or integer type numbers

# Description

The `mget` function reads data in the input specified by the stream parameter `fd` and returns a vector of floating point data. The `mgeti` function reads data in the input specified by the stream parameter `fd` and returns a vector of integer data.

Data is read at the position at which the file pointer is currently pointing and advances the indicator appropriately.

The `type` parameter is a conversion specifier which may be set to any of the following flag characters (with default value "l"):

**Note** , On Windows, default behavior is to skip byte 13 (0x0D). mopen should be called with the `b` option, e.g. fd1=mopen(file1,'rb') , so that all bytes will be read without exception.

Data type:

d
: double

f
: float

l
: long

i
: int

s
: short

c
: character

Optional flag:

u..
: unsigned (in combination with one of the above types)

..l
: little endian (in combination with one of the above types)

..b
: big endian (in combination with one of the above types)

Bytes read are automatically swapped if necessary (by checking little=endian status).

This default swapping behavior can be suprressed by adding a flag in the mopen function.

Formats "l", "d", and "f" are only valid with the mget function.

# Examples

```
file1 = 'test1.bin';
file2 = 'test2.bin';
fd1=mopen(file1,'wb');
fd2=mopen(file2,'wb');
mput(1996,'ull',fd1);
mput(1996,'ull',fd2);
mclose(fd1);
mclose(fd2);

fd1=mopen(file1,'rb');
if 1996<>mget(1,'ull',fd1) ;write(%io(2),'Bug');end;
fd2=mopen(file2,'rb');
if 1996<>mget(1,'ull',fd2) ;write(%io(2),'Bug');end;
mclose(fd1);
mclose(fd2);
```

# See Also

mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mgetl, mgetstr, mopen, mprintf, mput, mputl, mputstr, mscanf, mseek, mtell, mdelete

# Name

mgetl — read lines from an ascii file

```
txt=mgetl(file_desc [,m])
```

# Parameters

file_desc
: a character string giving the file name or a logical unit returned by mopen

m
: an integer scalar. number of lines to read. Default value is -1.

txt
: a column vector of string

# Description

`mgetl` function allows to read a lines from an ascii file.

If `m` is omitted or is `-1` all lines till end of file occurs are read.

If `m` is given `mgetl` tries to read exactly m lines. This option is useful to sequentialy read part of a file. In this case if an end of file (EOF) occurs before `m` lines are read the read lines are returned (it is possible to check if EOF had occured using the meof function) issued.

`mgetl` allows to read files coming from Unix, Windows, or Mac operating systems.

# Examples

```
mgetl('SCI/etc/scilab.start',5)

mgetl SCI/modules/elementary_functions/macros/erf.sci

fd=mopen('SCI/etc/scilab.start','r')
mgetl(fd,10)
mclose(fd)
```

# See Also

mputl , mclose , mfscanf , mget , mput , mgetstr , mopen , read

# Authors

S. Steer

# Name

mgetstr — read a character string

```
str=mgetstr(n [,fd] )
```

# Parameters

n
: a positive scalar: The number of character to read.

fd
: scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

str
: a character string

# Description

mgetstr function allows to read a character string in a binary file. If EOF is reached before read completion only the properly read values will be returned.

# Examples

```
// open file descriptor as text with read mode
fd_r = mopen(SCI+'/ACKNOWLEDGEMENTS','rt')

// get 100 characters of fd_r
strs_1 = mgetstr(100, fd_r)

// get 200 next characters of fd_r
strs_2 = mgetstr(200, fd_r)

// close file descriptor
mclose(fd_r);
```

# See Also

mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell, mdelete

# Name

mkdir — Make new directory

```
mkdir('dirname')
mkdir('parentdir','newdir')
status=mkdir( ... )
[status,msg]=mkdir( ... )
```

# Description

mkdir('dirname') creates the directory dirname in the current directory, if dirname represents a relative path. Otherwise, dirname represents an absolute path and mkdir attempts to create the absolute directory dirname

mkdir('parentdir','dirname') creates the directory dirname in the existing directory parentdir, where parentdir is an absolute or relative pathname.

[status,message] = mkdir(...,'dirname') creates the directory dirname in the existing directory parentdir, returning the status, a message. Here, status is 1 for success, 2 if it already exists, -2 if it is a filename and 0 otherwise.

# Examples

```
// Absolute pathname
mkdir(TMPDIR+"/mkdir_example_1")
status_2 = mkdir(TMPDIR+"/mkdir_example_2")
[status_3,msg_3] = mkdir(TMPDIR+"/mkdir_example_3")

// Absolute pathname (parentdir + dirname)
[status_4,msg_4] = mkdir(TMPDIR,"mkdir_example_4")

// Relative pathname
cd TMPDIR;
[status_5,msg_5] = mkdir("mkdir_example_5")
[status_6,msg_6] = mkdir("mkdir_example_5/mkdir_example_6")
```

# See Also

cd, dir, rmdir

# Authors

A.C

# Name

mopen — open a file

```
[fd,err]=mopen(file [, mode, swap ])
```

# Parameters

file
: a character string. The pathname of the file to open.

mode
: a character string that controls whether the file is opened for reading (r), writing (w), or appending (a) and whether the file is opened for updating (+). The mode can also include a b parameter to indicate a binary file.

swap
: a scalar. If `swap` is present and `swap=0` then automatic bytes swap is disabled.

err
: a scalar. Error indicator. see merror.

fd
: scalar. The `fd` parameter returned by the function `mopen` is used as a file descriptor (it's a positive integer).

# Description

`mopen` may be used to open a file in a way compatible with the C `fopen` procedure. Without swap argument the file is supposed to be coded in "little endian IEEE format" and data are swaped if necessary to match the IEEE format of the processor.

The mode parameter controls the access allowed to the stream. The parameter can have one of the following values. In this list of values, the b character indicates a binary file

r
: Opens the file for reading.

rb
: Opens a binary file for reading.

rt
: Opens a text file for reading.

w
: Creates a new file for writing, or opens and truncates a file to zero length.

wb
: Creates a new binary file for writing, or opens and truncates a file to zero length.

wt
: Creates a text binary file for writing, or opens and truncates a file to zero length.

a or ab
: Appends (opens a file for writing at the end of the file, or creates a file for writing).

r+ or r+b
: Opens a file for update (reading and writing).

w+ or w+b
: Truncates to zero length or creates a file for update.

a+ or a+b
: Appends (opens a file for update, writing at the end of the file, or creates a file for writing).

When you open a file for update, you can perform both input and output operations on the resulting stream. However, an output operation cannot be directly followed by an input operation without a file-positioning operation (mseek() function). Also, an input operation cannot be directly followed by an output operation without an intervening file positioning operation, unless the input operation encounters the end of the file.

When you open a file for append (that is, when the mode parameter is a or a+), it is impossible to overwrite information already in the file. You can use the fseek() function to reposition the file pointer to any position in the file, but when output is written to the file, the current file pointer is ignored. All output is written at the end of the file and the file pointer is repositioned to the end of the output.

To open files in a way compatible with Fortran like functions use function `file`.

```
// open a SCI+'/ACKNOWLEDGEMENTS' as text and read only
fd_r = mopen(SCI+'/ACKNOWLEDGEMENTS','rt')

// read five lines of fd_r
mgetl(fd_r, 5)

// another way to read file
// here read five words
mfscanf(5,fd_r,'%s')

// close file descriptor associated to SCI+'/ACKNOWLEDGEMENTS' as text and read
mclose(fd_r);
```

```
// open a file as text with write property
fd_w = mopen(TMPDIR+'/write.txt','wt');

// write a line in fd_w
mputl('This is a line of text', fwd_w);
mclose(fd_w);

// read text
fd_r2 = mopen(TMPDIR+'/write.txt','rt');
mgetl(fd_r2)
mclose(fd_r2);
```

```
// read write a file as binary

// first we write file
fd_wb = mopen(TMPDIR+'/writeread.bin','wb')

// put values as binary
mput(2003,'l',fd_wb);
mput(2008,'i',fd_wb);
mput(2012,'s',fd_wb);
mput(98,'c',fd_wb);

// close file descriptor associated to TMPDIR+'/writeread.bin'
mclose(fd_wb);
```

```
// we read file
fd_rb = mopen(TMPDIR+'/writeread.bin','rb')

mget(fd_rb, 'l')
mget(fd_rb, 'i')
mget(fd_rb, 's')
mget(fd_rb, 'c')

mclose(fd_rb)
```

# See Also

mclose, merror, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetl, mgetstr, mopen, mprintf, mput, mputl, mputstr, mscanf, mseek, mtell, mdelete

# Name

movefile — Move file or directory

```
movefile('source','destination')
[status,message] = movefile('source','destination')
```

# Description

movefile('source','destination') moves the file or directory , source (and subdirectories) to the file or directory, destination.

If source is a directory, destination can not be a file. movefile replaces existing files without warning.

[status, message] = movefile('source','destination') moves source to destination, returning the status and a message.

Whatever the operating system, if the move succeeds, the status is 1 and the message is empty ; if the move fails, the status is 0 and the message is not empty.

# Examples

```
copyfile(SCI+"/etc/scilab.start",TMPDIR+"/scilab.start")
[status,message] = movefile(TMPDIR+"/scilab.start",TMPDIR+"/renamed_scilab.star
```

# See Also

copyfile

# Authors

Allan CORNET

# Name

mput — writes byte or word in a given binary format

```
mput(x [,type,fd])
```

# Parameters

x
: a vector of floating point or integer type numbers

fd
: scalar. The `fd` parameter returned by the function. Default value is -1 which stands for the last (`mopen`) opened file.

type
: a string. Give the binary format used to write all the entries of x.

# Description

The `mput` function writes data to the output specified by the stream parameter `fd`. Data is written at the position at which the file pointer is currently pointing and advances the indicator appropriately.

The `type` parameter is a conversion specifier which may be set to any of the following flag characters (with default value "l"):

"l","i","s","ul","ui","us","d","f","c","uc"
: for writing respectively a long, an int, a short, an unsigned long, an unsigned int, an unsigned short, a double, a float, a char and an unsigned char. The bytes which are wrote are automatically swapped if necessary (by checking little-endian status) in order to produce machine independent binary files ( in little-endian mode). This default swapping mode can be suppressed by adding a flag in the `mopen` function.

"..l" or "..b"
: It is also possible to write in little-endian or big-endian mode by adding a 'l' or 'b' character at the end of a type specification. For example "db" will write a double in big-endian mode.

# Examples

```
filen = 'test.bin';
mopen(filen,'wb');
mput(1996,'l');mput(1996,'i');mput(1996,'s');mput(98,'c');

// force little-endian
mput(1996,'ll');mput(1996,'il');mput(1996,'sl');mput(98,'cl');

// force big-endian
mput(1996,'lb');mput(1996,'ib');mput(1996,'sb');mput(98,'cb');

mclose();
mopen(filen,'rb');
if 1996<>mget(1,'l') then pause,end
if 1996<>mget(1,'i') then pause,end
if 1996<>mget(1,'s') then pause,end
if   98<>mget(1,'c') then pause,end
```

```
// force little-endian
if 1996<>mget(1,'ll') then pause,end
if 1996<>mget(1,'il') then pause,end
if 1996<>mget(1,'sl') then pause,end
if   98<>mget(1,'cl') then pause,end

// force big-endian
if 1996<>mget(1,'lb') then pause,end
if 1996<>mget(1,'ib') then pause,end
if 1996<>mget(1,'sb') then pause,end
if   98<>mget(1,'cb') then pause,end

mclose();
```

# See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetl , mgetstr , mopen , mprintf , mputl , mputstr , mscanf , mseek , mtell , mdelete

# Name

mputl — writes strings in an ascii file

```
r = mputl(txt ,file_desc)
```

# Parameters

r
: returns %t or %f to check if function has correctly written on the file.

file_desc
: A character string giving the name of the file or a logical unit returned by mopen.

txt
: a vector of strings.

# Description

`mputl` function allows to write a vector of strings as a sequence of lines in an ascii file.

# Examples

```
fd = mopen(TMPDIR+'/text_mputl.txt','wt');
mputl('Hello World',fd);
mclose(fd);

fd = mopen(TMPDIR+'/text_mputl.txt','rt');
disp(mgetl(fd));
mclose(fd);
```

# See Also

mget, mgetl, mclose, mfprintf, mput, mputstr, mopen, write

# Authors

S. Steer

Allan CORNET

# Name

mputstr — write a character string in a file

```
mputstr(str [, fd]);
```

# Parameters

fd
: scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

str
: a character string

# Description

mputstr function allows to write a character string in a binary file.

# See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

# Name

mseek — set current position in binary file.

```
mseek(n [,fd, flag])
```

# Parameters

n
: a positive scalar: The offset from origin in number of bytes.

fd
: scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

flag
: a string. specifies the origin. Default value 'set'.

# Description

The function `mseek()` sets the position of the next input or output operation on the stream `fd`. The new position is at the signed distance given by `n` bytes from the beginning, from the current position, or from the end of the file, according to the `flag` value which can be `'set'`, `'cur'` or `'end'`.

`mseek()` allows the file position indicator to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap will return zero until data is actually written into the gap. `mseek()`, by itself, does not extend the size of the file.

# Examples

```
file3='test3.bin'
fd1= mopen(file3,'wb');
for i=1:10, mput(i,'d'); end
mseek(0);
mput(678,'d');
mseek(0,fd1,'end');
mput(932,'d');
mclose(fd1)
fd1= mopen(file3,'rb');
res=mget(11,'d')
res1=[1:11]; res1(1)=678;res1($)=932;
if res1<>res ;write(%io(2),'Bug');end;
mseek(0,fd1,'set');

// trying to read more than stored data
res1=mget(100,'d',fd1);
if res1<>res ;write(%io(2),'Bug');end;
meof(fd1)
mclearerr(fd1)
mclose(fd1);
```

# See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

# Name

mtell — binary file management

```
mtell([fd])
```

# Parameters

fd

scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

# Description

The function `mtell()` returns the offset of the current byte relative to the beginning of the file associated with the named stream `fd`.

# See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

# Name

pathconvert — pathnames convertion between posix and windows.

```
paths=pathconvert(paths [,flagtrail [,flagexpand [,type]]])
```

# Parameters

paths
: a string matrix giving a set of pathnames

flagtrail
: boolean optional parameters. Its default value is TRUE.

flagexpand
: boolean optional parameter. Its default value depends on the MSDOS variable.

type
: a string `'u'` or `'w'`.

# Description

`pathconvert` can be used to convert a set of pathnames (given by a string matrix `paths`) from windows native filename to posix-style pathnames and back. The target style is given by the optional string `type` which can be `'u'` for Unix or `'w'` for Windows. The default style is set according to the value of `MSDOS`. If `MSDOS` is TRUE (resp. FALSE ) then default type is `'w'` (resp. `'u'`).

Windows pathnames starting with `name:` are converted to pathnames starting with `/cygdrive/name/` using the cygwin convention.

`flagtrail` is an optional boolean parameter. When its value is TRUE (default value) a trailing separator (`'\'` or `'/'`) is added at the end of the path if it was missing. If `flagtrail` is set to FALSE, the trailing separator is removed.

`flagexpand` is an optional boolean parameter. When its value is TRUE leading strings like `HOME`, `SCI` or ~ are expanded using environment variables.

# Examples

```
pathconvert("SCI/modules/fileio\macros/foo.sci",%f,%f,"u")
pathconvert("SCI/modules/fileio\macros/foo.sci",%f,%f,"w")
pathconvert("SCI/modules/fileio/macros/foo.sci",%f,%t,"w")
pathconvert("HOME/modules/fileio/macros/foo.sci",%t,%t,"w")
pathconvert("c:/tmp",%f,%t,"u")
pathconvert("/cygdrive/c/tmp",%f,%f,"w")
```

# See Also

basename , listfiles

# Name

pathsep — returns path separator for current platform

```
s = pathsep()
```

# Parameters

s
: a string

# Description

returns path separator. ( ':' on Linux or ';' on Windows )

# Examples

```
pathsep()
```

# Authors

A.C

# Name

pwd — print Scilab current directory
pwd — get Scilab current directory

```
pwd
x=pwd()
```

# Description

`pwd` returns in `ans` the Scilab current directory. `x=pwd()` returns in `x` the Scilab current directory.

# Examples

```
pwd
x=pwd()
```

# See Also

chdir , cd

# Name

removedir — Remove a directory

```
removedir('dirname')
[status] = removedir('dirname','s')
```

# Description

removedir('dirname') removes the directory dirname from the current directory. If the directory is not empty, files and subdirectories are removed. If dirname is not in the current directory, specify the relative path to the current directory or the full path for dirname.

[status] = removedir('dirname') removes the directory dirname and its contents from the current directory, returning the status. Here, status is %T for success and is %F for error.

removedir is used by rmdir.

# Examples

```
createdir(SCIHOME+'/Directory_test')
removedir(SCIHOME+'/Directory_test')
```

# See Also

mkdir , rmdir

# Authors

A.C

# Name

rmdir — Remove a directory

```
rmdir('dirname')
rmdir('dirname','s')
[status,message] = rmdir('dirname','s')
```

# Description

rmdir('dirname') removes the directory dirname from the current directory. If the directory is not empty, you must use the s argument. If dirname is not in the current directory, specify the relative path to the current directory or the full path for dirname.

rmdir('dirname','s') removes the directory dirname and its contents from the current directory.

[status, message] = rmdir('dirname','s') removes the directory dirname and its contents from the current directory, returning the status, and a message. Here, status is 1 for success and is 0 for error.

# Examples

```
mkdir(SCI,'Directory')
rmdir(SCI+'/Directory')
```

# See Also

cd , dir , mkdir

# Authors

A.C

# Name

save_format — format of files produced by "save"

# Description

Variables are saved by Scilab with the save function in the following format:

each variable record is appended consecutively to the file. The variable record begins with 6 long integer holding the variable name in encoded format (see the Remarks section below),

After that comes the variable type (long integer), then, depending on it, for:

Floating matrices (type 1)

row_size m (a long integer),

column_size n (a long integer),

real/complex flag it (a long integer in {0,1}),

data (n*m*(it+1) doubles)

Polynomials (type 2) and Size implicit indices (type 129)

row_size m (a long integer),

column_size n (a long integer),

real/complex flag it (long integer in {0,1}),

formal variable name (16 bytes),

index_table (m*n+1 long integers);

data ((N-1)*(it+1) doubles) , where N is the value of the last entry of the index_table

Booleans (type 4)

row_size m (a long integer),

column_size n (a long integer);

data (n*m long integers)

Floating sparse matrices (type 5)

row_size m (a long integer),

column_size n (a long integer),

real/complex_flag it (a long integer in {0,1}),

total_number_of_non_zero_elements nel (a long integer),

number_of_non_zero_elements_per_row (m long integers),

column_index_non_zero_elements (nel long integers),

non_zero_values (nel*(it+1) doubles)

Boolean sparse matrices (type 6)

row_size m (a long integer),

column_size n (a long integer),

unused it (a long integer),

total_number_of_non_zero_elements nel (a long integer),

number_of_non_zero_elements_per_row (m long integers),

column_index_non_zero_elements (nel long integers)

Matlab sparse matrix (type 7)
row_size m (a long integer),

column_size n (a long integer),

real/complex_flag it (a long integer in {0,1}),

total_number_of_non_zero_elements nel (a long integer),

number_of_non_zero_elements_per_column (n long integers),

row_index_non_zero_elements (nel long integers),

non_zero_values (nel*(it+1) doubles)

Integer matrices (type 8)
row_size m (a long integer),

column_size n (a long integer),

integer_type (a long integer): 1,2,4, or 11,12,14 for signed and unsigned 1,2,4 bytes integers;

data (n*m bytes for integer_type 1 or 11, n*m short integers for integer_type 2 or 12, n*m long integers for integer_type 4 or 14)

handles (type 9)
version (4 bytes)

row_size m (a byte),

column_size n (a byte),

data (m*n serialization_records)

A serialization_record is a flat representation of the C data structure associated with the corresponding graphic object. Each graphic object is defined by a (recursive) set of properties (see the get) function).

The saved serialization_record of a graphic object is structured as follow
type_length n (a byte)

type (n bytes, the ascii codes of the type name)

property_values record (variable length)

Strings (type 10)
row_size m (a long integer),

column_size n (a long integer),

index_table (n*m+1 long integers);

data (N long integers, the Scilab encoding of the characters (see code2str), where N is the value of the last entry of the index_table

Uncompiled functions (type 11)
nout (long integer),

lhs_names (6*nout long integers, see the Remarks section below),

nin (long integer),

rhs_names (6*nin long integers, see the Remarks section below);

code_length N (a long integer),

code (N long integers)

Compiled functions (type 13)
nout (long integer),

lhs_names (6*nout long integers, see the Remarks section below),

nin (long integer),

rhs_names (6*nin long integers, see the Remarks section below),

pseudo_code_length N (a long integer),

pseudo_code (N long integers)

Libraries (type 14)
path_length np (a long integer),

path_name (np long integers: the path character codes sequence, (see code2str)),

number of names nn (long integer),

names (6*nn long integers, see the Remarks section below);

Lists (type 15), tlists (type 16), mlists (type 17)
number of fields n (a long integer),

index (n+1 long integers);

variables_sequence (n variables, each one written according to its format)

Pointers (type 128)
Not handled

Function pointers (type 130)
function_ptr (a long integer,(see funptr))

function_name_code (6 long integers,see the Remarks section below);

# Remarks

Numbers (long interger, short integers, double) are stored using the little endian convention.

The variable names are stored as a sequence of 6 long integers, with a specific encoding. see the cvname.f file for details.

# See Also

save, load, listvarinfile, type, typeof

# Authors

compiled by Enrico Segre

# Name

scanf — Converts formatted input on standard input

```
[v_1,...v_n]=scanf (format);
```

# Parameters

format
: Specifies the format conversion.

# Description

The scanf functions get character data on standard input (%io(1)), interpret it according to a format, and returns the converted results.

The format parameter contains conversion specifications used to interpret the input.

The format parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.

- Any character except % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. see scanf_conversion for details.

# See Also

printf , read , fscanf , sscanf , scanf_conversion

# Name

scanf_conversion — scanf, sscanf, fscanf conversion specifications

# Description

Each conversion specification in the format parameter contains the following elements:

+
: The character % (percent sign)

+
: The optional assignment suppression character *

+
: An optional numeric maximum field width

+
: A conversion code

The conversion specification has the following syntax:

`[*][width][size]convcode`.

The results from the conversion are placed in v_i arguments unless you specify assignment suppression with * (asterisk). Assignment suppression provides a way to describe an input field that is to be skipped. The input field is a string of nonwhite-space characters. It extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion code indicates how to interpret the input field. You should not specify the v_i parameter for a suppressed field. You can use the following conversion codes:

%
: Accepts a single % (percent sign) input at this point; no assignment is done.

d, i
: Accepts a decimal integer;

u
: Accepts an unsigned decimal integer;

o
: Accepts an octal integer;

x
: Accepts a hexadecimal integer;

e,f,g
: Accepts a floating-point number. The next field is converted accordingly and stored through the corresponding parameter, which should be a pointer to a float. The input format for floating-point numbers is a string of digits, with the following optional characteristics:

+
: It can be a signed value.

+
: It can be an exponential value, containing a decimal point followed by an exponent field, which consists of an E or an e followed by an (optionally signed) integer.

+
: It can be one of the special values INF, NaN,

s
: Accepts a string of characters.

c
: character value is expected. The normal skip over white space is suppressed.

%lg
: get value as a double.

# See Also

scanf , scanf , fscanf

# Parte VI. Biblioteca de Gráficos

# Name

GlobalProperty — Customização de aparência dos objetos (curvas, superfícies...) num comando plot ou surf.

```
Nenhuma. GlobalProperty é um argumento opcional que pode ser utilizado dent
PropertyName deve ser um string definido a propriedade a ser ajustada. Prop
Como você pode ver, uma escrita completa dos nomes e valores das propriedad
```

# Descrição

Aqui está uma lista completa dos `PropertyName` que você pode especificar (utilizando plot ou surf) e seus associados disponíveis `PropertyValue`. Se não forem especificadas, estas propriedades estão disponíveis para ambos objetos `Polyline` e `Fac3d` (criados respectivamente por plot e surf) e, como dito anteriormente, eles são aplicados aos novos objetos criados (linhas ou superfícies).

Algumas vezes, você pode ter dois PropertyName correspondentes a uma propriedade: o primeiro é o nome Matlab equivalente, O segundo é o nome padrão utilizado pelo Scilab(i.e.: `Color` ou `Foreground` para uma linha, ver abaixo).

CData or ColorData:
: uma matriz de reais especificando as cores em todos os pontos definidos pela matriz `Z`. TEsta propriedade esta ligada à propriedade `data.color` do objeto (ver surface_properties). Note que esta propriedade está disponível para superfícies apenas.

CDataMapping ou ColorDataMapping:
: um string com valor `'scaled'` ou `'direct'`. Se um `data.color` estiver ajustado, cada índice de cor especifica um único valor para cada vértice. `cdata_mapping` determina se estes índices estão em escala para serem mapeados linearmente no mapa de cores corrente (modo`'scaled'`) ou aponta diretamente para este mapa de cores (modo `'direct'`). Esta propriedade é útil quando `color_flag` é igual a `2`,`3` ou `4`. Note que esta propriedade só existe em entidades `Fac3d`. Note também que `plot3d` tem modo `'direct'` por padrão e `surf` tem modo `'scaled'` por padrão.

Clipping:
: um string `"on"` ou `"off"` definindo o modo de recorte (`"on"` por padrão). É equivalente à propriedade `clip_state` property. Este campo contém a propriedade (ver polyline_properties). Note que esta propriedade ainda não está disponível para entidades de superfícies.

Color ou Foreground:
: um string definindo uma cor conhecida (ver color_list) ou um vetor `1x3` (ou `3x1`) RGB defindo um número de cor. O número de cor é dado como uma tripla `R`, `G`, `B` correspondendo respectivamente a to às intensidades de vermelho, verde e azul entre 0 e 1. Esta propriedade está ligada à propriedade `foreground` do objeto (ver polyline_properties). `Aviso:` Color não está disponível para objetos superfícies. A propriedade `Foreground`existe para objetos superfícies mas está ligada à propriedade do Matlab `EdgeColor` (ver surface_properties).

EdgeColor or Foreground:
: um string definindo uma cor conhecida (ver color_list) ou um vetor `1x3` (or `3x1`) RGB definindo um número de cor. O número de cor é dado pela tripla `R`, `G`, `B` correspondendo respectivamente a to às intensidades de vermelho, verde e azul entre 0 e 1. Esta propriedade está ligada à propriedade `foreground` da superfície (ver surface_properties). `Aviso:` para poligonais a propriedade `Foreground` existe com um signficado diferente (ver acima) e `EdgeColor` não existe.

FaceColor:
: um string com valor `'none'`, `'flat'` ou `'interp'` especificando o modo como as cores das facetas são renderizadas. Quando `'none'` é selecionado, uma malha da superfície é desenhada; se `'flat'` (modo padrão) é selecionado, os valores `Fac3d` `color.data` determinam uma cor

por faceta utilizando a cor do vértice da faceta. Se o valor é `'interp'`, uma gradação interpolada é feita na faceta utilizando `color.data` para determinar a cor em cada vértice da faceta.

LineStyle:
esta propriedade deve ser um string definindo um estilo de linha. Esta propriedade está ligada à propriedade `line_style` do objeto (ver polyline_properties ou surface_properties).

| Especificador | Estilo de linha |
|---|---|
| `-` | linha sólida (padrão) |
| `--` | linha tracejada |
| `:` | linha pontilhada |
| `-.` | linha tracejada-pontilhada |
| `none` | nenhuma linha |

Marker or MarkStyle:
um string definindo um tipo de marca. Note que se você especificar uma marca sem um estilo de linha, ambas linhas (com valor padrão sólido hablitado) e marcas são desenhadas . Esta propriedade está ligada às propriedades `mark_style` e `mark_mode` do objeto (ver polyline_properties ou surface_properties).

| Especificador | Tipo de marca |
|---|---|
| `+` | sinal de mais |
| `o` | círculo |
| `*` | asterísco |
| `.` | ponto |
| `x` | cruz |
| `'square'` or `'s'` | quadrado |
| `'diamond'` or `'d'` | rombo (diamante) |
| `^` | triângulo para cima |
| `v` | triângulo para baixo |
| `>` | triângulo para direita |
| `<` | triângulo para esquerda |
| `'pentagram'` | estrela de cinco pontas (pentagrama) |
| `'none'` | sem marca (padrão) |

MarkerEdgeColor ou MarkForeground:
um string definindo uma cor conhecida(ver color_list) ou um vetor `1x3` (ou `3x1`) RGB definindo um número de cor. O número de cor é dado pela tripla `R`, `G`, `B` correspondendo respectivamente às intensidades de vermelho, verde e azul entre 0 e 1. esta propriedade está ligada à propriedade `mark_foreground` do objeto (ver polyline_properties ou surface_properties).

MarkerFaceColor ou MarkBackground:
um string definindo uma cor conhecida (ver color_list) ou um vetor `1x3` (ou `3x1`) RGB definindo um número de cor. O número de cor é dado pela tripla `R`, `G`, `B` correspondendo respectivamente às intensidades de vermelho, verde e azul entre 0 e 1. Esta propriedade esta ligada à propriedade `mark_background` do objeto (ver polyline_properties ou surface_properties).

MarkerSize ou MarkSize:
um escalar definindo o tamanho do marcador em unidade `point`. Esta propriedade está ligada a propriedade `mark_size` do objeto com `mark_size_unit` habilitado para `"point"` (ver polyline_properties ou surface_properties).

Visible:
um string `"on"` ou `"off"` definindo o modo de visibilidade (`"on"` por padrão). Esta propriedade está ligada à propriedade `visible` do objeto (ver polyline_properties ou surface_properties).

X data:
um vetor ou matriz de reais (re-)definindo os dados fornecidos para todas as linhas ou superfícies esboçadas. A respeito das dimensões, note que estes novos dados devem corresponder a todos os anteriores especificados por `X` isto é, todas as matrizes devem ter as mesmas dimensões. Esta propriedade está ligada à propriedade `data.x` do objeto (ver polyline_properties ou surface_properties).

Y data:
um vetor ou matriz de reais (re-)definindo os dados fornecidos para todas as linhas ou superfícies esboçadas. A respeito das dimensões, note que estes novos dados devem corresponder a todos os anteriores especificados por `Y` data , isto é, todas as matrizes devem ter as mesmas dimensões. Esta propriedade está ligada à propriedade `data.y` do objeto (ver polyline_properties ou surface_properties).

Z data:
quando usado com `plot`, um vetor ou matriz de reais adicionando dados `Z` para todas as linhas esboçadas; com `surf`, uma matriz de reais (re-)definindo os dados fornecidos para todas as superfícies. A respeito das dimensões, note que estes novos dados devem corresponder a todos os anteriores especificados por `X` e `Y` data. Esta propriedade está ligada à propriedade `data.z` do objeto (ver polyline_properties ou surface_properties).

# Exemplos

```
// --------------------
// com o comando plot:
// --------------------
x=1:10; // Init.
plot(x,sin(x),'colo','red','linest','-.','marker','<','markeredg','cyan','marke
clf();

// a ordem das combinações e, {PropertyName,PropertyValue} não importa
plot(x,sin(x),'marker','p','markerfac','cyan','markersiz',10)
clf();

// a combinação de LineSpec com GlobalProperty mostra a predominância de Global
plot(x,x.*x,'*cya--','color','gr','linestyle','-','marker','sq','markersize',6,
clf();

//múltiplos esboços com diferentes LineSpecs e finalmente alguns GlobalProperty
clf();
t=0:%pi/20:2*%pi;
plot(t,sin(t),'ro-.',t,cos(t),'cya+',t,abs(sin(t)),'--mo','markstyl','diam')

// --------------------
// com o comando surf:
// --------------------


Z= [   0.0001    0.0013    0.0053   -0.0299   -0.1809   -0.2465   -0.1100   -0.
    0.0005    0.0089    0.0259   -0.3673   -1.8670   -2.4736   -1.0866   -0.160
    0.0004    0.0214    0.1739   -0.3147   -4.0919   -6.4101   -2.7589   -0.277
   -0.0088   -0.0871    0.0364    1.8559    1.4995   -2.2171   -0.2729    0.836
   -0.0308   -0.4313   -1.7334   -0.1148    3.0731    0.4444    2.6145    2.441
```

```
   -0.0336   -0.4990   -2.3552   -2.1722    0.8856   -0.0531    2.6416    2.406
   -0.0137   -0.1967   -0.8083    0.2289    3.3983    3.1955    2.4338    1.212
   -0.0014   -0.0017    0.3189    2.7414    7.1622    7.1361    3.1242    0.663
    0.0002    0.0104    0.1733    1.0852    2.6741    2.6725    1.1119    0.197
    0.0000    0.0012    0.0183    0.1099    0.2684    0.2683    0.1107    0.019

clf();
f=gcf();
f.figure_size = [610,724];
subplot(211)
surf(Z,'facecol','interp','ydat',101:110,'edgecol','mage')
subplot(212)
surf(Z,'edgeco','b','marker','d','markersiz',9,'markerfac','k','xdata',-50:-41)
```

# Ver Também

LineSpec, plot, surf, clf, polyline_properties, surface_properties

# Autor

F.Leray

# Name

Graphics — Resumo da biblioteca de gráficos

# Desenhos 2d

plot2d
: esboço 2d

plot2d2
: esboço 2d (funções de degraus)

plot2d3
: esboço 2d (barras veticais)

plot2d4
: esboço 2d (setas)

fplot2d
: esboço 2d de uma curva definida por uma função

champ
: esboço de campo vetorial 2d

champ1
: campo vetorial 2d com setas coloridas

fchamp
: campo direcional de uma EDO 2d de primeira ordem

contour2d
: curvas de nível de uma superfície em um esboço 2d

fcontour2d
: curvas de nível de uma superfície definida por uma função em um esboço 2d

grayplot
: esboço 2d de uma superfície utilizando-se cores

fgrayplot
: esboço 2d de uma superfície definida por uma função utilizando cores

Sgrayplot
: esboço 2d suave de uma superfície utilizando cores

Sfgrayplot
: esboço 2d suave de uma superfície definida por uma função utilizando cores

xgrid
: adiciona um grid em um esboço 2d

errbar
: adiciona barras de erro veriticais a um esboço 2d

histplot
: esboça um histograma

Matplot
: esboço 2d de uma matriz utilizando-se cores

# Desenhos 3d

plot3d
: esboço 3d de uma superfície

plot3d1
: esboço 3d em níveis de cinza ou de cores de uma superfície

fplot3d
: esboço 3d de uma superfície definida por uma função

fplot3d1
: esboço 3d em escala de cinza ou colorido de nível de uma superfície definida por uma função

param3d
: esboço 3d de uma curva parametrizada

param3d1
: esboço 3d de curvas parametrizadas

contour
: curvas de nível em uma superfície 3d

fcontour
: curvas de nível sobre uma superfície 3d definida por uma função

hist3d
: representação 3d de um histograma

genfac3d
: computa facetas de uma superfície 3d

eval3dp
: computa facetas 3d de uma superfície parametrizada

geom3d
: projeção 3d para 2d após um esboço 3d

# Desenho de linhas e poligonais

xpoly
: desenha uma poligonal ou um polígono

xpolys
: desenha um conjunto de poligonais ou polígonos

xrpoly
: desenha um polígono regular

xsegs
: desenha segmentos desconexos

xfpoly
: preenche um polígono

xfpolys
: preenche um conjunto de polígonos

# Desenho de retângulos

xrect
: desenha um retângulo

xfrect
: preenche um retângulo

xrects
: desenha ou preenche um conjunto de retângulos

# Desenho de arcos

xarc
: esboça parte de uma elipse

xarcs
: desenha partes de um conjunto de elipses

xfarc
: preenche parte de uma elipse

xfarcs
: preenche partes de um conjunto de elipses

# Desenho de setas

xarrows
: desenha um conjunto de setas

# Strings

xstring
: imprime strings

xstringl
: computa uma caixa que cerca strings

xstringb
: escreve strings em uma caixa

xtitle
: adidciona títulos a janelas de gráficos

titlepage
: adiciona título no meio de uma janela de gráficos

xinfo
: escreve um string de informação na subjanela de mensagens

# Molduras e eixos

graduate
: graduação de eixos

plotframe
: esboça uma moldura com escalas e grides. Esta função está obsoleta.

# Transformações de coordenadas

isoview
: ajusta escalas para esboço isométrico (não muda o tamanho da janela)

square
: ajusta escalas para esboço isométrico (muda o tamanho da janela)

scaling
: transformação afim de um conjunto de pontos

rotate
: rotação de um conjunto de pontos

xsetech
: ajusta a sub-janela de uma janela de gráficos para esboço

subplot
: divide uma janela de gráficos em uma matriz de sub-janelas

xgetech
: retorna a escala de gráficos corrente

xchange
: conversão de coordenadas reais para coordenadas pixeis

# Cores

colormap
: mapa de cores

getcolor
: abre um diálogo que exibe as cores no mapa de cores corrente

addcolor
: adiciona cores novas ao mapa de cores corrente

graycolormap
: mapa de cores linear cinza

hotcolormap
: mapa de cores de vermelho para amarelo

# Contexto gráfico

xset
: ajusta valores para o contexto gráfico. Função obsoleta

xget
: retorna valores correntes do contexto gráfico.

xlfont
: carrega uma fonte em um contexto gráfico ou pergunta a fonte carregada

getsymbol
: Diálogo para a seleção de um símbolo e seu tamanho.

# Gravação e carregamento

xsave
: salva gráficos em um arquivo

xload
: carrega gráficos salvos

xs2fig
: envia gráficos para um arquivo em sintaxe Xfig

xs2gif
: envia gráficos a um arquivo em sintaxe GIF

xs2ppm
: envia gráficos para um arquivo em sintaxe PPM

# Primitivas gráficas

xbasc
: limpa uma janela de gráficos

xclear
: limpa janela de gráficos

driver
: seleciona um driver gráfico

xinit
: inicialização de um driver de gráficos

xend
: termina uma sessão de gráficos

xbasr
: desenha novamente uma janela de gráficos

replot
: redesenha a janela de gráficos corrente com novas fronteiras

xpause
: suspende o Scilab

xselect
: restaura a janela de gráficos corrente

xdel
: deleta uma janela de gráficos

winsid
: retorna a lista de janelas de gráficos

xname
: muda o nome da janela de gráficos corrente

# Posição do mouse

xclick
: espera por um clique do mouse

locate
: seleção pelo mouse de um conjunto de pontos

xgetmouse
: retorna os eventos de mouse e posição corrente

# Editor interativo

edit_curv
: editor interativo de curvas gráficas

gr_menu
: editor gráfico interativo simples

sd2sci
: estrutura gr_menu para conversor de instrução Scilab

# Funções gráficas e controle automático

bode
: diagrama de Bode

gainplot
: esboço de magnitude

nyquist
: diagrama de Nyquist

m_circle
: esboço de um M-círculo

chart
: carta de Nichols

black
: diagrama de Black

evans
: lugar geométrico das raízes Evans

sgrid
: esboça linhas de grid de um s-plano

plzr
: esboço de pólo-zero

zgrid
: esboço de um z-grid

# Name

LineSpec — Customização rápida de linhas que aparecem em um esboço

```
Nenhuma. LineSpec é um argumento opcional que pode ser utilizado dentro de
Para especificar uma linha vermelha de traço longo com marcas de rombo, o s
```

# Descrição

Aqui está uma lista completa dos tipos `LineSpec` que você pode especificar (utilizando plot).

LineStyle:
: um string definindo o estilo de linha. Esta propriedade está ligada à propriedade `line_style` do objeto (ver polyline_properties).

| `especificador` | `estilo de linha` |
|---|---|
| `-` | linha sólida (padrão) |
| `--` | linha tracejada |
| `:` | linha pontilhada |
| `-.` | linha tracejada-pontilhada |

Color:
: um string definindo a cor da linha. Esta propriedade está ligada à propriedade `foreground` do objeto (ver polyline_properties).

| `especificador` | `cor` |
|---|---|
| `r` | vermelho |
| `g` | verde |
| `b` | azul |
| `c` | ciano |
| `m` | magenta |
| `y` | amarelo |
| `k` | preto |
| `w` | branco |

Uma tabela de cores padrão é utilizada para colorir curvas esboçadas se você não especificar as cores (nem com `LineSpec` nem com GlobalProperty). Quando linhas múltiplas são desenhadas, o comando plot automaticamente atribui em ciclos as cores a seguir:

| `R` | `G` | `B` |
|---|---|---|
| 0. | 0. | 1. |
| 0. | 0.5 | 0. |
| 1. | 0. | 0. |
| 0. | 0.75 | 0.75 |
| 0.75 | 0. | 0.75 |
| 0.75 | 0.75 | 0. |
| 0.25 | 0.25 | 0.25 |

Marker type:

um string definindo o tipo do marcador. Note que se você especificar um marcador (ou marca) sem um estilo de linha, apenas o marcador é desenhado. Esta propriedade está ligada à propriedade do objeto `mark_style` e `mark_mode` do objeto (ver polyline_properties).

| especificador | tipo de marcador |
|---|---|
| + | sinal de mais |
| o | círculo |
| * | asterísco |
| . | ponto |
| x | cruz |
| 'square' ou 's' | quadrado |
| 'diamond' ou 'd' | rombo (ou diamante) |
| ^ | triângulo para cima |
| v | triângulo para baixo |
| > | triângulo para a direita |
| < | triângulo para a esquerda |
| 'pentagram' | estrela de cinco pontas (pentagrama) |
| 'none' | nenhum marcador (padrão) |

# Exemplos

```
x=1:0.1:10; // inicialização
plot(x,sin(x),'r.->') // esboçando um linha de traços e pontos com um truângulo
clf();

// se você especificar um marcador sem um estilo de linha, apenas a marca é des
plot(x,sin(x),'d') // esboçando um linha de traços e pontos com um truângulo ap

x=1:10; // Init.
// a ordem das combinações não importa
plot(x,x.*x,'*cya--')

//múltiplos esboçoes com diferentes LineSpecs
clf();
t=0:%pi/20:2*%pi;
plot(t,sin(t),'ro-.',t,cos(t),'cya+',t,abs(sin(t)),'--mo')
```

# Ver Também

GlobalProperty, plot, clf

# Autor

F.Leray

# Name

Matplot — esboço 2d de uma matriz utilizando-se cores

```
Matplot(a,[strf,rect,nax])
Matplot(a,<opt_args>)
```

# Parâmetros

a
: matriz de reais de tamanho (n1,n2).

<opt_args>
: representa uma seqüência de declarações `key1=value1, key2=value2`,... onde `key1`, `key2,...` podem ser um dos seguintes:

  rect
  : ajusta as fronteiras do esboço. Se esta chave é fornecida, nem `frameflag` ou `strf` é fornecido, então o caractere `y` de `strf` é suposto como `7`. Ver valores abaixo.

  nax
  : ajusta a definição dos grides. Se esta chave é fornecida, nem `axesflag` ou `strf` é fornecido, então o caractere `z` cd `strf` é suposto como `1`. Ver valores abaixo.

  frameflag
  : especifica como a moldura do esboço é computado. O valor é um inteiro entre 0 e 8. Corresponde ao caractere `y` de `strf`. Ver abaixo.

  axesflag
  : especifica que tipo de eixos são desenhados ao redor do esboço. O valor é um inteiro entre 0 e 5. Corresponde ao caractere `z` de `strf`. Ver abaixo.

strf
: é um string de comprimento 3 `"xyz"`.

  default
  : o padrão é `"081"`.

  x
  : controls the display of captions.

    x=0
    : sem legendas.

    x=1
    : legendas são exibidas. Elas são dadas pelo argumento opcional `leg`.

  y
  : controla a computação dos intervalos de coordenadas reais a partir dos valores mínimos requeridos. Intervalos reais podem ser maiores que os requerimentos mínimos.

    y=0
    : sem computação, o esboço utiliza a escala anterior (ou padrão)

    y=1
    : a partir do argumento rect

    y=2
    : a partir dos valores mínimo/máximo dos dados x e y

y=3
constrói uma escala isométrica a partir do argumento rect

y=4
constrói um esboço em escala isométrica a partir dos valores mínimo/máximo dos dados x e y

y=5
aumentados para obtenção de bons eixos a partir do argumento rect

y=6
aumentados para obtenção de bons eixos a partir dos valores mínimo/máximo dos dados x e y

y=7
como y= 1 mas os anteriores são redesenhados para se utilizar a nova escala

y=8
como y= 2 mas os anteriores são redesenhados para se utilizar a nova escala

z
controla a exibição de informação na moldura ao redor do esboço. Se eixos são requeridos, o número de tiques pode ser especificado pelo argumento opcional `nax`.

z=0
nada é desenhado ao redor do esboço.

z=1
eixos são desenhados, o eixo y é exibido à esquerda.

z=2
o esboço é cercado por uma caixa sem tiques.

z=3
eixos são desenhados, o eixo y é exibido à direita.

z=4
eixos são desenhados centrados no meio da caixa de moldura.

z=5
eixos são desenhados cruzando-se no ponto `(0,0)`. Se o ponto `(0,0)` não estiver dentro da moldura, os eixos não aparecerão no gráfico.

rect
este argumento é utilizado quando o segundo caractere y do argumento `strf` é 1, 3 ou 5. É um vetor linha de tamanho quatro e fornece a dimensão da moldura: `rect=[xmin,ymin,xmax,ymax]`.

nax
este argumento opcional é utilizado quando o terceiro caractere z do argumento `strf` é 1. É um vetor linha de quatro entradas `[nx,Nx,ny,Ny]` onde `nx` (`ny`) é o número de sub-graduações no eixo x (y) axis e `Nx` (`Ny`) é o número de graduações no eixo x (y).

# Descrição

As entradas da matriz `int(a)` são utilizadas como entradas do mapa de cor no mapa de cores corrente. A cor associada a `a(i,j)` é utilizada para desenhar um pequeno quadrado de lado 1 com centro localizado em `(x=j,y=(n1-i+1))`.

Entre com o comando `Matplot()` para visualizar uma demonstração.

# Exemplos

```
Matplot([1 2 3;4 5 6])
clf()
// desenhando o mapa de cores corrente
Matplot((1:xget("lastpattern")))
```

# Ver Também

colormap, plot2d, Matplot1, Matplot_properties

# Autor

J.Ph.C.

# Name

Matplot1 — Esboço 2d de matrizes utilizando cores

```
Matplot1(a,rect)
```

# Parâmetros

a
: matriz de reais de tamanho (n1,n2).

rect
: `[xmin,ymin,xmax,ymax]`

# Descrição

As entradas da matriz `int(a)` são utilizadas como entradas de mapa de cores no mapa de cores corrente. `rect` especifica um retângulo na escala corrente e a matriz é desenhada dentro deste retângulo. Cada entrada da matriz será renderizada como um pequeno retângulo preenchido com sua cor associada.

# Exemplos

```
//--- primeiro exemplo
clf();
ax=gca();//obtendo o manipulador dos eixos correntes
ax.data_bounds=[0,0;10,10];//set the data_bounds
ax.box='on'; //desenha uma caixa
a=5*ones(11,11); a(2:10,2:10)=4; a(5:7,5:7)=2;
// primeira matriz no retângulo [1,1,3,3]
Matplot1(a,[1,1,3,3])
a=ones(10,10); a= 3*tril(a)+ 2*a;
// segunda matriz no retângulo [5,6,7,8]
Matplot1(a,[5,6,7,8])

//--- segundo exemplo (animação)
n=100;

clf();
f=gcf();//obtendo o manipulador da figura corrente
f.pixmap='on';//modo de buffer duplo
ax=gca();//obtendo o manipulador dos eixos corrente
ax.data_bounds=[0,0;10,10];//ajustando the data_bounds
ax.box='on'; //desenhando uma caixa
show_pixmap()
for k=-n:n,
  a=ones(n,n);
  a= 3*tril(a,k)+ 2*a;
  a= a + a';
  k1= 3*(k+100)/200;
  if k>-n then delete(gce()),end
  Matplot1(a,[k1,2,k1+7,9])
  show_pixmap() //enviando o buffer duplo para a tela
end
```

# Ver Também

colormap, plot2d, Matplot, grayplot, Matplot_properties

# Autor

J.Ph.C.

# Name

Matplot_properties — Descrição das propriedades da entidade Matplot

# Descrição

A entidade Matplot é uma folha na hierarquia de entidades gráficas. Representa esboços 2d de superfícies utilizando cores e imagens (ver `Matplot` e `Matplot1`).

parent:
: esta propriedade contém o manipulador da raiz. A raiz de uma entidade Matplot deve ser do tipo `"Axes"`.

children:
: esta propriedade contém um vetor com os galhos do manipulador. Contudo, manipuladores `Matplot` handles correntemente não possuem galhos.

visible:
: este campo contém o valor da propriedade `visible` para a entidade . Pode ser `"on"` ou `"off"` . Por padrão, o esboço é visível, o valor da propriedade é `"on"`. Se `"off"` o esboço não é desenhado na tela.

data:
: este campo define uma matriz de cores [mxn] `color` utilizando o mapa de cores corrente. A cor associada a `color(i,j)` é utilizanda para se esboçar um pequeno quadrado de lado 1 com centro localizado em `(x=j,y=(m-i+1))`.

clip_state:
: este campo contém o valor da propriedade `clip_state` para o Matplot. o valor de clip_state pode ser :

- `"off"` significa que o Matplot não é recortado.
- `"clipgrf"` significa que o Matplot é recortado fora da caixa dos eixos.
- `"on"` significa que o Matplot é recortado fora do retângulo dado pela propriedade clip_box.

clip_box:
: este campo determina a propriedade `clip_box`. Por padrão seu valor é uma matriz vazia se a propriedade clip_state é "off". Em outros casos, o vetor `[x,y,w,h]` (ponto superior esquerdo, largura, altura) define as porções do retângulo a ser exibido, contudo o valor da propriedade `clip_state` será alterado.

user_data:
: este campo pode armazenar qualquer variável Scilab na estrutura de dados do Matplot e recuperá-la.

# Exemplos

```
Matplot((1:xget("lastpattern")))
e=gce(); // obtendo a entidade corrente

e.data=e.data($:-1:1) // ordem inversa
```

# Ver Também

set, get, delete, grayplot, Matplot, Matplot1, graphics_entities, grayplot_properties

# Autor

F.Leray

# Name

Sfgrayplot — esboço 2d suave de uma superfície definida por uma função utilizando cores

```
Sfgrayplot(x,y,f,<opt_args>)
Sfgrayplot(x,y,f [,strf, rect, nax, zminmax, colminmax, mesh, colout])
```

# Parâmetros

x,y
: vetores linhas de reais de tamanhos n1 e n2.

f
: função do Scilab (z=f(x,y))

<opt_args>
: representa uma seqüência de declarações `key1=value1, key2=value2`,... onde `key1, key2, ...` podem ser um dos seguintes: strf, rect, nax, zminmax, colminmax, mesh, colout (ver plot2d para os três primeiros e fec para os quatro últimos).

strf,rect,nax
: ver plot2d.

zminmax, colminmax, mesh, colout
: ver fec.

# Descrição

`Sfgrayplot` é o mesmo que `fgrayplot` mas o esboço é suavizado. A função `fec` é utilizada para suavização. A superfície é esboçada assumindo-se que é linear em um conjunto de triângulos construídos a partir do grid (aqui, com n1=5, n2=3):

```
_____________
| /| /| /| /|
|/_|/_|/_|/_|
| /| /| /| /|
|/_|/_|/_|/_|
```

A função colorbar pode ser utilizada para se visualizar a escala de cores (mas você deve saber (ou computar) os valores mínimo e máximo).

Ao invés de Sfgrayplot, você pode usar Sgrayplot este pode ser um pouco mais rápido.

Entre com o comando `Sfgrayplot()` para visualizar uma demonstração.

# Exemplos

```
// exemplo #1: esboço de 4 superfícies
function z=surf1(x,y), z=x*y, endfunction
function z=surf2(x,y), z=x^2-y^2, endfunction
function z=surf3(x,y), z=x^3+y^2, endfunction
function z=surf4(x,y), z=x^2+y^2, endfunction
xbasc()
xset("colormap",[jetcolormap(64);hotcolormap(64)])
x = linspace(-1,1,60);
y = linspace(-1,1,60);
```

```
drawlater() ;
subplot(2,2,1)
   colorbar(-1,1,[1,64])
   Sfgrayplot(x,y,surf1,strf="041",colminmax=[1,64])
   xtitle("f(x,y) = x*y")
subplot(2,2,2)
   colorbar(-1,1,[65,128])
   Sfgrayplot(x,y,surf2,strf="041",colminmax=[65,128])
   xtitle("f(x,y) = x^2-y^2")
subplot(2,2,3)
   colorbar(-1,2,[65,128])
   Sfgrayplot(x,y,surf3,strf="041",colminmax=[65,128])
   xtitle("f(x,y) = x^3+y^2")
subplot(2,2,4)
   colorbar(0,2,[1,64])
   Sfgrayplot(x,y,surf4,strf="041",colminmax=[1,64])
   xtitle("f(x,y) = x^2+y^2")
drawnow() ;
xselect()

// exemplo #2: esboço de surf3 e adição de algumas linhas de contorno
function z=surf3(x,y), z=x^3+y^2, endfunction
xbasc()
x = linspace(-1,1,60);
y = linspace(-1,1,60);
xset("colormap",hotcolormap(128))
drawlater() ;
colorbar(-1,2)
Sfgrayplot(x,y,surf3,strf="041")
fcontour2d(x,y,surf3,[-0.1, 0.025, 0.4],style=[1 1 1],strf="000")
xtitle("f(x,y) = x^3+y^2")
drawnow() ;
xselect()

// exemplo #3: esboço de surf3 e uso dos argumentos opcionais zminmax e colout
//             para restringir o esboço em -0.5&lt;= z &lt;= 1
function z=surf3(x,y), z=x^3+y^2, endfunction
xbasc()
x = linspace(-1,1,60);
y = linspace(-1,1,60);
xset("colormap",jetcolormap(128))
drawlater() ;
zminmax = [-0.5 1]; colors=[32 96];
colorbar(zminmax(1),zminmax(2),colors)
Sfgrayplot(x, y, surf3, strf="041", zminmax=zminmax, colout=[0 0], colminmax=co
fcontour2d(x,y,surf3,[-0.5, 1],style=[1 1 1],strf="000")
xtitle("f(x,y) = x^3+y^2, com partes abaixo de z = -0.5 e acima de z = 1 removi
drawnow() ;
xselect()
```

# Ver Também

fec, fgrayplot, grayplot, Sgrayplot

# Autor

J.Ph.C.

# Name

Sgrayplot — esboço 2d suave de uma superfície utilizando cores

```
Sgrayplot(x,y,z,<opt_args>)
Sgrayplot(x,y,z [,strf, rect, nax, zminmax, colminmax, mesh, colout])
```

# Parâmetros

x,y
: vetores linhas de reais de tamanhos n1 e n2.

z
: matriz de reais de tamanho (n1,n2). `z(i,j)` é o valor da superfície no ponto (x(i),y(j)).

<opt_args>
: representa uma seqüência de declarações `key1=value1,key2=value2`,... onde `key1`, `key2,...` podem ser um dos seguintes: strf, rect, nax, zminmax, colminmax, mesh, colout.

strf
: é um string de comprimento 3 `"xyz"` (por padrão `strf= "081"`)

  x
  : controla a exibição de legendas

    x=0
    : sem legendas.

    x=1
    : legendas são exibidas. Elas são dadas pelo argumento opcional `leg`.

  y
  : controla a computação dos intervalos de coordenadas reais a partir dos valores mínimos requeridos. Intervalos reais podem ser maiores que os requerimentos mínimos.

    y=0
    : sem computação, o esboço utiliza a escala anterior (ou padrão)

    y=1
    : a partir do argumento rect

    y=2
    : a partir dos valores mínimo/máximo dos dados x e y

    y=3
    : construídos para uma escala isométrica a partir do argumento rect

    y=4
    : construídos para uma escala isométrica a partir dos valores mínimo/máximo dos dados x e y

    y=5
    : aumentados para obtenção de bons eixos a partir do argumento rect

    y=6
    : aumentados para obtenção de bons eixos a partir dos valores mínimo/máximo dos dados x e y

    y=7
    : como y= 1 mas os anteriores são redesenhados para se utilizar a nova escala

y=8
: lcomo y= 2 mas os anteriores são redesenhados para se utilizar a nova escala

z
: controla a exibição de informação na moldura ao redor do esboço. Se eixos são requeridos, o número de tiques pode ser especificado pelo argumento opcional `nax`.

z=0
: nada é desenhado ao redor do esboço.

z=1
: eixos são desenhados, o eixo y é exibido à esquerda.

z=2
: o esboço é cercado por uma caixa sem tiques.

z=3
: eixos são desenhados, o eixo y é exibido à direita.

z=4
: eixos são desenhados centrados no meio da caixa de moldura.

z=5
: eixos são desenhados cruzando-se no ponto `(0,0)`. Se o ponto `(0,0)` não estiver dentro da moldura, os eixos não aparecerão no gráfico.

rect
: este argumento é utilizado quando o segundo caractere y do argumento `strf` é 1, 3 ou 5. É um vetor linha de tamanho quatro e fornece a dimensão da moldura: `rect=[xmin,ymin,xmax,ymax]`.

nax
: este argumento opcional é utilizado quando o terceiro caractere z do argumento `strf` ié 1. É um vetor linha de quatro entradas `[nx,Nx,ny,Ny]` onde `nx` (`ny`) é o número de sub-graduações no eixo x (y) axis e `Nx` (`Ny`) é o número de graduações no eixo x (y).

zminmax, colminmax, mesh, colout
: ver fec.

# Descrição

`Sgrayplot` é o mesmo que `grayplot` mas o esboço é suavizado. A função `fec` é utilizada para suavização. A superfície é esboçada assumindo-se que é linear em um conjunto de triângulos construídos a partir do grid (aqui, com n1=5, n2=3):

```
_____________
| /| /| /| /|
|/_|/_|/_|/_|
| /| /| /| /|
|/_|/_|/_|/_|
```

A função colorbar pode ser utilizada para se visualizar a escala de cores.

O parâmetro `zminmax` é útil para propósitos de animação (veja um exemplo depois) e o parâmetro `colminmax` permite que o usuário escolha uma parte do mapa de cores corrente (ver a página de ajuda de fec).

Entre com o comando `Sgrayplot()` para visualizar uma demonstração.

# Exemplos

```
// exemplo #1
x=-10:10; y=-10:10;m =rand(21,21);
clf()
xset("colormap",hotcolormap(64))
Sgrayplot(x,y,m, strf="011", rect=[-20,-20,20,20])

// exemplo #2
t=-%pi:0.1:%pi; m=sin(t)'*cos(t);
clf()
xset("colormap",jetcolormap(64))
colorbar(-1,1)
Sgrayplot(t,t,m, strf="041")

// exemplo #3: exibindo animação de cos(t)*sin(x)sin(y).
n = 30;
nt = 100;
x = linspace(0,2*%pi,n);
y = linspace(0,%pi,n/2);
z = sin(x')*sin(y);
t = linspace(0,4*%pi,nt);
xselect(); clf()
f=gcf();
f.color_map=jetcolormap(64);
f.pixmap='on';
colorbar(-1,1)
Sgrayplot(x,y,cos(t(1))*z, strf="042", zminmax=[-1,1])
c=gce(),e=c.children
xtitle("Olhos de Kaa")
for i = 1:nt
  e.data(:,3)=matrix(cos(t(i))*z,-1,1);
  show_pixmap()
end
f.pixmap='off';
```

# Ver Também

fec, fgrayplot, grayplot, Sfgrayplot, colorbar

# Autor

J.Ph.C.

# Name

addcolor — adiciona cores novas ao mapa de cores corrente

```
new=addcolor(c)
```

# Parâmetros

new
: identificadores das cores definidas em c em uma nova tabela de cores

c
: matriz com 3 colunas, definição de cores RGB (vermelho, verde e azul)

# Descrição

`addcolor` adiciona novas cores fornecidas no argumento `c` ao mapa de cores corrente. `c` deve ser uma matriz com três colunas `[R G B]. R` é o componente vermelho, `G` é o componente verde, `B` é o componente azul). cada entrada em `c` deve ser um número não-negativo menor do que ou igual a 1.

Os identificadores das novas cores são retornados em `new`.

Se uma cor definida em `c` já está presente no mapa de cores, ela não será adicionada.

# Examplos

```
plot3d();
h = gcf();
h.color_map = jetcolormap(16);
addcolor(name2rgb('grey')/255);
```

# Ver Também

colormap, name2rgb

# Name

alufunctions — Funções de desenho de pixel

# Descrição

src is the source ie the "value of the pixel" which we want to draw. dst is the destination ie "value of the pixel" which is already drawn.

0
: clear ie "0" (limpo)

1
: and ie "src AND dst" (E)

2
: and reverse ie "src AND NOT dst" (E NÃO, "e reverso")

3
: copy ie "src" (cópia)

4
: and inverted ie "(NOT src) AND dst" (NÃO, E; "e invertido")

5
: noop ie "dst" ()

6
: xor ie "src XOR dst" (UM OU OUTRO)

7
: or ie "src OR dst" (OU)

8
: nor ie "(NOT src) AND (NOT dst)" (NÃO, E NÃO; nem um, nem outro)

9
: equiv ie "(NOT src) XOR dst" (NÃO UM OU O OUTRO; equivalência)

10
: invert ie "NOT dst" (NÃO)

11
: or reverse ie "src OR (NOT dst)" (OU NÃO; ou reverso)

12
: copy inverted ie "NOT src" (cópia invertida)

13
: or inverted ie "(NOT src) OR dst" (NÃO, OU invertido)

14
: nand ie "(NOT src) OR (NOT dst)" (NÃO OU NÃO, pelo menos um não)

15
: set ie "1" (ajustado)

# Name

arc_properties — Descrição das propriedades da entidade Arc (arco)

# Descrição

A entidade Arc é uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros para elipses e partes de elipses preenchidas ou não.

parent:
: esta propriedade contém o manipulador da raiz. A raiz de uma entidade Arc deve ser do tipo `"Axes"` ou `"Compound"`.

children:
: esta propriedade contém um vetor com os galhos do manipulador. Contudo, manipuladores arc não possuem galhos correntemente.

thickness:
: este campo contém a propriedade de linha `thickness` (espessura). Deve ser um inteiro positivo.

line_style:
: o valor da propriedade `line_style` deve ser um inteiro em [1 6]. 1 significa linha sólida e os demais valores significam tipos diferentes de tracejados.

line_mode:
: esta propriedade permite que se exiba ou não a linha representando o arco. O valor pode ser `"on"` ou `"off"`.

fill_mode:
: se o valor da propriedade `fill_mode` é `"on"` , o arco é preenchido com a cor `background` (de plano de fundo).

foreground:
: este campo contém o valor padrão da propriedade `foreground` (primeiro plano) para desenhar ou preencher o arco. Deve ser um índice de cor (relativo ao mapa de cores corrente).

background:
: este campo contém a cor usada para preencher o arco. Deve ser um índice de cor (relativo ao mapa de cores corrente).

data:
: esta propriedade retorna as coordenadas do ponto superior esquerdo, a largura e a altura do retângulo envolvente bem como os ângulos de froenteira do setor. É a matriz `[xleft,yup,[zup],width,height,a1,a2]` onde `a1` e `a2` são os ângulos de fronteira dados em graus.

  **Aviso:** nas versões do Scilab até a 4.1.2 `a1` e `a2` são dados em degree/64.

visible:
: este campo contém o valor da propriedade `visible` para a entidade. Pode ser `"on"` ou `"off"`. Se `"on"` o arco é desenhado, se `"off"` o arco não é exibido na tela.

arc_drawing_method:
: Este campo controla o tipo de discretização utilizado para renderizar o arco. Seu valor deve ser `"nurbs"` ou `"lines"`. Se `"nurbs"` for selecionado, o arco é renderizado utilizando-se superfícies e curvas nurbs. Isto resulta na exibição de uma superfície perfeita, não importa qual o ponto de vista seja. Se `"lines"` é selecionado, então o arco é aproximado com um número constante de linhas. Isto reduz o tempo de desenho, mas pontas agudas podem aparecer em uma ampliação. O uso do valor `"lines"` é desencorajado e só deve ser utilizado se for notada alguma perda na velocidade dos quadros ao se utilizar o valor `"nurbs"`.

clip_state:
este campo contém o valor da propriedade `clip_state` para o arco. O valor de clip_state pode ser :

- `"off"` significa que o arco não é recortado
- `"clipgrf"` significa que o arco é recortado fora da caixa dos eixos.
- `"on"` significa que o arco é recortado fora do arco dado pela propriedade clip_box.

clip_box:
este campo determina a propriedade `clip_box` property. Por padrão seu valor é uma matriz vazia se a propriedade clip_state é "off". Em outros casos, o vetor `[x,y,w,h]` (ponto superior esquerdo, largura e altura) define as porções do arco a serem exibidas, contudo o valor da propriedade `clip_state` será alterado.

user_data:
este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados da entidade arco e recuperá-la.

# Exemplos

```
a=get("current_axes");//obtendo o manipulador dos novos eixos criados
a.data_bounds=[-2,-2;2,2];

xarc(-1.5,1.5,3,3,0,360*64)

arc=get("hdl"); //obtendo o manipulador da entidade corrente (aqui é a entidade
arc.fill_mode="on";
arc.foreground=5;
arc.data(:,[3 6])=[2 270*64];
xfarc(-.5,1,.4,.6,0,360*64);
arc.visible="off";
```

# Ver Também

set, get, delete, xarc, xarcs, xfarc, xfarcs, graphics_entities

# Autor

Djalel ABDEMOUCHE

Jean-Baptiste SILVY

# Name

autumncolormap — Mapa de cores com tons do outono (vermelho, laranja, amarelo)

```
cmap=autumncolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`autumncolormap` computa um mapa de cores com `n` cores com tons do outono.

# Exemplos

```
f = scf();
plot3d1();
f.color_map = autumncolormap(32);
```

# Ver Também

colormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap

# Name

axes_properties — descrição das propriedades de entidades Axes (de eixos)

# Descrição

A entidade Axes é o segundo nível da hierarquia das entidades de gráfico. Esta entidade define os parâmetros que permitem a mudança de coordenadas e o esboço dos eixos tanto quanto os valores padrões dos parâmetros para criação de entidades galhos

Propriedades dos eixos

parent:
: este campo contém o manipulador da figura raiz

children:
: Fvetor FA contendo os manipuladores de todos os objetos gráficos galhos dos eixos. Estes objetos gráficos são do tipo `"Compound"`, `"Rectangle"`, `"Polyline"`, `"Segs"`, `"Arc"`, `"Grayplot"`,.. (ver Compound_properties, rectangle_properties, champ_properties, axis_properties, polyline_properties, segs_properties, grayplot_properties, surface_properties, param3d_properties, fec_properties, text_properties, legend_properties)

visible:
: este campo contém o valor da propriedade `visible` (visível) para os eixos. O valor pode ser `"on"` (ligado) ou `"off"` . (desligado). Por padrão, os eixos são visíveis, `"on"` , caso todos os galhos visíveis sejam exibidos na tela. Se `"off"` , os eixos e seus galhos não são esboçados.

axes_visible:
: um vetor de strings `1x3` string vector. Esta propriedade especifica se o eixo deve ser desenhado ou não. Seu valor pode ser `"on"` ou `"off"` para um ajuste global. Para agir em um único eixo, a sintaxe é `axes_visible(N)` onde `N` é `1,2` ou `3` correspondentes aos eixos `x,y` ou `z`.O dado de escala e, se requerido, os grides são desenhados se o valor for `"on"`. Note que ao criar uma simples entidade de eixo usando os comandos `gca()` (atalho para `get"current_axes")`) ou `gcf()` (atalho para `get(current_figure)`), a visibilidade dos eixos é ajustada para `"off"`.

axes_reverse:
: um vetor de strings `1x3` correspondentes aos três eixos (X,Y,Z). Para cada eixo, a propriedade especifica a direção dos valores crescentes. Se `"off"`, a direção padrão é usada. Se `"on"`, a direção é invertida. Também é possível utilizar apenas um string, `"on"` ou `"off"`, para ajustar simultaneamente os três dados.

grid:
: o valor do campo é um vetor `[x-grid,y-grid,z-grid]` onde `x-grid` controla o esboço para o eixo x, e `y-grid, z-grid` respectivamente para os eixos y e z. O valor padrão é `-1`, os grides não são desenhados, caso contrário, os grides são desenhados utilizando a cor dada indexada pelo valor do grid.

grid_position:
: Este string determina a posição do grido comparada às outras entidades gráficas. Seu valor pode ser `"foreground"` para desenhar o grid à frente das outras entidades ou `"background"` para desenhar o grid atrás.

x_location:
: especifica a localização do eixo y. Os valores possíveis são:

- `"bottom"`. Neste caso, o eixo y é desenhado na parte direita do retângulo de eixos.

- `"top"`. Neste caso, o eixo y é desenhado na parte esquerda do retângulo de eixos.
- `"middle"` . Neste caso, o eixo y é desenhado na posição mais próxima da coordenada 0 do eixo x.

y_location:
especifica a localização do eixo y. Os valores possíveis são:

- `"left"`. Neste caso, o eixo y é desenhado na parte direita do retângulo de eixos.
- `"right"`. Neste caso, o eixo y é desenhado na parte esquerda do retângulo de eixos.
- `"middle"`. Neste caso, o eixo y é desenhado na posição mais próxima da coordenada 0 do eixo x.

title:
um objeto anexado à entidade de eixos que retorna um manipulador gráfico de uma estrutura label (rótulo) (ver label_properties). Este campo define um título com opções sobre este label.

x_label:
um objeto anexado à entidade de eixos que retorna um manipulador gráfico de uma estrutura label (ver label_properties). Este campo define um label sobre o eixo x com opções sobre este label.

y_label:
um objeto anexado à entidade de eixos que retorna um manipulador gráfico de uma estrutura label (ver label_properties). Este campo define um label sobre o eixo y com opções sobre este label.

z_label:
um objeto anexado à entidade de eixos que retorna um manipulador gráfico de uma estrutura label (ver label_properties). Este campo define um label sobre o eixo z com opções sobre este label.

auto_ticks:
um vetor de strings `1x3` fornecendo o status auto_ticks para cada eixo. Esta propriedade especifica se cada eixo é graduado utilizando-se um algoritmo computacional ou não (graduações feitas pelo usuário). Os valores podem ser `"on"` ou `"off"` para ajuste global. Para agir em um único eixo, a sintaxe é `auto_ticks(N)` onde `N` é `1,2` ou `3` correspondentes aos eixos `x,y` ou `z`. Note que a edição de tiques (textos e/ou localizações) via `x_ticks, y_ticks` ou `z_ticks` ajusta automaticamente `auto_ticks` para `"off"` para os eixos correspondentes.

x_ticks.locations:
um vetor de reais contendo a localização das graduações sobre o eixo x. Esta propriedade pode ser editada especificando-se um novo vetor de reais (de mesmo tamanho). Para especificar graduações maiores ou menores, pode-se agir no tlist `x_ticks` definindo também um vetor de strings `x_ticks.labels`.

y_ticks.locations:
um vetor de reais contendo a localização das graduações sobre o eixo y. Esta propriedade pode ser editada especificando-se um novo vetor de reais (de mesmo tamanho). Para especificar graduações maiores ou menores, pode-se agir no tlist `y_ticks` definindo também um vetor de strings `y_ticks.labels`.

z_ticks.locations:
um vetor de reais contendo a localização das graduações sobre o eixo z. Esta propriedade pode ser editada especificando-se um novo vetor de reais (de mesmo tamanho). Para especificar graduações maiores ou menores, pode-se agir no tlist `z_ticks` definindo também um vetor de strings `z_ticks.labels`.

x_ticks.labels:
um vetor de strings contendo rótulos para as graduações no eixo x. Esta propriedade pode ser editada especificando-se um novo vetor (de mesmo tamanho). Para especificar graduações menores ou maiores, pode se agir no tlist `x_ticks` definindo também um vetor de reais `x_ticks.locations` .

y_ticks.labels:
um vetor de strings contendo rótulos para as graduações no eixo y. Esta propriedade pode ser editada especificando-se um novo vetor (de mesmo tamanho). Para especificar graduações menores ou maiores, pode se agir no tlist `y_ticks` definindo também um vetor de reais `y_ticks.locations` .

z_ticks.labels:
um vetor de strings contendo rótulos para as graduações no eixo z. Esta propriedade pode ser editada especificando-se um novo vetor (de mesmo tamanho). Para especificar graduações menores ou maiores, pode se agir no tlist `z_ticks` definindo também um vetor de reais `z_ticks.locations` .

box:
esta propriedade especifica se os eixos devem ser encerrados por uma caixa. Seu valor pode ser `"off"`, `"hidden_axes"`, `"back_half"` ou `"on".` Se o valor for `"off"`, a caixa não é desenhada. Se a propriedade for `"hidden_axes"`, apenas a moldura de trás é desenhada. Se o valor for `"back_half"`, os eixos X, Y e Z são desenhados também. Se a propriedade for `"on"` , a caixa toda é desenhada.

filled:
Esta propriedade especifica se o plano de fundo dos eixos devem ser desenhados ou não. Seu valor pode ser `"off"` ou `"on".` Se o valor for `"off"`, o plano de fundo não é desenhado e a caixa de eixos é transparente. Se a propriedade for `"on"` , o plano de fundo é desenhado utilizando-se a cor especificada pela propriedade `background`.

sub_ticks:
esta propriedade determina o número de subtiques a serem desenhados entre dois tiques principais. O valor de campo é o vetor `[nx,ny]` onde `nx` é o número de subtiques para o eixo x e `ny` é o correspondente para o eixo y.

font_style:
especifica a fonte a ser utilizada para exibir os rótulos. É um inteiro positivo que faz referência a uma das fontes carregadas. Seu valor deve estar entre 0, referenciando a primeira fonte e o número de fontes carregadas menos um, referenciando a última fonte carregada. Para mais informações, ver graphics_fonts.

font_size:
um inteiro especificando o tamanho da fonte usada para para os rótulos dos tiques. Se a propriedade `fractional_font` for `"off"` , apenas a parte inteira do valor é utilizada. Para mais informações, ver graphics_fonts.

font_color:
esta propriedade determina a cor dos rótulos dos tiques.

fractional_font:
esta propriedade especifica se os rótulos dos tiques serão exibidos com tamanhos de fonte fracionários. Seu valor pode ser `"on"` ou `"off"`. Se `"on"` , o valor em ponto flutuante de `font_size` é usado para exibição e a fonte não possui outro nome. Se `"off"` apenas a parte inteira é utilizada e a fonte não é suavizada.

isoview:
esta propriedade é utilizada para se obter escalas isométricas para os eixos x, y e z (por exemplo, para fazer com que a exibição da curva sin(x) versus cos(x) seja um círculo, não uma elipse). Os valores podem ser `"on"` ou `"off"`. Se o valor for `"on"`, os `data_bounds`

dos eixos mudam automaticamente de acordo com os valores da propriedade de figura correspondente `figure_size`.

cube_scaling:
esta propriedade é utilizada no modo `3d mode` para se obter uma nova escala para os eixos x, y e z. De fato, permite que os dados se encaixem em um cubo 1x1x1; o objetivo é uma melhor visualização de gráficos 3d caso as escalas dos eixos se diferenciem muito umas das outras. Os valores podem ser `"on"` ou `"off"` (que é o valor padrão). Na maior parte dos casos, ajuda a gerar uma visualização 3d semelhante a do Matlab.

view:
este campo é relacionado ao universo gráfico. Toma `"3d"` como valor correspondente a visualizações em três dimensões. Em outros casos, pode ser `"2d"` para esboço inicial 2d (valor padrão). Este flag também depende dos esboços com os quais o usuário entra: um comando `plot3d` por exemplo, vai alternar o flag `view` de `"2d"` para `"3d"`.

rotation_angles:
este campo é o vetor `[alpha,theta]`. Estes dois valores fornecem as coordenadas esféricas de obsevação (em graus).

log_flags:
string de três caracteres que ajusta a escala (linear ou logarítmica) o longo dos eixos. Cada caractere especifica a escala para os eixos x, y e z respectivamente. Os valores podem ser `'n'` para escala linear ou `'l'` para escala logarítmica.

tight_limits:
se está propriedade for `"on"` os eixos se adaptam para se encaixarem dentro dos valores máximos e mínimos de `data_bounds`. Se o campo for `"off"`, os eixos podem aumentar os limites de modo a produzir rótulos de tiques convenientes.

data_bounds:
este campo contém os valores de limite para as coordenadas x,y e z. É a matriz `[xmin,ymin,zmin;xmax,ymax,zmax]` ou `[xmin,ymin;xmax,ymax]`. Note que, para ter estritamente os limites de dados especificados, `tight_limits` deve ser ajustado para `"on"` (ver acima).

zoom_box:
este campo contém a caixa de zoom corrente, se forem fornecidas quaisquer coordenadas. É uma matriz vazia (sem zoom) ou vetor `[xmin,ymin,xmax,ymax,zmin,zmax]` (define um retângulo dado por dois vértices opostos ).

margins:
um vetor `[margin_left,margin_right,margin_top,margin_bottom]` especificando a porção das margens para estes eixos. Este vetor é composto de números em [0 1] com valores padrões: [0.125 0.125 0.125 0.125] (estes números são razões relativas aos valores correspondentes da propriedade de figura `figure_size`).

axes_bounds:
um vetor `[x_left,y_up,width,height]` especificando a porção da figura usada por estes eixos, onde `x_left`, `y_up`, `width` e `height` são números em [0 1] fornecendo respectivamente a posição do canto superior esquerdo e a dimensão dos eixos (testes números são razões relativas aos valores correspondentes da propriedade de figura `figure_size`).

hidden_axis_color:
esta propriedade define a cor do eixo escondido. É um índice relativo ao mapa de cores corrente.

user_data:
este campo pode armazenar qualquer variável Scilab na estrutura de dados de eixos e recuperá-la.

Propriedades para funções de alto-nível

As funções `plot`, `plot2dx`, `grayplot` e `matplot` utilizam as propriedades seguintes para decidir como misturar esboços consecutivos se isto não for declarado pelo argumento de chamada `frameflag` calling argument. O resultado da mistura é decidido através destas duas propriedades:

auto_clear:
: se esta propriedade for `"on"`, uma chamada a um gráfico de alto-nível reinicializa os eixos correntes e apaga todos os seus galhos antes de realizar o esboço. Se for `"off"` os esboços serão adicionados aos eixos correntes de acordo com a propriedade `"auto_scale"`.

auto_scale:
: uma propriedade para atualizar os limites de dados dos eixos. Se o valor for `"on"`, um novo esboço adaptará as propriedades de eixos correntes para se adequar aos esboços anteriores e correntes. Se o valor for `"off"` o novo esboço será esboçado nos limites de dados correntes.

Valores padrões dos galhos:

hiddencolor:
: esta propriedade controla as cores das partes escondidas. Toma como valor um índice relativo ao mapa de cores corrente. Em outro caso, se for um valor negativo, as partes escondidas tomam a mesma cor que a superfície.

line_mode:
: este campo contém o valor padrão da propriedade line_mode para objetos Segs, Rectangle, Legend, Axis, Plot3d, Fac3d, e Polyline. Seus valores podem ser `"on"` (padrão) ou `"off"`.

line_style:
: este campo contém o valor padrão da propriedade `line_style` para objetos Segs, Arcs, Rectangle e Polyline. `line_style` seleciona o tipo de linha que será utilizado nos desenhos de linhas. Seu valor deve ser um inteiro no intervalo [0 6]. 0 e 1 significam linhas sólidas, os outros selecionam linhas tracejadas (ver getlinestyle).

thickness:
: este campo contém o valor padrão da propriedade `thickness` para todos os objetos que utilizam linhas. Seu valor deve ser um inteiro positivo.

mark_mode:
: este campo contém o valor padrão da propriedade `mark_mode` para os objetos Segs, Rectangle, Legend, Axis, Plot3d, Fac3d e Polyline. Seu valor pode ser `"on"` ou `"off"` (padrão).

mark_style:
: este campo contém o valor padrão da propriedade `mark_style` para objetos Segs, Rectangle, Legend, Axis, Plot3d, Fac3d e Polyline. `mark_style` seleciona o tipo de marca a ser exibida. Seu valor deve ser um inteiro no intervalo [0 9] que significa: ponto, sinal de mais, cruz, estrela, diamante preenchido, diamante, triângulo para cima, triângulo para baixo, trevo ou círculo.

mark_size_unit:
: este campo contém o valor padrão da propriedade `mark_size_unit` para objetos Segs, Rectangle, Legend, Axis, Plot3d, Fac3d e Polyline. Se `mark_size_unit` é ajustado para `"point"`, então o valor de `mark_size` é diretamente fornecido em pontos. Quando `mark_size_unit` é ajustado para `"tabulated"`, `mark_size` é computado de acordo com o array de tamanho de fonte: logo seu valor deve ser um inteiro no intervalo [0 5] que significa 8pt, 10pt, 12pt, 14pt, 18pt e 24pt. Note que plot2d e funções puras do Scilab utilizam o modo `tabulated` como padrão; ao utilizar-se a função plot o modo `point` é automaticamente habilitado.

mark_size:
: este campo contém o valor padrão da propriedade `mark_size` para objetos Segs, Rectangle, Legend, Axis, Plot3d, Fac3d, e Polyline. `mark_size` seleciona o tamanho de fonte da marca

ser exibida. Deve ser um valor inteiro entre [0 5] que significa 8pt, 10pt, 12pt, 14pt, 18pt e 24pt (ver getmark).

mark_foreground:
este campo contém o valor padrão da propriedade `mark_foreground` criados sob os eixos. Objetos Polyline, Rectangle, Legend, Surface, Segment e Axis utilizam esta propriedade para especificar uma cor de primeiro plano (borda) para suas marcas. O valor deve ser um índice de cor relativo ao mapa de cores corrente. Note que o valor padrão é `-1` (preto padrão) e, mesmo que você mude o `color_map`, este valor `-1` sempre apontará para a cor preta padrão.

mark_background:
este campo contém o valor padrão da propriedade `mark_background` para todos os objetos criados sob os eixos. objetos Polyline, Rectangle, Legend, Surface, Segment e Axis utilizam esta propriedade para especificar uma cor de plano de fundo (face) para suas marcas. O valor deve ser um índice de cor relativo ao mapa de cores corrente. Note que o valor padrão é `-2` (branco padrão) e, mesmo que você mude o `color_map`, este valor `-2` sempre apontará para a cor branca padrão.

foreground:
este campo contém o valor padrão da propriedade `foreground` para eixos e para todos os objetos criados sob os eixos. O valor deve ser um índice de cor relativo ao mapa de cores corrente. Note que o valor padrão é `-1` (preto padrão) e, mesmo que você mude o `color_map`, este valor `-1` sempre apontará para a cor preta padrão.

background:
este campo controla o valor padrão da propriedade `background` para os eixos e para objetos criados sob os eixos. O valor deve ser um índice de cor relativo ao mapa de cores corrente. Note que o valor padrão é `-2` branco padrão) e mesmo que você mude o `color_map`, este valor `-2` sempre apontará para a cor branca padrão.

arc_drawing_mode:
este campo controla o valor padrão da propriedade default `arc_drawing_mode` para todos os objetos Arc criados sob este objeto Axes. Seu valor pode ser `"nurbs"` ou `"lines"`.

clip_state:
este campo contém o valor padrão da propriedade `clip_state` para todos os objetos. Seu valor pode ser:

- `"off"` isto indica todos os objetos criados depois que não estão recortados (valor padrão).
- `"clipgrf"` isto indica todos os objetos criados depois que estão recortados fora das fronteiras dos eixos.
- `"on"` isto indica todos os objetos criados depois que estão recortados fora do retângulo através da propriedade clip_box.

clip_box:
este campo contém o valor padrão da propriedade `clip_box` para todos os objetos. Deve ser uma matriz vazia se clip_state for `"off"`. Em outro caso, o recorte é dado por `[x,y,w,h]` (ponto superior esquerdo, largura, altura).

Nota sobre valores padrões:

Todas essas propriedades e campos listados herdam de valores padrões armazenados em um modelo de eixos. Estes valores podem ser visualizados e modificados. Para isto, utilize o comando `get("default_axes")` command : ele retorna o manipulador gráfico do modelo de eixos. Note que nenhuma janela de gráficos é criada por este comando. Note que os próximos eixos criados herdarão deste modelo (ver "exemplo sobre modelo de eixos" abaixo)

# Exemplos

```
lines(0) // desabilitando o paginamento vertical
a=get("current_axes")//obtendo o manipulador dos novos eixos criados
a.axes_visible="on"; // tornando os eixos visíveis
a.font_size=3; //ajustando o tamanho da fonte dos rótulos dos tiques
a.x_location="top"; //ajustando a posição do eixo x
a.data_bounds=[-100,-2,-1;100,2,1]; //ajustando os valores limites para as coor
a.sub_tics=[5,0];
a.labels_font_color=5;
a.grid=[2,2];
a.box="off";

// exemplo com eixos 3d
clf(); //limpando a janela de gráficos
x=0.1:0.1:2*%pi;plot2d(x-.3,sin(x)*7+.2);
a=gca(); // obtendo o manipulador dos eixos correntes
a.grid=[1 -1 -1]; //make x-grid
a.rotation_angles=[70 250]; //girando eixos com ângulos dados
a.grid=[1 6 -1]; //criando y-grid
a.view="2d"; //retornando a vista 2d
a.box="back_half";
a.labels_font_color=5;
a.children.children.thickness=4;
a.children.children.polyline_style=3;
a.view="3d"; //return te the 3d view
a.children.children.thickness=1;
a.children.children.foreground=2;
a.grid=[1 6 3]; //criando z-grid
a.parent.background=4;
a.background=7;
plot2d(cos(x)+1,3*sin(x)-3);
plot2d(cos(x)+7,3*sin(x)+3);
a.children(2).children.polyline_style=2;
a.children(1).children.polyline_style=4;
a.children(1).children.foreground=5;
a.children(2).children.foreground=14;
a.parent.figure_size= [1200,800];
a.box="on";
a.labels_font_size=4;
a.parent.background=8;
a.parent.figure_size= [400,200];
a.rotation_angles=[0 260];
delete(a.children(2));
delete(); // deletando objeto corrente

a = gca();
a.labels_font_size=1;
a.auto_clear= "on";
x=0:0.1:2.5*%pi;plot2d(10*cos(x),sin(x));

a.data_bounds(:,1) = [1;15] ; // ajustando limites positivos para o eixo x
a.log_flags = "lnn" ; // ajustando o eixo X para escala logarítmica
a.log_flags = "nnn" ; // voltando para escala linear

a=gca();
```

```
a.rotation_angles=[45 45];
a.data_bounds=[-20,-3,-2;20 3 ,2];
xrect([-4 0.5 8 1]);
a.auto_clear = "off" ;
a.isoview="on"; // modo de isovisualização
xrect([-2 0.25 4 0.5]);
a.children(1).fill_mode="on";
a.axes_visible="off";
a.children(1).data=[-2 0.25 -1 4 0.5];
a.children(2).data=[-4 0.5 1 8 1];
x=2*%pi*(0:7)/8;
xv=[.2*sin(x);.9*sin(x)];yv=[.2*cos(x);.9*cos(x)];
xsegs(10*xv,yv,1:8)
s=a.children(1);
s.arrow_size=1;
s.segs_color=5;
a.data_bounds //os valores limites para as coordenadas x,y e z
a.view="2d";
a.data_bounds=[-10,-1; 10,1]; // ajustando os valores limites para vistas bidim

// exemplo com eixos modelos
da=gda() // obtendo manipulador dos eixo modelos para ver e editar campos
// título padrão
da.title.text="My Default@Title"
da.title.foreground = 12;
da.title.font_size = 4;
// rótulos x padrões
da.x_label.text="x";
da.x_label.font_style = 8;
da.x_label.font_size = 2;
da.x_label.foreground = 5;
da.x_location = "middle";
// rótulos y padrões
da.y_label.text="y";
da.y_label.font_style = 3;
da.y_label.font_size = 5;
da.y_label.foreground = 3;
da.y_location = "right";
da.thickness = 2;
da.foreground = 7;
// o esboço
x=(0:0.1:2*%pi)';
plot2d(x,[sin(x),sin(2*x),sin(3*x)],style=[1,2,3],rect=[0,-2,2*%pi,2]);
sda() // de volta aos eixos modelos padrões
```

# Ver Também

lines, set, get, gca, gda, gcf, sda, sdf, scf, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

axis_properties — Descrição das propriedades da entidade Axis (eixo)

# Descrição

A entidade Axis é uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros para escala de eixos e aparência.

Propriedades

parent:
esta propriedade contém o manipulador da raiz. A raiz de uma entidade Axis deve ser do tipo `"Axes"` ou `"Compound"`.

visible:
este campo contém o valor da propriedade `visible` para a entidade. Pode ser `"on"` ou `"off"` . Por padrão, o eixo é visível, a propriedade é `"on"`. Se `"off"`, o eixo não é exibido na tela.

tics_direction:
especifica a direção dos tiques desenhados sob os eixos horizontal e vertical. Os possíveis valores para esta propriedade são:

- `"top"`. Neste caso, os tiques são desenhados na parte de cima do eixo horizontal.
- `"bottom"`. Neste caso, os tiques são desenhados na parte de baixo do eixo horizontal.
- `"left"`. Neste caso, os tiques são desenhados à esquerda do eixo vertical.
- `"right"`. Neste caso, os tiques são desenhados à direita do eixo vertical.

Os valores padrões são `"top"` para o eixo horizontal e `"right"` para o eixo vertical.

xtics_coord:
este campo contém as coordenadas x do eixo. É um vetor linha que contém valores crescentes da esquerda para a direita que fornecem as posições dos tiques para um eixo horizontal. Em outro caso, a entidade é um eixo vertical, esta propriedade contém a escala que define a origem x do eixo.

ytics_coord:
este campo contém as coordenadas y do eixo. É um vetor linha que contém valores crescentes do fundo para o topo que fornecem as posições dos tiques para um eixo vertical. Em outro caso, a entidade é um eixo horizontal, esta propriedade contém a escala que define a origem y do eixo.

tics_color:
o valor desta propriedade é o índice da cor utilizada para desenhar as linhas dos eixos e tiques.

tics_segment:
este campo contém um flag que controla a exibição do segmento base do eixo. O valor padrão é `"on"`, senão, sem exibição, a propriedade toma `"off"` como valor.

tics_style:
esta propriedade descreve como os tiques são dados. É um flag string que pode ter um desses valores:

- `"v"`. É o valor padrão. Neste caso, as posições dos tiques são dadas pelo vetor linha `xtics_coord` para o eixo horizontal (`ytics_coord` para o vertical).
- `"r"`. INeste caso, as posições dos tiques são dadas pelo vetor `[min,max,n]` onde n é o número de intervalos.

- `"i"`. Neste caso, o vetor fornecendo as posições dos tiques tem tamanho 4, `[k1,k2,a,n]` os valores crescem entre `k1*10^a` e `k2*10^a`, n é o número de intervalos.

sub_tics:
: este campo ajusta o número de tiques a serem desenhados entre dois tiques principais.

tics_labels:
: este campo é uma matriz de strings que contém strings a serem impressos ao longo dos eixos nas posições dos tiques.

labels_font_color:
: esta propriedade determina a cor dos rótulos dos tiques.

labels_font_size:
: um inteiro especificando o tamanho da fonte utilizada para imprimir os rótulos dos tiques. Se a propriedade `fractional_font` for `"off"` Apenas a parte inteira do valor é utilizada. Para mais informações, veja graphics_fonts.

fractional_font:
: esta propriedade especifica se os rótulos dos tiques serão exibidos utilizando-se um tamanho de fonte fracionário. Seu valor deve ser `"on"` ou `"off"`. Se `"on"`, o valor em ponto flutuante de `font_size` é utilizado para a exibição e retira-se o serrilhamento da fonte. Se `"off"`, apenas a parte inteira é utilzada e a fonte não é suavizada.

clip_state:
: este campo contém o valor da propriedade `clip_state` para o eixo. O valor de clip_state pode ser:

- `"off"` significa que o eixo não é recortado.
- `"clipgrf"` significa que o eixo é recortado fora da caixa dos eixos.
- `"on"` significa que o eixo é recortado fora do arco dado pela propriedade clip_box.

clip_box:
: este campo determina a propriedade `clip_box`. Por padrão seu valor é uma matriz vazia se a propriedade clip_state é "off". Em outros casos, o vetor `[x,y,w,h]` (ponto superior esquerdo, largura, altura) define as porções do eixo a ser exibido, contudo o valor da propriedade `clip_state` será alterado.

user_data:
: este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados da entidade axis e recuperá-la.

# Exemplos

```
a=get("current_axes");//obtendo manipulador dos novos eixos criados
a.data_bounds=[-1,-1;10,10];

drawaxis(x=2:7,y=4,dir='u');
a1=a.children(1)
a1.xtics_coord=[1 4 5  8 10];
a1.tics_color=2;
a1.labels_font_size=3;
a1.tics_direction="bottom";
a1.tics_labels= [" Fevereiro" "Maio"  "Junho" "Agosto"  "Outubro"];

drawaxis(x=1.2:1:10,y=5,dir='u',textcolor=13);
```

```
a2=get("hdl")
a2.sub_tics=0;
a2.tics_segment="off";
a2.ytics_coord=4;

drawaxis(x=-1,y=0:1:7,dir='r',fontsize=10,textcolor=5,ticscolor=6,sub_int=10)
a3=get("hdl");
a3.tics_labels= 'B'  +string(0:7);
a3.tics_direction="left";
```

# Ver Também

set, get, delete, drawaxis, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

bar — Histograma de barras

```
bar(y)
bar(x,y)
bar([h],x,y [,width [,color [,style]]])
```

# Parâmetros

h
: manipulador de eixos, (padrão: h=gca() ).

y
: escalar real, vetor de tamanho N ou uma matriz NxM.

x
: escalar real ou um vetor de tamanho N, (padrão: se y é um vetor, então x is é um vetor e o comprimento de x é igual ao comprimento de y. Se y é uma matriz, então x é um vetor e o comprimento de x é igual ao número de linhas de y)

width
: (opcional), escalar real, define a largura (uma porcentagem do espaço disponível) para a barra (padrão: 0.8, i.e 80%).

color
: (opcional), um string (padrão: 'blue'), especificando a cor da barra.

style:
: string, 'grouped' ou 'stacked' (padrão: 'grouped').

# Descrição

`bar(y,...)` : se y é um vetor então a função bar desenha uma poligonal que tem `polyline_style` tipo 6. Se y for um vetor, desenha o vetor y versus o vetor 1:size(y,'*') . Se y for uma matriz N*M, bar desenha M poligonais (tipo 6), cada poligonal corresponde a uma coluna de y versus o vetor x=1:size(y,1).

`bar(x,y,...)` : se y for um vetor, então a função bar desenha uma poligonal que tem `polyline_style` tipo 6, onde comprimento de x = comprimento de y. Se y é uma matriz NxM então a função bar desenha M poligonais que possuem tipo 6. Cada poligonal corresponde a uma coluna de y versus o vetor x.

`bar(h,...)` : define os eixos correntes onde o desenho é feito.

`bar(...,width,...)` : define a largura da(s) barra(s) em porcentagem (geralmente: 0<width<=1).

`bar(...,style,...)` : define como as barras serão desenhadas. Se y é uma matriz NxM (então temos M poligonais de tipo 6), então há dois modos de se desenhar as M barras. A opção style= 'grouped' permite centrar as M poligonais versus cada componente de x, e a opção style = 'stacked' permite empilhá-las.

`bar(...,color,...)` : define a cor da barra. Funções de barra utilizam o mesmo mapa de cores que na função

Se há várias chamadas a bar, a função `barhomogenize` permite homogeneizar a largura e estilo de todas as barras (i.e poligonais do tipo 6) inclusas nos eixos correntes.

# Exemplos

```
// primeiro exemplo: desenhando uma barra (i.e uma poligonal com polyline_style
// width=0.5 e color='yellow' e style='grouped', x=1:length(y).
scf(0);
y=[1 -3 5];
bar(y,0.5,'yellow');

// segundo exemplo: desenhando 3 barras (i.e três poligonais com polyline_style
scf(1);
x=[1 2 5]; y=[1  -5 6;3 -2 7;4  -3 8];
bar(x,y);

// terceiro exemplo: style='stacked'.
scf(2);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6  9];
bar(x,y,'stacked');

// quarto exemplo: width=0.2;color='green', style='grouped'
scf(3);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6 9];
bar(x,y,0.2,'green');
```

# Ver Também

barh, barhomogenize, plot, polyline_properties

# Autor

Farid Belahcene

# Name

barh — Exibição horizontal de um histograma de barras

```
barh(y)
barh(x,y)
barh([h],x,y [,width [,color [,style]]])
```

# Parâmetros

h
: manipulador de eixos, (padrão: h=gca() ).

y
: escalar real, vetor de tamanho N ou uma matriz NxM.

x
: escalar real ou um vetor de tamanho N, (padrão: se y é um vetor, então x é um vetor e o comprimento de x é igual ao comprimento de y. Se y é uma matriz, então x é um vetor e o comprimento de x é igual ao número de linhas de y)

width
: (opcional), escalar real, define a largura (uma porcentagem do espaço disponível) para a barra (padrão: 0.8, i.e. 80%).

color
: (opcional), um string (padrão: 'blue'), especificando a cor interior da barra barra.

style:
: string, 'grouped' ou 'stacked' (padrão: 'grouped').

# Descrição

`barh(y,...)` : se y é um vetor então a função bar desenha uma poligonal que tem `polyline_style` tipo 6. Se y for um vetor, desenha o vetor y versus o vetor 1:size(y,'*'). Se y for uma matriz N*M, bar desenha M poligonais (tipo 6), cada poligonal corresponde a uma coluna de y versus o vetor x=1:size(y,1)

`barh(x,y,...)` : se y for um vetor então a função desenha uma poligonal que tem `polyline_style` tipo 6, onde comprimento de x = comprimento de y. Se y é uma matriz NxM então a função desenha M poligonais que possuem tipo 6. Cada poligonal corresponde a uma coluna de y versus o vetor x.

`barh(h,...)` : define os eixos correntes onde o desenho é realizado.

`barh(...,width,...)` : define a largura da(s) barra(s) em porcentagem (geralmente: 0<width<1).

`barh(...,style,...)` : define como as barras serão desenhadas. Se y é uma matriz NxM (então temos M poligonais de tipo 6) então há dois modos de se desenhar as M barras. A opção style= 'grouped' permite centrar as M poligonais versus cada componente de x, e a opção style = 'stacked' permite empilhá-las.

`barh(...,color,...)` : define a cor da barra. Funções de barra utilizam o mesmo mapa de cores que na função `plot`.

Se há várias chamadas a barh, a função `barhomogenize` permite homogeneizar a largura e estilo de todas as barras (i.e poligonais do tipo 6) inclusas nos eixos correntes.

# Exemplos

```
// primeiro exemplo: desenhando uma barra (i.e uma poligonal com polyline_style
scf(0);
y=[1 -3 5];
barh(y,0.5,'yellow');

// segundo exemplo: desenhando três barras (i.e três poligonais com polyline_sty
scf(1);
x=[1 2 5]; y=[1  -5 6;3 -2 7;4  -3 8];
barh(x,y);

// terceiro exemplo: style='stacked'.
scf(2);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6  9];
barh(x,y,'stacked');

// quarto exemplo: width=0.2;color='green'; style='grouped'
scf(3);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6 9];
barh(x,y,0.2,'green');
```

# Ver Também

bar, barhomogenize, plot, polyline_properties

# Autor

Farid Belahcene

# Name

barhomogenize — Homogeniza todas as barras inclusas nos eixos correntes

```
barhomogenize()
barhomogenize([h[,'style'[,'width']]])
```

# Parâmetros

h
: manipulador de eixos, (padrão: h=gca() ).

style
: string, 'grouped' ou 'stacked' (padrão: 'grouped').

width
: (opcional) escalar real, define a largura (uma porcentagem do espaço disponível) para a barra (padrão: 0.8).

# Descrição

Se há várias chamadas a bar, a função `barhomogenize` permite homogeneizar a largura e estilo de todas as barras (i.e que têm `polyline_style` tipo 6) inclusas nos eixos correntes. Estas barras devem ter os mesmos dados x.

`barhomogenize( )` : toma os valores padrões, h=gca(), width=0.8, style='grouped'.

`barhomogenize(h,...)` : define os eixos correntes onde o desenho é realizado.

`barhomogenize(...,width,...)` : define a largura da(s) barra(s) em porcentagem(geralmente: 0<width<=1).

`barhomogenize(...,style,...)` : define como as barras são desenhadas. A opção 'grouped' permite centrar as M poligonais versus cada componente de x, e a opção 'stacked' permite empilhá-las.

# Exemplos

```
// Primeiro exemplo: criação de uma barra amarela (i.e uma poligonal com polyli
subplot(2,3,1)
xtitle('ex1: criação de uma barra amarela e três barras')
x=1:3; y1=1:3; y2=[4 3 5;6 7 8;9 10 11];
bar(x,y1,'yellow');bar(x,y2);
// homogeneização dessas quatro barras
subplot(2,3,2)
xtitle('homogenização agrupada')
x=1:3; y1=1:3; y2=[4 3 5;6 7 8;9 10 11];
bar(x,y1,'yellow');bar(x,y2);
barhomogenize();
// homogenização empilhada dessas quatro barras
subplot(2,3,3)
xtitle('homogenização empilhada')
x=1:3; y1=1:3; y2=[4 3 5;6 7 8;9 10 11];
bar(x,y1,'yellow');bar(x,y2);
barhomogenize('stacked',1);

// Segundo exemplo: criação de uma barra vermelha (i.e uma poligonal com polyli
subplot(2,3,4)
```

```
xtitle('ex2: criação de uma barra e duas poligonais')
x=1:10; y=sin(x)/2;
bar(x,y,'red')
x1=1:10;y1=[sin(x);cos(x)]
plot(x1,y1)
// modificando o polyline_style da segunda poligonal do esboço (esta poligonal
subplot(2,3,5)
xtitle('transformação da segunda poligonal em uma barra')
x=1:10; y=sin(x)/2;
bar(x,y,'red')
x1=1:10;y1=[sin(x);cos(x)]
plot(x1,y1)
e=gce(); e2=e.children(2); e2.polyline_style=6;
// homogenização da primeira barra (da função bar) e da segunda barra (da modif
subplot(2,3,6)
xtitle('homogeneização em grupo')

x=1:10; y=sin(x)/2;
bar(x,y,'red')
x1=1:10;y1=[sin(x);cos(x)]
plot(x1,y1)
e=gce(); e2=e.children(2); e2.polyline_style=6;
barhomogenize();
// mudança de estilo e largura
//barhomogenize('stacked',0.5);
//barhomogenize('stacked',1);
```

# Ver Também

bar, polyline_properties

# Autor

Farid Belacehne

# Name

bonecolormap — Mapa de cores cinza com um tom claro de azul

```
cmap=bonecolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`bonecolormap` computa um mapa de cores cinza com um tom azul claro

# Exemplos

```
f = scf();
plot3d1();
f.color_map = bonecolormap(32);
```

# Ver Também

colormap, autumncolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap

# Name

captions — Impressão de legendas de gráficos

```
hl=captions(h, strings [,location])
```

# Parâmetros

h
: vetor de manipuladores de entidades Polyline

strings
: n-vetor de strings, strings(i) é a legenda da i-ésima poligonal

hl
: um manipulador do tipo "`Legend`", pontos para a estrutura contendo todas as informações das legendas (ver legend_properties).

location
: string com possíveis valores

- `"in_upper_right"` : as legendas são exibidas no canto superior direito da caixa de eixos
- `"in_upper_left"`: as legendas são exibidas no canto superior esquerdo da caixa de eixos
- `"in_lower_right"`: as legendas são exibidas no canto inferior direito da caixa de eixos
- `"in_lower_left"`: as legendas são exibidas no canto inferior esquerdo da caixa de eixos
- `"out_upper_right"`: as legendas são exibidas à direita do canto superior direito da caixa de eixos
- `"out_upper_left"`: as legendas são exibidas à esquerda do canto superior esquerdo da caixa de eixos
- `"out_lower_right"`: as legendas são exibidas à direita do canto inferior direito da caixa de eixos
- `"out_lower_left"`: as legendas são exibidas à esquerda do canto inferior esquerdo da caixa de eixos
- `"upper_caption"`: as legendas são exibidas acima do canto superior esquerdo da caixa de eixos
- `"lower_caption"`: as legendas são exibidas abaixo do canto inferior esquerdo da caixa de eixos. Esta opção corresponde ao argumento `leg` de plot2d
- `"by_coordinates"`: o canto superior esquerdo da caixa das legendas é dada pelo campo "position" da estrutura de dados associada. As posições `x` e `y` são fornecidas como frações dos tamanhos de `axes_bounds.`

# Descrição

Coloca legendas no esboço corrente no canto inferior esquerdo da janela gráfica utilizando-se os strings especificados como rótulos. `captions` prefixa rótulos por uma nova chamada às poligonais correspondentes. O tipo e propriedades são recuperados do dado manipulador:

A função `captions` cria uma estrutura de dados Legend.

Há no máximo uma `Legend` associada a cada entidade Axes. Se a função caption for chamada novamente enquanto `Legend` ainda existe, a legenda antiga é apagada.

# Exemplos

```
t=0:0.1:2*%pi;
a=gca();a.data_bounds=[t(1) -1.8;t($) 1.8];
a.margins(4)=0.2;

plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[1,2 3]);
e=gce();
e.children(1).thickness=3;
e.children(2).line_style=4;

hl=captions(e.children,['cos(t)';'cos(2*t)';'cos(3*t)']);
hl=captions(e.children,['cos(t)';'cos(2*t)';'cos(3*t)'],'in_upper_right');

hl.legend_location='in_upper_right'
hl.fill_mode='on';
```

# Ver Também

plot2d, legend, polyline_properties, legend_properties

# Name

champ — esboço de campo vetorial 2d

```
champ(x,y,fx,fy,[arfact,rect,strf])
champ(x,y,fx,fy,<opt_args>)
```

# Parâmetros

x,y
: dois vetores que definem o grid

fx
: uma matriz que descreve o componente x do campo vetorial. `fx(i,j)` é o componente x do campo vetorial no ponto `(x(i),y(j))`.

fy
: uma matriz que descreve o componente y do campo vetorial. `fy(i,j)`é o componente y do campo vetorial no ponto `(x(i),y(j))`.

<opt_args>
: representa a seqüência de declarações `key1=value1,key2=value2`,... onde `key1`, `key2,...` podem ser um dos seguintes: arfact, rect, strf (ver abaixo).

arfact
: um argumento opcional do tipo real que fornece um fator de escala para a exibição das cabeças das setas no esboço (o valor padrão é 1.0).

rect
: um vetor `rect=[xmin,ymin,xmax,ymax]` que fornece as fronteiras da moldura gráfica à ser utilizada.

strf
: um string de comprimento 3 "xyz" que possui o mesmo significado do parâmetro `strf` de `plot2d`. O primeiro caractere não tem efeito com `champ`.

# Descrição

`champ` desenha um campo vetorial 2d. O comprimento das setas é proporcional é proporcional à intensidade do campo.

Se você quiser setas coloridas com a cor das setas dependendo da intensidade do campo, use `champ1`.

Entre com o comando `champ()` para visualizar uma demonstração.

# Exemplos

```
// usando rect como fronteiras do esboço
champ(-5:5,-5:5,rand(11,11),rand(11,11),rect=[-10,-10,10,10],arfact=2)
// usando (x,y) para conseguir obter
clf()
champ(-5:5,-5:5,rand(11,11),rand(11,11),2,[-10,-10,10,10],"021")
```

# Ver Também

champ1, fchamp, plot2d

# Autor

J.Ph.C.

# Name

champ1 — campo vetorial 2d com setas coloridas

```
champ1(x,y,fx,fy,[arfact,rect,strf])
```

# Parâmetros

x,y
: dois vetores que definem o grid

fx
: uma matriz que descreve o componente x do campo vetorial. `fx(i,j)` é o componente x do campo vetorial no ponto `(x(i),y(j))`.

fy
: uma matriz que descreve o componente y do campo vetorial. `fy(i,j)` é o componente y do campo vetorial no ponto `(x(i),y(j))`.

arfact
: um argumento opcional do tipo real que fornece um fator de escala para a exibição das cabeças das setas no esboço (o valor padrão é 1.0).

rect
: um vetor `rect=[xmin,ymin,xmax,ymax]` que fornece as fronteiras da moldura gráfica a ser utilizada.

frameflag
: ccontrola a computação dos intervalos de coordenadas verdadeiros a partir dos valores mínimos requeridos. O valor associado deve ser um inteiro no intervalo de 0 a 8.

axesflag
: especifica como os eixos são desenhados. O valor associado deve ser um inteiro no intervalo de 0 a 5.

strf
: um string de comprimento 3 "xyz" que possui o mesmo significado do parâmetro `strf` de `plot2d`. O primeiro caractere não tem efeito com `champ1`.

# Descrição

`champ1` desenha um campo vetorial 2d com setas coloridas. A cor das setas depende da intensidade do campo.

Se você quiser setas proporcionais à intensidade do campo, use `champ`.

Entre com o comando `champ1()` para visualizar uma demonstração.

# Exemplos

```
champ1(-5:5,-5:5,rand(11,11),rand(11,11),rect=[-10,-10,10,10],arfact=2)
```

# Ver Também

champ, fchamp, plot2d

# Autor

J.Ph.C.

# Name

champ_properties — Descrição das propriedades da entidade campo vetorial 2d

# Descrição

A entidade Champ é uma folha na hierarquia de entidade gráficas. Esta entidade define parâmetros para um campo vetorial 2d .

visible:
: este campo contém o valor da propriedade `visible` para a entidade. Pode ser `"on"` ou `"off"` . Se `"on"` o campo vetorial é desenhado, se `"off"` o campo vetorial não aparece na tela.

data:
: este campo define uma estrutura de dados `tlist` de tipo `"champdata"` composto de uma linha e índices de colunas de cada elemento: as coordenadas do grid x e y contidas respectivamente em `data.x` e `data.y`. Os campos complementares chamados `data.fx` e `data.fy` são matrizes que descrevem respectivamente os componentes `x` e `y` do campo vetorial no ponto `(x(i),y(j))`.

user_data:
: este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados da entidade Champ e recuperá-la.

line_style:
: o valor da propriedade `line_style` deve ser um valor inteiro no intervalo [0 9]. 0 significa linha sólida e os outros valores significam estilos de tracejados. Esta propriedade aplica-se a todas as linhas usadas para desenhar o campo vetorial.

thickness:
: este campo contém o valor da propriedade `thickness` (espessura) para todas as linhas utilizadas no desenho. Seu valor deve ser um inteiro não-negativo.

colored:
: se o valor desta propriedade for `"on"`, os vetores são coloridos utilizando-se uma cor porporcional à intensidade do campo.

clip_state:
: este campo contém o valor da propriedade `clip_state` para a entidade Champ. Pode ser :

- `"off"` significa que o campo vetorial não é recortado
- `"clipgrf"` siginifica que o campo vetorial é recortado fora da caixa da entidade Axes.
- `"on"` significa que o campo vetorial é recortado fora do retângulo dado pela propriedade clip_box.

clip_box:
: este campo contém o valor da propriedade `clip_box`. Seu valor pode ser uma matriz vazia se a propriedade clip_state é "off" .em outros casos, o vetor `[x,y,w,h]` (ponto superior esquerdo, largura, altura) define as porções do campo a serem exibidas, contudo, o valor da propriedade `clip_state` será modificado.

parent:
: esta propriedade contém o manipulador da raiz. A raiz de uma entidade campo vetorial 2d deve ser do tipo `"Axes"` ou `"Compound"`.

# Exemplos

```
a=get("current_axes");//obtendo o manipulador dos novos eixos criados
a.data_bounds=[-10,-10;10,10];
champ(-5:5,-5:5,rand(11,11),rand(11,11))

c=a.children

c.colored="on";
c.thickness=2;
c.data // exibindo um tlist do tipo "scichampdata"
a.data_bounds=[-5,-5;5,5];
```

# Ver Também

set, get, delete, champ, champ1, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

clear_pixmap — Apaga o buffer de pixmap ("mapa de píxeis")

```
clear_pixmap()
```

# Descrição

Se a propriedade de uma janela de gráficos `pixmap` for `"on"` as gravuras são enviadas para uma memória pixmap, ao invés de serem exibidas na tela.

A instrução `clear_pixmap()` apaga o pixmap, mas não a tela.

O modo pixmap pode ser usado para se obter animações suaves.

# Ver Também

figure_properties, show_pixmap

# Autor

Serge Steer INRIA

# Name

clf — Limpa ou reajusta a (janela da) figura gráfica corrente para valores padrões

```
clf(<opt_job_arg>)
clf(h,<opt_job_arg>)
clf(num,<opt_job_arg>)
```

# Parâmetros

h
: o manipulador da figura

num
: identificador da figura (figure_id)

<opt_job_arg>
: um string ( `'clear'` ou `'reset'` ) especificando o trabalho de `clf`.

# Descrição

O comando `clf` é utilizado para reajustar uma figura para seus valores padrões e/ou deletar todos os seus galhos.

Se o valor do string `opt_job_arg` for `'clear'` então todos os galhos da figura especificada são deletados.

Se o valor do string `opt_job_arg` for `'reset'`, então não só todos os galhos da figura são deletados, mas os valores das propriedades da figura são reajustados para seus valores padrões utilizando-se a figura modelo padrão (ver gdf). As únicas exceções são as propriedades `axes_size` e `figure_size` que não podem ser reajustadas se a figura está "aportada" com outros elementos.

`clf(num)` limpa, ou reajusta e limpa, a figura com `figure_id==num`.

`clf(h)` limpa, ou reajusta e limpa, a figura apontada pelo manipulador `h`.

`clf()` limpa, ou reajusta e limpa, a figura existente corrente.

`clf` deleta todos os galhos de janelas especificadas, incluindo menus e uicontrols adicionados pelo usuário. Para evitar que menus e uicontrols sejam deletados, o comando `delete(gca())` deve ser utilizado ao invés.

# Exemplos

```
f4=scf(4);  //criando figura com id==4 e tornando-a a figura corrente
f4.color_map = jetcolormap(64);
f4.figure_size = [400, 200];
plot2d()
clf(f4,'reset')

f0=scf(0); //criando figura com id==0 e tornando-a a figura corrente
f0.color_map=hotcolormap(128); // mudando o mapa de cores
t=-%pi:0.3:%pi;
plot3d1(t,t,sin(t)'*cos(t));

clf() // equivalent to clf(gcf(),'clear')
plot3d1(t,t,sin(t)'*cos(t));  // color_map inalterado
```

```
clf(gcf(),'reset')
plot3d1(t,t,sin(t)'*cos(t));  // color_map alterado (de volta ao padrão com 32 c
```

## See Also

set, get, gcf, gdf, scf, graphics_entities

## Authors

S. Steer & F.Leray INRIA

# Name

color — Retorna o identificador de uma cor

```
id=color(name)
id=color(r,g,b)
```

# Parâmetros

name
: nome de uma cor.

r,g,b
: valores RGB inteiros de uma cor.

id
: identificador da cor.

# Descrição

`color` retorna o identificador de cor correspondente ao seu argumento:

- `name` deve ser um nome de cor conhecido (ver color_list).
- `r`, `g` e `b` devem ser inteiros entre 0 e 255 correspondentes aos componentes de cor vermelho, verde e azul. Como de uso, 0 significa nenhuma intensidade e 255 significa total idensidade para a cor.

Se a cor requisitada não existe no mapa de cores corrente, então ela é adicionada ao mapa de cores.

Esta função pode ser utilizada para se especificar as cores de plano de fundo e de primeiro plano em esboços.

# Exemplos

```
x=linspace(-2*%pi,2*%pi,100)';
// utilizando cores existentes
plot2d(x,[sin(x),cos(x)],style=[color("red"),color("green")]);
// novas cores adicionadas ao mapa de core
e=gce(); p1=e.children(1); p2=e.children(2);
p1.foreground=color("purple"); p2.foreground=color("navy blue");
// utilizando valores RGB
p1.foreground=color(255,128,128);
```

# Ver Também

colormap, color_list, getcolor

# Name

color_list — Lista dos nomes das cores

# Descrição

Abaixo você encontrará os nomes das cores conhecidas pelo Scilab. Os valores RGB (vermelho, verde, azul) são dados após os nomes. Note que, algumas vezes, as cores possuem mais de um nome:

| | | | |
|---|---|---|---|
| scilab blue4 (azul) | 0 | 0 | 144 |
| scilabblue4 | 0 | 0 | 144 |
| scilab blue3 | 0 | 0 | 176 |
| scilabblue3 | 0 | 0 | 176 |
| scilab blue2 | 0 | 0 | 208 |
| scilabblue2 | 0 | 0 | 208 |
| scilab green4 (verde) | 0 | 144 | 0 |
| scilabgreen4 | 0 | 144 | 0 |
| scilab green3 | 0 | 176 | 0 |
| scilabgreen3 | 0 | 176 | 0 |
| scilab green2 | 0 | 208 | 0 |
| scilabgreen2 | 0 | 208 | 0 |
| scilab cyan4 (ciano) | 0 | 144 | 144 |
| scilabcyan4 | 0 | 144 | 144 |
| scilab cyan3 | 0 | 176 | 176 |
| scilabcyan3 | 0 | 176 | 176 |
| scilab cyan2 | 0 | 208 | 208 |
| scilabcyan2 | 0 | 208 | 208 |
| scilab red4 (vermelho) | 144 | 0 | 0 |
| scilabred4 | 144 | 0 | 0 |
| scilab red3 | 176 | 0 | 0 |
| scilabred3 | 176 | 0 | 0 |
| scilab red2 | 208 | 0 | 0 |
| scilabred2 | 208 | 0 | 0 |
| scilab magenta4 (magenta) | 144 | 0 | 144 |
| scilabmagenta4 | 144 | 0 | 144 |
| scilab magenta3 | 176 | 0 | 176 |
| scilabmagenta3 | 176 | 0 | 176 |
| scilab magenta2 | 208 | 0 | 208 |
| scilabmagenta2 | 208 | 0 | 208 |
| scilab brown4 (marrom) | 128 | 48 | 0 |
| scilabbrown4 | 128 | 48 | 0 |
| scilab brown3 | 160 | 64 | 0 |
| scilabbrown3 | 160 | 64 | 0 |
| scilab brown2 | 192 | 96 | 0 |

| | | | |
|---|---|---|---|
| scilabbrown2 | 192 | 96 | 0 |
| scilab pink4 (rosa) | 255 | 128 | 128 |
| scilabpink4 | 255 | 128 | 128 |
| scilab pink3 | 255 | 160 | 160 |
| scilabpink3 | 255 | 160 | 160 |
| scilab pink2 | 255 | 192 | 192 |
| scilabpink2 | 255 | 192 | 192 |
| scilab pink | 255 | 224 | 224 |
| scilabpink | 255 | 224 | 224 |

| | | | |
|---|---|---|---|
| snow (branco) | 255 | 250 | 250 |
| ghost white (branco) | 248 | 248 | 255 |
| ghostwhite | 248 | 248 | 255 |
| white smoke (branco) | 245 | 245 | 245 |
| whitesmoke | 245 | 245 | 245 |
| gainsboro (cinza) | 220 | 220 | 220 |
| floral white (branco) | 255 | 250 | 240 |
| floralwhite | 255 | 250 | 240 |
| old lace (creme) | 253 | 245 | 230 |
| oldlace | 253 | 245 | 230 |
| linen (creme) | 250 | 240 | 230 |
| antique white (creme) | 250 | 235 | 215 |
| antiquewhite | 250 | 235 | 215 |

| | | | |
|---|---|---|---|
| papaya whip (creme alaranjado) | 255 | 239 | 213 |
| papayawhip | 255 | 239 | 213 |
| blanched almond (creme alaranjado) | 255 | 235 | 205 |
| blanchedalmond | 255 | 235 | 205 |
| bisque (creme alaranja-do) | 255 | 228 | 196 |

| | | | |
|---|---|---|---|
| peach puff (pêssego claro) | 255 | 218 | 185 |
| peachpuff | 255 | 218 | 185 |
| navajo white (pêssego) | 255 | 222 | 173 |
| navajowhite | 255 | 222 | 173 |
| moccasin (pêssego) | 255 | 228 | 181 |
| cornsilk (branco creme) | 255 | 248 | 220 |
| ivory (marfim) | 255 | 255 | 240 |
| lemon chiffon (branco rosa) | 255 | 250 | 205 |
| lemonchiffon | 255 | 250 | 205 |
| seashell (branco-rosa) | 255 | 245 | 238 |

| | | | |
|---|---|---|---|
| honeydew (branco menta) | 240 | 255 | 240 |
| mint cream (creme de menta) | 245 | 255 | 250 |
| mintcream | 245 | 255 | 250 |
| azure (azul-bebê) | 240 | 255 | 255 |
| alice blue (azul-bebê) | 240 | 248 | 255 |
| aliceblue | 240 | 248 | 255 |
| lavender (lavanda) | 230 | 230 | 250 |
| lavender blush (lavanda claro) | 255 | 240 | 245 |
| lavenderblush | 255 | 240 | 245 |

| | | | |
|---|---|---|---|
| misty rose (rosa) | 255 | 228 | 225 |
| mistyrose | 255 | 228 | 225 |
| white (branco) | 255 | 255 | 255 |
| black (preto) | 0 | 0 | 0 |
| dark slate gray (cinza azulado escuro) | 47 | 79 | 79 |
| darkslategray | 47 | 79 | 79 |
| dark slate grey | 47 | 79 | 79 |
| darkslategrey | 47 | 79 | 79 |
| dim gray (cinza escuro) | 105 | 105 | 105 |
| dimgray | 105 | 105 | 105 |
| dim grey | 105 | 105 | 105 |
| dimgrey | 105 | 105 | 105 |
| slate gray (cinza azulado) | 112 | 128 | 144 |
| slategray | 112 | 128 | 144 |
| slate grey | 112 | 128 | 144 |
| slategrey | 112 | 128 | 144 |
| light slate gray (cinza azulado claro) | 119 | 136 | 153 |
| lightslategray | 119 | 136 | 153 |
| light slate grey | 119 | 136 | 153 |
| lightslategrey | 119 | 136 | 153 |
| gray (cinza) | 190 | 190 | 190 |
| grey | 190 | 190 | 190 |
| light grey (cinza claro) | 211 | 211 | 211 |
| lightgrey | 211 | 211 | 211 |
| light gray | 211 | 211 | 211 |
| lightgray | 211 | 211 | 211 |

| | | | |
|---|---|---|---|
| midnight blue (azul meia-noite) | 25 | 25 | 112 |

| | | | |
|---|---|---|---|
| midnightblue | 25 | 25 | 112 |
| navy (azul marinho) | 0 | 0 | 128 |
| navy blue | 0 | 0 | 128 |
| navyblue | 0 | 0 | 128 |
| cornflower blue (azul meio claro) | 100 | 149 | 237 |
| cornflowerblue | 100 | 149 | 237 |
| dark slate blue (púrpura azulado escuro) | 72 | 61 | 139 |
| darkslateblue | 72 | 61 | 139 |
| slate blue (púrpura azulado) | 106 | 90 | 205 |
| slateblue | 106 | 90 | 205 |
| medium slate blue (púrpura azulado médio) | 123 | 104 | 238 |
| mediumslateblue | 123 | 104 | 238 |
| light slate blue (púrpura azulado claro) | 132 | 112 | 255 |
| lightslateblue | 132 | 112 | 255 |
| medium blue (azul médio) | 0 | 0 | 205 |
| mediumblue | 0 | 0 | 205 |
| royal blue (azul real) | 65 | 105 | 225 |
| royalblue | 65 | 105 | 225 |
| blue (azul) | 0 | 0 | 255 |
| dodger blue (azul meio claro) | 30 | 144 | 255 |
| dodgerblue | 30 | 144 | 255 |
| deep sky blue (azul claro) | 0 | 191 | 255 |
| deepskyblue | 0 | 191 | 255 |
| sky blue (azul levemente esverdiado) | 135 | 206 | 235 |
| skyblue | 135 | 206 | 235 |
| light sky blue (azul levemente esverdiado) | 135 | 206 | 250 |
| lightskyblue | 135 | 206 | 250 |
| steel blue (azul metálico) | 70 | 130 | 180 |
| steelblue | 70 | 130 | 180 |
| light steel blue (cinza levemente azulado) | 176 | 196 | 222 |
| lightsteelblue | 176 | 196 | 222 |
| light blue | 173 | 216 | 230 |

| | | | |
|---|---|---|---|
| lightblue | 173 | 216 | 230 |
| powder blue (turquesa clara) | 176 | 224 | 230 |
| powderblue | 176 | 224 | 230 |

| | | | |
|---|---|---|---|
| pale turquoise (turquesa clara) | 175 | 238 | 238 |
| paleturquoise | 175 | 238 | 238 |
| dark turquoise (turquesa escura) | 0 | 206 | 209 |
| darkturquoise | 0 | 206 | 209 |
| medium turquoise (turquesa média) | 72 | 209 | 204 |
| mediumturquoise | 72 | 209 | 204 |
| turquoise (turquesa) | 64 | 224 | 208 |
| cyan (ciano) | 0 | 255 | 255 |
| light cyan (branco ciano) | 224 | 255 | 255 |
| lightcyan | 224 | 255 | 255 |
| cadet blue (azul cinza-esverdeado) | 95 | 158 | 160 |
| cadetblue | 95 | 158 | 160 |
| medium aquamarine (aquamarine média) | 102 | 205 | 170 |
| mediumaquamarine | 102 | 205 | 170 |
| aquamarine (aquamarine) | 127 | 255 | 212 |

| | | | |
|---|---|---|---|
| dark green (verde escuro) | 0 | 100 | 0 |
| darkgreen | 0 | 100 | 0 |
| dark olive green (verde-oliva escuro) | 85 | 107 | 47 |
| darkolivegreen | 85 | 107 | 47 |
| dark sea green (verde-mar claro) | 143 | 188 | 143 |
| darkseagreen | 143 | 188 | 143 |
| sea green (verde-mar) | 46 | 139 | 87 |
| seagreen | 46 | 139 | 87 |
| medium sea green (verde-mar médio) | 60 | 179 | 113 |
| mediumseagreen | 60 | 179 | 113 |
| light sea green (verde-mar claro) | 32 | 178 | 170 |
| lightseagreen | 32 | 178 | 170 |
| pale green (verde pálido) | 152 | 251 | 152 |

| | | | |
|---|---|---|---|
| palegreen | 152 | 251 | 152 |
| spring green (verde primavera) | 0 | 255 | 127 |
| springgreen | 0 | 255 | 127 |
| lawn green (verde grama) | 124 | 252 | 0 |
| lawngreen | 124 | 252 | 0 |
| green (verde) | 0 | 255 | 0 |
| chartreuse (verde claro) | 127 | 255 | 0 |
| medium spring green (verde azulado) | 0 | 250 | 154 |
| mediumspringgreen | 0 | 250 | 154 |
| green yellow (verde amarelado) | 173 | 255 | 47 |
| greenyellow | 173 | 255 | 47 |
| lime green (verde-lodo) | 50 | 205 | 50 |
| limegreen | 50 | 205 | 50 |
| yellow green (amarelo esverdeado) | 154 | 205 | 50 |
| yellowgreen | 154 | 205 | 50 |
| forest green (verde-floresta) | 34 | 139 | 34 |
| forestgreen | 34 | 139 | 34 |
| olive drab (verde-azeitona) | 107 | 142 | 35 |
| olivedrab | 107 | 142 | 35 |

| | | | |
|---|---|---|---|
| dark khaki (cáqui escuro) | 189 | 183 | 107 |
| darkkhaki | 189 | 183 | 107 |
| khaki | 240 | 230 | 140 |
| pale goldenrod (amarelo queimado pálido) | 238 | 232 | 170 |
| palegoldenrod | 238 | 232 | 170 |
| light goldenrod yellow (bege claro) | 250 | 250 | 210 |
| lightgoldenrodyellow | 250 | 250 | 210 |
| light yellow (amarelo claro) | 255 | 255 | 224 |
| lightyellow | 255 | 255 | 224 |
| yellow (amarelo) | 255 | 255 | 0 |
| gold (ouro) | 255 | 215 | 0 |
| light goldenrod (amarelo queimado claro) | 238 | 221 | 130 |
| lightgoldenrod | 238 | 221 | 130 |

| | | | |
|---|---|---|---|
| goldenrod (amarelo queimado) | 218 | 165 | 32 |
| dark goldenrod (amarelo queimado escuro) | 184 | 134 | 11 |
| darkgoldenrod | 184 | 134 | 11 |

| | | | |
|---|---|---|---|
| rosy brown (rosa com tonalidade marrom) | 188 | 143 | 143 |
| rosybrown | 188 | 143 | 143 |
| indian red (vermelho índio) | 205 | 92 | 92 |
| indianred | 205 | 92 | 92 |
| saddle brown (marrom sela) | 139 | 69 | 19 |
| saddlebrown | 139 | 69 | 19 |
| sienna (marrom) | 160 | 82 | 45 |
| peru (marrom) | 205 | 133 | 63 |
| burlywood (marrom claro) | 222 | 184 | 135 |
| beige (bege) | 245 | 245 | 220 |
| wheat (trigo) | 245 | 222 | 179 |
| sandy brown (marrom arenoso) | 244 | 164 | 96 |
| sandybrown | 244 | 164 | 96 |
| tan (marrom bronzeado) | 210 | 180 | 140 |
| chocolate (chocolate) | 210 | 105 | 30 |
| firebrick (vermelho sangue) | 178 | 34 | 34 |
| brown (marrom) | 165 | 42 | 42 |
| dark salmon (salmão escuro) | 233 | 150 | 122 |
| darksalmon | 233 | 150 | 122 |
| salmon (salmão) | 250 | 128 | 114 |
| light salmon (salmão claro) | 255 | 160 | 122 |
| lightsalmon | 255 | 160 | 122 |

| | | | |
|---|---|---|---|
| orange (laranja) | 255 | 165 | 0 |
| dark orange (laranja escuro) | 255 | 140 | 0 |
| darkorange | 255 | 140 | 0 |
| coral (laranja salmon) | 255 | 127 | 80 |
| light coral (salmon rosado) | 240 | 128 | 128 |
| lightcoral | 240 | 128 | 128 |
| tomato (tomate) | 255 | 99 | 71 |

| | | | |
|---|---|---|---|
| orange red (laranja avermelhado) | 255 | 69 | 0 |
| orangered | 255 | 69 | 0 |
| red (vermelho) | 255 | 0 | 0 |
| hot pink (rosa quente) | 255 | 105 | 180 |
| hotpink | 255 | 105 | 180 |
| deep pink (rosa vivo) | 255 | 20 | 147 |
| deeppink | 255 | 20 | 147 |
| pink (rosa claro) | 255 | 192 | 203 |
| light pink (rosa) | 255 | 182 | 193 |
| lightpink | 255 | 182 | 193 |

| | | | |
|---|---|---|---|
| pale violet red (violeta avermelhado claro) | 219 | 112 | 147 |
| palevioletred | 219 | 112 | 147 |
| maroon (vinho rosado) | 176 | 48 | 96 |
| medium violet red (violeta avermelhado médio) | 199 | 21 | 133 |
| mediumvioletred | 199 | 21 | 133 |
| violet red (violeta avermelhado) | 208 | 32 | 144 |
| violetred | 208 | 32 | 144 |
| magenta (magenta) | 255 | 0 | 255 |
| violet (violeta) | 238 | 130 | 238 |
| plum (ameixa) | 221 | 160 | 221 |
| orchid (orquídea) | 218 | 112 | 214 |
| medium orchid (orquídea médio) | 186 | 85 | 211 |
| mediumorchid | 186 | 85 | 211 |
| dark orchid (orquídea escuro) | 153 | 50 | 204 |
| darkorchid | 153 | 50 | 204 |
| dark violet (violeta escuro) | 148 | 0 | 211 |
| darkviolet | 148 | 0 | 211 |
| blue violet (violeta azulado) | 138 | 43 | 226 |
| blueviolet | 138 | 43 | 226 |

| | | | |
|---|---|---|---|
| purple (púrpura) | 160 | 32 | 240 |
| medium purple (púrpura claro) | 147 | 112 | 219 |
| mediumpurple | 147 | 112 | 219 |
| thistle (roxo acinzentado) | 216 | 191 | 216 |

| | | | |
|---|---|---|---|
| (daqui em diante são séries de gradações de cores) snow1 | 255 | 250 | 250 |
| snow2 | 238 | 233 | 233 |
| snow3 | 205 | 201 | 201 |
| snow4 | 139 | 137 | 137 |
| seashell1 | 255 | 245 | 238 |
| seashell2 | 238 | 229 | 222 |
| seashell3 | 205 | 197 | 191 |
| seashell4 | 139 | 134 | 130 |
| antiquewhite1 | 255 | 239 | 219 |
| antiquewhite2 | 238 | 223 | 204 |
| antiquewhite3 | 205 | 192 | 176 |
| antiquewhite4 | 139 | 131 | 120 |

| | | | |
|---|---|---|---|
| bisque1 | 255 | 228 | 196 |
| bisque2 | 238 | 213 | 183 |
| bisque3 | 205 | 183 | 158 |
| bisque4 | 139 | 125 | 107 |
| peachpuff1 | 255 | 218 | 185 |
| peachpuff2 | 238 | 203 | 173 |
| peachpuff3 | 205 | 175 | 149 |
| peachpuff4 | 139 | 119 | 101 |
| navajowhite1 | 255 | 222 | 173 |
| navajowhite2 | 238 | 207 | 161 |
| navajowhite3 | 205 | 179 | 139 |
| navajowhite4 | 139 | 121 | 94 |
| lemonchiffon1 | 255 | 250 | 205 |
| lemonchiffon2 | 238 | 233 | 191 |
| lemonchiffon3 | 205 | 201 | 165 |
| lemonchiffon4 | 139 | 137 | 112 |
| cornsilk1 | 255 | 248 | 220 |
| cornsilk2 | 238 | 232 | 205 |
| cornsilk3 | 205 | 200 | 177 |
| cornsilk4 | 139 | 136 | 120 |
| ivory1 | 255 | 255 | 240 |
| ivory2 | 238 | 238 | 224 |
| ivory3 | 205 | 205 | 193 |
| ivory4 | 139 | 139 | 131 |
| honeydew1 | 240 | 255 | 240 |
| honeydew2 | 224 | 238 | 224 |
| honeydew3 | 193 | 205 | 193 |
| honeydew4 | 131 | 139 | 131 |

| | | | |
|---|---|---|---|
| lavenderblush1 | 255 | 240 | 245 |
| lavenderblush2 | 238 | 224 | 229 |
| lavenderblush3 | 205 | 193 | 197 |
| lavenderblush4 | 139 | 131 | 134 |
| mistyrose1 | 255 | 228 | 225 |
| mistyrose2 | 238 | 213 | 210 |
| mistyrose3 | 205 | 183 | 181 |
| mistyrose4 | 139 | 125 | 123 |

| | | | |
|---|---|---|---|
| azure1 | 240 | 255 | 255 |
| azure2 | 224 | 238 | 238 |
| azure3 | 193 | 205 | 205 |
| azure4 | 131 | 139 | 139 |
| slateblue1 | 131 | 111 | 255 |
| slateblue2 | 122 | 103 | 238 |
| slateblue3 | 105 | 89 | 205 |
| slateblue4 | 71 | 60 | 139 |
| royalblue1 | 72 | 118 | 255 |
| royalblue2 | 67 | 110 | 238 |
| royalblue3 | 58 | 95 | 205 |
| royalblue4 | 39 | 64 | 139 |
| blue1 | 0 | 0 | 255 |
| blue2 | 0 | 0 | 238 |
| blue3 | 0 | 0 | 205 |
| blue4 | 0 | 0 | 139 |
| dodgerblue1 | 30 | 144 | 255 |
| dodgerblue2 | 28 | 134 | 238 |
| dodgerblue3 | 24 | 116 | 205 |
| dodgerblue4 | 16 | 78 | 139 |
| steelblue1 | 99 | 184 | 255 |
| steelblue2 | 92 | 172 | 238 |
| steelblue3 | 79 | 148 | 205 |
| steelblue4 | 54 | 100 | 139 |
| deepskyblue1 | 0 | 191 | 255 |
| deepskyblue2 | 0 | 178 | 238 |
| deepskyblue3 | 0 | 154 | 205 |
| deepskyblue4 | 0 | 104 | 139 |
| skyblue1 | 135 | 206 | 255 |
| skyblue2 | 126 | 192 | 238 |
| skyblue3 | 108 | 166 | 205 |
| skyblue4 | 74 | 112 | 139 |
| lightskyblue1 | 176 | 226 | 255 |

| | | | |
|---|---|---|---|
| lightskyblue2 | 164 | 211 | 238 |
| lightskyblue3 | 141 | 182 | 205 |
| lightskyblue4 | 96 | 123 | 139 |

| | | | |
|---|---|---|---|
| slategray1 | 198 | 226 | 255 |
| slategray2 | 185 | 211 | 238 |
| slategray3 | 159 | 182 | 205 |
| slategray4 | 108 | 123 | 139 |
| lightsteelblue1 | 202 | 225 | 255 |
| lightsteelblue2 | 188 | 210 | 238 |
| lightsteelblue3 | 162 | 181 | 205 |
| lightsteelblue4 | 110 | 123 | 139 |
| lightblue1 | 191 | 239 | 255 |
| lightblue2 | 178 | 223 | 238 |
| lightblue3 | 154 | 192 | 205 |
| lightblue4 | 104 | 131 | 139 |
| lightcyan1 | 224 | 255 | 255 |
| lightcyan2 | 209 | 238 | 238 |
| lightcyan3 | 180 | 205 | 205 |
| lightcyan4 | 122 | 139 | 139 |

| | | | |
|---|---|---|---|
| paleturquoise1 | 187 | 255 | 255 |
| paleturquoise2 | 174 | 238 | 238 |
| paleturquoise3 | 150 | 205 | 205 |
| paleturquoise4 | 102 | 139 | 139 |
| cadetblue1 | 152 | 245 | 255 |
| cadetblue2 | 142 | 229 | 238 |
| cadetblue3 | 122 | 197 | 205 |
| cadetblue4 | 83 | 134 | 139 |
| turquoise1 | 0 | 245 | 255 |
| turquoise2 | 0 | 229 | 238 |
| turquoise3 | 0 | 197 | 205 |
| turquoise4 | 0 | 134 | 139 |
| cyan1 | 0 | 255 | 255 |
| cyan2 | 0 | 238 | 238 |
| cyan3 | 0 | 205 | 205 |
| cyan4 | 0 | 139 | 139 |

| | | | |
|---|---|---|---|
| darkslategray1 | 151 | 255 | 255 |
| darkslategray2 | 141 | 238 | 238 |
| darkslategray3 | 121 | 205 | 205 |
| darkslategray4 | 82 | 139 | 139 |
| aquamarine1 | 127 | 255 | 212 |

| | | | |
|---|---|---|---|
| aquamarine2 | 118 | 238 | 198 |
| aquamarine3 | 102 | 205 | 170 |
| aquamarine4 | 69 | 139 | 116 |
| darkseagreen1 | 193 | 255 | 193 |
| darkseagreen2 | 180 | 238 | 180 |
| darkseagreen3 | 155 | 205 | 155 |
| darkseagreen4 | 105 | 139 | 105 |
| seagreen1 | 84 | 255 | 159 |
| seagreen2 | 78 | 238 | 148 |
| seagreen3 | 67 | 205 | 128 |
| seagreen4 | 46 | 139 | 87 |
| palegreen1 | 154 | 255 | 154 |
| palegreen2 | 144 | 238 | 144 |
| palegreen3 | 124 | 205 | 124 |
| palegreen4 | 84 | 139 | 84 |
| springgreen1 | 0 | 255 | 127 |
| springgreen2 | 0 | 238 | 118 |
| springgreen3 | 0 | 205 | 102 |
| springgreen4 | 0 | 139 | 69 |
| green1 | 0 | 255 | 0 |
| green2 | 0 | 238 | 0 |
| green3 | 0 | 205 | 0 |
| green4 | 0 | 139 | 0 |
| chartreuse1 | 127 | 255 | 0 |
| chartreuse2 | 118 | 238 | 0 |
| chartreuse3 | 102 | 205 | 0 |
| chartreuse4 | 69 | 139 | 0 |
| olivedrab1 | 192 | 255 | 62 |
| olivedrab2 | 179 | 238 | 58 |
| olivedrab3 | 154 | 205 | 50 |
| olivedrab4 | 105 | 139 | 34 |
| darkolivegreen1 | 202 | 255 | 112 |
| darkolivegreen2 | 188 | 238 | 104 |
| darkolivegreen3 | 162 | 205 | 90 |
| darkolivegreen4 | 110 | 139 | 61 |
| khaki1 | 255 | 246 | 143 |
| khaki2 | 238 | 230 | 133 |
| khaki3 | 205 | 198 | 115 |
| khaki4 | 139 | 134 | 78 |

| | | | |
|---|---|---|---|
| lightgoldenrod1 | 255 | 236 | 139 |
| lightgoldenrod2 | 238 | 220 | 130 |

| | | | |
|---|---|---|---|
| lightgoldenrod3 | 205 | 190 | 112 |
| lightgoldenrod4 | 139 | 129 | 76 |
| lightyellow1 | 255 | 255 | 224 |
| lightyellow2 | 238 | 238 | 209 |
| lightyellow3 | 205 | 205 | 180 |
| lightyellow4 | 139 | 139 | 122 |
| yellow1 | 255 | 255 | 0 |
| yellow2 | 238 | 238 | 0 |
| yellow3 | 205 | 205 | 0 |
| yellow4 | 139 | 139 | 0 |
| gold1 | 255 | 215 | 0 |
| gold2 | 238 | 201 | 0 |
| gold3 | 205 | 173 | 0 |
| gold4 | 139 | 117 | 0 |
| goldenrod1 | 255 | 193 | 37 |
| goldenrod2 | 238 | 180 | 34 |
| goldenrod3 | 205 | 155 | 29 |
| goldenrod4 | 139 | 105 | 20 |
| darkgoldenrod1 | 255 | 185 | 15 |
| darkgoldenrod2 | 238 | 173 | 14 |
| darkgoldenrod3 | 205 | 149 | 12 |
| darkgoldenrod4 | 139 | 101 | 8 |

| | | | |
|---|---|---|---|
| rosybrown1 | 255 | 193 | 193 |
| rosybrown2 | 238 | 180 | 180 |
| rosybrown3 | 205 | 155 | 155 |
| rosybrown4 | 139 | 105 | 105 |
| indianred1 | 255 | 106 | 106 |
| indianred2 | 238 | 99 | 99 |
| indianred3 | 205 | 85 | 85 |
| indianred4 | 139 | 58 | 58 |
| sienna1 | 255 | 130 | 71 |
| sienna2 | 238 | 121 | 66 |
| sienna3 | 205 | 104 | 57 |
| sienna4 | 139 | 71 | 38 |
| burlywood1 | 255 | 211 | 155 |
| burlywood2 | 238 | 197 | 145 |
| burlywood3 | 205 | 170 | 125 |
| burlywood4 | 139 | 115 | 85 |

| | | | |
|---|---|---|---|
| wheat1 | 255 | 231 | 186 |
| wheat2 | 238 | 216 | 174 |
| wheat3 | 205 | 186 | 150 |

| | | | |
|---|---|---|---|
| wheat4 | 139 | 126 | 102 |
| tan1 | 255 | 165 | 79 |
| tan2 | 238 | 154 | 73 |
| tan3 | 205 | 133 | 63 |
| tan4 | 139 | 90 | 43 |

| | | | |
|---|---|---|---|
| chocolate1 | 255 | 127 | 36 |
| chocolate2 | 238 | 118 | 33 |
| chocolate3 | 205 | 102 | 29 |
| chocolate4 | 139 | 69 | 19 |
| firebrick1 | 255 | 48 | 48 |
| firebrick2 | 238 | 44 | 44 |
| firebrick3 | 205 | 38 | 38 |
| firebrick4 | 139 | 26 | 26 |
| brown1 | 255 | 64 | 64 |
| brown2 | 238 | 59 | 59 |
| brown3 | 205 | 51 | 51 |
| brown4 | 139 | 35 | 35 |
| salmon1 | 255 | 140 | 105 |
| salmon2 | 238 | 130 | 98 |
| salmon3 | 205 | 112 | 84 |
| salmon4 | 139 | 76 | 57 |
| lightsalmon1 | 255 | 160 | 122 |
| lightsalmon2 | 238 | 149 | 114 |
| lightsalmon3 | 205 | 129 | 98 |
| lightsalmon4 | 139 | 87 | 66 |

| | | | |
|---|---|---|---|
| orange1 | 255 | 165 | 0 |
| orange2 | 238 | 154 | 0 |
| orange3 | 205 | 133 | 0 |
| orange4 | 139 | 90 | 0 |
| darkorange1 | 255 | 127 | 0 |
| darkorange2 | 238 | 118 | 0 |
| darkorange3 | 205 | 102 | 0 |
| darkorange4 | 139 | 69 | 0 |
| coral1 | 255 | 114 | 86 |
| coral2 | 238 | 106 | 80 |
| coral3 | 205 | 91 | 69 |
| coral4 | 139 | 62 | 47 |
| tomato1 | 255 | 99 | 71 |
| tomato2 | 238 | 92 | 66 |
| tomato3 | 205 | 79 | 57 |
| tomato4 | 139 | 54 | 38 |

| | | | |
|---|---|---|---|
| orangered1 | 255 | 69 | 0 |
| orangered2 | 238 | 64 | 0 |
| orangered3 | 205 | 55 | 0 |
| orangered4 | 139 | 37 | 0 |
| red1 | 255 | 0 | 0 |
| red2 | 238 | 0 | 0 |
| red3 | 205 | 0 | 0 |
| red4 | 139 | 0 | 0 |
| deeppink1 | 255 | 20 | 147 |
| deeppink2 | 238 | 18 | 137 |
| deeppink3 | 205 | 16 | 118 |
| deeppink4 | 139 | 10 | 80 |
| hotpink1 | 255 | 110 | 180 |
| hotpink2 | 238 | 106 | 167 |
| hotpink3 | 205 | 96 | 144 |
| hotpink4 | 139 | 58 | 98 |
| pink1 | 255 | 181 | 197 |
| pink2 | 238 | 169 | 184 |
| pink3 | 205 | 145 | 158 |
| pink4 | 139 | 99 | 108 |
| lightpink1 | 255 | 174 | 185 |
| lightpink2 | 238 | 162 | 173 |
| lightpink3 | 205 | 140 | 149 |
| lightpink4 | 139 | 95 | 101 |

| | | | |
|---|---|---|---|
| palevioletred1 | 255 | 130 | 171 |
| palevioletred2 | 238 | 121 | 159 |
| palevioletred3 | 205 | 104 | 137 |
| palevioletred4 | 139 | 71 | 93 |
| maroon1 | 255 | 52 | 179 |
| maroon2 | 238 | 48 | 167 |
| maroon3 | 205 | 41 | 144 |
| maroon4 | 139 | 28 | 98 |
| violetred1 | 255 | 62 | 150 |
| violetred2 | 238 | 58 | 140 |
| violetred3 | 205 | 50 | 120 |
| violetred4 | 139 | 34 | 82 |

| | | | |
|---|---|---|---|
| magenta1 | 255 | 0 | 255 |
| magenta2 | 238 | 0 | 238 |
| magenta3 | 205 | 0 | 205 |
| magenta4 | 139 | 0 | 139 |
| orchid1 | 255 | 131 | 250 |

| | | | |
|---|---|---|---|
| orchid2 | 238 | 122 | 233 |
| orchid3 | 205 | 105 | 201 |
| orchid4 | 139 | 71 | 137 |
| plum1 | 255 | 187 | 255 |
| plum2 | 238 | 174 | 238 |
| plum3 | 205 | 150 | 205 |
| plum4 | 139 | 102 | 139 |
| mediumorchid1 | 224 | 102 | 255 |
| mediumorchid2 | 209 | 95 | 238 |
| mediumorchid3 | 180 | 82 | 205 |
| mediumorchid4 | 122 | 55 | 139 |
| darkorchid1 | 191 | 62 | 255 |
| darkorchid2 | 178 | 58 | 238 |
| darkorchid3 | 154 | 50 | 205 |
| darkorchid4 | 104 | 34 | 139 |

| | | | |
|---|---|---|---|
| purple1 | 155 | 48 | 255 |
| purple2 | 145 | 44 | 238 |
| purple3 | 125 | 38 | 205 |
| purple4 | 85 | 26 | 139 |
| mediumpurple1 | 171 | 130 | 255 |
| mediumpurple2 | 159 | 121 | 238 |
| mediumpurple3 | 137 | 104 | 205 |
| mediumpurple4 | 93 | 71 | 139 |

| | | | |
|---|---|---|---|
| thistle1 | 255 | 225 | 255 |
| thistle2 | 238 | 210 | 238 |
| thistle3 | 205 | 181 | 205 |
| thistle4 | 139 | 123 | 139 |

| | | | |
|---|---|---|---|
| gray0 | 0 | 0 | 0 |
| grey0 | 0 | 0 | 0 |
| gray1 | 3 | 3 | 3 |
| grey1 | 3 | 3 | 3 |
| gray2 | 5 | 5 | 5 |
| grey2 | 5 | 5 | 5 |
| gray3 | 8 | 8 | 8 |
| grey3 | 8 | 8 | 8 |
| gray4 | 10 | 10 | 10 |
| grey4 | 10 | 10 | 10 |
| gray5 | 13 | 13 | 13 |
| grey5 | 13 | 13 | 13 |
| gray6 | 15 | 15 | 15 |
| grey6 | 15 | 15 | 15 |

| | | | |
|---|---|---|---|
| gray7 | 18 | 18 | 18 |
| grey7 | 18 | 18 | 18 |
| gray8 | 20 | 20 | 20 |
| grey8 | 20 | 20 | 20 |
| gray9 | 23 | 23 | 23 |
| grey9 | 23 | 23 | 23 |
| gray10 | 26 | 26 | 26 |
| grey10 | 26 | 26 | 26 |
| gray11 | 28 | 28 | 28 |
| grey11 | 28 | 28 | 28 |
| gray12 | 31 | 31 | 31 |
| grey12 | 31 | 31 | 31 |
| gray13 | 33 | 33 | 33 |
| grey13 | 33 | 33 | 33 |
| gray14 | 36 | 36 | 36 |
| grey14 | 36 | 36 | 36 |
| gray15 | 38 | 38 | 38 |
| grey15 | 38 | 38 | 38 |
| gray16 | 41 | 41 | 41 |
| grey16 | 41 | 41 | 41 |
| gray17 | 43 | 43 | 43 |
| grey17 | 43 | 43 | 43 |
| gray18 | 46 | 46 | 46 |
| grey18 | 46 | 46 | 46 |
| gray19 | 48 | 48 | 48 |
| grey19 | 48 | 48 | 48 |
| gray20 | 51 | 51 | 51 |
| grey20 | 51 | 51 | 51 |
| gray21 | 54 | 54 | 54 |
| grey21 | 54 | 54 | 54 |
| gray22 | 56 | 56 | 56 |
| grey22 | 56 | 56 | 56 |
| gray23 | 59 | 59 | 59 |
| grey23 | 59 | 59 | 59 |
| gray24 | 61 | 61 | 61 |
| grey24 | 61 | 61 | 61 |
| gray25 | 64 | 64 | 64 |
| grey25 | 64 | 64 | 64 |
| gray26 | 66 | 66 | 66 |
| grey26 | 66 | 66 | 66 |
| gray27 | 69 | 69 | 69 |
| grey27 | 69 | 69 | 69 |

| | | | |
|---|---|---|---|
| gray28 | 71 | 71 | 71 |
| grey28 | 71 | 71 | 71 |
| gray29 | 74 | 74 | 74 |
| grey29 | 74 | 74 | 74 |
| gray30 | 77 | 77 | 77 |
| grey30 | 77 | 77 | 77 |
| gray31 | 79 | 79 | 79 |
| grey31 | 79 | 79 | 79 |
| gray32 | 82 | 82 | 82 |
| grey32 | 82 | 82 | 82 |
| gray33 | 84 | 84 | 84 |
| grey33 | 84 | 84 | 84 |
| gray34 | 87 | 87 | 87 |
| grey34 | 87 | 87 | 87 |
| gray35 | 89 | 89 | 89 |
| grey35 | 89 | 89 | 89 |
| gray36 | 92 | 92 | 92 |
| grey36 | 92 | 92 | 92 |
| gray37 | 94 | 94 | 94 |
| grey37 | 94 | 94 | 94 |
| gray38 | 97 | 97 | 97 |

| | | | |
|---|---|---|---|
| grey38 | 97 | 97 | 97 |
| gray39 | 99 | 99 | 99 |
| grey39 | 99 | 99 | 99 |
| gray40 | 102 | 102 | 102 |
| grey40 | 102 | 102 | 102 |
| gray41 | 105 | 105 | 105 |
| grey41 | 105 | 105 | 105 |
| gray42 | 107 | 107 | 107 |
| grey42 | 107 | 107 | 107 |
| gray43 | 110 | 110 | 110 |
| grey43 | 110 | 110 | 110 |
| gray44 | 112 | 112 | 112 |
| grey44 | 112 | 112 | 112 |
| gray45 | 115 | 115 | 115 |
| grey45 | 115 | 115 | 115 |
| gray46 | 117 | 117 | 117 |
| grey46 | 117 | 117 | 117 |
| gray47 | 120 | 120 | 120 |
| grey47 | 120 | 120 | 120 |
| gray48 | 122 | 122 | 122 |

| | | | |
|---|---|---|---|
| grey48 | 122 | 122 | 122 |
| gray49 | 125 | 125 | 125 |
| grey49 | 125 | 125 | 125 |
| gray50 | 127 | 127 | 127 |
| grey50 | 127 | 127 | 127 |
| gray51 | 130 | 130 | 130 |
| grey51 | 130 | 130 | 130 |
| gray52 | 133 | 133 | 133 |
| grey52 | 133 | 133 | 133 |
| gray53 | 135 | 135 | 135 |
| grey53 | 135 | 135 | 135 |
| gray54 | 138 | 138 | 138 |
| grey54 | 138 | 138 | 138 |
| gray55 | 140 | 140 | 140 |
| grey55 | 140 | 140 | 140 |
| gray56 | 143 | 143 | 143 |
| grey56 | 143 | 143 | 143 |
| gray57 | 145 | 145 | 145 |
| grey57 | 145 | 145 | 145 |
| gray58 | 148 | 148 | 148 |
| grey58 | 148 | 148 | 148 |
| gray59 | 150 | 150 | 150 |
| grey59 | 150 | 150 | 150 |
| gray60 | 153 | 153 | 153 |
| grey60 | 153 | 153 | 153 |

| | | | |
|---|---|---|---|
| gray61 | 156 | 156 | 156 |
| grey61 | 156 | 156 | 156 |
| gray62 | 158 | 158 | 158 |
| grey62 | 158 | 158 | 158 |
| gray63 | 161 | 161 | 161 |
| grey63 | 161 | 161 | 161 |
| gray64 | 163 | 163 | 163 |
| grey64 | 163 | 163 | 163 |
| gray65 | 166 | 166 | 166 |
| grey65 | 166 | 166 | 166 |
| gray66 | 168 | 168 | 168 |
| grey66 | 168 | 168 | 168 |
| gray67 | 171 | 171 | 171 |
| grey67 | 171 | 171 | 171 |
| gray68 | 173 | 173 | 173 |
| grey68 | 173 | 173 | 173 |

| | | | |
|---|---|---|---|
| gray69 | 176 | 176 | 176 |
| grey69 | 176 | 176 | 176 |
| gray70 | 179 | 179 | 179 |
| grey70 | 179 | 179 | 179 |
| gray71 | 181 | 181 | 181 |
| grey71 | 181 | 181 | 181 |
| gray72 | 184 | 184 | 184 |
| grey72 | 184 | 184 | 184 |
| gray73 | 186 | 186 | 186 |
| grey73 | 186 | 186 | 186 |
| gray74 | 189 | 189 | 189 |
| grey74 | 189 | 189 | 189 |
| gray75 | 191 | 191 | 191 |
| grey75 | 191 | 191 | 191 |
| gray76 | 194 | 194 | 194 |
| grey76 | 194 | 194 | 194 |
| gray77 | 196 | 196 | 196 |
| grey77 | 196 | 196 | 196 |
| gray78 | 199 | 199 | 199 |
| grey78 | 199 | 199 | 199 |
| gray79 | 201 | 201 | 201 |
| grey79 | 201 | 201 | 201 |
| gray80 | 204 | 204 | 204 |
| grey80 | 204 | 204 | 204 |
| gray81 | 207 | 207 | 207 |
| grey81 | 207 | 207 | 207 |
| gray82 | 209 | 209 | 209 |
| grey82 | 209 | 209 | 209 |
| gray83 | 212 | 212 | 212 |
| grey83 | 212 | 212 | 212 |
| gray84 | 214 | 214 | 214 |
| grey84 | 214 | 214 | 214 |
| gray85 | 217 | 217 | 217 |
| grey85 | 217 | 217 | 217 |
| gray86 | 219 | 219 | 219 |
| grey86 | 219 | 219 | 219 |
| gray87 | 222 | 222 | 222 |
| grey87 | 222 | 222 | 222 |
| gray88 | 224 | 224 | 224 |
| grey88 | 224 | 224 | 224 |
| gray89 | 227 | 227 | 227 |

| | | | |
|---|---|---|---|
| grey89 | 227 | 227 | 227 |
| gray90 | 229 | 229 | 229 |
| grey90 | 229 | 229 | 229 |
| gray91 | 232 | 232 | 232 |
| grey91 | 232 | 232 | 232 |
| gray92 | 235 | 235 | 235 |
| grey92 | 235 | 235 | 235 |
| gray93 | 237 | 237 | 237 |
| grey93 | 237 | 237 | 237 |
| gray94 | 240 | 240 | 240 |
| grey94 | 240 | 240 | 240 |
| gray95 | 242 | 242 | 242 |
| grey95 | 242 | 242 | 242 |
| gray96 | 245 | 245 | 245 |
| grey96 | 245 | 245 | 245 |
| gray97 | 247 | 247 | 247 |
| grey97 | 247 | 247 | 247 |
| gray98 | 250 | 250 | 250 |
| grey98 | 250 | 250 | 250 |
| gray99 | 252 | 252 | 252 |
| grey99 | 252 | 252 | 252 |
| gray100 | 255 | 255 | 255 |
| grey100 | 255 | 255 | 255 |

| | | | |
|---|---|---|---|
| dark grey (cinza escuro) | 169 | 169 | 169 |
| darkgrey | 169 | 169 | 169 |
| dark gray | 169 | 169 | 169 |
| darkgray | 169 | 169 | 169 |
| dark blue (azul escuro) | 0 | 0 | 139 |
| darkblue | 0 | 0 | 139 |
| dark cyan (ciano escuro) | 0 | 139 | 139 |
| darkcyan | 0 | 139 | 139 |
| dark magenta (magenta escuro) | 139 | 0 | 139 |
| darkmagenta | 139 | 0 | 139 |
| dark red (vermelho escuro) | 139 | 0 | 0 |
| darkred | 139 | 0 | 0 |
| light green (verde claro) | 144 | 238 | 144 |
| lightgreen | 144 | 238 | 144 |

# Ver Também

color, name2rgb, rgb2name

# Name

colorbar — Desenha uma barra de cores

```
colorbar(umin, umax [, colminmax, fmt])
```

# Parâmetros

umin
: escalar real, o valor mínimo associado ao esboço

umax
: escalar real, o valor máximo associado ao esboço

colminmax
: (opcional) um vetor com dois componentes inteiros

fmt
: (optional) um string para se ajustar o formato de exbição para as graduações da barra de cores

# Descrição

Desenha uma barra de cores para um plot3d, fec, Sgrayplot, etc... Esta função deve ser chamada **BEFORE** do plot3d, fec, Sgrayplot,... porque ela ajusta e modifica a moldura para o esboço. Desta forma, a barra de cores não é parte do esboço associado e, assim, não é modificada por uma ampliação/diminuição ou rotação.

O argumento opcional `colminmax` pode ser utilizado para precisar a primeira cor (associada a umin) e a última cor (associada a umax) do mapa de cores corrente. Por padrão `colminmax=[1 nb_colors]` onde nb_colors é o número de cores do mapa de cores corrente.

O argumento opcional `fmt` é um string com formato C, como `"%.2f"`, `"%e"`, etc...

Para os dois argumentos opcionais, você pode utilizar a sintaxe `keyword=value` que é útil para fornecer o formato sem fornecer `colminmax` (veja o útilmo exemplo).

# Examples

```
// exemplo 1
x = linspace(0,1,81);
z = cos(2*%pi*x)'*sin(2*%pi*x);
zm = min(z); zM = max(z);
xbasc()
xset("colormap",jetcolormap(64))
colorbar(zm,zM)
Sgrayplot(x,x,z)
xtitle("The function cos(2 pi x)sin(2 pi y)")

// exemplo 2
x = linspace(0,1,81);
z = cos(2*%pi*x)'*sin(2*%pi*x);
zm = min(z); zM = max(z);
zz = abs(0.5*cos(2*%pi*x)'*cos(2*%pi*x));
zzm = min(zz); zzM = max(zz);
xbasc();
xset("colormap",jetcolormap(64))
```

```
drawlater() ;
subplot(2,2,1)
   colorbar(zm,zM)
   Sgrayplot(x,x,z, strf="031", rect=[0 0 1 1])
   xtitle("um Sgrayplot com barra de cores")
subplot(2,2,2)
   colorbar(zm,zM)
   plot3d1(x,x,z)
   xtitle("um plot3d1 com barra de cores")
subplot(2,2,3)
   colorbar(zzm,zzM)
   plot3d1(x,x,zz)
   xtitle("um plot3d1 com barra de cores")
subplot(2,2,4)
   colorbar(zzm,zzM)
   Sgrayplot(x,x,zz, strf="031", rect=[0 0 1 1])
   xtitle("um Sgrayplot com barra de cores")
drawnow() ;

// exemplo 3
x = linspace(0,1,81);
zz = abs(0.5*cos(2*%pi*x)'*cos(2*%pi*x));
zzm = min(zz); zzM = max(zz);
[xf,yf,zf]=genfac3d(x,x,zz);
nb_col = 64;
xbasc()
xset("colormap",hotcolormap(nb_col))
drawlater() ;
colorbar(zzm,zzM,fmt="%.1f")
nbcol = xget("lastpattern")
zcol = dsearch(zf, linspace(zzm, zzM, nb_col+1));
plot3d(xf, yf, list(zf, zcol), flag = [-2 6 4])
xtitle("um plot3d com cores de gradação interpolada")
drawnow() ;
xselect()
```

# See Also

colormap

# Authors

B. Pincon, S. Steer

# Name

colordef — Ajusta os valores de cor padrões para exibição de diferentes esquemas de cores

```
colordef(color_scheme)
colordef(f,color_scheme)
colordef('new',color_scheme)
```

# Parâmetros

color_scheme
: string com valores possíveis: 'white', 'black','none'

f
: manipulador para uma figura gráfica

# Descrição

- `colordef('white')` ajusta o mapa de cores padrão da figura (ver gdf) para `jetcolormap(64)`, a cor de plano de fundo padrão da figura para cinza claro, a cor de plano de fundo dos eixos (ver gda) para branco e as cores de primeiro plano da fonte e das linhas dos eixos para preto.
- `colordef('black')` ajusta o mapa de cores padrão da figura (ver gdf) para `jetcolormap(64)`, a cor de plano de fundo padrão da figura para cinza escuro, a cor de plano de fundo dos eixos (ver gda) para preto e as cores de primeiro plano da fonte e das linhas dos eixos para branco.
- `colordef('none')` ajusta o mapa de cores padrão da figura (ver gdf) para `hsvcolormap(64)`, a cor de plano de fundo padrão da figura para cinza escuro a cor de plano de fundo dos eixos (ver gda) para preto e as cores de primeiro plano da fonte e das linhas dos eixos para branco.
- `colordef(f,color_scheme)` ajusta as propriedades de cor da figura fornecida pelo manipulador `f` tanto quanto as propriedades de cor de seus eixos correntes.
- `colordef('new',color_scheme)` cria uma nova janela de gráficos e ajusta suas propriedades de cor bem como as propriedades de seus eixos.

# Exemplos

```
// Adicionar aqui instruções Scilab e comentários
```

# Ver Também

gdf, gda, figure_properties, axes_properties

# Autor

S. Steer
: INRIA

# Name

colormap — mapa de cores

# Descrição

Um mapa de cores `cmap` é definido por uma matriz m x 3 . m é o número de cores. A cor de número i é dada pela tripla `cmap(i,1)`, `cmap(i,2)` `cmap(i,3)` correspondentes às intensidades de vermelho, verde e azul entre 0 e 1.

De início, 32 cores são definidas no mapa de cores. Você pode modificar o mapa de cores de uma figura através da opção `set(f,"color_map",cmap)` onde `f` é o manipulador da figura.

Cada cor no mapa de cores possui um identificador que você deve utilizar para especificar a cor na maioria das funções de esboço. Para visualizar os identificadores, use a função getcolor.

As funções hotcolormap, jetcolormap e graycolormap fornecem mapas de cores com variação contínua de cores.

Você pode obter o mapa de cores padrão através de `cmap=get(sdf(),"color_map")`.

# Exemplos

```
n=64;
r=linspace(0,1,n)';
g=linspace(1,0,n)';
b=ones(r);
cmap=[r g b];
f=gcf(); f.color_map=cmap;
plot3d1()
f.color_map=get(sdf(),"color_map");
```

# Ver Também

autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap, color, getcolor

# Name

Compound_properties — Descrição das propriedades da entidade compound

# Descrição

A entidade Compound é uma entidade de terceiro nível na hierarquia de entidades gráficas. Esta entidade define interdependências entre várias entidades gráficas e suas propriedades de visibilidade global.

parent:
: esta propriedade contém o manipulador da raiz. A raiz de uma entidade Text deve ser do tipo `"Axes"` ou `"Compound"`.

children:
: um vetor contendo manipuladores para os objetos gráficos galhos do Compound. Estes objetos gráficos podem ser do tipo `"Compound"`, `"Rectangle"`, `"Polyline"`, `"Patch"`, `"Segs"`, `"Arc"`, `"Grayplot"`,..

visible:
: este campo contém o valor da propriedade visible para a entidade . Pode ser `"on"` ou `"off"`. Por padrão, o valor é `"on"` onde as entidades gráficas galhos do Compound são desenhadas. Se `"off"` nenhum dos galhos do Compound é desenhado na tela.

user_data:
: este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados da figura e recuperá-la.

# Ver Também

glue, unglue, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

contour — curvas de nível em uma superfície 3d

```
contour(x,y,z,nz,[theta,alpha,leg,flag,ebox,zlev])
contour(x,y,z,nz,<opt_args>)
```

# Parâmetros

x,y
: vetores de reais de tamanhos n1 e n2.

z
: matriz de reais de tamanho (n1,n2), os valores da função sobre o gride ou uma função Scilab que define uma superfície `z=f(x,y)`.

nz
: os valores de nível ou o número de níveis.

-
: Se `nz` for um inteiro, seu valor fornece o número de níveis igualmente espaçados de zmin a zmax como segue:

```
z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
```

Note que os níveis `zmin` e `zmax` não são desenhados (genericamente eles são reduzidos a pontos) mas podem ser adicionados através de

```
[im,jm] = find(z == zmin);     // ou zmax
plot2d(x(im)',y(jm)',-9,"000")
```

-
: Se `nz` for um vetor, `nz(i)` fornece o valor da i-ésima curva de nível. Note que isto pode ser útil para se visualizar as curvas de nível `zmin` e `zmax` para se adicionar uma tolerância eps: `nz=[zmin+%eps,..,zmax-%eps]`.

<opt_args>
: uma seqüência de declarações `key1=value1, key2=value2`, ... onde keys podem ser `theta`,`alpha`,`leg`,`flag`, `ebox`,`zlev` (ver abaixo). Neste caso, a ordem não possui significado especial.

theta, alpha
: valores reais de dados em graus, as coordenadas esféricas do ponto de observação.

leg
: string definindo os rótulos para cada eixo com @ como um separador de campos, por exemplo "X@Y@Z".

flag
: um vetor real de tamanho três. `flag=[mode,type,box]`.

mode
: string de representação de modo.

mode=0:
: as curvas são desenhadas na superfície definida por (x,y,z).

mode=1:
as curvas são desenhadas em um esboço 3d e sobre o plano definido pela equação z=zlev.

mode=2:
as curvas são desenhadas em um esboço 2D.

type
um inteiro (tipo de escala)

type=0
o esboço é feito utilizando-se a escala 3d corrente (definida por uma chamada anterior a `param3d`, `plot3d`, `contour` ou `plot3d1`).

type=1
re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são especificadas pelo valor do argumento opcional `ebox`.

type=2
re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são computadas utilizando-se dados fornecidos.

type=3
fronteiras 3d isométricas com fronteiras da caixa dadas por `ebox`, de modo semelhante a `type=1`

type=4
fronteiras 3d isométricas derivadas dos dados, de modo semelhante a `type=2`

type=5
fronteiras 3d isométricas expandidas com fronteiras fornecidas por `ebox`, de modo semelhante a `type=1`

type=6
fronteiras 3d isométricas expandidas derivadas dos dados, de modo semelhante a `type=2`

box
um inteiro (moldura ao redor do esboço).

box=0
nada é desenhado ao redor do esboço.

box=1
não implementado (é como box=0).

box=2
apenas os eixos atrás da superfície são desenhados.

box=3
uma caixa cercando a superfície é desenhada e legendas são adicionadas.

box=4
uma caixa cercando a superfície é desenhada e legendas e eixos são adicionados.

ebox
usado quando `type` em `flag` é 1. Especifica as fronteiras do esboço através do vetor `[xmin,xmax,ymin,ymax,zmin,zmax]`.

zlev
número real.

# Descrição

`contour` desenha curvas de nível de uma superfície z= f(x,y). As curvas de nível são desenhadas sobre uma superfície 3d. Os argumentos opcionais são os mesmos que os da função `plot3d` (exceto por `zlev`)e seus significados são os mesmos. Eles controlam o desenho das curva de nível em um esboço 3d. Apenas `flag(1)=mode` tem significado especial.

mode=0
: as curvas são desenhadas na superfície definida por (x,y,z).

mode=1
: as curvas são desenhadas em um esboço 3d e sobre o plano definido pela equação z=zlev.

mode=2
: as curvas são desenhadas em um esboço 2D.

Você pode modificar o formato dos pontos flutuantes impressos sobre os níveis utilizando `xset("fpf",string)` onde `string` fornece o formato em na sintaxe de formato C (por exemplo `string="%.3f"`). Use `string=""` para voltar ao formato padrão e use `string=" "` para suprimir a impressão.

Geralmente é usado `contour2d` para se esboçar curvas de nível em um esboço 2d.

Entre com o comando `contour()` para visualizar uma demonstração.

# Exemplos

```
t=linspace(-%pi,%pi,30);
function z=my_surface(x,y),z=x*sin(x)^2*cos(y),endfunction

contour(t,t,my_surface,10)
// mudando o formato da impressão dos níveis
xset("fpf","%.1f")
xbasc()
contour(t,t,my_surface,10)
// 3D
xbasc()
z=feval(t,t,my_surface);
plot3d(t,t,z);contour(t,t,z+0.2*abs(z),20,flag=[0 2 4]);
```

# Ver Também

contour2d, plot3d

# Autor

J.Ph.C.

# Name

contour2d — curvas de nível de uma superfície em um esboço 2d

```
contour2d(x,y,z,nz,[style,strf,leg,rect,nax])
contour2d(x,y,z,nz,<opt_args>)
```

# Parâmetros

x,y
: vetores de reais de tamanhos n1 e n2: o grid.

z
: matriz de reais de tamanho (n1,n2), os valores da função sobre o grid ou uma função Scilab que define uma superfície `z=f(x,y)`.

nz
: os valores de nível ou o número de níveis.

  Se `nz` for um inteiro, seu valor fornece o número de níveis igualmente espaçados de zmin a zmax como segue:

  ```
  z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
  ```

  Note que os níveis `zmin` e `zmax` não são desenhados (genericamente eles são reduzidos a pontos) mas podem ser adicionados através de

  ```
  [im,jm] = find(z == zmin); // ou zmax
  plot2d(x(im)',y(jm)',-9,"000")
  ```

  Se `nz` for um vetor, `nz(i)` fornece o valor da i-ésima curva de nível.

<opt_args>
: representa uma seqüência de declarações `key1=value1,key2=value2`,... onde `key1`, `key2,...` pode ser um dos seguintes: style, leg, rect, nax, strf ou axesflag e frameflag (ver plot2d)

style,strf,leg,rect,nax
: ver plot2d. O argumento `style` fornece os estilos de tracejado ou cores que serão usadas para as curvas de nível. Deve ter o mesmo tamanho que o número de níveis.

# Descrição

`contour2d` desenha curvas de nível de uma superfície `z=f(x,y)` em um esboço 2d. Os valores de `f(x,y)` são dados pela matriz `z` nos pontos de grid definidos por por `x` e `y`.

Você pode modificar o formato dos pontos flutuantes impressos sobre os níveis utilizando `xset("fpf",string)` onde `string` fornece o formato em na sintaxe de formato C (por exemplo `string="%.3f"`). Use `string=""` para voltar ao formato padrão e use `string=" "` para suprimir a impressão. Este último recurso é útil em conjunção com legends para exibir os números de nível em uma legenda e não diretamente em curvas de nível como de uso (ver Exemplos).

Os argumentos opcionais `style,strf,leg,rect,nax`, podem ser passados através de uma seqüência de declarações `key1=value1, key2=value2`, ... onde podem ser `style`,`strf`,`leg`,`rect`,`nax`. Neste caso, a ordem não tem significado especial.

Use `contour` para esboçar curvas de nível em uma superfície 3d.

# Exemplos

```
contour2d(1:10,1:10,rand(10,10),5,rect=[0,0,11,11])
// mudando o formato de impressão dos níveis
xset("fpf","%.2f")
clf()
contour2d(1:10,1:10,rand(10,10),5,rect=[0,0,11,11])

// agora, um exemplo com números de níveis desenhados em uma legenda
// Cuidado, há alguns truques aqui...
x = linspace(0,4*%pi,80);
z = cos(x')*cos(x);
clf(); f=gcf();
xset("fpf"," ")  // truque 1: isto faz com que alguns números de níveis não seja
                 //          impressos sobre as curvas de nível
f.color_map=jetcolormap(7);
// truque 2: para fazer com que se possa colocar a legenda à direita sem
//          sem interferir com as curvas de nível use rect com um valor xmax
//          grande o suficiente
contour2d(x,x,z,-0.75:0.25:0.75,frameflag=3,rect=[0,0,5*%pi,4*%pi])
// truque 3: use legends (note que a função mais prática legend
//          não funcionará quando um dos níveis for formado por duas curvas)
legends(string(-0.75:0.25:0.75),1:7,"lr");
xtitle("Algumas curvas de nível da função cos(x)cos(y)")
```

# Ver Também

contour, contour2di, plot2d

# Autor

J.Ph.C.

# Name

contour2di — Computa curvas de nível em um esboço 2d

```
[xc,yc]=contour2di(x,y,z,nz)
```

# Parâmetros

x,y
: dois vetores de reais de tamanhos n1 e n2 definindo o grid.

z
: matriz de reais de tamanho (n1,n2), os valores da função.

nz
: os valores de nível ou o número de níveis.

  Se `nz` for um inteiro, seu valor fornece o número de níveis igualmente espaçados de zmin a zmax como segue:

  ```
  z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
  ```

  Note que os níveis `zmin` e `zmax` não são desenhados (genericamente eles são reduzidos a pontos) mas podem ser adicionados através de

  ```
  [im,jm] = find(z == zmin);     // ou zmax
  plot2d(x(im)',y(jm)',-9,"000")
  ```

  Se `nz` for um vetor, `nz(i)` fornece o valor da i-ésima curva de nível.

xc,yc
: vetores de tamanhos idênticos contendo as definições dos contornos. Ver detalhes abaixo.

# Descrição

`contour2di` computa as curvas de nível de uma superfície `z=f(x,y)` em um esboço 2d. Os valores de `f(x,y)` asão dados pela matriz `z` nos pontos de grid definidos por `x` e `y`.

`xc(1)` contém o nível associado ao primeiro caminho de contorno, `yc(1)` contém o número `N1` de pontos definindo este caminho de contorno e (`xc(1+(1:N1))`, `yc(1+(1:N1))` ) contém as coordenadas dos pontos do caminho. O segundo caminho começa em `xc(2+N1)` e `yc(2+N1)` e assim por diante.

# Exemplos

```
[xc,yc]=contour2di(1:10,1:10,rand(10,10),5);
k=1;n=yc(k);c=1;
while k+yc(k)<size(xc,'*')
  n=yc(k);
  plot2d(xc(k+(1:n)),yc(k+(1:n)),c)
  c=c+1;
  k=k+n+1;
end
```

## Ver Também

contour, fcontour, fcontour2d, contour2d, plot2d, xset

## Autor

J.Ph.C.

# Name

contourf — Curvas de nível preenchidas de uma superfície em um esboço 2d

```
contourf(x,y,z,nz,[style,strf,leg,rect,nax])
```

# Parâmetrtos

x,y
: dois vetores de reais de tamanhos n1 e n2 definindo o grid.

z
: matriz de reais de tamanho (n1,n2), os valores da função.

nz
: os valores de nível ou o número de níveis.

- Se `nz` for um inteiro, seu valor fornece o número de níveis igualmente espaçados de zmin a zmax como segue:

  ```
  z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
  ```

  Note que os níveis `zmin` e `zmax` não são desenhados (genericamente eles são reduzidos a pontos) mas podem ser adicionados através de

  ```
  [im,jm] = find(z == zmin);     // ou zmax
  plot2d(x(im)',y(jm)',-9,"000")
  ```

- Se `nz` for um vetor, `nz(i)` fornece o valor da i-ésima curva de nível.

style,strf,leg,rect,nax
: ver `plot2d`. O argumento `style` fornece as cores a serem utilizadas pelas curvas de nível. Deve ter o mesmo tamanho que o número de níveis.

# Descrição

`contourf` pinta a superfície entre duas curvas de nível consecutivas `z=f(x,y)` em um esboço 2d. Os valores de `f(x,y)` são fornecidos pela matriz `z` nos pontos de grid definidos por `x` e `y`.

Você pode mudar o formato dos pontos flutuantes impressos sobre os níveis utilizando `xset("fpf",string)` onde `string` fornece o formato em sintaxe de formato C ( por exemplo `string="%.3f"`). Use `string=""` para voltar ao formato padrão.

Entre com o comando `contourf()` para visualizar uma demonstração.

# Exemplos

```
contourf(1:10,1:10,rand(10,10),5,1:5,"011"," ",[0,0,11,11])

function z=peaks(x,y)
x1=x(:).*.ones(1,size(y,'*'));
```

```
y1=y(:)'.*.ones(size(x,'*'),1);
z =  (3*(1-x1).^2).*exp(-(x1.^2) - (y1+1).^2) ...
   - 10*(x1/5 - x1.^3 - y1.^5).*exp(-x1.^2-y1.^2) ...
   - 1/3*exp(-(x1+1).^2 - y1.^2)
endfunction

function z=peakit()
x=-4:0.1:4;y=x;z=peaks(x,y);
endfunction

z=peakit();

levels=[-6:-1,-logspace(-5,0,10),logspace(-5,0,10),1:8];
m=size(levels,'*');
n = fix(3/8*m);
r = [(1:n)'/n; ones(m-n,1)];
g = [zeros(n,1); (1:n)'/n; ones(m-2*n,1)];
b = [zeros(2*n,1); (1:m-2*n)'/(m-2*n)];
h = [r g b];
xset('colormap',h);
xset('fpf',' ');
xbasc();
contourf([],[],z,[-6:-1,-logspace(-5,0,10),logspace(-5,0,10),1:8],0*ones(1,m))

xset('fpf','');
xbasc();
contourf([],[],z,[-6:-1,-logspace(-5,0,10),logspace(-5,0,10),1:8]);
```

# Ver Também

contour, fcontour, fcontour2d, contour2di, plot2d, xset

# Autor

J.Ph.C.

# Name

coolcolormap — Mapa de cores de ciano a magenta

```
cmap=coolcolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`coolcolormap` computa um mapa de cores com `n` cores variando de ciano a magenta.

# Exemplos

```
f = scf();
plot3d1();
f.color_map = coolcolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap

# Name

coppercolormap — Mapa de cores de preto a um tom claro de cobre

```
cmap=coppercolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`coppercolormap` computa um mapa de cores com `n` cores que variam de preto a um tom claro de cobre.

# Exemplos

```
f = scf();
plot3d1();
f.color_map = coppercolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap

# Name

copy — Copia uma entidade de gráficos

```
copy(h)
copy(h,h_axes)
h_copy=copy(h)
```

# Parâmetros

h
: manipulador da entidade a ser copiada

h_axes
: manipulador da entidade raiz para destino. Deve ser uma entidade Axes (de eixos).

h_copy
: manipulador da entidade copiada.

# Descrição

Esta rotina pode ser usada para copiar uma entidade de gráficos com todos os seus valores de propriedades, retornando o manipulador da nova entidade. Se `h_axes` for omitido, a entidade gráfica é clonada e copiada no mesmo na mesma entidade Axes raiz.

# Exemplos

```
subplot(211);a1=gca();
plot2d()
e=gce();
p1=e.children(1);
p2=copy(p1);p2.data(:,2)=p2.data(:,2)-0.5;
subplot(212);a2=gca();
a2.data_bounds=a1.data_bounds;
copy(p1,a2);
```

# Ver Também

get, set, delete, move, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

delete — Deleta uma entidade gráfica e seus galhos

```
delete(h)
delete(h,"callback")
delete()
delete("all")
```

# Parâmetros

h
: manipulador da entidade gráfica a ser deletada. `h` pode ser um vetor de manipuladores, em qual caso todos os objetos identificados por h(i) serão deletados.

"all"
: palavra-chave string (opcional).

# Descrição

Esta rotina pode ser utilizada para deletar uma entidade gráfica identificada pelo manipulador dado como argumento. Neste caso, todos os galhos desta entidade gráfica serão deletados. Sem qualaquer argumento, `delete` remove a entidade corrente. Com o argumento `"all"` deleta todas as entidades da figura corrente.

O argumento `"callback"` ainda não é manipulado.

# Exemplos

```
subplot(211);
t=1:10;plot2d(t,t.^2),
subplot(223);
plot3d();
subplot(224);
plot2d();
xfrect(1,0,3,1);
a=get("current_axes")
delete(); //deletando o novo objeto gráfico criado
delete(a.children); //deletando todos os galhos dos eixos corentes
delete(a); //deletando os eixos
delete("all"); //deletando todos os objetos gráficos da figura
```

# Ver Também

get, set, copy, move, is_handle_valid, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

dragrect — Arrasta retângulos com o mouse

```
[final_rect,btn]=dragrect(initial_rect)
```

# Parâmetros

initial_rect
: matrtiz 4xn contendo as definições iniciais do retângulo. Cada coluna contém [x_esquerdo; y_topo; largura; altura]. Se apenas um retângulo estiver presente, initial_rect pode ser também um vetor.

final_rect
: um retângulo [x_esquerdo, y_topo, largura, altura]

btn
: um inteiro o número do botão do mouse clicado

# Descrição

`dragrect` arrasta um ou mais retângulos para qualquer lugar da tela. A matriz rect 4xn define os retângulos. Cada coluna de `initial_rect` deve conter a posição inicial do retângulo como valores [x_esquerdo;y_topo;largura;altura]. Quando um botão é clicado, `dragrect` retorna as definições dos retângulos finais em `final_Rect`.

# Exemplos

```
xsetech(frect=[0,0,100,100])
r=dragrect([10;10;30;10])
xrect(r)
```

# Ver Também

xrect, xrects, xclick, xgetmouse

# Name

draw — Desenha uma entidade

```
draw()
draw(h)
```

# Parâmetros

h

a o manipulador de uma entidade a ser desenhada. `h` pode ser um vetor de manipuladores. Neste caso, todos os objetos identificados por h(i) will serão desenhados

# Descrição

Esta função pode ser utilizada para desenhar entidades especificadas por `h` mesmo se a propriedade de figura raiz `immediate_drawing` for `"off"`. Se nenhum argumento for especificado, o objeto corrente é desenhado. Note que a janela não deve ser redimensionada; se for, esses objetos serão escondidos.

# Exemplos

```
subplot(212)
a=gca();
plot2d
drawlater

subplot(211)
plot2d1 // modo de desenho padrão

e=gce();
draw(e.children(2)) // desenhando uma única poligonal dos segundos eixos

e.children(1).visible='off'; // podemos escolher tornar uma linha invisível

draw(e) // desenha o Compound e seus galhos &lt;=&gt; desenha todas as poligona

drawnow // ...agora!

e.children(1).visible='on';
```

# Ver Também

get, set, drawnow, drawlater, graphics_entities

# Autores

Djalel ABDEMOUCHE, F.Leray

# Name

drawaxis — Desenha um eixo

```
drawaxis([options])
// opções:  x,y,dir,sub_int,fontsize,format_n,seg,textcolor,ticscolor,tics
```

# Parâmetros

dir=string
: é utilizado para se especificar a direção dos tiques. `string` can pode ser escolhido entre 'u','r','d','l' e 'l' é o valor padrão. Os valores 'u','r','d','l' correspondem respectivamente a "para cima", "para direita", "para baixo" e "para esquerda".

tics=string
: um flag definindo como os tiques são fornecidos. `string` pode ser escolhido entre 'v','r' e 'i', e , 'v' é o valor padrão.

x,y
: dois vetores que fornecem as posições dos tiques.

val= string matrix
: uma matriz de strings que, quando dada, fornece os strings a serem impressos ao longo dos eixos nas posições dos tiques.

fontsize=int
: especifica o tamanho de fonte a ser utilizado para se exibir os valores ao longo dos eixos. O valor padrão é `-1` que corresponde ao tamanho de fonte atual

format_n=string
: formato a ser utilizado para se exibir os números ao longo dos eixos

seg= 1 ou 0
: um flag que controla a exibição do segmento base do eixo (o valor padrão é 1).

sub_int=integer
: um inteiro que fornece o número de subintervalos entre dois tiques maiores.

textcolor=integer
: especifica a cor a ser utilizada para a exibição de valores ao longo dos eixos. O valor padrão é `-1`, que corresponde a cor corrente.

ticscolor=integer
: especifica a cor a ser utilizada para o desenho dos tiques. O valor padrão é `-1`, que corresponde à cor corrente.

# Descrição

`drawaxis` é utilizado para desenhar um eixo na direção horizontal ou vertical. A direção do eixo é dada por dir. `dir = 'u'` ou `'d'` fornece a direção com os tiques indo para cima (`'u'`) ou para baixo (`'d'`). `dir = 'r'` ou `'l'` fornece um eixo vertical com os tiques indo para a direita (`'r'`) ou esquerda (`'l'`).

`x` e `y` fornecem as posições dos tiques dos eixos. Se o eixo é horizontal, então `y` deve ser um escalar, ou pode ser omitido, e `x` é um vetor Scilab. O significado de `x` é controlado por `tics`.

Se `tics='v'` então `x` fornece as posições dos tiques ao longo do eixo x.

Se `tics='r'` então `x` deve ter tamanho 3. `x=[xmin,xmax,n]` e `n` fornece o número de intervalos.

Se `tics='i'` então `x` deve ser de tamanho 4, `x=[k1,k2,a,n]`. Então `xmin=k1*10^a`, `xmax=k2*10^a` e `n` fornece o número de intervalos.

Se `y` for omitido, então o eixo será posicionado ao topo da moldura se `dir='u'` ou ao fundo se `dir='d'`

Por padrão, os números são impressos ao longo do eixo. Eles são impressos utilizando-se um formato padrão que pode ser alterado através de `format_n`. ITambém é possível exibir strings dados, e não números, isto é feito se `val` é fornecido. O tamanho de `val` deve corresponder ao número de tiques.

# Exemplos

```
plot2d(1:10,1:10,1,"020")
// eixo horizontal
drawaxis(x=2:7,y=4,dir='u',tics='v')
// eixo horizontal na parte superior da moldura
drawaxis(x=2:7,dir='u',tics='v')
// eixo horizontal na parte inferior da moldura
drawaxis(x=2:7,dir='d',tics='v')

// eixo horizontal dado por um intervalo
drawaxis(x=[2,7,3],y=4,dir='d',tics='r')

// eixo vertical
drawaxis(x=4,y=2:7,dir='r',tics='v')
drawaxis(x=2,y=[2,7,3],dir='l',tics='r')
drawaxis(y=2:7,dir='r',tics='v')
drawaxis(y=2:7,dir='l',tics='v')

// eixo horizontal com strings exibidos nas posições dos tiques
drawaxis(x=2:7,y=8,dir='u',tics='v',val='A'+string(1:6));
// eixo vertical com strings exibidos nas posições dos tiques
drawaxis(x=8,y=2:7,dir='r',tics='v',val='B'+string(1:6));

// eixo horizontal dado com um intervalo 'i'.
drawaxis(x=[2,5,0,3],y=9,dir='u',tics='i');
drawaxis(x=9,y=[2,5,0,3],dir='r',tics='i',sub_int=5);

// eixo horizontal de novo
drawaxis(x=2:7,y=4,dir='u',tics='v',fontsize=10,textcolor=9,ticscolor=7,seg=0,s
```

# Autor

J.Ph.C.

# Name

drawlater — Torna galhos dos eixos invisíveis.

```
drawlater()
```

# Descrição

Esta função pode ser utilizada para não se exibir imediatamente na figura corrente os próximos objetos gráficos a serem criados por uma chamada a funções de alto-nível tais como funções de esboço ou ajuste de propriedades de objetos existentes. A propriedade `immediate_drawing` da figura corrente é ajustada para `'off'` para se adiar os próximos desenhos.

Pode ser utilizada especialmente com a função

Para reabilitar `immediate_drawing` para a figura corrente, você pode utilizar a função.

Aviso: note que entre chamadas às funções drawlater e drawnow, a figura corrente pode ser modificada. Logo, estas funções devem ser usadas com cuidado.

# Exemplos

```
//Exemplo 1:  uma entidade Axes / uma entidade Figure
drawlater();
xfarc(.25,.55,.1,.15,0,64*360);
xfarc(.55,.55,.1,.15,0,64*360);
xfrect(.3,.8,.3,.2);
xfrect(.2,.7,.5,.2);
xfrect(.32,.78,.1,.1);
xfrect(.44,.78,.14,.1);
xfrect(-.2,.4,1.5,.8);
xstring(0.33,.9,"A Scilab Car");
a=get("current_axes");
a.children(1).font_size=4;
a.children(1).font_style=4;
a.children(1).background=5;
a.children(3).background=8;
a.children(4).background=8;
a.children(5).background=17;
a.children(6).background=17;
a.children(7).background=25;
a.children(8).background=25;
xclick();drawnow();

//Exemplo 2: duas entidades Axes/ uma entidade Figure

subplot(212)
a=gca();
drawlater // o que estará presente nestes eixos será desenhado depois
plot2d // desenhará estes eixos e seus galhos depois...

subplot(211) // aviso: mudamos os eixos
plot2d1 // modo de desenho padrão

draw(a) // ...os eixos são visíveis apenas se você não resdesenhar a janela
drawnow() // tudo está visível
```

## Ver Também

get, set, drawnow, graphics_entities

## Autor

Djalel ABDEMOUCHE, F.Leray

# Name

drawnow — Desenha entidades gráficas escondidas

```
drawnow()
```

# Descrição

Considerando a figura corrente, esta função pode ser usada para desenhar as entidades gráficas escondidas e todos os seus galhos, que podem ter sido adiados por uma chamada anterior à função `drawlater`. A propriedade `immediate_drawing` da figura corrente é ajustada para `"on"` .

Pode ser usado especialmente com a função `drawlater`.

# Exemplos

```
f=get("current_figure") // manipulador da figura corrente

drawlater()
subplot(221);
t=1:10;plot2d(t,t.^2)
subplot(222);
x=0:1:7;plot2d(x,cos(x),2)
subplot(234);
plot2d(t,cos(t),3);
subplot(235);
plot2d(x,sin(2*x),5);
subplot(236);
plot2d(t,tan(2*t));

draw(gca()); //desenhando os eixos correntes e seus galhos
draw(f.children([3 4])); // desenhando os eixos especificados e seus galhos
drawnow(); // desenhando a figura corrente
```

# Ver Também

get, set, drawlater, graphics_entities

# Autores

Djalel ABDEMOUCHE, F.Leray

# Name

edit_curv — editor interativo de curvas gráficas

```
[x,y,ok,gc] = edit_curv(y)
[x,y,ok,gc] = edit_curv(x,y)
[x,y,ok,gc] = edit_curv(x,y,job)
[x,y,ok,gc] = edit_curv(x,y,job,tit)
[x,y,ok,gc] = edit_curv(x,y,job,tit,gc)
```

# Parâmetros

x
: vetor de coordenadas x

y
: vetor de coordenadas y

job
: um string formado por um a três dos caracteres 'a','x','y'

  'a'
  : adicionar pontos à curva editada

  'x'
  : modificar coordenadas x dos pontos das curvas editadas

  'y'
  : modificar coordenadas y dos pontos das curvas editadas

tit
: um vetor de três strings que fornecem a legenda da curva

gc
: uma lista de parâmetros de janela de gráficos: `gc=list(rect,nax)`

  rect
  : bordas dos gráficos (ver plot2d para detalhes)

  nax
  : parâmetros de graduação (ver plot2d para detalhes)

ok
: indicador. Se ok== %t retorna com menu 'ok' senão, retorna com menu 'abort': lista (objetos gráficos criados sob `edit_curv`

# Descrição

`edit_curv` é um editor de curvas gráficas interativo. Para adicionar um ponto novo, apenas clique na localização desejada, o ponto adicionado será conectado à extremidade mais próxima. Para mover um ponto, clique nele, arraste o mouse para a nova posição e clique para ajustar a nova posição.

# Autor

Serge Steer ; ; ; ; ;

# Name

errbar — adiciona barras de erro veriticais a um esboço 2d

```
errbar(x,y,em,ep)
```

# Parâmetros

x,y,em,ep
: quatro matrizes de mesmo tamanho.

# Descrição

`errbar` adiciona barras de erro verticais a um esboço 2d. `x` e `y` têm o mesmo significado que em `plot2d`. `em(i,j)` e `ep(i,j)` são os intervalos de erro sobre os valores `y(i,j)`: `[y(i,j)-em(i,j),y(i,j)+ep(i,j)]`.

Entre com o comando `errbar()` para visualizar uma demonstração.

# Exemplos

```
t=[0:0.1:2*%pi]';
y=[sin(t) cos(t)]; x=[t t];
plot2d(x,y)
errbar(x,y,0.05*ones(x),0.03*ones(x))
```

# Ver Também

plot2d

# Autor

J.Ph.C.

# Name

eval3d — valores de uma função em um grid

```
[z]=eval3d(fun,x,[y])
```

# Parâmetros

fun
: função que aceita vetores como argumentos

x,y
: 2 vetores de tamanhos (1,n1) e (1,n2). (valor padrão para `y` : `y=x`).

z
: matriz de tamanho (n1,n2).

# Descrição

Esta função retorna uma matriz `z(n1,n2)`. `z(i,j)=fun(x(i),y(j))`. Se a função `fun` não aceitar argumentos do tipo vetor, use a primitiva `feval`.

# Exemplos

```
x=-5:5;y=x;
deff('[z]=f(x,y)',['z= x.*y']);
z=eval3d(f,x,y);
plot3d(x,y,z);

deff('[z]=f(x,y)',['z= x*y']);
z=feval(x,y,f);
plot3d(x,y,z);
```

# Ver Também

feval

# Autor

Steer S.

# Name

eval3dp — computa facetas 3d de uma superfície parametrizada

```
[Xf,Yf,Zf]=eval3dp(fun,p1,p2)
```

# Parâmetros

Xf,Yf,Zf
: matrizes de tamanho (4,n-1*m-1). `Xf(:,i)` ,`Yf(:,i)` e `Zf(:,i)` são respectivamente as coordenadas nos eixos x, y e z dos 4 pontos da i-ésima faceta de quatro lados.

fun
: uma função do Scilab.

p1
: um vetor de tamanho `n`.

p2
: um vetor de tamanho `m`.

# Descrição

`eval3dp` computa uma representação de uma faceta de quatro lados de uma superfície 3d parametrizada definida pela função `fun`. `fun(p1,p2)` computa as coordenadas nos eixos x, y e z dos pontos correspondentes da superfície como [x(i),y(i),z(i)]=fun(p1(i),p2(i)). Isto é usado para eficiência.

# Exemplos

```
p1=linspace(0,2*%pi,10);
p2=linspace(0,2*%pi,10);
deff("[x,y,z]=scp(p1,p2)",["x=p1.*sin(p1).*cos(p2)";..
                           "y=p1.*cos(p1).*cos(p2)";..
                           "z=p1.*sin(p2)"])
[Xf,Yf,Zf]=eval3dp(scp,p1,p2);
plot3d(Xf,Yf,Zf)
```

# Ver Também

genfac3d, plot3d

# Name

event handler functions — Protótipo de funções que podem se utilizadas como gerenciadoras de eventos

```
envent_handler_function(win,x,y,ibut)
```

# Parâmetros

win
: número da janela onde o evento ocorreu

x
: coordenada X em píxeis do ponteiro do mouse no momento onde o evento ocorreu

y
: coordenada Y em píxeis do ponteiro do mouse no momento onde o evento ocorreu

ibut
: número correspondente ao tipo de evento

# Descrição

Quando o modo de gerenciador de eventos está habilitado, o Scilab chamaráo gerenciador de eventos especificado a cada vez que um evento ocorre na janela gráfica. A função de gerenciamento de evento deve concordar com o protótipo acima. A cada vez que um evento ocorre, a função é chamada e os quatro eventos são ajustados de acordo com o número da janela, a posição do mouse e o tipo do evento.

O parâmetro `ibut` toma um dos seguintes valores dependendo do tipo do evento ocorrido:

ibut==0
: o botão esquerdo foi pressionado

ibut==1
: o botão do meio foi pressionado

ibut==2
: o botão direito foi pressionado

ibut==3
: o botão esquerdo do mouse foi clicado

ibut==4
: o botão do meio do mouse foi clicado

ibut==5
: o botão direito do mouse foi clicado

ibut==10
: o botão esquerdo do mouse foi clicado duas vezes

ibut==11
: o botão do meio do mouse foi clicado duas vezes

ibut==12
: o botão direito do mouse foi clicado duas vezes

ibut==-5
: o botão esquerdo do mouse foi liberado

ibut==-4
: o botão do meio do mouse foi liberado

ibut==-3
: o botão direito do mouse foi liberado

ibut==-1
: o ponteiro do mouse foi movido

ibut > =32
: uma tecla com código ascii(ibut) foi pressionada

ibut < =-32
: uma tecla com código ascii(-ibut) foi liberadav

ibut > =1000+32
: uma tecla com código ascii(ibut-1000) foi pressionada enquanto a tecla CTRL estava sendo pressionada.

ibut==-1000
: a janela de gráficos foi fechada

Por exemplo, digamos que o nome da função de gerenciamento de eventos seja fooHandler para a janela de número 0. Um clique com o botão esquerdo na janela na posição [100,150] (em píxeis) será equivalente a chamar fooHandler( 0, 100, 150, 3 ).

Ver figure_properties ou seteventhandler para mais informações sobre como especificar o nome do gerenciador de eventos.

# Exemplos

```
function my_eventhandler(win,x,y,ibut)
  if ibut==-1000 then return,end
  [x,y]=xchange(x,y,'i2f')
  xinfo(msprintf('Evento de código %d na posição do mouse é (%f,%f)',ibut,x,y))
endfunction

plot2d()
fig = gcf() ;
fig.event_handler = 'my_eventhandler' ;
fig.event_handler_enable = "on" ;
//agora:
//   -  mova o mouse sobre a janela gráfica
//   -  pressione e libere as teclas com shift pressionado ou não e com Ctrl pr
//   -  pressione um botão, espere um pouco e libere
//   -  pressione e libere um botão
//   -  clique duas vezes em um botão

fig.event_handler_enable = "off" ; //suprimindo o gerenciador de eventos
```

# Ver Também

figure_properties, seteventhandler, xgetmouse, xclick

# Autor

Jean-Baptiste Silvy

# Name

fac3d — Esboço de uma superfície 3d (obsoleta)

```
fac3d(x,y,z,[theta,alpha,leg,flag,ebox])
fac3d1(x,y,z,[theta,alpha,leg,flag,ebox])
```

# Descrição

Estas funções estão obsoletas e foram substituídas por `plot3d` e `plot3d1`.

# Ver Também

plot3d, plot3d1

# Name

fchamp — campo direcional de uma EDO 2d de primeira ordem

```
fchamp(f,t,xr,yr,[arfact,rect,strf])
fchamp(f,t,xr,yr,<opt_args>)
```

# Parâmetros

f
: uma external (função ou string) ou uma lista descrevendo a EDO.

  -
  : Pode ser um nome de função `f`, onde `f` se supõe ser uma função do tipo `y=f(t,xy [p1,..pn])`. `f` retorna um vetor coluna de tamanho 2, `y`, que fornece o valor do campo direcional `f` no ponto `xy=[x,y]` e no tempo `t`.

  -
  : Também pode ser um objeto do tipo lista, `list(f,P1,..Pn)` onde `f` é uma função do tipo `y=f(t,xy,p1,..pn)` e `Pi` fornece o valor do parâmetro `pi`.

t
: o tempo selecionado.

xr,yr
: dois vetores linhas de tamanhos n1 e n2 que definem o grid sobre o qual o campo direcional é computado.

<opt_args>
: representa uma seqüência de decarações `key1=value1,key2=value2`, ... onde `key1`, `key2,...` pode ser um dos seguintes : arfact, rect, strf (ver abaixo).

arfact,rect,strf
: argumentos opcionais, ver `champ`.

# Descrição

`fchamp` é utilizado para desenhar o campo direcional de uma EDO 2d de primeira ordem definida pela função externa `f`. Note que se a EDO é autônoma, o argumento `t` não possui utilidade, mas deve ser fornecido.

Entre com o comando `fchamp()` para visualizar uma demonstração.

# Exemplos

```
deff("[xdot] = derpol(t,x)",..
        ["xd1 = x(2)";..
         "xd2 = -x(1) + (1 - x(1)**2)*x(2)";..
         "xdot = [ xd1 ; xd2 ]"])
xf= -1:0.1:1;
yf= -1:0.1:1;
fchamp(derpol,0,xf,yf)
clf()
fchamp(derpol,0,xf,yf,1,[-2,-2,2,2],"011")
```

# Ver Também

champ, champ1

# Autor

J.Ph.C.

# Name

fcontour — curvas de nível sobre uma superfície 3d definida por uma função

```
fcontour(xr,yr,f,nz,[theta,alpha,leg,flag,ebox,zlev])
fcontour(xr,yr,f,nz,<opt_args>)
```

# Descrição

Esta função está obsoleta. Está agora inclusa na função contour.

# Autor

J.Ph.C.

# Name

fcontour2d — curvas de nível de uma superfície definida por uma função em um esboço 2d

```
fcontour2d(xr,yr,f,nz,[style,strf,leg,rect,nax])
fcontour2d(xr,yr,f,nz,<opt_args>)
```

# Descrição

Esta função está obsoleta. Está agora inclusa na função contour2d.

# Autor

J.Ph.C.

# Name

fec — Esboço pseudo-colorido triangular de uma função definida por uma malha triangular

```
fec(x,y,triangles,func,<opt_args>)
fec(x,y,triangles,func,[strf,leg,rect,nax,zminmax,colminmax,colout,mesh])
```

# Parâmetros

x,y
: dois vetores de tamanho `n`, `(x(i),y(i))` fornece as coordenadas do nó `i`

func
: vetor de tamanho `n` : `func(i)` fornece o valor no nó `i` da função para a qual queremos o esboço pseudo-colorido.

triangles
: é uma matriz `[Ntr,5]`. Cada linha de `triangles` especifica um triângulo da malha `triangle(j) = [number,node1,node2,node3,flag]`. `node1,node2,node3` são os números dos nós que constituem o triângulo. number é o número do triângulo e flag é um inteiro que não é usado na função fec

<opt_args>
: representa uma seqüência de declarações `key1=value1, key2=value2`,... onde `key1`, `key2,...` podem ser um dos seguintes: strf, leg, rect, nax, zminmax, colminmax, colout, mesh (ver plot2d para os quatro primeiros).

strf,leg,rect,nax
: ver plot2d

zminmax
: vetor com dois compontentes[zmin zmax] (útil para animações em particular)

colminmax
: vetor de dois inteiros positivos [colmin colmax]

colout
: vetor de dois inteiros [under_min_col upper_max_col]

mesh
: escalar booleano, valor padrão %f (deve ser verdadeiro se você também quiser exibir a malha)

# Descrição

Esta função é boa para se desenhar soluções de elementos finitos triangulares lineares ou simplesmente para se exibir uma função definida sobre uma triangularização. A interpolação de cores é feita através de um software de computação e, portanto, não é tão rápida.

A função colorbar pode se utilizada para se visualizar a escala de cores (ver a seção de exemplos).

O argumento `zminmax` fornece os valores de z asosicados às primeira e últimas cores (do mapa de cores corrente). Mais exatamente, se o mapa de cores possue nc cores e se notarmos *dz = (zmax-zmin)/nc*, então a parte da triangularização onde *zmin + (i-1)dz <= z < zmin + i dz* é preenchida com a cor *i*). Por padrão *zmin = min(func)* e *zmax = max(func)*. Se você quiser uma animação com valores da função que variam no tempo, tome para zmin e zmax os valores máximo e mínimo globais ou algo próximo.

O argumento `colout` permite que o usuário escolha as cores para as duas regiões extremas *{func < zmin}* and *{func > zmax}*, `under_min_col` e `upper_max_col` podem ser iguais (independentemente) a:

-1
: neste caso a mesma cor que a zona de vizinhança é utilizada (CUIDADO: você não vê que o limite correspondente é cruzado), este é o caso padrão.

0
: neste caso, a região de extremidade não é pintanda.

k
: (k sendo um índice de cor válido no mapa de cores corrente) a região de extremidade é pintada com a cor k.

Se você não quiser usar o mapa de cores completo, você pode utilizar o argumento *1 <= colmin < colmax <= nc* (nc sendo o número de cores do mapa de cores corrente) para utilizar apenas a sub-parte [colmin,colmax] do mapa de cores. (por padrão, todas as cores do mapa de cores são utilizadas).

Ver arquivos de demonstração `demos/fec`:

`fec.ex1` é um simples arquivo de exemplo no qual uma malha e uma função sobre esta malha é completamente construída em sintaxe do Scilab

`fec.ex2` é um exemplo no qual a malha e os valores da função foram computados por um construtor de malhas externo (malha do tipo amdba) e um programa externo. Um conjunto de macros ( fornecido pelo arquivo `macros.sci`) pode ser utilizado para ler os arquivos de dados no Scilab e esboçar os resultados.

# Exemplos

```
// definindo uma mini-triangularização (4 vértices, 2 triângulos)
x = [0 1 0 -1];
y = [0 0 1  1];
T = [1 1 2 3 1;
     2 3 4 1 1];
z = [0 1 0 -1];  // valores da função em cada vértice
xbasc()
xset("colormap",jetcolormap(64))
subplot(1,2,1)
   colorbar(-1,1)
   fec(x,y,T,z,strf="040",mesh=%t)
   xtitle("exemplo de fec (com a malha)")
subplot(1,2,2)
   colorbar(-1,1)
   fec(x,y,T,z,strf="040")  // rmq: mesh=%f por padrão
   xtitle("exemplo de fec (em a malha)")
xselect()

// este exemplo mostra o efeito de zminmax e usa os
// dados de exemplos anteriores (você tem que executá-los primeiro)
xbasc()
xset("colormap",jetcolormap(64))
colorbar(-0.5,0.5)  // cuidado, a barra de cores deve ser ajustada na mão!
fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], mesh=%t)
xtitle("exemplo de fec: utilizando o argumento zminmax")
xselect()

// este exemplo mostra os efeitos de zminmax e colout. Utiliza
// também os dados de exemplos anteriores (você tem que executá-los primeiro)
xbasc()
xset("colormap",jetcolormap(64))
```

```
subplot(2,2,1)
   colorbar(-0.5,0.5)
   fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[0 0], mesh=%t)
   xtitle("exemplo de fec: utilizando zminmax e colout =[0 0]")
subplot(2,2,2)
   colorbar(-0.5,0.5)
   fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[32 32], mesh=%t)
   xtitle("exemplo de fec : utilizando zminmax e colout =[32 32]")
subplot(2,2,3)
   colorbar(-0.5,0.5)
   fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[-1 0], mesh=%t)
   xtitle("exemplo de fec : utilizando zminmax e colout =[-1 0]")
subplot(2,2,4)
   colorbar(-0.5,0.5)
   fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[0 -1], mesh=%t)
   xtitle("exemplo de fec: utilizando zminmax e colout =[0 -1]")
xselect()

// este exemplo mostra um recurso de colminmax:
// utilizando dois mapas de cores para dois subesboços.
// Também utiliza dados de exemplos anteriores.
xbasc()
xset("colormap",[hotcolormap(64);jetcolormap(64)])
subplot(1,2,1)
   colorbar(-1,1,[1 64])
   fec(x,y,T,z,strf="040", colminmax=[1 64], mesh=%t)
   xtitle("fec utilizando mapa de cores hot ")
subplot(1,2,2)
   colorbar(-1,1,[65 128])
   fec(x,y,T,z,strf="040", colminmax=[65 128], mesh=%t)
   xtitle("fec utilizando mapa de cores jet ")
xselect()
```

# Ver Também

colorbar, Sfgrayplot, Sgrayplot

# Name

fec_properties — Descrição das propriedades da entidade Fec

# Descrição

A entidade Fec é uma folha na hierarquia de entidade gráficas. Representa um esboço 2d de elementos finitos (ver `fec`, `Sgrayplot`).

parent:
: esta propriedade contém o manipulador para a raiz. A raiz da entidade Fec deve ser do tipo `"Axes"` ou `"Compound"`.

children:
: esta propriedade contém um vetor com os galhos do manipulador . Contudo, manipuladores `Fec` correntemente não possuem galhos.

visible:
: este campo contém o valor da propriedade `visible` para a entidade . Pode ser `"on"` ou `"off"` . Por padrão, o esboço é visível, a propriedade é `"on"`. Se `"off"` o esboço não é exibido na tela.

data:
: uma matriz de três colunas `[x,y,f]`, onde `x(i)` e `y(i)` são as coordenadas do i-ésimo nó. `f(i)` é o valor associado ao nó `i`.

triangles:
: uma matriz de cinco colunas `[tn,n1,n2,n3,flag]`. `tn(j)` é o número do triângulo. `n1(j)`, `n2(j)` e `n3(j)` são índices dos nós que constituem o triângulo. (`flag(j)` não é usado).

z_bounds:
: este vetor de tamanho 2, `[zmin,zmax]`, fornece os valores z associados às primeira e última cores (do mapa de cores corrente). Mais exatamente, se o mapa de cores possui nc cores e se notarmos `dz = (zmax-zmin)/nc`, então a parte da triângularização onde `zmin + (i-1)dz <= z < zmin + i dz` é preenchida com a cor `i`). Por padrão o valor da propriedade `z_bounds` é `[0,0]`. Neste caso, `zmin` e `zmax` são automaticamente ajustados para os valores mínimo e máximo do argumento `func`.

outside_color:
: este vetor de tamanho 2, `[cmin, cmax]`, define a cor utilizada quando os valores dos nós estão fora do intervalo `z_bounds = [zmin,zmax]`. Quando os valores dos nós são menores do que `zmin`, a cor com índice `cmin` é utilizada. Quando os valores dos nós são menores do que `zmax` a cor com índice `cmax` é utilizada. Por padrão, o valor da propriedade `outside_color` é `[0,0]`. Neste caso, `cmin` e `cmax` são automaticamente ajustados para os limites do mapa de cores. Se `cmin` ou `cmax` são negativos, então os valores fora do intervalo z_bounds não são exibidos, eles aparecem transparentes.

color_range:
: este vetor de tamanho 2, `[rmin, rmax]`, permite utilizar apenas uma parte do mapa de cores para a exibição. `rmin` e `rmax` são os índices do mapa de cores. Se ambos forem maiores do que 1, então o mapa de cores utilizados para exibir a entidade fec é `colormap(rmin:rmax)` onde `colormap` é o mapa de cores da figura raiz. Por padrão, O valor da propriedade `color_range` é `[0,0]`. Neste caso, todo o mapa de cores é utilizado.

line_mode:
: Se `"on"`, o wireframe (esqueleto) encerrando os triângulos é desenhado. Se `"off"`, apenas os triângulos de dentro são desenhados.

foreground:
: este índice de cor especifica a cor da malha. Se a propriedade `line_mode` for `"on"`, o wireframe é desenhado utilizando-se esta cor.

clip_state:
este campo contém o valor da propriedade `clip_state` para o fec. o valor de clip_state pode ser :

- `"off"` significa que o fec não é recortado.
- `"clipgrf"` significa que o fec é recortado fora da caixa dos eixos.
- `"on"` significa que o fec é recortado fora do retângulo dado pela propriedade clip_box.

clip_box:
este campo determina a propriedade `clip_box`. Por padrão seu valor é uma matriz vazia se a propriedade clip_state é "off". Em outros casos, o vetor `[x,y,w,h]` (ponto superior esquerdo, largura, altura) define as porções do retângulo a ser exibido, contudo o valor da propriedade `clip_state` será alterado.

user_data:
este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados da entidade Fec e recuperá-la.

# Exemplos

```
x=-10:10; y=-10:10;m =rand(21,21);
Sgrayplot(x,y,m);
a=get("current_axes");
f=a.children.children(1)
f.data(:,3)=(1:size(f.data,1))';
a.parent.color_map=hotcolormap(64);
```

# Ver Também

set, get, delete, fec, Sgrayplot, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

fgrayplot — esboço 2d de uma superfície definida por uma função utilizando cores

```
fgrayplot(x,y,f,[strf,rect,nax])
fgrayplot(x,y,f,<opt_args>)
```

# Parâmetros

x,y
: vetor linha de reais

f
: função externa do tipo `y=f(x,y)`.

<opt_args>
: representa uma seqüência de declarações `key1=value1, key2=value2`,... onde `key1`, `key2, ...` pode ser um dos seguintes: rect, nax, strf ou axesflag e frameflag (ver `plot2d`).

strf,rect,nax
: ver `plot2d`.

# Descrição

`fgrayplot` realiza um esboço 2d de uma superfície dada por `z=f(x,y)` em um grid definido por `x` e `y`. Cada retângulo no grid é preenchido com um nível de cinza ou de cor dependendo do valor médio de `z` nas quinas do retângulo.

Entre com o comando `fgrayplot()` para visualizar uma demonstração.

# Exemplos

```
t=-1:0.1:1;
deff("[z]=my_surface(x,y)","z=x**2+y**2")
fgrayplot(t,t,my_surface,rect=[-2,-2,2,2])
```

# Ver Também

grayplot, plot2d, Sgrayplot, Sfgrayplot

# Autor

J.Ph.C.

# Name

figure_properties — Descrição das propriedades da entidade gráfica Figure (figura)

# Descrição

A figura é o topo da hierarquia de entidades gráficas. Esta entidade contém um número de propriedades designadas a controlar muitos aspectos da exibição de objetos gráficos do Scilab. Estas propriedades estão dividas em duas categorias. Propriedades que contém informação sobre a própria figura e outras relacionadas a valores padrões para criação de galhos.

Figure properties:

children:
: estes manipuladores representam o vetor dos galhos da figura. Note que todos os galhos de figuras são do tipo `"Axes"`. Também tenha em mente que, ao se criar uma entidade figura (utilizando o comando scf), uma entidade `Axes` (de eixos) é simultaneamente criada.

figure_style:
: o valor deste campo determina o estilo da figura. Desde o Scilab 5.0, o modo gráfico antigo foi desabilitado. Esta propriedade é mantida por compatibilidade, mas não pode ser alterada.

figure_position:
: este campo contém a posição em pixel da janela de gráficos na tela. É um vetor `[x,y]` definindo a posição do canto superior esquerdo da janela. A posição `[0,0]` é o canto superior esquerdo da tela.

  A posição inicial de uma janela de gráficos é tomada a partir da entidade figura padrão (ver gdf). A única excessão é quando o valor `figure_position` da figura padrão é `[-1,-1]`. Neste caso, A posição inicial da janela de gráficos é ajustada automaticamente pelo sistema de janelamento.

figure_size:
: esta propriedade controla o tamanho em pixel da janela de gráficos da tela. O tamanho é o vetor `[width,height]`.

axes_size:
: usado para especificar o tamanho da janela de gráficos virtual. O tamanho é o vetor `[width,height]` . A janela de gráficos virtual deveria ser maior que a parte realmente visível na tela. Esta propriedade não poderia ser alterada se a figura está aportada com outros elementos.

auto_resize:
: esta propriedade determina se a janela de gráficos é redimensionada. Se o valor é `"on"` então a propriedade `axes_size` é igualada a `figure_size` e os galhos eixos são ampliados de modo correspondente. Se o valor é `"off"`, `axes_size` não pode ser redimensionado quando `figure_size` é modificado.

viewport:
: posição da parte visível do gráfico na janela

figure_name:
: este campo contém o nome da figura. O nome é um string exibido no topo da janela de gráficos. Pode conter um único substring `%d` que será substituido por `figure_id`. Em nenhum outro caso o caractere % é permitido dentro do nome.

figure_id:
: este campo contém o identificador da figura. É um número inteiro que é ajustado na criação da figura e não pode ser mudado depois.

info_message:
este string ajusta o texto que será exibido na barra de informações da janela.

color_map:
propriedade que define o mapa de cores utilizado pela figura. O mapa de cores é uma matriz `m` por `3`. `m` é o número de cores. O número da cor i é dado pela tripla 3-uple `R`, `G`, `B` correspondentes respectivamente às intensidades de vermelho, verde e azul entre 0 e 1.

pixmap:
esta propriedade controla o status de pixmap de uma janela de gráficos. Se o valor é `"off"` os gráficos são diretamente exibidos na tela. Se é `"on"` os gráficos são feitos em um pixmap e são enviados à janela através do comando `show_pixmap()`.

pixel_drawing_mode:
este campo define a operação bit a bit usada para desenhar o pixel na tela. O modo padrão é `copy` que significa dizer que o pixel é desenhado como requerido. De modo mais geral, a operação bit a bit é realizada entre a cor do pixel fonte e a cor do pixel já presente na tela. As operações são: `"clear"`, `"and"`, `"andReverse"` , `"copy"`, `"andInverted"` , `"noop"` , `"xor"` , `"or"` , `"nor"` , `"equiv"` , `"invert"` , `"orReverse"` , `"copyInverted"` , `"orInverted"` , `"nand"` , `"set"` ,

immediate_drawing:
esta propriedade controla a exibição da figura. Pode ser `"on"` (modo padrão) ou `"off"`. É utilizada par atrasar uma grande sucessão de comandos gráficos (implicando vários esboços ou re-esboços). Note que, ao se utilizar os comandos `drawlater` ou `drawnow` É utilizada par atrasar uma grande sucessão de comandos gráficos (implicando vários esboços ou re-esboços). Note que, ao se utilizar os comandos `'off'` ou `'on'`).

background:
esta propriedade controla a cor de plano de fundo da figura. O valor é um índice de cor correspondente ao mapa de cores corrente.

event_handler
um string. O nome da função do Scilab que deverá manipular os eventos. Note que definir um string vazio desabilitará o manipulador de eventos. Para mais informações sobre o manipulador de eventos, veja a ajuda event handler functions.

event_handler_enable
habilita ou desabilita o manipulador de eventos. Seu valor deve ser "on" ou "off".

user_data:
este campo pode ser utilizado para se armazenar qualquer variável Scilab na estrutura de dados da entidade figura e recuperá-la.

tag:
este campo pode ser utilizado para armazenar um string geralmente utilizado para identificar o controle. Ele permite fornecer um "nome". Principalmente utilizado em conjunção a findobj().

Valores padrões para galhos:

visible:
este campo determina se o conteúdo da figura deve ser redesenhado. O valor pode ser `"on"` ou `"off"`.

rotation_style:
este campo está relacionado ao botão "3D Rot". Toma `unary` como valor (padrão) para rotacionar apenas o esboço 3d selecionado. Em outros casos o valor pode ser `multiple`: todos os esboços 3d são rotacionados.

Nota sobre valores padrões:

todas estas propriedades e campos listados herdam valores de valores padrões armazenados num modelo de figura. Estes valores padrões podem ser vistos e salvos. Para fazê-lo, use o comando `get("default_figure")`: ele retorna o manipulador gráfico de uma figura. Note que nenhuma janela de gráficos é criada por este comando. As próximas figuras criadas herdarão valores deste modelo (ver o exemplo 2 abaixo).

# Exemplos

```
lines(0) // desabilitando paginamento vertical
//Exemplo 1
f=get("current_figure") //obtendo o manipulador da figura corrente:
                                          //se nenhum existir, cria uma figura
f.figure_position
f.figure_size=[200,200]
f.background=2
f.children  // pode-se ver que uma entidade Axes já existe
delete(f);
f=gcf(); // atalho de macro  &lt;=&gt; f=get("current_figure")
f.pixmap = "on" // ajusta o status de pixmap para on
plot2d() // nada acontece na tela...
show_pixmap() // ...exibe o pixmap na tela
//Exemplo 2 : configurações de default_figure
df=get("default_figure") // obtendo os valores padrões (o atalho é gdf() )
// vamos mudar os padrões...
df.color_map=hotcolormap(128)
df.background= 110 // ajustando o plano de fundo para um tipo de amarelo (note
scf(122); // criando uma nova figura de número122 com o novo padrão
plot2d()
scf(214);
t=-%pi:0.3:%pi;
plot3d(t,t,sin(t)'*cos(t),35,45,'X@Y@Z',[15,2,4]);
```

# Ver Também

lines, set, get, scf, gcf, gdf, gca, gda, axes_properties, show_pixmap, clear_pixmap, hotcolormap, event handler functions

# Autor

Djalel ABDEMOUCHE

# Name

fplot2d — esboço 2d de uma curva definida por uma função

```
fplot2d(xr,f,[style,strf,leg,rect,nax])
fplot2d(xr,f,<opt_args>)
```

# Parâmetros

xr
: vetor

f
: função externa do tipo `y=f(x)` i.e. uma função scilab ou rotina dinamicamente linkada ("ligada") referida como um string.

style,strf,leg,rect,nax
: ver `plot2d`

<opt_args>
: ver `plot2d`

# Descrição

`fplot2d` esboça uma curva definida pela função externa `f`. A curva é aproximada por uma interpolação linear seccional utilizando os pontos `(xr(i),f(xr(i))`. Os valores de f(x) são obtidos por `feval(xr,f)`.

Entre com o comando `fplot2d()` para visualizar uma demonstração.

# Exemplos

```
deff("[y]=f(x)","y=sin(x)+cos(x)")
x=[0:0.1:10]*%pi/10;
fplot2d(x,f)
clf();
fplot2d(1:10,'parab')
```

# Ver Também

plot2d, feval, paramfplot2d

# Autor

J.Ph.C.

# Name

fplot3d — esboço 3d de uma superfície definida por uma função

```
fplot3d(xr,yr,f,[theta,alpha,leg,flag,ebox])
fplot3d(xr,yr,f,<opt_args>)
```

# Parâmetros

xr
: vetor linha de tamanho n1.

yr
: vetor linha de tamanho n2.

f
: external do tipo z=f(x,y).

theta,alpha,leg,flag,ebox
: ver `plot3d`.

<opt_args>
: ver `plot3d`.

# Descrição

`fplot3d` esboça uma superfície definida pela função externa `f` no grid definido por `xr` e `yr`.

Entre com o comando `fplot3d()` para ver uma demonstração.

# Exemplos

```
deff('z=f(x,y)','z=x^4-y^4')
x=-3:0.2:3 ;y=x ;
clf() ;fplot3d(x,y,f,alpha=5,theta=31)
```

# Ver Também

plot3d

# Autor

J.Ph.C.

# Name

fplot3d1 — Esboço 3d em escala de cinza ou colorido de nível de uma superfície definida por uma função

```
fplot3d1(xr,yr,f,[theta,alpha,leg,flag,ebox])
fplot3d1(xr,yr,f,<opt_args>)
```

# Parâmetros

xr
: vetor linha de tamanho n1.

yr
: vetor linha de tamanho n2.

f
: função externa (external) do tipo z=f(x,y).

theta,alpha,leg,flag,ebox
: ver `plot3d1`.

<opt_args>
: ver `plot3d`.

# Descrição

`fplot3d1` esboça em 3d, em escala de cinza ou colorido, níveis de uma superfície definida pela função externa `f` no grid definido por `xr` e `yr`.

Entre com o comando `fplot3d1()` para visualizar uma demonstração.

# Exemplos

```
deff('z=f(x,y)','z=x^4-y^4')
x=-3:0.2:3 ;y=x ;
clf() ;fplot3d1(x,y,f,alpha=5,theta=31)
```

# Ver Também

plot3d1

# Autor

J.Ph.C.

# Name

gca — Retorna o manipulador da entidade Axes corrente

```
a = gca()
```

# Parâmetros

a
: manipulador, o manipulador da entidade Axes corrente.

# Descrição

Esta rotina retorna o manipulador da entidade Axes da figura corrente

# Exmplos

```
subplot(211)
a=gca() //obtendo os eixos correntes
a.box="off";
t=-%pi:0.3:%pi;plot3d(t,t,sin(t)'*cos(t),80,50,'X@Y@Z',[5,2,4]);
subplot(212)
plot2d(); //esboço simples
a=gca() //obtendo os eixos correntes
a.box="off";
a.x_location="middle";
a.parent.background=4;
delete(gca()) // deletando os eixos correntes
xdel(0) //deletando uma janela de gráficos
```

# Ver Também

gda, gcf, gce, get, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

gce — Retorna o manipulador da entidade corrente

```
e = gce()
```

# Parâmetros

e
: manipulador da entidade corrente

# Descrição

Esta rotina retorna o manipulador da última (e ainda existente) entidade criada.

# Exemplos

```
a=gca() //obtendo o manipulador dos novos eixos criados
a.data_bounds=[1,1;10,10];
a.axes_visible = 'on' ;
for i=1:5
  xfrect(7-i,9-i,3,3);
  e=gce();
  e.background=i;
end
delete(); // deletando a entidade corrente
delete(gce()); // deletando a entidade corrente
delete(gcf()); // deletando a figura corrente
```

# Ver Também

gcf, gca, get, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

gcf — Retorna o manipulador da janela GUI ou de gráficos corrente.

```
h = gcf()
```

# Parâmetros

h
: manipulador

# Descrição

Esta função retorna o manipulador da janela gráfica corrente

# Exemplos

```
f=gcf(); // criando uma figura
f.figure_size= [610,469]/2;
f.figure_name= "Foo";

f=figure(); // criando uma figura
h=uicontrol(f,"style","listbox","position", [10 10 150 160]); // criando um lis
set(h, "string", "item 1|item 2|item3");// preenchendo a lista
set(h, "value", [1 3]); // selecionando os itens 1 e 3 na lista
gcf()

scf(0); // tornando a janela de gráficos 0 a figura corrente
gcf()  // obtendo o manipulador gráfico da figura corrente

figure(f) // tornando a janela GUI f a figura corrente
gcf() // retorna o manipulador gráfico da figura corrente
```

# Ver Também

gdf, gca, gce, get, scf, set, graphics_entities, uicontrol

# Autor

Serge Steer, INRIA

# Name

gda — Retorna o manipulador dos eixos padrões.

```
a = gda()
a = get("default_axes")
```

# Parâmetros

a
: o manipulador dos eixos padrões.

# Descrição

Os eixos padrões são uma entidade gráfica (entidade Axes) que nunca é desenhada. Ela usada como referência para os valores de propriedades padrões dos eixos. Estes valores de propriedades padrões são usados para inicializar novos eixos dentro de figuras.

A função `gda` retorna o manipulador dos eixos padrões. O usuário pode utilizar este manipulador para ajustar ou obter os valores padrões de propriedades dos eixos.

Note que uma entidade gráfica padrão equivalente também existe para entidades Figure (ver `gdf`).

# Exemplos

```
a=gda() // obtendo o manipulador dos eixos modelos
// ajustando suas propriedades
a.box="off";
a.foreground=2;
a.labels_font_size=3;
a.labels_font_color=5;
a.sub_tics=[5 5 3];
a.x_location="top";

//os eixos modelos são agora usados
subplot(211) //criando uma nova entidade Axes
plot2d()

// veja outros modelos de propriedades
a.background=color('gray')
a.grid=[5 5 5];
subplot(212)
t=0:0.1:5*%pi;
plot2d(sin(t),cos(t))

set(a,"default_values",1); // retornando para os valores padrões do modelo
                           // ver a função sda()
clf()
plot2d(sin(t),cos(t))
```

# Ver Também

gdf, sda, sdf, clf, gca, get, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

gdf — Retorna o manipulador da figura corrente

```
f = gdf()
f = get("default_figure")
```

# Parâmetros

f
: manipulador da figura corrente

# Descrição

A figura padrão é uma entidade gráfica (entidade Figure) que nunca é desenhada. É usada como referência para os valores padrões das propriedades da figura. Estes valores padrões são utilizados para iniciar novas figuras.

A função `gdf` retorna o manipulador da figura corrente. O usuário pode utilizar este manipulador para ajustar ou obter os valores padrões das propriedades da figura.

Note que uma entidade gráfica padrão equivalente também existe para entidades Axes (de eixos) (ver `gda`).

# Exemplos

```
f=gdf() // obtendo o manipulador da figura modelo
// ajustando suas propriedades
f.background=7;
f.figure_name="Function gdf()";
f.figure_position=[-1 100];
f.auto_resize="off";
f.figure_size=[300 461];
f.axes_size=[600 400];
plot2d() //criando uma figura
scf(1);
plot3d() //criando uma segunda figura
set(f,"default_values",1); // retornando para os valores padrões do modelo da f
                           // ver a função sdf()
scf(2);
plot2d()
```

# Ver Também

gda, sdf, sda, gcf, get, scf, set, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

ged — Editor Gráfico do Scilab

```
ged(action, fignum)
```

# Parâmetros

action
: Real: ação a ser executada na janela gráfica dada por `fignum`:

- 1: Ajusta a janela `fignum` como sendo a corrente.
- 2: Redesenha a janela `fignum`.
- 3: Limpa a janela `fignum`.
- 4: Pede que o usuário selecione uma entidade gráfica para efetuar cópia.
- 5: Cola a última entidade gráfica copiada pela ação `4`.
- 6: Pede ao usuário para que selecione uma entidade gráfica e então a mova.
- 7: Pede ao usuário que selecione uma entidade gráfica para efetuar deleção.
- 8: Inicia uma GUI para se editar propriedades da janela.
- 9: Inicia uma GUI para se editar propriedade dos eixos correntes.
- 10: Inicia um selecionador para selecionar um objeto e editá-lo através da GUI do Editor Gráfico.
- 11: Para o selecionador de entidades

fignum
: Real: número da janela de gráficos, a figura a ser editada

# Descrição

`ged` inicia um Editor Gráfico do Scilab sobre a figura de número `fignum` e executa a ação fornecida por `action`.

# Autor

V.C.

# Name

genfac3d — computa facetas de uma superfície 3d

```
[xx,yy,zz]=genfac3d(x,y,z,[mask])
```

# Parâmetros

xx,yy,zz
: matrizes de tamanho (4,n-1xm-1). `xx(:,i)` ,`yy(:,i)` e `zz(:,i)` são respectivamente as coordenadas nos eixos x, y e z dos 4 pontos da i-ésima faceta de quatro lados

x
: vetor de coordenadas no eixo y de tamanho n

y
: vetor de coordenadas no eixo y de tamanho n

z
: matriz de tamanho (m,n). `z(i,j)` é o valor da superfície no ponto (x(i),y(j)).

mask
: matriz de valores booleanos opcional de mesmo tamanho que `z` utilizada para selecionar entradas de `z` a serem representadas por facetas

# Descrição

`genfac3d` computa uma faceta de quatro lados de uma superfície 3d `z=f(x,y)` definida por `x`, `y` e `z`.

# Exemplos

```
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
[xx,yy,zz]=genfac3d(t,t,z);
plot3d(xx,yy,zz)
```

# Ver Também

eval3dp, plot3d

# Name

geom3d — projeção 3d para 2d após um esboço 3d

```
[x,y]=geom3d(x1,y1,z1)
```

# Parâmetros

x1,y1,z1
: vetores reais de mesmo tamanho (pontos em 3d).

x,y
: vetores reais de mesmos tamanhos que `x1`, `y1` e `z1`.

# Descrição

Após o uso de uma função de esboço 3d como `plot3d`, `plot3d1` ou `param3d`, `geom3d` gives the mapping between a point in 3D space `(x1(i),y1(i),z1(i))` e o ponto correspondente `(x(i),y(i))` no plano 2d projetado. Então, todas as primitivas de gráficos 2d que funcionam em `(x,y)` podem ser usadas para superposição no esboço 3d.

# Exemplos

```
deff("[z]=surface(x,y)","z=sin(x)*cos(y)")
t=%pi*(-10:10)/10;
// esboço 3d da superfície
fplot3d(t,t,surface,35,45,"X@Y@Z")
// agora, (t,t,sin(t).*cos(t)) é uma curva na superfície
// que pode ser desenhada utilizando-se geom3d e xpoly
[x,y]=geom3d(%pi/2,0,surface(%pi/2,0))
```

# Autor

J.Ph.C.

# Name

get — Recupera um valor de propriedade de uma entidade de gráficos ou um objeto Interface do Usuário.

```
h=get(prop)
val=get(h,prop)
val=h.prop
```

# Parâmetros

h
: manipulador da entidade da qual se deseja recuperar uma propriedade. `h` pode ser um vetor de manipuladores e, neste caso, get retorna os valores da propriedade definidos para todos os objetos identificados por h. `h` também pode ser *0* para se retornar as propriedades do objeto raiz.

prop
: string com o nome da propriedade.

val
: objeto Scilab, o valor da propriedade.

# Descrição

Esta rotina pode ser utilizada para se recuperar o valor de uma propriedade especificada de uma entidade de gráficos ou objeto GUI. Neste caso, é equivalente a se usar o operador ponto ('.') em um manipulador. Por exemplo, `get(h,"background")` é equivalente a `h.background`.

Nomes de propriedades são strings. Para obter a lista de todas as propriedade existentes ver graphics_entities ou uicontrol para objetos de Interface do Usuário

`get` também pode ser chamado com apenas uma propriedade como argumento. Neste caso, a propriedade deve ser uma das seguintes:

current_entity or hdl
: retorna o manipulador da última (e ainda existente) entidade criada. `get('current_entity')` e `get('hdl')` equivalem a gce.

current_figure
: retorna o manipulador da figura (Figure) corrente. `get('current_figure')` equivale a gcf.

current_axes
: retorna o manipulador da entidade de eixos (Axes) corrente. `get('current_axes')` equivale a gca.

default_figure
: retorna o manipulador da figura padrão. `get('default_figure')` equivale a gdf.

default_axes
: retorna o manipulador dos eixos padrões. `get('default_axes')`equivale a gda.

figures_id
: retorna um vetor contendo os ids de todas as figuras gráficas abertas. `get('figures_id')` equivale a winsid.

# Exemplos

```
// para entidades gráficas
clf()

// objetos gráficos simples
subplot(121);
x=[-.2:0.1:2*%pi]';
plot2d(x-2,x.^2);
subplot(122);
xrect(.2,.7,.5,.2);
xrect(.3,.8,.3,.2);
xfarc(.25,.55,.1,.15,0,64*360);
xfarc(.55,.55,.1,.15,0,64*360);
xstring(0.2,.9,"Exemplo &lt;&lt;UM CARRO&gt;&gt;");

h=get("current_entity") //obtendo o novo objeto criado
h.font_size=3;

f=get("current_figure") //obtendo a figura corrente
f.figure_size
f.figure_size=[700 500];
f.children
f.children(2).type
f.children(2).children
f.children(2).children.children.thickness=4;

a=get("current_axes") //obtendo os eixos correntes
a.children.type
a.children.foreground //obtendo a cor de primeiro plano de um conjunto de objet
a.children.foreground=9;

// para objetos de Interface do Usuário
h=uicontrol('string', 'Button'); // Abrindo uma janela com um botão.
p=get(h,'position'); // obtendo o aspecto geométrico do botão
disp('Largura do botão: ' + string(p(3))); // imprimindo a largura do botão
close(); // fechando a figura
```

# Ver Também

uicontrol, root_properties, graphics_entities, set

# Autor

Djalel ABDEMOUCHE

# Name

get_figure_handle — Retorna o manipulador de uma figura a partir de seu id

```
f = get_figure_handle(figure_id)
```

# Parâmetros

figure_id
: inteiro, o id da figura a ser recuperada

f
: o manipulador da figura correspondente

# Descrição

`get_figure_handle` permite recuperar o manipulador de uma figura gráfica desde que se conheça o seu identificador. Se uma figura com o id especificado existir, ela é retornada. Em caso contrário, uma matriz vazia é retornada.

# Exemplos

```
// criando algumas figuras
scf(0);
scf(5);
scf(12);

// obtendo o manipulador da figura com id 5
f5 = get_figure_handle(5);
// a figura corrente ainda é aquela com id 12
gcf()
// obtendo uma figura inexistente
f42 = get_figure_handle(42);
```

# Ver Também

set, get, gcf, scf, graphics_entities

# Autor

Jean-Baptiste Silvy INRIA

# Name

getcolor — abre um diálogo que exibe as cores no mapa de cores corrente

```
c=getcolor(title,[cini])
c=getcolor()
```

# Parâmetros

title
: string, o título do diálogo

cini
: identificador da cor selecionada inicial. O valor padrão é 1.

c
: identificador da cor selecionada ou [] se o usuário tiver clicado no botão "Cancel" (cancelar).

# Descrição

`getcolor` abre uma janela que exibe o palete do mapa de cores corrente. O usuário pode clicar numa cor para exibir seu identificador e valores RGB. `getcolor` retorna o identificador da cor selecionada ou [] se o usuário clicar no botão "Cancel".

# Ver Também

color, colormap, getmark, getfont

# Name

getfont — Diálogo para selecionar fonte da letra **. Função obsoleta.**

```
[fId,fSize]=getfont()
[fId,fSize]=getfont(str)
fnt=getfont()
fnt=getfont(str)
fnt=getfont(S=str,font=[fId,fSize])
```

# Parâmetros

str
: caractere (por exemplo: "a")

fId
: inteiro, o número da fonte selecionada

fSize
: inteiro, o tamanho da fonte selecionada

fnt
: vetor [fId,fSize]

# Descrição

Esta função, que foi projetada para trabalhar com a função xset, também está obsoleta. Utilize o editor de propriedade ged ao invés.

`getfont` abre uma janela gráfica para seleção de uma fonte. O usuário deve escolher uma fonte e um tamanho clicando no caractere exibido correspondente. Apertar uma tecla faz com que o caractere exibido mude.

# Exemplos

```
[fId,fSize]=getfont();
xset("font",fId,fSize)
plot2d(0,0,rect=[0 0 10 10],axesflag=0)
xstring(5,5,"string")
```

# Ver Também

ged, text_properties

# Name

getlinestyle — Diálogo para seleção de estilo de linha. **Função obsoleta.**

```
k=getlinestyle()
```

# Parâmetros

k

inteiro, o estilo de linha selecionado ou [] se o botão "Cancel" (cancelar) for clicado.

# Descrição

Esta função, que foi projetada para trabalhar com a função xset, **também está obsoleta**. Utilize o editor de propriedades ged.

`getlinestyle` abre uma janela de gráfico para selecionar um estilo de linha.

# Exemplos

```
x=0:0.1:10;
plot2d(x,sin(x))
e=gce(); // armazenando o Compound contendo o esboço
e.children(1).line_style = getlinestyle();
```

# Ver Também

ged, set, segs_properties, segs_properties

# Name

getmark — Diálogo para selecionar marcas (símbolos). **Função obsoleta.**

```
[mark,mrkSize]=getmark()
```

# Parâmetros

mark
: inteiro, o número da marca selecionada

mrkSize
: inteiro, o tamanho da marca selecionada

# Descrição

Esta função, que foi projetada para funcionar com a função xset, **também está obsoleta**. Utilize o editor de propriedades ged ao invés.

`getmark` abre uma janela de gráficos para selecionar uma marca (símbolo).

# Exemplos

```
x=0:0.1:10;
[mark,mrkSize]=getmark();
plot2d(x,sin(x),style=-mark);
clf();
plot2d(x,sin(x))
e=gce(); // armazenando o Compound contendo o esboço
[mark,mrkSize]=getmark();
e.children(1).mark_style = mark;
```

# Ver Também

ged, set, segs_properties, segs_properties

# Name

getsymbol — diálogo para a seleção de um símbolo e seu tamanho. **Função obsoleta.**

```
c=getsymbol([title])
```

# Parâmetros

title
: string, título do diálogo

c
: vetor de tamanho 2 `[n,sz]`.

# Descrição

Esta função, que foi projetada para trabalhar com a função xset, **também está obsoleta**. Utilize o editor de propriedades ged ao invés.

`getsymbol` abre uma caixa de diálogo de escolha com título `title` se fornecido, onde o usuário pode selecionar um símbolo e seu tamanho. `getsymbol` retorna o identificador do símbolo `n` e o identificador de seu tamanho `sz`.

# Ver Também

ged, set, segs_properties, segs_properties

# Name

glue — Cola um conjunto de entidades gráficas em um Compound ("composto")

```
glue(H)
h_agreg=glue(H)
```

# Parâmetros

H
: um vetor de manipuladores

h_agreg
: manipulador da entidade Compound.

# Descrição

Dado um vetor de manipuladores, esta função cola as entidades correspondentes em um único Compound e retorna o manipulador desta nova entidade.

# Exemplos

```

```

# Ver Também

get, set, move, unglue, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

graduate — graduação de eixos

```
[xi,xa,np]=graduate( xmi, xma,n1,n2)
[xi,xa,np]=graduate( xmi, xma)
```

# Parâmetros

xmi,xma
: escalares reais

n1, n2
: inteiros com valores padrões 3,10

xi, xa
: escalares reais

np
: inteiro

# Descrição

graduate procura pelo intervalo mínimo `[xi,xa]` e um número de tiques `np` tais que:

`xi <= xmi <= xma <= xa`

`xa - xi / np = k(10^n)`,`k` em `[1 3 5]` para um inteiro `n`

`n1 < np < n2`

# Exemplos

```
y=(0:0.33:145.78)';
xbasc();plot2d1('enn',0,y)
[ymn,ymx,np]=graduate(mini(y),maxi(y))
rect=[1,ymn,prod(size(y)),ymx];
xbasc();plot2d1('enn',0,y,1,'011',' ',rect,[10,3,10,np])
```

# Ver Também

xsetech, plot2d

# Autor

S. Steer 1992;

# Name

graphics_entities — Descrição das estruturas de dados de entidades gráficas

# Descrição

No Scilab, uma janela de gráficos e os desenhos contidos nela são representados por entidades hierárquicas. O topo do nível hierárquico é a Figure (figura). cada Figure define pelo menos um galho do tipo Axes (eixos). Cada entidade Axes contém um conjunto de folhas que são objetos gráficos básicos como Polylines, Rectangles, Arcs, Segs,... (poligonais, retângulos, arcos e segmentos). Também pode ser do tipo Compound (composto) que são conjuntos recursivos de entidades. O interesse principal no novo modo de gráficos é facilitar a mudança de propriedades. O novo modo de gráficos fornece um conjunto de rotinas gráficas de alto-nível (ver set, get) utilizadas para controlar propriedades dos objetos como dados, cores, coordenadas e escalas, cores e aparências sem que seja necessário entrar novamente os comandos de gráficos iniciais.

Entidades gráficas do Scilab estão associadas a variáveis do tipo `handle`. (manipuladores). O manipulador é um identificador único que é associado a cada instância de uma entidade gráfica criada. Utilizando-se manipuladores, é possível tanger propriedades de entidades através das rotinas "set" e "get". Os manipuladores também são utilizados para manipular objetos gráficos, para movê-los, para fazer cópias ou deletá-los.

Figure:
: a entidade figura é o topo da hierarquia de entidades gráficas. Esta entidade define os parâmetros para a figura em si tanto quanto os valores padrões dos parâmetros para criação de galhos. Os galhos das figuras são as entidades de eixos.

  O manipulador da figura corrente (a figura para onde os desenhos foram enviados) pode ser obtido utilizando-se `get("current_figure")` e pode ser ajustado utilizando-se `set("current_figure",h)`, onde `h` pode ser tanto um manipulador da figura ou um idenficador da figura `figure_id` neste último caso, se não existir, um é automaticamente criado.

  Veja figure_properties para detalhes.

Axes:
: a entidade de eixos está no segundo nível da hierarquia de entidades gráficas. Esta entidade define parâmetros para a mudança de coordenadas dos eixos e o desenho dos eixos tanto quanto os parâmetros padrões para criação de seus galhos. Ver axes_properties for details. para detalhes. O manipulador dos eixos correntes pode ser obtido utilizando-se `get("current_axes")`.

Compound:
: a entidade composta é simplesmente um vetor de galhos com uma única propriedade (propriedade de visibilidade). É usada para colar um conjunto de entidades.

  Ver as documentações glue, unglue e Compound_properties.

Axis:
: a entidade eixo é uma folha da hierarquia de entidades gráficas. Esta entidade define os parâmetros de escala de um eixo e aparência.

  Ver axis_properties para detalhes.

Polyline:
: a entidade poligonal é uma folha na hierarquia de entidades gráficas. define propriedades de desenho de poligonais e extensões poligonais 2d e 3d.

  Ver polyline_properties para detalhes.

Arc:
: a entidade arco é uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros para elipses e partes de elipses.

Ver arc_properties para detalhes.

Rectangle:
a entidade retângulo é uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros para retângulos e retângulos preenchidos.

Ver rectangle_properties para detalhes.

Surface:
a entidade superfície é uma folha na hierarquia de entidades gráficas. possui sub-tipos `Fac3d` ou `Plot3d`. Esta entidade define parâmetros para esboço de superfícies 3d.

Ver surface_properties para detalhes.

Fec:
a entidade Fec é uma folha na hierarquia de entidades gráficas. Representa esboços de elementos 2d finitos.

Ver fec_properties para detalhes

Grayplot:
a entidade Grayplot é uma folha na hierarquia de entidades gráficas. Representa esboços 2d de superfícies utilizando cores e imagens.

Ver grayplot_properties para detalhes.

Matplot:
a entidade Matplot é uma folha na hierarquia de entidades gráficas. Representa esboços 2d utilizando matrizes de inteiros.

Ver Matplot_properties para detalhes.

Segs:
a entidade de segmentos é uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros um conjunto de flechas ou segmentos coloridos.

Ver segs_properties para detalhes

Champ:
a entidade Champ é uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros para esboços 2d de campos vetoriais.

Ver champ_properties para detalhes.

Text:
a entidade texto é uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros para impressão de strings.

Ver text_properties para detalhes.

Label:
a entidade rótulo é um galho da entidade gráfica `Axes` (de eixos). Esta entidade define parâmetros para os rótulos dos 3 eixos x, y e z desenhados em uma janela de gráficos.

Ver label_properties para detalhes.

Legend:
a legenda é uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros para legendas desenhadas impressas sob gráficos `plot2dx`. Esta entidade requer futuros desenvolvimentos.

Ver legend_properties para detalhes.

# Exemplos

```
//execute este exemplo linha por linha

scf() //criando figura no modo de entidade

//obtendo o manipulador da entidade Figure e exibindo suas propriedades
f=get("current_figure")
a=f.children // o manipulador da galho Axes
x=(1:10)';  plot2d(x,[x.^2 x.^1.5])
e=a.children //Compound de duas poligonais

p1=e.children(1) //as propriedades das últimas poligonais desenhadas

p1.foreground=5;  // mudando a cor da poligonal
e.children.thickness=5; // mudando a espessura das poligonais

delete(e.children(2))

move(e.children,[0,30]) //transladando a poligonal

a.axes_bounds=[0 0 0.5 0.5];

subplot(222) //criando uma nova entidade Axes
plot(1:10);
a1=f.children(1); //obtendo seu manipulador
copy(e.children,a1); //copiando a poligonal do primeiro esboço nos novos eixos
a1.data_bounds=[1 0;10 100]; //mudando os limites dos eixos

set("current_figure",10) //criando uma nova figura com figure_id=10
plot3d() //os desenhos são enviados à nova figura com figure_id=10
set("current_figure",f) //torna a figura anterior a corrente
plot2d(x,x^3)  //os desenhos são enviados à figura inicial
```

# Ver Também

set, get, move, draw, delete, object_editor, plot, surf

# Name

graphics_fonts — Descrição de fontes utilizadas em figuras gráficas

# Descrição

Algumas entidades gráficas como Text, Axes, Label ou Legend exibem um ou mais strings em suas figuras gráficas. A aparência dos strings exibidos pode ser modificada especificando diferentes fontes e tamanhos de caracteres.

Mudando a fonte

Fontes utilizadas em figuras gráficas são selecionadas de um conjunto de fontes chamado fontes carregadas. Em outras palavras, as fontes carregadas são aquelas correntemente disponíveis em figuras gráficas. Uma lista destas fontes pode ser obtida utilizando-se a função `xlfont` sem argumento. Por padrão, o Scilab possui um conjunto de 11 fontes carregadas. Este conjunto pode ser modificado e estendido utilizando-se a função `xlfont` com um nome de fonte como argumento. A fonte carregada pode ser de um arquivo ou mesmo uma uma do sistema. Para conhecer a lista de fontes disponíveis no sistema, utilize o comando `xlfont('AVAILABLE_FONTS')`. Para mais informações sobre como manipular fontes, veja xlfont.

Aqui está a lista das 11 fontes padrões

| **Número da fonte** | **Família de Fontes** | **Negrito** | **Itálico** |
|---|---|---|---|
| 0 | Monospaced | Não | Não |
| 1 | ScilabSymbols | Não | Não |
| 2 | Serif | Não | Não |
| 3 | Serif | Não | Sim |
| 4 | Serif | Sim | Não |
| 5 | Serif | Sim | Sim |
| 6 | SansSerif | Não | Não |
| 7 | SansSerif | Não | Sim |
| 8 | SansSerif | Sim | Não |
| 9 | SansSerif | Sim | Sim |
| 10 | SansSerif | Sim | Sim |

A fonte utilizada por uma entidade gráfica pode ser alterada através da propriedade `font_style`. Esta é um inteiro positivo fazendo referência a uma das fontes. Seu valor deve estar entre 0, fazendo referência à primeira fonte, e o número de fontes carregadas menos um, fazendo referência à última fonte.

A propriedade `fractional_font` controla o anti-serrilhamento da fonte. Seu valor pode ser `'on'` ou `'off'`. Se for `'on'`a fonte é suavizada, em caso contrário, ela não é.

Mudando o tamanho dos caracteres

O tamanho do texto de uma entidade gráfica é modificado através da propriedade `font_size`. É um escalar especificando o tamanho dos caracteres exibidos.

O tamanho do caractere do Scilab é diferente do tamanho do caractere Java. Aqui está uma equivalência entre as duas escalas.

| **Tamanho Scilab** | **Tamanho Java** |
|---|---|
| 0 | 8 |

| | |
|---|---|
| 1 | 10 |
| 2 | 12 |
| 3 | 14 |
| 4 | 18 |
| 5 | 24 |
| 6 | 30 |
| 7 | 36 |
| 8 | 42 |
| 9 | 48 |
| 10 | 54 |

O tamanho do caractere pode não ser um inteiro. Neste caso, o resultado depende da propriedade `fractional_font` da entidade. Se a propriedade `fractional_font` for `'on'`, então o tamanho da fonte é interpolado entre os dois inteiros mais próximos. Por exemplo, um tamanho de fonte 2.5 exibe caracteres com tamanho Java 13. Se a propriedade `fractional_font` for `'off'`, então a fonte exibida é truncada para a sua parte inteira. Por exemplo, um tamanho de fonte 2.5 exibe caracteres Java de tamanho 12.

# Ver Também

xlfont, graphics_entities

# Name

graycolormap — mapa de cores linear cinza

```
cmap=graycolormap(n)
```

# Parâmetros

n
: inteiro >= 1, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`graycolormap` computa um mapa de cores com `n` cores cinzas variando linearmente do preto para o branco.

# Exemplos

```
f = scf();
plot3d1();
f.color_map = graycolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap, xset

# Name

grayplot — esboço 2d de uma superfície utilizando-se cores

```
grayplot(x,y,z,[strf,rect,nax])
grayplot(x,y,z,<opt_args>)
```

# Parâmetros

x,y
: vetores de reais de tamanhos n1 e n2. .

z
: matriz de reais de tamanho (n1,n2). `z(i,j)` é o valor da superfície no ponto(x(i),y(j)).

<opt_args>
: representa uma seqüência de declarações `key1=value1, key2=value2`,... onde `key1`, `key2,...` podem ser um dos seguintes: rect, nax, strf ou axesflag e frameflag (ver `plot2d` e `plot2d_old_version`).

strf,rect,nax
: ver `plot2d`.

# Descrição

`grayplot` faz um esboço 2d de uma superfície dada por `z` em um grid definido por `x` e `y`. Cada retângulo no grid é preenchido com um nível de cinza ou de cor dependendo do valor médio de `z` nas quinas do retângulo.

Entre com o comando `grayplot()` para visualizar uma demonstração.

# Exemplos

```
x=-10:10; y=-10:10;m =rand(21,21);
grayplot(x,y,m,rect=[-20,-20,20,20])
t=-%pi:0.1:%pi; m=sin(t)'*cos(t);
clf()
grayplot(t,t,m)
```

# Ver Também

fgrayplot, plot2d, Sgrayplot, Sfgrayplot

# Autor

J.Ph.C.

# Name

grayplot_properties — description of the grayplot entities properties

# Descrição

A entidade Grayplot é uma folha na hierarquia de entidades gráficas. Representa esboços 2d de superfícies utilizando cores e imagens (ver `grayplot`, `Sgrayplot`, `fgrayplot` e `Sfgrayplot`).

parent:
esta propriedade contém o manipulador para a raiz. A raiz de uma entidade grayplot deve ser do tipo `"Axes"`.

children:
esta propriedade contém um vetor com os galhos do manipulador. Contudo, manipuladores grayplot não possuem galhos correntemente.

visible:
este campo contém o valor da propriedade `visible` para a entidade. Pode ser `"on"` ou `"off"` . Por padrão, o grayplot é visível, a propriedade é `"on"`. Se `"off"` o grayplot não é exibido na tela.

data:
este campo define uma estrutura de dados do tipo `tlist` de tipo `"grayplotdata"` composto de uma linha e índices de colunas de cada elemento: as coordenadas do grid x e y estão contidas respectivamente em `data.x` e `data.y`. O campo complementar chamado `data.z` é o valor da superfície no ponto `(x(i),y(j))` (modo escalado) ou o valor centrado da superfície definido entre dois pontos consecutivos `x(i)` e `y(j)` (modo direto). Se `data_mapping` (ver abaixo) for ajustado para `"scaled"`, todo o dado z é utilizado para realizar uma interpolação de cores, enquanto que, se o valor de `data_mapping` for `"direct"`, a última linha e coluna dos índices de dados z são ignoradas e a cor é determinada diretamente no mapa de cores através dos índices da submatriz `data.z(1:$-1,1:$-1)`.

data_mapping:
por padrão, o valor desta propriedade é `"scaled"` : os índices das cores utilizadas para colorir são proporcionais ao valor das coordenadas `z`. Em outro caso, a propriedade possui valor `"direct"` onde o esboço é um grayplot e os índices das cores de pintura são fornecidos pelos dados (ver acima).

clip_state:
este campo contém o valor da propriedade `clip_state` para o grayplot. o valor de clip_state pode ser :

- `"off"` significa que o grayplot não é recortado.
- `"clipgrf"` significa que o grayplot é recortado fora da caixa dos eixos.
- `"on"` significa que o grayplot é recortado fora do retângulo dado pela propriedade clip_box.

clip_box:
este campo determina a propriedade `clip_box` . Por padrão seu valor é uma matriz vazia se a propriedade clip_state é "off". Em outros casos, o vetor `[x,y,w,h]` (ponto superior esquerdo, largura, altura) define as porções do retângulo a ser exibido, contudo o valor da propriedade `clip_state` será alterado.

user_data:
este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados da entidade grayplot e recuperá-la.

# Exemplos

```
m=5;n=5;
M=round(32*rand(m,n));
grayplot(1:m,1:n,M)

a=get("current_axes");
a.data_bounds=  [-1,-1;7,7]
h=a.children

h.data_mapping="direct";

// um esboço 2D de uma matriz utilizando cores
xbasc()
a=get("current_axes");
a.data_bounds=  [0,0;4,4];

b=5*ones(11,11); b(2:10,2:10)=4; b(5:7,5:7)=2;
Matplot1(b,[1,1,3,3]);

h=a.children
for i=1:7
  xclick(); // um click ajusta os dados do Matplot
  h.data=h.data+4;
end
```

# Ver Também

set, get, delete, grayplot, Matplot, Matplot1, graphics_entities, Matplot_properties

# Autores

Djalel ABDEMOUCHE & F.Leray

# Name

graypolarplot — Esboço polar 2d de uma superfície utilizando-se cores

```
graypolarplot(theta,rho,z,[strf,rect])
```

# Parâmetros

theta
: vetor de tamanho n1, a discretização do ângulo em radianos.

rho
: vetor de tamanho n2, a discretização do raio

z
: matriz de reais de tamanho (n1,n2). `z(i,j)` é o valor da superfície no ponto (theta(i),rho(j)).

strf
: string de comprimento 3 `"xy0"`.

  default
  : o padrão é `"030"`.

  x
  : controla a exibição de legendas.

    x=0
    : sem legendas.

    x=1
    : com legendas. Elas são fornecidas pelo argumento opcional `leg`.

  y
  : controla a computação da moldura.

    y=0
    : as molduras correntes (definidas por uma chamada anterior a uma função de esboço de alto-nível) são utilizadas. Útil ao se superpor esboços múltiplos.

    y=1
    : o argumento opcional `rect` é utilizado para se especificar as fronteiras do esboço.

    y=2
    : as fronteiras do esboço são computadas utilizando-se os valores mínimo e máximo de `x` e `y`.

    y=3
    : como `y=1` mas produz escala de isovisualização.

    y=4
    : como `y=2` mas produz escala de isovisualização.

    y=5
    : como `y=1` mas `plot2d` pode mudar as fronteiras do esboço e os tiques dos eixos para se obter boas graduações. Quando o botão de zoom é ativado, este modo é usado.

    y=6
    : como `y=2` mas `plot2d` pode mudar as fronteiras do esboço e os tiques dos eixos para se obter boas graduações. Quando o botão de zoom é ativado, este modo é usado.

y=7
: como `y=5` mas a escala do novo esboço é misturada à escala corrente.

y=8
: como `y=6` mas a escala do novo esboço é misturada à escala corrente.

leg
: string. É usado quando o primeiro caractere x do argumento `strf` é 1. `leg` possui a forma `"leg1@leg2@...."` onde `leg1`, `leg2`, etc. são respectivamente as legendas das primeira, segunda, etc. curvas. O padrão é " ".

rect
: É usado quando o segundo caractere y do argumento `strf` é 1, 3 ou 5. É um vetor linha de tamanho 4 e fornece as dimensões da moldura: `rect=[xmin,ymin,xmax,ymax]`.

# Descrição

Toma um esboço 2d de uma superfície dada por z em um grid de coordenadas polares definido por `rho` e `theta`. Cada região é preenchida com um nível de cinza ou de cor dependendo do valor médio de `z` nas quinas do grid.

# Exemplos

```
rho=1:0.1:4;theta=(0:0.02:1)*2*%pi;
z=30+round(theta'*(1+rho^2));
f=gcf();
f.color_map= hotcolormap(128);
clf();graypolarplot(theta,rho,z)
```

# Name

havewindow — Retorna o modo de janela do Scilab

```
havewindow()
```

# Descrição

Retorna %t se o Scilab possui a sua própria janela e %f se não, i.e. se o Scilab foi chamado com "scilab -nw". (`nw` significa "sem janela").

# Name

hist3d — representação 3d de um histograma

```
hist3d(f,[theta,alpha,leg,flag,ebox])
hist3d(list(f,x,y),[theta,alpha,leg,flag,ebox])
```

# Parâmetros

mtx
: matriz de tamanho (m,n) definindo o histograma `mtx(i,j)=F(x(i),y(j))`, onde `x` e `y` asão tomados como `0:m` e `0:n`.

list(mtx,x,y)
: onde f é uma matriz de tamanho (m,n) definindo o histograma `mtx(i,j)=F(x(i),y(j))`, com vetores `x` e `y` de tamanho (1,n+1) e (1,m+1).

theta,alpha,leg,flag,ebox
: ver `plot3d`.

# Descrição

`hist3d` representa um histograma 2d como um esboço 3d. Os valores são associados aos intervalos `[x(i) x(i+1)[ X [y(i) y(i+1)[`.

Entre com o comando `hist3d()` para visualizar uma demonstração.

# Exemplos

```
hist3d(10*rand(10,10));

Z = zeros(100,5);
A = abs(rand(40,5));
Z(1:40,:) = A;
scf();
hist3d(Z);

Index = find(Z==0);
Z(Index) = %nan;
scf();
hist3d(Z);

A = abs(rand(10,5));
Z(91:100,:) = A;
scf();
hist3d(Z);
```

# Ver Também

histplot, plot3d

# Autores

Steer S. & JPhilippe C.

# Name

histplot — esboça um histograma

```
histplot(n, data, <opt_args>)
histplot(x, data, <opt_args>)
```

# Parâmetros

n
: inteiro positivo (número de classes)

x
: vetor crescente definindo as classes (`x` pode ter pelo menos dois componentes)

data
: vetor (dados a serem analisados)

<opt_args>
: representa uma seqüência de declarações `key1=value1,key2=value2` ,... onde `key1`, `key2,...` pode ser qualquer normalização ou parâmetro de plot2d opcional (`style,strf,leg, rect,nax, logflag,frameflag, axesflag`) No caso de normalização, o valor correspondente deve ser um escalar booleano (valor padrão %t).

# Descrição

Esta função esboça um histograma do vetor `data` utilizando classes `x`. Quando o número `n` de classes é fornecido ao invés de `x`, as classes são escolhidas de modo igualmente espaçado e *x(1) = min(data) < x(2) = x(1) + dx < ... < x(n+1) = max(data)* com *dx = (x(n+1)-x(1))/n*.

As classes são definidas por C1 = [x(1), x(2)] e Ci = ( x(i), x(i+1)] para i >= 2. Notando Nmax o número total de `data` (Nmax = comprimento de data) e Ni o número de componentes de `data` em Ci, o valor do histograma para x em Ci é igual a *Ni/(Nmax (x(i+1)-x(i)))* quando `normalization` (normalização) for verdadeiro (caso padrão) senão, é simplesmente igual a Ni. Quando a normalização ocorre, o histograma verifica:

```
 x(n+1)
/
|   h(x) dx = 1,  when x(1)<=min(data) e max(data) <= x(n+1))
/
 x(1)
```

Qualquer plot2d parâmetro (opcional) pode ser fornecido; por exemplo, para esboçar um histograma com a cor número 2 (azul, se o mapa de cores padrão for utilizado) e para restringir o esboço ao retângulo [-3,3]x[0,0.5], você pode utilizar `histplot(n,data, style=2, rect=[-3,0,3,0.5])`.

Entre com o comando `histplot()` para visualizar uma demonstração.

# Exemplos

```
// exemplo #1: variações ao redor de um histograma de uma amostra gaussiana ale
d=rand(1,10000,'normal');  // a amostra gaussiana aleatória
clf();histplot(20,d)
clf();histplot(20,d,normalization=%f)
```

```
clf();histplot(20,d,leg='rand(1,10000,''normal'')',style=5)
clf();histplot(20,d,leg='rand(1,10000,''normal'')',style=16, rect=[-3,0,3,0.5])

// exemplo #2: histograma de uma amsostra binomial (B(6,0.5)) aleatória
d = grand(1000,1,"bin", 6, 0.5);
c = linspace(-0.5,6.5,8);
xbasc()
subplot(2,1,1)
   histplot(c, d, style=2)
   xtitle("histograma normalizado")
subplot(2,1,2)
   histplot(c, d, normalization=%f, style=5)
   xtitle("histograma não normalizado")

// exemplo #3: histograma de uma amostra exponencial aleatória
lambda = 2;
X = grand(100000,1,"exp", 1/lambda);
Xmax = max(X);
xbasc()
histplot(40, X, style=2)
x = linspace(0,max(Xmax),100)';
plot2d(x,lambda*exp(-lambda*x),strf="000",style=5)
legend(["histograma de amostra exponencial aleatória" "curva de densidade exata
```

# Ver Também

hist3d, plot2d, dsearch

# Name

hotcolormap — mapa de cores de vermelho para amarelo

```
cmap=hotcolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`hotcolormap` computa um mapa de cores com `n` cores quentes que variam do vermelho para o amarelo.

# Exemplos

```
f = scf();
plot3d1();
f.color_map = hotcolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap

# Name

hsv2rgb — Converte cores HSV para RGB

```
[r,g,b] = hsv2rgb(h,s,v)
rgb = hsv2rgb(h,s,v)
[r,g,b] = hsv2rgb(hsv)
rgb = hsv2rgb(hsv)
```

# Parâmetros

h
: um vetor de tamanho n. Os valores "hue" (de matiz)

s
: um vetor de tamanho n. Os valores "saturation" (de saturação)

v
: um vetor de tamanho n. Os valores "value" (de valor).

hsv
: uma matriz n x 3. Cada linha contém uma tripla [hue saturation value].

r
: um vetor coluna de tamanho n. Os valores "red" (de vermelho).

g
: um vetor coluna de tamanho n. Os valores "green" (de verde).

b
: um vetor coluna de tamanho n. Os valores "blue" (de azul).

rgb
: uma matriz n x 3. Cada linha contém uma tripla [red green blue].

# Descrição

A função hsv2rgb converte mapas de cores entre os espaços de cores RGB e HSV. Como a matiz varia de 0 a 1.0 , as cores correspondentes podem variar partindo do vermelho entre amarelo, verde, ciano, azul, magenta, e preto, até vermelho novamente, de modo que há na verdade valores de vermelho tanto em 0 quanto em 1.0 . À medida em que a saturação varia de 0 a 1.0, as cores correspondentes (matizes) variam de insaturadas (gradações de cinza) a completamente saturadas (nenhum componente branco). à medida em que o valor, ou brilho, varia de 0 a 1.0, as cores correspondentes vão se tornando mais brilhantes.

# Exemplos

```
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
plot3d1(t,t,z)
f=gcf();f.pixmap='on';
for h=0:0.1:1
  hsv=[h*ones(32,1) linspace(0,1,32)' 0.7*ones(32,1)];
  f.color_map=hsv2rgb(hsv);
  show_pixmap()
  xpause(10000)
end
for v=0:0.1:1
```

```
  hsv=[ones(32,1) linspace(0,1,32)' v*ones(32,1)];
  f.color_map=hsv2rgb(hsv);
  show_pixmap()
  xpause(10000)
end
```

# Autor

Serge Steer
INRIA

# Name

hsvcolormap — Mapa de cores de matiz-saturação-valor (hue-saturation-value)

```
cmap = hsvcolormap(n)
```

# Parâmetros

n
: inteiro >= 1, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`hsvcolormap` computa um mapa de cores com `n` cores. Este mapa de cores varia o componente de matiz do modelo de cor hsv (hue-saturation-value, "matiz-saturação-valor "). As cores começam do vermelho, passando por amarelo, verde, azul celeste, azul, magenta, e retornam para o vermelho. O mapa é particularmente útil para exibir funções periódicas.

# Exemplos

```
t=[0:0.1:2*%pi]'; z=sin(t)*cos(t');
f=gcf();f.color_map=hsvcolormap(64);
plot3d1(t,t,z,35,45,"X@Y@Z",[-2,2,2])
```

# Autor

Serge Steer
: INRIA

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap

# Name

is_handle_valid — Verifica se um conjunto de manipuladores gráficos ainda é válido

```
isValid = is_handle_valid(h)
```

# Parâmetros

h
: matriz de manipuladores gráficos

isValid
: matriz de booleanos com o mesmo tamanho que `h`

# Descriçãp

`is_handle_valid` testa se um conjunto de manipuladores ainda é válido. Um manipulador válido é um que ainda não foi deletado. O resultado, `isValid`, é uma matriz de booleanos tal que `isValid(i,j)` é verdadeiro se `h(i,j)` é válido e falso em caso contrário.

# Exemplos

```
// verificando se os objetos correntes são válidos
is_handle_valid([gcf(), gca(), gce()])

// criando 11 poligonais
plot([0:10; 0:10; 0:10], [0:10; 0:0.5:5; 0:2:20]);

// verificando a validade das poligonais
axes = gca();
polylines = axes.children(1).children
is_handle_valid(polylines)

// deletando algumas poligonais
delete(polylines(3:7));
// imprimindo a validade
is_handle_valid(polylines)
```

# Ver Também

delete, graphics_entities

# Autor

Jean-Baptiste Silvy INRIA

# Name

isoview — ajusta escalas para esboço isométrico (não muda o tamanho da janela)

```
isoview(xmin,xmax,ymin,ymax)
```

# Parâmetros

xmin,xmax,ymin,ymax
: quatro valores reais

# Descrição

Esta função está obsoleta, use a opção de plot2d `frameflag=4` que permite redimensionamento da janela, ao invés.

`isoview` é usado para obter escalas isométricas nos eixos x e y. O tamanho da janela de gráficos não é alterado. O retângulo `xmin, xmax, ymin, ymax` será contido no quadro computado da janela de gráficos. `isoview` ajusta as escalas gráficas correntes e pode ser usado em conjunção com rotinas que requerem escalas gráficas correntes (por exemplo `strf="x0z"` em `plot2d`).

# Exemplos

```
t=[0:0.1:2*%pi]';
plot2d(sin(t),cos(t))
xbasc()
isoview(-1,1,-1,1)
plot2d(sin(t),cos(t),1,"001")
xset("default")

plot2d(sin(t),cos(t),frameflag=4)
```

# Ver Também

square, xsetech

# Autor

Steer S.

# Name

jetcolormap — Mapa de cores de gradação do azul para o vermelho

```
cmap=jetcolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com três colunas `[R,G,B]`.

# Descrição

`jetcolormap` computa o mapa de cores com `n` cores variando do azul, passando por azul claro, verde, amarelo, laranja e então vermelho.

# Exemplos

```
f = scf();
plot3d1();
f.color_map = jetcolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap

# Name

label_properties — Descrição de propriedades da entidade Label (rótulo)

# Descrição

A entidade Label é um galho da entidade `Axes` (de eixos). Quando uma entidade Axes é construída, os manipuladores das entidades Title e Label são criados juntos e são parte das propriedade da entidade Axes. Logo, as propriedades destes sub-objetos são editáveis via o manipulador da entidade Axes (ver `gca` e `gda`). Esta entidade define os parâmetros para impressão de rótulos:

parent:
: esta propriedade contém o manipulador para a raiz. A raiz de uma entidade Label deve ser do tipo `"Axes"` .

  Note que, por ora, a entidade Label é exclusivamente utilizada na construção de `title`, `x_label`, `y_label` e `z_label`.

visible:
: este campo contém o valor da propriedade `visible` para a entidade. Pode ser `"on"` ou `"off"` . Por padrão, o rótulo é visível, o valor da propriedade é `"on"`. Se `"off"` o rótulo não é impresso na tela.

text:
: a matriz contendo os strings do objeto. As linhas da matriz são exibidas horizontalmente e as colunas verticalmente.

font_foreground:
: este campo contém o índice de cor utilizado para exibir o rótulo `text`. Seu valor deve ser um índice de cor (relativo ao mapa de cores corrente).

foreground:
: este campo contém a cor utilizada para exibir o rótulo e a linha ao redor da caixa, se houver. Deve ser um índice de cor (relativo ao mapa de cores corrente).

background:
: este campo contém a cor a ser usada para preencher a caixa, se houver. Deve ser um índice de cor (relativo ao mapa de cores corrente).

fill_mode:
: este campo toma valores `"on"` ou `"off"`. Se `"on"` uma caixa é desenhada ao redor do texto com uma linha em sua borda e um plano de fundo.

font_style:
: especifica a fonte a ser utilizada para exibir o rótulo. É um inteiro positivo fazendo referência uma das fontes carregadas. Seu valor deve estar entre 0, fazendo referência à primeira fonte, e o número de fontes carregadas menos um, fazendo referência à última fonte. Para mais informações, veja graphics_fonts.

font_size:
: é um escalar especificando o tamanho dos caracteres exibidos. Se a propriedade `fractional_font` for `"off"`, apenas a parte inteira é utilizada. Para mais informações, veja graphics_fonts.

fractional_font:
: esta propriedade especifica se o texto será utilizado utilizando-se tamanhos de fonte fracionários. Seu valor deve ser `"on"` ou `"off"`. Se `"on"`, o valor em ponto flutuante de `font_size` é utilizado para a exibição e evita-se o serrilhamento da fonte. Se `"off"`, apenas a parte inteira é utilizada e a fonte não é suavizada.

font_angle:
: um escalar que permite girar o rótulo. A fonte é girada em sentido horário em um ângulo dado em graus. Note que mudar `font_angle` desabilitará automaticamente a propriedade `auto_rotation`.

auto_rotation:
: se `"on"`, o Scilab computa automaticamente o melhor ângulo de rotação para a exibição do rótulo. Se `"off"`, o rótulo pode ser girado manualmente através da propriedade `font_angle`.

position:
: este vetor 2d permite posicionar o rótulo na tela. A posição é armazenada em unidades de dados dos eixos. Note que mudar `font_angle` desabilitará automaticamente a propriedade `auto_position`.

auto_position:
: se `"on"`, o Scilab computa automticamente a melhor posição para a exibição do rótulo na janela de gráficos. Se `"off"`, o rótulo pode ser posicionado manualmente através da propriedade `position`.

# Exemplos

```
a=get("current_axes");
a.title
type(a.title)
plot3d()
a.x_label
a.y_label
a.z_label
xtitle("Meu título","meu rótulo de eixo x", "Volume","Mês")

t=a.title;
t.foreground=9;
t.font_size=5;
t.font_style=5;
t.text="SCILAB";

x_label=a.x_label;
x_label.text=" Peso"
x_label.font_style= 5;
a.y_label.foreground = 12;
```

# Ver Também

set, get, delete, xtitle, graphics_entities, axes_properties, text_properties

# Autor

Djalel ABDEMOUCHE

# Name

legend — Imprime uma legenda para o gráfico

```
hl=legend([h,] string1,string2, ... [,pos] [,boxed])
hl=legend([h,] strings [,pos] [,boxed])
```

# Parâmetros

h
: manipulador gráfico de uma entidade de eixos ou vetor de manipuladores de entidades poligonais. O valor padrão é o manipulador dos eixos correntes.

string1,string2, ...
: strings, stringsi é a legenda da i-ésima curva

strings
: vetor de strings de dimensão n, strings(i) é a legenda da i-ésima curva

pos
: (opcional) especifica onde imprimir as legendas; este parâmetro pode ser um flag inteiro (ou, de modo equivalente, um string) ou um vetor [x,y] que fornece as coordenadas da quina superior esquerda da caixa de legenda. No primeiro caso, os valores possíveis são:

  1
  : as legendas são impressas no canto superior direito

  2
  : as legendas são impressas no canto superior esquerdo

  3
  : as legendas são impressas no canto inferior esquerdo

  4
  : as legendas são impressas no canto inferior direito

  5
  : localização interativa com o mouse

  -1
  : as legendas são impressas à direita do canto superior direito

  -2
  : as legendas são impressas à esquerda do canto superior esquerdo

  -3
  : as legendas são impressas à esquerda do canto inferior esquerdo

  -4
  : as legendas são impressas à direita do canto inferior direito

  -5
  : as legendas são impressas acima do canto superior esquerdo

  -6
  : as legendas são impressas abaixo do canto inferior esquerdo

boxed
: um booleano (valor padrão %t) que ajusta se a caixa será desenhada ou não

hl
: um manipulador, pontos no Compound contendo todas as legendas.

# Descrição

Põe uma legenda no esboço corrente utilizando os strings especificados como rótulos. legend pre-anexa rótulos por uma re-chamada à linha correspondente ou remendo. A re-chamada é de tipo e propriedades são recobradas dos dados manipuladores:

Quando chamada sem o argumento manipulador (ou com manipulador de uma entidade Axes) a função procura vetores de manipuladores de entidades poligonais construídos primeiro que são galhos dos eixos dados.

Na localização interativa (opt= 5) move-se a caixa de legenda com o mouse e aperta-se o botão esquerdo para liberá-la.

# Exemplos

```
t=linspace(0,%pi,20);
a=gca();a.data_bounds=[t(1) -1.8;t($) 1.8];

plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-5,2 3]);
e=gce();
e1=e.children(1);e1.thickness=2;e1.polyline_style=4;e1.arrow_size_factor = 1/2;
e.children(2).line_style=4;
e3=e.children(3);e3.line_mode='on';e3.mark_background=5;

hl=legend(['cos(t)';'cos(2*t)';'cos(3*t)']);
```

# Ver Também

plot2d, xstring, captions, polyline_properties

# Name

legend_properties — Descrição das propriedades da entidade Legend (legenda)

# Descrição

A entidade Legend é uma folha na hierarquia de entidade gráficas. Esta entidade define parâmetros para legendas impressas abaixo de gráficos `plot2dx` ou criadas pela função captions. Para cada linha esboçada, a legenda exibe uma amostra do tipo da linha, de sua marca e de sua cor.

parent:
: esta propriedade contém o manipulador da raiz. A raiz de uma entidade Legends deve ser do tipo `"Compound"`. Este Compound contém também o restante das entidade gráficas.

children:
: esta propriedade contém um vetor com os galhos do manipulador. contudo, manipuladores `legend` não possuem galhos correntemente.

visible:
: este campo contém o valor da propriedade `visible` para a entidade . Pode ser `"on"` ou `"off"` . Se `"on"` a legenda é impressa, se `"off"`, a legenda não é exibida na tela.

text:
: este campo é um vetor de strings que contém as legendas para cada objeto anotado.

font_size:
: um escalar especificando o tamanho da fonte. Se a propriedade `fractional_font` for `"off"`, apenas a parte inteira é utilizada. Para mais informações veja graphics_fonts.

font_style:
: especifica a fonte utilizada para exibir os rótulos da legenda. É um inteiro positivo fazendo referência a uma das fontes carregadas. Seu valor deve estar entre 0, fazendo referência à primeira fonte, e o número de fontes caregadas menos um, fazendo referência à última fonte. Para mais informações veja graphics_fonts.

font_color
: um índice de cor, esta propriedade determina a cor do texto.

fractional_font:
: esta propriedade especifica se os textos são exibidos utilizando-se tamanhos de fonte fracionários. Seu valor deve ser `"on"` ou `"off"`. Se `"on"`, o valor em ponto flutuante de `font_size` é utilizado para a exibição e evita-se o serrilhamento da fonte. Se `"off"`, apenas a parte inteira é utilizada e a fonte não é suavizada.

links:
: um array linha de manipuladores. Eles se referem às poligonais associadas.

legend_location
: um string, especifica a localização da legenda

- `"in_upper_right"` : as legendas são exibidas no canto superior direito da caixa de eixos.
- `"in_upper_left"`: as legendas são exibidas no canto superior esquerdo da caixa de eixos.
- `"in_lower_right"`: as legendas são exibidas no canto inferior direito da caixa de eixos.
- `"in_lower_left"`: as legendas são exibidas no canto inferior esquerdo da caixa de eixos.
- `"out_upper_right"`: as legendas são exibidas à direita do canto superior direito da caixa de eixos.

- `"out_upper_left"`: as legendas são exibidas à esquerda do canto superior esquerdo da caixa de eixos.
- `"out_lower_right"`: as legendas são exibidas à direita do canto inferior direito da caixa de eixos.
- `"out_lower_left"`: as legendas são exibidas à esquerda do canto inferior esquerdo da caixa de eixos.
- `"upper_caption"`: as legendas são exibidas acima do canto superior esquerdo da caixa de eixos.
- `"lower_caption"`: as legendas são exibidas abaixo do canto inferior esquerdo da caixa de eixos. Esta opção corresponde ao argumento `leg` de plot2d
- `"by_coordinates"`: o canto superior esquerdo da caixa de legendas é fornecido pelo campo "position" da estrutura de dados associada. As posições `x` e `y` são fornecidas como frações dos `axes_bounds`

position
: as coordenadas do canto superior esquerdo da legenda. As posições `x` e `y` são dadas como frações dos tamanhos `axes_bounds`. Este campo pode ser ajustado se `legend_location=="by_coordinates" ou obtido de outros ajustes de legend_location.`

line_mode
: este campo especifica se um retângulo é desenhado ao redor da legenda ou não. Deve ser `"on"` ou `"off"`. Se `"on"` , o retângulo é desenhado utilizando-se as seguintes propriedades.

thickness
: este campo fornece a espessura da linha utilizada para desenhar o retângulo.

foreground
: este campo contém o índice de cor da linha utilizada para desenhar o contorno do retângulo.

fill_mode
: este campo especifica se o plano de fundo da legenda será pintado ou não. Deve ser `"on"` ou `"off"`. Se `"on"` , o plano de fundo é pintado utilizando-se a cor definida no campo `background`.

background
: este campo contém o índice de cor da linha utilizada para pintar a área do retângulo.

clip_state:
: este campo contém o valor da propriedade `clip_state` para todos os objetos. O valor de clip_state pode ser :

- `"off"` significa que todos os objetos criados posteriormente não são recortados. (valor padrão).
- `"clipgrf"` significa que todos os objetos criados posteriormente são recortados fora da caixa dos eixos.
- `"on"` significa que todos os objetos criados posteriormente são recortados fora do retângulo dado pela propriedade clip_box.

clip_box:
: este campo contém o valor padrão da propriedade `clip_box` para todos os objetos. O seu valor é uma matriz vazia se clip_state é `"off"`. Em outros casos, o recorte é dado por `[x,y,w,h]` (ponto superior esquerdo, largura, altura).

user_data:
este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados do texto e recuperá-la.

# Exemplos

```
// inicialização de x
x=[0:0.1:2*%pi]';
plot2d(x,[sin(x) sin(2*x) sin(3*x)],[1,2,3],leg="L1@L2@L3")
a=get("current_axes");
l=a.children(2);
l.links
l.text=["sin(x)";"sin(2*x)";"sin(3*x)"];
l.visible="off"; // invisível
l.font_size = 2;
l.font_style = 5;
l.visible='on';
```

# Ver Também

plot2d, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

legends — Imprime uma legenda para o gráfico

```
legends(strings,style,<opt_args>)
```

# Parâmetros

strings
: vetor de strings de dimensão n, strings(i) é a legenda da i-ésima curva

style
: vetor linha de inteiros de dimensão n (os estilos de esboço, terceiro parâmetro de plot2d) ou uma matriz 2 x n, style(1,k) contém o estilo de esboço para a k-ésima curva e style(2,k) contém o estilo de linha (se style(1,k)>0) ou cor da marca (se style(1,k)<0).

<opt_args>
: representa uma seqüência de declarações `key1=value1,key2=value2`,... onde `key1`, `key2,...` podem ser um dos seguintes:

opt
: (opcional) especifica onde imprimir as legendas; este parâmetro pode ser um flag inteiro (ou, de modo equivalente, um string) ou um vetor [x,y] que fornece as coordenadas da quina superior esquerda da caixa de legenda. No primeiro caso, os valores possíveis são:

1 ou "ur"
: as legendas são impressas no canto superior direito.

2 ou "ul"
: as legendas são impressas no canto superior esquerdo.

3 ou "ll"
: as legendas são impressas no canto inferior esquerdo.

4 ou "lr"
: as legendas são impressas no canto inferior direito.

5 ou "?"
: localização interativa com o mouse (padrão).

6 ou "below"
: as legendas são impressas abaixo do gráfico (que é redimensionado de acordo).

with_box
: um booleano (valor padrão %t) que ajusta se a caixa será desenhada ou não.

font_size
: um inteiro (valor padrão 1) que ajusta o tamanho da fonte para os nomes na legenda.

# Descrição

Põe uma legenda no esboço corrente utilizando strings especificados como rótulos.

Na localização interativa (opt=5 ou opt="?") move-se a caixa de legenda com o mouse e aperta-se o botão esquerdo para liberá-la.

Esta função permite uma localização mais flexível de legendas que o argumento de plot2d leg.

# Exemplos

```
// Exemplo 1
t=0:0.1:2*%pi;
plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-1,2 3]);
legends(['cos(t)';'cos(2*t)';'cos(3*t)'],[-1,2 3],opt="lr")

scf() ;
xset("line style",2);plot2d(t,cos(t),style=5);
xset("line style",4);plot2d(t,sin(t),style=3);
legends(["sin(t)";"cos(t)"],[[5;2],[3;4]], with_box=%f, opt="?")

// Exemplo 2
scf() ;
subplot(221)
t=0:0.1:2*%pi;
plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-1,2 3]);
legends(['cos(t)';'cos(2*t)';'cos(3*t)'],[-1,2 3], opt=3 )

subplot(222)
xset("line style",2);plot2d(t,cos(t),style=5);
xset("line style",4);plot2d(t,sin(t),style=3);
legends(["sin(t)";"cos(t)"],[[5;2],[3;4]], with_box=%f, opt=6 )

subplot(223)
xset("line style",2);plot2d(t,cos(t),style=5);
xset("line style",4);plot2d(t,sin(t),style=3);
legends(["sin(t)";"cos(t)"],[[5;2],[3;4]], with_box=%f, opt=1, font_size=2 )

subplot(224)
t=0:0.1:2*%pi;
plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-1,2 3]);
legends(['cos(t)';'cos(2*t)';'cos(3*t)'],[-1,2 3], opt=2, font_size=1 )
```

# Ver Também

plot2d, xstring, xtitle, legend

# Name

locate — seleção pelo mouse de um conjunto de pontos

```
x=locate([n,flag])
```

# Parâmetros

x
: matriz de tamanho (2,n1). n1=n se o parâmetro n for dado.

n,flag
: valores inteiros

# Descrição

`locate` é usado para obter as coordenadas de um ou mais pontos selecionados com o mouse em uma janela de gráficos. As coordenadas são fornecidas usando-se a escala de gráficos corrente.

Se n>0, n pontos são selecionados e suas coordenadas são retornadas na matriz `x`.

Se n<=0, os pontos são selecionados até que o usuário clique com o botão esquerdo do mouse, que significa "parar". O último ponto (clicado com o botão esquerdo) não é retornado.

x=locate() é o mesmo que `x=locate(-1)`.

Se `flag=1` uma cruz é desenhada nos pontos onde o mouse foi clicado.

# Ver Também

xclick, xgetmouse

# Autores

S.S. & J.Ph.C

# Name

mesh — Esboço 3d de uma malha

```
mesh(Z)
mesh(X,Y,Z)
mesh(...,<GlobalProperty>)
mesh(...,<color>,<GlobalProperty>)
mesh(<axes_handle>,...)
```

# Parâmetros

Z
: matriz de reais definindo a altura da superfície. Não pode ser omitida. O dado Z é uma matriz `mxn`.

X,Y
: duas matrizes de reais: sempre ajustadas juntas, esses dados definem um novo grid padrão. Estes novos componentes `X` e `Y` do grid devem coincidir com as dimensões de `Z` (ver descrição abaixo).

color
: uma matriz de reais opcional definindo a cor para cada ponto `(X(j),Y(i))` do grid (ver descrição abaixo).

<GlobalProperty>
: este argumento opcional representa uma seqüência de pares de declarações `{PropertyName,PropertyValue}` que definem propriedades globais dos objetos aplicadas a todas as curvas criadas neste esboço. Para uma completa visualização das propriedades disponíveis veja GlobalProperty.

<axes_handle>
: este argumento opcional força o esboço a aparecer dentro dos eixos selecionados por `axes_handle` ao invés dos eixos correntes (ver gca).

# Descrição

`mesh` desenha uma superfície parametrizada utilizando um grid retangular definido pelas coordenadas `X` e `Y` (se `{X,Y}` não são especificados, este grid é determinado utilizando-se as dimensões da matriz `Z` ) ; em cada ponto deste grid, uma coordenada Z é dada utilizando-se a matriz `Z`. `mesh` é baseado no comando `surf` com a opção padrão `color_mode` = índice de branco (dentro do mapa de cores corrente) e `color_flag` = 0.

Especificação de dados de entrada:

Neste parágrafo, para sermos mais claros, não mencionaremos os argumentos opcionais `GlobalProperty` já que eles não interferem na entrada de dados (exceto pelas propriedades `"Xdata"`, `"Ydata"` e `"Zdata"`, ver GlobalProperty). Assume-se que todos estes argumentos opcionais poderiam estar presentes também.

Se `Z` é a única matriz especificada, (Z) esboça a matriz `Z` versus o grid definido por `1:size(Z,2)` ao longo do eixo x e `1:size(Z,1)` ao longo do eixo y.

# Observação

Para habilitar-se o modo de transparência, você deve ajustar a opção `color_mode` para 0.

# Exemplo

```
[X,Y]=meshgrid(-1:.1:1,-1:.1:1);
Z=X.^2-Y.^2;
xtitle('z=x2-y ^2');
mesh(X,Y,Z);
```

# Ver Também

surf, meshgrid, plot2d, LineSpec, GlobalProperty

# Autor

F.Belahcene

# Name

milk_drop — Função 3d gota de leite

```
z=milk_drop(x,y)
```

# Parâmetros

x,y
: dois vetores linhas de tamanhos n1 e n2.

z
: matriz de tamanho (n1,n2).

# Descrição

`milk_drop` é uma função que representa uma gota de leite caindo no leite. Pode ser usada para testar funções `eval3d` e `plot3d`.

# Exemplos

```
x=-2:0.1:2; y=x;
z=eval3d(milk_drop,x,y);
plot3d(x,y,z)
```

# Ver Também

eval3d, plot3d

# Autor

Steer S.

# Name

move — Move uma entidade gráfica e seus galhos.

```
move(h,xy)
move(h,xy,"alone")
```

# Parâmetros

h
: manipulador da entidade a ser movida

xy
: um array `[dx,dy]` que fornece o vetor translação a ser aplicado

"alone"
: palavra-chave string (opcional)

# Descrição

Esta rotina pode ser usada para aplicar uma translação em uma entidade de gráficos. Se a entidade tiver galhos (unidades de dependência hierárquica), elas também serão transladadas.

Dada a palavra chave `"alone"` , apenas a unidade especificada precisa ser redesenhada. Deve ser especialmente usada com a propriedade `pixel_drawing_mode` da entidade de figura (ver como desenhar objetos no modo de desenho `"xor"`).

# Exemplos

```

```

# Ver Também

get, set, draw, figure_properties, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

name2rgb — Retorna valores RGB correspondentes ao nome da cor

```
rgb=name2rgb(name)
```

# Parâmetros

name
: nome da cor

rgb
: vetor de valores RGB inteiros de uma cor

# Descrição

`name2rgb` retorna os valores RGB (vermelho, verde, azul) inteiros de uma cor dado o seu nome. O resultado é um vetor `[r,g,b]` onde `r`, `g` e `b` são inteiros entre 0 e 255 correspondentes aos componentes de cor vermelho, verde e azul. Como de uso, 0 significa nenhuma intensidade e 255 significa intensidade total da cor.

Se uma cor não for encontrada, é retornado `[]`.

A lista de todas as cores conhecidas é dada por color_list.

# Exemplos

```
rgb=name2rgb("gold")
rgb2name(rgb)
```

# Ver Também

color, color_list, rgb2name

# Name

newaxes — Cria uma nova entidade Axes (de eixos)

```
a=newaxes()
```

# Parâmetros

a
: o manipulador para a nova entidade Axes criada

# Descrição

`newaxes()` é usado para criar uma nova entidade `Axes` (ver graphics_entities) na figura corrente. As propriedades dessa entidade são herdadas da entidade `default_axes` (ver gda)

# Exemplos

```
clf()
a1=newaxes();
a1.axes_bounds=[0,0,1.0,0.5];
t=0:0.1:20;
plot(t,acosh(t),'r')
a2=newaxes();
a2.axes_bounds=[0,0.5,1.0,0.5];
x=0:0.1:4;
plot(x,sinh(x))
legend('sinh')

sca(a1); //tornando os primeiros eixos os eixos correntes
plot(t,asinh(t),'g')
legend(['acosh','asinh'])
```

# Ver Também

subplot, gda, sca

# Autor

S. Steer, INRIA

# Name

nf3d — Facetas retangulares para parâmetros da função plot3d

```
[xx,yy,zz]=nf3d(x,y,z)
```

# Parâmetros

x,y,x,xx,yy,zz
: 6 matrizes de reais

# Descrição

Função utilitária. Usada para transformar facetas retangulares codificadas em três matrizes x,y e z para o código Scilab para facetas, aceito por plot3d.

# Exemplos

```
//Uma esfera...
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
x= cos(u)'*cos(v);
y= cos(u)'*sin(v);
z= sin(u)'*ones(v);
//plot3d2(x,y,z) é equivalente a...
[xx,yy,zz]=nf3d(x,y,z); plot3d(xx,yy,zz)
```

# Ver Também

plot3d, plot3d2

# Name

object_editor — Descrição das capacidades do editor de objetos gráficos
graphic — Descrição das capacidades do editor de objetos gráficos
menus — Descrição das capacidades do editor de objetos gráficos

# Descrição

Os gráficos do Scilab permitem que o usuário interaja com os gráficos antes e depois de tê-los desenhado. Cada janela gráfica e seus conteúdos são representados por entidades hierárquicas. O topo da hierarquia é a figura (entidade Figure). Cada figura define pelo menos uma entidade galho do tipo Axes (eixos). Cada entidade Axes contém entidades folhas que são objetos gráficos básicos como, Rectangles, Arcs, Segs,... (retângulos, arcos, segmentos). Também contém o tipo Compound que são conjuntos recursivos de entidades.

O principal interesse do novo modo gráfico é tornar mudanças de propriedades mais fáceis. Este novo modo de gráficos fornece um conjunto de rotinas gráficas de alto-nível (ver set, get) utilizadas para controlar propriedades dos objetos tais como dados, coordenadas e escalas, cores e aparências sem que seja necessário entrar com os comandos gráficos iniciais novamente.

Entidades gráficas são associadas a variáveis Scilab do tipo `handle`. a qual chamaremos manipulador. Um manipulador é um identificador único que é associado a cada instância de uma entidade gráfica criada. Utilizando manipuladores, é possível alcançar propriedades de entidade através das rotinas "set" e "get" . Os manipuladores também são utilizados para manipular objetos gráficos, para movê-los, para fazer copiá-los ou deletá-los.

Para completar e utilizar a capacidade de um manipulador gráfico ao seu máximo, um editor de objetos gráficos também foi criado. É um conjunto de interfaces Tcl/Tk disponíveis para cada tipo de objeto gráfico (ver graphics_entities para mais detalhes) que pode ser habilitado para cada janela de gráficos. Para fazê-lo funcionar, selecione o menu `Edit` na janela de gráficos. Sete operações de edição gráfica estão disponíveis:

Select figure as current:
: permite que uma figura seja a corrente.

Redraw figure:
: redesenha o conteúdo da janela de gráficos

Erase figure:
: apaga o conteúdo da janela de gráficos. Sua ação corresponde a um comando `clf`.

Os últimos oito itens são especialmente dedicados ao editor gráfico:

Copy object:
: utilizando o mouse, permite que o usuário selecione um objeto 2d (como uma curva, um retângulo...) e coloque na área de transferência. Logo, por uma nova chamada a `Paste object`, (colar objeto), o objeto é copiado nos eixos correntes selecionados.

Paste object:
: permite que o usuário cole um objeto anterior posto na àrea de transferência nos eixos correntes selecionados.

Move object:
: utilizando o mouse, permite que o usuário mova um objeto 2d (como uma curva, um retângulo...) dentro dos eixos correntes selecionados.

Delete object:
: utilizando o mouse, permite que o usuário selecione um objeto 2d(como uma curva, um retângulo...) dentro dos eixos correntes selecionados e delete-o instantaneamente.

Figure Properties:
: lança a interface Tcl/Tk para o objeto Figure aplicado ao manipulador da figura da janela de gráficos.

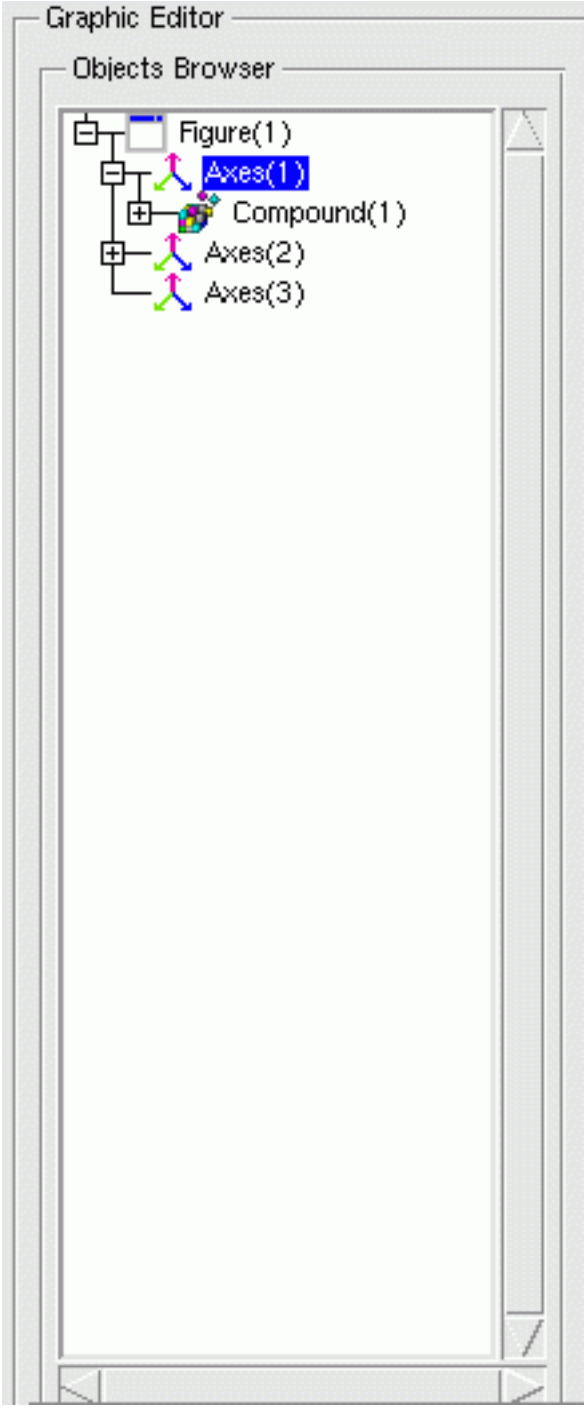
Current Axes Properties:
: lança a interface Tcl/Tk para o objeto Axes aplicado ao manipulador dos eixos correntes da janela de gráficos.

Start Entity Picker:
: inicia um manipulador de eventos na janela de gráficos para pegar os cliques do mouse sobre objetos gráficos e lança a interface Tcl/Tk correspondente. O clique com botão esquerdo do mouse permite edição de objetos e com botão direito realiza um movimento do objeto selecionado. Note que, por ora, este recurso só se aplica a objetos 2d.

Stop Entity Picker:
: para a ação de Entity Picker terminando o manipulador de eventos na janela de gráficos.

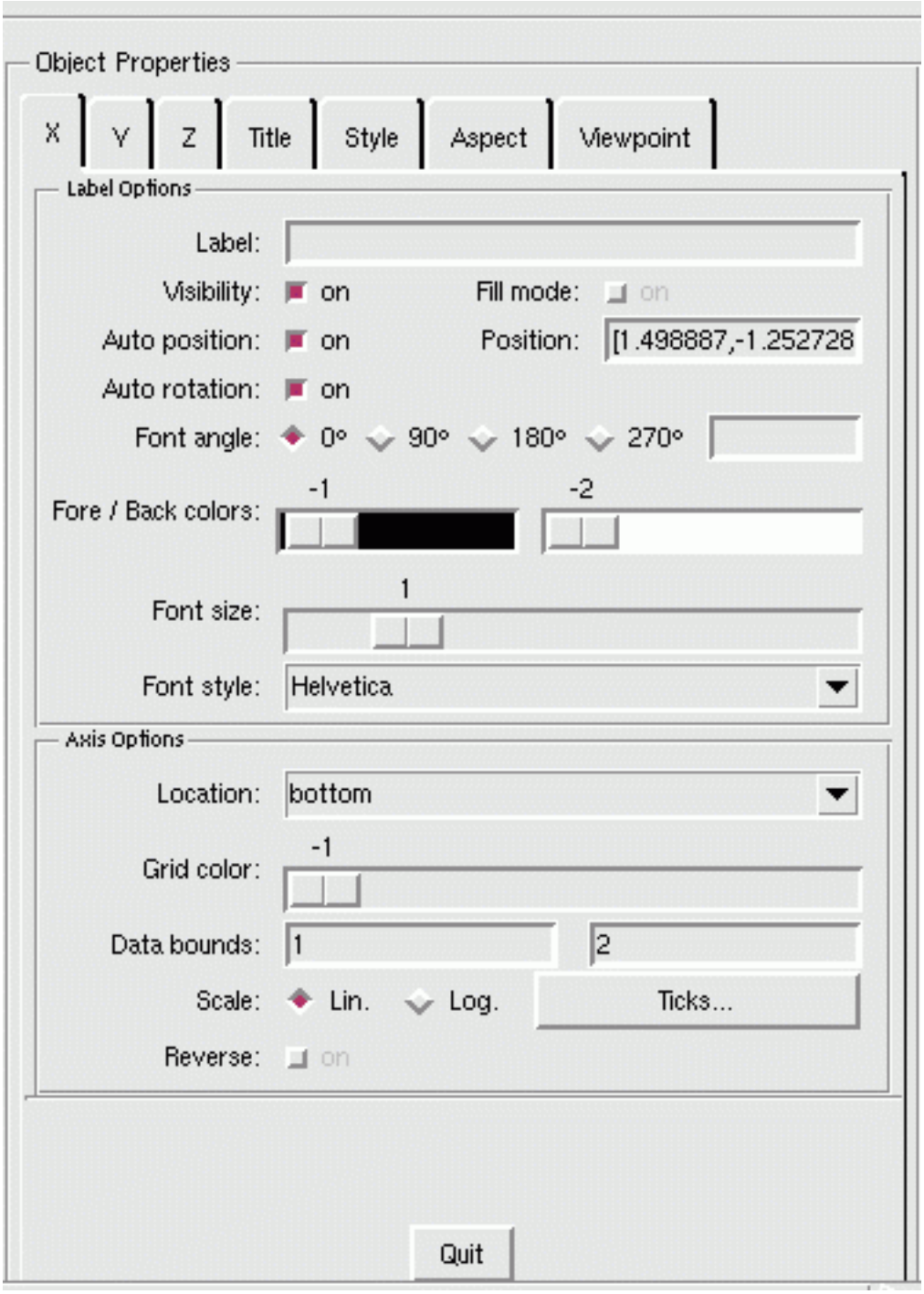
Uma vez que a interface gráfica é habilitada (utilizando as opções `Figure Properties` ou `Current Axes Properties`), duas áreas principais aparecem:

Um tree selector (seletor de árvore):
: posicionado à direita do editor gráfico, um seletor de árvores hierárquicas especifica que objeto é correntemente editado . Pode ser usado para alternar de um objeto gráfico para outro fornecido estando eles na mesma janela de gráficos.




Um notebook (caderno):

a segunda área representa um caderno composto de diferentes páginas de propriedades (como `Style`, `Data`, `Clipping`...) dependendo do objeto gráfico selecionado. Utilizando-se este editor, pode-se editar mais facilmente todo o conjunto de propriedade gráficas de cada objeto gráfico (como através dos comandos `"set"` e `"get"`). Aqui está um exemplo de um caderno para a entidade Axes exibindo as suas propriedades:




Furthermore, você poderá colocar legendas/anotações em sua figura utilizando primitivas de amostra fornecidas no menu `Insert` na janela gráfica. Utilizando o mouse e seguindo as instruções na sub-janela de mensagem, você poderá acrescentar:

Line:

Desenhe uma linha entre dois cliques do mouse. A linha vive nos eixos onde o primeiro ponto foi selecionado.

Polyline:
Desenhe uma poligonal clicando no botão esquerdo para definir o caminho da linha e clique no botão direito para completar o desenho. A poligonal vive nos eixos onde o primeiro ponto foi selecionado.

Arrow:
Desenhe uma seta entre dois cliques do mouse. A seta vive nos eixos onde o primeiro ponto foi selecionado.

Double arrow:
Desenhe uma seta dupla entre dois cliques do mouse. A seta dupla vive nos eixos onde o primeiro ponto foi selecionado.

Text:
Abra uma caixa de diálogo para inserir o texto, então clique na janela da figura para posicioná-lo. O texto vive nos eixos onde o primeiro ponto foi selecionado.

Rectangle:
Desenhe um retângulo: dois cliques com o botão direito definem os cantos superior esquerdo e inferior direito do retângulo. O retângulo vive nos eixos onde o primeiro ponto foi selecionado.

Circle:
Desenhe um círculo: 2dois cliques com o botão direito definem os cantos superior esquerdo e inferior direito da caixa em que o círculo vive. O retângulo vive nos eixos onde o primeiro ponto foi selecionado.

# Ver Também

graphics_entities, set, get, clf, plot

# Autor

F.Leray INRIA

# Name

oceancolormap — Mapa de cores linear azul

```
cmap=oceancolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`oceancolormap` computa um mapa de cores com `n` cores azuis que variam linearmente do preto para o azul

# Exemplos

```
f = scf();
plot3d1();
f.color_map = oceancolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap

# Name

oldplot — Esboço simples (versão antiga)

```
oldplot(x,y,[xcap,ycap,caption])
oldplot(y)
```

# Parâmetros

x,y
: dois vetores de tamanhos iguais

xcap,ycap,caption
: strings ou matrizes de strings

# Descrição

Esboça `y` em função `x`. `xcap` e `ycap` são legendas para os eixos x e y respectivamente e `caption` é a legenda do esboço.

IChamado com apenas um argumento, `oldplot(y)` esboça o vetor `y` ou, se `y` for uma matriz, esboça todos os vetores linhas em um mesmo esboço. Isto é feito em relação ao vetor 1: <número de colunas de `y`>.

`oldplot` está obsoleto. Use `plot2d` ou `plot` ao invés.

# Exemplos

```
x=0:0.1:2*%pi;
// esboço simples
oldplot(sin(x))
// usando legendas
xbasc()
oldplot(x,sin(x),"sin","tempo","esboço de seno")
// esboço de duas funções

xbasc()
oldplot([sin(x);cos(x)])
```

# Ver Também

plot2d, plot

# Autor

J.Ph.C.

# Name

param3d — esboço 3d de uma curva parametrizada

```
param3d(x,y,z,[theta,alpha,leg,flag,ebox])
```

# Parâmetros

x,y,z
: três vetores de mesmo tamanho (pontos da curva parametrizada).

theta, alpha
: valores reais fornecendo em graus as coordenadas esféricas do ponto de observação. *os valores padrões são 35 e 45 graus.*

leg
: string definindo rótulos para cada eixo com @ como separador de campo, por exemplo"X@Y@Z".

flag=[type,box]
: `type` e `box` possuem o mesmo significado que em `plot3d`:

  type
  : um inteiro (tipo de escala).

    type=0
    : o esboço é feito utilizando-se a escala 3d corrente (definida por uma chamada anterior a `param3d`, `plot3d`, `contour` ou `plot3d1`).

    type=1
    : re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são especificadas pelo valor do argumento opcional `ebox`.

    type=2
    : re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são computadas utilizando-se dados fornecidos. *Este é o valor padrão*

    type=3
    : fronteiras 3d isométricas com fronteiras da caixa dadas por `ebox`, de modo semelhante a `type=1`.

    type=4
    : fronteiras 3d isométricas derivadas dos dados, de modo semelhante a `type=2`.

    type=5
    : fronteiras 3d isométricas expandidas com fronteiras fornecidas por `ebox`, de modo semelhante a `type=1`.

    type=6
    : fronteiras 3d isométricas expandidas derivadas dos dados, de modo semelhante a `type=2`.Note que as fronteiras dos eixos podem ser customizadas através das propriedades da entidade de eixos (ver axes_properties).

  box
  : um inteiro (moldura ao redor do esboço).

    box=0
    : nada é desenhado ao redor do esboço.

    box=1
    : não implementado (é como box=0).

box=2
: apenas os eixos atrás da superfície são desenhados.

box=3
: uma caixa cercando a superfície é desenhada e legendas são adicionadas.

box=4
: uma caixa cercando a superfície é desenhada e legendas e eixos são adicionados. Note que o aspecto dos eixos também pode ser customizado através das propriedades da entidade Axes (ver axes_properties). *Este é o padrão*

ebox
: especifica as fronteiras do esboço através do vetor `[xmin,xmax,ymin,ymax,zmin,zmax]`. Este argumento é utilizado junto com `type` em `flag` se este for ajustado para `1`, `3` ou `5` (volte acima para ver o comportamento correspondente). Se `flag` estiver faltando, `ebox` não é levado em conta. Note que, quando especificado, o argumento `ebox` age no campo `data_bounds` field que também pode ser ajustado através das propriedades da entidade de eixos (ver axes_properties). O valor padrão de ebox é [0,1,0,1,0,1].

# Descrição

`param3d` é usado para esboçar uma curva 3d definida por suas coordenadas `x`, `y` e `z`. Note que os dados também podem ser ajustados ou recebidos através das propriedades da entidade superfície (ver surface_properties).

Note que propriedades como `rotation angles`, `colors` e `thickness` da superfície esboçada também podem ser ajustadas através das propriedades de param3d (ver param3d_properties).

Use `param3d1` para realizar esboços múltiplos.

Entre com o comando `param3d()` para visualizar uma demonstração.

# Exemplos

```
t=0:0.1:5*%pi;
param3d(sin(t),cos(t),t/10,35,45,"X@Y@Z",[2,3])

e=gce() //o manipulador para a poligonal 3d

e.foreground=color('red');

a=gca(); //o manipulador para os eixos
a.rotation_angles=[10 70];
```

# Ver Também

param3d1, plot3d

# Autor

J.Ph.C.

# Name

param3d1 — esboço 3d de curvas parametrizadas

```
param3d1(x,y,z,[theta,alpha,leg,flag,ebox])
param3d1(x,y,list(z,colors),[theta,alpha,leg,flag,ebox])
```

# Parâmetros

x,y,z
: matrizes de mesmo tamanho (nl,nc).

  Cada coluna i das matrizes corresponde às coordenadas da i-ésima curva. Você pode fornecer uma cor específica para cada curva utilizando `list(z,colors)` ao invés de `z`, onde `colors` é um vetor de tamanho `nc`. Se `color(i)` for negativo, a curva é esboçada utilizando-se a marca com identificador `abs(style(i))`; se `style(i)` for estritamente positivo, uma linha simples com idenficador de cor `style(i)` ou uma linha tracejada com identificador de tipo de tracejado `style(i)` é utilizada.

theta,alpha
: valores reais fornecendo em graus as coordenadas esféricas do ponto de observação. *Os valores padrões são 35 e 45 graus.*

leg
: string definindo rótulos para cada eixo com @ como separador de campo, por exemplo " X@Y@Z" .

flag=[type,box]
: `type` e `box` possuem o mesmo significado que em `plot3d`:

  type
  : um inteiro (tipo de escala).

    type=0
    : o esboço é feito utilizando-se a escala 3d corrente (definida por uma chamada anterior a `param3d`, `plot3d`, `contour` ou `plot3d1`).

    type=1
    : re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são especificadas pelo valor do argumento opcional `ebox`.

    type=2
    : re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são computadas utilizando-se dados fornecidos. *Este é o valor padrão*

    type=3
    : fronteiras 3d isométricas com fronteiras da caixa dadas por `ebox`, de modo semelhante a `type=1`.

    type=4
    : fronteiras 3d isométricas derivadas dos dados, de modo semelhante a `type=2`.

    type=5
    : fronteiras 3d isométricas expandidas com fronteiras fornecidas por `ebox`, de modo semelhante a `type=1`.

    type=6
    : fronteiras 3d isométricas expandidas derivadas dos dados, de modo semelhante a `type=2`. Note que as fronteiras dos eixos podem ser customizadas através das propriedades da entidade de eixos (ver axes_properties).

box
: um inteiro (moldura ao redor do esboço).

box=0
: nada é desenhado ao redor do esboço.

box=1
: não implementado (é como box=0).

box=2
: apenas os eixos atrás da superfície são desenhados.

box=3
: uma caixa cercando a superfície é desenhada e legendas são adicionadas.

box=4
: uma caixa cercando a superfície é desenhada e legendas e eixos são adicionados. Note que o aspecto dos eixos também pode ser customizado através das propriedades da entidade Axes (ver axes_properties). *Este é o valor padrão.*

ebox
: especifica as fronteiras do esboço através do vetor `[xmin,xmax,ymin,ymax,zmin,zmax]`. Este argumento é utilizado junto com `type` em `flag` se este estiver ajustado para `1`, `3` ou `5` (volte acima para ver o comportamento correspondente). Se `flag` estiver faltando, `ebox` não é levado em conta. Note que, quando especificado, o argumento `ebox` age no campo `data_bounds` fque também pode ser ajustado através das propriedades da entidade de eixos (ver axes_properties). O valor padrão de `ebox` é `[0,1,0,1,0,1]`.

# Descrição

`param3d1` usado para esboçar curvas 3d definidas por suas coordenadas `x`, `y` e `z`. Note que os dados também podem ser ajustados ou recebidos através das propriedades da entidade Surface (ver surface_properties).

Note que propriedades como `rotation angles`, `colors` e `thickness` das curvas esboçadas também podem ser ajustadas através das propriedades de param3d (ver param3d_properties).

Entre com o comando `param3d1()` para visualizar uma demonstração.

# Exemplos

```
xset('window',20) // criando janela de número 20
t=[0:0.1:5*%pi]';
param3d1([sin(t),sin(2*t)],[cos(t),cos(2*t)],..
         list([t/10,sin(t)],[3,2]),35,45,"X@Y@Z",[2,3])

xdel(20);
a=get("current_axes");//obtendo manipulador dos novos eixos criados
t=[0:0.1:5*%pi]';
param3d1([sin(t),sin(2*t)],[cos(t),cos(2*t)],[t/10,sin(t)])
a.rotation_angles=[65,75];
a.data_bounds=[-1,-1,-1;1,1,2]; //limites fornecidos por data_bounds
a.thickness = 2;
h=a.children //obtendo o manipulador para a entidade Param3d: um Compound compo
h.children(1).foreground = 3 // primeira curva
curve2 = h.children(2);
curve2.foreground = 6;
curve2.mark_style = 2;
```

## Ver Também

param3d, plot3d, plot2d, gca, xdel, delete

## Autor

J.Ph.C.

# Name

param3d_properties — description of the 3D curves entities properties

# Descrição

A entidade Param3d é uma folha na hierarquia de entidades gráficas.

visible:
: este campo contém o valor da propriedade `visible` para a entidade. Pode ser `"on"` ou `"off"` . Por padrão, as curvas são visíveis, a propriedade é `"on"` . Se `"off"` as curvas não são desenhadas na tela.

data:
: este campo contém as coordenadas de curvas 3d. É a matriz `[X,Y,Z]` tal que `X(:,i)`, `Y(:,i)`, `Z(:,i)` contém as coordenadas 3d da i-ésima curva. Este campo contém o valor padrão da propriedade `line_style` para objetos segmentos, arcos, retângulos e poligonais.

line_mode:
: este campo contém o valor padrão da propriedade `line_mode` para a poligonal. O valor pode ser `"on"` (linha desenhada) ou `"off"` (nenhuma linha desenhada).

line_style:
: este campo seleciona o tipo de linha a ser utilizada para desenhar linhas. O valor deve ser um inteiro em [0 6]. 0 significa linha sólida, o restante seleciona tracejados (ver getlinestyle).

polyline_style:
: este campo seleciona o modo como as curvas são desenhadas: modos interpolado (padrão), escada, esboço de barras, em setas e preenchido estão disponíveis por índices inteiros em [1:5].

mark_mode:
: este campo contém o valor padrão da propriedade `mark_mode`. Deve ser `"on"` ou `"off"` (valor padrão).

mark_style:
: este campo contém o valor padrão da propriedade `mark_style`. A propriedade `mark_style` seleciona o tipo de marca a ser exibida. o valor deve ser um inteiro em [0 9] que significa: ponto, sinal de mais, cruz, estrela, rombo preenchido, rombo, triângulo para cima, triângulo para baixo, trevo e círculo.

mark_size_unit:
: este campo contém o valor padrão da propriedade `mark_size_unit`. Se `mark_size_unit` for ajustado para `"point"`, então o valor de `mark_size` é diretamente dado em pontos. Quando `mark_size_unit` é ajustado para `"tabulated"`, `mark_size` é computado em relação ao array de tamanho de fonte: logo, seu valor deve ser um inteiro em [0 5] que significa 8pt, 10pt, 12pt, 14pt, 18pt e 24pt. Note que param3d e funções puras do Scilab utilizam o modo `tabulated` como padrão; quando se utiliza a função plot , o modo `point` é automaticamente habilitado.

mark_size:
: este campo contém o valor padrão da propriedade `mark_size`. A propriedade `mark_size` seleciona o tamanho de fonte da marca a ser exibida. Deve ser um valor inteiro em [0 5] que significa 8pt, 10pt, 12pt, 14pt, 18pt e 24pt (ver getmark).

mark_foreground:
: este campo contém o valor da propriedade `mark_foreground` que é a cor da borda das marcas. O valor deve ser um índice de cor (relativo ao mapa de cores corrente).

mark_background:
: este campo contém o valor da propriedade `mark_background` que é a cor da face das marcas. O valor deve ser um índice de cor (relativo ao mapa de cores corrente).

thickness:
este campo contém o valor padrão da propriedade `thickness` (espessura) das linhas utilizadas para desenhar os eixos e curvas. Deve ser um inteiro positivo.

foreground:
este campo contém o índice de cor utilizado para desenhar as curvas. O valor deve ser um índice de cor (relativo ao mapa de cores corrente).

clip_state:
este campo contém o valor padrão da propriedade `clip_state`. O valor pode ser:

- `"off"` significa que todas as curvas criadas posteriormente não serão recortadas(valor padrão).
- `"clipgrf"` significa que todas as cruvas criadas posteriormente serão recortadas fora do fora das fronteiras da entidade Axes.
- `"on"` significa que todas as curvas criadas posteriormente serão recortadas fora do retângulo dado pela propriedade clip_box.

clip_box:
este campo contém o valor padrão da propriedade `clip_box`. É uma matriz vazia se clip_state é `"off"`. Em outros casos, o recorte é dado pelo vetor `[x,y,w,h]` (ponto superior esquerdo, largura, altura).

user_data:
este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados da entidade Param3d e recuperá-la.

parent:
esta propriedade contém o manipulador para a raiz. A raiz de uma entidade curvas 3d deve ser do tipo `"Axes"` ou `"Compound"`.

# Exemplos

```
a=get("current_axes");//obtendo o manipulador dos novos eixos criados
t=[0:0.1:5*%pi]';
param3d1([sin(t),sin(2*t)],[cos(t),cos(2*t)],[t/10,sin(t)])

a.rotation_angles=[65,75];
a.data_bounds=[-1,-1,-1;1,1,2]; //limites dados por data_bounds
a.thickness = 2;
h=a.children //obtendo o manipulador da entidade param3d: um Compound composto
h.children(1).foreground = 3 // primeira curva
curve2 = h.children(2);
curve2.foreground = 6;
curve2.mark_style = 2;
```

# Ver Também

set, get, delete, param3d, param3d1, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

paramfplot2d — Esboço animado 2d, curva definida por uma função

```
paramfplot2d(f,x,theta)
paramfplot2d(f,x,theta,flag)
paramfplot2d(f,x,theta,flagrect)
```

# Parâmetros

x
: vetor de reais.

f
: função `y=f(x,t)` . f é uma função Scilab ou rotina dinamicamente ligada (referida através de um string).

theta
: vetor de reais (conjunto de parâmetros).

flag
: string `'no'` ou `'yes'`: Se "yes" a janela é limpa entre dois esboços consecutivos.

rect
: "retângulo" [xmin, xmax, ymin, ymax] (1 x 4 vetor de reais)

# Descrição

Esboço animado da função x-->f(x,t) para t= theta(1) ,theta(2), etc. f pode ser uma função do Scilab ou uma rotina dinamicamente ligada desde que y=f(x,t) seja avaliado como y=feval(x(:),t,f). Ver feval. A função f tal que x,t -> f(x,t) = `R^N` é avaliada em x= vetor de `R^N` e t=número real. x é um N-vetor de valores x e para cada t em theta, f(x,t)=N-vetor de valores y.

# Exemplos

```
deff('y=f(x,t)','y=t*sin(x)')
x=linspace(0,2*%pi,50);theta=0:0.05:1;
paramfplot2d(f,x,theta);
```

# Ver Também

plot2d, feval, fplot2d

# Name

pie — Desenha um gráfico de torta

```
pie(x)
pie(x[,sp[,txt]])
```

# Parâmetros

x
: um escalar real ou vetor de reais positivos.

sp
: um escalar real ou vetor de reais.

txt
: uma célula ou vetor de strings.

# Descrição

`pie(x)`: se o tamanho de x é N então a função desenha um gráfico de torta de N partes, a área da i-ésima parte é igual a (x(i)/sum(x))*( superfície do círculo unitário).

`pie(x,sp)`: o vetor sp permite cortar um ou vários pedaços da torta, (o tamanho de sp deve ser igual a N). Se o valor do i-ésimo índice de sp é diferente de 0, então a i-ésima parte é separada das outras por um espaço, senão, se for igual a zero, então é anexada às outras.

`pie(x,txt)`: o vetor txt permite escrever um texto para cada parte da torta, o i-ésimo componente de txt corresponde à i-ésima parte da torta (padrão: são escritas as porcentagens que correspondem às partes da superfície). O tamanho de txt deve ser igual a N.

# Exemplos

```
// primeiro exemplo: um argumento de entrada x=[1 2 5]
scf(0);
pie([1 2 5]);

// segundo exemplo: dois argumentos de entrada x=[5 9  4 6 3], sp=[0 1 0 1 0], 
scf(1);
pie([5 9 4 6 3],[0 1 0 1 0]);

// terceiro exemplo: três argumentos de entrada, x=[3 4 6 2], sp=[0 1 0 0], txt=
scf(2);
pie([3 4 6 2],[0 1 0 0],["parte 1","parte 2","parte 3","parte 4"]);
```

# Ver Também

xfpolys

# Autor

Farid Belahcene

# Name

pinkcolormap — Coloração de tons em sépia de imagens em preto e branco

```
cmap=pinkcolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`pinkcolormap` computa um mapa de cores que fornece uma coloração de tons em sépia de imagens em preto e branco

# Exemplos

```
f = scf();
plot3d1();
f.color_map = pinkcolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap

# Name

plot — Esboço 2d

```
plot(y,<LineSpec>,<GlobalProperty>)
plot(x,y,<LineSpec>,<GlobalProperty>)
plot(x1,y1,<LineSpec1>,x2,y2,<LineSpec2>,...xN,yN,<LineSpecN>,<GlobalProperty1>
plot(<axes_handle>,...)
```

# Parâmetros

x
: uma matriz ou vetor de reais. Se omitido, é assumido como sendo o vetor `1:n` onde `n` é o número de pontos de curva dado pelo parâmetro `y`.

y
: uma matriz de reais ou um vetor. `y` também pode ser uma função definida como um macro ou uma primitiva.

<LineSpec>
: este argumento opcional deve ser usado como um atalho para especificar um modo de desenhar uma linha. Podemos ter um `LineSpec` por `y` ou `{x,y}` previamente entrados. As opções `LineSpec` lidam com os especificadores LineStyle, Marker e Color (ver LineSpec). Estes especificadores determinam o estilo de linha, de marcas e a cor das linhas esboçadas.

<GlobalProperty>
: este argumento opcional representa uma seqüência de pares de declarações `{PropertyName,PropertyValue}` tque define propriedades globais de objetos a serem aplicadas a todas as curvas criadas pelo esboço. Para uma visualização completa de das propriedades disponíveis veja GlobalProperty.

<axes_handle>
: este argumento opcional força o esboço a aparecer dentro dos eixos selecionados fornecidos por `axes_handle` ao invés dos eixos correntes (ver gca).

# Descrição

`plot` esboça um conjunto de curvas 2d. `plot` foi reconstruído para lidar melhor com a sintaxe do Matlab. Para melhorar a compatibilidade gráfica com o Matlab, utilize `plot` (ao invés de plot2d).

Especificação de entrada de dados:

Neste parágrafo, para sermos mais claros, não mencionaremos os argumentos opcionais `LineSpec` ou `GlobalProperty` já que eles não interferem na entrada de dados (exceto pelas propriedades `"Xdata"`, `"Ydata"` e `"Zdata"`, ver GlobalProperty). É assumido que todos estes argumentos podem estar presentes também.

Se `y` é um vetor, plot(y) esboça um vetor `y` versus o vetor `1:size(y,'*')`.

Se `y` é uma matriz, plot(y) esboça cada coluna de `y` versus o vetor `1:size(y,1)`.

Se `x` e `y` são vetores, plot(x,y) esboça o vetor `y` versus o vetor `x`. Os vetores `x` e `y` devem ter o mesmo número de entradas.

Se `x` é um vetor e `y` uma matriz plot(x,y) esboça cada coluna de `y` versus o vetor `x`. Neste caso, o número de colunas de `y` deve ser igual ao número de entradas de `x`.

Se `x` e `y` são matrizes, plot(x,y) esboça cada coluna de `y` versus a coluna correspondente de `x`. Neste caso, os tamanhos `x` e `y` devem ser os mesmos.

Finalmente, se apenas `x` ou `y` é uma matriz, o vetor é esboçado versus cada linha ou cada coluna da matriz. A escolha é feita dependendo se a dimensão de linha ou coluna do vetor coincide com a dimensão de linha ou coluna da matriz. No caso de uma matriz quadrada (apenas `x` ou penas `y`), a prioridade é dada a colunas ao invés de linhas (ver exemplos abaixo).

`y` também pode ser uma função definida como um macro ou uma primitiva. Neste caso, os dados `x` devem ser fornecidos (como um vetor ou uma matriz) e a computação correspondente de `y(x)` é feita implicitamente.

Os argumentos `LineSpec` e `GlobalProperty` devem ser utilizados para customizar o esboço. Aqui está uma lista completa das opções disponíveis.

LineSpec
: esta opção pode ser utilizada para se especificar, de um modo curto e fácil, como as curvas são desenhadas. Deve sempre ser um string contendo referências aos especificadores LineStyle, Marker e Color.

  Essas referências devem ser ajustadas denro do string (a ordem não é importante) de modo a não ter ambigüidades. Por exemplo, para especificar uma linha vermelha de traço longo com marcas de rombos, pode-se escrever : `'r--d'` ou `'--dire'` ou `'--reddiam'` ou outra sentença sem ambigüidade... ou de modo completo `'diamondred--'` (ver LineSpec).

  Note que os estilos de linha e marcas (e tamanhos) e as cores, podem ser (re*)ajustados através das propriedades da entidade poligonal (ver polyline_properties).

GlobalProperty
: esta opção pode ser utilizada para especificar como as linhas serão desenhadas com mais opções que via `LineSpec`. Deve sempre ser um par de declarações constituidos de um string definindo `PropertyName`, (nome da propriedade) e seu valor associado `PropertyValue` (que pode ser um string, um inteiro ou qualquer outra coisa... dependendo do tipo de `PropertyName`). Utilizando-se `GlobalProperty`, pode-se ajustar várias propriedades : todas as propriedades disponíveis via LineSpec e mais: a cor da marca (plano de fundo e primeiro plano), a visibilidade, o recorte e a espessura das curvas. (ver GlobalProperty)

  Note que todas as propriedades podem ser (re-)ajustadas através das propriedades de entidades poligonais (ver polyline_properties).

# Observações

Por padrão, esboços sucessivos são superpostos. Para limpar o esboço anterior, use `clf()`. Para habilitar o modo `auto_clear` (limpeza automática) como modo padrão, edite seus eixos fazendo o seguinte:

da=gda();

da.auto_clear = 'on'

Para uma melhor exibição, a função `plot` pode modificar a propriedade `box` de seu Axes (eixos) raíze. Isto acontece quando uma entidade Axes é criada por uma chamada a `plot` ou é vazia antes da chamada. Se um dos eixos é centrado na origem, box é desabilitado. Em caso contrário, box é habilitado.

Para mais informações sobre a propriedade box e sobre o posicionamento dos eixos, veja axes_properties

Uma tabela de cores padrão é utilizada para colorir as curvas esboçadas quando você não especifica as cores. Ao desenhar linhas múltiplas, o comando plot automaticamente atribui as cores abaixo de modo cíclico. Aqui estão as cores utilizadas:

| R | G | B |
|---|---|---|
| 0. | 0. | 1. |

| | | |
|---|---|---|
| 0. | 0.5 | 0. |
| 1. | 0. | 0. |
| 0. | 0.75 | 0.75 |
| 0.75 | 0. | 0.75 |
| 0.75 | 0.75 | 0. |
| 0.25 | 0.25 | 0.25 |

Entre com o comando `plot` para visualizar uma demonstração.

# Exemplos

```
// inicialização de x
x=[0:0.1:2*%pi]';
//esboço simples
plot(sin(x))
clf()
plot(x,sin(x))
//esboços múltiplos
clf()
plot(x,[sin(x) sin(2*x) sin(3*x)])
clf()

// eixo à direita
plot(x,sin(x))
a=gca(); // manipulador da entidade Axes
a.y_location ="right";
clf()

// eixo centrado em (0,0)
plot(x-4,sin(x),x+2,cos(x))
a=gca(); // manipulador da entidade Axes
a.x_location = "middle";
a.y_location = "middle";

// algumas operações em entidades criadas por plot...
a=gca();
a.isoview='on';
a.children // listando os galhos da entidade Axes: aqui, é um galho Compound co
poly1= a.children.children(2); //armazenando um manipulador Polyline em poly1
poly1.foreground = 4; // outro modo de se mudar o estilo...
poly1.thickness = 3;  // ...e a espessura de uma curva.
poly1.clip_state='off' // controle de recorte
a.isoview='off';

//exemplos com LineSpec e GlobalProperty:
clf();
t=0:%pi/20:2*%pi;
plot(t,sin(t),'ro-.',t,cos(t),'cya+',t,abs(sin(t)),'--mo')
scf(2)
plot([t ;t],[sin(t) ;cos(t)],'xdat',[1:2])
scf(3)
axfig3 = gca();
scf(4) // deveria permanecer em branco
plot(axfig3,[t ;t],[sin(t) ;cos(t)],'zdat',[1:2],'marker','d','markerfac','gree
```

```
xdel(winsid())

//especificação de dados
t=-%pi:0.1:%pi;
size(t)
plot(t) // esboços simples de y versus tamanho do vetor t
clf(); // limpando figura

plot(t,sin(t)); // esboça sin(t) versus t
clf();

t=[1     1     1     1
   2     3     4     5
   3     4     5     6
   4     5     6     7];

plot(t) // esboça cada coluna t column versus tamanho de linha
clf();

subplot(221)
plot(t,sin(t)); // esboça sin(t) versus t coluna por coluna desta vez
xtitle("sin(t) versus t")
subplot(222)
plot(t,sin(t)')
xtitle("sin(t)'' versus t")
subplot(223)
plot(t',sin(t))
a=gca();
a.data_bounds=[0 -1;7 1]; // para ver a linha vertical escondida pelo eixo y
xtitle("sin(t) versus t''")
subplot(224)
plot(t',sin(t)')
xtitle("sin(t)'' versus t''")

clf();

//caso especial 1
//x : vector ([5 6 7 8]) and y : matrix (t)
x=[5 6 7 8]
plot(x,t);
plot(x',t); // idem, x é automaiticamente transposto para corresponder a t (aqu
clf()

// apenas um caso de possibilidade de correspondência: como realizar quatro esb
// x é um vetor 1x4 (vector) y is uma matriz não-quadrada 4x5
subplot(221);
plot(x,[t [8;9;10;12]]');
subplot(222);
plot(x',[t [8;9;10;12]]');
subplot(223);
plot(x,[t [8;9;10;12]]');
subplot(224);
plot(x',[t [8;9;10;12]]');
clf()

//caso especial 2
// caso onde apenas x ou y é uma matriz quadrada
//x : matrix (t) e y  : vetor ([1 2 3 4])
```

```
plot(t,[1 2 3 4]) // equivalente a plot(t,[1 1 1 1;2 2 2 2;3 3 3 3;4 4 4 4])
plot(t,[1;2;3;4]) // o mesmo esboço
clf();

// t é transposto: note a prioridade dada ao tratamento das colunas
plot(t',[1 2 3 4]) // equivalente a plot(t',[1 1 1 1;2 2 2 2;3 3 3 3;4 4 4 4])
plot(t',[1 2 3 4]') // o mesmo esboço
clf();

// y é uma função definida por...
// ..uma primitiva
plot(1:0.1:10,sin) // equivalente a plot(1:0.1:10,sin(1:0.1:10))
clf();

// ...uma macro:
deff('[y]=toto(x)','y=x.*x')
plot(1:10,toto)
```

# Ver Também

plot2d, surf, scf, clf, xdel, delete, LineSpec, GlobalProperty

# Autor

F.Leray

# Name

plot2d — esboço 2d

```
plot2d([x],y)
plot2d([x],y,<opt_args>)
```

# Parâmetros

x
: uma matriz ou vetor de reais. Se omitido, é assumido como sendo o vetor `1:n` onde `n` é o número de pontos da curva dados pelo parâmetro `y`.

y
: uma matriz ou vetor de reais

<opt_args>
: representa uma seqüência de sentenças `key1=value1,key2=value2`,... onde `key1`, `key2,...` podem ser um dos seguintes:

  style
  : ajusta o estilo da curva. O valor associado deve ser um vetor de reais com valores inteiros (positivos ou negativos)

  rect
  : as fronteiras mínimas requeridas para o esboço. O valor associado deve ser um vetor de reais com quatro entradas: `[xmin,ymin,xmax,ymax]`.

  logflag
  : ajusta a escala (linear ou logaritmica) ao longo dos eixos. O valor associado deve ser um string com um dos possíveis valores: `"nn"`, `"nl"` , `"ln"` e `"ll"` .

  frameflag
  : controla a computação dos intervalos de coordenadas reais a partir dos valores mínimos requeridos. O valor associado deve ser um inteiro entre 0 e 8

  axesflag
  : especifica como os eixos são desenhados. O valor associado deve ser um inteiro entre 0 e 5

  nax
  : ajusta os rótulos dos eixos e a definição de tiques. O valor associado deve ser um vetor de quatro entradas inteiras `[nx,Nx,ny,Ny]`

  leg
  : ajusta a legenda das curvas. O valor associado deve ser um string

# Descrição

`plot2d` esboça um conjunto de curvas 2d. Se você está familiarizado com a sintaxe do Matlab da função `plot` , você deve usar plot.

Se `x` e `y` são vetores, plot2d(x,y,<opt_args>) esboça o vetor `y` versus o vetor `x`. Os vetores `x` e `y` vectors devem ter o mesmo número de entradas.

Se `x` é um vetor e `y` uma matriz plot2d(x,y,<opt_args>) esboça cada coluna de `y` versus o vetor `x`. Neste caso, o número de colunas de `y` deve ser igual ao número de entradas de `x`.

Se `x` e `y` são matrizes, plot2d(x,y,<opt_args>) esboça cada coluna de `y` versus a coluna correspondente de `x`. Neste caso, `x` e `y` devem ter o mesmo tamanho.

Se `y` é um vetor, plot2d(y,<opt_args>) esboça o vetor `y` versus o vetor `1:size(y,'*')`.

Se `y` é uma matriz, plot2d(y,<opt_args>) esboça cada coluna de `y` versus o vetor `1:size(y,1)`.

Os argumentos `<opt_args>` devem ser usados para customizar o esboço.

style
: esta opção deve ser usada para especificar como as curvas são desenhadas. Se esta opção for especificada, o valor associado deve ser um vetor com tantas entradas quanto as curvas.

- Se `style(i)` é estritamente crescente, a curva é desenhada como uma linha simples e `style(i)` define o índice da cor para utilizada para desenhar a curva (ver getcolor). Note que o estilo de linha e a espessura podem ser ajustados através das propriedades da entidade poligonal (ver polyline_properties).

  Interpolação linear é feita entre os dados pontos da curva.

- Se `style(i)` é negativo ou zero, os pontos dados são desenhados utilizando-se marcas, `abs(style(i))` define a marca com identificador utilizada. Note que as cores das marcas e seus tamanhos podem ser ajustados através das propriedades da entidade poligonal (ver polyline_properties).

logflag
: esta opção é usada para ajustar a escala (linear ou logarítmica) ao longo dos eixos O valor associado é um string com um dos seguintes valores: `"nn"`, `"nl"` , `"ln"` e `"ll"`. `"l"` significa escala e graduação logarítimica e `"n"` escala normal.

rect
: esta opção pode ser usada para ajustar as fronteiras mínimas requeridas para o esboço. Se esta opção for especificada, o valor associado deve ser um vetor real de quatro entradas:

  `[xmin,ymin,xmax,ymax]`. `xmin` e `xmax` definem os limites para as abscissas, enquanto `ymin` e `ymax` definem os limites para as ordenadas.

  Este argumento pode ser utilizado junto com a opção `frameflag` para especificar como os limites dos eixos derivam do argumento dado `rect`. Se a opção `frameflag` não for fornecida, é suposta como sendo `frameflag=7`.

  As fronteiras dos eixos também podem ser customizadas através das propriedades da entidade de eixos (Axes) (ver axes_properties).

frameflag
: esta opção pode ser usada para controlar a computação dos intervalos de coordenadas reais a partir de valores mínimos requeridos. Mudanças reais podem ser maiores que os requerimentos mínimos.

  frameflag=0
  : não há computação, o esboço usa a escala anterior (ou padrão).

  frameflag=1
  : os intervalos reais são dados pela opção `rect`.

  frameflag=2
  : os intervalos reais são computados a partir dos valores máximos e mínimos dos dados `x` e `y`.

  frameflag=3
  : os intevalo reais são os intervalos fornecidos pela opção `rect` e aumentados para se obter uma escala isométrica.

  frameflag=4
  : os intervalos reais são computados a partir dos valores máximos e mínimos de `x` e `y` e aumentados para se obter uma escala isométrica.

frameflag=5
os intervalos reais são os intervalos fornecidos pela opção `rect` e aumentados para se obter rótulos de eixos embelezados.

frameflag=6
os intervalos reais são computados a partir dos valores mínimos dos dados `x` e `y` e então aumentados para se obter rótulos de eixos embelezados.

frameflag=7
é como `frameflag=1` , mas os esboços anteriores são redesenhados em uma nova escala. É utilizado para adicionar o gráfico corrente a um anterior.

frameflag=8
é como `frameflag=2` , mas os esboços anteriores são redesenhados em uma nova escala. É utilizado para adicionar o gráfico corrente a um anterior.

frameflag=9
é como `frameflag=8` mas a imagem é aumentada enlarged to para se obter rótulos de eixos embelezados. Este é o alor padrão

Os limites dos eixos também podem ser customizados através das propriedades da entidade de eixos (ver axes_properties)

axesflag
esta opção pode ser utilizada para especificar como os eixos são desenhados. O valor associado deve ser um inteiro entre 0 e 5 :

axesflag=0
nada é desenhado ao redor do esboço (axes_visible=["off" "off"];box="off").

axesflag=1
os eixos são desenhados, o eixo y é exibido à esquerda (axes_visible=["on" "on"];box="on",y_location="left").

axesflag=2
o esboço é cercado por uma caixa sem tiques. (axes_visible=["off" "off"];box="on").

axesflag=3
os eixos são desenhados, o eixo y é exibido à direita (axes_visible=["on" "on"];box="off",y_location="right").

axesflag=4
os eixos são desenhados centrados no meio da caixa de moldura

axesflag=5
os eixos são desenhados para se cruzarem no ponto (0,0) . Se o ponto (0,0) não estiver dentro da moldura, os eixos não aparecerão no gráfico (axes_visible=["on" "on"];box="on",y_location="middle").

axesflag=9
os eixos são desenhados, o eixo y é exibido à esquerda (axes_visible=["on" "on"];box="off",y_location="left"). Este é o valor padrão

O aspecto dos eixos também pode ser customizado através das propriedades da entidade de eixos (ver axes_properties).

nax
esta opção pode ser utilizada para se especificar os rótulos dos eixos e a definição de tiques quando a opção `axesflag=1` é utilizada. O valor associado deve ser um vetor de inteiros com quatro entradas `[nx,Nx,ny,Ny]`.

`Nx` fornece o número de tiques principais utilizados no eixo x (**não mais levado em conta**), `nx` fornece o número de subtiques entre dois tiques principais no eixo x.

`Ny` e `ny` fornecem as informações similares para o eixo y.

Se a opção `axesflag` não for ajustada, a opção `nax` supõe que a opção `axesflag` foi ajustada para 1.

leg

esta opção pode ser usada para se ajustar as legendas das curvas. deve ser um string com a forma `"leg1@leg2@...."` onde `leg1` , `leg2` , etc. São respectivamente as legendas para a primeira curva, segunda curva etc. o padrão é `" "`.

As legendas de curvas são desenhadas abaixo do eixo x. Esta opção não é tão flexível, use as funções captions ou legend preferivelmente.

# Mais Informações

Para obter mais informações sobre a sintaxe antiga de plot 2d, veja a documentação plot2d_old_version.

Por padrão, esboços sucessivos são superpostos. para limpar o esboço anterior use `clf()`.

Entre com o comando `plot2d()` para uma demonstração.

Outras funções plot2d de alto nível existem:

- plot2d2 é o mesmo que `plot2d` mas as curvas são supostas seccionalmente constantes.
- plot2d3 é o mesmo que `plot2d` mas a curva é esboçada com barras verticais.
- plot2d4 é o mesmo que `plot2d` mas a curva é esboçada com flechas.

# Exemplos

```
// inicialização de x
x=[0:0.1:2*%pi]';
//esboço simples
plot2d(sin(x));
clf();
plot2d(x,sin(x));
//múltiplos esboços
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
// múltiplos esboços sem fornecer as dimensões da moldura
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)],rect=[0,0,6,0.5]);
//múltiplos esboços com legendas dados tiques + estilo
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)],..
       [1,2,3],leg="L1@L2@L3",nax=[2,10,2,10],rect=[0,-2,2*%pi,2]);
// isovisualização
clf();
plot2d(x,sin(x),1,frameflag= 4);
// escala
clf();
plot2d(x,sin(x),1,frameflag= 6);
// auto-escala com esboços anteriores + estilo
```

```
clf();
plot2d(x,sin(x),-1);
plot2d(x,2*sin(x),12);
plot2d(2*x,cos(x),3);
// eixo à direita
clf();
plot2d(x,sin(x),leg="sin(x)");
a=gca(); // manipulador da entidade Axes
a.y_location ="right";
// eixo centrado em (0,0)
clf();
plot2d(x-4,sin(x),1,leg="sin(x)");
a=gca(); // manipulador da entidade Axes
a.x_location = "middle";
a.y_location = "middle";
// algumas operações sobre entidades criadas por plot2d ...
a=gca();
a.isoview='on';
a.children // lista dos galhos dos eixos
// há um Compound formado por duas poligonais e uma legenda
poly1= a.children(1).children(1); //armazena um manipulador de poligonal em poly
poly1.foreground = 4; // outra maneira de se mudar o estilo...
poly1.thickness = 3;  // ...e a espessura de uma curva.
poly1.clip_state='off'; // controle de recorte
leg = a.children(2); // armazena manipulador de legenda em leg
leg.font_style = 9;
leg.line_mode = "on";
a.isoview='off';
```

# Ver Também

plot, plot2d1, plot2d2, plot2d3, plot2d4, clf, xdel, delete

# Name

plot2d1 — Esboço 2d em escala logarítmica (obsoleto)

```
plot2d1(str,x,y,[style,strf,leg,rect,nax])
```

# Parâmetros

str
: string de comprimento 3 `"abc"`.

a
: pode ter um dos seguintes valores: `e`, `o` ou `g`.

e
: significa "vazio". Especifica que o valor de `x` não é utilizado (os valores de `x` são supostos regularmente espaçados, ie 1:<número de linhas de `y`>). De qualquer forma, o usuário deve fornecer o valor de `x`, 1 por exemplo: `plot2d1("enn",1,y)`.

o
: significa "um". Se há várias curvas, todas possuem o mesmo valor x: `x` é um vetor coluna de tamanho n1 e `y` é uma matriz de tamanho(nl,nc). Por exemplo: `x=[0:0.1:2*%pi]';plot2d1("onn",x,[sin(x) cos(x)])`.

g
: significa"geral". `x` e `y` devem ter o mesmo tamanho (nl,nc). Cada coluna de `y` é esboçada em relação à coluna correspondente de `x`. `nc` curvas são esboçadas utilizando-se `nl` pontos.

b, c
: pode possuir valores `n` (normal) ou `l` (logarítmico).

b=l
: um eixo logarítmico é utilizado em x

c=l
: um eixo logarítmico é utilizado em y

x,y,[style,strf,leg,rect,nax]
: estes argumentos possuem o mesmo significado que na função `plot2d`.

opt_args
: estes argumentos possuem o mesmo significado que na função `plot2d`.

# Descrição

Esta função está obsoleta. Utilize `plot2d` ao invés.

`plot2d1` esboça um conjunto de curvas 2d. É o mesmo que `plot2d` mas com mais um argumento `str` que habilita escala logarítmica. Ainda, permite especificar apenas um vetor coluna para `x` quando é o mesmo para todas as curvas.

Por padrão, esboços sucessivos são superpostos. Para limpar o esboço anterior, use `clf`.

Entre com o comando `plot2d1()` para visualizar uma demonstração.

# Exemplos

```
// esboços múltiplos sem fornecer x
x=[0:0.1:2*%pi]';
plot2d1("enn",1,[sin(x) sin(2*x) sin(3*x)])
// esboços múltiplos fornecendo apenas x
clf()
plot2d1("onn",x,[sin(x) sin(2*x) sin(3*x)])
// esboço logarítmico
x=[0.1:0.1:3]'; clf()
plot2d1("oll",x,[exp(x) exp(x^2) exp(x^3)])
```

# Ver Também

plot2d, plot2d2, plot2d3, plot2d4, clf

# Autor

J.Ph.C.

# Name

plot2d2 — esboço 2d (funções de degraus)

```
plot2d2([x],y)
plot2d2([x],y,<opt_args>)
plot2d2([logflag],x,y,[style,strf,leg,rect,nax])
```

# Parâmetros

args
: ver `plot2d` para uma descrição dos parâmetros.

# Descrição

`plot2d2` é o mesmo que `plot2d` mas as funções dadas por `(x,y)` são supostas seccionalmente constantes.

Por padrão, esboços sucessivos são superpostos. Para limpar os esboços anteriores, use `clf()`.

Entre com o comando `plot2d2()` para visualizar uma demonstração. Note que todos os modos propostos por `plot2dxx` (xx = 1 a 4) podem ser habilitados utilizando-se `plot2d` e ajustando a opção `polyline_style` para o número correspondente.

# Exemplos

```
// esboçando uma função de degrau de valor i no segmento [i,i+1]
// o último segmento não é desenhado
plot2d2([1:4],[1:4],1,"111","função de degrau",[0,0,5,5])
// compare o seguinte com plot2d
x=[0:0.1:2*%pi]';
clf()
plot2d2(x,[sin(x) sin(2*x) sin(3*x)])
  // apenas no novo modo gráfico
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
e=gce();
e.children(1).polyline_style=2;
e.children(2).polyline_style=2;
e.children(3).polyline_style=2;
```

# Ver Também

plot2d, plot2d3, plot2d4, subplot, clf, polyline_properties

# Autor

J.Ph.C.

# Name

plot2d3 — esboço 2d (barras veticais)

```
plot2d3([logflags,] x,y,[style,strf,leg,rect,nax])
plot232(y)
plot2d3(x,y <,opt_args>)
```

# Parâmetros

args
: ver `plot2d` para uma descrição dos parâmetros.

# Descrição

`plot2d3` é o mesmo que `plot2d` mas as curvas são esboçadas utilizando-se barras verticais.

Por padrão, esboços sucessivos são superpostos. Para limpar esboços anteriores, use `clf()`.

Entre com o comando `plot2d3()` para visualizar uma demonstração. Note que todos os modos propostos por `plot2dxx` (xx = 1 a 4) podem ser habilitados utilizando-se `plot2d` podem ser habilitados utilizando-se `polyline_style` para o número correspondente.

# Exemplos

```
// compare o seguinte com plot2d1
x=[0:0.1:2*%pi]';
plot2d3(x,[sin(x) sin(2*x) sin(3*x)])
// no novo modo de gráficos apenas
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
e=gce();
e.children(1).polyline_style=3;
e.children(2).polyline_style=3;
e.children(3).polyline_style=3;
```

# Ver Também

plot2d, plot2d2, plot2d4, clf, polyline_properties

# Autor

J.Ph.C.

# Name

plot2d4 — esboço 2d (setas)

```
plot2d4([logflag,] x,y,[style,strf,leg,rect,nax])
plot2d4(y)
plot2d4(x,y <,opt_args>)
```

# Parâmetros

args
: ver `plot2d` para uma descrição dos parâmetros.for a description of parameters.

# Descrição

`plot2d4` é o mesmo que `plot2d` mas as curvas são esboçadas utilizando-se setas. Isto pode ser útil para se esboçar solulções de uma EDO (equação diferencial ordinária) em um espaço fásico.

Por padrão, esboços sucessivos são superpostos. Para limpar o esboço anterior, use `clf()`.

Entre com o comando `plot2d4()` para visualizar uma demonstração. Note que todos os modos propostos por `plot2dxx` (xx = 1 a 4) podem ser habilitados utilizando-se `plot2d` e ajustando-se a opção `polyline_style` para o número correspondente.

# Exemplos

```
// compare o seguinte com plot2d1
x=[0:0.1:2*%pi]';
plot2d4(x,[sin(x) sin(2*x) sin(3*x)])
 // apenas no novo modo gráfico
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
e=gce();
e.children(1).polyline_style=4;
e.children(2).polyline_style=4;
e.children(3).polyline_style=4;
```

# Ver Também

fchamp, plot2d, plot2d2, plot2d3, subplot, clf, polyline_properties

# Autor

J.Ph.C.

# Name

plot2d_old_version — **As sintaxes descritas abaixo estão obsoletas**

```
plot2d([logflag],x,y,[style,strf,leg,rect,nax])
```

# Parâmetros

x,y
: duas matrizes (ou vetores colunas).

- De maneira usual, `x` é uma matriz de mesmo tamanho que `y` (a coluna j de `y` é esboçada em relação à coluna j de `x`)
- Se todas as colunas de `x` são iguais (ie, as abscissas de todas as curvas são as mesmas), `x` pode ser simplesmente o vetor coluna destas abscissas (`x` é então um vetor coluna de comprimento igual a dimensão de linha de `y`).
- Quando `x` não é fornecido, é suposto como sendo o vetor [1; 2; ...; dimensão de linha de y].

style
: é um vetor real de tamanho nc. O estilo a ser utilizado na curva `i` é definido por `style(i)`. O estilo padrão é `1:nc` (1 para a primeira curva, 2 para a segunda, etc.).

- Se `style(i)` é negativo ou zero, a curva é esboçada com marca de identificador `abs(style(i))`; use `xset()` para ajustar o identificador de marca e xget('mark') para obter o identificador de marca corrente.
- Se `style(i)` for estritamente positivo, uma linha simples de identificador de cor `style(i)` ou uma linha tracejada com identificador de tracejado `style(i)` é utilizada; use `xset()` para vizualizar os identificadores de cor.
- Quando apenas uma curva é desenhada, `style` pode ser um vetor linha de tamanho `[sty,pos]` onde `sty` é utilizado para especificar o estilo e `pos` é um inteiro entre 1 e 6 que especifica uma posição a ser utilizada para a legenda. Pode ser útil quando o usuário deseja desenhar múltiplas curvas em um esboço chamando a função `plot2d` várias vezes e quer fornecer uma legenda para cada curva. **Esta opção não possui mais efeito no modo gráfico novo. Utilize a função captions para ajustar as legendas quando todas as curvas forem desenhadas.**

strf
: string de comprimento 3 `"xyz"` (por padrão `strf= "081"`)

x
: controla a exibição de legendas.

x=0
: sem legendas.

x=1
: legendas são exibidas. Elas são dadas pelo argumento opcional `leg`.

y
: controla a computação dos intervalos de coordenadas reais a partir dos valores mínimos requeridos. Intervalos reais podem ser maiores que os requerimentos mínimos.

y=0
: sem computação, o esboço utiliza a escala anterior (ou padrão)

y=1
: a partir do argumento rect

y=2
: a partir dos valores mínimo/máximo dos dados x e y

y=3
: constrói uma escala isométrica a partir do argumento rect

y=4
: constrói um esboço isométrico a partir dos valores mínimo/máximo dos dados x e y

y=5
: aumentados para obtenção de bons eixos a partir do argumento rect

y=6
: aumentados para obtenção de bons eixos a partir dos valores mínimo/máximo dos dados x e y

y=7
: como y= 1 mas os anteriores são redesenhados para se utilizar a nova escala

y=8
: como y= 2 mas os anteriores são redesenhados para se utilizar a nova escala

z
: controla a exibição de informação na moldura ao redor do esboço. Se eixos são requeridos, o número de tiques pode ser especificado pelo argumento opcional `nax`.

z=0
: nada é desenhado ao redor do esboço.

z=1
: eixos são desenhados, o eixo y é exibido à esquerda.

z=2
: o esboço é cercado por uma caixa sem tiques.

z=3
: eixos são desenhados, o eixo y é exibido à direita.

z=4
: eixos são desenhados centrados no meio da caixa de moldura.

z=5
: eixos são desenhados cruzando-se no ponto `(0,0)`. Se o ponto `(0,0)` não estiver dentro da moldura, os eixos não aparecerão no gráfico.

leg
: um string. É utilizado quando o primeiro caractere x do argumento `strf` é 1. `leg` tem a forma `"leg1@leg2@...."` onde `leg1`, `leg2`, etc. são respectivamente as legendas para a primeira, segunda, etc. curvas. O padrão é `" "` .

rect
: este argumento é utilizado quando o segundo caractere y do argumento `strf` is 1, 3 ou 5. é 1, 3 ou 5. É um vetor linha de tamanho quatro e fornece a dimensão da moldura: `rect=[xmin,ymin,xmax,ymax]`.

nax
: este argumento opcional é utilizado quando o terceiro caractere z do argumento `strf` é 1. É um vetor linha de quatro entradas `[nx,Nx,ny,Ny]` onde `nx` (`ny`) é o número de sub-graduações no eixo x (y) axis e `Nx` (`Ny`) é o número de graduações no eixo x (y).

logflag
: um string formado por dois caracteres h (para eixo horizontal) e v (for para eixo vertical) cada um desses caracteres pode assumir valores "n" ou "l". "l" corresponde à graduação logarítmica e "n" à graduação normal. Por exemplo "ll" corresponde a um esboço com ambos os eixos x e y em graduação logarítmica. O valor padrão é "nn".

# Descrição

`plot2d` esboça um conjunto de curvas 2d. Esboço linear seccional é utilizado.

Por padrão, esboços sucessivos são superpostos. Para limpar o esboço anterior use `xbasc()`.

Veja o significado dos parâmetros acima para uma completa descrição.

Entre com o comando `plot2d()` para uma demonstração.

Outras funções plot2d de alto-nível existem:

- plot2d2: é o mesmo que `plot2d` mas as curvas são supostas seccionalmente constantes.
- plot2d3: é o mesmo que `plot2d` mas a curva é esboçada com barras verticais.
- plot2d4: é o mesmo que `plot2d` mas a curva é esboçada com flechas.

# Exemplos

```
//esboço simples
x=[0:0.1:2*%pi]';
plot2d(sin(x))
xbasc()
plot2d(x,sin(x))
//esboços múltiplos
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
// esboços múltiplos fornecendo as dimensões da moldura
// sintaxe antiga e sintaxe nova
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],1:3,"011","",[0,0,6,0.5])
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],rect=[0,0,6,0.5])
//esboços múltiplos com legendas e tiques dados// velha sintaxe e nova sintaxe
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],..
      [1,2,3],"111","L1@L2@L3",[0,-2,2*%pi,2],[2,10,2,10]);
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],..
  [1,2,3],leg="L1@L2@L3",nax=[2,10,2,10],rect=[0,-2,2*%pi,2])
// isovisualização
xbasc()
plot2d(x,sin(x),1,"041")
// escala
xbasc()
plot2d(x,sin(x),1,"061")
// auto-escala com esboços anteriores
xbasc()
plot2d(x,sin(x),1)
plot2d(x,2*sin(x),2)
plot2d(2*x,cos(x),3)
```

```
// eixo à direita
xbasc()
plot2d(x,sin(x),1,"183","sin(x)")
// eixo centrado
xbasc()
plot2d(x,sin(x),1,"184","sin(x)")
// eixo centrado em(0,0
xbasc()
plot2d(x-4,sin(x),1,"185","sin(x)")
```

# Ver Também

plot2d1, plot2d2, plot2d3, plot2d4, xbasc, xset

# Autor

J.Ph.C.

# Name

plot3d — esboço 3d de uma superfície

```
plot3d(x,y,z,[theta,alpha,leg,flag,ebox])
plot3d(x,y,z,<opt_args>)

plot3d(xf,yf,zf,[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,zf,<opt_args>)

plot3d(xf,yf,list(zf,colors),[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,list(zf,colors),<opt_args>)
```

# Parâmetros

x,y
: vetores linhas de tamanhos n1 e n2 (coordenadas dos eixos x e y). Estas coordenadas devem ser monótonas.

z
: matriz de tamanho (n1,n2). `z(i,j)` é o valor da superfície no ponto (x(i),y(j)).

xf,yf,zf
: matrizes de tamanho (nf,n). Elas definem as facetas usadas para desenhar a superfície. Há `n` facetas. Cada faceta `i` é definida por um polígono com `nf` pontos. As coordenadas dos eixos x, y e z dos pontos da i-ésima faceta são dados respectivamente por `xf(:,i)`, `yf(:,i)` e `zf(:,i)`.

colors
: um vetor de tamanho n fornecendo as cores de cada faceta ou uma matriz de tamanho (nf,n) fornecendo a cor próxima a cada borda da faceta (a cor da faceta é interpolada).

<opt_args>
: representa uma seqüência de sentenças `key1=value1, key2=value2`,... onde `key1`, `key2,...` pode ser um dos seguintes: theta, alpha , leg, flag, ebox (ver definições abaixo).

theta, alpha
: valores reais de dados em graus, as coordenadas esféricas de observação do ponto.

leg
: string definindo os rótulos para cada eixo com @ como um separador de campos, por exemplo "X@Y@Z".

flag
: um vetor real de tamanho três. `flag=[mode,type,box]`.

  mode
  : um inteiro (cor da superfície).

    mode>0
    : a superfície é pintada com a cor `"mode"` ; a borda da faceta é desenhada com o estilo e linha e cor correntes.

    mode=0:
    : uma malha da superfície é desenhada.

    mode<0:
    : a superfície é pintada com a cor `"-mode"` ; a borda da faceta não é desenhada.

      Note que o tratamento de cor da superfície pode ser feito utilizando-se as opções `color_mode` e `color_flag` através das propriedades da entidade superfície (ver surface_properties).

type
um inteiro (tipo de escala).

type=0:
o esboço é feito utilizando-se a escala 3d corrente (definida por uma chamada anterior a `param3d`, `plot3d`, `contour` ou `plot3d1`).

type=1:
re-escala automaticamente caixas 3d com relção de aspecto extrema, as fronteiras são especificadas pelo valor do argumento opcional `ebox`.

type=2:
re-escala automaticamente caixas 3d com relção de aspecto extrema, as fronteiras são computadas utilizando-se dados fornecidos.

type=3:
fronteiras 3d isométricas com fronteiras da caixa dadas por `ebox`, de modo semelhante a `type=1`.

type=4:
fronteiras 3d isométricas derivadas dos dados, de modo semelhante a `type=2`.

type=5:
fronteiras 3d isométricas expandidas com fronteiras fornecidas por `ebox`, de modo semelhante a `type=1`.

type=6:
fronteiras 3d isométricas expandidas derivadas dos dados, de modo semelhante a `type=2`.

Note que as fronteiras dos eixos podem ser customizadas através das propriedades da entidade de eixos (ver axes_properties).

box
um inteiro (moldura ao redor do esboço).

box=0:
nada é desenhado ao redor do esboço.

box=1:
não implementado (é como box=0).

box=2:
apenas os eixos atrás da superfície são desenhados.

box=3:
uma caixa cercando a superfície é desenhada e legendas são adicionadas.

box=4:
uma caixa cercando a superfície é desenhada e legendas e eixos são adicionados.

Note que o aspecto dos eixos pode ser customizado através das propriedades da entidade de eixos (ver axes_properties).

ebox
especifica as fronteiras do esboço através do vetor `[xmin,xmax,ymin,ymax,zmin,zmax]`. Este argumento é utilizado junto com `type` em `flag`, se este for ajustado para `1`, `3` ou `5` (volte acima para ver o comportamento correspondente). Se `flag` estiver faltando, `ebox` não é levado em conta.

Note que, quando especificado, o argumento `ebox` age no campo `data_bounds` que também pode ser ajustado através das propriedades da entidade de eixos (ver axes_properties).

# Descrição

`plot3d(x,y,z,[theta,alpha,leg,flag,ebox])` desenha a superfície parametrizada `z=f(x,y)`.

`plot3d(xf,yf,zf,[theta,alpha,leg ,flag,ebox])` desenha uma superfície definida por um conjunto de facetas. Você pode desenhar esboços multiplos substituindo `xf`, `yf` e `zf` por múltiplas matrizes montadas por linhas como `[xf1 xf2 ...]`, `[yf1 yf2 ...]` e `[zf1 zf2 ...]`. Note que os dados também podem ser ajustados ou recebidos através das propriedades da entidade superfície (ver surface_properties).

Você pode fornecer uma cor específica para cada faceta utilizando `list(zf,colors)` ao invés de `zf`, onde `colors` é um vetor de tamanho `n`. Se `colors(i)` for positivo, fornece a cor da faceta `i` e a borda da faceta é desenhada utilizando-se o estilo de linha e cor correntes. Se `colors(i)` for negativo, o identificador de cor `-colors(i)` é utilizado e a borda da faceta não é desenhada.

Também é possível obter cores interpoladas para as facetas. Para este tipo de cor, o argumento deve ser uma matriz nf x n fornecendo as cores próximas a cada borda da faceta de cada faceta. Neste caso, valores positivos para cores significam que as bordas não seram desenhadas. Note que as cores também podem ser ajustadas através das propriedades da entidade Surface (superfície) (via `tlist` affectations) e editadas utilizando-se a opção `color_flag` (ver surface_properties).

Os arugumentos opcionais `theta, alpha, leg ,flag, ebox`, podem ser passados por uma seqüência de argumentos `key1=value1, key2=value2`, ... Neste caso, a ordem não tem significado especial. Note que todos estes argumentos, exceto `flag` podem ser customizados através de propriedades da entidade de eixos (ver axes_properties). Como descrito anteriormente, a opção `flag` lida com as propriedades da entidade superfície para `mode` (ver surface_properties) e as propriedades de eixos (Axes) para `type` e `box` (ver axes_properties).

Você pode utilizar a função `genfac3d` para computar facetas de quatro lados da superfície `z=f(x,y)`. `eval3dp` também pode ser utilizado.

Entre com o comando `plot3d()` para visualizar uma demonstração.

# Exemplos

```
// esboço simples utilizando z=f(x,y)
t=[0:0.3:2*%pi]';
z=sin(t)*cos(t');
plot3d(t,t,z)
// o mesmo esboço utilizando facetas computadas por genfac3d
[xx,yy,zz]=genfac3d(t,t,z);
clf()
plot3d(xx,yy,zz)
// esboços múltiplos
clf()
plot3d([xx xx],[yy yy],[zz 4+zz])
// esboços múltiplos utilizando-se cores
clf()
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
// esboço simples utilizando ponto de observação e legendas
clf()
plot3d(1:10,1:20,10*rand(10,20),alpha=35,theta=45,flag=[2,2,3])
```

```
// esboço de uma esfera utilizando utilizando facetas computadas por eval3dp
deff("[x,y,z]=sph(alp,tet)",["x=r*cos(alp).*cos(tet)+orig(1)*ones(tet)";..
     "y=r*cos(alp).*sin(tet)+orig(2)*ones(tet)";..
     "z=r*sin(alp)+orig(3)*ones(tet)"]);
r=1; orig=[0 0 0];
[xx,yy,zz]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
clf();plot3d(xx,yy,zz)
clf();
f=gcf();
f.color_map = hotcolormap(128);
r=0.3;orig=[1.5 0 0];
[xx1,yy1,zz1]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
cc=(xx+zz+2)*32;cc1=(xx1-orig(1)+zz1/r+2)*32;
clf();plot3d1([xx xx1],[yy yy1],list([zz,zz1],[cc cc1]),theta=70,alpha=80,flag=

//operações disponíveis apenas no novo modo de gráficos
delete(gcf());
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
[xx,yy,zz]=genfac3d(t,t,z);
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
e=gce();
f=e.data;
TL = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,6*rand(f.z)); // matriz de co
e.data = TL;
TL2 = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,4*rand(1,800)); // vetor de
e.data = TL2;
TL3 = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,[20*ones(1,400) 6*ones(1,400
e.data = TL3;
TL4 = tlist(["3d" "x" "y" "z"],f.x,f.y,f.z); // nenhuma cor
e.data = TL4;
e.color_flag=1 // índice de cor proporcional à altitude (coordenada z)
e.color_flag=2; // de volta ao modo padrão
e.color_flag= 3; // modo de gradação interpolada (baseada na cor azul padrão)
clf()
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
h=gce(); //obtendo manipulador da entidade corrente (aqui é a superfície)
a=gca(); //obtendo eixos correntes
a.rotation_angles=[40,70];
a.grid=[1 1 1]; //criando grides
a.data_bounds=[-6,0,-1;6,6,5];
a.axes_visible="off"; //os eixo estão escondidos
a.axes_bounds=[.2 0 1 1];
h.color_flag=1; //colorindo de acordo com z
h.color_mode=-2;  //removendo as bordas das facetas ajustando color_mode para a
h.color_flag=2; //colorindo de acordo com as cores fornecidas
h.color_mode = -1; // colocando de volta as bordas das facetas ajustando color_
f=gcf();//obtendo o manipulador da figura raiz
f.color_map=hotcolormap(512);
c=[1:400,1:400];
TL.color = [c;c+1;c+2;c+3];
h.data = TL;
h.color_flag=3; // modo de gradação interpolada
```

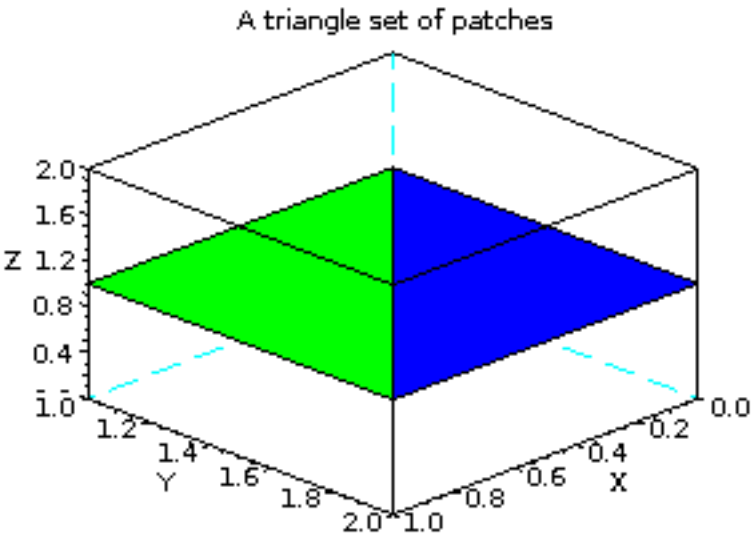
We can use the plot3d function to plot a set of patches (triangular, quadrangular, etc).

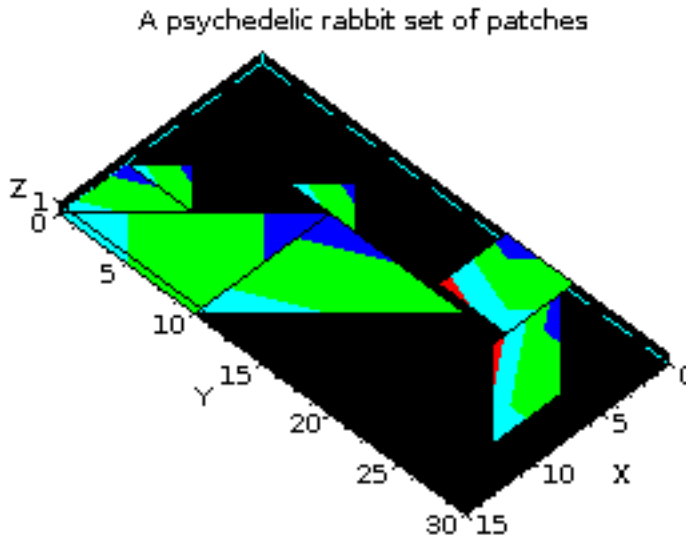
```
// The plot3d function to draw patches:
```

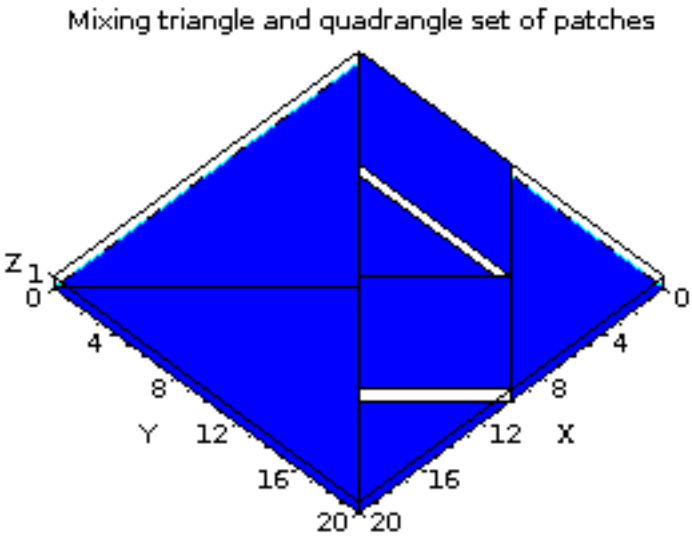
```
// patch(x,y,[z])
// patch(x,y,[list(z,c)])
// The size of x : number of points in the patches x number of patches
// y and z have the same sizes as x
// c:
// - a vector of size number of patches: the color of the patches
// - a matrix of size number of points in the patches x number of
//   patches: the color of each points of each patches

// Example 1: a set of triangular patches

x = [0 0;
     0 1;
     1 1];

y = [1 1;
     2 2;
     2 1];

z = [1 1;
     1 1;
     1 1];

tcolor = [2 3]';

subplot(2,2,1);
plot3d(x,y,list(z,tcolor));
xtitle('A triangle set of patches');

// Example 2: a mixture of triangular and quadrangular patches

xquad = [5, 0;
         10,0;
         15,5;
         10,5];

yquad = [15,0;
         20,10;
         15,15;
         10,5];

zquad = ones(4,2);

xtri = [ 0,10,10, 5, 0;
        10,20,20, 5, 0;
        20,20,15,10,10];

ytri = [ 0,10,20, 5,10;
        10,20,20,15,20;
         0, 0,15,10,20];

ztri = zeros(3,5);

subplot(2,2,3);
plot3d(xquad,yquad,zquad);
plot3d(xtri,ytri,ztri);
xtitle('Mixing triangle and quadrangle set of patches');
```

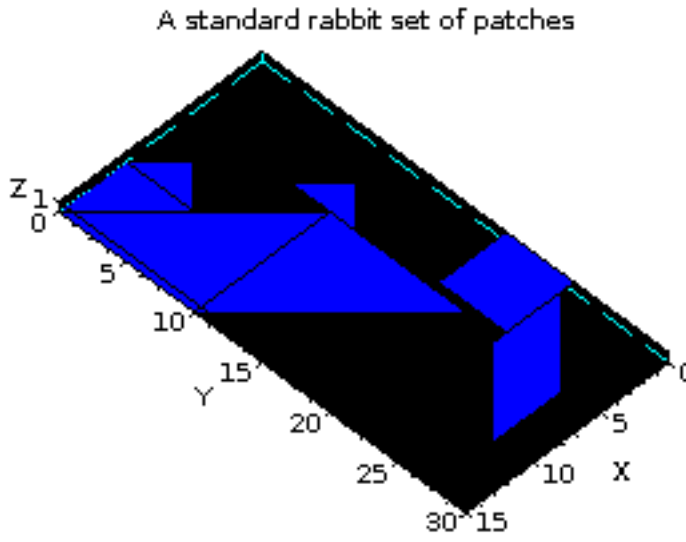
```
// Example 3: some rabbits

rabxtri = [ 5,  5, 2.5,  7.5, 10;
            5, 15, 5,   10,   10;
           15, 15, 5,   10,   15];

rabytri = [10, 10, 9.5,  2.5, 0;
           20, 10, 12,   5,   5;
           10   0   7    0    0];
rabztri = [0,0,0,0,0;
           0,0,0,0,0;
           0,0,0,0,0];

rabtricolor_byface = [2 2 2 2 2];

rabtricolor = [2,2,2,2,2;
               3,3,3,3,3;
               4,4,4,4,4];

rabxquad = [0, 1;
            0, 6;
            5,11;
            5, 6];

rabyquad = [18,23;
            23,28;
            23,28;
            18,23];

rabzquad = [1,1;
            1,1;
            1,1;
            1,1];

rabquadcolor_byface = [2 2];

rabquadcolor = [2,2;
                3,3;
                4,4;
                5,5];

subplot(2,2,2);
plot3d(rabxtri, rabytri, list(rabztri,rabtricolor));
plot3d(rabxquad,rabyquad,list(rabzquad,rabquadcolor));
h = gcf();
h.children(1).background = 1;
xtitle('A psychedelic rabbit set of patches');

subplot(2,2,4);
plot3d(rabxtri, rabytri, list(rabztri,rabtricolor_byface));
plot3d(rabxquad,rabyquad,list(rabzquad,rabquadcolor_byface));
h = gcf();
h.children(1).background = 1;
xtitle('A standard rabbit set of patches');
```

The result of the preceding example:



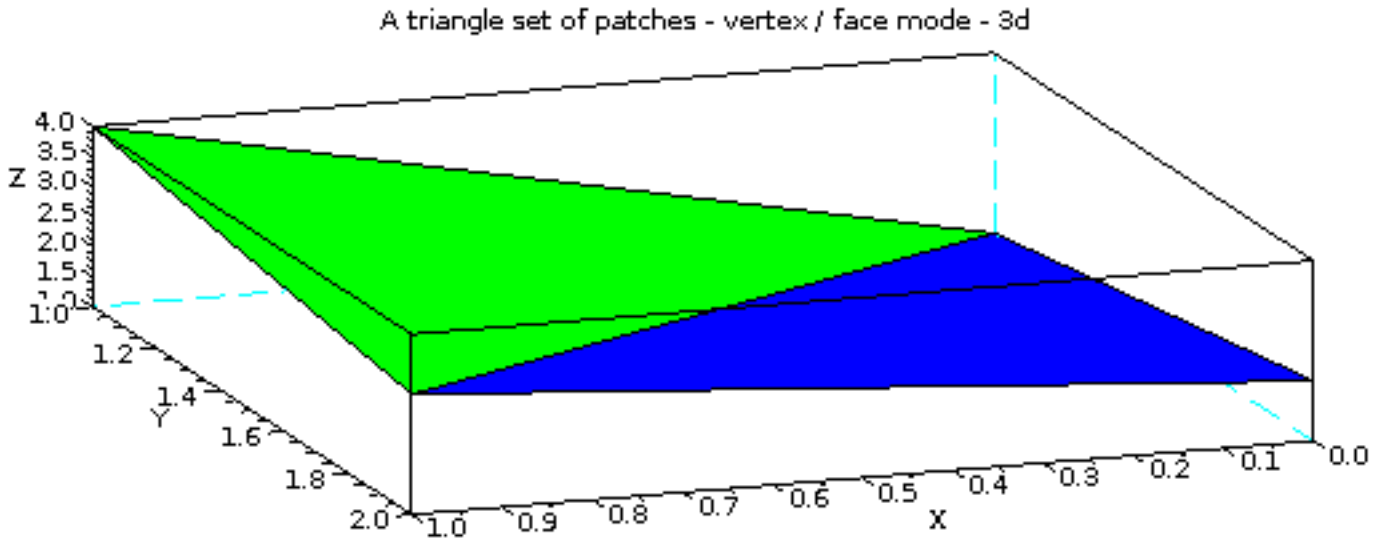



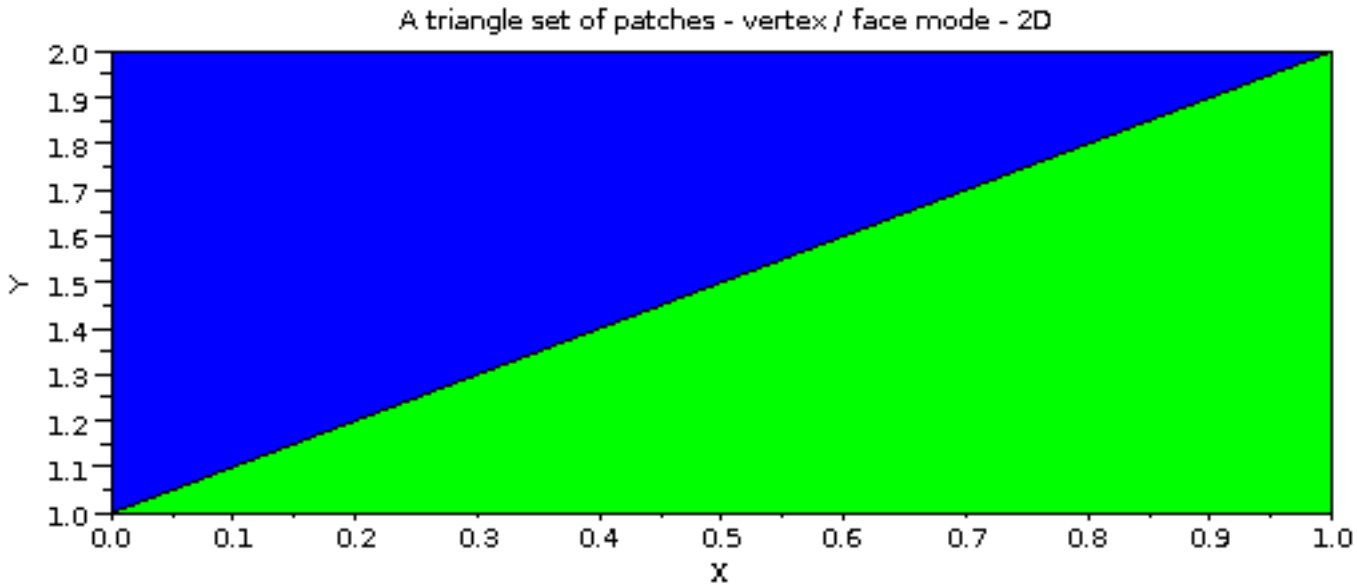

We can also use the plot3d function to plot a set of patches using vertex and faces.

```
// Vertex / Faces example: 3D example

// The vertex list contains the list of unique points composing each patch
// The points common to 2 patches are not repeated in the vertex list

vertex = [0 1 1;
          0 2 2;
          1 2 3;
          1 1 4];

// The face list indicates which points are composing the patch.
face = [1 2 3;
        1 3 4];

tcolor = [2 3]';
```

```
// The formula used to translate the vertex / face representation into x, y, z

xvf = matrix(vertex(face,1),size(face,1),length(vertex(face,1))/size(face,1))';
yvf = matrix(vertex(face,2),size(face,1),length(vertex(face,1))/size(face,1))';
zvf = matrix(vertex(face,3),size(face,1),length(vertex(face,1))/size(face,1))';

scf();
subplot(2,1,1);
plot3d(xvf,yvf,list(zvf,tcolor));
xtitle('A triangle set of patches - vertex / face mode - 3d');

// 2D test
// We use the 3D representation with a 0 Z values and then switch to 2D represe

// Vertex / Faces example: 3D example

// The vertex list contains the list of unique points composing each patch
// The points common to 2 patches are not repeated in the vertex list

vertex = [0 1;
          0 2;
          1 2;
          1 1];

// The face list indicates which points are composing the patch.
face = [1 2 3;
        1 3 4];

// The formula used to translate the vertex / face representation into x, y, z

xvf = matrix(vertex(face,1),size(face,1),length(vertex(face,1))/size(face,1))';
yvf = matrix(vertex(face,2),size(face,1),length(vertex(face,1))/size(face,1))';
zvf = matrix(zeros(vertex(face,2)),size(face,1),length(vertex(face,1))/size(fac

subplot(2,1,2);
plot3d(xvf,yvf,list(zvf,tcolor));
xtitle('A triangle set of patches - vertex / face mode - 2D');
a = gca();
a.view = '2d';
```

The result of the preceding example:




How to set manually some ticks

```
plot3d();
h = gca();
h.x_ticks = tlist(['ticks','locations','labels'],[-2,-1,0,1,2],['-2','-1','0','
h.y_ticks = tlist(['ticks','locations','labels'],[-4,-3-2,-1,0,1,2,3,4],['-4','
h.z_ticks = tlist(['ticks','locations','labels'],[-1,0,1],['Point 1','Point 2',
```

# Ver Também

eval3dp, genfac3d, geom3d, param3d, plot3d1, clf, gca, gcf, xdel, delete, axes_properties

# Autor

J.Ph.C.

# Name

plot3d1 — Esboço 3d em níveis de cinza ou de cores de uma superfície

```
plot3d1(x,y,z,[theta,alpha,leg,flag,ebox])
plot3d1(xf,yf,zf,[theta,alpha,leg,flag,ebox])

plot3d1(x,y,z,<opts_args>)
plot3d1(xf,yf,zf,<opts_args>)
```

# Parâmetros

x,y
: vetores linhas de tamanhos n1 e n2 (coordenadas dos eixos x e y). Estas coordenadas devem ser monótonas.

z
: matriz de tamanho (n1,n2). `z(i,j)` é o valor da superfície no ponto (x(i),y(j)).

xf,yf,zf
: matrizes de tamanho (nf,n). Elas definem as facetas usadas para desenhar a superfície. Há `n` facetas. Cada faceta `i` é definida por um polígono com `nf` pontos. As coordenadas dos eixos x, y e z dos pontos da i-ésima faceta são dados respectivamente por `xf(:,i)`, `yf(:,i)` e `zf(:,i)`.

<opt_args>
: representa uma seqüência de declarações `key1=value1, key2=value2`,... onde `key1`, `key2,...` pode ser um dos seguintes: theta, alpha , leg, flag, ebox (ver definições abaixo).

theta, alpha
: valores reais de dados em graus, as coordenadas esféricas do ponto de observação.

leg
: string definindo os rótulos para cada eixo com @ como um separador de campos, por exemplo "X@Y@Z".

flag
: um vetor real de tamanho três. `flag=[mode,type,box]`.

  mode
  : um inteiro (cor da superfície).

    mode>0
    : a superfície é pintada com a cor `"mode"` ; a borda da faceta é desenhada com o estilo e linha e cor correntes.

    mode=0:
    : uma malha da superfície é desenhada.

    mode<0:
    : a superfície é pintada com a cor `"-mode"` ; a borda da faceta não é desenhada.

      Note que o tratamento de cor da superfície pode ser feito utilizando-se as opções `color_mode` e `color_flag` através das propriedades da entidade Surface (ver surface_properties).

  type
  : um inteiro (tipo de escala).

    type=0:
    : o esboço é feito utilizando-se a escala 3d corrente (definida por uma chamada anterior a `param3d`, `plot3d`, `contour` ou `plot3d1`).

type=1:
re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são especificadas pelo valor do argumento opcional `ebox`.

type=2:
re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são computadas utilizando-se dados fornecidos.

type=3:
fronteiras 3d isométricas com fronteiras da caixa dadas por `ebox`, de modo semelhante a `type=1`.

type=4:
fronteiras 3d isométricas derivadas dos dados, de modo semelhante a `type=2`.

type=5:
fronteiras 3d isométricas expandidas com fronteiras fornecidas por `ebox`, de modo semelhante a `type=1`.

type=6:
fronteiras 3d isométricas expandidas derivadas dos dados, de modo semelhante a `type=2`.

Note que as fronteiras dos eixos podem ser customizadas através das propriedades da entidade Axes (ver axes_properties).

box
um inteiro (moldura ao redor do esboço).

box=0:
nada é desenhado ao redor do esboço.

box=1:
não implementado (é como box=0).

box=2:
apenas os eixos atrás da superfície são desenhados.

box=3:
uma caixa cercando a superfície é desenhada e legendas são adicionadas.

box=4:
uma caixa cercando a superfície é desenhada e legendas e eixos são adicionados.

Note que o aspecto dos eixos pode ser customizado através das propriedades da entidade Axes (ver axes_properties).

ebox
especifica as fronteiras do esboço através do vetor `[xmin,xmax,ymin,ymax,zmin,zmax]`. Este argumento é utilizado junto com `type` em `flag` : é ajustado para `1`, `3` ou `5` (volte acima para ver o comportamento correspondente). Se `flag` estiver faltando, `ebox` não é levado em conta.

Note que, quando especificado, o argumento `ebox` age no campo `data_bounds` que também pode ser ajustado através das propriedades da entidade de eixos (ver axes_properties).

# Descrição

`plot3d1` esboça uma superfície com cores dependendo do nível z da superfície. Esta função de esboço especial também pode ser habilitada ajustando `color_flag=1` após um `plot3d` (ver surface_properties)

Entre com o comando `plot3d1()` para visualizar uma demonstração.

# Exemplos

```
// esboço simples utilizando z=f(x,y)
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
plot3d1(t,t,z)
// mesmo esboço utilizando facetas computadas por genfac3d
[xx,yy,zz]=genfac3d(t,t,z);
clf();
plot3d1(xx,yy,zz)
// esboços múltiplos
clf();
plot3d1([xx xx],[yy yy],[zz 4+zz])
// esboço simples com ponto de vista e legendas
clf() ;
plot3d1(1:10,1:20,10*rand(10,20),35,45,"X@Y@Z",[2,2,3])
// mesmo esboço sem grid
clf()
plot3d1(1:10,1:20,10*rand(10,20),35,45,"X@Y@Z",[-2,2,3])
// esboço de uma esfera utilizando facetas computadas por eval3dp
deff("[x,y,z]=sph(alp,tet)",["x=r*cos(alp).*cos(tet)+orig(1)*ones(tet)";..
"y=r*cos(alp).*sin(tet)+orig(2)*ones(tet)";..
"z=r*sin(alp)+orig(3)*ones(tet)"]);
r=1; orig=[0 0 0];
[xx,yy,zz]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
clf()

plot3d(xx,yy,zz)
e=gce();
e.color_flag=1;
scf(2);
plot3d1(xx,yy,zz) // os dois gráficos são similares
```

# Ver Também

plot3d, gca, gce, scf, clf

# Autor

J.Ph.C.

# Name

plot3d2 — Esboço de superfície definida por facetas retangulares

```
plot3d2(X,Y,Z [,vect,theta,alpha,leg,flag,ebox])
plot3d2(X,Y,Z, <opt_args>)
```

# Parâmetros

X, Y, Z:
: três matrizes de reais definindo uma estrutura de dados.

vect
: vetor de reais.

<opt_args>
: representa uma seqüência de declarações `key1=value1, key2=value2`,... onde `key1`, `key2, ...` podem ser um dos seguintes: theta, alpha ,leg,flag,ebox (ver definições abaixo).

theta, alpha
: valores reais de dados em graus, as coordenadas esféricas do ponto de observação.

leg
: string definindo os rótulos para cada eixo com @ como um separador de campos, por exemplo "X@Y@Z".

flag
: um vetor real de tamanho três. `flag=[mode,type,box]`.

  mode
  : um inteiro (cor da superfície).

    mode>0
    : a superfície é pintada com a cor `"mode"` ; a borda da faceta é desenhada com os estilos correntes de linha e cor.

    mode=0:
    : uma malha da superfície é desenhada.

    mode<0:
    : a superfície é pintada com a cor `"-mode"` ; a borda da faceta não é desenhada.

      Note que o tratamento de cor da superfície pode ser feito utilizando-se as opções `color_mode` e `color_flag` através das propriedades da entidade superfície (ver surface_properties).

  type
  : um inteiro (tipo de escala).

    type=0:
    : o esboço é feito utilizando-se a escala 3d corrente (definida por uma chamada anterior a `param3d`, `plot3d`, `contour` ou `plot3d1`).

    type=1:
    : re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são especificadas pelo valor do argumento opcional `ebox`.

    type=2:
    : re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são computadas utilizando-se dados fornecidos.

type=3:
fronteiras 3d isométricas com fronteiras da caixa dadas por `ebox`, de modo semelhante a `type=1`.

type=4:
fronteiras 3d isométricas derivadas dos dados, de modo semelhante a `type=2`.

type=5:
fronteiras 3d isométricas expandidas com fronteiras fornecidas por `ebox`, de modo semelhante a `type=1`.

type=6:
fronteiras 3d isométricas expandidas derivadas dos dados, de modo semelhante a `type=2`.

Note que as fronteiras dos eixos podem ser customizadas através das propriedades da entidade Axes (ver axes_properties).

box
um inteiro (moldura ao redor do esboço)

box=0:
nada é desenhado ao redor do esboço.

box=1:
não implementado (é como box=0)

box=2:
apenas os eixos atrás da superfície são desenhados.

box=3:
uma caixa cercando a superfície é desenhada e legendas são adicionadas.

box=4:
uma caixa cercando a superfície é desenhada e legendas e eixos são adicionados.

Note que o aspecto dos eixos pode ser customizado através das propriedades da entidade Axes (ver axes_properties).

ebox
especifica as fronteiras do esboço através do vetor `[xmin,xmax,ymin,ymax,zmin,zmax]`. Este argumento é utilizado junto com `type` em `flag`, se for ajustado para `1`, `3` ou `5` (volte acima para ver o comportamento correspondente). Se `flag` estiver faltando, `ebox` não é levado em conta.

Note que, quando especificado, o argumento `ebox` age no campo `data_bounds` que também pode ser ajustado através das propriedades da entidade Axes (ver axes_properties).

# Descrição

`plot3d2` esboça uma superfície definida por facetas retangulares. (X,Y,Z) são três matrizes que descrevem uma superfície. A superfície é composta de polígonos de quatro lados.

As coordenadas x de uma faceta são dadas por X(i,j),X(i+1,j),X(i,j+1),X(i+1,j+1). De modo semelhante Y e Z são coordenadas Y e Z.

O vetor `vect` é usado quando múltiplas superfícies são codificadas nas mesmas matrizes (X,Y,Z). `vect(j)` fornece a linha na qual a codificação da j-ésima superfície começa. Como em `plot3d`, as mesmas propriedades são editáveis (ver surface_properties e axes_properties).

# Exemplos

```
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
X = cos(u)'*cos(v);
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
plot3d2(X,Y,Z);
// apenas no novo modo de gráficos
e=gce();
e.color_mode=4; // mudando a cor
f=e.data;
TL = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,10*(f.z)+1);
e.data=TL;
e.color_flag=2;
```

# Ver Também

plot3d, genfac3d

# Name

plot3d3 — Esboço de malha de uma superfície definida por facetas retangulares

```
plot3d3(X,Y,Z [,vect,theta,alpha,leg,flag,ebox])
plot3d3(X,Y,Z, <opt_args>)
```

# Parâmetros

X, Y, Z:
: 3 três matrizes de reais definindo uma estrutura de dados.

vect
: vetor de reais.

<opt_args>
: representa uma seqüência de declarações `key1=value1, key2=value2`,... onde `key1`, `key2, . . .` podem ser um dos seguintes: theta, alpha ,leg,flag,ebox (ver definições abaixo).

theta, alpha
: valores reais de dados em graus, as coordenadas esféricas do ponto de observação.

leg
: string definindo os rótulos para cada eixo com @ como um separador de campos, por exemplo "X@Y@Z".

flag
: um vetor real de tamanho quatro. `flag=[vertical_color, horizontal_color,type,box]`.

  vertical_color
  : um inteiro (cor da superfície), o padrão é 3.

    Índice do mapa de cores definindo a cor a ser utilizada para se desenhar as bordas verticais.

  horizontal_color
  : um inteiro (cor da superfície), o padrão é 4.

    Índice do mapa de cores definindo a cor a ser utilizada para se desenhar as bordas horizontais.

  type
  : um inteiro (escala) o padrão é 2.

    type=0:
    : o esboço é feito utilizando-se a escala 3d corrente (definida por uma chamada anterior a `param3d`, `plot3d`, `contour` ou `plot3d1`).

    type=1:
    : re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são especificadas pelo valor do argumento opcional `ebox`.

    type=2:
    : re-escala automaticamente caixas 3d com relação de aspecto extrema, as fronteiras são computadas utilizando-se dados fornecidos.

    type=3:
    : fronteiras 3d isométricas com fronteiras da caixa dadas por `ebox`, de modo semelhante a `type=1`.

    type=4:
    : fronteiras 3d isométricas derivadas dos dados, de modo semelhante a `type=2`.

type=5:
fronteiras 3d isométricas expandidas com fronteiras fornecidas por `ebox`, de modo semelhante a `type=1`.

type=6:
fronteiras 3d isométricas expandidas derivadas dos dados, de modo semelhante a `type=2`.

Note que as fronteiras dos eixos podem ser customizadas através das propriedades da entidade de eixos (ver axes_properties).

box
um inteiro (moldura ao redor do esboço), o padrão é 4.

box=0:
nada é desenhado ao redor do esboço.

box=1:
não implementado (é como box=0)

box=2:
apenas os eixos atrás da superfície são desenhados.

box=3:
uma caixa cercando a superfície é desenhada e legendas são adicionadas.

box=4:
uma caixa cercando a superfície é desenhada e legendas e eixos são adicionados.

Note que o aspecto dos eixos pode ser customizado através das propriedades da entidade Axes (ver axes_properties).

ebox
especifica as fronteiras do esboço através do vetor `[xmin,xmax,ymin,ymax,zmin,zmax]`. TEste argumento é utilizado junto com `type` em `flag`, se for ajustado para `1`, `3` ou `5` (volte acima para ver o comportamento correspondente). Se `flag` estiver faltando, `ebox` não é levado em conta. Note que, quando especificado, o argumento `ebox` age no campo `data_bounds` que também pode ser ajustado através das propriedades da entidade Axes (ver axes_properties).

# Descrição

`plot3d3` realiza um esboço de malha de uma superfície definida por facetas retangulares. (X,Y,Z) são três matrizes que descrevem uma superfície. A superfície é composta por polígonos de quatro lados.

As coordenadas X de uma faceta são dadas por X(i,j),X(i+1,j),X(i,j+1),X(i+1,j+1). De modo semelhante Y e Z são coordenadas Y e Z.

O vetor `vect` é usado quando múltiplas superfícies são codificadas nas mesmas matrizes (X,Y,Z). `vect(j)` fornece a linha na qual a codificação da j-ésima superfície começa. Ver `plot3d2` para uma descrição completa. Como um esboço de malha, a única propriedade disponível que você pode editar é a opção `visible` (ver axes_properties).

# Exemplos

```
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
X = cos(u)'*cos(v);
```

```
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
plot3d3(X,Y,Z);
// apenas no novo modo de gráficos
e=gce(); // obtendo a entidade corrente
e.visible='off';
e.visible='on'; // de volta à visão da malha
```

# Ver Também

plot3d2, plot3d, param3d

# Name

plotframe — esboça uma moldura com escalas e grides. Esta função está obsoleta.

```
plotframe(rect,tics,[arg_opt1,arg_opt2,arg_opt3])
plotframe(rect,<opts_args>)
```

# Parameters

rect
: vetor `[xmin,ymin,xmax,ymax]`.

tics
: vetor `[nx,mx,ny,my]` onde `mx, nx` (resp. `my, ny`) são o número de intervalos e subintervalos do eixo x (resp. eixo y) .

arg_optX
: até três argumentos opcionais escolhidos entre.

  flags
  : vetor `[wantgrids,findbounds]` onde `wantgrids` é uma variável booleana (`%t` ou `%f`) que indica o gridding. `findbounds` é uma variável booleana. Se `findbounds` for `%t`, os limites dados em `rect` podem ser ligeiramente modificados (na verdade sempre aumentados) a fim de se obter graduações mais simples: então `tics(2)` e `tics(4)` são ignorados.

  Captions
  : vetor de três strings `[title,x-leg,y-leg]` correspondentes respectivamente ao título do esboço e às legendas dos eixos x e y. Aviso: o "C" maiúsculo é importante.

  subwin
  : um vetor de tamanho 4 definindo a sub-janela. A sub-janela é especificada através do parâmetro `subwin=[x,y,w,h]` (ponto superior esquerdo, largura, altura). Os valores em `subwin` são especificados utilizando-se a proporção de largura ou altura da janela de gráficos corrente (ver `xsetech`).

<opts_args>
: representa uma seqüência de declarações **key1=value1**, **key2=value2**,... onde **key1**, **key2**,... podem ser um dos seguintes: **tics**, **flags**, **captions** ou **subwin**. Estes argumentos têm o mesmo significado que aqueles utilizados na primeira forma da rotina.

# Descrição

`plotframe` é utilizado com funções de esboço 2d plot2d, plot2d1,... para ajustar uma moldura gráfica. Deve ser usado antes de `plot2d` que deve ser chamado com o modo de superposição "000" .

**Esta função está obsoleta.**

# Exemplos

```
x=[-0.3:0.8:27.3]';
y=rand(x);
rect=[min(x),min(y),max(x),max(y)];
tics=[4,10,2,5];    //4 x-intervalos e 2 y-intervalos
plotframe(rect,tics,[%f,%f],["Meu esboço","x","y"],[0,0,0.5,0.5])
plot2d(x,y,2,"000")
plotframe(rect,tics=tics,flags=[%t,%f],Captions=["Meu esboço com grides","x","y
```

```
plot2d(x,y,3,"000")
plotframe(rect,tics,[%t,%t],..
["Meu esboço com grides e fronteiras automáticas","x","y"],[0,0.5,0.5,0.5])
plot2d(x,y,4,"000")
plotframe(rect,flags=[%f,%t],tics=tics,..
          Captions=["Meu esboço sem grides mas com fronteiras automáticas ","x"
          subwin=[0.5,0.5,0.5,0.5])
plot2d(x,y,5,"000")
```

# Ver Também

plot2d, graduate, xsetech

# Name

plzr — Esboço de pólo-zero

```
plzr(sl)
```

# Parâmetros

sl
: lista ( `syslin`)

# Descrição

Produz um esboço de pólo-zero do sistema linear `sl` (lista `syslin`)

# Exemplos

```
s=poly(0,'s');
n=[1+s   2+3*s+4*s^2        5; 0        1-s            s];
d=[1+3*s   5-s^3           s+1;1+s     1+s+s^2      3*s-1];
h=syslin('c',n./d);
plzr(h);
```

# Ver Também

trzeros, roots, syslin

# Name

polarplot — Esboço de coordenadas polares

```
polarplot(theta,rho,[style,strf,leg,rect])
polarplot(theta,rho,<opt_args>)
```

# Parâmetros

rho
: vetor, valores dos raio

theta
: vetor de mesmo tamanho que rho, valores dos ângulos.

<opt_args>
: uma seqüência de declarações `key1=value1, key2=value2`, ... onde keys podem ser `style`,`leg`,`rect`,`strf` ou `frameflag`

style
: vetor de reais de tamanho nc. O estilo a ser usado pela curva `i` é definido por `style(i)`. O valor padrão é `1:nc` (1 para a primeira curva, 2 para a segunda, etc.).

  -
  : Se `style(i)` for negativo, a curva é esboçada utilizando-se a marca com idenfiticador `abs(style(i))+1`; use `xset()` para ver os identificadores das marcas.

  -
  : Se `style(i)` for estritamente positivo, uma linha simples com identificador de cor `style(i)` ou linha tracejada com identificador de estilo de tracejado `style(i)` é usada; use `xset()` para ver os identificadores de cor.

  -
  : Quando apenas uma curva é desenhada, `style` pode ser um vetor linha de tamanho 2 `[sty,pos]` onde `sty` é utilizado para se especificar o estilo e `pos` é um inteiro entre 1 e 6 que especifica a posição para a legenda. Isto pode ser útil quando o usuário deseja desenhar curvas múltiplas em um esboço chamando a função `plot2d` várias vezes e quer fornecer uma legenda para cada curva.

strf
: istring de comprimento 3 `"xy0"`.

  default
  : o padrão é `"030"`.

  x
  : controla a exibição de legendas.

    x=0
    : sem legendas.

    x=1
    : com legendas. Elas são fornecidas pelo argumento opcional `leg`.

  y
  : controla a computação da moldura. É o mesmo que frameflag

    y=0
    : as fronteiras correntes (definidas por uma chamada anterior a uma função de esboço de alto-nível) são utilizadas. Útil ao se superpor esboços múltiplos.

y=1
: o argumento opcional `rect` é utilizado para se especificar as fronteiras do esboço.

y=2
: as fronteiras do esboço são computadas utilizando-se os valores mínimo e máximo de `x` e `y`.

y=3
: como `y=1` mas produz escala de isovisualização.

y=4
: como `y=2` mas produz escala de isovisualização.

y=5
: como `y=1` mas `plot2d` pode mudar as fronteiras do esboço e os tiques dos eixos para se obter boas graduações. Quando o botão de zoom é ativado, este modo é usado.

y=6
: como `y=2` mas `plot2d` pode mudar as fronteiras do esboço e os tiques dos eixos para se obter boas graduações. Quando o botão de zoom é ativado, este modo é usado.

y=7
: como `y=5` mas a escala do novo esboço é misturada à escala corrente.

y=8
: como `y=6` mas a escala do novo esboço é misturada à escala corrente.

leg
: string. É usado quando o primeiro caractere x do argumento `strf` é 1. `leg` possui a forma `"leg1@leg2@...."` onde `leg1`, `leg2`, etc. são respectivamente as legendas das primeira, segunda, etc. curvas. O padrão é " " .

rect
: é usado quando o segundo caractere y do argumento `strf` é 1, 3 ou 5. É um vetor linha de tamanho 4 e fornece as dimensões da moldura: `rect=[xmin,ymin,xmax,ymax]`.

# Descrição

polarplot cria um esboço de coordenadas polares do ângulo theta versus o raio rho. theta é o ângulo do eixo x ao vetor raio especificado em radianos; rho é o comprimento do vetor raio especificado em unidades de espaço-de-dados.

# Exemplos

```
t= 0:.01:2*%pi;
clf();polarplot(sin(7*t),cos(8*t))

clf();polarplot([sin(7*t') sin(6*t')],[cos(8*t') cos(8*t')],[1,2])
```

# Name

polyline_properties — Descrição das propriedades da entidade Polyline (poligonal)

# Descrição

A entidade Polyline uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros para poligonais.

parent:
esta propriedade contém o manipulador para a raiz. A raiz de uma entidade Polyline deve ser do tipo `"Axes"` ou `"Compound"`.

children:
esta propriedade contém um vetor com os galhos do manipulador. Contudo, manipuladores poligonais não possuem galhos correntemente.

visible:
este campo contém o valor da propriedade `visible` para a entidade. Pode ser `"on"` ou `"off"` . Por padrão, a poligonal é visível, o valor da propriedade é `"on"` . Se `"off"` a poligonal não é impressa na tela.

data:
esta propriedade contém o valor os valores para as coordenadas x e y. O componente z deve ser adicionado no caso de eixos 3d. é uma matriz de duas (três) colunas `[x,y,[z]]` determinando os pontos.

closed:
este valor determina se a poligonal é fechada ou não: o valor pode ser `"on"` ou `"off"` (o valor padrão depende da primitiva utilizada para criar a poligonal).

line_mode:
este campo contém o valor padrão da propriedade `line_mode` para a poligonal. O valor pode ser `"on"` (linha desenhada) ou `"off"` (nenhuma linha desenhada).

fill_mode:
se o campo `polyline_style` é diferente de 5, preenche o plano de fundo da curva com a cor definida pela propriedade `background`.

line_style:
o valor da propriedade `line_style` deve ser um inteiro em [1 6]. 1 significa linha sólida e os demais valores tipos diferentes de tracejados. (ver getlinestyle).

thickness:
este campo contém a propriedade de linha `thickness` (espessura). Deve ser um inteiro positivo.

arrow_size_factor:
este inteiro permite ajustar o tamanho das flechas desenhadas na poligonal. O tamanho real das flechas é o produto entre a propriedade `thickness` e o fator de tamanho (size factor).

polyline_style:
esta propriedade ajusta vários modos de desenho de poligonais:

- Se o valor é 0 ou 1 linhas são desenhadas utilizando-se dois pontos consecutivos.
- Se o valor é 2 a poligonal produz um esboço escada. Dois pontos consecutivos são ligados por uma linha horizontal seguida de uma vertical.
- Se o valor é 3 a poligonal produz um esboço de barras. Para cada ponto (x,y) dado, uma linha vertical é desenhada de (x,y) a (x,0).

- Se o valor é 4 flechas são desenhadas entre dois pontos consecutivos.
- Se o valor é 5 a poligonal é preenchida (remendos).
- Se o valor é 6 a poligonal é um objeto barra como no Matlab. As propriedades `bar_shift` e `bar_width` comandam sua aparência.

foreground:
este campo contém o valor padrão da propriedade `foreground` (primeiro plano) utilizada para desenhar a poligonal. O valor deve ser um índice de cor (realtivo ao mapa de cores corrente).

background:
este campo contém a cor usada para preencher o plano de fundo da poligonal. Deve ser um índice de cor (relativo ao mapa de cores corrente).

interp_color_vector:
este campo contém os índices das cores utilizadas para preencher a poligonal quando a propriedade `interp_color_mode` é ajustada para `"on"`. Define os intervalos dos índices do mapa de cores a serem utilizados para preencher cada segmento. Por exemplo, o primeiro segmento será preenchido por todas as cores cujo índice está entre os dois primeiros elementos do vetor. Só é aplicável se a poligonal é definida por 3 ou 4 pontos. Logo, o tamanho do vetor deve coincidir com esta dimensão.

interp_color_mode:
este modo determina se é utilizada gradação interpolada par preencher a poligonal: o valor pode ser `"on"` ou `"off"`. Note que `interp_color_vector` deve ser definido antes e ajustar esta propriedade para "on" (ver acima).

mark_mode:
este campo contém o valor padrão da propriedade `mark_mode`. Deve ser `"on"` (marcas desenhadas) ou `"off"` (nenhuma marca desenhada).

mark_style:
A propriedade `mark_style` é utilizada para selecionar o tipo de marca utilizada quando a propriedade `mark_mode` é `"on"`. O valor é um inteiro em [0 14] que significa: ponto, sinal de mais, cruz, estrela, rombo preenchido, rombo, triângulo para cima, triângulo para baixo, rombo mais, círculo, asterísco, quadrado, triângulo para direita, triângulo para esquerda e pentagrama.

mark_size_unit:
este campo contém o valor padrão da propriedade `mark_size_unit` property value. Se `mark_size_unit` for ajustado para `"point"`, então o valor de `mark_size` é diretamente dado em pontos. Quando `mark_size_unit` é ajustado para `"tabulated"`, `mark_size` é computado em relação ao array de tamanho de fonte: logo, seu valor deve ser um inteiro em [0 5] que significa 8pt, 10pt, 12pt, 14pt, 18pt e 24pt. Note que plot2d e funções puras do Scilab utilizam o modo `tabulated` como padrão; quando se utiliza a função plot, o modo `point` é automaticamente habilitado.

mark_size:
a propriedade `mark_size` é utilizada para selecionar o tipo de tamanho das marcas quando a propriedade `mark_mode` é `"on"`. O valor deve ser um inteiro entre 0 e 5 signficando 8pt, 10pt, 12pt, 14pt, 18pt e 24pt.

mark_foreground:
este campo contém o valor da propriedade `mark_foreground` que é a cor da borda das marcas. O valor deve ser um índice de cor (relativo ao mapa de cores corrente).

mark_background:
este campo contém o valor da propriedade `mark_background` que é a cor da face das marcas. O valor deve ser um índice de cor (relativo ao mapa de cores corrente).

x_shift:
: este campo contém a compensação computada por uma chamada à função bar (ou recomputada por uma chamada à função barhomogenize) e é usada para realizar uma boa representação de barras verticais. Note que esta compensação é levada em conta para todos os outros `polyline_style`. A unidade é expressa nas coordenadas do usuário.

y_shift:
: este campo contém a compensação computada por uma chamada à função bar (ou recomputada por uma chamada à função barhomogenize) e é usada para realizar uma boa representação de barras verticais. Note que esta compensação é levada em conta para todos os outros `polyline_style`. TA unidade é expressa nas coordenadas do usuário.

z_shift:
: este campo contém a compensação que o usuário pode especificar. Note que esta compensação é levada em conta para todos os `polyline_style`. A unidade é expressa nas coordenadas do usuário.

bar_width:
: este campo determina a largura da poligonal selecionada quando sua propriedade `polyline_style` é ajustada para modo de barra (caso 6) : a unidade é expressa nas coordenadas do usuário.

clip_state:
: este campo contém o valor da propriedade `clip_state` para a poligonal. O valor de clip_state pode ser :

- `"off"` significa que a poligonal não é recortada.
- `"clipgrf"` significa que a poligonal é recortada fora da caixa dos eixos.
- `"on"` significa que a poligonal é recortada fora do retângulo dado pela propriedade clip_box.

clip_box:
: este campo determina a propriedade `clip_box`. Por padrão seu valor é uma matriz vazia se a propriedade clip_state é "off". Em outros casos, o vetor `[x,y,w,h]` (ponto superior esquerdo, largura, altura) define as porções da poligonal a serem exibidas, contudo o valor da propriedade `clip_state` será alterado.

user_data:
: este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados da entidade poligonal e recuperá-la.

# Exemplos

```
a=get("current_axes")//obtendo o manipulador dos novos eixos criados
a.data_bounds=[-2,-2;2,2];

xpoly(sin(2*%pi*(0:5)/5),cos(2*%pi*(0:5)/5),"lines",0)
p=get("hdl"); //obtendo o manipulador da entidade corrente (aqui é a entidade p
p.foreground=2;
p.thickness=3;
p.mark_style=9;
d=p.data;d(1,:)=[0 0];p.data=d;
a.rotation_angles=[0 45];

p.data=[(-2:0.1:2)' sin((-2:0.1:2)*%pi)']
p.mark_mode="off";
p.polyline_style=3;
```

```
p.line_style=4;
```

## Ver Também

set, get, delete, xpoly, xfpoly, xpolys, xfpolys, graphics_entities

## Autor

Djalel ABDEMOUCHE

# Name

rainbowcolormap — Mapa de cores que varia como um de arco-íris

```
cmap=rainbowcolormap(n)
```

# Parâmetros

n
: inteiro >= 1, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`rainbowcolormap` computa um mapa de cores com `n` cores variando como um arco-íris.

# Exemplos

```
f = scf();
plot3d1();
f.color_map = rainbowcolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, springcolormap, summercolormap, whitecolormap, wintercolormap

# Name

rectangle_properties — Descrição de propriedades da entidade Rectangle (retângulo)

# Descrição

A entidade Rectangle é uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros para retângulos preenchidos ou não.

parent:
: esta propriedade contém o manipulador da raiz. A raiz de uma entidade Rectangle deve ser do tipo `"Axes"` ou `"Compound"`.

children:
: esta propriedade contém um vetor com os galhos do manipulador. Contudo, manipuladores rectangle não possuem galhos correntemente.

mark_mode:
: este campo contém o valor padrão da propriedade `mark_mode` para a poligonal. Deve ser `"on"` (marcas desenhadas) ou `"off"` (nenhuma marca desenhada).

mark_style:
: a propriedade `mark_style` é utilizada para selecionar o tipo de marca utilizada quando a propriedade `mark_mode` é `"on"`. O valor é um inteiro em [0 14] que significa: ponto, sinal de mais, cruz, estrela, rombo preenchido, rombo, triângulo para cima, triângulo para baixo, rombo mais, círculo, asterísco, quadrado, triângulo para direita, triângulo para esquerda e pentagrama.

mark_size_unit:
: este campo contém o valor padrão da propriedade `mark_size_unit` property value. Se `mark_size_unit` for ajustado para `"point"`, então o valor de `mark_size` é diretamente dado em pontos. Quando `mark_size_unit` é ajustado para `"tabulated"`, `mark_size` é computado em relação ao array de tamanho de fonte: logo, seu valor deve ser um inteiro em [0 5] que significa 8pt, 10pt, 12pt, 14pt, 18pt e 24pt. Note que xrect como padrão; quando se utiliza a função `tabulated` como padrão; quando se utiliza a função plot o modo `point` é automaticamente habilitado.

mark_size:
: A propriedade `mark_size` é utilizada para escolher o tamanho das marcas quando a propriedade `mark_mode` é `"on"`. Deve ser um valor inteiro em [0 5] que significa 8pt, 10pt, 12pt, 14pt, 18pt e 24pt.

mark_foreground:
: este campo contém o valor da propriedade `mark_foreground` que é a cor da borda das marcas. O valor deve ser um índice de cor (relativo ao mapa de cores corrente).

mark_background:
: este campo contém o valor da propriedade `mark_background` que é a cor da face das marcas. O valor deve ser um índice de cor (relativo ao mapa de cores corrente).

line_mode:
: este campo contém o valor padrão da propriedade `line_mode` para a poligonal. O valor pode ser `"on"` (linha desenhada) ou `"off"` (nenhuma linha desenhada).

fill_mode:
: Se o valor da propriedade `fill_mode` é `"on"`, o retângulo é preenchido com a cor de primeiro plano, `mark_mode` também deve ter valor `"off"`. Senão, o valor da propriedade é `"off"` apenas a forma o contorno do retângulo é desenhado utilizando-se a cor de primeiro plano.

line_style:
: a propriedade `line_style` deve ser um inteiro em [1 6]. 1 stands significa linha sólida, o restante seleciona tracejados.

thickness:
: este campo contém o valor padrão da propriedade `thickness` (espessura) da linha. Deve ser um inteiro positivo.

foreground:
: este campo contém a cor utilizada para desenhar a linha de contorno do retângulo. Seu valor deve ser um índice de cor (relativo ao mapa de cores corrente).

background:
: este campo contém a cor utilizada para preencher a parte interna do retângulo. Seu valor deve ser um índice de cor (relativo ao mapa de cores corrente).

data:
: esta propriedade retorna as coordenadas do ponto superior esquerdo do retângulo, sua altura e largura em unidades de dados dos eixos. O resultado é uma matriz `[xleft,yup,[zup],width,height]`

visible:
: este campo contém o valor da propriedade `visible` para a entidade. Pode ser `"on"` ou `"off"` . Por padrão, o retângulo é visível, a propriedade é `"on"`. Se `"off"` o retângulo não é exibido na tela.

clip_state:
: este campo contém o valor da propriedade `clip_state` para o retângulo. o valor de clip_state pode ser :

- `"off"` ignifica que o retângulo não é recortado.
- `"clipgrf"` significa que o retângulo é recortado fora da caixa dos eixos.
- `"on"` significa que o retângulo é recortado fora do retângulo dado pela propriedade clip_box.

clip_box:
: este campo determina a propriedade `clip_box` property. Por padrão seu valor é uma matriz vazia se a propriedade clip_state é "off". Em outros casos, o vetor `[x,y,w,h]` (ponto superior esquerdo, largura, altura) define as porções do retângulo a ser exibido, contudo o valor da propriedade `clip_state` será alterado.

user_data:
: este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados da entidade retângulo e recuperá-la.

# Exemplos

```
a=get("current_axes");//obtendo o manipulador dos novos eixos criados
a.data_bounds=[-2,-2;2,2];

xrect(-1,1,2,2)

r=get("hdl");//obtendo o manipulador da entidade corrente (aqui, a entidade Rec
r.type
r.parent.type
r.foreground=13;
r.line_style=2;
r.fill_mode="on";
r.background=color('red');
r.clip_box=[-1 1;1 1];
r.data(:,[3 4])=[1/2 1/2];
```

```
r.data(:,[1 2])=[1/2 1/2];
r.clip_state="off"
```

# Ver Também

set, get, delete, xrect, xfrect, xrects, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

relocate_handle — Movimentação de manipuladores dentro da hierarquia gráfica

```
relocate_handle( movedHandles, parent )
```

# Parâmetros

movedHandles
: vetor de manipuladores realocados

parent
: nova raiz dos manipuladores

# Descrição

A função relocate_handle permite mover manipuladores das suas localizações em uma hierarquia gráfica para outra localização. Todas as entidades são realocadas sob o mesmo manipulador raiz especificado pelo parâmetro **parent**.

Já que não são todos os manipuladores que são compatíveis com todos os outros, existem algumas restrições ao se realocar manipuladores. Por exemplo, não é permitido mover um manipulador Axes sob um manipulador Polyline. Para mais informações sobre compatibilidade veja a página graphics_entities.

Esta rotina é particularmente útil para se mover um objeto de uma entidade Axes para outra, ou para se move objetos Axes de uma entidade Figure para outra.

# Exemplos

```
x = 0:10 ;

// esboçando uma primeira poligonal
plot(x,x^2) ;
axes1 = gca() ;
poly1 = gce() ;

// esboçando uma segunda em outra janela
scf() ;
plot( x,x )  ;
axes2 = gca() ;
poly2 = gce()  ;
poly2bis = copy( poly2 ) ; // criando uma cópia da poligonal

// pondo ambas as poligonais na mesma janela
relocate_handle( poly2bis, axes1 ) ;
```

# Ver Também

graphics_entities, copy, delete, swap_handles

# Autor

Jean-Baptiste Silvy

# Name

replot — redesenha a janela de gráficos corrente com novas fronteiras

```
replot(rect,[handle])
```

# Parâmetros

rect
: vetor linha de tamanho 4.

handle
: argumento opcional. Manipulador(es) do tipo Axes para selecionar uma ou várias entidades Axes válidas. Disponível apenas no novo modo de gráficos.

# Descrição

`replot` é utilizado para redesenhar o conteúdo da janela de gráficos atual com novas fronteiras definidas por `rect=[xmin,ymin,xmax,ymax]`. Sob a sintaxe de gráficos antiga, funciona apenas com o driver `"Rec"`.

Sob o novo modo de gráficos, esta transformação pode ser aplicada a eixos específicos fornecidos por manipuladores gráficos Axes via o argumento handle. Se handle não for especificado, as novas fronteiras são aplicadas aos eixos correntes da figura corrente. A transformação muda os valores de `data_bounds` para estes eixos. Note que a propriedade de eixos `tight_limits` também deve ser ajustada para "on" para selecionar estritamente estas fronteiras (ver axes_properties).

# Exemplos

```
backupstyle='?'

// primeiro exemplo
x=[0:0.1:2*%pi]';
plot2d(x,sin(x))
replot([-1,-1,10,2])

// segundo exemplo
xdel(winsid());
plot() // esboçando demonstração
f=gcf();
replot([-1,-1,10,2],f.children(1)) // especificando o valor do manipulador dos
replot([-3,-2,8,4],f.children(2))
```

# Ver Também

xbasr, xbasc, clf

# Autor

J.Ph.C.

# Name

rgb2name — Retorna o nome de uma cor

```
names=rgb2name(r,g,b)
names=rgb2name(rgb)
```

# Parâmetros

r,g,b
: RGB (vermelho, verde, azul) valores inteiros de uma cor.

rgb
: vetor RGB de valores inteiros de uma cor.

names
: nomes da cor.

# Descrição

`rgb2name` retorna o nome da cor correspondente aos valores RGB dados pelo seu argumento. Um vetor de nomes de cores pode ser retornado se a cor possui mais de um nome. `r`, `g` e `b` devem ser inteiros entre 0 e 255 correspondentes aos componentes das cores vermelho, verde e azul. Como de uso, 0 significa intensidade nenhuma e 255 significa intensidade total da cor. Os valores RGB também podem ser fornecidos pelo vetor `[r,g,b]`.

Se não for encontrada nenhuma cor `[]` é retornado.

A lista de todas as cores conhecidas é dada por color_list.

# Exemplos

```
rgb2name(255,128,128)
rgb2name([255 215 0])
// retorna a cor de número 10 do mapa de cores corrente acha seu nome
cmap=get(gcf(),"color_map");
rgb2name(cmap(10,:)*255)
```

# Ver Também

color, color_list, name2rgb

# Name

rotate — rotação de um conjunto de pontos

```
xy1=rotate(xy,[theta,orig])
```

# Parâmetros

xy
: matriz de tamanho (2,.).

xy1
: matriz de tamanho (2,.).

theta
: real, ângulo em radianos; o valor padrão é 0.

orig
: centro de rotação; o valor padrão é [0;0].

# Descrição

`rotate` executa a rotação com o ângulo `theta`:

```
xy1(:,i) = M(theta) *xy (:,i) + orig
```

onde `M` é a matriz de rotação correspondente.

# Exemplos

```
xsetech([0,0,1,1],[-1,-1,1,1])
xy=[(0:0.1:10);sin(0:0.1:10)]/10;
for i=2*%pi*(0:10)/10,
  [xy1]=rotate(xy,i);
  xpoly(xy1(1,:),xy1(2,:),"lines")
end
```

# Name

rotate_axes — Rotação interativa de um manipulador Axes

```
rotate_axes()
rotate_axes(h)
```

# Parâmetros

h
: manipulador Axes ou Figure. Especifica sobre qual eixo se efetuará a rotação.

# Descrição

`rotate_axes` é utilizado para realizar uma rotação interativa sobre um objeto Axes. Quando a função é chamada, pede-se ao usuário que clique duas vezes na janela gráfica. O primeiro clique inicializa uma rotação e o segundo termina.

Se um manipulador Axes for especificado como argumento de entrada, a rotação será aplicada sobre ele. Se um manipulador Figure for especificado, o primeiro clique determina qual objeto Axes rotacionar. Se a função for chamada sem argumento, a rotação se aplicará sobre a Figura corrente.

# Exemplos

```
clf();
// criando dois pares de eixos em uma figura
subplot(2, 1, 1);
plot2d;
subplot(2, 1, 2);
plot3d;

// girando apenas o segundo
axes2 = gca();
rotate_axes(axes2);

// girando o selecionado
rotate_axes();
// ou
rotate_axes(gcf());
```

# Ver Também

zoom_rect, axes_properties

# Autor

Jean-Baptiste Silvy INRIA

# Name

rubberbox — Caixa tipo liga de borracha para seleção de um retângulo

```
[final_rect,btn]=rubberbox()
[final_rect,btn]=rubberbox(initial_rect)
[final_rect,btn]=rubberbox(edition_mode)
[final_rect,btn]=rubberbox(initial_rect, edition_mode)
```

# Parâmetros

initial_rect
: vetor de duas a quatro entradas. Com quatro entradas fornece o retângulo inicial definido por [x_esquerdo, y_topo, largura, altura]. Com duas entradas, largura e altura são supostas 0.

edition_mode
: um booleano, se edition_mode==%t; um aperto de botão seleciona a primeira quina, soltar seleciona a quina oposta. Se edition_mode== %f, aperto de botão ou clique seleciona a primeira quina, um clique é requerido para selecionar a quina oposta. O valor padrão é edition_mode=%f

final_rect
: um retângulo definido por [x_esquerdo, y_topo, largura, altura]

btn
: um inteiro, o número do botão do mouse clicado

# Descrição

`rubberbox(initial_rect)` trilha uma caixa liga de borracha na janela de gráficos corrente, seguindo o mouse. Quando um botão é clicado, `rubberbox` retorna a definição dos retângulos finais `final_Rect`. Se o argumento `initial_rect` estiver presente, o usuário deve clicar para fixar a posição da quina inicial.

# Exemplos

```
xsetech(frect=[0,0,100,100])
[x,y]=xclick();r=rubberbox([x;y;30;10])
xrect(r)
r=rubberbox()
```

# Ver Também

xrect, xrects, xclick, xgetmouse, dragrect

# Name

sca — Ajusta a entidade de eixos corrente

```
a=sca(a)
```

# Parâmetros

a
: o manipulador da entidade Axes (de eixos)

# Descrição

`sca(a)` é usado para ajustar a entidade `Axes` corrente (ver graphics_entities) como sendo aquela apontada pelo manipulador `a`. As funções de esboço geralmente utilizam a entidade de eixos corrente.

# Exemplos

```
clf()
a1=newaxes();
a1.axes_bounds=[0,0,1.0,0.5];
t=0:0.1:20;
plot(t,acosh(t),'r')
a2=newaxes();
a2.axes_bounds=[0,0.5,1.0,0.5];
x=0:0.1:4;
plot(x,sinh(x))

sca(a1); //tornando os primeiros eixos correntes
plot(t,asinh(t),'g')
legend(['acosh','asinh'])
sca(a2); //tornando os segundos eixos correntes
legend('sinh')
```

# Ver Também

subplot, gda, newaxes

# Autor

S. Steer, INRIA

# Name

scaling — transformação afim de um conjunto de pontos

```
xy1=scaling(xy,factor,[orig])
```

# Parâmetros

xy1
: matriz de tamanho (2,.).

xy
: matriz de tamanho (2,.).

factor
: escalar real, coeficiente da transformação linear.

orig
: vetor de deslocamento; o valor padrão é [0;0].

# Descrição

`scaling` executa uma transformação afim de um conjunto de pontos definidos pelas coordenadas `xy`:

`xy1(:,i) = factor * xy(:,i) + orig`.

# Name

scf — Ajusta a (janela) de figura gráfica corrente

```
f = scf()
f = scf(h)
f = scf(num)
```

# Parâmetros

h
: o manipulador da figura

num
: identificador da figura

f
: manipulador da figura corrente

# Descrição

A figura corrente é o destino do desenho gráfico. A função `scf` permite modificar esta figura corrente ou criar uma caso ainda não exista.

`scf(num)` ajusta a figura com `figure_id==num` como figura corrente. Se ainda não existir, ela é criada.

`scf(h)` ajusta a figura apontada pelo manipulador `h` ajusta a figura apontada pelo manipulador

`scf()` é equivalente a `scf(max(winsid())+1)` . Pode ser utilizado para criar uma nova janela de gráficos.

# Exemplos

```
f4=scf(4); //criando figura com id==4 e a torna a figura corrente
f0=scf(0); //criando figura com id==0 e a torna a figura corrente
plot2d() //desenhando na figura corrente (id=0)
scf(f4); // ajustando a primeira figura criada como sendo a corrente
plot3d() //desenhando na figura corrente (id=4)
```

# Ver Também

set, get, gcf, clf, get_figure_handle, graphics_entities

# Autor

S. Steer INRIA

# Name

sd2sci — estrutura gr_menu para conversor de instrução Scilab

```
txt=sd2sci(sd [,sz [,orig]])
```

# Parâmetros

sd
: estrutura de dados construída por gr_menu.

sz
: vetor de números ou strings com dois componentes, fornece os fatores de zoom x e y

orig
: vetor de números ou strings, fornece o vetor translação da origem

# Descrição

Dada uma estrutura de dados `sd` gerada por `gr_menu` `sd2sci` forma um vetor de instruções do Scilab correspondente ao gráfico editado por `gr_menu`.

Os parâmetros opcionais `sz` e `orig` permitem ampliar e deslocar o gráfico inicial.

Se `sz` ou `orig` são fornecidos por strings, as instruções geradas são relativas a expressões formais.

# Ver Também

execstr

# Autor

Serge Steer INRIA 1988

# Name

sda — Ajusta eixos padrões

```
sda()
a = gda(); set(a,"default_values",1)
```

# Parâmetros

a
: manipulador dos eixos padrões

# Descrição

Esta rotina reajusta o os eixos modelos para valores padrões.

# Exemplos

```
x=[0:0.1:2*%pi]';
f=get("default_figure"); // obtendo o manipulador da figura modelo
a=get("default_axes");   // obtendo o manipulador dos eixos modelo
                         // ajustando suas propriedades
f.figure_size=[1200 900];
f.figure_position=[0 0];
a.background=4;
a.box="off";
a. tics_color=5;
a.labels_font_color=25;
a.labels_font_size=4;
a.sub_tics=[7 3];
a.x_location="middle";
a.y_location="middle";
a.tight_limits="on";
a.thickness=2;
a.grid=[-1 24];
subplot(221);
plot2d(x-2,sin(x))
subplot(222);
plot2d(x-6,[2*cos(x)+.7 2*cos(x)+.9 cos(2*x) .2+sin(3*x)],[-1,-2,-3 -4])
sda() // obtendo os valores padrões do modelo dos eixos
subplot(223);
plot2d(x-2,sin(x))
subplot(224);
plot2d(x-6,[2*cos(x)+.7 2*cos(x)+.9 cos(2*x) .2+sin(3*x)],[-1,-2,-3 -4])
xdel(0)
plot2d(x-2,sin(x))
```

# Ver Também

sdf, gda, gdf, set, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

sdf — Ajusta figura padrão

```
sdf()
f = gdf(); set(f,"default_values",1)
```

# Parâmetros

f
: manipulador da figura padrão.

# Descrição

Esta rotina reajusta a figura modelo para valores padrões.

# Exemplos

```
x=[0:0.1:2*%pi]';
f=get("default_figure"); // obtendo o manipulador da figura modelo
a=get("default_axes");   // obtendo o manipulador dos eixos modelos
                         // ajustando suas propriedades
f.background=4;
f.auto_resize="off";
f.figure_size=[400 300];
f.axes_size=[600 400];
f.figure_position=[0 -1];
a.x_location="top";
a.y_location="left";
for (i=1:6)
  xset("window",i) // criando figura com identificador i
  plot2d(x,[sin(x) cos(x)],[i -i])
  xclick();
  if i == 4, sdf(); end // obtendo os valores padrões da figura modelo
 end
```

# Ver Também

sda, gdf, gda, set, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

secto3d — Conversão de superfícies 3d

```
[m[,x]]=secto3d(seclist,npas)
[m]=secto3d(seclist ,x)
```

# Parâmetros

seclist
: uma lista cujos elementos são matrizes (2,.)

npas
: um inteiro

m
: uma matriz

x
: um vetor

# Descrição

Considerando uma superfície dada através de uma lista `seclist` de secções no plano `(x,z)`, `[m[,x]]=secto3d(seclist [,npas])` retorna uma matriz `m` que contém a discretização regular da superfície.

- A i-ésima linha da matriz m corresponde à i-ésima secção
- A j-ésima coluna da matriz m corresponde ao x(j)

Cada secção `seclist(i)` é descrita por uma matriz (2,.) que fornece respectivamente as coordenadas x e z de pontos.

`[m]=secto3d(seclist ,x)`neste caso o vetor x fornece a discretização do eixo x para todas as secções.

# Ver Também

plot3d

# Autor

Steer S.; ;

# Name

segs_properties — Descrição das propriedades da entidade Segments (de segmentos)

# Descrição

A entidade Segments é uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros para um conjunto de segmentos ou setas coloridas.

parent:
esta propriedade contém o manipulador da raiz. A raiz de uma entidade de Segments deve ser do tipo `"Axes"` ou `"Compound"`.

children:
esta propriedade contém um vetor com os galhos do manipulador. Contudo, manipuladores Segs não possuem galhos correntemente.

visible:
este campo contém o valor da propriedade `visible` para a entidade. Pode ser `"on"` ou `"off"` . Por padrão, o os segmentos são visíveis, a propriedade é `"on"`. Se `"off"` os segmentos não são exibidos na tela.

data:
este campo é uma matriz de duas (três) `[x,y,[z]]` que fornece as coordenadas das extremidades dos segmentos. Se `xv=matrix(x,2,-1)` e `yv=matrix(y,2,-1)` então `xv(:,k)` e `yv(:,k)` são as extremidades dos segmentos de número `k`.

line_mode:
este campo contém o valor padrão da propriedade `line_mode` para os segmentos. O valor pode ser `"on"` (linha desenhada) ou `"off"` (nenhuma linha desenhada).

line_style:
O valor da propriedade `line_style` deve ser um inteiro em [0 6]. 0 significa linha sólida, o restante seleciona tracejados. Esta propriedade se aplica a todos os segmentos.

thickness:
este campo contém o valor da propriedade `thickness` (espessura) para todos os segmentos. Seu valor deve ser um inteiro não-negativo.

arrow_size:
fator que especifica o tamanho das cabeças das setas. Com um valor negativo, o tamanho também depende do comprimento da seta. Para desenhar um segmento, o valor deve ser ajustado para 0.

segs_color:
este campo especifica a cor a ser utilizada para desenhar cada segmento. Cada elemento é um índice de cor relativo ao mapa de cores corrente.

mark_mode:
este campo contém o valor padrão da propriedade `mark_mode` para a poligonal. Deve ser `"on"` (marcas desenhadas) ou `"off"` (nenhuma marca desenhada)

mark_style:
a propriedade `mark_style` é utilizada para selecionar o tipo de marca utilizada quando a propriedade `mark_mode` é `"on"`. O valor é um inteiro em [0 14] que significa: ponto, sinal de mais, cruz, estrela, rombo preenchido, rombo, triângulo para cima, triângulo para baixo, rombo mais, círculo, asterísco, quadrado, triângulo para direita, triângulo para esquerda e pentagrama.

mark_size_unit:
este campo contém o valor padrão da propriedade `mark_size_unit`. Se `mark_size_unit` for ajustado para `"point"`, então o valor de `mark_size` é diretamente dado em pontos. Quando

`mark_size_unit` é ajustado para `"tabulated"`, `mark_size` é computado em relação ao array de tamanho de fonte: logo, seu valor deve ser um inteiro em [0 5] que significa 8pt, 10pt, 12pt, 14pt, 18pt e 24pt. Note que plot2d e funções puras do Scilab utilizam o modo `tabulated` como padrão; quando se utiliza a função plot, o modo `point` é automaticamente habilitado.

mark_size:
a propriedade `mark_size` é utilizada para selecionar o tipo de tamanho das marcas quando a propriedade `mark_mode` é `"on"`. O valor deve ser um inteiro entre 0 e 5 signficando 8pt, 10pt, 12pt, 14pt, 18pt e 24pt.

mark_foreground:
este campo contém o valor da propriedade `mark_foreground` que é a cor da borda das marcas. O valor deve ser um índice de cor (relativo ao mapa de cores corrente).

mark_background:
este campo contém o valor da propriedade `mark_background` que é a cor da face das marcas. O valor deve ser um índice de cor (relativo ao mapa de cores corrente).

clip_state:
este campo contém o valor da propriedade `clip_state` para os segmentos. O valor de clip_state pode ser:

- `"off"` significa que os segmentos não são recortados.
- `"clipgrf"` significa que os segmentos são recortados fora da caixa dos eixos.
- `"on"` significa que os sementos são recortada fora do retângulo dado pela propriedade `clip_box`.

clip_box:
este campo determina a propriedade `clip_box`. Por padrão seu valor é uma matriz vazia se a propriedade clip_state é "off". Em outros casos, o vetor `[x,y,w,h]` (ponto superior esquerdo, largura, altura) define as porções dos segmentos a serem exibidas, contudo o valor da propriedade `clip_state` será alterado.

user_data:
este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados da entidade de segmentos e recuperá-la.

# Exemplos

```
a=get("current_axes");//obtendo o manipulador dos novos eixos criados
a.data_bounds=[-10,-10;10,10];
x=2*%pi*(0:7)/8;
xv=[2*sin(x);9*sin(x)];
yv=[2*cos(x);9*cos(x)];
xsegs(xv,yv,1:8)

s=a.children
s.arrow_size=1;
s.segs_color=15:22;
for j=1:2
  for i=1:8
    h=s.data(i*2,j);
    s.data(i*2,j)=s.data(i*2-1,j);
    s.data(i*2-1,j)=  h;
  end
end
```

```
s.segs_color=5; //ajustando todas as cores para 5

s.clip_box=[-4,4,8,8];
a.thickness=4;
xrect(s.clip_box);
```

## Ver Também

set, get, delete, xsegs, graphics_entities

## Autor

Djalel ABDMOUCHE

# Name

set — Ajusta um valor de propriedade de uma objeto entidade gráfica ou de um objeto Interface do Usuário (User Interface)

```
set(prop,val)
set(h,prop)
set(h,prop,val)
h.prop=val
```

# Parâmetros

h
: manipulador da entidade da qual a propriedade nomeada se deseja ajustar, `h` pode ser um vetor de manipuladores, em tal caso, serão ajustados valores de propriedades para todos os objetos identificados por h

prop
: string, nome da propriedade a ser ajustada.

val
: valor a ser dado à propriedade

# Descrição

Esta rotina pode ser utilizada para se recuperar o valor de uma propriedade especificada de uma entidade de gráficos ou objeto GUI. Neste caso, é equivalente a se usar o operador ponto ('.') em um manipulador. Por exemplo, `set(h,"background",5)` é equivalente a `h.background = 5`.

Nomes de propriedades são strings. O tipo do valor ajustado dependo do tipo do manipulador e da propriedade.

Para obter a lista de todas as propriedade existentes ver graphics_entities ou uicontrol para objetos de Interface do Usuário

`set` também pode ser chamado com apenas uma propriedade como argumento. Neste caso, a propriedade deve ser uma das seguintes:

current_entity or hdl
: `set('current_entity',h)` ou `set('hdl',h)` ajusta uma nova entidade como a corrente. Neste caso, o valor deve ser um manipulador gráfico.

current_figure
: `set('current_figure',fig)` ajusta uma nova figura como a corrente. É equivalente a scf. Neste caso, o valor deve ser um manipulador FIgure.

current_axes
: `set('current_axes',axes)` ajusta novos eixos como os correntes. É equivalente a sca. Neste caso, o valor deve ser um manipulador Axes.

`set` também pode ser chamado com manipuladores gráficos e com propriedades como argumentos. O manipulador deve ser de uma figura padrão ou de eixos padrões. A propriedade deve ser `"default_values"`. Neste caso, a entidade padrão é reajustada para os valores que possuía na inicialização do Scilab. `set("default_values",h)` equivale a sda ou sdf.

# Exemplos

```
clf()
set("auto_clear","off") ;
// Exemple of a Plot 2D
x=[-.2:0.1:2*%pi]';
plot2d(x-.3,[sin(x-1) cos(2*x)],[1 2] );
a=get("current_axes");
p1=a.children.children(1);
p2=a.children.children(2);
// ajustando as propriedades nomeadas para os valores especificados nos objetos
set(p2,"foreground",13);
set(p2,"polyline_style",2);
set(a,'tight_limits',"on");
set(a,"box","off");
set(a,"sub_tics",[ 7 0 ]);
set(a,"y_location","middle")
set(p2,'thickness',2);
set(p1,'mark_mode',"on");
set(p1,'mark_style',3);
plot2d(x-2,x.^2/20);
p3= a.children(1).children;
set([a p1 p2 p3],"foreground",5)
```

# Ver Também

get, delete, copy, move, graphics_entities, uicontrol

# Autor

Djalel ABDEMOUCHE

# Name

seteventhandler — Define um gerenciador de eventos para uma janela gráfica

```
seteventhandler(sfun_name)
seteventhandler('')
```

# Parâmetros

sfun_name
: string. O nome da função do Scilab com a qual se pretende gerenciar os eventos

# Descrição

A função permite que o usuário defina um gerenciador de eventos particular para janela gráfica corrente. `seteventhandler('')` remove o gerenciador.

Para mais informações sobre o gerenciador de eventos, veja a página event handler functions.

# Exemplos

```
function my_eventhandler(win,x,y,ibut)
  if ibut==-1000 then return,end
  [x,y]=xchange(x,y,'i2f')
  xinfo(msprintf('Evento de código %d na posição do mouse é (%f,%f)',ibut,x,y))
endfunction

plot2d()
seteventhandler('my_eventhandler')
//agora:
//   -  mova o mouse sobre a janela gráfica
//   -  pressione e libere as teclas com shift pressionado ou não e com Ctrl pr
//   -  pressione um botão, espere um pouco e libere
//   -  pressione e libere um botão
//   -  clique duas vezes em um botão

seteventhandler('') //suprimindo o gerenciador de eventos
```

# Ver Também

addmenu, xgetmouse, xclick, xchange, event handler functions, figure_properties

# Name

show_pixmap — Envia o buffer de pixmap à tela

```
show_pixmap()
```

# Descrição

Se a propriedade `pixmap` de uma janela gráfica é `"on"` os desenhos são enviados à memória pixmap ao invés da tela de exibição

A instrução `show_pixmap()` envia o pixmap à tela.

O modo pixmap pode ser usado para se obter animações suaves. Esta propriedade pode ser encontrada entre os campos de entidades da figura (ver figure_properties).

# Exemplos

```
f=gcf();f.pixmap='on'; //ajustando o modo pixmap
a=gca();a.data_bounds=[0 0; 10 10];
//construindo dois retângulos
xrects([0;10;1;1],5);r1=gce();r1=r1.children;
xrects([0;1;1;1],13);r2=gce();r2=r2.children;
//laço de animação
for k=1:1000
  //desenhando os retângulos no buffer pixmap
  move(r1,[0.01,-0.01]);move(r2,[0.01,0.01])
  //exibindo o buffer pixmap
  show_pixmap()
end
```

# Ver Também

figure_properties, clear_pixmap

# Autor

Serge Steer INRIA

# Name

show_window — Restaura uma janela de gráficos

```
show_window([figure])
```

# Parâmetros

figure
: número ou manipulador da figura a ser exibida

# Descrição

Sem parâmetros, `show_window` restaura a janela de gráficos corrente, mesmo se estiver iconificada. Em caso contrário, restaura a janela especificada por seu número ou por seu manipulador. Se nenhuma janela existir, uma é criada.

# Autor

J.Ph.C.

# Name

springcolormap — Mapa de cores com tons da primavera (magenta, amarelo)

```
cmap=springcolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`springcolormap` computa um mapa de cores com `n` cores variando de magenta a amarelo.

# Exemplos

```
f = scf();
plot3d1();
f.color_map = springcolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, summercolormap, whitecolormap, wintercolormap

# Name

square — ajusta escalas para esboço isométrico (muda o tamanho da janela)

```
square(xmin,ymin,xmax,ymax)
```

# Parâmetros

xmin,xmax,ymin,ymax
: quatro valores reais

# Descrição

`square` é usado para definir escalas isométricas nos eixos x e y. Os valores requeridos `xmin, xmax, ymin, ymax` são os limites da moldura de gráficos e `square` muda as dimensões da janela de gráficos para se obter um esboço isométrico. `square` ajusta as escalas gráficas correntes e pode ser usado em conjunção com rotinas de gráfico que requerem a escala de gráficos corrente (por exemplo `fstrf="x0z"` em `plot2d`).

# Exemplos

```
t=[0:0.1:2*%pi]';
plot2d(sin(t),cos(t))
xbasc()
square(-1,-1,1,1)
plot2d(sin(t),cos(t))
xset("default")
```

# Ver Também

isoview, xsetech

# Autor

Steer S.

# Name

stringbox — Computa o retângulo de fronteira de um texto ou rótulo.

```
rect = stringbox( string, x, y, [angle, [fontStyle, [fontSize]]] )
rect = stringbox( Handle )
```

# Parâmetros

rect
: uma matriz 2x4 contendo as quatro coordenadas dos vértices do retângulo de fronteira.

string
: matriz de strings a ser encerrada

x,y
: escalares reais, coordenadas do ponto inferior esquerdo dos strings.

angle
: ângulo de rotação dos strings em sentido horário e em graus ao redor do ponto `(x,y)`.

fonStyle
: inteiro especificando o tipo da fonte.

fontSize
: inteiro especificando o tamanho da fonte.

Handle
: um manipulador gráfico do tipo Text ou Label.

# Descrição

`stringbox` retorna os vértices de um retângulo de fronteira de um texto ou rótulo ou um string que será exibido de certo modo. As coordenadas são dadas com a escala gráfica corrente. O primeiro vértice corresponde às coordenadas do texto `(x,y)`, ponto superior esquerdo sem rotação, os vértices seguintes são dados em sentido horário na matriz resultante.

O resultado pode ser impreciso com um driver Postscript.

# Exemplos

```
// exibindo eixos
axes = gca() ;
axes.axes_visible = 'on' ;
axes.data_bounds = [ 1, 1 ; 10, 10 ] ;

// exibindo rótulos para eixos
xtitle( 'stringbox', 'X', 'Y' )  ;

// obtendo a caixa de fronteixa do rótulo X
stringbox( axes.x_label )

// imprimindo um string
str = [ "Scilab", "não" , "é", "Skylab" ] ;
xstring( 4, 9, str ) ;

//modificando o texto
```

```
e = gce() ;
e.font_angle = 90 ;
e.font_size    = 6  ;
e.font_style   = 7 ;
e.box = 'on' ;

// obtendo sua caixa de fronteira
stringbox( e )
// ou
rect = stringbox( str, 4, 9, 90, 7, 6 )

// clicando e descobrindo se o texto foi acertado
hit = xclick()  ;
hit = hit( 2 : 3 ) ;

if hit(1) >= rect(1,1)  & hit(1) <=  rect(1,2) & hit(2) <= rect(2,2) & hit(2) >=
  disp('Você acertou o texto.') ;
else
  disp('Você errou.')
end;
```

# Ver Também

xstring, xstringl, xstringb

# Autor

Jean-Baptiste Silvy

# Name

subplot — divide uma janela de gráficos em uma matriz de sub-janelas

```
subplot(m,n,p)
subplot(mnp)
```

# Parâmetros

m,n,p
: inteiros positivos

mnp
: um inteiro com notação decimal `mnp`

# Descrição

subplot(m,n,p) ou subplot(mnp) parte a janela de gráficos em uma matriz m-por-n de sub-janelas e seleciona a p-ésima sub-janela para esboçar o gráfico corrente. O número uma sub-janela nas matrizes é contado linha por linha, i.e. a sub-janela correspondente ao elemento (i,j) da matriz tem número (i-1)*n + j.

# Exemplos

```
subplot(221)
plot2d()
subplot(222)
plot3d()
subplot(2,2,3)
param3d()
subplot(2,2,4)
hist3d()
```

# Ver Também

plot2d, plot3d, xstring, xtitle

# Name

summercolormap — Mapa de cores com tons do verão (verde, amarelo)

```
cmap=summercolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`summercolormap` computa um mapa de cores com `n` cores variando do verde para o amarelo.

# Exemplos

```
f = scf();
plot3d1();
f.color_map = summercolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, whitecolormap, wintercolormap

# Name

surf — Esboço de superfície 3d

```
surf(Z,<GlobalProperty>)
surf(Z,color,<GlobalProperty>)
surf(X,Y,Z,<color>,<GlobalProperty>)
surf(<axes_handle>,...)
```

# Parâmetros

Z
: uma matriz real definindo as alturas da superfície. Não pode ser omitida. O dado Z é uma matriz `mxn`.

X,Y
: dois vetores ou matrizes reais: sempre ajustados juntos, estes dados definem um novo grid padrão. Estes novos componentes `X` e `Y` do grid devem coincidir com as dimensões de `Z` (ver a descrição abaixo).

color
: um matriz de reais opcional definindo valores de cores para cada ponto `(X(j),Y(i))` do grid (ver descrição abaixo).

<GlobalProperty>
: esta propriedade rerpresenta uma seqüência de pares de sentenças `{PropertyName,PropertyValue}` que definem propriedades globais de objetos aplicadas a todas as curvas criadas neste esboço. Para uma completa visualização das propriedades disponíveis veja GlobalProperty.

<axes_handle>
: este argumento opcional força os eixos a aparecerem dentro dos eixos selecionados dados por `axes_handle` ao invés dos eixos correntes (ver gca).

# Descrição

`surf` desenha uma superfície parametrizada colorida utilizando um grid retangular definido pelas coordenadas `X` e `Y` (se `{X,Y}` não forem especificados, este grid é determinado utilizando-se as dimensões da matriz `Z`) ; em cada ponto deste grid, uma coordenada Z é dada utilizando-se uma matriz `Z` (apenas dados obrigatórios). `surf` foi criada para lidar melhor com a sintaxe do Matlab. Para melhorar a compatibilidade gráfica, usuários do Matlab devem utilizar `surf` (ao invés de plot3d).

Especificação de entrada de dados:

Neste parágrafo, e para sermos mais claros, não mencionaremos os argumentos opcionais de `GlobalProperty` ojá que eles não interferem na entrada de dados (exceto pelas propriedades `"Xdata"`, `"Ydata"` e `"Zdata"`, ver GlobalProperty). É assumido que todos estes argumentos opcionais podem estar presentes também.

Se `Z` é a única matriz especificada, surf(Z) esboça a matriz `Z` versus o grid definido por `1:size(Z,2)` ao longo do eixo x e `1:size(Z,1)` ao longo do eixo.

Se uma tripla `{X,Y,Z}` for fornecida, `Z` deve ser uma matriz de size(`Z`)= [`mxn`], `X` or `Y` can be :

- a) um vetor: se `X` for um vetor, length(`X`)=`n`. Respectivamente, se `Y` for um vetor, length(`Y`)=`m`.

  b) uma matriz: neste caso, size(`X`) (ou size(`Y`)) deve ser igual a size(`Z`).

Especificação de entrada de cores:

Como citado acima, a superfície é criada sobre um gride retangular de apoio. Consideremos duas variáveis independentes `i` e `j` tais que:

```
a) 1<=i<=m and 1<=j<=n.

b) i-1,j-1 ---- i-1,j ---- i-1,j+1 --.. |

    |            |            |         | i direction

  i,j-1  ----  i,j  -----  i,j+1 --..  V

    |            |            |

    :            :            :

                              -------- > j direction
```

Este grid retangular imaginário é utilizado para se contruir o suporte da superfície real sobre o plano `XY` . De fato, os dados `X`,`Y` e `Z` possuem o mesmo tamanho (mesmo se `X` ou `Y` for um vetor, ver abaixo) e podem ser considerados como três funções `x(i,j)`, `y(i,j)` e `z(i,j)` especificando a superficie desejada. Se `X` ou `Y` forem vetores, eles são internamente tratados para produzir boas matrizes de dimensões correspondentes às dimensões da matriz `Z` (e o grid é forçosamente uma região retangular).

Considerando as três funções `x(i,j)`, `y(i,j)` e `z(i,j)`, a porção da superfície definida entre dois `i` e `j` consecutivos é chamada remendo.

Por padrão, quando nenhuma matriz de cores é adicionada a uma chamada a surf, o parâmetro de cor é ligado o dado `Z` . Quando uma matriz de cores `color` é dada, pode ser aplicada aos remendos de duas formas diferentes: nos vértices, ou no centro de cada remendo.

É por isto que, se `Z` é uma matriz [`m`x`n`] , a dimensão da matriz `color C` pode ser [`m`x`n`] (uma cor definida por vértice) ou [`m-1`x`n-1`] (uma cor definida por remendo).

A representação de cores também varia quando se especifica algumas GlobalProperties (propriedades globais):

A propriedade `FaceColor` ajusta o padrão de gradação: pode ser `'interp'` ou `'flat'` (modo padrão). Quando `'interp'` é selecionado, é realizado uma interpolação bilinear de cores no remendo. Se size(`C`) é igual a size(`Z`)-1 (i.e. for fornecida apenas uma cor ao remendo) então a cor dos vértices definindo o remendo é ajustada para a dada cor do remendo.

Quando `'flat'` (modo padrão) é habilitado, é utilizada uma representação de cores de facetadas (uma cor por remendo). Se size(`C`) é igual a size(`Z`) (i.e. foi fornecida apenas uma cor por os vértices), as últimas linha e coluna de `C` são ignoradas.

Os argumentos `GlobalProperty` devem ser usados para customizar a superfície. Aqui está uma breve descrição de como funcionam:

GlobalProperty
: esta opção pode ser utilizada para especificar como todas as superfícies serão desenhadas. Deve sempre ser um par de sentenças constituidos de um string definindo `PropertyName`, (nome da propriedade) e seu valor associado `PropertyValue` (que pode ser um inteiro ou outra coisa... dependendo do tipo de `PropertyName`). Note que você pode ajustar múltiplas propriedades : a cor da face e da borda, dados de cores, mapeamento de dados de cores, cor do marcador (plano

de fundo e primeiro plano), visibilidade, recorte, espessura das bordas da superfície... (ver GlobalProperty )

Note que todas essas propriedades podem ser (re-)ajustadas através das propriedades da entidade superfície (ver surface_properties).

# Observações

Por padrão, esboços sucessivos de superfície são superpostos. Para limpar o esboço anteirior, use `clf()`. Para habilitar o modo `auto_clear` (limpeza automática) como modo padrão, edite seus eixos padrões fazendo:

da=gda();

da.auto_clear = 'on'

Entre com o comando `surf` para visualizar uma demonstração.

# Exemplos

```
// inicialização de Z

Z= [   0.0001    0.0013    0.0053   -0.0299   -0.1809   -0.2465   -0.1100   -0.
    0.0005    0.0089    0.0259   -0.3673   -1.8670   -2.4736   -1.0866   -0.160
    0.0004    0.0214    0.1739   -0.3147   -4.0919   -6.4101   -2.7589   -0.277
   -0.0088   -0.0871    0.0364    1.8559    1.4995   -2.2171   -0.2729    0.836
   -0.0308   -0.4313   -1.7334   -0.1148    3.0731    0.4444    2.6145    2.441
   -0.0336   -0.4990   -2.3552   -2.1722    0.8856   -0.0531    2.6416    2.406
   -0.0137   -0.1967   -0.8083    0.2289    3.3983    3.1955    2.4338    1.212
   -0.0014   -0.0017    0.3189    2.7414    7.1622    7.1361    3.1242    0.663
    0.0002    0.0104    0.1733    1.0852    2.6741    2.6725    1.1119    0.197
    0.0000    0.0012    0.0183    0.1099    0.2684    0.2683    0.1107    0.019

//superfície simples
surf(Z); // note que X e Y são determinados pelas dimensões de Z

//a mesma superfície com faces vermelhas e bordas azuis
scf(2); // nova figura de número 2
surf(Z,'facecol','red','edgecol','blu")

// inicialização de X e Y
// NB: aqui, X tem as mesmas linhas e Y tem as mesmas colunas
X = [ -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666

Y= [   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000   -3
   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333   -2.333
   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667   -1.666
   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.000
```

```
   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333   -0.333
    0.3333    0.3333    0.3333    0.3333    0.3333    0.3333    0.3333    0.333
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.000
    1.6667    1.6667    1.6667    1.6667    1.6667    1.6667    1.6667    1.666
    2.3333    2.3333    2.3333    2.3333    2.3333    2.3333    2.3333    2.333
    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.000

// exemplo 1
scf(3)
surf(X,Y,Z)

//exemplo 2
// como você pode ver, o grid não é necessariamente retangular
scf(4)
X(1,4) = -1.5;
Y(1,4) = -3.5;
Z(1,4) = -2;
surf(X,Y,Z)

// exemplo 3
// X e Y são vetores => mesmo comportamento que a amostra 1
// com vetores, o grid é inevitavelmente retangular
scf(5)// nova figura de número 5
X=[ -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.66
Y=X;
surf(X,Y,Z)


//exemplos com LineSpec e GlobalProperty:
xdel(winsid()) // destruindo todas as figuras existentes
surf(Z,Z+5) // array de cores especificado
e=gce();
e.cdata_mapping='direct' // o padrão é 'scaled' relativo ao mapa de cores color
e.color_flag=3; // modo de gradação interpolada. O padrão é 4 (modo 'flat') par

scf(2)
surf(X,Y,Z,'colorda',ones(10,10),'edgeco','cya','marker','penta','markersiz',20

scf(3)
surf(Z,'cdatamapping','direct')
scf(4)
surf(Z,'facecol','interp') // interpolated shading mode (color_flag == 3)

scf(10)
axfig10=gca();
scf(11);
surf(axfig10,Z,'ydat',[100:109],'marker','d','markerfac','green','markeredg','y

xdel(winsid())
```

# Ver Também

plot2d, clf, xdel, delete, LineSpec, GlobalProperty

# Autor

F.Leray

# Name

surface_properties — Descrição das propriedades da entidade Surface (superfície)

# Descrição

A entidade Surface é uma folha das hierarquia de entidades gráficas. Duas classes aparecem sob este tipo de entidade : `Plot3d` e `Fac3d` de acordo com a função de esboço ou com o modo como os dados foram inseridos. As entidades `Fac3d` e `Plo3d` são semelhantes, mas `Fac3d` é mais completa e aceita mais opções que `Plot3d`. Para sempre ter entidades `Fac3d` simplesmente use `genfac3d` para pré-construir matrizes antes de usar os comandos `plot3d` ou use o comando `surf`.

Aqui estão as propriedades contidas em uma entidade superfície:

parent:
: esta propriedade contém o manipulador da raiz. A raiz de uma entidade Surface pode ser `"Axes"` ou `"Compound"`.

children:
: esta propriedade contém um vetor com os galhos do manipulador. Contudo, manipuladores de superfície não possuem galhos correntemente.

visible:
: este campo contém o valor padrão da propriedade `visible` para a entidade. Pode ser `"on"` ou `"off"` . Por padrão, superfícies são visíveis , o valor da propriedade é `"on"`. Se for `"off"` os gráficos 3d não são exibidos na tela.

surface_mode:
: este campo contém o valor padrão da propriedade `surface_mode` para a superfície. Pode ser `"on"` (superfície desenhada) ou `"off"` (nenhuma superfície desenhada).

foreground:
: se `color_mode >= 0`, contém o índice da cor a ser usada nas bordas. Se não, foreground não é usado. O valor foreground deve ser um índice inteiro de cor (relativo ao mapa de cores corrente).

thickness:
: este campo contém o valor de `thickness` (espessura) das linhas usadas para desenhar facetas ou contornos. Deve ser um inteiro positivo.

mark_mode:
: este campo contém o valor padrão da propriedade `mark_mode` para a superfície. Seu valor pode ser `"on"` (marcas desenhadas) ou `"off"` (marcas não desenhadas).

mark_style:
: o valor da propriedade `mark_style` é usado para selecionar o tipo de marca usada quando a propriedade `mark_mode` é `"on"`. O valor deve ser um inteiro no intervalo [0 14] que significa: ponto, mais, cruz, estrela, rombo preenchido, rombo, triângulo para cima, triângulo para baixo, rombo mais, círculo, asterísco, quadrado, trângulo para direita, triângulo para esquerda e pentagrama.

mark_size_unit:
: este campo contém o valor padrão da propriedade `mark_size_unit`. Se `mark_size_unit` é ajustado para `"point"`, então o valor de `mark_size` é dado diretamente em pontos. Quando `mark_size_unit` é ajustado para `"tabulated"`, `mark_size` é computado de acordo com o array de tamanho de fonte: logo, seu valor deve ser um inteiro no intervalo [0 5] que significa 8pt, 10pt, 12pt, 14pt, 18pt e 24pt. Note que plot3d e funções puras do Scilab usam o modo `tabulated` como padrão; quando se utiliza as funções surf (ou plot para linhas 2d) o modo `point` é automaticamente habilitado.

mark_size:
a propriedade `mark_size` é utilizada para selecionar o tamanho de fonte das marcas quando a propriedade `mark_mode` está `"on"`. O valor deve ser um inteiro entre 0 e 5 que significa 8pt, 10pt, 12pt, 14pt, 18pt e 24pt.

mark_foreground:
este campo contém o valor padrão da propriedade `mark_foreground` que é a cor da borda das marcas. Seu valor deve ser um índice de cor (relativo ao mapa de cores corrente).

mark_background:
este campo contém o valor padrão da propriedade `mark_background` que é a cor da face das marcas. Seu valor deve ser um índice de cor (relativo ao mapa de cores corrente).

data:
este campo define uma estrutura de dados `tlist` do tipo "3d" composto de índices de uma linha e uma coluna de cada elemento como as coordenadas x, y e z contidas respectivamente em `data.x`,`data.y` e `data.z`. O campo complementar `data.color` está disponível caso um vetor ou matriz reais de cores seja especificado. Se nenhum for, `data.color` não é listado. A superfície é pintada de acordo com as propriedades `color_mode` e `color_flag`.

color_mode:
um inteiro no intervalo `[-size(colormap) ; size(colormap)]` definindo a cor da faceta quando o valor de `color_flag` é `0`. Como dito antes, se `color_mode > 0`, as bordas são desenhadas usando a cor `foreground`. Se `color_mode` é ajustado para `0`, uma malha das superfície é desenhada: faces da frente não têm cores. Finalmente, quando `color_mode < 0`, as faces da frente são pintadas com a cor `-color_mode` mas nenhuma borda é exibida.

color_flag:
este campo é usado para definir o algoritimo usado para ajustar as cores das facetas. Note que as regras em `color_mode`, `foreground` e `hiddencolor` ainda são aplicadas nesse caso.

- `color_flag == 0`
  - Todas as facetas são pintadas usando-se o índice de cores e o método definido por `color_mode` (ver acima).
- `color_flag == 1`
  - Todas as facetas são pintadas utilizando-se um índice de cor por faceta proporcional a `z`. O valor mínimo de `z` é pintado utilizando-se o índice 1, enquanto o valor máximo de `z` é pintado utilizando-se o índice mais alto. As bordas das facetas pode ser adicionalmente desenhadas dependendo do valor de `color_mode` (ver acima).
- Os três casos restantes (`color_flag== 2,3 or 4`) só estão disponíveis para a entidade `Fac3d` . Então, o valor de `data.color` é usado para ajustar cores para as facetas (índices do mapa de cores corrente) se existirem. Se não, o `color_mode` corrente é utilizado para pintar as facetas.
- `color_flag == 2 ('flat' shading)`
  - Todas as facetas são pintadas utilizando-se o índice de cor dado na propriedade `data.color` (uma cor por faceta é necessária). Dois são os casos possíveis:
  - `data.color` contém um vetor `color` vector : se `color(i)` for positivo, fornece a cor da faceta `i` e a borda da faceta é desenhada com o estilo de linha e cor correntes. Se `color(i)` for negativo, o a cor de identificador `-color(i)` é usado e a borda da faceta não é desenhada.

    `data.color` contém uma matriz de cores de tamanho (nf,n) onde `n` significa o número de facetas e `nf` o número de pontos definindo a faceta poligonal. Para os vértices `nf` definindo

cada faceta, o algoritmo computa o valor médio do índice de cores (da matriz de índices de cores) : os `nf` vértices da mesma faceta terão o mesmo valor de índice de cor.

- `color_flag == 3 ('interpolated' shading)`
  - Pinta as facetas por interpolação das cores dos vértices. Os índices das cores dos vértices são dados pela propriedade `data.color` (uma cor por vértice é necessária). Dois são os casos possíveis:
  - `data.color` contém um vetor `colors` : então há poucos dados para completar o modo de gradação interpolada. De fato, uma matriz de cores de tamanho (nf,n) (onde `n` é o número de facetas e `nf` o número de pontos definindo a faceta poligonal) é necessária para realizar esta operação. Para cada faceta, o algoritmo copia o único valor de índice de cor da faceta nos `nf` vértices de índices de cores definindo a borda da faceta.

    `data.color` contém uma matriz de cores de tamanho (nf,n) (ver acima para definições de `nf` e `n`), o modo de gradação interpolada pode ser completado normalmente utilizando-se estes índices de cores.
- `color_flag == 4 (Matlab-like 'flat' shading)`
  - É o mesmo que `color_flag==2` com uma pequena diferença quando `data.color` é uma matriz. Todas as facetas são pintadas com o índice de cor fornecido pela propriedade `data.color` (uma cor por faceta é necessária). Dois casos são então possíveis:
  - `data.color` contém um vetor `color` vector : se `color(i)` é positivo, fornece a cor da faceta `i` e a borda da faceta é desenhada com o estilo de linha e cor correntes. Se `color(i)` é negativo, a cor de identificador `-color(i)` é utilizado e a borda da faceta não é desenhada.

    `data.color` contém uma matriz de cores de tamanho (nf,n) onde `n` é o número de facetas e `nf` o número de pontos definindo a faceta poligonal. Para os `nf` vértices definindo cada faceta, o algoritimo toma a cor do primeiro vértice definindo o retículo (faceta).

cdata_mapping:
: `específico para manipuladores Fac3d`. Um string definindo um valor `'scaled'` ou `'direct'`. Se um `data.color` é definido, cada dado de índice de cor especifica um valor único para vértice. `cdata_mapping` determina se estes índices estão em escala para serem mapeados linearmente no mapa de cores corrente (modo `'scaled'`) ou aponta diretamente para o mapa de cores (modo `'direct'`). Esta propriedade é útil quando `color_flag` é igual a `2`,`3` ou `4`.

hiddencolor:
: esta propriedade contém o índice de cor usado para desenhar as faces atrás de uma superfície. Seu valor deve ser um inteiro positivo (índice de cores relativo ao mapa de cores corrente). Se for um inteiro negativo, a mesma cor que a face "visível" é aplicada para a parte traseira.

clip_state:
: este campo contém o valor da propriedade `clip_state` para a superfície. O valor de clip_state pode ser:

- `"off"` significa que a superfície não é recortada.
- `"clipgrf"` significa que a superfície é recortada fora da caixa dos eixos.
- `"on"` significa que a superfície é recortada fora do retângulo dado pela propriedade clip_box.

clip_box:
: este campo determina a propriedade `clip_box`. Por padrão seu valor é uma matriz vazia se a propriedade clip_state é "off". Em outros casos, o vetor `[x,y,w,h]` (ponto superior esquerdo, largura, altura) define as porções da poligonal a serem exibidas, contudo o valor da propriedade `clip_state` será alterado.

user_data:
este campo pode ser usado para armazenar qualquer variável Scilab na estrutura de dados da superfície e recuperá-la.

# Exemplos

```
//criando uma figura
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
[xx,yy,zz]=genfac3d(t,t,z);
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
h=get("hdl") //obtendo manipulador da entidade corrente (aqui, é a superfície)
a=gca(); //obtendo eixos correntes
a.rotation_angles=[40,70];
a.grid=[1 1 1];
//criando grides
a.data_bounds=[-6,0,-1;6,6,5];
a.axes_visible="off";
//os eixos estão escondidos a.axes_bounds=[.2 0 1 1];
f=get("current_figure");
//obtendo o manipulador da figura raiz
f.color_map=hotcolormap(64);
//mudando o mapa de cores da figura
h.color_flag=1;
//colorindo de acordo com z
h.color_mode=-2;
//removendo as bordas das facetas
h.color_flag=2;
//colorindo de acordo com as dadas cores
h.data.color=[1+modulo(1:400,64),1+modulo(1:400,64)];
//gradação
h.color_flag=3;

scf(2); // criando segunda janela e utilizando o comando surf
subplot(211)
surf(z,'cdata_mapping','direct','facecol','interp')

subplot(212)
surf(t,t,z,'edgeco','b','marker','d','markersiz',9,'markeredg','red','markerfac
e=gce();
e.color_flag=1 // índice de cor proporcional à altitude (coordenada z)
e.color_flag=2; // de volta ao modo padrão
e.color_flag= 3; // modo de gradação interpolada (baseada na cor azul padrão po
```

# Ver Também

set, get, delete, plot3d, plot3d1, plot3d2, surf, graphics_entities

# Autores

Djalel ABDEMOUCHE & F.Leray

# Name

swap_handles — Permuta dois manipuladores em uma hierarquia gráfica

```
swap_handle( handle1, handle2 )
```

# Parâmetros

handle1
: primeiro manipulador da permutação

handle2
: segundo manipulador da permutação

# Descrição

A função **swap_handles** permite permutar dois manipuladores em uma hierarquia gráfica. O primeiro manipulador tomará a posição do segundo e vice-versa.

Desde que nem todos os manipuladores são compatíveis com cada um dos outros, existem algumas restrições ao se trocar manipuladores. Por exemplo, não é permitido trocar um manipulador Polyline com um manipulador Axes, já que não seriam compatíveis com suas novas raízes. Mais informação sobre compatibilidade pode ser encontrada na página graphics_entities.

Esta rotina pode ser utilizada em galhos com uma mesma raiz para se trocar os seus índices.

# Exemplos

```
//-----------------//
// Primeiro Exemplo//
//-----------------//

// criando um retângulo
xrect( 0.5, 0.5,0.5,0.5) ;
rect = gce() ;

// criando um círculo
xarc( 0.5, 0.5, 0.5, 0.5, 0, 64 * 360 ) ;
circle = gce() ;

// criando uma seta
xpoly([0,1],[0,1]) ;
arrow = gce() ;
arrow.polyline_style = 4 ;
arrow.arrow_size_factor = 4 ;

// obtendo a lista de galhos
axes = gca()  ;
axes.children

// mudando a ordem
swap_handles( rect, arrow ) ;
swap_handles( arrow, circle ) ;

// obtendo a nova ordem
axes.children
```

```
//-----------------//
//  Segundo Exemplo//
//-----------------//

// criando duas janelas
plot2d ;
axes1 = gca() ;

scf() ;
fec ;
axes2 = gca() ;

// trocando os seus eixos
// note que o mapa de cores não muda.
swap_handles( axes1, axes2 ) ;
```

# Ver Também

graphics_entities, copy, delete, relocate_handle

# Autor

Jean-Baptiste Silvy

# Name

text_properties — Descrição das propriedades da entidade Text (texto)

# Descrição

A entidade Text é uma folha na hierarquia de entidades gráficas. Esta entidade define parâmetros para escrita de strings.

parent:
esta propriedade contém o manipulador da raiz. A raiz de uma entidade Text deve ser do tipo `"Axes"` ou `"Compound"`.

children:
esta propriedade contém um vetor com os galhos do manipulador. Contudo, manipuladores text não possuem galhos correntemente.

visible:
este campo contém o valor da propriedade `visible` para a entidade. Pode ser `"on"` ou `"off"` . Por padrão, o texto é visível, a propriedade é `"on"`. Se `"off"`, o texto não é exibido na tela.

text:
a matriz contendo os strings do objeto. As linhas da matriz são exibidas horizontalmente, e as colunas verticalmente.

alignment:
especifica como os strings serão alinhados em suas colunas. o valor deve ser '**left**', '**center**' ou '**right**'.

data:
este campo é o vetor `[x,y,[z]]` da origem do texto na unidade de dados dos eixos.

box:
o valor deste campo pode ser `"on"` ou `"off"`. Se `"on"` uma caixa é desenhada ao redor do texto com uma linha em sua borda e um plano de fundo.

line_mode:
esta propriedade booleana permite desenhar apenas o plano de fundo da caixa quando a propriedade `box` é `"on"`. Se `line_mode` é `"off"` , a linha da caixa não é desenhada.

fill_mode:
esta propriedade booleana permite desenhar ou não o plano de fundo da caixa quando a propriedade `box` é `"on"`. Se `fill_mode` for `"off"`, o plano de fundo da caixa não é transparente.

text_box:
um vetor bidimensional especificando o tamanho do retângulo em coordenadas do usuário. O retângulo é utilizado quando a propriedade text_box_mode está ajustada pára '**centered**' ou '**filled**'.

text_box_mode:
Pode ter três valores diferentes : '**off**', '**centered**' ou '**filled**'. Se '**off**', os strings são exibidos utilizando-se a dada fonte e o campo data especifica a posição do ponto inferior esquerdo do texto. Se '**centered**', o texto é exibido no meio do retângulo cujo tamanho é fornecido por text_box. Se '**filled**', o tamanho de fonte dos strings será expandido para se preencher o retângulo.

Ao se utilizar os modos '**off**' ou '**centered**' , o tamanho dos textos se mantém constante sob ampliação. Eles são os melhores modos para se criar anotações em um gráfico. Por outro lado, ao se utilizar o modo '**filled**', o tamanho do texto segue a escala gráfica. É então possível ampliar objetos textos.

font_foreground:
este campo contém a cor utilizada para exibir os caracteres do texto. O valor deve ser um índice de cor (relativo ao mapa de cores corrente).

foreground:
este campo contém a cor utilizada para exibir a linha na borda da caixa. O valor deve ser um índice de cores (relativo ao mapa de cores corrente).

background:
este campo contém a cor utilizada para preencher a caixa ao redor do texto. O valor deve ser um índice de cor (relativo ao mapa de cores corrente).

font_size:
é um escalar especificando o tamanho dos caracteres exibidos. Se a propriedade `fractional_font` é `"off"` , apenas a parte inteira do valor é utilzada. Para mais informações, veja graphics_fonts.

font_style:
especifica a fonte a ser utilizada para exibir os strings. É um inteiro positivo fazendo referência a uma das fontes carregadas. Seu valor deve estar entre 0, referência à primeira fonte, e o número de fontes carregadas menos um, referência à última fonte. Para mais informações, veja graphics_fonts.

fractional_font:
Esta propriedade indica se o texto será exibido utilizando-se tamanhos de fonte fracionários. Pode ser `"on"` ou `"off"`. Se `"on"` o valor em ponto flutuante de `font_size` é utilizado para exibição e retira-se o serrilhamento da fonte. Se `"off"` , apenas a parte inteira é utilizada e a fonte não é suavizada.

font_angle:
esta propriedade determina a orientação do string. Especifica o valor da rotação em graus.

clip_state:
este campo contém o valor da propriedade `clip_state` para o texto. O valor de clip_state pode ser:

- `"off"` significa que o texto não é recortado.
- `"cliprf"` significa que o texto é recortado fora da caixa dos eixos.
- `"on"` significa que o texto é recortado fora do retângulo dado pela propriedade clip_box.

clip_box:
este campo contém o valor da propriedade `clip_box`. Seu valor deve ser uma matriz vazia se clip_state é "off" ou o vetor `[x,y,w,h]` (ponto superior esquerdo, largura, altura).

user_data:
este campo pode ser utilizado para armazenar qualquer variável Scilab na estrutura de dados da entidade texto e recuperá-la.

# Exemplos

```
a=get("current_axes");
a.data_bounds=[0,0;1,1];
a.axes_visible = 'on' ;

xstring(0.5,0.5,"Scilab is not esilaB",0,0)

t=get("hdl")   //obtendo o manipulador do novo objeto criado
```

```
t.font_foreground=6; // alterando as propriedades da fonte
t.font_size=5;
t.font_style=5;

t.text=["SCILAB","não";"é","esilaB"] ; // alterando
t.font_angle=90 ; // virando os strings
t.text_box = [0,0] ;
t.text_box_mode = 'centered' ; // o texto está agora centrado em [0.5,0.5].
t.alignment = 'center' ;
t.box = 'on' ; // desenhando uma caixa ao redor do texto
```

# Ver Também

set, get, delete, xtitle, graphics_entities

# Autor

Djalel ABDEMOUCHE, Jean-Baptiste SILVY

# Name

title — Exibe um título em uma janela gráfica

```
title(my_title)
title(my_title,<Property>)
title(<axes_handle>,<my_title>,<Property>)
```

# Parâmetros

my_title
: string, o título a ser exibido

<Property>
: argumento opcional, representa um par de declarações `{PropertyName,PropertyValue}`que definem propriedades de objetos globais aplicadas o título criado.

<axes_handle>
: este argumento opcional força o título a parecer dentro dos eixos selecionados dados por `axes_handle` ao invés dos eixos correntes (ver gca).

# Descrição

`title` exibe um título numa janela de gráficos.

Os argumentos Property (propriedade) devem ser usados para customizar o título. Aqui está uma lista completa das opções disponíveis.

Property :

`backgroundcolor` : este campo contém a cor utilizada para preencher a caixa, se houver. Seu valor deve ser um índice de cor (relativo ao mapa de cores corrente).

`color` : este campo contém a cor a ser utilizada para exibir o texto de título. Seu valor deve ser um índice de cor (relativo ao mapa de cores corrente).

`edgecolor` :este campo contém a cor usada para exibir a linha ao redor da caixa, se houver alguma. Seu valor deve ser um índice de cor (relativo ao mapa de cores corrente).

`fontname` : sete fontes diferentes estão disponíveis: "Courrier", "Symbol", "Times", "Times Italic", "Times Bold", "Definida pelo Usuário". A propriedade font_size é um índice no intervalo [0 6] que é associado a um dos nomes anteriores.

`fontsize` : a propriedade é utilizada para selecionar o tipo de tamanho do título. Seu valor deve ser um inteiro entre 0 e 5 que significa 8pt, 10pt, 12pt, 14pt, 18pt e 24pt.

`position` : este vetor 2d permite que você posicione manualmente o título na tela. A posição é armazenada nas unidades de dados dos eixos.

`rotation` : escalar que permite que você gire o título. A fonte é girada em sentido anti-horário com o ângulo dado em graus.

`visible` : este campo contém o valor da propriedade visible para o título. Pode ser "on" ou "off". Por padrão, o título é visível, a propriedade é "on" . Se "off" , o título não é exibido na tela.

# Exemplos

```
// exibindo um título com várias propriedades
title('meu título');
// mudando a cor de fonte do título
title('meu título','color','blue');
// mudando a cor ao redor da caixa
title('meu título','edgecolor','red');
// mudando a posição do título
title('meu título','position',[0.3 0.8]);
// mudando o tamanho da fonte
title('meu título','fontsize',3);
// dando uma rotação
title('meu título','rotation',90);

// podemos realizar todas essas modificações apenas pela instrução abaixo:
title('meu título','color','blue','edgecolor','red','fontsize',3,'rotation',90,
```

# Ver Também

label_properties, titlepage, xtitle

# Autor

F.Belahcene

# Name

titlepage — adiciona título no meio de uma janela de gráficos

```
titlepage(str)
```

# Parameters

str
: matriz de strings

# Descrição

`titlepage` exibe a matriz de strings `str` no meio da janela de gráficos corrente com uma fonte a maior possível.

# Ver Também

xtitle

# Autor

S. S.

# Name

twinkle — Faz uma entidade gráfica piscar

```
twinkle(h,[n])
```

# Parâmetros

h
: manipulador de uma entidade gráfica

n
: inteiro

# Descrição

`twinkle` faz com que a entidade gráfica dada pelo seu manipulador h pisque. Pode ser usado para encontrar o objeto gráfico correspondente a um manipulador gráfico em uma janela de gráficos. Por padrão, a entidade gráfica pisca 5 vezes, mas você pode mudar este número através do argumento opcional `n`.

# Exemplos

```
x=linspace(-2*%pi,2*%pi,100)';
plot2d(x,[sin(x),cos(x)]);
e=gce(); p1=e.children(1); p2=e.children(2);
// fazendo piscar o esboço de cos
twinkle(p1)
// fazendo piscar o esboço de sin piscar
twinkle(p2,10)
// eixos piscando
twinkle(gca())
```

# Ver Também

graphics_entities

# Name

unglue — Descola uma objeto Compound e o substitui por um galho individual

```
unglue(h)
H=unglue(h)
```

# Parâmetros

h
: manipulador de um Compound

H
: um vetor de manipuladores das entidades resultantes após um unCompound.

# Descrição

Dado um manipulador de uma entidade Compound, destrói o Compound e desembrulha as entidades elementares associadas à sua raiz. `glue` retorna um vetor de manipuladores desses galhos individuais.

# Ver Também

get, set, copy, glue, graphics_entities

# Autor

Djalel ABDEMOUCHE

# Name

unzoom — Diminui a ampliação

```
unzoom()
unzoom(H)
```

# Parâmetros

H
: um vetor de manipuladores Figure ou Axes

# Descrição

`unzoom()` é usado para remover o efeito de ampliação em todos os eixos da figura gráfica corrente:

`unzoom(H)` é usado para remover o efeito de ampliação em todos os Figures e Axes dados pelo vetor de manipuladores `H`. A remoção do efeito de ampliação para uma entidade Figure é o mesmo que remover o efeito de ampliação para todos os seus Axes galhos.

# Exemplos

```
clf()
x=0:0.01:6*%pi;
plot2d(x,sin(x^2))
zoom_rect([16,-1,18,1])
unzoom()

//aplicação a subesboços
clf()
x=0:0.01:6*%pi;
subplot(211)
plot2d(x,cos(x))
a1=gca();
subplot(212)
plot2d(x,cos(2*x))
a2=gca();
rect=[3 -2 7 10]; // um retângulo especificado nos eixos correntes (os últimos)
zoom_rect(rect)
unzoom(a1) // diminuição de ampliação aplicado ao primeiro esboço apenas
unzoom(a2) // diminuição de ampliação aplicado ao segundo esboço apenas
zoom_rect(rect) // aplicando ampliação de novo
unzoom(gcf()) // diminuição de ampliação aplicada a todos os eixos, equivalente
```

# Ver Também

zoom_rect, axes_properties

# Autor

Serge Steer INRIA

Jean-Baptiste Silvy INRIA

# Name

whitecolormap — Mapa de cores completamente branco

```
cmap=whitecolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

O mapa de cores é completamente branco

# Exemplos

```
f = scf();
plot3d1();
f.color_map = whitecolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, wintercolormap

# Name

winsid — retorna a lista de janelas de gráficos

```
x=winsid()
```

# Parâmetros

x
: vetor linha

# Descrição

`winsid` é usado para retornar a lista de janelas de gráficos como um vetor de números das janelas.

# Name

wintercolormap — Mapa de cores com tons do inverno (azul, verde)

```
cmap=wintercolormap(n)
```

# Parâmetros

n
: inteiro >= 3, o tamanho do mapa de cores.

cmap
: matriz com 3 colunas `[R,G,B]`.

# Descrição

`wintercolormap` computa um mapa de cores com `n` cores que variam do azul para o verde.

# Exemplos

```
f = scf();
plot3d1();
f.color_map = wintercolormap(32);
```

# Ver Também

colormap, autumncolormap, bonecolormap, coolcolormap, coppercolormap, graycolormap, hotcolormap, hsvcolormap, jetcolormap, oceancolormap, pinkcolormap, rainbowcolormap, springcolormap, summercolormap, whitecolormap

# Name

xarc — esboça parte de uma elipse

```
xarc(x,y,w,h,a1,a2)
```

# Parâmetros

x,y,w,h
: quatro valores reais definindo um retângulo

a1,a2
: valores reais definindo um setor

# Descrição

`xarc` desenha parte de uma elipse contida no retângulo `(x,y,w,h)` ( (ponto superior esquerdo, largura, altura), e no setor definido pelo ângulo `alpha1`e o ângulo `alpha1+alpha2`. `alpha1` e `alpha2` são respectivamente dados por `a1/64` graus e `a2/64` graus. Esta função usa a cor e a escala gráfica correntes.

# Exemplos

```
// escala de isovisualização
plot2d(0,0,-1,"031"," ",[-2,-2,2,2])
xset("color",3)
xarc(-1,1,2,2,0,90*64)
xarc(-1.5,1.5,3,3,0,360*64)
```

# Ver Também

xarcs, xfarc, xfarcs

# Autor

J.Ph.C.

# Name

xarcs — desenha partes de um conjunto de elipses

```
xarcs(arcs,[style])
```

# Parâmetros

arcs
: matriz de tamanho (6,n) descrevendo as elipses.

style
: vetor linha de tamanho n fornecendo o estilo a ser usado.

# Descrição

`xarcs` desenha parte de um conjunto de elipses descritas por `arcs`: `arcs=[x y w h a1 a2;x y w h a1 a2;...]'` onde cada elipse é definida por 6 parâmetros `(x,y,w,h,a1,a2)` (ver `xarc`).

`x, y, w, h` são especificados nas coordenadas do usuário.

`style(i)` fornece a cor utilizada para desenhar a elipse número `i`.

# Exemplos

```
plot2d(0,0,-1,"031"," ",[-1,-1,1,1])
arcs=[-1.0 0.0 0.5; // x superior esquerdo
       1.0 0.0 0.5; // y superior esquerdo
       0.5 1.0 0.5; // largura
       0.5 0.5 1.0; // altura
       0.0 0.0 0.0; // ângulo 1
       180*64 360*64 90*64]; // ângulo 2
xarcs(arcs,[1,2,3])
```

# Ver Também

xarc, xfarc, xfarcs

# Autor

J.Ph.C.;

# Name

xarrows — desenha um conjunto de setas

```
xarrows(nx,ny,[arsize,style])
```

# Parâmetros

nx,ny
: vetores ou matrizes de reais de mesmo tamanho

arsize
: escalar real, o tamanho da cabeça da seta. O tamanho padrão pode ser obtido ajustando arsize para -1.

style
: matriz ou escalar. Se `style` for um escalar positivo, fornece a cor a ser usada em todas as setas. Se for negativo, a cor corrente é utilizada. Se for um vetor `style(i)` fornece a cor a ser utilizada no vetor `i`.

# Descrição

`xarrows` desenha um conjunto de setas dadas por `nx` e `ny`. Se `nx` e `ny` forem vetores, a i-ésima seta é definida por `(nx(i),ny(i))-->(nx(i+1),ny(i+1))`. Se `nx` e `ny` são matrizes:

```
nx=[xi_1 x1_2 ...; xf_1 xf_2 ...]
ny=[yi_1 y1_2 ...; yf_1 yf_2 ...]
```

a `k`-ésima seta é definida por `(xi_k,yi_k)-->(xf_k,yf_k)`.

`xarrows` utiliza a escala de gráficos corrente que pode ser ajustada através de uma chamada a uma função de esboço de alto-nível como `plot2d`.

# Exemplos

```
x=2*%pi*(0:9)/8;
x1=[sin(x);9*sin(x)];
y1=[cos(x);9*cos(x)];
plot2d([-10,10],[-10,10],[-1,-1],"022")
xset("clipgrf")
xarrows(x1,y1,1,1:10)
xset("clipoff")
```

# Autor

J.Ph.C.

# Name

xbasc — limpa uma janela de gráficos

```
xbasc([window-id])
```

# Parâmetros

window-id
: escalar inteiro ou vetor de escalares inteiros

# Descrição

Sem argumentos, esta função limpa a janela de gráficos corrente e apaga os gráficos gravados. Em outro caso, ela limpa as janelas de gráficos cujos números estão inclusos no vetor `window-id`. Por exemplo `xbasc(1:3)` limpa as janelas 1, 2 e 3 e apaga seus gráficos gravados correspondentes. Se uma dessas janelas não existir, ela é automaticamente criada.

`xbasc` deleta cada galho das janelas especificadas, incluindo menus e uicontrols adicionados pelo usuário. Para evitar que menus e uicontrols sejam apagados, o comando `delete(gca())` pode ser utilizado ao invés.

A função `xbasc` está obsoleta. Para apagar uma figura, por favor, use ao invés `clf` ou `delete`.

# Ver Também

clf, xclear

# Name

xbasr — desenha novamente uma janela de gráficos

```
xbasr(win_num)
```

# Descrição

`xbasr` é usado para exibir novamente o conteúdo de uma janela de gráficos com i `win_num`. Funciona apenas com o driver `"Rec"`.

# Ver Também

driver, replot

# Autor

J.Ph.C.

# Name

xchange — conversão de coordenadas reais para coordenadas pixeis

```
[x1,y1,rect]=xchange(x,y,dir)
```

# Parâmetros

x,y
: duas matrizes de tamanho (n1,n2) (coordenadas de um conjunto de pontos)

dir
: parâmetro utilizado para especificar o tipo de conversão (ver "Description" para detalhes)

x1,y1
: duas matrizes de tamanho (n1,n2) (coordenadas de um conjunto de pontos)

rect
: um vetor de tamanho 4

# Descrição

Após o uso de uma função de gráficos, `xchange` computa coordenadas pixel a partir de coordenadas reais e vice-versa de acordo com o valor de parâmetro `dir`:"f2i" (float para int) significa real para pixel e "i2f" (int para float) significa pixel para real. `x1` e `y1` são as coordenadas de um conjunto de pontos definidos pelas antigas coordenadas `x` e `y`.

`rect` são as coordenadas em pixel do retângulo no qual o esboço foi feito: [ponto superior esquerdo, largura, altura].

# Exemplos

```
t=[0:0.1:2*%pi]';
plot2d(t,sin(t))
[x,y,rect]=xchange(1,1,"f2i")
[x,y,rect]=xchange(0,0,"i2f")
```

# Autor

J.Ph.C.

# Name

xclear — limpa janela de gráficos

```
xclear([window-id])
```

# Parâmetros

window-id
: vetor de inteiros ou escalar inteiro

# Descrição

Sem argumentos, esta função limpa a figura corrente ajustando a sua propriedade visible para 'off'. Em caso contrário, ela limpa as janelas cujos números estão inclusos no vetor `window-id`. Por exemplo `xclear(1:3)` limpa as janelas 1, 2 e 3. Se uma dessas janelas não existir, então ela é automaticamente criada.

A função `xclear` está obsoleta. Para limpar uma figura, por favor, utilize a função `clf` ao invés, ou a propriedade `visible`.

# Ver Também

xbasc

# Autor

J.Ph.C.

# Name

xclick — espera por um clique do mouse

```
[ibutton,xcoord,yxcoord,iwin,cbmenu]=xclick([flag])
```

# Parâmetros

ibutton
: escalar real (valor inteiro): número do botão do mouse, código da tecla... (ver descrição abaixo)

xcoord
: escalar real: coordenada x do ponteiro do mouse quando ocorreu o clique, na escala gráfica corrente.

ycoord
: escalar real: coordenada y do ponteiro do mouse quando ocorreu o clique, na escala gráfica corrente.

iwin
: escalar real (valor inteiro): número da janela onde a ação ocorreu.

cbmenu
: string: callback associado a um menu se `xclick` retorna devido a um clique em um menu. Neste caso, `ibutton`, `xcoord`, `ycoord`, e `iwin` tomam valores arbitrários.

flag
: escalar real (valor inteiro): se presente, a fila de eventos de clique não é limpa ao se entrar xclick.

# Descrição

`xclick` espera pelo clique do mouse na janela de gráficos.

Se for chamado com 3 argumentos do lado esquerdo, espera pelo clique do mouse na janela de gráficos corrente.

Se chamado com 4 ou 5 argumentos do lado esquerdo, espera pelo clique do mouse em qualquer janela de gráficos.

Os valores de `ibutton` estão descritos abaixo:

ibutton==0
: o botão esquerdo foi pressionado

ibutton==1
: o botão do meio foi pressionado

ibutton==2
: o botão direito foi pressionado

ibutton==3
: o botão esquerdo do mouse foi clicado

ibutton==4
: o botão do meio do mouse foi clicado

ibutton==5
: o botão direito do mouse foi clicado

ibutton==10
: o botão esquerdo do mouse foi clicado duas vezes

ibutton==11
: o botão do meio do mouse foi clicado duas vezes

ibutton==12
: o botão direito do mouse foi clicado duas vezes

ibutton >=32
: uma tecla com código ASCII `ibutton` foi pressionada

ibutton <=-32
: uma tecla com código ASCII `-ibutton` foi liberada

ibutton >=1000+32
: uma tecla com código ASCII `ibutton-1000` foi pressionada enquanto a tecla CTRL estava sendo pressionada.

ibutton==-1000
: a janela de gráficos foi fechada

  Aviso: `ibutton` era igual a -100 para fechamento de janelas gráficas até o Scilab 4.1.2, mas este código foi modificado (no Scilab 5.0), pois também era o código retornado pela liberação da tecla `d`.

ibutton==-2
: um menu dinâmico foi selecionado e seu callback é retornado em `cbmenu`.

# Ver Também

locate, xgetmouse, seteventhandler

# Autores

J.Ph.C.

V.C.

# Name

xdel — deleta uma janela de gráficos

```
xdel([win-nums])
```

# Parâmetros

win-nums
: inteiro ou vetor de inteiros

# Descrição

`xdel` deleta a janela de gráficos `win-nums` ou a janela de gráficos corrente, se não for fornecido nenhum argumento.

# Autor

J.Ph.C.

# Name

xfarc — preenche parte de uma elipse

```
xfarc(x,y,w,h,a1,a2)
```

# Parâmetros

x,y,w,h
: quatro valores reais definindo um retângulo

a1,a2
: valores reais definindo um vetor

# Descrição

`xfarc` preenche parte de uma elipse contida no retângulo `(x,y,w,h)` (ponto superior esquerdo, largura, altura), no setor definido pelo ângulo `alpha1` e o ângulo `alpha1+alpha2`. `alpha1` e `alpha2` são, respectivamente, `a1/64` graus e `a2/64` graus. Esta função usa a cor e a escala gráfica corrente.

# Exemplos

```
// escala de isovisualização
plot2d(0,0,-1,"031"," ",[-2,-2,2,2])
xfarc(-0.5,0.5,1,1,0,90*64)
xset("color",2)
xfarc(0.5,0.5,1,1,0,360*64)
```

# Ver Também

xarc, xarcs, xfarcs

# Autor

J.Ph.C.;

# Name

xfarcs — preenche partes de um conjunto de elipses

```
xfarcs(arcs,[style])
```

# Parâmetros

arcs
: matriz de tamanho (6,n) descrevendo as elipses.

style
: vetor linha de tamanho n fornecendo as cores a serem utilizadas.

# Descrição

`xarcs` preenche parte de um conjunto de elipses descritas por `arcs`: `arcs=[x y w h a1 a2;x y w h a1 a2;...]'` onde cada elipse é definida por 6 parâmetros `(x,y,w,h,a1,a2)` (ver `xfarc`).

`x, y, w, h` são especificados nas coordenadas do usuário.

`style(i)` fornece o número da cor para preencher a elipse número `i`.

# Exemplos

```
plot2d(0,0,-1,"031"," ",[-1,-1,1,1])
arcs=[-1.0 0.0 0.5; // x superior esquerdo
       1.0 0.0 0.5; // y superior esquerdo
       0.5 1.0 0.5; // largura
       0.5 0.5 1.0; // altura
       0.0 0.0 0.0; // ângulo 1
       180*64 360*64 90*64]; // ângulo 2
xfarcs(arcs,[1,2,3])
```

# Ver Também

xarc, xfarc, xfarc

# Autor

J.Ph.C.

# Name

xfpoly — preenche um polígono

```
xfpoly(xv,yv,[close])
```

# Parâmetros

xv,yv
: dois vetores de mesmo tamanho (os pontos do polígono).

close
: inteiro. Se close= 1, a linha poligonal é fechada; o valor padrão é 0.

# Descrição

`xfpoly` preenche um polígono com a cor corrente. Se `close` é igual a 1 um ponto é adicionado à poligonal `xv,yv` para definir o polígono.

# Exemplos

```
x=sin(2*%pi*(0:4)/5);
y=cos(2*%pi*(0:4)/5);
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("color",5)
xfpoly(x,y)

// apenas para novo estilo de gráficos
e=gce(); // retorna a última entidade corrente (a última criada aqui é a poligo
e.fill_mode='off';
e.closed = 'off' // a poligonal está aberta agora

xset("default")
```

# Ver Também

xfpolys, xpoly, xpolys

# Autor

J.Ph.C.

# Name

xfpolys — preenche um conjunto de polígonos

```
xfpolys(xpols,ypols,[fill])
```

# Parâmetros

xpols,ypols
: matrizes de mesmo tamanho (p,n) (pontos dos polígonos).

fill
: vetor de tamanho n ou tamanho (p,n)

# Descrição

`xfpolys` preenche um conjunto de polígonos de mesmo tamanho definidos pelas matrizes `xpols` e `ypols`. As coordenadas de cada polígono são armazenadas em uma coluna de `xpols` e `ypols`.

Os polígonos podem ser preenchidos por uma dada cor (preenchimento monótono) ou pintados com cores interpoladas (preenchimento por graduação).

preenchimento monótono
: Neste caso, `fill` deve ser um vetor de tamanho `n`. O padrão para preenchimento de um polígono de número i é dado por `fill(i)`:

- se `fill(i)<0`, o polígono é preenchido com padrão de identificador `-fill(i)`.
- se `fill(i)=0`, o polígono é desenhado no estilo de traço (ou cor) corrente e não é preenchido.
- se `fill(i)>0`, o polígono é preenchido com padrão de identificador `fill(i)`. Então seu contorno é desenhado com o estilo de traço (ou cor) corrente e fechado, se necessário.

preenchimento interpolado
: Neste caso, `fill` deve ser uma matriz com o mesmo mesmo tamanho de `xpols` e `ypols`. Note que `p` deve ser igual a 3 ou 4.

  `fill(k,i)` fornece a cor da `k` -ésima borda do polígono `i`.

# Exemplos

```
a=gca();a.data_bounds=[0,-10;210,40];a.foreground=color('red');
x1=[0,10,20,30,20,10,0]';
y1=[15,30,30,15,0,0,15]';
xpols=[x1 x1 x1 x1]; xpols=xpols+[0,60,120,180].*.ones(x1);
ypols=[y1 y1 y1 y1];
xfpolys(xpols,ypols,[-1,0,1,2])

// cores interpoladas
clf()
f=gcf();
a=gca();a.data_bounds=[0,-10;40,30];a.isoview='on';
x1=[0,10,20,10]';
y1=[10,0,10,20]';
c=linspace(2,100,4)';
xpols=[x1 x1+20 x1+10 x1+10];
ypols=[y1 y1    y1+10 y1-10];
```

```
cols= [c c($:-1:1) c([3 4 1 2]) c]
f.color_map=jetcolormap(max(cols));
xfpolys(xpols,ypols,cols)

// cores interpoladas
clf()
f=gcf();
x11=[0;20;20;0];y11=[10;10;30;30];c11=[10;10;30;30];
x12=x11;y12=y11+20;c12=[20;20;1;1];c12=[30;30;10;10];
x21=[0;30;30;0]+22;y21=[20;20;30;30];c21=[20;20;30;30];
x22=x21;y22=y21+10;c22=[30;30;20;20];
x31=[0;40;40;0]+55;y31=[0;0;30;30];c31=[0;0;30;30];
x32=x31;y32=y31+30;c32=[30;30;0;0];
X=[x11 x12 x21 x22 x31 x32];Y=[y11 y12 y21 y22 y31 y32];C=([c11 c12 c21 c22  c3
a=gca();a.isoview='on';
a.data_bounds=[min(X),min(Y);max(X),max(Y)];
f=gcf();f.color_map=graycolormap(max(C));
xfpolys(X,Y,C)
```

# Ver Também

xfpoly, xpoly, xpolys

# Autor

J.Ph.C.

# Name

xfrect — preenche um retângulo

```
xfrect(x,y,w,h)
xfrect(rect) // rect =[x,y,w,h]
```

# Parâmetros

x,y,w,h
: quatro valores reais definindo um retângulo

# Descrição

`xrect` preenche um retângulo definido por `[x,y,w,h]` (ponto superior esquerdo, largura, altura) usando a escala e o estilo correntes.

# Exemplos

```
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("color",5)
xfrect(-1,1,2,2)
xset("default")
```

# Ver Também

xrect, xrects

# Autor

J.Ph.C.

# Name

xget — retorna valores correntes do contexto gráfico. **Esta função está obsoleta**.

```
[x1]=xget(str,[flag])
xget()
```

# Parâmetros

str
: string.

flag
: opcional. Ajustado para 1 fornece um modo verboso.

# Descrição

**Aviso: esta função está obsoleta**. Use a representação de objetos gráficos do Scilab (ver as ajudas de set e get bem como a página graphics_entities).

Esta função é utilizada para se obter valores de um contexto de gráficos no tópico especificado pelo string `str`. Quando chamada sem argumento, um menu de escolha é criado exibindo os valores correntes e mudanças podem ser realizadas através de botões de alternância.

number=xget("alufunction")
: recebe o número da função lógica usada para desenhar. Ver `xset`.

str=xget("auto clear")
: recebe o status de auto-limpeza (`"on"` ou `"off"`).

color=xget("background")
: recebe a cor de fundo da janela de gráficos corrente.

rect=xget("clipping")
: recebe a zona de recorte como o retângulo rect=[x,y,w,h] (ponto superior esquerdo, largura, altura).

c=xget("color")
: recebe a cor padrão para funções de preenchimento, desenho de linha ou texto. c é um inteiro projetado no intervalo [0,whiteid]. 0 significa preenchimento com preto e whiteid preenchimento com branco. O valor de whitid `xget("white")`.

cmap=xget("colormap")
: recebe o mapa decores utilizado para a janela de gráficos corrente como uma matriz m x 3 RGB (vermelho, verde, azul).

dash=xget("dashes")
: recebe o estilo de traço dash= [dash_number] onde dash_number é o identifcador do traço. Esta palavra-chave está obsoleta, use `xget("color")` ou `xget("line style")` ao invés.

font=xget("font")
: recebe font=[fontid,fontsize], a fonte padrão e o tamanho padrão para fontes.

fontsize=xget("font size")
: recebe o tamanho padrão para fontes.

color=xget("foreground")
: recebe a cor de primeiro plano do objeto Axes corrente. O resultado é um índice do mapa de cores correspondente à cor.

str=xget("fpf")
: recebe o formato em ponto flutuante do número do número de exibição em funções de contorno . Note que `str` é " " quando o formato padrão é utilizado.

color=xget("hidden3d")
: recebe o número de cor para faces escondidas em plot3d.

pat=xget("lastpattern")
: recebe o identificador do último padrão ou cor disponível no mapa de cores da janela corrente. Na verdade `pat+1` e `pat+2` também estão disponíveis e correspondem a padrões preto e branco respectivamente.

type=xget("line mode")
: recebe o modo de desenho de linha. type=1 é o modo absoluto e type=0 é o modo relativo. (Aviso: type=0 tem defeitos)

xget("line style")
: recebe o estilo de linhas padrão(1: sólido, >1 para linhas tracejadas).

mark=xget("mark")
: recebe o identificador do estilo de marcas padrão e o tamanho de marcas padrão. `mark=[markid,marksize]`.

marksize=xget("mark size")
: recebe o tamanho de marcas padrão.

pat=xget("pattern")
: recebe o padrão corrente ou a cor corrente. `pat` é um inteiro no intervalo `[1,last]`. Quando se usa preto ou branco, 0 é usado para preenchimento com preto e `last` para preenchimento com branco. O valor de `last` pode ser obtido com `xget("lastpattern")`.

value=xget("thickness")
: recebe a espessura das linhas em pixel (0 e 1 têm o mesmo significado: 1 pixel de espessura).

flag=xget("use color")
: recebe o flag 0 (para uso apenas de preto e branco) ou 1 (uso de cores). Ver `xset`.

[x,y]=xget("viewport")
: recebe a posição corrente da parte visível de gráficos.

dim=xget("wdim")
: recebe a largura e a altura da janela de gráficos corrente dim=[largura,altura].

win=xget("window")
: recebe o número de janela corrente `win`.

pos=xget("wpos");
: recebe a posição do ponto superior esquerdo da janela de gráficos pos=[x,y].

# Ver Também

xset, getcolor, getsymbol, ged, set, graphics_entities

# Autor

J.Ph.C.

# Name

xgetech — retorna a escala de gráficos corrente

```
[wrect,frect,logflag,arect]=xgetech()
```

# Parâmetros

wrect,frect
: vetores de reais

logflag
: string de tamanho 2 "xy"

# Descrição

`xgetech` retorna a escala de gráficos corrente (da janela corrente). O retângulo [xmin,ymin,xmax,ymax] dado por `frect` é o tamanho de toda a janela de gráficos. O esboço será feito na região da janela de gráficos corrente especificada por `wrect`.

`wrect=[x,y,w,h]` (ponto superior esquerdo, largura, altura) descreve uma região dentro de uma janela de gráficos. Os valores em `wrect` são especificados utilizando-se a proporção de largura e altura da janela de gráficos:

`wrect=[0 0 1 1]` significa que toda a janela de gráficos será utilizada.

`wrect=[0.5 0 0.5 1]` significa que a região de gráficos é a metade direita da janela de gráficos.

`logflag` é um string de tamanho 2 "xy", onde x e y podem ser "n" ou "l". "n" significa escala normal (linear) e "l" significa escala logarítmica. x corresponde ao eixo x e y corresponde ao eixo y.

`arect=[x_left, x_right,y_up,y_down]` fornece o tamanho da moldura dentro da sub-janela. O quadro gráfico é especificado (como `wrect`) utilizando-se a proporção de largura ou altura da subjanela de gráficos corrente. O valor padrão é `1/8*[1,1,1,1]`. Se `arect` não for fornecido, o valor corrente permanece inalterado.

# Exemplos

```
// primeira subjanela
xsetech([0,0,1.0,0.5])
plot2d()
// então, xsetech, é usado para ajustar a segunda sub-janela
xsetech([0,0.5,1.0,0.5])
grayplot()
// obtendo as escalas gráficas da primeira sub-janela
xsetech([0,0,1.0,0.5])
[wrect,frect,logflag,arect]=xgetech();
// obtendo as escalas gráficas da segunda sub-janela
xsetech([0,0.5,1.0,0.5])
[wrect,frect,logflag,arect]=xgetech();
xbasc();
xset('default')
```

# Ver Também

xsetech

# Autor

J.Ph.C.;

# Name

xgetmouse — retorna os eventos de mouse e posição corrente

```
[rep [,win]]=xgetmouse([sel])
```

# Parâmetros

sel
: vetor de booleanos `[getmotion, getrelease]`. O valor padrão é `[%t, %f]`

rep
: vetor de tamanho 3, [x,y,ibutton].

win
: número da figura onde o evento ocorreu

# Descrição

Se o cursor do mouse estiver localizado na janela de gráficos corrente, `xgetmouse` retorna em `rep` a posição corrente do cursor (x,y) e o valor `ibutton`. O valor de `ibutton` indica o tipo de evento:

ibutton==0
: o botão esquerdo do mouse foi pressionado

ibutton==1
: o botão do meio do mouse foi pressionado

ibutton==2
: o botão direito do mouse foi pressionado

ibutton==3
: o botão esquerdo do mouse foi clicado

ibutton==4
: o botão do meio do mouse foi clicado

ibutton==5
: o botão direito do mouse foi clicado

ibutton==10
: o botão esquerdo do mouse foi clicado duas vezes

ibutton==11
: o botão do meio do mouse foi clicado duas vezes

ibutton==12
: o botão direito do mouse foi clicado duas vezes

ibutton==-5
: o botão esquerdo do mouse foi liberado

ibutton==-4
: o botão do meio do mouse foi liberado

ibutton==-3
: o botão direito do mouse foi liberado

ibutton==-1
: o cursor foi movido

ibutton > =32
: uma tecla com código ASCII ascii(ibutton) foi pressionada

ibutton < =-32
: uma tecla com código ASCII ascii(-ibutton) foi liberada

ibutton > =1000+32
: uma tecla com código ASCII ascii(ibutton-1000) foi pressionada enquanto o botão CTRL estava sendo pressionado

ibutton==-1000
: a janela de gráficos foi fechada

AVISO: Em versões anteriores do Scilab (<5.0), o usuário podia fornecer um flag (indicador) para precisar se a fila de eventos do mouse deveria ser limpa ao se entrar xgetmouse. Está opção está obsoleta agora e será removida no Scilab 5.1.

# Exemplos

```
//  seleção de retângulo
clf();  // apagando/criando janela
a=gca();a.data_bounds=[0 0;100 100];//ajustando cooredenadas do usuário
xtitle(" drawing a rectangle ") //adicionando um título
xselect(); //pondo a janela no topo

[b,xc,yc]=xclick(); //retornando um ponto
xrect(xc,yc,0,0) //desenhando a entidade Rectangle
r=gce();// o manipulador do retângulo
rep=[xc,yc,-1];first=%f;

while rep(3)==-1 do // movimentação do mouse...
  rep=xgetmouse();
  xc1=rep(1);yc1=rep(2);
  ox=mini(xc,xc1);
  oy=maxi(yc,yc1);
  w=abs(xc-xc1);h=abs(yc-yc1);
  r.data=[ox,oy,w,h]; //mudando a origem, largura e altura do retângulo
  first=%f;
end
```

# Ver Também

locate, xclick, seteventhandler

# Autor

S. Steer

# Name

xgraduate — Graduação de eixos

```
[xi,xa,np1,np2,kMinr,kMaxr,ar]=xgraduate(xmi,xma)
```

# Parâmetros

xmi,xma
: escalares reais

xi, xa, kMinr, kMaxr, ar
: escalares reais

np1,np2
: inteiro

# Descrição

`xgraduate` retorna as graduações de eixos que são utilizadas pelas rotinas de esboço (com o flag de embelezamento habilitado). Retorna um intervalo `[xi,xa]` que contém o dado intervalo `[xmi,xma]` e tal que

`xi= kMinr*10^ar`, `xa=kMaxr*10^ar` e o intervalo pode ser dividido em `np2` intervalos e cada intervalo é dividido em `np1` subintervalos.

# Exemplos

```
[x1,xa,np1,np2,kMinr,kMaxr,ar]=xgraduate(-0.3,0.2)
```

# Ver Também

graduate, plot2d

# Autor

J.P.C ; ;

# Name

xgrid — adiciona um grid em um esboço 2d

```
xgrid([style])
```

# Parâmetros

style
: inteiro

# Descrição

`xgrid` adiciona um grid em um esboço 2d. `style` é o identificador de tipo de tracejado ou de cor utilizado para o esboço do grid. Use `xset()` para o significado do identificador.

# Exemplos

```
x=[0:0.1:2*%pi]';
plot2d(sin(x))
xgrid(2)
```

# Ver Também

xset, plot2d

# Autor

J.Ph.C.

# Name

xinfo — escreve um string de informação na subjanela de mensagens

```
xinfo(info)
```

# Parâmetros

info
: string

# Descrição

`xinfo` escreve o string `info` na subjanela de mensagens da janela de gráficos corrente.

# Name

xlfont — carrega uma fonte em um contexto gráfico ou pergunta a fonte carregada

```
xlfont(font-name)
xlfont(font-filename)
xlfont('reset')
xlfont(font-name,font-id)
xlfont(font-filename,font-id)
xlfont(font-name,font-id,bold)
xlfont(font-name,font-id,bold,italic)
fonts=xlfont('AVAILABLE_FONTS')
fonts=xlfont()
```

# Parâmetros

font-name
: string, nome da família da fonte

font-filename
: string, nome do arquivo de uma fonte true type

font-id
: inteiro >= 0 .

fonts
: um vetor coluna de nomes de fontes

bold
: um booleano, %t para negrito

italic
: um booleano, %t para itálico

# Descrição

Sem argumentos, `xlfont()` retorna a lista das fontes correntemente carregadas.

xlfont('AVAILABLE_FONTS') retorna a lista de fontes disponíveis no seu sistema.

xlfont('reset') reajusta para o índice de fontes inicial.

Com argumentos, `xlfont` é usado para carregar uma nova fonte em tamanhos diferentes no contexto gráfico.

As famílias de fonte padrões são "Monospaced" (0), "Symbol" (1), "Serif" (2), "Serif Italic" (3), "Serif Bold" (4), "Serif Bold Italic" (5), "SansSerif" (6), "SansSerif Italic" (7), "SansSerif Bold" (8), "SansSerif Bold Italic" (9). Estas fontes são automaticamente carregadas quando necessário, então `xlfont` não é necessariamente requerido para elas. Na verdade `xlfont` é essencialmente útil para carregar uma nova fonte.

# Exemplos

```
xlfont('reset');
xlfont()

// este exemplo pode n##o funcionar se seu sistema n##o possuir a fonte Monospa
```

```
xlfont("Monospaced",10,%t,%t);
xstring(1,0,'A title');
figure_entity = gcf();
axes_entity = figure_entity.children;
title_entity = axes_entity.children;
title_entity.font_style = 10;

xlfont()

xlfont(SCI+'/thirdparty/fonts/scilabsymbols.ttf')
title_entity.font_style = 11; // utilizando a fonte scilabsymbols.ttf
title_entity.font_size  = 4; // tamanho da fonte scilabsymbols.ttf

xlfont()

xlfont('reset');
```

# Ver Também

getfont

# Autor

Allan CORNET

# Name

xload — carrega gráficos salvos

```
xload(file_name,[win_num])
```

# Parâmetros

file_name
: string, o nome do arquivo

win_num
: inteiro, o número da janela de gráficos. Se não for fornecido, a janela de gráficos corrente é utilizada

# Descrição

`xload` recarrega os gráficos contidos no arquivo `file_name` na janela de gráficos `win_num`.

Desde o Scilab 5.0, todos os manipuladores uimenu ou uicontrol são também carregados.

Para arquivos contendo novos gráficos, a função `load` pode ser utilizada ao invés de `xload`. `xload` não recupera o número da janela, nem seu tamanho ou dimensões.

# Exemplos

```
//novo estilo
t=0:0.01:10;
subplot(211),plot2d(t,sin(t))
subplot(212),plot2d(t,sin(3*t))
save(TMPDIR+'/foo.scg',gcf())
clf()
load(TMPDIR+'/foo.scg')

a=gca();
curve=a.children.children; //manipulador da curva
save(TMPDIR+'/foo.scg',curve)
delete(curve)
load(TMPDIR+'/foo.scg')
```

# Ver Também

xsave, load, save

# Autor

J.Ph.C.

# Name

xname — muda o nome da janela de gráficos corrente

```
xname(name)
```

# Parâmetros

name
: string, o novo nome da janela de gráficos.

# Descrição

`xname` muda o nome da janela de gráficos corrente.

# Autor

J.Ph.C.

# Name

xnumb — Imprime números

```
xnumb(x,y,nums,[box,angle])
```

# Parâmetros

x,y,nums
: vetores de mesmo tamanho

box
: valor inteiro

angle
: vetor opcional de mesmo tamanho de `x`

# Descrição

`xnumb` imprime os valores de `nums(i)` na posição `x(i),y(i)` na escala corrente. Se `box` tem valor 1, um retângulo é desenhado ao redor dos números. Se `angle` for dado, fornece a direção para escrita do string.

# Exemplos

```
plot2d([-100,500],[-100,600],[-1,-1],"022")
x=0:100:200;
xnumb(x,500*ones(x),[10,20,35],1)
```

# Ver Também

xstring

# Autor

J.Ph.C.

# Name

xpause — suspende o Scilab

```
xpause(microsecs)
```

# Descrição

`xpause` suspende o processo corrente pelo número de microssegundos especificados pelo argumento. A suspensão real pode tornar-se mais longa devido a outras atividades do sistema, ou por causa do tempo gasto no processamento da chamada.

# Autor

J.Ph.C.

# Name

xpoly — desenha uma poligonal ou um polígono

```
xpoly(xv,yv [,dtype [,close]])
```

# Parâmetros

xv,yv
: matrizes de mesmo tamanho (os pontos da poligonal).

dtype
: string (estilo de desenho). O valor padrão é "lines".

close
: inteiro. Se close= 1,a poligonal é fechada; o valor padrão é 0.

# Descrição

`xpoly` desenha uma poligonal descrita pelos vetores de coordenadas `xv` e `yv`. Se `xv` e `yv` forem matrizes, serão considerados vetores obtidos por concatenação de suas colunas. `dtype` pode ser `"lines"` para uso do estilo de linha corrente ou `"marks"` para utilizar a marca corrente para desenho da poligonal.

# Exemplos

```
x=sin(2*%pi*(0:4)/5);
y=cos(2*%pi*(0:4)/5);

plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("color",5)
xpoly(x,y,"lines",1) // por padrão é fechado

// apenas no novo estilo de gráficos
e=gce(); // retorna a entidade corrente (a última criada: aqui, é a poligonal)
e.closed = 'off' // a poligonal está agora aberta
```

# Ver Também

xfpoly, xfpolys, xpolys

# Autor

J.Ph.C.

# Name

xpolys — desenha um conjunto de poligonais ou polígonos

```
xpolys(xpols,ypols,[draw])
```

# Parâmetros

xpols,ypols
: matrizes de mesmo tamanho (p,n) (pontos das poligonais)

draw
: vetor de tamanho n

# Descrição

`xpolys` desenha um conjunto de poligonais usando marcas ou linhas tracejadas. As coordenadas de cada poligonal são armazenadas em uma coluna de `xpols` e `ypols`.

O estilo da poligonal i é dado por `draw(i)`:

- Se `draw(i)` for negativo, a marca com identificador `-draw(i)` i é utilizada para desenhar a poligonal i (marcas são desenhadas utilizando-se o padrão corrente). Use `xset()` para ver o significado dos identificadores.
- Se `draw(i)` for estritamente positivo, o estilo de linha (ou cor) com identificador `draw(i)` é utilizado para desenhar a poligonal i. Use `xset()` para ver o significado dos idenficadores.

# Exemplos

```
plot2d(0,0,-1,"012"," ",[0,0,1,1])
rand("uniform")
xset("color",3)
xpolys(rand(3,5),rand(3,5),[-1,-2,0,1,2])
xset("default")
```

# Ver Também

xfpoly, xfpolys, xpoly

# Autor

J.Ph.C.

# Name

xrect — desenha um retângulo

```
xrect(x,y,w,h)
xrect(rect) // rect =[x,y,w,h]
```

# Parameters

x,y,w,h
: quatro valores reais definindo o retângulo

# Descrição

`xrect` desenha um retângulo definido por `[x,y,w,h]` (ponto superior esquerdo, largura, altura) usando a escala e o estilo correntes.

AVISO: por favor, note que a altura é positiva indo para baixo.

# Exemplos

```
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("color",5)
xrect(-1,1,2,2)
xset("default")
```

# Ver Também

xfrect, xrects

# Autor

J.Ph.C.

# Name

xrects — desenha ou preenche um conjunto de retângulos

```
xrects(rects,[fill])
```

# Parâmetros

rects
: matrtiz de tamanho (4,n)

fill
: vetor de tamanho n.

# Descrição

`xrects` desenha ou preenche retângulos. Cada coluna de `rects` descreve um retângulo (ponto superior esquerdo, largura, altura): `rects=[x1 y1 w1 h1;x2 y2 w2 h2;...]'`.

`fill(i)` fornece o padrão para preenchimento ou desenho do retângulo i:

Se `fill(i)<0`, o retângulo `i` é desenhado utilizando-se o estilo de linha (ou cor) `-fill(i)`

Se `fill(i)>0`, o retângulo `i` é preenchido utilizando-se o padrão (ou cor) `fill(i)`

Se `fill(i)=0`, o retângulo `i` é desenhado utilizando-se o estilo de linha corrente (ou cor).

AVISO: por favor, note que a altura é positiva indo para baixo.

# Exemplos

```
plot2d([-100,500],[-50,50],[-1,-1],"022")
cols=[-34,-33,-32,-20:5:20,32,33,34];
x=400*(0:14)/14; step=20;
rects=[x;10*ones(x);step*ones(x);30*ones(x)];
xrects(rects,cols)
xnumb(x,15*ones(x),cols)
```

# Ver Também

xfrect, xrect

# Autor

J.Ph.C.

# Name

xrpoly — desenha um polígono regular

```
xrpoly(orig,n,r,[theta])
```

# Parâmetros

orig
: vetor de tamanho 2.

n
: inteiro, número de lados.

r
: escalar real.

theta
: escalar real, ângulo em radianos; 0 é o valor padrão.

# Descrição

`xrpoly` desenha um polígono regular com `n` lados contindo no círculo de diâmetro `r` e com a origem do círculo posta em um ponto `orig`. `theta` especifica a rotação do ângulo em radianos. Esta função utiliza as escalas gráficas correntes.

# Exemplos

```
plot2d(0,0,-1,"012"," ",[0,0,10,10])
xrpoly([5,5],5,5)
```

# Ver Também

xrect

# Name

xsave — salva gráficos em um arquivo

```
xsave(filename,[win_num])
```

# Parâmetros

file_name
: string, nome do arquivo

win_num
: inteiro, o número da janela de gráficos. Se não for forneceido, a janela de gráficos corrente é utilizada.

# Descrição

`xsave` salva os gráficos contidos na janela de gráficos `win_num` no arquivo binário `file_name` e pode ser carregado com `xload`.

Desde o Scilab 5.0, todos os manipuladores uimenu ou uicontrol também são salvos.

Para novos gráficos `xsave(file_name,win_num)`use preferivelmente `save(file_name,scf(win_num)`.

# Exemplos

```
//novo estilo
t=0:0.01:10;
subplot(211),plot2d(t,sin(t))
subplot(212),plot2d(t,sin(3*t))
save(TMPDIR+'/foo.scg',gcf())
clf()
load(TMPDIR+'/foo.scg')

a=gca();
curve=a.children.children; //handle on the curve
save(TMPDIR+'/foo.scg',curve)
delete(curve)
load(TMPDIR+'/foo.scg')
```

# Ver Também

xload, save, load

# Autor

J.Ph.C.

# Name

xsegs — desenha segmentos desconexos

```
xsegs(xv,yv,[style])
```

# Parâmetros

xv,yv
: matrizes de mesmo tamanho

style
: vetor ou escalar. Se `style` is a positive scalar, it gives the color to use for all segments. Se `style` for negativo, então a cor corrente é utilizada. Se `style` for um vetor, então `style(i)` fornece a cor a ser utilizada no segmento `i`.

# Descrição

`xsegs` desenha um conjunto de segmentos desconexos dados por `xv` e `yv`. Se `xv` e `yv` forem matrizes, serão considerados como vetores através da concatenação de suas colunas. As coordenadas dos dois pontos definindo um segmento são dadas por dois valores consecutivos de `xv` e `yv`:

`(xv(i),yv(i))-->(xv(i+1),yv(i+1))`.

Por exemplo, utilizando matrizes de tamanho (2,n), os segmentos são definidos por:

```
xv=[xi_1 xi_2 ...; xf_1 xf_2 ...]
yv=[yi_1 yi_2 ...; yf_1 yf_2 ...]
```

e os segmentos são `(xi_k,yi_k)-->(xf_k,yf_k)`.

# Exemplos

```
x=2*%pi*(0:9)/10;
xv=[sin(x);9*sin(x)];
yv=[cos(x);9*cos(x)];
plot2d([-10,10],[-10,10],[-1,-1],"022")
xsegs(xv,yv,1:10)
```

# Autor

J.Ph.C.

# Name

xselect — restaura a janela de gráficos corrente

```
xselect()
```

# Descrição

`xselect` restaura a janela de gráficos corrente. Se tal janela não existir, uma é criada.

Aviso: está função está obsoleta e será removida no Scilab 5.1. Ela foi substituída pela função show_window.

# Ver Também

show_window

# Autores

J.Ph.C.

Jean-Baptiste Silvy

# Name

xset — ajusta valores para o contexto gráfico. Função obsoleta

```
xset(choice-name,x1,x2,x3,x4,x5)
xset()
```

# Parâmetros

choice-name
: string

x1,...,x5
: dependem de `choice-name`

# Descrição

Aviso: está função está obsoleta. Utilize a representação de objetos do Scilab ao invés (ver as documentações das funções set e get bem como a página graphics_entities).

`xset` é utilizado para ajustar valores padrões do contexto da janela de gráficos corrente.

Quando chamado sem argumento, um menu de escolha é criado exibindo os valores correntes e mudanças podem ser realizadas através e botões de escolha.

Use `xset()` para exibir ou ajustar a cor corrente, marca e fontes utilizadas.

xset("alufunction",number)
: usado para ajustar a função lógica para desenho. A função lógica utilizada é ajustada por `x1`. Os valores usuais são: 3 para cópia (padrão), 6 para animação e 0 para limpeza. Ver alufunctions para mais detalhes.

xset("auto clear","on"|"off")
: alterna o modo de limpeza automática para gráficos entre "on" e "off" . Quando o modo de limpeza automática é "on", esboços sucessivos não são superpostos, ie, uma operação `xbasc()` (a janela de gráficos é limpa e os gráficos gravados associados são apagados) é realizada antes de cada função gráfica de alto-nível. O valor padrão é "off".

xset("background",color)
: ajusta a cor de plano de fundo do objeto Axes corrente. O argumento `color` é um índice no mapa de cores da cor a ser utilizada.

xset("clipping",x,y,w,h)
: ajusta a zona de recorte (a zona da janela de gráficos onde os esboços podem ser desenhados) como sendo o retângulo (x,y,w,h) (ponto superior esquerdo, largura, altura). Esta função utiliza a utiliza as coordenadas correntes do esboço.

xset("color",value)
: ajusta a cor padrão para funções de preechimento, linha ou impressão de textos. `value` é um inteiro projetado no intervalo [0,whiteid]. 0 é usado para preenchimento com preto e whiteid para preenchimento com branco. O valor do identificador do branco pode ser obtido através de `xget("white")`.

xset("colormap",cmap)
: ajusta o mapa de cores como sendo uma matriz m x 3. m é o número de cores. A cor de número i é dada como sendo a tripla cmap(i,1), cmap(i,2), cmap(i,3) correspondentes respectivamente às intensidades de vermelho, verde e azul entre 0 e 1.

xset("dashes",i)
no modo preto e branco (`xset("use color",0)`), ajusta o estilo de tracejado como o estilo i (0 para linha sólida). No modo colorido (`xset("use color",1 )`) é utilizado para ajustar a cor da linha, da marca e do texto. Esta palavra-chave está obsoleta, utilize `xset('color',i)` ou `xset('line style',i)` ao invés.

xset("default")
reajusta o contexto gráfico para valores padrões.

xset("font",fontid,fontsize) :
ajusta a fonte corrente e seu tamanho. Note que `fontsize` se aplica a todas as fontes, não só a `fontid` .

xset("font size",fontsize)
ajusta o tamanho da fonte.

xset("foreground",color)
ajusta a cor de primeiro plano do objeto Axes corrente. O argumento `color` é um índice no mapa de cores da cor a ser utilizada.

xset("fpf",string)
ajusta o formato de exibição de ponto flutuante para funções de contorno. `string` é um string fornecendo o formato em sintaxe C (por exemplo `string="%.3f"`). Use `string=""` para retornar ao formato padrão.

xset("hidden3d",colorid) :
ajusta o número de cor para as faces escondidas em um `plot3d. colorid=0` suprime o desenho de faces escondidas de objetos 3d. Isto é tecnicamente chamdo 'culling' e acelera a renderização de superfícies fechadas.

xset("line mode",type)
esta função é utilizada para se ajustar o modo de desenho da linha. O modo absoluto é ajustado com type= 1 e o modo relativo com type=0. (aviso: o modo type= 0 possui defeitos)

xset("line style",value)
ajusta o estilo de linha corrente (1: sólida, >1 estilos tracejados).

xset("mark",markid,marksize)
ajusta a marca corrente e seu tamanho. Use `xset()` para visualizar as marcas. Note que `marksize` se aplica a todas as marcas, não apenas a `markid` .

xset("mark size",marksize)
ajusta o tamanho da marca.

xset("pattern",value)
ajusta o padrão (pattern, não confundir com padrão = default) corrente para funções de preenchimento. O valor é um inteiro projetado no intervalo [0,whiteid]. 0 é utilizado para preenchimento com preto e whiteid para preenchimento com branco. O valor de whiteid pode ser obtido com `xget("white")`. "pattern" é equivalente a "color".

xset("pixmap",flag)
se flag=0, os gráficos são diretamente exibidos na tela. Se flag=1, os gráficos são feitos em um pixmap (mapa de pixel) e enviados à janela de gráficos através do comando `xset("wshow")`. O pixmap é limpo com o comando `xset("wwpc")`. Note que o comando usual `xbasc()` também limpa o pixmap.

xset("thickness",value)
ajusta o valor de espessura das linhas em pixel (0 e 1 têm o mesmo significado: 1 pixel de espessura).

xset("use color",flag)
: se flag=1, então `xset("pattern",.)` ou `xset("dashes",.)` será utilizado de modo a modificar a cor padrão para desenho ou para padrões de preenchimento. Se flag=0 então retorna-se ao modo de escalas de cinza e tracejados.

xset("viewport",x,y)
: ajusta a posição do cursor.

xset("wdim",width,height)
: ajusta a largura e altura da janela de gráficos corrente. Esta opção não é utilizada pelo driver Postscript.

xset("wpdim",width,height)
: ajusta a largura e altura da janela de gráficos física corrente (que pode ser diferente do tamanho atual no modo wresize 1). Esta opção não é utilizada pelo driver Postscript.

xset("window",window-number)
: ajusta a janela `window-number` como sendo a janela corrente e cria a janela se esta não existir.

xset("wpos",x,y)
: ajusta a posição do ponto superior esquerdo da janela de gráficos.

xset("wresize",flag)
: se flag= 1então o gráfico é automaticamente redimensionado para preencher a janela de gráficos.

```
xdel();
xset("wresize",1);
plot2d();
xset("wdim",1000,500)
```

Se flag=0 a escala do gráfico é inalterada quando a janela de gráfico é redimensionada. O cursor no canto superior esquerdo ou as setas do teclado podem ser utilizadas para rolar o gráfico .

```
xdel();
plot2d();
xset("wresize",0);
xset("wdim",1000,500)
```

xset("wshow")
: ver `xset("pixmap",1)` acima.

xset("wwpc")
: ver `xset("pixmap",1)` acima.

# Ver Também

xget, getcolor, getsymbol, ged, set, graphics_entities

# Autor

J.Ph.C.

# Name

xsetech — ajusta a sub-janela de uma janela de gráficos para esboço

```
xsetech(wrect,[frect,logflag])
xsetech(wrect=[...],frect=[..],logflag="..", arect=[...])
xsetech()
```

# Parâmetros

wrect
: vetor de tamanho 4 definindo a sub-janela a ser utilizada.

frect
: vetor de tamanho 4.

logflag
: string de tamanho 2 "xy", onde x e y podem ser "n" ou "l". "n" escala normal e "l" significa escala logarítmica. x refere-se ao eixo x e y ao eixo y.

arect
: vetor de tamanho 4.

# Descrição

`xsetech` é principalmente utilizado para ajustar a sub-janela da janela de gráficos que será utilizada para esboço. A sub-janela é especificada através do parâmetro `wrect=[x,y,w,h]` (ponto superior esquerdo, largura, altura). Os valores em `wrect` a são especificados utilizando-se a proporção de largura ou altura da janela de gráficos corrente. Por exemplo, `wrect=[0,0,1,1]` significa que toda a janela será utilizada e `wrect=[0.5,0,0.5,1]` significa que a região gráfica será a metade direita da janela de gráficos.

`xsetech` também ajusta as escalas gráficas correntes para esboço 2d e pode ser utilizada juntamente a rotinas gráficas que requerem a escala de gráficos corrente (por exemplo `strf="x0z"` ou `frame-flag=0` em `plot2d`).

`frect=[xmin,ymin,xmax,ymax]` é utilizado para ajustar a escala gráfica e é igual ao argumento `rect` de `plot2d`. Se `frect` não for fornecido, o valor corrente da escala de gráficos permanece inalterado. O valor padrão de `rect` é `[0,0,1,1]` (na criação de janelas, quando se volta para o valor padrão com `xset('default')` ou quando se limpa eventos gráficos gravados `xbasc()`).

`arect=[x_left, x_right,y_up,y_down]` é utilizado para ajustar a moldura gráfica dentro da sub-janela. A moldura gráfica é especificada (como `wrect`) utilizando-se a proporção de largura ou altura da sub-janela de gráficos corrente. O valor padrão é `1/8*[1,1,1,1]`. Se `arect` não for fornecido, o valor padrão permanece inalterado.

# Exemplos

```
// para obter uma explanação gráfica dos parâmetros de xsetech entre com:
exec('SCI/modules/graphics/demos/xsetechfig.sce');

// aqui xsetech é usado para dividir a janela de gráficos em duas partes
// o primeiro xsetech é usado para se ajustar a primeira sub-janela
// e a escala de gráficos corrente
xsetech([0,0,1.0,0.5],[-5,-3,5,3])
// chamamos plot2d com a opção "001" para utilizar a escala de gráficos
// ajustada por xsetech
```

```
plot2d([1:10]',[1:10]',1,"001"," ")
// então xsetech é usado para ajustar a segunda sub-janela
xsetech([0,0.5,1.0,0.5])
// a escala de gráficos é ajustada por xsetech para [0,0,1,1] por padrão
// e nós a modificamos utilizando o argumento rect em plot2d
plot2d([1:10]',[1:10]',1,"011"," ",[-6,-6,6,6])
// quatro esboços em uma única janela de gráficos
xbasc()
xset("font",2,0)
xsetech([0,0,0.5,0.5]); plot3d()
xsetech([0.5,0,0.5,0.5]); plot2d()
xsetech([0.5,0.5,0.5,0.5]); grayplot()
xsetech([0,0.5,0.5,0.5]); histplot()
// de volta aos valores padrões para a sub-janela
xsetech([0,0,1,1])
// um esboço com arect modificado
xbasc()
xset("default")
xsetech(arect=[0,0,0,0])
x=1:0.1:10;plot2d(x',sin(x)')
xbasc()
xsetech(arect=[1/8,1/8,1/16,1/4])
x=1:0.1:10;plot2d(x',sin(x)')
xbasc()
xset("default")
```

# Ver Também

xgetech, subplot, isoview, square

# Autor

J.Ph.C.

# Name

xsetm — Abre uma caixa de diálogo para ajustar valores do contexto gráfico. **Função obsoleta.**

```
xsetm()
```

# Descrição

Esta função, bem como a função xset estão fortemente ligadas ao modo gráfico antigo, que não está mais disponível. O novo modo gráfico é muito mais flexível quanto ao ajuste de parâmetros (ver a ajuda das funções set e get bem como a página de graphics_entities). É possível iniciar um editor de propriedades mais conveniente através de ged.

# Ver Também

xset, ged, set, graphics_entities

# Autor

J.Ph.C. ENPC

# Name

xstring — imprime strings

```
xstring(x,y,str,[angle,box])
```

# Parâmetros

x,y
: escalares reais, coordenadas do ponto inferior esquerdo dos strings.

str
: matriz de strings

angle
: real, ângulo horário em graus; O valor padrão é 0.

box
: inteio, o valor padrão é 0

# Descrição

`xstring` imprime a matriz de strings `str` na localização `x,y` (ponto inferior esquerdo) na escala de gráficos corrente: cada linha da matriz significa uma linha de texto e os elementos das linhas são palavras separadas por espaços em branco. Se `angle` for dado, fornece a inclinação em graus para escrever os strings. Se `box` for 1 e `angle` for 0, uma caixa é desenhada ao redor dos strings.

# Exemplos

```
plot2d([0;1],[0;1],0)
xstring(0.5,0.5,["Scilab" "não"; "é" "esilaB"])
//outro exemplo
alphabet=["a" "b" "c" "d" "e" "f" "g" ..
          "h" "i" "j" "k" "l" "m" "n" ..
          "o" "p" "q" "r" "s" "t" "u" ..
          "v" "w" "x" "y" "z"];
xbasc()
plot2d([0;1],[0;2],0)
xstring(0.1,1.8,alphabet)     // alfabeto
xstring(0.1,1.6,alphabet,0,1) // alfabeto em uma caixa
xstring(0.1,1.4,alphabet,20)  // ângulo
xset("font",1,1)              // usando fontes symbol
xstring(0.1,0.1,alphabet)
xset("font",1,3)              // mudando tamanho da fonte
xstring(0.1,0.3,alphabet)
xset("font",1,24); xstring(0.1,0.6,"a") //alfa grande
xset("default")
```

# Ver Também

titlepage, xnumb, xstringb, xstringl, xtitle

# Autor

J.Ph.C.

# Name

xstringb — escreve strings em uma caixa

```
xstringb(x,y,str,w,h,[option])
```

# Parâmetros

x,y,w,h
: vetor de 4 escalares reais definindo a caixa

str
: matriz de strings

option
: string

# Descrição

`xstringb` desenha a matriz de strings `str` centrada dentro do retângulo `rect=[x,y,w,h]` (ponto inferior esquerdo, largura, altura) na escala gráfica corrente.

Se `option` for fornecido com o valor `"fill"`, o tamanho do caractere é computado de modo a preencher o retângulo o máximo possível.

Entre com o comando `xstringb()` para uma demonstração.

# Exemplos

```
str=["Scilab" "não";"é" "elisaB"];
plot2d(0,0,[-1,1],"010"," ",[0,0,1,1]);
r=[0,0,1,0.5];
xstringb(r(1),r(2),str,r(3),r(4),"fill");
xrect(r(1),r(2)+r(4),r(3),r(4))
r=[r(1),r(2)+r(4)+0.01,r(3),r(4)/2];
xrect(r(1),r(2)+r(4),r(3),r(4))
xstringb(r(1),r(2),str,r(3),r(4),"fill");
r=[r(1),r(2)+r(4)+0.01,r(3),r(4)/2];
xrect(r(1),r(2)+r(4),r(3),r(4))
xstringb(r(1),r(2),str,r(3),r(4),"fill");
```

# Ver Também

titlepage, xstring, xstringl, xtitle

# Autor

J.Ph.C.

# Name

xstringl — computa uma caixa que cerca strings

```
rect=xstringl(x,y,str,[fontId,fontSize])
```

# Parâmetros

rect
: vetor de reais de 4 entradas definindo a caixa

x,y
: escalares reais, coordenadas do ponto infeirior esquerdo

str
: matriz de strings

fontId
: um inteiro especificando o tipo de fonte

fontSize
: um inteiro especificando o tamanho da fonte

# Descrição

`xstringl` retorna em `rect=[x,y,w,h]` (ponto superior esquerdo, largura, altura) o tamanho do retângulo na escala de gráficos corrente que cerca os strings `str` escritos na localização `x,y` (ponto inferior esquerdo).

O resultado pode ser aproximado utilizando-se um driver Postscript.

# Exemplos

```
plot2d([0;1],[0;1],0)
str=["Scilab" "não";"é" "elisaB"];
r=xstringl(0.5,0.5,str)
xrects([r(1) r(2)+r(4) r(3) r(4)]')
xstring(r(1),r(2),str)

plot2d([0;1],[0;1],0)
str=["Scilab" "não";"é" "Matlab"];
r2 = xstringl(0.5,0.5,str,2,5)
xrects([r2(1) r2(2)+r2(4) r2(3) r2(4)]')
xstring(r2(1),r2(2),str)

txt2=gce();
txt2.font_size = 5;
txt2.font_style = 2;
```

# Ver Também

titlepage, xstring, xstringl, xtitle, stringbox

# Autor

J.Ph.C.

# Name

xtitle — adidciona títulos a janelas de gráficos

```
xtitle(title,[x_label,[y_label,[z_label]]],<opts_args>)
```

# Parâmetros

title,x_label,y_label, z_label
: matrizes de strings

<opt_args>
: uma seqüência de declarações `key1=value1, key2=value2`, ... onde keys podem ser `boxed` (ver abaixo). Nesse caso, a ordem não tem valor especial

boxed
: um valor inteiro. Se for 1, um retângulo é desenhado ao redor de cada título.

# Descrição

`xtitle` adiciona títulos a um esboço 2d ou 3d. `title` é o título geral e **x_label**, **y_label** e **z_label** são os títulos dos três eixos. Se os argumentos são matrizes, cada linha das matrizes é exibida em uma linha diferente.

Entre com o comando `xtitle()` para visualizar uma demonstração.

# Exemplos

```
// desenhando uma superfície
plot3d() ;
// pondo os títulos
xtitle( 'Minha superfície é azul', 'eixo X', 'eixo Y', 'eixo Z' ) ;
// desenhando uma caixa ao redor dos títulos
xtitle( 'Minha superfície é azul', 'eixo X', 'eixo Y', 'eixo Z' , boxed = 1 ) ;
```

# Ver Também

titlepage label_properties

# Autor

J.Ph.C.

# Name

zoom_rect — Amplia uma seleção da figura gráfica corrente

```
zoom_rect()
zoom_rect(rect)
zooom_rect(h)
zoom_rect(h,rect)
```

# Parâmetros

rect
: vetor de tamanho 4 `[xmin,ymin,xmax,ymax]` fornece o retângulo a ser ampliado

h
: manipulador gráfico do tipo Figure ou Axes. Especifica sobre que eixos se realizará a ampliação.

# Descrição

`zoom_rect` é a função utilizada para realizar ampliações dentro de um conjunto de objetos Axes.

O argumento de entrada `h` especifica sobre que eixos se realizará a ampliação. Se `h` for um manipulador do tipo Figure, então a ampliação ocorrerá sobre seus galhos Axes. Se `h` for um manipulador Axes, a ampliação será aplicada apenas sobre este objeto. Se `h` não for especificado, então a ampliação será aplicada sobre a entidade Figure corrente.

Se o argumento de entrada `rect` for especificada, então a propriedade `zoom_box` dos eixos ampliados será modificada por este argumento (ver axes_properties). Seus limites ao longo dos eixos X e Y serão substituídos por `rect`. Se `rect` não for especificado zoom_rect será uma ampliação interativa. Requer-se que o usuário selecione um retângulo utilizando o mouse. A nova propriedade `zoom_box` dos eixos ampliados é então computada encontrando-se as interseções do retângulo com suas caixas de eixos.

# Exemplos

```
clf()
x=0:0.01:6*%pi;
plot2d(x,sin(x^2))
zoom_rect([16,-1,18,1])
//mais zoom
zoom_rect([16,0,16.2,1])
//de volta ao original
unzoom()
// ajustando o zoom através de axes_properties
a=gca();
a.zoom_box=[16,0,16.2,1];
a.zoom_box=[];

//ajustando zoom de sub-esboços
clf()
x=0:0.01:6*%pi;
subplot(211)
plot2d(x,cos(x))
subplot(212)
plot2d(x,cos(2*x))
```

```
rect=[3 -2 7 10]; //um retângulo especificado nas coordenadas dos eixos corrente
zoom_rect(rect)
unzoom()
//ajustando os eixos subjascentes globais como sendo os correntes
f=gcf();set('current_axes',f.children($))
rect=[0.4 0 0.6 1] //um retângulo especificado em razão do tamanho da janela
zoom_rect(rect)
rect=[0.4 0.2 0.6 0.8]; //um retângulo especificado em razão do tamanho da jane
zoom_rect(rect)

// ampliação interativa sob a figura corrente
zoom_rect();
// ou
zoom_rect(gcf());
```

# Ver Também

unzoom, axes_properties

# Autor

Serge Steer INRIA

Jean-Baptiste Silvy INRIA

# Name

Math rendering in Scilab graphics — Display mathematical equations in Scilab graphics through the LaTeX or MathML languages.

# Usage

Starting from Scilab 5.2, it is possible to write LaTeX or MathML expression.

LaTeX texts must start and end by $ (dollar symbol) while MathML texts must start by < and end by > and being syntactically valide.

On the first use (these libraries are loading on the fly only when needed), note that the MathML engine is slower to load than LaTeX.

```
// Example with LaTeX / MathML ticks:
plot2d();
a=gca();

mathml="<mrow>;<mfrac><mrow><mi>d</mi><mi>y</mi></mrow><mrow><mi>d</mi><mi>x</m
<mfrac><mn>1</mn><msup><mi>y</mi><mn>2</mn></msup></mfrac></mrow>";
// LaTeX and MathML mixed expression
a.x_ticks.labels=[mathml;"1";"$\sin(x)$";"3";"$\cos(a) - test$";"5";"6";"7"];
```

# LaTeX description

The rendering engine is based on the Java library JLaTeXMath. JLaTeXMath is an implementation of the mathematic mode of LaTeX. All LaTeX base commands are handle (don't hesitate to submit a bug report if missing). On the contrary, TeX commands, like `\over are not supported.`

```
xtitle('$\textstyle\sum_{n=1}^{+\infty}\frac1{n^2}=\frac{\pi^2}{6}$')
xtitle('$\big(\bigg)$')
xtitle('$\mbox{Vector field for }\ddot{\theta}=\sin\theta$')
xtitle('$\JLaTeXMath\ \mathfrak{and}\ \mathtt{Scilab}$')
```

JLaTeXMath provides several fonts with the commands `\mathbb`, `\mathscr`, `\mathcal`, `\mathbf`, `\mathit`, `\mathsf`, `\mathtt`, `\mathfrak`, `\mathds`, `\mathrm`, with their bold versions when they are available with the command `\boldsymbol`:

```
xtitle('$\mathbb{SCILAB}\ \mathsf{or}\ \boldsymbol{\mathfrak{Scilab}}$')
xtitle('$\mathscr{C}\mbox{ n''est pas }\boldsymbol{\mathcal{C}}$')
```

Different LaTeX packages are available: *`amsmath`*, *`amssymb`*, *`stmaryrd`*, *`amsxtra`* and *`accents`* with some commands of *`graphics`*. Most of the commands of these packages are available (some of *`amsmath`* are missing for example).

```
xtitle('$\sideset{_\alpha^\beta}{_\gamma^\delta}\prod$')
xtitle('$\hat{\accentset{\star}{\hat h}}\undertilde{ABC}$')
xtitle('$\begin{pmatrix}\mathfrak{a}&\alpha\\\mathbb{A}&\mathcal{A}\end{pmatrix
\begin{bmatrix}\mathfrak{a}&\alpha\\\mathbb{A}&\mathcal{A}\end{bmatrix}$')
xstring(0.5,0.5,'$\left(\frac{\pi}{\sqrt[3]{2}}\middle|\sqrt{\frac{1+\frac1x}{x
xtitle('$\doublecup\ddag\fatbslash\lll\oplus\ovee\circledcirc\circlearrowright$
```

```
xtitle('$\rotatebox{180}{\boxed{\JLaTeXMath}}\ \reflectbox{\JLaTeXMath}$')
xtitle('$\scalebox{0.6}{\sum_{n=1}^{+\infty}\frac1{n^2}=\frac{\pi^2}6}$')
xtitle('$\fcolorbox{black}{Tan}{\JLaTeXMath}$')
xtitle('$\textcolor{Magenta}{\mathfrak{Scilab}}\mbox{ and }\textcolor{Green}{\ma
```

It is also possible to define new commands or new environments:

```
xtitle('$\newcommand{\op}{\left(}\newcommand{\cp}{\right)} \op\frac12\cp$')
xtitle('$\newcommand{\myfrac}[2]{\frac{\mathfrak{#1}}{\mathcal{#2}}}\myfrac{A}{
```

# MathML description

The MathML rendering is based on Jeuclid. Jeuclid is a MathML implementation which covers the whole specification. Therefor, all the MathML language is supported within Scilab.

Due to the size of the Jeuclid library, on the first use, it can take up to a few seconds to load. However, next uses are much faster.

```
plot3d();
a=get("current_axes");
a.x_label.font_size= 5;
a.x_label.text="<mrow><mfrac><mrow><mn>1</mn></mrow><mrow><mn>2</mn></mrow></mf
<mrow><mfrac><mrow><mfrac><mrow><mi>a</mi></mrow><mrow><mi>b</mi></mrow></mfrac
<mi>c</mi></mrow><mrow><mi>d</mi></mrow></mfrac></mrow></mfrac></mrow></mfenced
```

# See Also

xtitle, axes_properties, label_properties, legend_properties, text_properties, xstringb, xstringl, xstring

# Parte VII. Gráficos : exportando e imprimindo

# Name

driver — seleciona um driver gráfico

```
driver(driver_name)
current_driver=driver()
```

# Parâmetros

driver_name
: string, driver a ser selecionado.

# Descrição

Esta função seleciona um driver gráfico, ou, sem argumentos, retorna o nome do driver gráfico corrente. Na maioria das vezes, o usuário pode ignorar esta função e modificar o driver por uma chamada a funções de alto nível como `xbasc`. O driver selecionado pode ser um dos seguintes:

"X11"
: saída para a tela do computador.

"Pos"
: saída em formato Postscript.

"Rec"
: saída para a tela do computador. É o mesmo que `X11`.

"Fig"
: saída em formato XFig.

"GIF"
: saída em formato Gif.

"PPM"
: saída em formato PPM.

# Observação

Para converter arquivos "GIF" ou "PPM" para outro formato de imagem para construir uma animação pode-se utilizar o programa "convert" para ImageMagic (http://www.imagemagick.org/)

Por exemplo, se é gerada uma seqüência de arquivos Gif nomeada `img*.gif` é possível construir um arquivo Gif animado (chamado anim.gif) por

```
convert -delay 10  img*.gif anim.gif
```

# Ver Também

xbasc

# Autor

J.Ph.C.

# Name

xend — termina uma sessão de gráficos

```
xend()
```

# Descrição

`xend` é usado para encerrar uma sessão de gráficos. Sob os drivers Postscript, Xfig ou Gif `xend` fecha o arquivo que foi aberto por `xinit`.

# Exemplos

```
driver("Pos")
xinit("foo.ps")
plot2d()
xend()
driver("X11")
```

# Ver Também

xinit

# Autor

J.Ph.C.

# Name

xinit — inicialização de um driver de gráficos

```
xinit(FileName)
xinit()
```

# Parâmetros

FileName
: string: nome do arquivo a exportar

# Descrição

Para os drivers Postscript, Xfig, Gif ou PPM, `FileName` deve ser especificado. É o nome do arquivo onde todas as operações gráficas são registradas.

Para os drivers (X11 ou Rec), `xinit` deve ser chamado sem nenhum argumento e abre uma janela de gráficos vazia.

# Exemplos

```
driver("Pos")
xinit("foo.ps")
plot2d()
xend()
driver("X11")
```

# Ver Também

driver, xend, scf

# Autores

J.Ph.C.

Jean-Baptiste Silvy

# Name

xs2bmp — envia gráficos para um arquivo em sintaxe BMP

```
xs2bmp(win_num,filen)
```

# Parâmetros

win_num
: escalar inteiro

filen
: string, nome do arquivo

# Descrição

`xs2bmp` envia os gráficos gravados em uma janela `win_num` para o arquivo `filen` no formato BMP.

# Exemplos

```
scf(0)
plot2d()
//Exportando para BMP
xs2bmp(0,'foo.bmp');
```

# Ver Também

xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

# Autor

A.C

# Name

xs2emf — envia gráficos para um arquivo em sintaxe EMF (apenas para Windows)

```
xs2emf(win_num,filen [,orientation])
```

# Parâmetros

win_num
: escalar inteiro

filen
: string, o nome do arquivo.

orientation
: caractere opcional com valores possíveis 'p' (retrato) ou 'l' (paisagem). O valor padrão é 'p'.

# Descrição

`xs2emf` envia os gráficos gravados em uma janela `win_num` para o arquivo `filen` no formato EMF.

Para o formato EMF, criamos um arquivo EPS que será convertido para o formato EMFpor pstoedit.

# Exemplos

```
if MSDOS then
  scf(0);
  plot2d();
  //Exportando para EMF
  xs2emf(0,'foo.emf');
end
```

# Ver Também

xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig

# Autor

A.C

# Name

xs2eps — salva gráficos em um arquivo Postscript

```
xs2eps(win_num,filen [,orientation])
```

# Parâmetros

win_num
: vetor de inteiros ou escalar inteiro

filen
: string, nome do arquivo

orientation
: caractere opcional, com possíveis valores 'p' (retrato) ou 'l' (paisagem). O valor padrão é 'p'.

# Descrição

`xs2eps` salva os gráficos gravados da janela `win_num` em um arquivo `filen` em sintaxe Postscript. Note que `filen` não deve ter extensão.

`xs2eps` produz um arquivo Postscript encapsulado completo.

# Exemplos

```
scf(0);
plot2d();
//Exportando para EPS
filename='foo.eps';
xs2eps(0,filename);
```

# Ver Também

figure_size property, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

# Name

xs2fig — envia gráficos para um arquivo em sintaxe FIG

```
xs2fig(win_num, filen [,orientation])
```

# Parâmetros

win_num
: escalar inteiro

filen
: string, nome do arquivo.

orientation
: caractere opcional, com valores possíveis 'p' (retrato) ou 'l' (paisagem). O valor padrão é 'p'.

# Descrição

`xs2fig` envia os gráficos gravados da janela `win_num` para o arquivo `filen` em formato FIG.

Para o formato FIG, criamos um arquivo EPS que será convertido para o formato FIG por pstoedit.

Para exportar arquivos FIG, o GPL Ghostscript (32bits) precisa estar instalado.

Link para GPL Ghostscript : http://www.ghostscript.com/awki

# Exemplos

```
//exemplo simples
scf(0);
plot2d();
xs2fig(0,'foo.fig');
```

# Ver Também

xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2emf

# Autor

S.K

# Name

xs2gif — envia gráficos a um arquivo em sintaxe GIF

```
xs2gif(win_num,filen)
```

# Parâmetros

win_num
: escalar inteiro ou vetor de inteiros.

filen
: string, nome do arquivo

# Descrição

`xs2gif` envia os gráficos gravados da janela window `win_num` para o arquivo `filen` em formato GIF.

Para converter uma seqüência de arquivos GIF em um arquivo GIF animado, pode-se usar o programa "convert" para ImageMagic (http://www.imagemagick.org/)

Por exemplo, se for gerada uma seqüência de arquivos GIF chamada `img*.gif` é possível construir um arquivo GIF animado (chamado anim.gif) através da seqüência

```
convert -delay 10  img*.gif anim.gif
```

# Exemplos

```
scf(0)
plot2d()
//Exportando para GIF
xs2gif(0,'foo.gif');
```

# Ver Também

xs2bmp, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

# Name

xs2jpg — envia gráficos a um arquivo em sintaxe JPG

```
xs2jpg(win_num,filen)
```

# Parâmetros

win_num
: escalar inteiro

filen
: string, nome do arquivo

# Descrição

`xs2jpg` envia os gráficos gravados da janela `win_num` para o arquivo `filen` em formato JPG.

# Exemplos

```
scf(0);
plot2d();
//Exportando para JPG
xs2jpg(0,'foo.jpg');
```

# Ver Também

xs2bmp, xs2gif, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

# Autor

S.K

# Name

xs2pdf — salva gráficos a um arquivo PDF

```
xs2pdf(win_num,filen [,orientation])
```

# Parâmetros

win_num
: escalar inteiro

filen
: string, nome do arquivo

orientation
: caractere opcional, com possíveis valores 'p' (retrato) ou 'l' (paisagem). O valor padrão é 'p'.

# Descrição

`xs2pdf` salva os gráficos gravados da janela `win_num` em um arquivo `filen` in PDF syntax. em sintaxe Postscript. Note que `filen` não deve ter extensão.

# Exemplos

```
scf(0);
plot2d();
//Exportando para PDF
filename='foo'; // Sem extensão !
xs2pdf(0,filename);
```

# Ver Também

figure_size property, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2svg, xs2ps, xs2fig, xs2emf

# Name

xs2png — envia gráficos a um arquivo em sintaxe PNG

```
xs2png(win_num,filen)
```

# Parâmetros

win_num
: escalar inteiro

filen
: string, nome do arquivo

# Descrição

`xs2png` envia os gráficos gravados da janela `win_num` em um arquivo `filen` em formato PNG.

# Exemplos

```
scf(0)
plot2d()
//Exportando para PNG
xs2png(0,'foo.png');
```

# Ver Também

xs2bmp, xs2gif, xs2jpg, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

# Autor

S.K

# Name

xs2ppm — envia gráficos para um arquivo em sintaxe PPM

```
xs2ppm(win_num,filen)
```

# Parâmetros

win_num
: vetor de inteiros ou escalar inteiro

filen
: string, o nome do arquivo

# Descrição

`xs2ppm` envia os gráficos gravados em uma janela `win_num` para o arquivo `filen` em formato PPM.

# Exemplos

```
scf(0)
plot2d()
//exportando para PPM
xs2ppm(0,(foo.ppm');
```

# Ver Também

xs2bmp, xs2gif, xs2jpg, xs2png, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

# Name

xs2ps — envia gráficos a um arquivo em sintaxe PS

```
xs2ps(win_num,filen,[orientation])
```

# Parâmetros

win_num
: vetor de inteiros ou escalar inteiro

filen
: string, nome do arquivo

orientation
: caractere opcional, com possíveis valores 'p' (retrato) ou 'l' (paisagem). O valor padrão é 'p'.

# Descrição

`xs2ps` salva os gráficos gravados da janela `win_num` em um arquivo `filen` em sintaxe Postscript. Note que `filen` não deve ter extensão.

Note que o arquivo Postscript gerado não pode ser impresso diretamente desde requer cabeçalho. A função `xs2eps` pode ser usada diretamente para produzir um arquivo Postscript encapsulado com cabeçalho.

# Exemplos

```
scf(0);
plot2d();
// exportando para Postscript
filename=foo.ps;
xs2ps(0,filename);
```

# Ver Também

figure_size property, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2fig, xs2emf

# Name

xs2svg — Salva gráficos em um arquivo SVG

```
xs2svg(win_num,filen [,orientation])
```

# Parâmetros

win_num
: vetor de inteiros ou escalar inteiro

filen
: string, nome do arquivo

orientation
: caractere opcional, com possíveis valores 'p' (retrato) ou 'l' (paisagem). O valor padrão é 'p'

# Descrição

`xs2svg` salva os gráficos gravados da janela `win_num` em um arquivo `filen` em sintaxe Postscript. Note que `filen` não deve ter extensão.

# Exemplos

```
scf(0)
plot2d()
//exportando para SVG
filename='foo.svg'
xs2svg(0,filename);
```

# Ver Também

figure_size property, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2ps, xs2fig, xs2emf

# Parte VIII. Booleanos

# Name

bool2s — converte uma matriz de valores booleanos para uma matriz de valores 0 ou 1

```
bool2s(x)
```

# Parâmetros

x
: um vetor ou matriz de valores booleanos ou uma matriz constante

# Descrição

Se `x` é uma matriz de valores booleanos, `bool2s(x)` retorna uma matriz onde os valores "true" ("verdadeiro") são substituidos por 1 e os valores "false" ("falso") são substituidos por 0.

Se `x` é uma matriz "padrão", `bool2s(x)` retorna uma matriz onde os valores não-nulos são substituídos por 1.

# Exemplos

```
bool2s([%t %t %f %t])
bool2s([2.3 0 10 -1])
```

# Ver Também

boolean, find

# Name

find — encontra índices de elementos verdadeiros em uma matriz ou vetor de booleanos

```
[ii]=find(x [,nmax])
[i1,i2,..]=find(x [,nmax])
```

# Parâmetros

x
: pode ser um vetor, matriz ou hipermatriz de booleanos, uma matriz ou hipermatriz "padrão".

nmax
: um inteiro fornecendo o número máximo de índices a serem retornados. o valor padrão é -1, que significa "todos". Esta opção pode ser usada por eficiência, para evitar uma busca por todos os índices.

ii, i1, i2, ..
: vetores de índices inteiros ou matrizes vazias

# Descrição

Se `x` é uma matriz booleana,

`ii=find(x)` retorna o vetor de índices `i` para os quais `x(i)` é "true" ("verdadeiro").Se nenhum elemento "true" for encontrado, retorna uma matriz vazia.

`[i1,i2,..]=find(x)` retorna vetores de índices `i1` (para linhas) e `i2` (para colunas),... tais que `x(i1(n),i2(n),..)` ié "true" ("verdadeiro"). Se nenhum elemento "true" for encontrado, retorna matrizes vazias em `i1`, `i2`, ...

Se `x` é uma matriz ou hipermatriz padrão `find(x)` é interpretado como `find(x<>0)`

`find([])` retorna `[]`

# Exemplos

```
A=rand(1,20);
w=find(A<0.4)
A(w)
w=find(A>100)

B=rand(1,20);
w=find(B<0.4,2) //no máximo dois valores retornados

H=rand(4,3,5); //uma hipermatriz
[i,j,k]=find(H>0.9)

H(i(1),j(1),k(1))
```

# Ver Também

boolean, extraction, insertion, vectorfind

# Parte IX. CACSD

# Name

black — diagrama de Black (carta de Nichols)

```
black( sl,[fmin,fmax] [,step] [,comments] )
black( sl,frq [,comments] )
black(frq,db,phi [,comments])
black(frq,repf [,comments])
```

# Parâmetros

sl
: lista ( sistema linear `syslin`)

fmin,fmax
: reais (limites de freqüência)

frq
: vetor linha ou matriz (freqüências)

db,phi
: vetores linhas ou matrizes (módulo, fase)

repf
: vetores linhas ou matrizes (resposta de freqüência complexa)

step
: real

comments
: string

# Descrição

Diagrama de Black (carta de Nichols) para um sistema linear `sl`. `sl` pode ser um sistema SIMO de tempo contínuo ou discreto (ver `syslin`). No caso de múltiplas saídas, elas são esboçadas com símbolos diferentes.

As freqüências são dadas pelos limites `fmin`,`fmax` (em Hz) ou por um vetor linha (ou uma matriz para múltiplas saídas) `frq`.

`step` é o passo de discretização ( logarítmica). (ver `calfrq` para escolha do valor padrão).

`comments` é um vetor de strings (legendas).

`db,phi` são matrizes de módulos (em Db) e fases (em graus). (Uma linha para cada resposta).

`repf` matriz de números complexos. Uma linha para cada resposta.

Para esboçar o grid de iso-ganho e iso-fase de `y/(1+y)` use chart().

Valores padrões para `fmin` e `fmax` são `1.d-3`, `1.d+3` se `sl` for de tempo contínuo ou `1.d-3`, `0.5`/sl.dt (freqüência de Nyquist) se `sl` for de tempo discreto.

# Exemplos

```
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
```

```
clf();black(h,0.01,100);
chart(list(1,0));

h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
clf()
black([h1;h],0.01,100,['h1';'h'])
chart(list(1,0));
```

## Ver Também

bode, nyquist, chart, freq, repfreq, calfrq, phasemag

# Name

bode — diagrama de Bode

```
bode(sl,[fmin,fmax] [,step] [,comments] )
bode(sl,frq [,comments] )
bode(frq,db,phi [,comments])
bode(frq, repf [,comments])
```

# Parâmetros

sl
: `syslin` lista (sistema linear SISO ou SIMO) em tempo contínuo ou discreto

fmin,fmax
: real (limites de freqüência (em Hz))

step
: real (passo logarítmico.)

comments
: vetor de strings (legendas).

frq
: vetor linha ou matriz (freqüências (em Hz) ) (uma linha para cada subsistema SISO).

db
: vetor linha ou matriz ( magnitudes (em Db)). (um para cada subsistema SISO).

phi
: vetor linha ou matriz ( fases (em graus)) (um para cada subsistema SISO ).

repf
: vetor linha ou matriz de números complexos (freqüência de resposta complexa).

# Descrição

Diagrama de bode , i.e magnitude e fase da freqüência de resposta de `sl`.

`sl` pode ser um sistema SIMO de tempo contínuo ou discreto (ver `syslin`). No caso de múltiplas saídas, elas são esboçadas com símbolos diferentes.

As freqüências são dadas pelos limites `fmin,fmax` (em Hz) ou por um vetor linha (ou uma matriz para múltiplas saídas) `frq`.

`step` é o passo de discretização ( logarítmica). (ver `calfrq` para escolha do valor padrão).

`comments` é um vetor de strings (legendas).

`db,phi` são matrizes de módulos (em Db) e fases (em graus). (Uma linha para cada resposta).

`repf` matriz de números complexos. Uma linha para cada resposta.

Valores padrões para `fmin` e `fmax` são `1.d-3`, `1.d+3` se `sl` for de tempo contínuo ou `1.d-3`, `0.5`/sl.dt (freqüência de Nyquist) se `sl` for de tempo discreto. Discretização automática das freqüências é feita por `calfrq`.

# Exemplos

```
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
tit='(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)';
bode(h,0.01,100,tit);
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
clf()
bode([h1;h],0.01,100,['h1';'h'])
```

# Ver Também

black, nyquist, gainplot, repfreq, g_margin, p_margin, calfrq, phasemag

# Name

chart — carta de Nichols

```
chart([flags])
chart(gain [,flags])
chart(gain,phase [,flags])
```

# Parâmetros

gain
: vetor de reais ( ganhos (em DB, decibel))

phase
: vetor de reais (fases (em graus))

flags
: uma lista de no máximo quatro flags (sup [,leg [,cm [,cphi]]])

  sup
  : 1 indica superposição sobre o esboço anterior e 0 indica que nenhuma superposição é feita

  leg
  : 1 indica que as legendas são impressas, 0 indica sem legendas

  cm
  : número da cor para curvas de ganho

  cphi
  : número da cor para curvas de fase

# Descrição

Esboça a carta de Nichols: curvas de nível de iso-ganho e iso-fase dey/(1+y) no plano de fase/ganho.

`chart` pode ser utilizado em conjunção a black.

Os valore padrões para `gain` e `phase` são respectivamente:

`[-12 -8 -6 -5 -4 -3 -2 -1.4 -1 -.5 0.25 0.5 0.7 1 1.4 2 2.3 3 4 5 6 8 12]`

`[-(1:10) , -(20:10:160)]`

# Exemplos

```
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
black(h,0.01,100)
chart(list(1,0,2,3));

clf()
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
black([h1;h],0.01,100,['h1';'h'])
set(gca(),'data_bounds',[-180 -30;180 30]) //aumentando a moldura
chart(list(1,0));
```

## Ver Também

nyquist, black

# Name

evans — lugar geométrico das raízes Evans

```
evans(H [,kmax])
```

# Parâmetros

H
: lista (sistema linear `syslin`)

kmax
: real (ganho máximo desejado para o esboço)

# Descrição

Fornece o lugar geométrico das raízes Evans para um sistema linear em estado-espaço ou forma de transferência H(s) (lista `syslin`). Este é o lugar geométrico das raízes de `1+k*H(s)=1+k*N(s)/D(s)`, no plano dos complexos. Para uma amostra seleta de ganhos `k <= kmax`, a parte imaginária das raízes de `D(s)+k*N(s)` é esboçada versus a parte real.

Para obter o ganho num dado ponto do lugar geométrico, você pode simplesmente executar a seguinte instrução: `k=-1/real(horner(h,[1,%i]*locate(1)))` e clicar no ponto desejado sobre lugar geométrico das raízes. Se as coordenadas dos pontos selecionados estão no vetor de reais 2 x 1 `P=locate(1)` este `k` resolve a equação `k*N(w) + D(w) =0` com `w=P(1)+%i*P(2)=[1,%i]*P`.

# Exemplos

```
H=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));
evans(H,100)
P=3.0548543 - 8.8491842*%i;    //P=ponto selecionado
k=-1/real(horner(H,P));
Ns=H('num');Ds=H('den');
roots(Ds+k*Ns)     //contém P como raiz particular
// outro
clf();s=poly(0,'s');n=1+s;
d=real(poly([-1 -2 -%i %i],'s'));
evans(n,d,100);
//
clf();n=real(poly([0.1-%i 0.1+%i,-10],'s'));
evans(n,d,80);
```

# Ver Também

kpure, krac2, locate

# Name

gainplot — esboço de magnitude

```
gainplot(sl,fmin,fmax [,step] [,comments] )
gainplot(frq,db,phi [,comments])
gainplot(frq, repf [,comments])
```

# Parâmetros

sl
: lista (sistema linear `syslin` SIMO).

fmin,fmax
: escalares reais (intervalo de freqüência).

step
: real (passo de discretização (escala logarítmica))

comments
: string

frq
: matriz (freqüências linha por linha)

db,phi
: matrizes (magnitudes e fases correspondentes a `frq`)

repf
: matriz de complexos. Uma linha para cada resposta de freqüência.

# Descrição

É o mesmo que Bode, mas esboça apenas a magnitude.

# Exemplos

```
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
gainplot(h,0.01,100,'(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)')
clf()
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
gainplot([h1;h],0.01,100,['h1';'h'])
```

# Ver Também

bode, black, nyquist, freq, repfreq, g_margin, p_margin

# Name

m_circle — esboça os contornos de iso-ganho y/(1+y) plano complexo

```
m_circle()
m_circle(gain)
```

# Parâmetros

gain
: vetor de ganhos (em DB, decibel). O valor padrão é

  gain
  : =[-12 -8 -6 -5 -4 -3 -2 -1.4 -1 -.5 0.25 0.5 0.7 1 1.4 2 2.3 3 4 5 6 8 12]

# Descrição

`m_circle` desenha os contornos de iso-ganho fornecidos pelo argumento `gain` no plano complexo (Re,Im).

O valor padrão para `gain` é:

`[-12 -8 -6 -5 -4 -3 -2 -1.4 -1 -.5 0.25 0.5 0.7 1 1.4 2 2.3 3 4 5 6 8 12]`

`m_circle` é utilizado com `nyquist`.

# Exemplos

```
//Exemplo 1 :
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
nyquist(h,0.01,100,'(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)')
m_circle();
//Exemplo 2:
xbasc();
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
nyquist([h1;h],0.01,100,['h1';'h'])
m_circle([-8 -6 -4]);
```

# Ver Também

nyquist, chart, black

# Autor

S.Steer.;

# Name

nyquist — diagrama de Nyquist

```
nyquist( sl,[fmin,fmax] [,step] [,comments] )
nyquist( sl, frq [,comments] )
nyquist(frq,db,phi [,comments])
nyquist(frq, repf [,comments])
```

# Parâmetros

sl
: `syslin` (sistema linear SIMO em tempo contínuo ou discreto)

fmin,fmax
: reais (limites de freqüência (em Hz))

step
: real (passo de discretização logarítmica)

comments
: vetor de strings (legendas).

frq
: vetor linha ou matriz (freqüências (em Hz) ) (um linha para cada saída de `sl`).

db,phi
: vetores linhas ou matrizes de módulo (em Db) e fases. (uma linha para cada saída de `sl`).

repf
: vetor linha ou matriz de números complexos (freqüência de resposta complexa, uma para cada saída de `sl`)

# Descrição

Diagrama de Nyquist i.e parte imaginária versus parte real da freqüência de resposta de `sl`.

Para sistmeas de tempo contínuo `sl(2*%i*%pi*w)` é esboçado. Para sistemas de tempo discreto ou sistemas discretizados `sl(exp(2*%i*%pi*w*fd)` é utilizado ( `fd=1` para sistemas de tempo discreto e `fd=sl('dt')` para sistemas discretizados)

`sl` para sistemas discretizados `syslin`). No caso de múltiplas saídas, elas são esboçadas com símbolos diferentes.

As freqüências são dadas pelos limites `fmin,fmax` (em Hz) ou por um vetor linha (ou uma matriz para múltiplas saídas) `frq`.

`step` é o passo de discretização ( logarítmica). (ver `calfrq` para escolha do valor padrão).

`comments` é um vetor de strings (legendas).

`db,phi` são matrizes de módulos (em Db) e fases (em graus). (Uma linha para cada resposta).

`repf` é uma matriz de números complexos. Uma linha para cada resposta.

Valores padrões para `fmin` e `fmax` são `1.d-3`, `1.d+3` se `sl` for de tempo contínuo `1.d-3`, `0.5`/sl.dt (freqüência de Nyquist) se `sl` for de tempo discreto.

Discretização automática das freqüências é feita por `calfrq`.

# Exemplos

```
clf();
s=poly(0,'s');
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01));
comm='(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)';
nyquist(h,0.01,100,comm);
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
clf();
nyquist([h1;h],0.01,100,['h1';'h'])
clf();nyquist([h1;h])
```

# Ver Também

bode, black, calfrq, freq, repfreq, phasemag

# Name

routh_t — tabela de Routh

```
r=routh_t(h [,k]).
```

# Parâmetros

h
: matriz quadrada de razões de polinômios

# Descrição

`r=routh_t(h,k)` computa a tabela de Routh do denominador do sistema descrito pela matriz de transferência SISO `h` com a resposta pelo ganho `k`.

se `k=poly(0,'k')` teremos uma matriz de polinômios com variável livre `k`, expressão formal da tabela de Routh.

# Name

sgrid — esboça linhas de grid de um s-plano

```
sgrid()
sgrid('new')
sgrid(zeta,wn [,color])
```

# Descrição

Usado em conjunto com evans, esboça linhas de taxa de amortecimento constante (`zeta`) e freqüência natural (`wn`).

sgrid()
: adiciona um grid sobre uma raiz do s-plano contínuo existente com valores padrões para `zeta` e `wn`.

sgrid('new')
: limpa a tela de gráficos e então esboça um grid de s-plano padrão

sgrid(zeta,wn [,color])
: é o mesmo que `sgrid()` mas utiliza a taxa de amortecimento e a freqüência natural fornecidas.

# Exemplos

```
H=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));
evans(H,100)
sgrid()
sgrid(0.6,2,7)
```

# Ver Também

evans

# Name

zgrid — esboçode um z-grid

```
zgrid()
```

# Descrição

Esboça as linhas de grid de um Z-plano: linhas de fator constante de amortecimento (zeta) e freqüência natural (Wn) são esboçadas dentro do círculo unitário do Z-plano.

Curvas de iso-freqüência são exibidas em freqüência*passo no intervalo [0,0.5]. O limite superior corresponde à freqüência de Shannon `( 1/dt > 2*f )`.

# Ver Também

frep2tf, freson

# Name

abcd — state-space matrices

```
[A,B,C,D]=abcd(sl)
```

# Parameters

sl
: linear system (`syslin` list) in state-space or transfer form

A,B,C,D
: real matrices of appropriate dimensions

# Description

returns the `A,B,C,D` matrices from a linear system `Sl`.

Utility function. For transfer matrices `Sl` is converted into state-space form by `tf2ss`.

The matrices `A,B,C,D` are the elements `2` to `5` of the `syslin` list `Sl`, i.e. `[A,B,C,D] = Sl(2:5)`.

# Examples

```
A=diag([1,2,3]);B=[1;1;1];C=[2,2,2];
sys=syslin('c',A,B,C);
sys("A")
sys("C")
[A1,B1,C1,D1]=abcd(sys);
A1
systf=ss2tf(sys);
[a,b,c,d]=abcd(systf)
spec(a)
c*b-C*B
c*a*b-C*A*B
```

# See Also

syslin , ssrand

# Name

abinv — AB invariant subspace

```
[X,dims,F,U,k,Z]=abinv(Sys,alpha,beta,flag)
```

# Parameters

Sys
: `syslin` list containing the matrices `[A,B,C,D]`.

alpha
: (optional) real number or vector (possibly complex, location of closed loop poles)

beta
: (optional) real number or vector (possibly complex, location of closed loop poles)

flag
: (optional) character string `'ge'` (default) or `'st'` or `'pp'`

X
: orthogonal matrix of size nx (dim of state space).

dims
: integer row vector `dims=[dimR,dimVg,dimV,noc,nos]` with `dimR<=dimVg<=dimV<=noc<=nos`. If `flag='st'`, (resp. `'pp'`), `dims` has 4 (resp. 3) components.

F
: real matrix (state feedback)

k
: integer (normal rank of `Sys`)

Z
: non-singular linear system (`syslin` list)

# Description

Output nulling subspace (maximal unobservable subspace) for `Sys` = linear system defined by a syslin list containing the matrices [A,B,C,D] of `Sys`. The vector `dims=[dimR,dimVg,dimV,noc,nos]` gives the dimensions of subspaces defined as columns of `X` according to partition given below. The `dimV` first columns of `X` i.e `V=X(:,1:dimV)`, span the AB-invariant subspace of `Sys` i.e the unobservable subspace of `(A+B*F,C+D*F)`. (`dimV=nx` iff `C^(-1)(D)=X`).

The `dimR` first columns of `X` i.e. `R=X(:,1:dimR)` spans the controllable part of `Sys` in `V`, `(dimR<=dimV)`. (`dimR=0` for a left invertible system). `R` is the maximal controllability subspace of `Sys` in `kernel(C)`.

The `dimVg` first columns of `X` spans `Vg`=maximal AB-stabilizable subspace of `Sys`. `(dimR<=dimVg<=dimV)`.

`F` is a decoupling feedback: for `X=[V,X2]` `(X2=X(:,dimV+1:nx))` one has `X2'*(A+B*F)*V=0 and (C+D*F)*V=0`.

The zeros od `Sys` are given by : `X0=X(:,dimR+1:dimV); spec(X0'*(A+B*F)*X0)` i.e. there are `dimV-dimR` closed-loop fixed modes.

If the optional parameter `alpha` is given as input, the `dimR` controllable modes of `(A+BF)` in `V` are set to `alpha` (or to `[alpha(1), alpha(2), ...]`. (`alpha` can be a vector (real or complex pairs) or a (real) number). Default value `alpha=-1`.

If the optional real parameter `beta` is given as input, the `noc-dimV` controllable modes of `(A+BF)` "outside" `V` are set to `beta` (or `[beta(1),beta(2),...]`). Default value `beta=-1`.

In the `X,U` bases, the matrices `[X'*(A+B*F)*X,X'*B*U;(C+D*F)*X,D*U]` are displayed as follows:

```
[A11,*,*,*,*,*]  [B11 * ]
[0,A22,*,*,*,*]  [0   * ]
[0,0,A33,*,*,*]  [0   * ]
[0,0,0,A44,*,*]  [0  B42]
[0,0,0,0,A55,*]  [0   0 ]
[0,0,0,0,0,A66]  [0   0 ]

[0,0,0,*,*,*]    [0  D2]
```

where the X-partitioning is defined by `dims` and the U-partitioning is defined by `k`.

`A11` is `(dimR x dimR)` and has its eigenvalues set to `alpha(i)'s`. The pair `(A11,B11)` is controllable and `B11` has `nu-k` columns. `A22` is a stable `(dimVg-dimR x dimVg-dimR)` matrix. `A33` is an unstable `(dimV-dimVg x dimV-dimVg)` matrix (see `st_ility`).

`A44` is `(noc-dimV x noc-dimV)` and has its eigenvalues set to `beta(i)'s`. The pair `(A44,B42)` is controllable. `A55` is a stable `(nos-noc x nos-noc)` matrix. `A66` is an unstable `(nx-nos x nx-nos)` matrix (see `st_ility`).

`Z` is a column compression of `Sys` and `k` is the normal rank of `Sys` i.e `Sys*Z` is a column-compressed linear system. `k` is the column dimensions of `B42,B52,B62` and `D2`. `[B42;B52;B62;D2]` is full column rank and has rank `k`.

If `flag='st'` is given, a five blocks partition of the matrices is returned and `dims` has four components. If `flag='pp'` is given a four blocks partition is returned. In case `flag='ge'` one has `dims=[dimR,dimVg,dimV,dimV+nc2,dimV+ns2]` where `nc2` (resp. `ns2`) is the dimension of the controllable (resp. stabilizable) pair `(A44,B42)` (resp. (`[A44,*;0,A55],[B42;0]`)). In case `flag='st'` one has `dims=[dimR,dimVg,dimVg+nc,dimVg+ns]` and in case `flag='pp'` one has `dims=[dimR,dimR+nc,dimR+ns]`. `nc` (resp. `ns`) is here the dimension of the controllable (resp. stabilizable) subspace of the blocks 3 to 6 (resp. 2 to 6).

This function can be used for the (exact) disturbance decoupling problem.

```
DDPS:
   Find u=Fx+Rd=[F,R]*[x;d] which rejects Q*d and stabilizes the plant:

    xdot = Ax+Bu+Qd
       y = Cx+Du+Td

DDPS has a solution if Im(Q) is included in Vg + Im(B) and stabilizability
assumption is satisfied.
Let G=(X(:,dimVg+1:$))'= left annihilator of Vg i.e. G*Vg=0;
B2=G*B; Q2=G*Q; DDPS solvable iff [B2;D]*R + [Q2;T] =0 has a solution.
The pair F,R  is the solution  (with F=output of abinv).
Im(Q2) is in Im(B2) means row-compression of B2=>row-compression of Q2
Then C*[(sI-A-B*F)^(-1)+D]*(Q+B*R) =0   (<=>G*(Q+B*R)=0)
```

## Examples

```
nu=3;ny=4;nx=7;
nrt=2;ngt=3;ng0=3;nvt=5;rk=2;
flag=list('on',nrt,ngt,ng0,nvt,rk);
Sys=ssrand(ny,nu,nx,flag);my_alpha=-1;my_beta=-2;
[X,dims,F,U,k,Z]=abinv(Sys,my_alpha,my_beta);
[A,B,C,D]=abcd(Sys);dimV=dims(3);dimR=dims(1);
V=X(:,1:dimV);X2=X(:,dimV+1:nx);
X2'*(A+B*F)*V
(C+D*F)*V
X0=X(:,dimR+1:dimV); spec(X0'*(A+B*F)*X0)
trzeros(Sys)
spec(A+B*F)   //nr=2 evals at -1 and noc-dimV=2 evals at -2.
clean(ss2tf(Sys*Z))

// 2nd Example
nx=6;ny=3;nu=2;
A=diag(1:6);A(2,2)=-7;A(5,5)=-9;B=[1,2;0,3;0,4;0,5;0,0;0,0];
C=[zeros(ny,ny),eye(ny,ny)];D=[0,1;0,2;0,3];
sl=syslin('c',A,B,C,D);//sl=ss2ss(sl,rand(6,6))*rand(2,2);
[A,B,C,D]=abcd(sl);  //The matrices of sl.
my_alpha=-1;my_beta=-2;
[X,dims,F,U,k,Z]=abinv(sl,my_alpha,my_beta);dimVg=dims(2);
clean(X'*(A+B*F)*X)
clean(X'*B*U)
clean((C+D*F)*X)
clean(D*U)
G=(X(:,dimVg+1:$))';
B2=G*B;nd=3;
R=rand(nu,nd);Q2T=-[B2;D]*R;
p=size(G,1);Q2=Q2T(1:p,:);T=Q2T(p+1:$,:);
Q=G\Q2;   //a valid [Q;T] since
[G*B;D]*R + [G*Q;T]  // is zero
closed=syslin('c',A+B*F,Q+B*R,C+D*F,T+D*R); // closed loop: d-->y
ss2tf(closed)       // Closed loop is zero
spec(closed('A'))   //The plant is not stabilizable!
[ns,nc,W,sl1]=st_ility(sl);
[A,B,C,D]=abcd(sl1);A=A(1:ns,1:ns);B=B(1:ns,:);C=C(:,1:ns);
slnew=syslin('c',A,B,C,D);  //Now stabilizable
//Fnew=stabil(slnew('A'),slnew('B'),-11);
//slnew('A')=slnew('A')+slnew('B')*Fnew;
//slnew('C')=slnew('C')+slnew('D')*Fnew;
[X,dims,F,U,k,Z]=abinv(slnew,my_alpha,my_beta);dimVg=dims(2);
[A,B,C,D]=abcd(slnew);
G=(X(:,dimVg+1:$))';
B2=G*B;nd=3;
R=rand(nu,nd);Q2T=-[B2;D]*R;
p=size(G,1);Q2=Q2T(1:p,:);T=Q2T(p+1:$,:);
Q=G\Q2;   //a valid [Q;T] since
[G*B;D]*R + [G*Q;T]  // is zero
closed=syslin('c',A+B*F,Q+B*R,C+D*F,T+D*R); // closed loop: d-->y
ss2tf(closed)       // Closed loop is zero
spec(closed('A'))
```

## See Also

cainv , st_ility , ssrand , ss2ss , ddp

## Authors

F.D.

# Name

arhnk — Hankel norm approximant

```
[slm]=arhnk(sl,ord,[tol])
```

# Parameters

sl
: linear system (`syslin` list)

ord
: integer, order of the approximant

tol
: threshold for rank determination in `equil1`

# Description

computes `slm`, the optimal Hankel norm approximant of the stable continuous-time linear system `sl` with matrices `[A,B,C,D]`.

# Examples

```
A=diag([-1,-2,-3,-4,-5]);B=rand(5,1);C=rand(1,5);
sl=syslin('c',A,B,C);
slapprox=arhnk(sl,2);
[nk,W]=hankelsv(sl);nk
[nkred,Wred]=hankelsv(slapprox);nkred
```

# See Also

equil , equil1 , hankelsv

# Name

arl2 — SISO model realization by L2 transfer approximation

```
h=arl2(y,den0,n [,imp])
h=arl2(y,den0,n [,imp],'all')
[den,num,err]=arl2(y,den0,n [,imp])
[den,num,err]=arl2(y,den0,n [,imp],'all')
```

# Parameters

y
: real vector or polynomial in `z^-1`, it contains the coefficients of the Fourier's series of the rational system to approximate (the impulse response)

den0
: a polynomial which gives an initial guess of the solution, it may be `poly(1,'z','c')`

n
: integer, the degree of approximating transfer function (degree of den)

imp
: integer in `(0,1,2)` (verbose mode)

h
: transfer function `num/den` or transfer matrix (column vector) when flag `'all'` is given.

den
: polynomial or vector of polynomials, contains the denominator(s) of the solution(s)

num
: polynomial or vector of polynomials, contains the numerator(s) of the solution(s)

err
: real constant or vector , the l2-error achieved for each solutions

# Description

`[den,num,err]=arl2(y,den0,n [,imp])` finds a pair of polynomials `num` and `den` such that the transfer function `num/den` is stable and its impulse response approximates (with a minimal l2 norm) the vector `y` assumed to be completed by an infinite number of zeros.

If `y(z) = y(1)(1/z)+y(2)(1/z^2)+ ...+ y(ny)(1/z^ny)`

then l2-norm of `num/den - y(z)` is `err`.

`n` is the degree of the polynomial `den`.

The `num/den` transfer function is a L2 approximant of the Fourier's series of the rational system.

Various intermediate results are printed according to `imp`.

`[den,num,err]=arl2(y,den0,n [,imp],'all')` returns in the vectors of polynomials `num` and `den` a set of local optimums for the problem. The solutions are sorted with increasing errors `err`. In this case `den0` is already assumed to be `poly(1,'z','c')`

# Examples

```
v=ones(1,20);
clf();
plot2d1('enn',0,[v';zeros(80,1)],2,'051',' ',[1,-0.5,100,1.5])

[d,n,e]=arl2(v,poly(1,'z','c'),1)
plot2d1('enn',0,ldiv(n,d,100),2,'000')
[d,n,e]=arl2(v,d,3)
plot2d1('enn',0,ldiv(n,d,100),3,'000')
[d,n,e]=arl2(v,d,8)
plot2d1('enn',0,ldiv(n,d,100),5,'000')

[d,n,e]=arl2(v,poly(1,'z','c'),4,'all')
plot2d1('enn',0,ldiv(n(1),d(1),100),10,'000')
```

# See Also

ldiv , imrep2ss , time_id , armax , frep2tf

# Name

arma — Scilab arma library

# Description

Armax processes can be coded with Scilab tlist of type `'ar'`. `armac` is used to build `Armax` scilab object. An `'ar'` tlist contains the fields `['a','b','d','ny','nu','sig']`.

armac
: this function creates a Scilab tlist which code an Armax process `A(z^-1)y= B(z^-1)u + D(z^-1)sig*e(t)`

```
-->ar=armac([1,2],[3,4],1,1,1,sig);

-->ar('a')
 ans  =

!   1.    2. !
-->ar('sig')
 ans  =

    1.
```

armap(ar [,out])
: Display the armax equation associated with `ar`

armap_p(ar [,out])
: Display the armax equation associated with `ar` using polynomial matrix display.

[A,B,D]=armap2p(ar)
: extract polynomial matrices from ar representation

armax
: is used to identify the coefficients of a n-dimensional ARX process `A(z^-1)y= B(z^-1)u + sig*e(t)`

armax1
: armax1 is used to identify the coefficients of a 1-dimensional ARX process `A(z^-1)y= B(z^-1)u + D(z^-1)sig*e(t)`

arsimul
: armax trajectory simulation.

narsimul
: armax simulation ( using rtitr)

odedi
: Simple tests of ode and arsimul. Tests the option 'discret' of ode

prbs_a
: pseudo random binary sequences generation

reglin
: Linear regression

# Example

```
// Example extracted from the demo arma3.dem.sce in the cacsd module
// Spectral power estimation
// ( form Sawaragi et all)
m = 18;
a = [1,-1.3136,1.4401,-1.0919,+0.83527];
b = [0.0,0.13137,0.023543,0.10775,0.03516];
u = rand(1,1000,'n');
z = arsimul(a,b,[0],0,u);

//----Using macro mese
[sm,fr]=mese(z,m);

//----The theorical result

function gx=gxx(z,a,b)
  w  = exp(-%i*2*%pi*z*(0:4))'
  gx = abs(b*w)^2/(abs(a*w)^2);
endfunction

res=[];
for x=fr
  res=[ res, gxx(x,a,b)];
end

//----using armax estimation of order  (4,4)
// it's a bit tricky because we are not supposed to know the order

[arc,la,lb,sig,resid]=armax(4,4,z,u);
res1=[];
for x=fr
  res1=[ res1, gxx(x,la(1),lb(1))];
end

//-- visualization of the results
plot2d([fr;fr;fr]',[20*log10(sm/sm(1));20*log10(res/res(1));20*log10(res1/res1(
legend(["Using macro mese";"Theoritical value";"Arma identifcation"])
xtitle("Spectral power","frequency","spectral estimate")
```

# Authors

J.P.C ; ;

# Name

arma2p — extract polynomial matrices from ar representation

```
[A,B,D]=arma2p(ar)
```

# Parameters

A,B,D
: three polynomial matrices

ar
: Scilab 'ar' tlist for arma storage (see `armac`).

# Description

this function extract polynomial matrices `(A,B,D)` from an armax description.

# Examples

```
a=[1,-2.851,2.717,-0.865].*.eye(2,2)
b=[0,1,1,1].*.[1;1];
d=[1,0.7,0.2].*.eye(2,2);
sig=eye(2,2);
ar=armac(a,b,d,2,1,sig)
// extract polynomial matrices from ar representation
[A,B,D]=arma2p(ar);
```

# See Also

arma , armax , armax1 , arsimul , armac

# Name

armac — Scilab description of an armax process

```
[ar]=armac(a,b,d,ny,nu,sig)
```

# Parameters

a=[Id,a1,..,a_r]
: is a matrix of size (ny,r*ny)

b=[b0,.....,b_s]
: is a matrix of size (ny,(s+1)*nu)

d=[Id,d1,..,d_p]
: is a matrix of size (ny,p*ny);

ny
: dimension of the output y

nu
: dimension of the output u

sig
: a matrix of size (ny,ny)

# Description

This function creates a description as a tlist of an ARMAX process

`ar` is defined by

```
ar=tlist(['ar','a','b','d','ny','nu','sig'],a,b,d,ny,nu,sig);
```

and thus the coefficients of `ar` can be retrieved by e.g. `ar('a')` .

# Examples

```
a=[1,-2.851,2.717,-0.865].*.eye(2,2)
b=[0,1,1,1].*.[1;1];
d=[1,0.7,0.2].*.eye(2,2);
sig=eye(2,2);
ar=armac(a,b,d,2,1,sig)
// extract polynomial matrices from ar representation
[A,B,D]=arma2p(ar);
```

# See Also

arma , armax , armax1 , arsimul , arma2p , tlist

# Name

armax — armax identification

```
[arc,la,lb,sig,resid]=armax(r,s,y,u,[b0f,prf])
```

# Parameters

y
: output process y(ny,n); ( ny: dimension of y , n : sample size)

u
: input process u(nu,n); ( nu: dimension of u , n : sample size)

r and s
: auto-regression orders r >=0 et s >=-1

b0f
: optional parameter. Its default value is 0 and it means that the coefficient b0 must be identified. if bof=1 the b0 is supposed to be zero and is not identified

prf
: optional parameter for display control. If prf =1, the default value, a display of the identified Arma is given.

arc
: a Scilab arma object (see armac)

la
: is the list(a,a+eta,a-eta) ( la = a in dimension 1) ; where eta is the estimated standard deviation. , a=[Id,a1,a2,...,ar] where each ai is a matrix of size (ny,ny)

lb
: is the list(b,b+etb,b-etb) (lb =b in dimension 1) ; where etb is the estimated standard deviation. b=[b0,.....,b_s] where each bi is a matrix of size (nu,nu)

sig
: is the estimated standard deviation of the noise and resid=[ sig*e(t0),....] (

# Description

armax is used to identify the coefficients of a n-dimensional ARX process

```
A(z^-1)y= B(z^-1)u + sig*e(t)
```

where e(t) is a n-dimensional white noise with variance I. sig an nxn matrix and A(z) and B(z):

```
A(z) = 1+a1*z+...+a_r*z^r; ( r=0 => A(z)=1)
B(z) = b0+b1*z+...+b_s z^s ( s=-1 => B(z)=0)
```

for the method see Eykhoff in trends and progress in system identification, page 96. with `z(t)=[y(t-1),..,y(t-r),u(t),...,u(t-s)]` and `coef= [-a1,..,-ar,b0,...,b_s]` we can write `y(t)= coef* z(t) + sig*e(t)` and the algorithm minimises `sum_{t=1}^N ( [y(t)- coef'z(t)]^2)` where t0=maxi(maxi(r,s)+1,1))).

# Examples

```
//-Ex1- Arma model : y(t) = 0.2*u(t-1)+0.01*e(t-1)
ny=1,nu=1,sig=0.01;
Arma=armac(1,[0,0.2],[0,1],ny,nu,sig)  //defining the above arma model
u=rand(1,1000,'normal');     //a random input sequence u
y=arsimul(Arma,u); //simulation of a y output sequence associated with u.
Armaest=armax(0,1,y,u);   //Identified model given u and y.
Acoeff=Armaest('a');    //Coefficients of the polynomial A(x)
Bcoeff=Armaest('b')     //Coefficients of the polynomial B(x)
Dcoeff=Armaest('d');    //Coefficients of the polynomial D(x)
[Ax,Bx,Dx]=arma2p(Armaest)   //Results in polynomial form.

//-Ex2- Arma1: y_t -0.8*y_{t-1} + 0.2*y_{t-2} =  sig*e(t)
ny=1,nu=1;sig=0.001;
// First step: simulation the Arma1 model, for that we define
// Arma2: y_t -0.8*y_{t-1} + 0.2*y_{t-2} = sig*u(t)
// with normal deviates for u(t).
Arma2=armac([1,-0.8,0.2],sig,0,ny,nu,0);
//Definition of the Arma2 arma model (a model with B=sig and without noise!)
u=rand(1,10000,'normal');  // An input sequence for Arma2
y=arsimul(Arma2,u); // y = output of Arma2 with input u
//                     can be seen as output of Arma1.
// Second step: identification. We look for an Arma model
// y(t) + a1*y(t-1) + a2 *y(t-2) =  sig*e(t)
Arma1est=armax(2,-1,y,[]);
[A,B,D]=arma2p(Arma1est)
```

# See Also

imrep2ss , time_id , arl2 , armax , frep2tf

# Authors

J-Ph. Chancelier.

# Name

armax1 — armax identification

```
[arc,resid]=armax1(r,s,q,y,u [,b0f])
```

# Parameters

y
: output signal

u
: input signal

r,s,q
: auto regression orders with r >=0, s >=-1.

b0f
: optional parameter. Its default value is 0 and it means that the coefficient b0 must be identified. if bof=1 the b0 is supposed to be zero and is not identified

arc
: is tlist with type "ar" and fields a, b, d, ny, nu, sig

  a
  : is the vector `[1,a1,...,a_r]`

  b
  : is the vector `[b0,.....,b_s]`

  d
  : is the vector `[1,d1,....,d_q]`

  sig
  : resid=[ sig*echap(1),....,];

# Description

armax1 is used to identify the coefficients of a 1-dimensional ARX process:

```
A(z^-1)y= B(z^-1)u + D(z^-1)sig*e(t)
e(t) is a 1-dimensional white noise with variance 1.
A(z)= 1+a1*z+...+a_r*z^r; ( r=0 => A(z)=1)
B(z)= b0+b1*z+...+b_s z^s ( s=-1 => B(z)=0)
D(z)= 1+d1*z+...+d_q*z^q  ( q=0 => D(z)=1)
```

for the method, see Eykhoff in trends and progress in system identification) page 96. with

```
z(t)=[y(t-1),..,y(t-r),u(t),...,
      u(t-s),e(t-1),...,e(t-q)]
```

and

```
coef= [-a1,..,-ar,b0,...,b_s,d1,...,d_q]'
```

```
y(t)= coef'* z(t) + sig*e(t).
```

a sequential version of the AR estimation where e(t-i) is replaced by an estimated value is used (RLLS). With q=0 this method is exactly a sequential version of armax

# Important notice

In Scilab versions up to 4.1.2 the returned value in `arc.sig` is the square of `sig` square. To be conform with the help, the display of arma models and the armax function, starting from Scilab-5.0 version the returned `arc.sig` is `sig`.

# Authors

J.-Ph.C; ;

# Name

arsimul — armax simulation

```
[z]=arsimul(a,b,d,sig,u,[up,yp,ep])
[z]=arsimul(ar,u,[up,yp,ep])
```

# Parameters

ar
: an armax process. See armac.

a
: is the matrix `[Id,a1,...,a_r]` of dimension (n,(r+1)*n)

b
: is the matrix `[b0,.....,b_s]` of dimension (n,(s+1)*m)

d
: is the matrix `[Id,d_1,.....,d_t]` of dimension (n,(t+1)*n)

u
: is a matrix (m,N), which gives the entry u(:,j)=u_j

sig
: is a (n,n) matrix e_{k} is an n-dimensional Gaussian process with variance I

up, yp
: optional parameter which describe the past. `up=[ u_0,u_{-1},...,u_{s-1}]`; `yp=[ y_0,y_{-1},...,y_{r-1}]`; `ep=[ e_0,e_{-1},...,e_{r-1}]`; if they are omitted, the past value are supposed to be zero

z
: `z=[y(1),....,y(N)]`

# Description

simulation of an n-dimensional armax process `A(z^-1) z(k)= B(z^-1)u(k) + D(z^-1)*sig*e(k)`

```
A(z)= Id+a1*z+...+a_r*z^r;  ( r=0  => A(z)=Id)
B(z)= b0+b1*z+...+b_s z^s;  ( s=-1 => B(z)=[])
D(z)= Id+d1*z+...+d_t z^t;  ( t=0  => D(z)=Id)
```

z et e are in `R^n` et u in `R^m`

# Method

a state-space representation is constructed and ode with the option "discr" is used to compute z

# Authors

J-Ph.C.

# Name

augment — augmented plant

```
[P,r]=augment(G)
[P,r]=augment(G,flag1)
[P,r]=augment(G,flag1,flag2)
```

# Parameters

G
: linear system (`syslin` list), the nominal plant

flag1
: one of the following (upper case) character string: `'S'` , `'R'` , `'T'` `'SR'` , `'ST'` , `'RT'` `'SRT'`

flag2
: one of the following character string: `'o'` (stands for 'output', this is the default value) or `'i'` (stands for 'input').

P
: linear system (`syslin` list), the ``augmented'' plant

r
: 1x2 row vector, dimension of `P22 = G`

# Description

If `flag1='SRT'` (default value), returns the "full" augmented plant

```
    [ I | -G]   -->'S'
    [ 0 |  I]   -->'R'
P = [ 0 |  G]   -->'T'
    [-------]
    [ I | -G]
```

`'S'` , `'R'` , `'T'` refer to the first three (block) rows of `P` respectively.

If one of these letters is absent in `flag1`, the corresponding row in `P` is missing.

If `G` is given in state-space form, the returned `P` is minimal. `P` is calculated by: `[I,0,0;0,I,0;-I,0,I;I,0,0]*[I,-G;0,I;I,0]`.

The augmented plant associated with input sensitivity functions, namely

```
    [ I | -I]   -->'S'  (input sensitivity)
    [ G | -G]   -->'R'  (G*input sensitivity)
P = [ 0 |  I]   -->'T'  (K*G*input sensitivity)
    [-------]
    [ G | -G]
```

is obtained by the command `[P,r]=augment(G,flag,'i')`. For state-space `G`, this `P` is calculated by: `[I,-I;0,0;0,I;0,0]+[0;I;0;I]*G*[I,-I]` and is thus generically minimal.

Note that weighting functions can be introduced by left-multiplying `P` by a diagonal system of appropriate dimension, e.g., `P = sysdiag(W1,W2,W3,eye(G))*P`.

Sensitivity functions can be calculated by `lft`. One has:

For output sensitivity functions [P,r]=augment(P,'SRT'): lft(P,r,K)=[inv(eye()+G*K);K*inv(eye()+G*K);G*K*inv(eye()+G*K)];

For input sensitivity functions [P,r]=augment(P,'SRT','i'): lft(P,r,K)=[inv(eye()+K*G);G*inv(eye()+K*G);K*G*inv(eye()+G*K)];

# Examples

```
G=ssrand(2,3,2); //Plant
K=ssrand(3,2,2); //Compensator
[P,r]=augment(G,'T');
T=lft(P,r,K);   //Complementary sensitivity function
Ktf=ss2tf(K);Gtf=ss2tf(G);
Ttf=ss2tf(T);T11=Ttf(1,1);
Oloop=Gtf*Ktf;
Tn=Oloop*inv(eye(Oloop)+Oloop);
clean(T11-Tn(1,1));
//
[Pi,r]=augment(G,'T','i');
T1=lft(Pi,r,K);T1tf=ss2tf(T1); //Input Complementary sensitivity function
Oloop=Ktf*Gtf;
T1n=Oloop*inv(eye(Oloop)+Oloop);
clean(T1tf(1,1)-T1n(1,1))
```

# See Also

lft , sensi

# Name

balreal — balanced realization

```
[slb [,U] ] = balreal(sl)
```

# Parameters

sl,slb
: linear systems (`syslin` lists)

# Description

Balanced realization of linear system `sl=[A,B,C,D]`. `sl` can be a continuous-time or discrete-time state-space system. sl is assumed stable.

```
slb=[inv(U)*A*U ,inv(U)*B , C*U , D]
```

is the balanced realization.

`slb` is returned as a `syslin` list.

# Examples

```
A=diag([-1,-2,-3,-4,-5]);B=rand(5,2);C=rand(1,5);
sl=syslin('c',A,B,C);
[slb,U]=balreal(sl);
Wc=clean(ctr_gram(slb))
W0=clean(obs_gram(slb))
```

# See Also

ctr_gram , obs_gram , hankelsv , equil , equil1

# Name

bilin — general bilinear transform

```
[sl1]=bilin(sl,v)
```

# Parameters

sl,sl1
: linear systems (`syslin` lists)

v
: real vector with 4 entries (`v=[a,b,c,d]`)

# Description

Given a linear system in state space form, `sl=syslin(dom,A,B,C,D)` (`syslin` list), `sl1=bilin(sl,v)` returns in `sl1` a linear system with matrices `[A1,B1,C1,D1]` such that the transfer function `H1(s)=C1*inv(s*eye()-A1)*B1+D1` is obtained from `H(z)=C*inv(z*eye()-A)*B+D` by replacing `z` by `z=(a*s+b)/(c*s+d)`. One has `w=bilin(bilin(w,[a,b,c,d]),[d,-b,-c,a])`

# Examples

```
s=poly(0,'s');z=poly(0,'z');
w=ssrand(1,1,3);
wtf=ss2tf(w);v=[2,3,-1,4];a=v(1);b=v(2);c=v(3);d=v(4);
[horner(wtf,(a*z+b)/(c*z+d)),ss2tf(bilin(w,[a,b,c,d]))]
clean(ss2tf(bilin(bilin(w,[a,b,c,d]),[d,-b,-c,a]))-wtf)
```

# See Also

horner , cls2dls

# Name

bstap — hankel approximant

```
[Q]=bstap(Sl)
```

# Parameters

sl
: linear system (`syslin` list) assumed continuous-time and anti-stable.

Q
: best stable approximation of `Sl` (`syslin` list).

# Description

Computes the best approximant `Q` of the linear system `Sl`

where

||T||

is the H-infinity norm of the Hankel operator associated with `Sl`.

# See Also

syslin

# Name

cainv — Dual of abinv

```
[X,dims,J,Y,k,Z]=cainv(Sl,alfa,beta,flag)
```

# Parameters

Sl
: `syslin` list containing the matrices `[A,B,C,D]`.

alfa
: real number or vector (possibly complex, location of closed loop poles)

beta
: real number or vector (possibly complex, location of closed loop poles)

flag
: (optional) character string `'ge'` (default) or `'st'` or `'pp'`

X
: orthogonal matrix of size nx (dim of state space).

dims
: integer row vector `dims=[nd1,nu1,dimS,dimSg,dimN]` (5 entries, nondecreasing order).If `flag='st'`, (resp. `'pp'`), `dims` has 4 (resp. 3) components.

J
: real matrix (output injection)

Y
: orthogonal matrix of size ny (dim of output space).

k
: integer (normal rank of `Sl`)

Z
: non-singular linear system (`syslin` list)

# Description

`cainv` finds a bases `(X,Y)` (of state space and output space resp.) and output injection matrix `J` such that the matrices of Sl in bases (X,Y) are displayed as:

```
                [A11,*,*,*,*,*]                 [*]
                [0,A22,*,*,*,*]                 [*]
X'*(A+J*C)*X =  [0,0,A33,*,*,*]   X'*(B+J*D) =  [*]
                [0,0,0,A44,*,*]                 [0]
                [0,0,0,0,A55,*]                 [0]
                [0,0,0,0,0,A66]                 [0]

       Y*C*X =  [0,0,C13,*,*,*]          Y*D =  [*]
                [0,0,0,0,0,C26]                 [0]
```

The partition of `X` is defined by the vector `dims=[nd1,nu1,dimS,dimSg,dimN]` and the partition of `Y` is determined by `k`.

Eigenvalues of `A11` `(nd1 x nd1)` are unstable. Eigenvalues of `A22` `(nu1-nd1 x nu1-nd1)` are stable.

The pair `(A33, C13)` `(dimS-nu1 x dimS-nu1, k x dimS-nu1)` is observable, and eigenvalues of `A33` are set to `alfa`.

Matrix `A44` `(dimSg-dimS x dimSg-dimS)` is unstable. Matrix `A55` `(dimN-dimSg,dimN-dimSg)` is stable

The pair `(A66,C26)` `(nx-dimN x nx-dimN)` is observable, and eigenvalues of `A66` set to `beta`.

The `dimS` first columns of `X` span S= smallest (C,A) invariant subspace which contains Im(B), `dimSg` first columns of `X` span Sg the maximal "complementary detectability subspace" of `Sl`

The `dimN` first columns of `X` span the maximal "complementary observability subspace" of `Sl`. (`dimS=0` if B(ker(D))=0).

If `flag='st'` is given, a five blocks partition of the matrices is returned and `dims` has four components. If `flag='pp'` is given a four blocks partition is returned (see abinv).

This function can be used to calculate an unknown input observer:

```
// DDEP: dot(x)=A x + Bu + Gd
//           y= Cx   (observation)
//           z= Hx    (z=variable to be estimated, d=disturbance)
//  Find: dot(w) = Fw + Ey + Ru such that
//          zhat = Mw + Ny
//           z-Hx goes to zero at infinity
//  Solution exists iff Ker H contains Sg(A,C,G) inter KerC (assuming detectabi
//i.e. H is such that:
// For any W which makes a column compression of [Xp(1:dimSg,:);C]
// with Xp=X' and [X,dims,J,Y,k,Z]=cainv(syslin('c',A,G,C));
// [Xp(1:dimSg,:);C]*W = [0 | *] one has
// H*W = [0 | *]  (with at least as many aero columns as above).
```

# See Also

abinv , dt_ility , ui_observer

# Name

calfrq — frequency response discretization

```
[frq,bnds,split]=calfrq(h,fmin,fmax)
```

# Parameters

h
: Linear system in state space or transfer representation (`see syslin`)

fmin,fmax
: real scalars (min and max frequencies in Hz)

frq
: row vector (discretization of the frequency interval)

bnds
: vector `[Rmin Rmax Imin Imax]` where `Rmin` and `Rmax` are the lower and upper bounds of the frequency response real part, `Imin` and `Imax` are the lower and upper bounds of the frequency response imaginary part,

split
: vector of frq splitting points indexes

# Description

frequency response discretization; `frq` is the discretization of `[fmin,fmax]` such that the peaks in the frequency response are well represented.

Singularities are located between `frq(split(k)-1)` and `frq(split(k))` for `k>1`.

# Examples

```
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
[f1,bnds,spl]=calfrq(h1,0.01,1000);
rf=repfreq(h1,f1);
plot2d(real(rf)',imag(rf)')
```

# See Also

bode, black, nyquist, freq, repfreq, logspace

# Name

canon — canonical controllable form

```
[Ac,Bc,U,ind]=canon(A,B)
```

# Parameters

Ac,Bc
: canonical form

U
: current basis (square nonsingular matrix)

ind
: vector of integers, controllability indices

# Description

gives the canonical controllable form of the pair `(A,B)`.

`Ac=inv(U)*A*U, Bc=inv(U)*B`

The vector `ind` is made of the `epsilon_i`'s indices of the pencil `[sI - A , B]` (decreasing order). For example with `ind=[3,2]`, `Ac` and `Bc` are as follows:

```
      [*,*,*,*,*]           [*]
      [1,0,0,0,0]           [0]
Ac=   [0,1,0,0,0]        Bc=[0]
      [*,*,*,*,*]           [*]
      [0,0,0,1,0]           [0]
```

If `(A,B)` is controllable, by an appropriate choice of `F` the `*` entries of `Ac+Bc*F` can be arbitrarily set to desired values (pole placement).

# Examples

```
A=[1,2,3,4,5;
   1,0,0,0,0;
   0,1,0,0,0;
   6,7,8,9,0;
   0,0,0,1,0];
B=[1,2;
   0,0;
   0,0;
   2,1;
   0,0];
X=rand(5,5);A=X*A*inv(X);B=X*B;    //Controllable pair
[Ac,Bc,U,ind]=canon(A,B);  //Two indices --> ind=[3.2];
index=1;for k=1:size(ind,'*')-1,index=[index,1+sum(ind(1:k))];end
Acstar=Ac(index,:);Bcstar=Bc(index,:);
s=poly(0,'s');
p1=s^3+2*s^2-5*s+3;p2=(s-5)*(s-3);
//p1 and p2 are desired closed-loop polynomials with degrees 3,2
c1=coeff(p1);c1=c1($-1:-1:1);c2=coeff(p2);c2=c2($-1:-1:1);
```

```
Acstardesired=[-c1,0,0;0,0,0,-c2];
//Acstardesired(index,:) is companion matrix with char. pol=p1*p2
F=Bcstar\(Acstardesired-Acstar);   //Feedbak gain
Ac+Bc*F         // Companion form
spec(A+B*F/U)   // F/U is the gain matrix in original basis.
```

# See Also

obsv_mat , cont_mat , ctr_gram , contrss , ppol , contr , stabil

# Authors

F.D.

# Name

ccontrg — central H-infinity controller

```
[K]=ccontrg(P,r,gamma);
```

# Parameters

P
: `syslin` list (linear system in state-space representation)

r
: 1x2 row vector, dimension of the 2,2 part of `P`

gamma
: real number

# Description

returns a realization `K` of the central controller for the general standard problem in state-space form.

Note that gamma must be > gopt (ouput of `gamitg`)

P contains the parameters of plant realization `(A,B,C,D)` (`syslin` list) with

```
B = ( B1 , B2 ) ,         C= ( C1 ) ,    D = ( D11  D12)
                             ( C2 )          ( D21  D22)
```

`r(1)` and `r(2)` are the dimensions of `D22` (rows x columns)

# See Also

gamitg , h_inf

# Authors

P. Gahinet (INRIA);

# Name

cls2dls — bilinear transform

```
[sl1]=cls2dls(sl,T [,fp])
```

# Parameters

sl,sl1
: linear systems (`syslin` lists)

T
: real number, the sampling period

fp
: prevarping frequency in hertz

# Description

given `sl=[A,B,C,D]` (`syslin` list),a continuous time system `cls2dls` returns the sampled system obtained by the bilinear transform `s=(2/T)*(z-1)/(z+1)`.

# Examples

```
s=poly(0,'s');z=poly(0,'z');
sl=syslin('c',(s+1)/(s^2-5*s+2));  //Continuous-time system in transfer form
slss=tf2ss(sl);  //Now in state-space form
sl1=cls2dls(slss,0.2);  //sl1= output of cls2dls
sl1t=ss2tf(sl1) // Converts in transfer form
sl2=horner(sl,(2/0.2)*(z-1)/(z+1))   //Compare sl2 and sl1
```

# See Also

horner

# Name

colinout — inner-outer factorization

```
[Inn,X,Gbar]=colinout(G)
```

# Parameters

G
: linear system (`syslin` list) `[A,B,C,D]`

Inn
: inner factor (`syslin` list)

Gbar
: outer factor (`syslin` list)

X
: row-compressor of `G` (`syslin` list)

# Description

Inner-outer factorization (and column compression) of (lxp) `G =[A,B,C,D]` with l<=p.

`G` is assumed to be fat (`l<=p`) without zero on the imaginary axis and with a `D` matrix which is full row rank.

`G` must also be stable for having `Gbar` stable.

Dual of `rowinout`.

# See Also

syslin , rowinout

# Name

colregul — removing poles and zeros at infinity

```
[Stmp,Ws]=colregul(Sl,alfa,beta)
```

# Parameters

Sl,Stmp
: `syslin` lists

alfa,beta
: reals (new pole and zero positions)

# Description

computes a prefilter `Ws` such that `Stmp=Sl*Ws` is proper and with full rank `D` matrix.

Poles at infinity of `Sl` are moved to `alfa`;

Zeros at infinity of `Sl` are moved to `beta`;

`Sl` is a assumed to be a left invertible linear system (`syslin` list) in state-space representation with possibly a polynomial `D` matrix.

# See Also

invsyslin , inv , rowregul , rowshuff

# Authors

F. D. , R. N. ;

# Name

cont_frm — transfer to controllable state-space

```
[sl]=cont_frm(NUM,den)
```

# Parameters

NUM
: polynomial matrix

den
: polynomial

sl
: `syslin` list, `sl=[A,B,C,D]`.

# Description

controllable state-space form of the transfer `NUM/den`.

# Examples

```
s=poly(0,'s');NUM=[1+s,s];den=s^2-5*s+1;
sl=cont_frm(NUM,den);
slss=ss2tf(sl);       //Compare with NUM/den
```

# See Also

tf2ss , canon , contr

# Name

cont_mat — controllability matrix

```
Cc=cont_mat(A,B)
Cc=cont_mat(sl)
```

# Parameters

a,b
: two real matrices of appropriate dimensions

sl
: linear system (`syslin` list)

# Description

`cont_mat` returns the controllability matrix of the pair `A,B` (resp. of the system `sl=[A,B,C,D]`).

```
Cc=[B, AB, A^2 B,..., A^(n-1) B]
```

# See Also

ctr_gram , contr , canon , st_ility

# Name

contr — controllability, controllable subspace, staircase

```
n=contr(A,B [,tol])
[n,U]=contr(A,B [,tol])
[n,U,ind,V,Ac,Bc]=contr(A,B,[,tol])
```

# Parameters

A, B
: real matrices

tol
: tolerance parameter

n
: dimension of controllable subspace.

U
: orthogonal change of basis which puts `(A,B)` in canonical form.

V
: orthogonal matrix, change of basis in the control space.

Ac
: block Hessenberg matrix `Ac=U'*A*U`

Bc
: is `U'*B*V`.

ind
: p integer vector associated with controllability indices (dimensions of subspaces `B, B +A*B,...=ind(1),ind(1)+ind(2),...`)

# Description

`[n,[U]]=contr(A,B,[tol])` gives the controllable form of an `(A,B)` pair.(`dx/dt = A x + B u` or `x(n+1) = A x(n) +b u(n)`). The `n` first columns of `U` make a basis for the controllable subspace.

If `V=U(:,1:n)`, then `V'*A*V` and `V'*B` give the controllable part of the `(A,B)` pair.

The pair `(Bc, Ac)` is in staircase controllable form.

```
                    |B |sI-A      *  . . .  *      *       |
                    | 1|    11       .      .      .       |
                    |  |  A    sI-A    .    .      .       |
                    |  |   21      22    .  .      .       |
                    |  |        .     .     *      *       |
[U'BV|sI - U'AU] = |0 |     0    .     .                  |
                    |  |            A     sI-A     *       |
                    |  |             p,p-1    pp           |
                    |  |                                   |
                    |0 |         0          0   sI-A       |
                    |  |                              p+1,p+1|
```

# Reference

Slicot library (see ab01od in SCIDIR/routines/slicot).

# Examples

```
W=ssrand(2,3,5,list('co',3));  //cont. subspace has dim 3.
A=W("A");B=W("B");
[n,U]=contr(A,B);n
A1=U'*A*U;
spec(A1(n+1:$,n+1:$))  //uncontrollable modes
spec(A+B*rand(3,5))
```

# See Also

canon , cont_mat , unobs , stabil , st_ility

# Name

contrss — controllable part

```
[slc]=contrss(sl [,tol])
```

# Parameters

sl
: linear system (`syslin` list)

tol
: is a threshold for controllability (see `contr`). default value is `sqrt(%eps)`.

# Description

returns the controllable part of the linear system `sl = (A,B,C,D)` in state-space form.

# Examples

```
A=[1,1;0,2];B=[1;0];C=[1,1];sl=syslin('c',A,B,C);  //Non minimal
slc=contrss(sl);
sl1=ss2tf(sl);sl2=ss2tf(slc);      //Compare sl1 and sl2
```

# See Also

cont_mat , ctr_gram , cont_frm , contr

# Name

copfac — right coprime factorization

```
[N,M,XT,YT]=copfac(G [,polf,polc,tol])
```

# Parameters

G
: `syslin` list (continuous-time linear system )

polf, polc
: respectively the poles of `XT` and `YT` and the poles of `n` and `M` (default values =-1).

tol
: real threshold for detecting stable poles (default value `100*%eps`)

N,M,XT,YT
: linear systems represented by `syslin` lists

# Description

`[N,M,XT,YT]=copfac(G,[polf,polc,[tol]])` returns a right coprime factorization of `G`.

`G = N*M^-1` where `N` and `M` are stable, proper and right coprime. (i.e. `[N M]` left-invertible with stability)

`XT` and `YT` satisfy:

`[XT -YT].[M N]' = eye` (Bezout identity)

`G` is assumed stabilizable and detectable.

# See Also

syslin , lcf

# Name

csim — simulation (time response) of linear system

```
[y [,x]]=csim(u,t,sl,[x0 [,tol]])
```

# Parameters

u
: function, list or string (control)

t
: real vector specifying times with, `t(1)` is the initial time (`x0=x(t(1))`).

sl
: list (`syslin`)

y
: a matrix such that `y=[y(t(i)]`, i=1,..,n

x
: a matrix such that `x=[x(t(i)]`, i=1,..,n

tol
: a 2 vector [atol rtol] defining absolute and relative tolerances for ode solver (see ode)

# Description

simulation of the controlled linear system `sl`. `sl` is assumed to be a continuous-time system represented by a `syslin` list.

`u` is the control and `x0` the initial state.

`y` is the output and `x` the state.

The control can be:

1. a function : `[inputs]=u(t)`

2. a list : `list(ut,parameter1,....,parametern)` such that: `inputs=ut(t,parameter1,....,parametern)` (`ut` is a function)

3. the string `"impuls"` for impulse response calculation (here `sl` is assumed SISO without direct feed through and `x0=0`)

4. the string `"step"` for step response calculation (here `sl` is assumed SISO without direct feed-through and `x0=0`)

5. a vector giving the values of u correponding to each t value.

# Examples

```
s=poly(0,'s');rand('seed',0);w=ssrand(1,1,3);w('A')=w('A')-2*eye();
t=0:0.05:5;
//impulse(w) = step (s * w)
clf(0);xset("window",0);xselect();
plot2d([t',t'],[(csim('step',t,tf2ss(s)*w))',0*t'])
clf(1);xset("window",1);xselect();
```

```
plot2d([t',t'],[(csim('impulse',t,w))',0*t'])
//step(w) = impulse (s^-1 * w)
clf(3);xset("window",3);xselect();
plot2d([t',t'],[(csim('step',t,w))',0*t'])
clf(4);xset("window",4);xselect();
plot2d([t',t'],[(csim('impulse',t,tf2ss(1/s)*w))',0*t'])

//input defined by a time function
deff('u=input(t)','u=abs(sin(t))')
clf();plot2d([t',t'],[(csim(input,t,w))',0*t'])
```

# See Also

syslin , dsimul , flts , ltitr , rtitr , ode , impl

# Name

ctr_gram — controllability gramian

```
[Gc]=ctr_gram(A,B [,dom])
[Gc]=ctr_gram(sl)
```

# Parameters

A,B
: two real matrices of appropriate dimensions

dom
: character string (`'c'` (default value) or `'d'`)

sl
: linear system, `syslin` list

# Description

Controllability gramian of `(A,B)` or `sl` (a `syslin` linear system).

`dom` character string giving the time domain : `"d"` for a discrete time system and `"c"` for continuous time (default case).

# Examples

```
A=diag([-1,-2,-3]);B=rand(3,2);
Wc=ctr_gram(A,B)
U=rand(3,3);A1=U*A/U;B1=U*B;
Wc1=ctr_gram(A1,B1)    //Not invariant!
```

# See Also

equil1 , obs_gram , contr , cont_mat , cont_frm , contrss

# Authors

S. Steer INRIA 1988

# Name

dbphi — frequency response to phase and magnitude representation

```
[db,phi] =dbphi(repf)
```

# Parameters

db,phi
: vector of gains (db) and phases (degrees)

repf
: vector of complex frequency response

# Description

`db(k)` is the magnitude of `repf(k)` expressed in dB i.e. `db(k)=20*log(abs(repf(k)))/log(10)` and `phi(k)` is the phase of `repf(k)` expressed in degrees.

# See Also

repfreq , bode

# Name

dcf — double coprime factorization

```
[N,M,X,Y,NT,MT,XT,YT]=dcf(G,[polf,polc,[tol]])
```

# Parameters

G
: `syslin` list (continuous-time linear system)

polf, polc
: respectively the poles of `XT` and `YT` and the poles of `N` and `M` (default values =`-1`).

tol
: real threshold for detecting stable poles (default value `100*%eps`).

N,M,XT,YT,NT,MT,X,Y
: linear systems represented by `syslin` lists

# Description

returns eight stable systems `(N,M,X,Y,NT,MT,XT,YT)` for the doubly coprime factorization

`G` must be stabilizable and detectable.

# See Also

copfac

# Name

ddp — disturbance decoupling

```
[Closed,F,G]=ddp(Sys,zeroed,B1,D1)
[Closed,F,G]=ddp(Sys,zeroed,B1,D1,flag,alfa,beta)
```

# Parameters

Sys
: `syslin` list containing the matrices `(A,B2,C,D2)`.

zeroed
: integer vector, indices of outputs of `Sys` which are zeroed.

B1
: real matrix

D1
: real matrix. `B1` and `D1` have the same number of columns.

flag
: string `'ge'` or `'st'` (default) or `'pp'`.

alpha
: real or complex vector (loc. of closed loop poles)

beta
: real or complex vector (loc. of closed loop poles)

# Description

Exact disturbance decoupling (output nulling algorithm). Given a linear system, and a subset of outputs, z, which are to be zeroed, characterize the inputs w of Sys such that the transfer function from w to z is zero. `Sys` is a linear system {A,B2,C,D2} with one input and two outputs ( i.e. Sys: u-->(z,y) ), part the following system defined from `Sys` and `B1,D1`:

```
xdot =  A x + B1  w + B2  u
   z = C1 x + D11 w + D12 u
   y = C2 x + D21 w + D22 u
```

outputs of Sys are partitioned into (z,y) where z is to be zeroed, i.e. the matrices C and D2 are:

```
C=[C1;C2]          D2=[D12;D22]
C1=C(zeroed,:)     D12=D2(zeroed,:)
```

The matrix `D1` is partitioned similarly as `D1=[D11;D21]` with `D11=D1(zeroed,:)`. The control is u=Fx+Gw and one looks for matriced `F,G` such that the closed loop system: w-->z given by

```
xdot= (A+B2*F)  x + (B1 + B2*G) w
  z = (C1+D12F) x + (D11+D12*G) w
```

has zero transfer transfer function.

`flag='ge'` no stability constraints. `flag='st'` : look for stable closed loop system (A+B2*F stable). `flag='pp'` : eigenvalues of A+B2*F are assigned to `alfa` and `beta`.

Closed is a realization of the `w-->y` closed loop system

```
xdot= (A+B2*F)  x + (B1 + B2*G) w
  y = (C2+D22*F) x + (D21+D22*G) w
```

Stability (resp. pole placement) requires stabilizability (resp. controllability) of (A,B2).

# Examples

```
rand('seed',0);nx=6;nz=3;nu=2;ny=1;
A=diag(1:6);A(2,2)=-7;A(5,5)=-9;B2=[1,2;0,3;0,4;0,5;0,0;0,0];
C1=[zeros(nz,nz),eye(nz,nz)];D12=[0,1;0,2;0,3];
Sys12=syslin('c',A,B2,C1,D12);
C=[C1;rand(ny,nx)];D2=[D12;rand(ny,size(D12,2))];
Sys=syslin('c',A,B2,C,D2);
[A,B2,C1,D12]=abcd(Sys12);  //The matrices of Sys12.
my_alpha=-1;my_beta=-2;flag='ge';
[X,dims,F,U,k,Z]=abinv(Sys12,my_alpha,my_beta,flag);
clean(X'*(A+B2*F)*X)
clean(X'*B2*U)
clean((C1+D12*F)*X)
clean(D12*U);
//Calculating an ad-hoc B1,D1
G1=rand(size(B2,2),3);
B1=-B2*G1;
D11=-D12*G1;
D1=[D11;rand(ny,size(B1,2))];

[Closed,F,G]=ddp(Sys,1:nz,B1,D1,'st',my_alpha,my_beta);
closed=syslin('c',A+B2*F,B1+B2*G,C1+D12*F,D11+D12*G);
ss2tf(closed)
```

# See Also

abinv , ui_observer

# Authors

F.D.

# Name

des2ss — descriptor to state-space

```
[Sl]=des2ss(A,B,C,D,E [,tol])
[Sl]=des2ss(Des)
```

# Parameters

A,B,C,D,E
: real matrices of appropriate dimensions

Des
: list

Sl
: `syslin` list

tol
: real parameter (threshold) (default value `100*%eps`).

# Description

Descriptor to state-space transform.

`Sl=des2ss(A,B,C,D,E)` returns a linear system `Sl` equivalent to the descriptor system `(E,A,B,C,D)`.

For index one `(E,A)` pencil, explicit formula is used and for higher index pencils `rowshuff` is used.

`Sl=des2ss(Des)` with `Des=list('des',A,B,C,D,E)` returns a linear system `Sl` in state-space form with possibly a polynomial `D` matrix.

A generalized Leverrier algorithm is used.

# Examples

```
s=poly(0,'s');G=[1/(s-1),s;1,2/s^3];
S1=tf2des(G);S2=tf2des(G,"withD");
W1=des2ss(S1);W2=des2ss(S2);
clean(ss2tf(W1))
clean(ss2tf(W2))
```

# See Also

des2tf , glever , rowshuff

# Name

des2tf — descriptor to transfer function conversion

```
[S]=des2tf(sl)
[Bfs,Bis,chis]=des2tf(sl)
```

# Parameters

sl
: list (linear system in descriptor form)

Bfs, Bis
: two polynomial matrices

chis
: polynomial

S
: rational matrix

# Description

Given the linear system in descriptor form i.e. `Sl=list('des',A,B,C,D,E)`, `des2tf` converts `sl` into its transfer function representation:

```
S=C*(s*E-A)^(-1)*B+D
```

Called with 3 outputs arguments `des2tf` returns `Bfs` and `Bis` two polynomial matrices, and `chis` polynomial such that:

```
S=Bfs/chis - Bis
```

`chis` is the determinant of `(s*E-A)` (up to a xcative constant);

# Examples

```
s=poly(0,'s');
G=[1/(s+1),s;1+s^2,3*s^3];
Descrip=tf2des(G);Tf1=des2tf(Descrip)
Descrip2=tf2des(G,"withD");Tf2=des2tf(Descrip2)
[A,B,C,D,E]=Descrip2(2:6);Tf3=C*inv(s*E-A)*B+D
```

# See Also

glever , pol2des , tf2des , ss2tf , des2ss , rowshuff

# Authors

F. D.

# Name

dhinf — H_infinity design of discrete-time systems

```
[AK,BK,CK,DK,(RCOND)] = dishin(A,B,C,D,ncon,nmeas,gamma)
```

# Parameters

A
: the n-by-n system state matrix A.

B
: the n-by-m system input matrix B.

C
: the p-by-n system output matrix C.

D
: the p-by-m system matrix D.

ncon
: the number of control inputs. m >= ncon >= 0, p-nmeas >= ncon.

nmeas
: the number of measurements. p >= nmeas >= 0, m-ncon >= nmeas.

gamma
: the parameter gamma used in `H_infinity` design. It is assumed that gamma is sufficiently large so that the controller is admissible. gamma >= 0.

AK
: the n-by-n controller state matrix AK.

BK
: the n-by-nmeas controller input matrix BK.

CK
: the ncon-by-n controller output matrix CK.

DK
: the ncon-by-nmeas controller matrix DK.

RCOND
: a vector containing estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller. (See the description of the algorithm in [1].)

  RCOND
  : (1) contains the reciprocal condition number of the matrix R3,

  RCOND
  : (2) contains the reciprocal condition number of the matrix R1 - R2'*inv(R3)*R2

  RCOND
  : (3) contains the reciprocal condition number of the matrix V21,

  RCOND
  : (4) contains the reciprocal condition number of the matrix St3,

  RCOND
  : (5) contains the reciprocal condition number of the matrix V12,

RCOND
(6) contains the reciprocal condition number of the matrix Im2 + DKHAT*D22,

RCOND
(7) contains the reciprocal condition number of the X-Riccati equation,

RCOND
(8) contains the reciprocal condition number of the Z-Riccati equation.

# Description

`[AK,BK,CK,DK,(RCOND)] = dhinf(A,B,C,D,ncon,nmeas, gamma)` To compute the matrices of an H-infinity (sub)optimal n-state controller

```
    | AK | BK |
K = |----|----|,
    | CK | DK |
```

for the discrete-time system

```
    | A  | B1  B2  |   | A | B |
P = |----|---------| = |---|---|,
    | C1 | D11 D12 |   | C | D |
    | C2 | D21 D22 |
```

and for a given value of gamma, where B2 has column size of the number of control inputs (ncon) and C2 has row size of the number of measurements (nmeas) being provided to the controller.

# References

[1] P.Hr. Petkov, D.W. Gu and M.M. Konstantinov. Fortran 77 routines for Hinf and H2 design of linear discrete-time control systems. Report99-8, Department of Engineering, Leicester University, April 1999.

# Examples

```
//example from Niconet report SLWN1999-12
//Hinf
A=[-0.7  0    0.3  0   -0.5 -0.1
   -0.6  0.2 -0.4 -0.3  0    0
   -0.5  0.7 -0.1  0    0   -0.8
   -0.7  0    0   -0.5 -1    0
    0    0.3  0.6 -0.9  0.1 -0.4
    0.5 -0.8  0    0    0.2 -0.9];
B=[-1 -2 -2  1  0
    1  0  1 -2  1
   -3 -4  0  2 -2
    1 -2  1  0 -1
    0  1 -2  0  3
    1  0  3 -1 -2];
C=[ 1 -1  2 -2  0 -3
   -3  0  1 -1  1  0
    0  2  0 -4  0 -2
```

```
    1 -3  0  0  3  1
    0  1 -2  1  0 -2];
D=[1 -1 -2  0  0
   0  1  0  1  0
   2 -1 -3  0  1
   0  1  0  1 -1
   0  0  1  2  1];

ncon=2
nmeas=2
gam=111.30;
[AK,BK,CK,DK] = dhinf(A,B,C,D,ncon,nmeas,gam)
```

# See Also

hinf , h_inf

# Name

dhnorm — discrete H-infinity norm

```
hinfnorm=dhnorm(sl,[tol],[normax])
```

# Parameters

sl
: the state space system (`syslin` list) (discrete-time)

tol
: tolerance in bisection step, default value `0.01`

normax
: upper bound for the norm , default value is `1000`

hinfnorm
: the discrete infinity norm of `Sl`

# Description

produces the discrete-time infinity norm of a state-space system (the maximum over all frequencies on the unit circle of the maximum singular value).

# See Also

h_norm , linfn

# Name

dscr — discretization of linear system

```
[sld [,r]]=dscr(sl,dt [,m])
```

# Parameters

sl
: `syslin` list containing `[A,B,C,D]`.

dt
: real number, sampling period

m
: covariance of the input noise (continuous time)(default value=0)

r
: covariance of the output noise (discrete time) given if `m` is given as input

sld
: sampled (discrete-time) linear system, `syslin` list

# Description

Discretization of linear system. `sl` is a continuous-time system:

`dx/dt=A*x+B*u (+ noise)`.

`sld` is the discrete-time system obtained by sampling `sl` with the sampling period `dt`.

# Examples

```
s=poly(0,'s');
Sys=syslin('c',[1,1/(s+1);2*s/(s^2+2),1/s])
ss2tf(dscr(tf2ss(Sys),0.1))
```

# See Also

syslin , flts , dsimul

# Name

dsimul — state space discrete time simulation

```
y=dsimul(sl,u)
```

# Parameters

sl
: `syslin` list describing a discrete time linear system

u
: real matrix of appropriate dimension

y
: output of `sl`

# Description

Utility function. If `[A,B,C,D]=abcd(sl)` and `x0=sl('X0')`, `dsimul` returns `y=C*ltitr(A,B,u,x0)+D*u` i.e. the time response of `sl` to the input `u`. `sl` is assumed to be in state space form (`syslin` list).

# Examples

```
z=poly(0,'z');
h=(1-2*z)/(z^2-0.2*z+1);
sl=tf2ss(h);
u=zeros(1,20);u(1)=1;
x1=dsimul(sl,u)   //Impulse response
u=ones(1,20);
x2=dsimul(sl,u);  //Step response
```

# See Also

syslin , flts , ltitr

# Name

dt_ility — detectability test

```
[k, [n [,U [,Sld ] ] ]]=dt_ility(Sl [,tol])
```

# Parameters

Sl
: linear system (`syslin` list)

n
: dimension of unobservable subspace

k
: dimension of unstable, unobservable subspace ( `k<=n`).

U
: orthogonal matrix

Sld
: linear system (`syslin` list)

tol
: threshold for controllability test.

# Description

Detectability test for `sl`, a linear system in state-space representation. `U` is a basis whose `k` first columns span the unstable, unobservable subspace of `Sl` (intersection of unobservable subspace of `(A,C)` and unstable subspace of `A`). Detectability means `k=0`.

`Sld = (U'*A*U,U'*B,C*U,D)` displays the "detectable part" of `Sl=(A,B,C,D)`, i.e.

```
          [*,*,*]
U'*A*U = [0,*,*]
          [0,0,*]

C*U = [0,0,*]
```

with `(A33,C3)` observable (dimension `nx-n`), `A22` stable (dimension `n-k`) and `A11` unstable (dimension `k`).

# Examples

```
A=[2,1,1;0,-2,1;0,0,3];
C=[0,0,1];
X=rand(3,3);A=inv(X)*A*X;C=C*X;
W=syslin('c',A,[],C);
[k,n,U,W1]=dt_ility(W);
W1("A")
W1("C")
```

# See Also

contr , st_ility , unobs , stabil

# Name

dtsi — stable anti-stable decomposition

```
[Ga,Gs,Gi]=dtsi(G,[tol])
```

# Parameters

G
: linear system (`syslin` list)

Ga
: linear system (`syslin` list) antistable and strictly proper

Gs
: linear system (`syslin` list) stable and strictly proper

Gi
: real matrix (or polynomial matrix for improper systems)

tol
: optional parameter for detecting stables poles. Default value: `100*%eps`

# Description

returns the stable-antistable decomposition of `G`:

`G = Ga + Gs + Gi`, `(Gi = G(oo))`

`G` can be given in state-space form or in transfer form.

# See Also

syslin , pbig , psmall , pfss

# Name

equil — balancing of pair of symmetric matrices

```
T=equil(P,Q)
```

# Parameters

P, Q
: two positive definite symmetric matrices

T
: nonsingular matrix

# Description

`equil` returns `t` such that:

`T*P*T'` and `inv(T)'*Q*inv(T)` are both equal to a same diagonal and positive matrix.

# Examples

```
P=rand(4,4);P=P*P';
Q=rand(4,4);Q=Q*Q';
T=equil(P,Q)
clean(T*P*T')
clean(inv(T)'*Q*inv(T))
```

# See Also

equil1 , balanc , ctr_gram

# Name

equil1 — balancing (nonnegative) pair of matrices

```
[T [,siz]]=equil1(P,Q [,tol])
```

# Parameters

P, Q
: two non-negative symmetric matrices

T
: nonsingular matrix

siz
: vector of three integers

tol
: threshold

# Description

`equil1` computes `t` such that:

`P1=T*P*T'` and `Q1=inv(T)'*Q*inv(T)` are as follows:

`P1 = diag(S1,S2,0,0)` and `Q1 = diag(S1,0,S3,0)` with `S1,S2,S3` positive and diagonal matrices with respective dimensions `siz=[n1,n2,n3]`

`tol` is a threshold for rank determination in SVD

# Examples

```
S1=rand(2,2);S1=S1*S1';
S2=rand(2,2);S2=S2*S2';
S3=rand(2,2);S3=S3*S3';
P=sysdiag(S1,S2,zeros(4,4));
Q=sysdiag(S1,zeros(2,2),S3,zeros(2,2));
X=rand(8,8);
P=X*P*X';Q=inv(X)'*Q*inv(X);
[T,siz]=equil1(P,Q);
P1=clean(T*P*T')
Q1=clean(inv(T)'*Q*inv(T))
```

# See Also

balreal , minreal , equil , hankelsv

# Authors

S. Steer 1987

# Name

feedback — feedback operation

```
Sl=Sl1/.Sl2
```

# Parameters

Sl1,Sl2
: linear systems (`syslin` list) in state-space or transfer form, or ordinary gain matrices.

Sl
: linear system (`syslin` list) in state-space or transfer form

# Description

The feedback operation is denoted by `/.` (slashdot). This command returns `Sl=Sl1*(I+Sl2*Sl1)^-1`, i.e the (negative) feedback of `Sl1` and `Sl2`. `Sl` is the transfer `v -> y` for `y = Sl1 u`, `u = v - Sl2 y`.

The result is the same as `Sl=LFT([0,I;I,-Sl2],Sl1)`.

Caution: do not use with decimal point (e.g. `1/.1` is ambiguous!)

# Examples

```
S1=ssrand(2,2,3);S2=ssrand(2,2,2);
W=S1/.S2;
ss2tf(S1/.S2)
//Same operation by LFT:
ss2tf(lft([zeros(2,2),eye(2,2);eye(2,2),-S2],S1))
//Other approach: with constant feedback
BigS=sysdiag(S1,S2); F=[zeros(2,2),eye(2,2);-eye(2,2),zeros(2,2)];
Bigclosed=BigS/.F;
W1=Bigclosed(1:2,1:2);   //W1=W (in state-space).
ss2tf(W1)
//Inverting
ss2tf(S1*inv(eye()+S2*S1))
```

# See Also

lft , sysdiag , augment , obscont

# Name

findABCD — discrete-time system subspace identification

```
[SYS,K] = findABCD(S,N,L,R,METH,NSMPL,TOL,PRINTW)
SYS = findABCD(S,N,L,R,METH)

[SYS,K,Q,Ry,S,RCND] = findABCD(S,N,L,R,METH,NSMPL,TOL,PRINTW)
[SYS,RCND] = findABCD(S,N,L,R,METH)
```

# Parameters

S
: integer, the number of block rows in the block-Hankel matrices

N
: integer, the system order

L
: integer, the number of output

R
: matrix, relevant part of the R factor of the concatenated block-Hankel matrices computed by a call to findr.

METH
: integer, an option for the method to use

  = 1
  : MOESP method with past inputs and outputs;

  = 2
  : N4SID method;

  = 3
  : combined method: A and C via MOESP, B and D via N4SID.

  Default: METH = 3.

NSMPL
: integer, the total number of samples used for calculating the covariance matrices and the Kalman predictor gain. This parameter is not needed if the covariance matrices and/or the Kalman predictor gain matrix are not desired. If NSMPL = 0, then K, Q, Ry, and S are not computed. Default: NSMPL = 0.

TOL
: the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW
: integer, switch for printing the warning messages.

  PRINTW
  : = 1: print warning messages;

  PRINTW
  : = 0: do not print warning messages.

  Default: PRINTW = 0.

SYS
: computes a state-space realization SYS = (A,B,C,D) (an syslin object)

K
: the Kalman predictor gain K (if NSMPL > 0)

Q
: state covariance

Ry
: output covariance

S
: state-output cross-covariance

RCND
: vector, reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions

# Description

Finds the system matrices and the Kalman gain of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP and/or N4SID).

- [SYS,K] = findABCD(S,N,L,R,METH,NSMPL,TOL,PRINTW) computes a state- space realization SYS = (A,B,C,D) (an ss object), and the Kalman predictor gain K (if NSMPL > 0). The model structure is:

  ```
  x(k+1) = Ax(k) + Bu(k) + Ke(k),   k >= 1,
  y(k)   = Cx(k) + Du(k) + e(k),
  ```

  where x(k) and y(k) are vectors of length N and L, respectively.

- [SYS,K,Q,Ry,S,RCND] = findABCD(S,N,L,R,METH,NSMPL,TOL,PRINTW) also returns the state, output, and state-output (cross-)covariance matrices Q, Ry, and S (used for computing the Kalman gain), as well as the vector RCND of length lr containing the reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions, where

```
lr = 4,  if Kalman gain matrix K is not required, and
lr = 12, if Kalman gain matrix K is required.
```

Matrix R, computed by findR, should be determined with suitable arguments METH and JOBD. METH = 1 and JOBD = 1 must be used in findR, for METH = 1 in findABCD; METH = 1 must be used in findR, for METH = 3 in findABCD.

# Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
```

```
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;
[R,N1,SVAL] = findR(S,Y',U');
N=3;
SYS1 = findABCD(S,N,1,R) ;SYS1.dt=0.1;

SYS1.X0 = inistate(SYS1,Y',U');

Y1=flts(U,SYS1);
clf();plot2d((1:nsmp)',[Y',Y1'])
```

# See Also

findAC , findBD , findBDK , findR , sorder , sident

# Name

findAC — discrete-time system subspace identification

```
[A,C] = findAC(S,N,L,R,METH,TOL,PRINTW)
[A,C,RCND] = findAC(S,N,L,R,METH,TOL,PRINTW)
```

# Parameters

S
: integer, the number of block rows in the block-Hankel matrices

N
: integer

L
: integer

R
: matrix, relevant part of the R factor of the concatenated block-Hankel matrices computed by a call to findr.

METH
: integer, an option for the method to use

= 1
: MOESP method with past inputs and outputs;

= 2
: N4SID method;

Default: METH = 3.

TOL
: the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW
: integer, switch for printing the warning messages.

PRINTW
: = 1: print warning messages;

= 0
: do not print warning messages.

Default: PRINTW = 0.

A
: matrix, state system matrix

C
: matrix, output system matrix

RCND
: vector of length 4, condition numbers of the matrices involved in rank decision

# Description

finds the system matrices A and C of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP or N4SID).

- [A,C] = findAC(S,N,L,R,METH,TOL,PRINTW) computes the system matrices A and C. The model structure is: x(k+1) = Ax(k) + Bu(k) + Ke(k), k >= 1, y(k) = Cx(k) + Du(k) + e(k), where x(k) and y(k) are vectors of length N and L, respectively.
- [A,C,RCND] = findAC(S,N,L,R,METH,TOL,PRINTW) also returns the vector RCND of length 4 containing the condition numbers of the matrices involved in rank decisions.

Matrix R, computed by findR, should be determined with suitable arguments METH and JOBD.

# Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;L=1;
[R,N,SVAL] = findR(S,Y',U');

N=3;
METH=3;TOL=-1;
[A,C] = findAC(S,N,L,R,METH,TOL);
```

# See Also

findABCD , findBD , findBDK , findR , sorder , sident

# Name

findBD — initial state and system matrices B and D of a discrete-time system

```
[(x0) (,B (,D)) (,V) (,rcnd)] = findBD(jobx0,comuse (,job),A (,B),C (,D),Y
(,U,tol,printw,ldwork))
```

# Parameters

jobx0
: integer option to specify whether or not the initial state should be computed:

=
: 1 : compute the initial state x0;

=
: 2 : do not compute the initial state (possibly, because x0 is known to be zero).

comuse
: integer option to specify whether the system matrices B and D should be computed or used:

=
: 1 : compute the matrices B and D, as specified by job;

=
: 2 : use the matrices B and D, as specified by job;

=
: 3 : do not compute/use the matrices B and D.

job
: integer option to determine which of the system matrices B and D should be computed or used:

=
: 1 : compute/use the matrix B only (D is known to be zero);

=
: 2 : compute/use the matrices B and D.

job must not be specified if jobx0 = 2 and comuse = 2, or if comuse = 3.

A
: state matrix of the given system

B
: optionnal, input matrix of the given system

C
: output matrix of the given system

D
: optionnal, direct feedthrough of the given system

Y
: the t-by-l output-data sequence matrix. Column j of Y contains the t values of the j-th output component for consecutive time increments.

U
: the t-by-m input-data sequence matrix (input when jobx0 = 1 and comuse = 2, or comuse = 1). Column j of U contains the t values of the j-th input component for consecutive time increments.

tol
: optionnal, tolerance used for estimating the rank of matrices. If tol > 0, then the given value of tol is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than 1/tol is considered to be of full rank. Default: m*n*epsilon_machine where epsilon_machine is the relative machine precision.

printw
: optionnal, switch for printing the warning messages.

=
: 1: print warning messages;

=
: 0: do not print warning messages.

Default: printw = 0.

ldwork
: (optional) the workspace size. Default : computed by the formula LDWORK = MAX( minimum workspace size needed, 2*CSIZE/3, CSIZE - ( m + l )*t - 2*n*( n + m + l ) - l*m ) where CSIZE is the cache size in double precision words.

x0
: initial state vector

Br
: system input matrix

Dr
: system direct feedthrough matrix

V
: the n-by-n orthogonal matrix which reduces A to a real Schur form (output when jobx0 = 1 or comuse = 1).

rcnd
: (optional) the reciprocal condition numbers of the matrices involved in rank decisions.

# Description

findBD function for estimating the initial state and the system matrices B and D of a discrete-time system, using SLICOT routine IB01CD.

```
   [x0,Br,V,rcnd] = findBD(1,1,1,A,C,Y,U)
[x0,Br,Dr,V,rcnd] = findBD(1,1,2,A,C,Y,U)
      [Br,V,rcnd] = findBD(2,1,1,A,C,Y,U)
    [B,Dr,V,rcnd] = findBD(2,1,2,A,C,Y,U)
      [x0,V,rcnd] = findBD(1,2,1,A,B,C,Y,U)
      [x0,V,rcnd] = findBD(1,2,2,A,B,C,D,Y,U)
        [x0,rcnd] = findBD(2,2)      // (Set x0 = 0, rcnd = 1)
      [x0,V,rcnd] = findBD(1,3,A,C,Y)
```

Note: the example lines above may contain at the end the parameters tol, printw, ldwork.

FINDBD estimates the initial state and/or the system matrices Br and Dr of a discrete-time system, given the system matrices A, C, and possibly B, D, and the input and output trajectories of the system.

The model structure is :

```
x(k+1) = Ax(k) + Bu(k),   k >= 1,
y(k)   = Cx(k) + Du(k),
```

where x(k) is the n-dimensional state vector (at time k),

u(k) is the m-dimensional input vector,

y(k) is the l-dimensional output vector,

and A, B, C, and D are real matrices of appropriate dimensions.

# Comments

1. The n-by-m system input matrix B is an input parameter when jobx0 = 1 and comuse = 2, and it is an output parameter when comuse = 1.

2. The l-by-m system matrix D is an input parameter when jobx0 = 1, comuse = 2 and job = 2, and it is an output parameter when comuse = 1 and job = 2.

3. The n-vector of estimated initial state x(0) is an output parameter when jobx0 = 1, but also when jobx0 = 2 and comuse <= 2, in which case it is set to 0.

4. If ldwork is specified, but it is less than the minimum workspace size needed, that minimum value is used instead.

# Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;L=1;
[R,N,SVAL] = findR(S,Y',U');

N=3;
METH=3;TOL=-1;
[A,C] = findAC(S,N,L,R,METH,TOL);
[X0,B,D] = findBD(1,1,2,A,C,Y',U')
SYS1=syslin(1,A,B,C,D,X0);

Y1=flts(U,SYS1);
clf();plot2d((1:nsmp)',[Y',Y1'])
```

# See Also

inistate , findx0BD , findABCD , findAC , findBD

# Authors

V. Sima, Katholieke Univ. Leuven, Belgium, May 2000. (Revisions: V. Sima, July 2000)

# Name

findBDK — Kalman gain and B D system matrices of a discrete-time system

```
[B,D,K] = findBDK(S,N,L,R,A,C,METH,JOB,NSMPL,TOL,PRINTW)
[B,D,RCND] = findBDK(S,N,L,R,A,C,METH,JOB)
[B,D,K,Q,Ry,S,RCND] = findBDK(S,N,L,R,A,C,METH,JOB,NSMPL,TOL,PRINTW)
```

# Parameters

S
: integer, the number of block rows in the block-Hankel matrices

N
: integer

L
: integer

R
: matrix, relevant part of the R factor of the concatenated block-Hankel matrices computed by a call to findR.

A
: square matrix

C
: matrix

METH
: integer, an option for the method to use

  = 1
  : MOESP method with past inputs and outputs;

  = 2
  : N4SID method;

  Default: METH = 2.

JOB
: an option specifying which system matrices should be computed:

  = 1
  : compute the matrix B;

  = 2
  : compute the matrices B and D.

  Default: JOB = 2.

NSMPL
: integer, the total number of samples used for calculating the covariance matrices and the Kalman predictor gain. This parameter is not needed if the covariance matrices and/or the Kalman predictor gain matrix are not desired. If NSMPL = 0, then K, Q, Ry, and S are not computed. Default: NSMPL = 0.

TOL
: the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW
: integer, switch for printing the warning messages.

  PRINTW
  : = 1: print warning messages;

  PRINTW
  : = 0: do not print warning messages.

  Default: PRINTW = 0.

SYS
: computes a state-space realization SYS = (A,B,C,D) (an syslin object)

K
: the Kalman predictor gain K (if NSMPL > 0)

Q
: state covariance

Ry
: output covariance

S
: state-output cross-covariance

RCND
: he vector of length 12 containing the reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions.

# Description

finds the system matrices B and D and the Kalman gain of a discrete-time system, given the system order, the matrices A and C, and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP or N4SID).

- [B,D,K] = findBDK(S,N,L,R,A,C,METH,JOB,NSMPL,TOL,PRINTW) computes the system matrices B (if JOB = 1), B and D (if JOB = 2), and the Kalman predictor gain K (if NSMPL > 0). The model structure is:

  ```
  x(k+1) = Ax(k) + Bu(k) + Ke(k),   k >= 1,
  y(k)   = Cx(k) + Du(k) + e(k),
  ```

  where x(k) and y(k) are vectors of length N and L, respectively.

- [B,D,RCND] = findBDK(S,N,L,R,A,C,METH,JOB) also returns the vector RCND of length 4 containing the reciprocal condition numbers of the matrices involved in rank decisions.

- [B,D,K,Q,Ry,S,RCND] = findBDK(S,N,L,R,A,C,METH,JOB,NSMPL,TOL,PRINTW) also returns the state, output, and state-output (cross-)covariance matrices Q, Ry, and S (used for computing the Kalman gain), as well as the vector RCND of length 12 containing the reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions.

Matrix R, computed by findR, should be determined with suitable arguments METH and JOBD. METH = 1 and JOBD = 1 must be used in findR, for METH = 1 in findBDK. Using METH = 1 in FINDR and METH = 2 in findBDK is allowed.

The number of output arguments may vary, but should correspond to the input arguments, e.g.,

```
         B = findBDK(S,N,L,R,A,C,METH,1)  or
     [B,D] = findBDK(S,N,L,R,A,C,METH,2)  or
[B,D,RCND] = findBDK(S,N,L,R,A,C,METH,2)
```

# Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;L=1;
[R,N,SVAL] = findR(S,Y',U');

N=3;
METH=3;TOL=-1;
[A,C] = findAC(S,N,L,R,METH,TOL);
[B,D,K] = findBDK(S,N,L,R,A,C);
SYS1=syslin(1,A,B,C,D);

SYS1.X0 = inistate(SYS1,Y',U');

Y1=flts(U,SYS1);
clf();plot2d((1:nsmp)',[Y',Y1'])
```

# See Also

findABCD , findAC , findBD , findR , sorder , sident

# Name

findR — Preprocessor for estimating the matrices of a linear time-invariant dynamical system

```
[R,N [,SVAL,RCND]] = findR(S,Y,U,METH,ALG,JOBD,TOL,PRINTW)
[R,N] = findR(S,Y)
```

# Parameters

S
: the number of block rows in the block-Hankel matrices.

Y

U

METH
: an option for the method to use:

  1
  : MOESP method with past inputs and outputs;

  2
  : N4SI15 0 1 1 1000D method.

  Default: METH = 1.

ALG
: an option for the algorithm to compute the triangular factor of the concatenated block-Hankel matrices built from the input-output data:

  1
  : Cholesky algorithm on the correlation matrix;

  2
  : fast QR algorithm;

  3
  : standard QR algorithm.

  Default: ALG = 1.

JOBD
: an option to specify if the matrices B and D should later be computed using the MOESP approach:

  =
  : 1 : the matrices B and D should later be computed using the MOESP approach;

  =
  : 2 : the matrices B and D should not be computed using the MOESP approach.

  Default: JOBD = 2. This parameter is not relevant for METH = 2.

TOL
: a vector of length 2 containing tolerances:

  TOL
  : (1) is the tolerance for estimating the rank of matrices. If TOL(1) > 0, the given value of TOL(1) is used as a lower bound for the reciprocal condition number.

    Default: TOL(1) = prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

TOL
(2) is the tolerance for estimating the system order. If TOL(2) >= 0, the estimate is indicated by the index of the last singular value greater than or equal to TOL(2). (Singular values less than TOL(2) are considered as zero.)

When TOL(2) = 0, then S*epsilon_machine*sval(1) is used instead TOL(2), where sval(1) is the maximal singular value. When TOL(2) < 0, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor. Default: TOL(2) = -1.

PRINTW
a switch for printing the warning messages.

=
1: print warning messages;

=
0: do not print warning messages.

Default: PRINTW = 0.

R

N
the order of the discrete-time realization

SVAL
singular values SVAL, used for estimating the order.

RCND
vector of length 2 containing the reciprocal condition numbers of the matrices involved in rank decisions or least squares solutions.

# Description

findR Preprocesses the input-output data for estimating the matrices of a linear time-invariant dynamical system, using Cholesky or (fast) QR factorization and subspace identification techniques (MOESP or N4SID), and estimates the system order.

[R,N] = findR(S,Y,U,METH,ALG,JOBD,TOL,PRINTW) returns the processed upper triangular factor R of the concatenated block-Hankel matrices built from the input-output data, and the order N of a discrete-time realization. The model structure is:

```
x(k+1) = Ax(k) + Bu(k) + w(k),   k >= 1,
y(k)   = Cx(k) + Du(k) + e(k).
```

The vectors y(k) and u(k) are transposes of the k-th rows of Y and U, respectively.

[R,N,SVAL,RCND] = findR(S,Y,U,METH,ALG,JOBD,TOL,PRINTW) also returns the singular values SVAL, used for estimating the order, as well as, if meth = 2, the vector RCND of length 2 containing the reciprocal condition numbers of the matrices involved in rank decisions or least squares solutions.

[R,N] = findR(S,Y) assumes U = [] and default values for the remaining input arguments.

# Examples

```
//generate data from a given linear system
```

```
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
U=(ones(1,1000)+rand(1,1000,'normal'));
Y=(flts(U,SYS)+0.5*rand(1,1000,'normal'));
// Compute R

[R,N,SVAL] = findR(15,Y',U');
SVAL
N
```

# See Also

findABCD , findAC , findBD , findBDK , sorder , sident

# Name

findx0BD — Estimates state and B and D matrices of a discrete-time linear system

```
[X0,B,D] = findx0BD(A,C,Y,U,WITHX0,WITHD,TOL,PRINTW)
[x0,B,D,V,rcnd] = findx0BD(A,C,Y,U)
```

# Parameters

A
: state matrix of the system

C
: C matrix of the system

Y
: system output

U
: system input

WITHX0
: a switch for estimating the initial state x0.

=
: 1: estimate x0;

=
: 0: do not estimate x0.

Default: WITHX0 = 1.

WITHD
: a switch for estimating the matrix D.

=
: 1: estimate the matrix D;

=
: 0: do not estimate the matrix D.

Default: WITHD = 1.

TOL
: the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW
: a switch for printing the warning messages.

=
: 1: print warning messages;

=
: 0: do not print warning messages.

Default: PRINTW = 0.

X0
: intial state of the estimated linear system.

B
: B matrix of the estimated linear system.

D
: D matrix of the estimated linear system.

V
: orthogonal matrix which reduces the system state matrix A to a real Schur form

rcnd
: estimates of the reciprocal condition numbers of the matrices involved in rank decisions.

# Description

findx0BD Estimates the initial state and/or the matrices B and D of a discrete-time linear system, given the (estimated) system matrices A, C, and a set of input/output data.

[X0,B,D] = findx0BD(A,C,Y,U,WITHX0,WITHD,TOL,PRINTW) estimates the initial state X0 and the matrices B and D of a discrete-time system using the system matrices A, C, output data Y and the input data U. The model structure is :

```
x(k+1) = Ax(k) + Bu(k),   k >= 1,
y(k)   = Cx(k) + Du(k),
```

The vectors y(k) and u(k) are transposes of the k-th rows of Y and U, respectively.

[x0,B,D,V,rcnd] = findx0BD(A,C,Y,U) also returns the orthogonal matrix V which reduces the system state matrix A to a real Schur form, as well as some estimates of the reciprocal condition numbers of the matrices involved in rank decisions.

```
    B = findx0BD(A,C,Y,U,0,0)  returns B only, and
[B,D] = findx0BD(A,C,Y,U,0)    returns B and D only.
```

# Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;L=1;
[R,N,SVAL] = findR(S,Y',U');

N=3;
METH=3;TOL=-1;
[A,C] = findAC(S,N,L,R,METH,TOL);
```

```
[X0,B,D,V,rcnd] = findx0BD(A,C,Y',U');
SYS1=syslin(1,A,B,C,D,X0);

Y1=flts(U,SYS1);
clf();plot2d((1:nsmp)',[Y',Y1'])
```

# See Also

findBD , inistate

# Name

flts — time response (discrete time, sampled system)

```
[y [,x]]=flts(u,sl [,x0])
[y]=flts(u,sl [,past])
```

# Parameters

u
: matrix (input vector)

sl
: list (linear system `syslin`)

x0
: vector (initial state ; default value=`0`)

past
: matrix (of the past ; default value=`0`)

x,y
: matrices (state and output)

# Description

- State-space form:

`sl` is a discrete linear system given by its state space representation (see syslin ):

`sl=syslin('d',A,B,C,D)`:

```
x[t+1] = A x[t] + B u[t]
y[t] = C x[t] + D u[t]
```

or, more generally, if `D` is a polynomial matrix (`p = degree(D(z))`):

```
D(z) = D_0 + z D_1 + z^2 D_2 +..+ z^p D_p
y[t] = C x[t] + D_0 u[t] + D_1 u[t+1] +..+ D_[p] u[t+p]
```

- Transfer form:

`y=flts(u,sl[,past])`. Here `sl` is a linear system in transfer matrix representation i.e

`sl=syslin('d',transfer_matrix)` (see `syslin`).

```
past = [u     ,...,  u   ]
       [ -nd           -1]
       [y     ,...,  y   ]
       [ -nd           -1]
```

is the matrix of past values of u and y.

`nd` is the maximum of degrees of lcm's of each row of the denominator matrix of `sl`.

```
u=[u0 u1 ... un]  (input)
y=[y0 y1 ... yn]  (output)
```

p is the difference between maximum degree of numerator and maximum degree of denominator

# Examples

```
sl=syslin('d',1,1,1);u=1:10;
y=flts(u,sl);
plot2d(y)
[y1,x1]=flts(u(1:5),sl);y2=flts(u(6:10),sl,x1);
y-[y1,y2]

//With polynomial D:
z=poly(0,'z');
D=1+z+z^2; p =degree(D);
sl=syslin('d',1,1,1,D);
y=flts(u,sl);[y1,x1]=flts(u(1:5),sl);
y2=flts(u(5-p+1:10),sl,x1);  // (update)
y-[y1,y2]

//Delay (transfer form): flts(u,1/z)
// Usual responses
z=poly(0,'z');
h=syslin(0.1,(1-2*z)/(z^2+0.3*z+1))
imprep=flts(eye(1,20),tf2ss(h));   //Impulse response
clf();
plot(imprep,'b')
u=ones(1,20);
stprep=flts(ones(1,20),tf2ss(h));   //Step response
plot(stprep,'g')

// Other examples
A=[1 2 3;0 2 4;0 0 1];B=[1 0;0 0;0 1];C=eye(3,3);Sys=syslin('d',A,B,C);
H=ss2tf(Sys); u=[1;-1]*(1:10);
//
yh=flts(u,H); ys=flts(u,Sys);
norm(yh-ys,1)
//hot restart
[ys1,x]=flts(u(:,1:4),Sys);ys2=flts(u(:,5:10),Sys,x);
norm([ys1,ys2]-ys,1)
//
yh1=flts(u(:,1:4),H);yh2=flts(u(:,5:10),H,[u(:,2:4);yh(:,2:4)]);
norm([yh1,yh2]-yh,1)
//with D&lt;&gt;0
D=[-3 8;4 -0.5;2.2 0.9];
Sys=syslin('d',A,B,C,D);
H=ss2tf(Sys); u=[1;-1]*(1:10);
rh=flts(u,H); rs=flts(u,Sys);
norm(rh-rs,1)
//hot restart
[ys1,x]=flts(u(:,1:4),Sys);ys2=flts(u(:,5:10),Sys,x);
norm([ys1,ys2]-rs,1)
//With H:
yh1=flts(u(:,1:4),H);yh2=flts(u(:,5:10),H,[u(:,2:4); yh1(:,2:4)]);
```

```
norm([yh1,yh2]-rh)
```

## See Also

ltitr, dsimul, rtitr

# Name

fourplan — augmented plant to four plants

```
[P11,P12,P21,P22]=fourplan(P,r)
```

# Parameters

P
: `syslin` list (linear system)

r
: 1x2 row vector, dimension of `P22`

P11,P12,P21,P22
: `syslin` lists.

# Description

Utility function.

`P` being partitioned as follows:

```
P=[ P11 P12;
    P21 P22]
```

with `size(P22)=r` this function returns the four linear systems `P11,P12,P21,P22`.

# See Also

lqg , lqg2stan , lqr , lqe , lft

# Name

frep2tf — transfer function realization from frequency response

```
[h [,err]]=frep2tf(frq,repf,dg [,dom,tols,weight])
```

# Parameters

frq
: vector of frequencies in Hz.

repf
: vector of frequency response

dg
: degree of linear system

dom
: time domain (`'c'` or `'d'` or `dt`)

tols
: a vector of size 3 giving the relative and absolute tolerance and the maximum number of iterations (default values are `rtol=1.e-2; atol=1.e-4, N=10`).

weight
: vector of weights on frequencies

h
: SISO transfer function

err
: error (for example if `dom='c' sum(abs(h(2i*pi*frq) - rep)^2)/size(frq,*)`)

# Description

Frequency response to transfer function conversion. The order of `h` is a priori given in `dg` which must be provided. The following linear system is solved in the least square sense.

```
weight(k)*(n( phi_k) - d(phi_k)*rep_k)=0, k=1,..,n
```

where `phi_k=  2*%i*%pi*frq` when `dom='c'` and `phi_k=exp(2*%i*%pi*dom*frq` if not. If the `weight` vector is not given a default penalization is used (when `dom='c'`).

A stable and minimum phase system can be obtained by using function `factors`.

# Examples

```
s=poly(0,'s');
h=syslin('c',(s-1)/(s^3+5*s+20))
frq=0:0.05:3;repf=repfreq(h,frq);
clean(frep2tf(frq,repf,3))

Sys=ssrand(1,1,10);
frq=logspace(-3,2,200);
[frq,rep]=repfreq(Sys,frq);  //Frequency response of Sys
```

```
[Sys2,err]=frep2tf(frq,rep,10);Sys2=clean(Sys2)//Sys2 obtained from freq. resp
[frq,rep2]=repfreq(Sys2,frq); //Frequency response of Sys2
clf();bode(frq,[rep;rep2])   //Responses of Sys and Sys2
[gsort(spec(Sys('A'))),gsort(roots(Sys2('den')))] //poles


dom=1/1000; // Sampling time
z=poly(0,'z');
h=syslin(dom,(z^2+0.5)/(z^3+0.1*z^2-0.5*z+0.08))
frq=(0:0.01:0.5)/dom;repf=repfreq(h,frq);
[Sys2,err]=frep2tf(frq,repf,3,dom);
[frq,rep2]=repfreq(Sys2,frq); //Frequency response of Sys2
clf();plot2d1("onn",frq',abs([repf;rep2])');
```

# See Also

imrep2ss , arl2 , time_id , armax , frfit

# Name

freq — frequency response

```
[x]=freq(A,B,C [,D],f)
[x]=freq(NUM,DEN,f)
```

# Parameters

A, B, C, D
: real matrices of respective dimensions `nxn, nxp, mxn, mxp`.

NUM,DEN
: polynomial matrices of dimension `mxp`

x
: real or complex matrix

# Description

`x=freq(A,B,C [,D],f)` returns a real or complex `mxp*t` matrix such that:

`x(:,k*p:(k+1)*p)= C*inv(f(k)*eye()-A)*B + D`.

Thus, for `f` taking values along the imaginary axis or on the unit circle `x` is the continuous or discrete time frequency response of `(A,B,C,D)`.

`x=freq(NUM,DEN,f)` returns a real or complex matrix `x` such that columns `k*(p-1)+1` to `k*p` of `x` contain the matrix `NUM(f(k))./DEN(f(k))`

# Examples

```
s=poly(0,'s');
sys=(s+1)/(s^3-5*s+4)
rep=freq(sys("num"),sys("den"),[0,0.9,1.1,2,3,10,20])
[horner(sys,0),horner(sys,20)]
//
Sys=tf2ss(sys);
[A,B,C,D]=abcd(Sys);
freq(A,B,C,[0,0.9,1.1,2,3,10,20])
```

# See Also

repfreq , horner

# Name

freson — peak frequencies

```
fr=freson(h)
```

# Parameters

h
: `syslin` list

fr
: vector of peak frequencies in Hz

# Description

returns the vector of peak frequencies in Hz for the SISO plant `h`

# Examples

```
h=syslin('c',-1+%s,(3+2*%s+%s^2)*(50+0.1*%s+%s^2))
fr=freson(h)
bode(h)
g=20*log(abs(repfreq(h,fr)))/log(10)
```

# See Also

frep2tf , zgrid , h_norm

# Name

fspecg — stable factorization

```
[gm]=fspecg(g).
```

# Parameters

g,gm
: `syslin` lists (linear systems in state-space representation)

# Description

returns `gm` with `gm` and `gm^-1` stable such that:

```
gtild(g)*g = gtild(gm)*gm
```

`g` and `gm` are continuous-time linear systems in state-space form.

Imaginary-axis poles are forbidden.

# Name

fstabst — Youla's parametrization

```
[J]=fstabst(P,r)
```

# Parameters

P
: `syslin` list (linear system)

r
: 1x2 row vector, dimension of `P22`

J
: `syslin` list (linear system in state-space representation)

# Description

Parameterization of all stabilizing feedbacks.

`P` is partitioned as follows:

```
P=[ P11 P12;
    P21 P22]
```

(in state-space or transfer form: automatic conversion in state-space is done for the computations)

`r` = size of `P22` subsystem, (2,2) block of `P`

```
J =[J11 J12;
    J21 J22]
```

`K` is a stabilizing controller for `P` (i.e. `P22`) iff `K=lft(J,r,Q)` with `Q` stable.

The central part of `J` , `J11` is the lqg regulator for `P`

This `J` is such that defining `T` as the 2-port `lft` of `P` and `J` : `[T,rt]=lft(P,r,J,r)` one has that `T12` is inner and `T21` is co-inner.

# Examples

```
ny=2;nu=3;nx=4;
P22=ssrand(ny,nu,nx);
bigQ=rand(nx+nu,nx+nu);bigQ=bigQ*bigQ';
bigR=rand(nx+ny,nx+ny);bigR=bigR*bigR';
[P,r]=lqg2stan(P22,bigQ,bigR);
J=fstabst(P,r);
Q=ssrand(nu,ny,1);Q('A')=-1;  //Stable Q
K=lft(J,r,Q);
A=h_cl(P,r,K); spec(A)
```

# See Also

obscont , lft , lqg , lqg2stan

# Name

g_margin — gain margin and associated crossover frequency

```
gm=g_margin(h)
[gm,fr]=g_margin(h)
```

# Parameters

h
: a SISO linear system (see :syslin).

gm
: a number, the gain margin (in dB) if any of `Inf`

fr
: a number, the associated frequency in hertz, or an empty matrix if the gain margin does not exist.

# Description

Given a SISO linear system in continuous or discrete time, `g_margin` returns `gm`, the gain margin in dB of `h` and `fr`, the achieved corresponding frequency in hz.

The gain margin, if it exists, is the minimal value of the system gain at points where the nyquist plot crosses the negative real axis. In other words the gain margin is `20*log10(1/g)` where `g` is the open loop gain of `h` when the frequency response phase of `h` equals -180°

The algorithm uses polynomial root finder to solve the equations:

h(s)=h(-s)
: for the continuous time case.

h(z)=h(1/z)
: for the discrete time case.

# Examples

```
h=syslin('c',-1+%s,3+2*%s+%s^2) //continuous time case
[g,fr]=g_margin(h)
[g,fr]=g_margin(h-10)
nyquist(h-10)

h = syslin(0.1,0.04798*%z+0.0464,%z^2-1.81*%z+0.9048);//discrete time case
[g ,fr]=g_margin(h);
show_margins(h)
```

# See Also

p_margin, show_margins, repfreq, black, bode, chart, nyquist

# Authors

Serge Steer, INRIA

# Name

gamitg — H-infinity gamma iterations

```
[gopt]=gamitg(G,r,prec [,options]);
```

# Parameters

G
: `syslin` list (plant realization )

r
: 1x2 row vector (dimension of `G22`)

prec
: desired relative accuracy on the norm

option
: string `'t'`

gopt
: real scalar, optimal H-infinity gain

# Description

`gopt=gamitg(G,r,prec [,options])` returns the H-infinity optimal gain `gopt`.

`G` contains the state-space matrices `[A,B,C,D]` of the plant with the usual partitions:

```
B = ( B1 , B2 ) ,    C = ( C1 ) ,    D = ( D11  D12)
                         ( C2 )          ( D21  D22)
```

These partitions are implicitly given in `r`: `r(1)` and `r(2)` are the dimensions of `D22` (rows x columns)

With `option='t'`, `gamitg` traces each bisection step, i.e., displays the lower and upper bounds and the current test point.

# See Also

ccontrg , h_inf

# Authors

P. Gahinet

# Name

gcare — control Riccati equation

```
[X,F]=gcare(Sl)
```

# Parameters

Sl
: linear system (`syslin` list)

X
: symmetric matrix

F
: real matrix

# Description

Generalized Control Algebraic Riccati Equation (GCARE). `X` = solution , `F` = gain.

The GCARE for `Sl=[A,B,C,D]` is:

```
(A-B*Si*D'*C)'*X+X*(A-B*Si*D'*C)-X*B*Si*B'*X+C'*Ri*C=0
```

where `S=(eye()+D'*D)`, `Si=inv(S)`, `R=(eye()+D*D')`, `Ri=inv(R)` and `F=-Si*(D'*C+B'*X)` is such that `A+B*F` is stable.

# See Also

gfare

# Name

gfare — filter Riccati equation

```
[Z,H]=gfare(Sl)
```

# Parameters

Sl
: linear system (`syslin` list)

Z
: symmetric matrix

H
: real matrix

# Description

Generalized Filter Algebraic Riccati Equation (GFARE). `Z` = solution, `H` = gain.

The GFARE for `Sl=[A,B,C,D]` is:

```
(A-B*D'*Ri*C)*Z+Z*(A-B*D'*Ri*C)'-Z*C'*Ri*C*Z+B*Si*B'=0
```

where `S=(eye()+D'*D)`, `Si=inv(S)`, `R=(eye()+D*D')`, `Ri=inv(R)` and `H=-(B*D'+Z*C')*Ri` is such that `A+H*C` is stable.

# See Also

gcare

# Name

gfrancis — Francis equations for tracking

```
[L,M,T]=gfrancis(Plant,Model)
```

# Parameters

Plant
: `syslin` list

Model
: `syslin` list

L,M,T
: real matrices

# Description

Given the the linear plant:

```
x'= F*x + G*u
y = H*x + J*u
```

and the linear model

```
xm'= A*xm + B*um
ym = C*xm + D*um
```

the goal is for the plant to track the model i.e. `e = y - ym ---> 0` while keeping stable the state x(t) of the plant. `u` is given by feedforward and feedback

```
u = L*xm + M*um + K*(x-T*xm) = [K , L-K*T] *(x,xm) + M*um
```

The matrices T,L,M satisfy generalized Francis equations

```
F*T + G*L = T*A
H*T + J*L = C
      G*M = T*B
      J*M = D
```

The matrix `K` must be chosen as stabilizing the pair `(F,G)` See example of use in directory `demos/tracking`.

# Examples

```
Plant=ssrand(1,3,5);
[F,G,H,J]=abcd(Plant);
nw=4;nuu=2;A=rand(nw,nw);
```

```
st=maxi(real(spec(A)));A=A-st*eye(A);
B=rand(nw,nuu);C=2*rand(1,nw);D=0*rand(C*B);
Model=syslin('c',A,B,C,D);
[L,M,T]=gfrancis(Plant,Model);
norm(F*T+G*L-T*A,1)
norm(H*T+J*L-C,1)
norm(G*M-T*B,1)
norm(J*M-D,1)
```

# See Also

lqg , ppol

# Name

gtild — tilde operation

```
Gt=gtild(G)
Gt=gtild(G,flag)
```

# Parameters

G
: either a polynomial or a linear system (`syslin` list) or a rational matrix

Gt
: same as G

flag
: character string: either `'c'` or `'d'` (optional parameter).

# Description

If `G` is a polynomial matrix (or a polynomial), `Gt=gtild(G,'c')` returns the polynomial matrix `Gt(s)=G(-s)'`.

If `G` is a polynomial matrix (or a polynomial), `Gt=gtild(G,'d')` returns the polynomial matrix `Gt=G(1/z)*z^n` where n is the maximum degree of `G`.

For continuous-time systems represented in state-space by a `syslin` list, `Gt = gtild(G,'c')` returns a state-space representation of `G(-s)'` i.e the `ABCD` matrices of `Gt` are `A',-C', B', D'`. If `G` is improper ( `D= D(s)`) the `D` matrix of `Gt` is `D(-s)'`.

For discrete-time systems represented in state-space by a `syslin` list, `Gt = gtild(G,'d')` returns a state-space representation of `G(-1/z)'` i.e the (possibly improper) state-space representation of `-z*C*inv(z*A-B)*C + D(1/z)` .

For rational matrices, `Gt = gtild(G,'c')` returns the rational matrix `Gt(s)=G(-s)` and `Gt = gtild(G,'d')` returns the rational matrix `Gt(z)= G(1/z)'`.

The parameter `flag` is necessary when `gtild` is called with a polynomial argument.

# Examples

```
//Continuous time
s=poly(0,'s');G=[s,s^3;2+s^3,s^2-5]
Gt=gtild(G,'c')
Gt-horner(G,-s)'   //continuous-time interpretation
Gt=gtild(G,'d');
Gt-horner(G,1/s)'*s^3  //discrete-time interpretation
G=ssrand(2,2,3);Gt=gtild(G);   //State-space (G is cont. time by default)
clean((horner(ss2tf(G),-s))'-ss2tf(Gt))   //Check

// Discrete-time
z=poly(0,'z');
Gss=ssrand(2,2,3);Gss('dt')='d'; //discrete-time
Gss(5)=[1,2;0,1];   //With a constant D matrix
G=ss2tf(Gss);Gt1=horner(G,1/z)';
Gt=gtild(Gss);
Gt2=clean(ss2tf(Gt)); clean(Gt1-Gt2)  //Check
```

```
//Improper systems
z=poly(0,'z');
Gss=ssrand(2,2,3);Gss(7)='d'; //discrete-time
Gss(5)=[z,z^2;1+z,3];    //D(z) is polynomial
G=ss2tf(Gss);Gt1=horner(G,1/z)';  //Calculation in transfer form
Gt=gtild(Gss);    //..in state-space
Gt2=clean(ss2tf(Gt));clean(Gt1-Gt2)  //Check
```

# See Also

syslin , horner , factors

# Name

h2norm — H2 norm

```
[n]=h2norm(Sl [,tol])
```

# Parameters

Sl
: linear system (`syslin` list)

n
: real scalar

# Description

produces the H2 norm of a linear continuous time system `Sl`.

(For `Sl` in state-space form `h2norm` uses the observability gramian and for `Sl` in transfer form `h2norm` uses a residue method)

# Name

h_cl — closed loop matrix

```
[Acl]=h_cl(P,r,K)
[Acl]=h_cl(P22,K)
```

# Parameters

P, P22
: linear system (`syslin` list), augmented plant or nominal plant respectively

r
: 1x2 row vector, dimensions of 2,2 part of `P` (`r=[rows,cols]=size(P22)`)

K
: linear system (`syslin` list), controller

Acl
: real square matrix

# Description

Given the standard plant `P` (with `r=size(P22)`) and the controller `K`, this function returns the closed loop matrix `Acl`.

The poles of `Acl` must be stable for the internal stability of the closed loop system.

`Acl` is the `A`-matrix of the linear system `[I -P22;-K I]^-1` i.e. the `A`-matrix of `lft(P,r,K)`

# See Also

lft

# Authors

F. D.

# Name

h_inf — H-infinity (central) controller

```
[Sk,ro]=h_inf(P,r,romin,romax,nmax)
[Sk,rk,ro]=h_inf(P,r,romin,romax,nmax)
```

# Parameters

P
: `syslin` list : continuous-time linear system (``augmented'' plant given in state-space form or in transfer form)

r
: size of the `P22` plant i.e. 2-vector `[#outputs,#inputs]`

romin,romax
: a priori bounds on `ro` with `ro=1/gama^2`; (`romin=0` usually)

nmax
: integer, maximum number of iterations in the gama-iteration.

# Description

`h_inf` computes H-infinity optimal controller for the continuous-time plant `P`.

The partition of `P` into four sub-plants is given through the 2-vector `r` which is the size of the `22` part of `P`.

`P` is given in state-space e.g. `P=syslin('c',A,B,C,D)` with `A,B,C,D` = constant matrices or `P=syslin('c',H)` with `H` a transfer matrix.

`[Sk,ro]=H_inf(P,r,romin,romax,nmax)` returns `ro` in `[romin,romax]` and the central controller `Sk` in the same representation as `P`.

(All calculations are made in state-space, i.e conversion to state-space is done by the function, if necessary).

Invoked with three LHS parameters,

`[Sk,rk,ro]=H_inf(P,r,romin,romax,nmax)` returns `ro` and the Parameterization of all stabilizing controllers:

a stabilizing controller `K` is obtained by `K=lft(Sk,r,PHI)` where `PHI` is a linear system with dimensions `r'` and satisfy:

`H_norm(PHI) < gamma`. `rk (=r)` is the size of the `Sk22` block and `ro = 1/gama^2` after `nmax` iterations.

Algorithm is adapted from Safonov-Limebeer. Note that `P` is assumed to be a continuous-time plant.

# See Also

gamitg , ccontrg , leqr

# Authors

F.Delebecque INRIA (1990)

# Name

h_inf_st — static H_infinity problem

```
[Kopt,gamaopt]=h_inf_stat(D,r)
```

# Parameters

D
: real matrix

r
: 1x2 vector

Kopt
: matrix

# Description

computes a matrix `Kopt` such that largest singular value of:

`lft(D,r,K)=D11+D12*  K*inv(I-D22*K)*  D21` is minimal (Static `H_infinity` four blocks problem).

D is partionned as `D=[D11 D12; D21 D22]` where `size(D22)=r=[r1 r2]`

# Authors

F.D. ;

# Name

h_norm — H-infinity norm

```
[hinfnorm [,frequency]]=h_norm(sl [,rerr])
```

# Parameters

sl
: the state space system (`syslin` list)

rerr
: max. relative error, default value `1e-8`

hinfnorm
: the infinity norm of `Sl`

frequency
: frequency at which maximum is achieved

# Description

produces the infinity norm of a state-space system (the maximum over all frequencies of the maximum singular value).

# See Also

linfn , linf , svplot

# Name

hankelsv — Hankel singular values

```
[nk2,W]=hankelsv(sl [,tol])
[nk2]=hankelsv(sl [,tol])
```

# Parameters

sl

`syslin` list representing the linear system (state-space).

tol

tolerance parameter for detecting imaginary axis modes (default value is `1000*%eps`).

# Description

returns `nk2`, the squared Hankel singular values of `sl` and `W = P*Q` = controllability gramian times observability gramian.

`nk2` is the vector of eigenvalues of `W`.

# Examples

```
A=diag([-1,-2,-3]);
sl=syslin('c',A,rand(3,2),rand(2,3));[nk2,W]=hankelsv(sl)
[Q,M]=pbig(W,nk2(2)-%eps,'c');
slr=projsl(sl,Q,M);hankelsv(slr)
```

# See Also

balreal , equil , equil1

# Name

hinf — H_infinity design of continuous-time systems

```
[AK,BK,CK,DK,(RCOND)] = hinf(A,B,C,D,ncon,nmeas,gamma)
```

# Parameters

A
: the n-by-n system state matrix A.

B
: the n-by-m system input matrix B.

C
: the p-by-n system output matrix C.

D
: the p-by-m system matrix D.

ncon
: the number of control inputs. m >= ncon >= 0, p-nmeas >= ncon.

nmeas
: the number of measurements. p >= nmeas >= 0, m-ncon >= nmeas.

gamma
: the parameter gamma used in `H_infinity` design. It is assumed that gamma is sufficiently large so that the controller is admissible. gamma >= 0.

AK
: the n-by-n controller state matrix AK.

BK
: the n-by-nmeas controller input matrix BK.

CK
: the ncon-by-n controller output matrix CK.

DK
: the ncon-by-nmeas controller matrix DK.

RCOND
: a vector containing estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller. (See the description of the algorithm in [1].)

  RCOND
  : (1) contains the reciprocal condition number of the control transformation matrix TU,

  RCOND
  : (2) contains the reciprocal condition number of the measurement transformation matrix TY,

  RCOND
  : (3) contains an estimate of the reciprocal condition number of the X-Riccati equation,

  RCOND
  : (4) contains an estimate of the reciprocal condition number of the Y-Riccati equation.

# Description

`[AK,BK,CK,DK,(RCOND)] = hinf(A,B,C,D,ncon,nmeas,gamma)` To compute the matrices of an H-infinity (sub)optimal n-state controller

```
    | AK | BK |
K = |----|----|,
    | CK | DK |
```

for the continuous-time system

```
    | A  | B1  B2  |   | A | B |
P = |----|---------| = |---|---|,
    | C1 | D11 D12 |   | C | D |
    | C2 | D21 D22 |
```

and for a given value of gamma, where B2 has column size of the number of control inputs (ncon) and C2 has row size of the number of measurements (nmeas) being provided to the controller.

# References

[1] P.Hr. Petkov, D.W. Gu and M.M. Konstantinov. Fortran 77 routines for Hinf and H2 design of continuous-time linear control systems. Report98-14, Department of Engineering, Leicester University, August 1998.

# Examples

```
//example from Niconet report SLWN1999-12
//Hinf
A=[-1  0  4  5 -3 -2
   -2  4 -7 -2  0  3
   -6  9 -5  0  2 -1
   -8  4  7 -1 -3  0
    2  5  8 -9  1 -4
    3 -5  8  0  2 -6];

B=[-3 -4 -2  1  0
    2  0  1 -5  2
   -5 -7  0  7 -2
    4 -6  1  1 -2
   -3  9 -8  0  5
    1 -2  3 -6 -2];

C=[ 1 -1  2 -4  0 -3
   -3  0  5 -1  1  1
   -7  5  0 -8  2 -2
    9 -3  4  0  3  7
    0  1 -2  1 -6 -2];

D=[ 1 -2 -3  0  0
    0  4  0  1  0
    5 -3 -4  0  1
```

```
    0  1  0  1 -3
    0  0  1  7  1];
Gamma=10.18425636157899;
[AK,BK,CK,DK] = hinf(A,B,C,D,2,2,Gamma)
```

# See Also

dhinf

# Name

imrep2ss — state-space realization of an impulse response

```
[sl]=imrep2ss(v [,deg])
```

# Parameters

v
: vector coefficients of impulse response, `v(:,k)` is the kth sample

deg
: integer (order required)

sl
: `syslin` list

# Description

Impulse response to linear system conversion (one input). `v` must have an even number of columns.

# Examples

```
s=poly(0,'s');
H=[1/(s+0.5);2/(s-0.4)]   //strictly proper
np=20;w=ldiv(H('num'),H('den'),np);
rep=[w(1:np)';w(np+1:2*np)'];   //The impulse response
H1=ss2tf(imrep2ss(rep))
z=poly(0,'z');
H=(2*z^2-3.4*z+1.5)/(z^2-1.6*z+0.8)     //Proper transfer function
u=zeros(1,20);u(1)=1;
rep=rtitr(H('num'),H('den'),u);   //Impulse rep.
//   <=> rep=ldiv(H('num'),H('den'),20)
w=z*imrep2ss(rep)   //Realization with shifted impulse response
// i.e strictly proper to proper
H2=ss2tf(w);
```

# See Also

frep2tf , arl2 , time_id , armax , markp2ss , ldiv

# Name

inistate — Estimates the initial state of a discrete-time system

```
X0 = inistate(SYS,Y,U,TOL,PRINTW)
X0 = inistate(A,B,C,Y,U);
X0 = inistate(A,C,Y);

[x0,V,rcnd] = inistate(SYS,Y,U,TOL,PRINTW)
```

# Parameters

SYS
: given system, syslin(dt,A,B,C,D)

Y
: the output of the system

U
: the input of the system

TOL
: TOL is the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number.

  Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW
: PRINTW is a switch for printing the warning messages.

  =
  : 1: print warning messages;

  =
  : 0: do not print warning messages.

  Default: PRINTW = 0.

X0
: the estimated initial state vector

V
: orthogonal matrix which reduces the system state matrix A to a real Schur form

rcnd
: estimate of the reciprocal condition number of the coefficient matrix of the least squares problem solved.

# Description

inistate Estimates the initial state of a discrete-time system, given the (estimated) system matrices, and a set of input/output data.

X0 = inistate(SYS,Y,U,TOL,PRINTW) estimates the initial state X0 of the discrete-time system SYS = (A,B,C,D), using the output data Y and the input data U. The model structure is :

```
x(k+1) = Ax(k) + Bu(k),   k >= 1,
```

```
y(k)   = Cx(k) + Du(k),
```

The vectors y(k) and u(k) are transposes of the k-th rows of Y and U, respectively.

Instead of the first input parameter SYS (an syslin object), equivalent information may be specified using matrix parameters, for instance, X0 = inistate(A,B,C,Y,U); or X0 = inistate(A,C,Y);

[x0,V,rcnd] = inistate(SYS,Y,U,TOL,PRINTW) returns, besides x0, the orthogonal matrix V which reduces the system state matrix A to a real Schur form, as well as an estimate of the reciprocal condition number of the coefficient matrix of the least squares problem solved.

# See Also

findBD , findx0BD

# Name

invsyslin — system inversion

```
[sl2]=invsyslin(sl1)
```

# Parameters

sl1,sl2
: `syslin` lists (linear systems in state space representation)

# Description

Utility function. Computes the state form of the inverse `sl2` of the linear system `sl1` (which is also given in state form).

The `D`-matrix is supposed to be full rank. Old stuff used by `inv(S)` when `S` is a `syslin` list.

# See Also

rowregul , inv

# Name

kpure — continuous SISO system limit feedback gain

```
K=kpure(sys [,tol])
[K,R]=kpure(sys [,tol])
```

# Parameters

sys
: SISO linear system (syslin)

tol
: vector with 2 elements `[epsK epsI]`. `epsK` is a tolerance used to determine if two values of `K` can be considered as equal `epsI` is a tolerance used to determine if a root is imaginary or not. The default value is `[1e-6 1e-6 ]`

K
: Real vector, the vector of gains for which at least one closed loop pole is imaginary.

R
: Complex vector, the imaginary closed loop poles associated with the values of `K`.

# Description

`K=kpure(sys)` computes the gains `K` such that the system `sys` feedback by `K(i)` (`sys/.K(i)`) has poles on imaginary axis.

# Examples

```
s=poly(0,'s');
h=syslin('c',(s-1)/(1+5*s+s^2+s^3))
clf();evans(h)
K=kpure(h)
hf=h/.K(1)
roots(denom(hf))
```

# See Also

evans , krac2

# Name

krac2 — continuous SISO system limit feedback gain

```
g=krac2(sys)
```

# Parameters

sys
: SISO linear system (syslin)

g
: constant

# Description

`krac2(sys)` computes the gains `g` such that the system `sys` fedback by g (`sys/.g`) has 2 real equal poles.

# Examples

```
h=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));
clf();evans(h,100)
g=krac2(h)
hf1=h/.g(1);roots(denom(hf1))
hf2=h/.g(2);roots(denom(hf2))
```

# See Also

evans , kpure

# Name

lcf — normalized coprime factorization

```
[N,M]=lcf(sl)
```

# Parameters

sl
: linear system given in state space or transfer function (`syslin` list)

N,M
: two linear systems (`syslin` list)

# Description

Computes normalized coprime factorization of the linear dynamic system `sl`.

```
sl = M^-1 N
```

# Authors

F. D.; ;

# Name

leqr — H-infinity LQ gain (full state)

```
[K,X,err]=leqr(P12,Vx)
```

# Parameters

P12
: `syslin` list

Vx
: symmetric nonnegative matrix (should be small enough)

K,X
: two real matrices

err
: a real number (l1 norm of LHS of Riccati equation)

# Description

`leqr` computes the linear suboptimal H-infinity LQ full-state gain for the plant `P12=[A,B2,C1,D12]` in continuous or discrete time.

`P12` is a `syslin` list (e.g. `P12=syslin('c',A,B2,C1,D12)`).

```
[C1' ]                [Q  S]
[    ]  * [C1 D12] = [    ]
[D12']                [S' R]
```

`Vx` is related to the variance matrix of the noise `w` perturbing `x`; (usually `Vx=gama^-2*B1*B1'`).

The gain `K` is such that `A + B2*K` is stable.

`X` is the stabilizing solution of the Riccati equation.

For a continuous plant:

```
(A-B2*inv(R)*S')'*X+X*(A-B2*inv(R)*S')-X*(B2*inv(R)*B2'-Vx)*X+Q-S*inv(R)*S'=0
```

```
K=-inv(R)*(B2'*X+S)
```

For a discrete time plant:

```
X-(Abar'*inv((inv(X)+B2*inv(R)*B2'-Vx))*Abar+Qbar=0
```

```
K=-inv(R)*(B2'*inv(inv(X)+B2*inv(R)*B2'-Vx)*Abar+S')
```

with `Abar=A-B2*inv(R)*S'` and `Qbar=Q-S*inv(R)*S'`

The 3-blocks matrix pencils associated with these Riccati equations are:

```
             discrete                          continuous
 |I  -Vx  0|   | A    0    B2|       |I   0   0|   | A    Vx    B2|
z|0   A'  0| - |-Q    I    -S|      s|0   I   0| - |-Q   -A'   -S |
 |0   B2' 0|   | S'   0     R|       |0   0   0|   | S'   -B2'   R|
```

# See Also

lqr

# Authors

F.D.;

# Name

lft — linear fractional transformation

```
[P1]=lft(P,K)
[P1]=lft(P,r,K)
[P1,r1]=lft(P,r,Ps,rs)
```

# Parameters

P
: linear system (`syslin` list), the ``augmented'' plant, implicitly partitioned into four blocks (two input ports and two output ports).

K
: linear system (`syslin` list), the controller (possibly an ordinary gain).

r
: 1x2 row vector, dimension of `P22`

Ps
: linear system (`syslin` list), implicitly partitioned into four blocks (two input ports and two output ports).

rs
: 1x2 row vector, dimension of `Ps22`

# Description

Linear fractional transform between two standard plants `P` and `Ps` in state space form or in transfer form (`syslin` lists).

`r= size(P22) rs=size(P22s)`

`lft(P,r, K)` is the linear fractional transform between `P` and a controller `K` (`K` may be a gain or a controller in state space form or in transfer form);

`lft(P,K)` is `lft(P,r,K)` with `r`=size of `K` transpose;

`P1= P11+P12*K* (I-P22*K)^-1 *P21`

`[P1,r1]=lft(P,r,Ps,rs)` returns the generalized (2 ports) lft of `P` and `Ps`.

`P1` is the pair two-port interconnected plant and the partition of `P1` into 4 blocks in given by `r1` which is the dimension of the `22` block of `P1`.

`P` and `R` can be PSSDs i.e. may admit a polynomial `D` matrix.

# Examples

```
s=poly(0,'s');
P=[1/s, 1/(s+1); 1/(s+2),2/s]; K= 1/(s-1);
lft(P,K)
lft(P,[1,1],K)
P(1,1)+P(1,2)*K*inv(1-P(2,2)*K)*P(2,1)   //Numerically dangerous!
ss2tf(lft(tf2ss(P),tf2ss(K)))
lft(P,-1)
```

```
f=[0,0;0,1];w=P/.f; w(1,1)
//Improper plant (PID control)
W=[1,1;1,1/(s^2+0.1*s)];K=1+1/s+s
lft(W,[1,1],K); ss2tf(lft(tf2ss(W),[1,1],tf2ss(K)))
```

# See Also

sensi , augment , feedback , sysdiag

# Name

lin — linearization

```
[A,B,C,D]=lin(sim,x0,u0)
[sl]=lin(sim,x0,u0)
```

# Parameters

sim
: function

x0, u0
: vectors of compatible dimensions

A,B,C,D
: real matrices

sl
: `syslin` list

# Description

linearization of the non-linear system `[y,xdot]=sim(x,u)` around `x0,u0`.

`sim` is a function which computes `y` and `xdot`.

The output is a linear system (`syslin` list) `sl` or the four matrices `(A,B,C,D)`

For example, if `ftz` is the function passed to ode e.g.

```
[zd]=ftz(t,z,u)
```

and if we assume that `y=x`

`[z]=ode(x0,t0,tf,list(ftz,u)` compute `x(tf)`.

If `simula` is the following function:

```
deff('[y,xd]=simula(x,u)','xd=ftz(tf,x,u); y=x;');
```

the tangent linear system `sl` can be obtained by:

```
[A,B,C,D]=lin(simula,z,u)
sl = syslin('c',A,B,C,D,x0)
```

# Examples

```
deff('[y,xdot]=sim(x,u)','xdot=[u*sin(x);-u*x^2];y=xdot(1)+xdot(2)')
sl=lin(sim,1,2);
```

## See Also

external , derivat

# Name

linf — infinity norm

```
linf(g [,eps],[tol])
```

# Parameters

g
: is a `syslin` linear system.

eps
: is error tolerance on n.

tol
: threshold for imaginary axis poles.

# Description

returns the L_infinity norm of `g`.

```
n=sup [sigmax(g(jw)]
   w
```

(sigmax largest singular value).

# See Also

h_norm , linfn

# Name

linfn — infinity norm

```
[x,freq]=linfn(G,PREC,RELTOL,options);
```

# Parameters

G
: is a `syslin` list

PREC
: desired relative accuracy on the norm

RELTOL
: relative threshold to decide when an eigenvalue can be considered on the imaginary axis.

options
: available options are `'trace'` or `'cond'`

x
: is the computed norm.

freq
: vector

# Description

Computes the Linf (or Hinf) norm of `G` This norm is well-defined as soon as the realization `G=(A,B,C,D)` has no imaginary eigenvalue which is both controllable and observable.

`freq` is a list of the frequencies for which `||G||` is attained,i.e., such that `||G (j om)|| = ||G||`.

If -1 is in the list, the norm is attained at infinity.

If -2 is in the list, `G` is all-pass in some direction so that `||G (j omega)|| = ||G||` for all frequencies omega.

The algorithm follows the paper by G. Robel (AC-34 pp. 882-884, 1989). The case `D=0` is not treated separately due to superior accuracy of the general method when `(A,B,C)` is nearly non minimal.

The `'trace'` option traces each bisection step, i.e., displays the lower and upper bounds and the current test point.

The `'cond'` option estimates a confidence index on the computed value and issues a warning if computations are ill-conditioned

In the general case (`A` neither stable nor anti-stable), no upper bound is prespecified.

If by contrast `A` is stable or anti stable, lower and upper bounds are computed using the associated Lyapunov solutions.

# See Also

h_norm

# Authors

P. Gahinet

# Name

linmeq — Sylvester and Lyapunov equations solver

```
[X(,sep)] = linmeq(task,A,(B,)C,flag,trans(,schur))
```

# Parameters

task
: integer option to determine the equation type:

  =1
  : solve the Sylvester equation (1a) or (1b);

  =2
  : solve the Lyapunov equation (2a) or (2b);

  =3
  : solve for the Cholesky factor op(X) the Lyapunov equation (3a) or (3b).

A
: real matrix

B
: real matrix

C
: real matrix

flag
: (optional) integer vector of length 3 or 2 containing options.

  task
  : = 1 : flag has length 3

    flag(1)
    : = 0 : solve the continuous-time equation (1a); otherwise, solve the discrete-time equation (1b).

    flag(2)
    : = 1 : A is (quasi) upper triangular;

    flag(2)
    : = 2 : A is upper Hessenberg;

    otherwise
    : A is in general form.

    flag(3)
    : = 1 : B is (quasi) upper triangular;

    flag(3)
    : = 2 : B is upper Hessenberg;

    otherwise,
    : B is in general form.

  task
  : = 2 : flag has length 2

flag(1)
if 0 solve continuous-time equation (2a), otherwise, solve discrete-time equation (2b).

flag(2)
= 1 : A is (quasi) upper triangular otherwise, A is in general form.

task
= 3 : flag has length 2

flag(1)
= 0 : solve continuous-time equation (3a); otherwise, solve discrete-time equation (3b).

flag(2)
= 1 : A is (quasi) upper triangular; otherwise, A is in general form.

Default: flag(1) = 0, flag(2) = 0 (, flag(3) = 0).

trans
(optional) integer specifying a transposition option.

=
0 : solve the equations (1) - (3) with op(M) = M.

=
1 : solve the equations (1) - (3) with op(M) = M'.

=
2 : solve the equations (1) with op(A) = A'; op(B) = B;

=
3 : solve the equations (1) with op(A) = A; op(B) = B'.

Default: trans = 0.

schur
(optional) integer specifying whether the Hessenberg-Schur or Schur method should be used. Available for task = 1.

= 1 : Hessenberg-Schur method (one matrix is reduced
to Schur form).

= 2 : Schur method (two matrices are reduced to Schur
form).

Default: schur = 1.

X

sep
(optional) estimator of Sep(op(A),-op(A)') for (2.a) or Sepd(A,A') for (2.b).

# Description

linmeq function for solving Sylvester and Lyapunov equations using SLICOT routines SB04MD, SB04ND, SB04PD, SB04QD, SB04RD, SB03MD, and SB03OD.

```
    [X] = linmeq(1,A,B,C,flag,trans,schur)
[X,sep] = linmeq(2,A,C,flag,trans)
    [X] = linmeq(2,A,C,flag,trans)
    [X] = linmeq(3,A,C,flag,trans)
```

linmeq solves various Sylvester and Lyapunov matrix equations:

```
op(A)*X + X*op(B) = C,                              (1a)

op(A)*X*op(B) + X = C,                              (1b)

op(A)'*X + X*op(A) = C,                             (2a)

op(A)'*X*op(A) - X = C,                             (2b)

op(A)'*(op(X)'*op(X)) + (op(X)'*op(X))*op(A) =
                      -  op(C)'*op(C),              (3a)

op(A)'*(op(X)'*op(X))*op(A) - op(X)'*op(X) =
                            - op(C)'*op(C),         (3b)
```

where op(M) = M, or M'.

# Comments

1. For equation (1a) or (1b), when schur = 1, the Hessenberg-Schur method is used, reducing one matrix to Hessenberg form and the other one to a real Schur form. Otherwise, both matrices are reduced to real Schur forms. If one or both matrices are already reduced to Schur/Hessenberg forms, this could be specified by flag(2) and flag(3). For general matrices, the Hessenberg-Schur method could be significantly more efficient than the Schur method.

2. For equation (2a) or (2b), matrix C is assumed symmetric.

3. For equation (3a) or (3b), matrix A must be stable or convergent, respectively.

4. For equation (3a) or (3b), the computed matrix X is the Cholesky factor of the solution, i.e., the real solution is op(X)'*op(X), where X is an upper triangular matrix.

# Revisions

V. Sima, Katholieke Univ. Leuven, Belgium, May 1999, May, Sep. 2000. V. Sima, University of Bucharest, Romania, May 2000.

# Examples

```
//(1a)
n=40;m=30;
A=rand(n,n);C=rand(n,m);B=rand(m,m);
X = linmeq(1,A,B,C);
norm(A*X+X*B-C,1)
//(1b)
flag=[1,0,0]
X = linmeq(1,A,B,C,flag);
norm(A*X*B+X-C,1)
```

```
//(2a)
A=rand(n,n);C=rand(A);C=C+C';
X = linmeq(2,A,C);
norm(A'*X + X*A -C,1)
//(2b)
X = linmeq(2,A,C,[1 0]);
norm(A'*X*A -X-C,1)
//(3a)
A=rand(n,n);
A=A-(max(real(spec(A)))+1)*eye(); //shift eigenvalues
C=rand(A);
X=linmeq(3,A,C);
norm(A'*X'*X+X'*X*A +C'*C,1)
//(3b)
A = [-0.02, 0.02,-0.10, 0.02,-0.03, 0.12;
      0.02, 0.14, 0.12,-0.10,-0.02,-0.14;
     -0.10, 0.12, 0.05, 0.03,-0.04,-0.04;
      0.02,-0.10, 0.03,-0.06, 0.08, 0.11;
     -0.03,-0.02,-0.04, 0.08, 0.14,-0.07;
      0.12,-0.14,-0.04, 0.11,-0.07, 0.04]

C=rand(A);
X=linmeq(3,A,C,[1 0]);
norm(A'*X'*X*A - X'*X +C'*C,1)
```

# See Also

sylv , lyap

# Authors

H. Xu, TU Chemnitz, FR Germany, Dec. 1998.

# Name

lqe — linear quadratic estimator (Kalman Filter)

```
[K,X]=lqe(P21)
```

# Parameters

P21
: `syslin` list

K, X
: real matrices

# Description

`lqe` returns the Kalman gain for the filtering problem in continuous or discrete time.

`P21` is a `syslin` list representing the system `P21=[A,B1,C2,D21]` `P21=syslin('c',A,B1,C2,D21) or P21=syslin('d',A,B1,C2,D21)`

The input to `P21` is a white noise with variance:

```
     [B1 ]                [Q  S]
BigV=[   ] [ B1' D21'] = [    ]
     [D21]                [S' R]
```

`X` is the solution of the stabilizing Riccati equation and `A+K*C2` is stable.

In continuous time:

```
(A-S*inv(R)*C2)*X+X*(A-S*inv(R)*C2)'-X*C2'*inv(R)*C2*X+Q-S*inv(R)*S'=0
```

```
K=-(X*C2'+S)*inv(R)
```

In discrete time:

```
X=A*X*A'-(A*X*C2'+B1*D21')*pinv(C2*X*C2'+D21*D21')*(C2*X*A'+D21*B1')+B1*B1'
```

`K=-(A*X*C2'+B1*D21')*pinv(C2*X*C2'+D21*D21')`

`xhat(t+1)= E(x(t+1)| y(0),...,y(t))` (one-step predicted `x`) satisfies the recursion:

```
xhat(t+1)=(A+K*C2)*xhat(t) - K*y(t).
```

# Examples

```
//Assume the equations
```

```
//.
//x = Ax + Ge
//y = Cx + v
//with
//E ee' = Q_e,    Evv' = R,    Eev' = N
//
//This is equivalent to
//.
//x = Ax  + B1 w
//y = C2x + D21 w
//with E { [Ge ]  [Ge v]' } = E { [B1w ] [B1w D21w]' } = bigR =
//         [ v ]                  [D21w]
//
//[B1*B1'  B1*D21';
// D21*B1'  D21*D21']
//=
//[G*Q_e*G' G*N;
// N*G' R]

//To find (B1,D21) given (G,Q_e,R,N) form bigR =[G*Q_e*G' G*N;N'*G' R].
//Then [W,Wt]=fullrf(bigR);  B1=W(1:size(G,1),:);
//D21=W(($+1-size(C2,1)):$,:)
//
//P21=syslin('c',A,B1,C2,D21);
//[K,X]=lqe(P21);

//Example:
nx=5;ne=2;ny=3;
A=-diag(1:nx);G=ones(nx,ne);
C=ones(ny,nx); Q_e(ne,ne)=1; R=diag(1:ny); N=zeros(ne,ny);
bigR =[G*Q_e*G' G*N;N'*G' R];
[W,Wt]=fullrf(bigR);B1=W(1:size(G,1),:);
D21=W(($+1-size(C,1)):$,:);
C2=C;
P21=syslin('c',A,B1,C2,D21);
[K,X]=lqe(P21);
//Riccati check:
S=G*N;Q=B1*B1';
(A-S*inv(R)*C2)*X+X*(A-S*inv(R)*C2)'-X*C2'*inv(R)*C2*X+Q-S*inv(R)*S'

//Stability check:
spec(A+K*C)
```

# See Also

lqr , observer

# Authors

F. D.

# Name

lqg — LQG compensator

```
[K]=lqg(P,r)
```

# Parameters

P
: `syslin` list (augmented plant) in state-space form

r
: 1x2 row vector = (number of measurements, number of inputs) (dimension of the 2,2 part of `P`)

K
: `syslin` list (controller)

# Description

`lqg` computes the linear optimal LQG (H2) controller for the "augmented" plant `P=syslin('c',A,B,C,D)` (continuous time) or `P=syslin('d',A,B,C,D)` (discrete time).

The function `lqg2stan` returns `P` and `r` given the nominal plant, weighting terms and variances of noises.

`K` is given by the following ABCD matrices: `[A+B*Kc+Kf*C+Kf*D*Kc,-Kf,Kc,0]` where `Kc=lqr(P12)` is the controller gain and `Kf=lqe(P21)` is the filter gain. See example in `lqg2stan`.

# See Also

lqg2stan , lqr , lqe , h_inf , obscont

# Authors

F.D.

# Name

lqg2stan — LQG to standard problem

```
[P,r]=lqg2stan(P22,bigQ,bigR)
```

# Parameters

P22
: `syslin` list (nominal plant) in state-space form

bigQ
: `[Q,S;S',N]` (symmetric) weighting matrix

bigR
: `[R,T;T',V]` (symmetric) covariance matrix

r
: `1x2` row vector = (number of measurements, number of inputs) (dimension of the 2,2 part of `P`)

P
: `syslin` list (augmented plant)

# Description

`lqg2stan` returns the augmented plant for linear LQG (H2) controller design.

`P22=syslin(dom,A,B2,C2)` is the nominal plant; it can be in continuous time (`dom='c'`) or discrete time (`dom='d'`).

```
.
x = Ax + w1 + B2u
y = C2x + w2
```

for continuous time plant.

```
x[n+1]= Ax[n] + w1 + B2u
    y = C2x + w2
```

for discrete time plant.

The (instantaneous) cost function is `[x' u'] bigQ [x;u]`.

The covariance of `[w1;w2]` is `E[w1;w2] [w1',w2'] = bigR`

If `[B1;D21]` is a factor of `bigQ`, `[C1,D12]` is a factor of `bigR` and `[A,B2,C2,D22]` is a realization of P22, then `P` is a realization of `[A,[B1,B2],[C1,-C2],[0,D12;D21,D22]`. The (negative) feedback computed by `lqg` stabilizes `P22`, i.e. the poles of `cl=P22/.K` are stable.

# Examples

```
ny=2;nu=3;nx=4;
P22=ssrand(ny,nu,nx);
bigQ=rand(nx+nu,nx+nu);bigQ=bigQ*bigQ';
```

```
bigR=rand(nx+ny,nx+ny);bigR=bigR*bigR';
[P,r]=lqg2stan(P22,bigQ,bigR);K=lqg(P,r);  //K=LQG-controller
spec(h_cl(P,r,K))      //Closed loop should be stable
//Same as Cl=P22/.K; spec(Cl('A'))
s=poly(0,'s')
lqg2stan(1/(s+2),eye(2,2),eye(2,2))
```

# See Also

lqg , lqr , lqe , obscont , h_inf , augment , fstabst , feedback

# Authors

F.D.

# Name

lqg_ltr — LQG with loop transform recovery

```
[kf,kc]=lqg_ltr(sl,mu,ro)
```

# Parameters

sl
: linear system in state-space form (`syslin` list)

mu,ro
: real positive numbers chosen ``small enough''

kf,kc
: controller and observer Kalman gains.

# Description

returns the Kalman gains for:

```
         x = a*x + b*u + l*w1
(sl)
         y = c*x + mu*I*w2

         z = h*x
```

Cost function:

```
         /+oo
         |
J    = E(| [z(t)'*z(t) + ro^2*u(t)'*u(t)]dt)
 lqg     |
         / 0
```

The lqg/ltr approach looks for `L,mu,H,ro` such that: J(lqg) = J(freq) where

```
        /+oo        *  *           *
J    =  | tr[S  W  W  S ] + tr[T  T]dw
 freq   |
        /0
```

and

```
 S = (I + G*K)^(-1)
 T = G*K*(I+G*K)^(-1)
```

# See Also

syslin

# Name

lqr — LQ compensator (full state)

```
[K,X]=lqr(P12)
```

# Parameters

P12
: `syslin` list (state-space linear system)

K,X
: two real matrices

# Description

`lqr` computes the linear optimal LQ full-state gain for the plant `P12=[A,B2,C1,D12]` in continuous or discrete time.

`P12` is a `syslin` list (e.g. `P12=syslin('c',A,B2,C1,D12)`).

The cost function is l2-norm of `z'*z` with `z=C1 x + D12 u` i.e. `[x,u]' * BigQ * [x;u]` where

```
      [C1' ]                [Q  S]
BigQ= [    ]  * [C1 D12] = [    ]
      [D12']                [S' R]
```

The gain `K` is such that `A + B2*K` is stable.

`X` is the stabilizing solution of the Riccati equation.

For a continuous plant:

```
(A-B2*inv(R)*S')'*X+X*(A-B2*inv(R)*S')-X*B2*inv(R)*B2'*X+Q-S*inv(R)*S'=0
```

```
K=-inv(R)*(B2'*X+S)
```

For a discrete plant:

```
X=A'*X*A-(A'*X*B2+C1'*D12)*pinv(B2'*X*B2+D12'*D12)*(B2'*X*A+D12'*C1)+C1'*C1;
```

```
K=-pinv(B2'*X*B2+D12'*D12)*(B2'*X*A+D12'*C1)
```

An equivalent form for `X` is

```
X=Abar'*inv(inv(X)+B2*inv(r)*B2')*Abar+Qbar
```

with `Abar=A-B2*inv(R)*S'` and `Qbar=Q-S*inv(R)*S'`

The 3-blocks matrix pencils associated with these Riccati equations are:

```
             discrete                            continuous
 |I   0    0|   | A    0    B2|         |I   0   0|   | A    0    B2|
z|0   A'   0| - |-Q    I    -S|        s|0   I   0| - |-Q   -A'   -S|
 |0   B2'  0|   | S'   0     R|         |0   0   0|   | S'  -B2'   R|
```

Caution: It is assumed that matrix R is non singular. In particular, the plant must be tall (number of outputs >= number of inputs).

# Examples

```
A=rand(2,2);B=rand(2,1);   //two states, one input
Q=diag([2,5]);R=2;     //Usual notations x'Qx + u'Ru
Big=sysdiag(Q,R);    //Now we calculate C1 and D12
[w,wp]=fullrf(Big);C1=wp(:,1:2);D12=wp(:,3:$);   //[C1,D12]'*[C1,D12]=Big
P=syslin('c',A,B,C1,D12);    //The plant (continuous-time)
[K,X]=lqr(P)
spec(A+B*K)    //check stability
norm(A'*X+X*A-X*B*inv(R)*B'*X+Q,1)  //Riccati check
P=syslin('d',A,B,C1,D12);    // Discrete time plant
[K,X]=lqr(P)
spec(A+B*K)   //check stability
norm(A'*X*A-(A'*X*B)*pinv(B'*X*B+R)*(B'*X*A)+Q-X,1) //Riccati check
```

# See Also

lqe , gcare , leqr

# Authors

F.D.;

# Name

ltitr — discrete time response (state space)

```
[X]=ltitr(A,B,U,[x0])
[xf,X]=ltitr(A,B,U,[x0])
```

# Parameters

A,B
: real matrices of appropriate dimensions

U,X
: real matrices

x0,xf
: real vectors (default value=0 for `x0`))

# Description

calculates the time response of the discrete time system

```
x[t+1] = Ax[t] + Bu[t].
```

The inputs `ui`'s are the columns of the `U` matrix

```
U=[u0,u1,...,un];
```

`x0` is the vector of initial state (default value : 0) ;

`X` is the matrix of outputs (same number of columns as `U`).

```
X=[x0,x1,x2,...,xn]
```

`xf` is the vector of final state `xf=X[n+1]`

# Examples

```
A=eye(2,2);B=[1;1];
x0=[-1;-2];
u=[1,2,3,4,5];
x=ltitr(A,B,u,x0)
x1=A*x0+B*u(1)
x2=A*x1+B*u(2)
x3=A*x2+B*u(3) //....
```

# See Also

rtitr , flts

# Name

macglov — Mac Farlane Glover problem

```
[P,r]=macglov(Sl)
```

# Parameters

Sl
: linear system (`syslin` list)

P
: linear system (`syslin` list), ``augmented'' plant

r
: 1x2 vector, dimension of `P22`

# Description

`[P,r]=macglov(Sl)` returns the standard plant `P` for the Glover-McFarlane problem.

For this problem ro_optimal = 1-hankel_norm(`[N,M]`) with `[N,M]=lcf(sl)` (Normalized coprime factorization) i.e.

gama_optimal = `1/sqrt(ro_optimal)`

# Authors

F. Delebecque INRIA

# Name

markp2ss — Markov parameters to state-space

```
[sl]=markp2ss(markpar,n,nout,nin)
```

# Parameters

markpar
: matrix

n,nout,nin
: integers

Sl
: `syslin` list

# Description

given a set of `n` Markov parameters stacked in the (row)-matrix `markpar` of size `noutX(n*nin)` `markp2ss` returns a state-space linear system `sl` (`syslin` list) such that with `[A,B,C,D]=abcd(sl)`:

```
C*B = markpar(1:nout,1:nin),
C*A*B =markpar(1:nout,nin+1:2*nin),....
```

# Examples

```
W=ssrand(2,3,4);   //random system with 2 outputs and 3 inputs
[a,b,c,d]=abcd(W);
markpar=[c*b,c*a*b,c*a^2*b,c*a^3*b,c*a^4*b];
S=markp2ss(markpar,5,2,3);
[A,B,C,D]=abcd(S);
Markpar=[C*B,C*A*B,C*A^2*B,C*A^3*B,C*A^4*B];
norm(markpar-Markpar,1)
//Caution... c*a^5*b is not C*A^5*B !
```

# See Also

frep2tf , tf2ss , imrep2ss

# Name

minreal — minimal balanced realization

```
slb=minreal(sl [,tol])
```

# Parameters

sl,slb
: `syslin` lists

tol
: real (threshold)

# Description

`[ae,be,ce]=minreal(a,b,c,domain [,tol])` returns the balanced realization of linear system `sl` (`syslin` list).

`sl` is assumed stable.

`tol` threshold used in `equil1`.

# Examples

```
A=[-eye(2,2),rand(2,2);zeros(2,2),-2*eye(2,2)];
B=[rand(2,2);zeros(2,2)];C=rand(2,4);
sl=syslin('c',A,B,C);
slb=minreal(sl);
ss2tf(sl)
ss2tf(slb)
ctr_gram(sl)
clean(ctr_gram(slb))
clean(obs_gram(slb))
```

# See Also

minss , balreal , arhnk , equil , equil1

# Authors

S. Steer INRIA 1987

# Name

minss — minimal realization

```
[slc]=minss( sl [,tol])
```

# Parameters

sl,slc
: `syslin` lists (linear systems in state-space form)

tol
: real (threshold for rank determination (see `contr`))

# Description

`minss` returns in `slc` a minimal realization of `sl`.

# Examples

```
sl=syslin('c',[1 0;0 2],[1;0],[2 1]);
ssprint(sl);
ssprint(minss(sl))
```

# See Also

contr , minreal , arhnk , contrss , obsvss , balreal

# Name

mucomp — mu (structured singular value) calculation

```
[BOUND, D, G] = mucomp(Z, K, T)
```

# Parameters

Z
: the complex n-by-n matrix for which the structured singular value is to be computed

K
: the vector of length m containing the block structure of the uncertainty.

T
: the vector of length m indicating the type of each block. T(I) = 1 if the corresponding block is real T(I) = 2 if the corresponding block is complex.

BOUND
: the upper bound on the structured singular value.

D, G
: vectors of length n containing the diagonal entries of the diagonal matrices D and G, respectively, such that the matrix `Z'*D^2*Z + sqrt(-1)*(G*Z-Z'*G) - bound^2*D^2` is negative semidefinite.

# Description

To compute an upper bound on the structured singular value for a given square complex matrix and given block structure of the uncertainty.

# Reference

Slicot routine AB13MD.

# Name

narsimul — armax simulation ( using rtitr)

```
[z]=narsimul(a,b,d,sig,u,[up,yp,ep])
[z]=narsimul(ar,u,[up,yp,ep])
```

# Description

ARMAX simulation. Same as arsimul but the method is different the simulation is made with rtitr

# Authors

J-Ph. Chancelier ENPC Cergrene; ;

# Name

nehari — Nehari approximant

```
[x]=nehari(R [,tol])
```

# Parameters

R
: linear system (`syslin` list)

x
: linear system (`syslin` list)

tol
: optional threshold

# Description

`[x]=nehari(R [,tol])returns` the Nehari approximant of `R`.

`R` = linear system in state-space representation (`syslin` list).

`R` is strictly proper and `- R~` is stable (i.e. `R` is anti stable).

```
|| R - X ||oo = min || R - Y ||oo
               Y in Hoo
```

# Name

noisegen — noise generation

```
b=noisegen(pas,Tmax,sig)
```

# Description

generates a Scilab function `[b]=Noise(t)` where `Noise(t)` is a piecewise constant function ( constant on `[k*pas,(k+1)*pas]` ). The value on each constant interval are random values from i.i.d Gaussian variables of standard deviation sig. The function is constant for `t<=0` and `t>=Tmax`.

# Examples

```
noisegen(0.5,30,1.0);
x=-5:0.01:35;
y=feval(x,Noise);
plot(x,y);
```

# Name

obs_gram — observability gramian

```
Go=obs_gram(A,C [,dom])
Go=obs_gram(sl)
```

# Parameters

A,C
: real matrices (of appropriate dimensions)

dom
: string (`"d'` or `"c"` (default value))

sl
: `syslin` list

# Description

Observability gramian of the pair `(A,C)` or linear system `sl` (`syslin` list). `dom` is the domain which can be

"c"
: continuous system (default)

"d"
: discrete system

# Examples

```
A=-diag(1:3);C=rand(2,3);
Go=obs_gram(A,C,'c');     // <=> w=syslin('c',A,[],C); Go=obs_gram(w);
norm(Go*A+A'*Go+C'*C,1)
norm(lyap(A,-C'*C,'c')-Go,1)
A=A/4; Go=obs_gram(A,C,'d');    //discrete time case
norm(lyap(A,-C'*C,'d')-Go,1)
```

# See Also

ctr_gram , obsvss , obsv_mat , lyap

# Name

obscont — observer based controller

```
[K]=obscont(P,Kc,Kf)
[J,r]=obscont(P,Kc,Kf)
```

# Parameters

P
: `syslin` list (nominal plant) in state-space form, continuous or discrete time

Kc
: real matrix, (full state) controller gain

Kf
: real matrix, filter gain

K
: `syslin` list (controller)

J
: `syslin` list (extended controller)

r
: 1x2 row vector

# Description

`obscont` returns the observer-based controller associated with a nominal plant `P` with matrices `[A,B,C,D]` (`syslin` list).

The full-state control gain is `Kc` and the filter gain is `Kf`. These gains can be computed, for example, by pole placement.

`A+B*Kc` and `A+Kf*C` are (usually) assumed stable.

`K` is a state-space representation of the compensator `K: y->u` in:

`xdot = A x + B u, y=C x + D u, zdot= (A + Kf C)z -Kf y +B u, u=Kc z`

`K` is a linear system (`syslin` list) with matrices given by: `K=[A+B*Kc+Kf*C+Kf*D*Kc,Kf,-Kc]`.

The closed loop feedback system `Cl: v ->y` with (negative) feedback `K` (i.e. `y = P u, u = v - K y`, or

```
xdot = A x + B u,
   y = C x + D u,
zdot = (A + Kf C) z - Kf y + B u,
   u = v -F z
```

) is given by `Cl = P/.(-K)`

The poles of `Cl( spec(cl('A')) )` are located at the eigenvalues of `A+B*Kc` and `A+Kf*C`.

Invoked with two output arguments `obscont` returns a (square) linear system `K` which parametrizes all the stabilizing feedbacks via a LFT.

Let `Q` an arbitrary stable linear system of dimension `r(2)`x`r(1)` i.e. number of inputs x number of outputs in `P`. Then any stabilizing controller `K` for `P` can be expressed as `K=lft(J,r,Q)`. The controller which corresponds to `Q=0` is `K=J(1:nu,1:ny)` (this `K` is returned by `K=obscont(P,Kc,Kf)`). `r` is `size(P)` i.e the vector [number of outputs, number of inputs];

# Examples

```
ny=2;nu=3;nx=4;P=ssrand(ny,nu,nx);[A,B,C,D]=abcd(P);
Kc=-ppol(A,B,[-1,-1,-1,-1]);  //Controller gain
Kf=-ppol(A',C',[-2,-2,-2,-2]);Kf=Kf';    //Observer gain
cl=P/.(-obscont(P,Kc,Kf));spec(cl('A'))   //closed loop system
[J,r]=obscont(P,Kc,Kf);
Q=ssrand(nu,ny,3);Q('A')=Q('A')-(maxi(real(spec(Q('A'))))+0.5)*eye(Q('A'))
//Q is a stable parameter
K=lft(J,r,Q);
spec(h_cl(P,K))  // closed-loop A matrix (should be stable);
```

# See Also

ppol , lqg , lqr , lqe , h_inf , lft , syslin , feedback , observer

# Authors

F.D. ; ;

# Name

observer — observer design

```
Obs=observer(Sys,J)
[Obs,U,m]=observer(Sys [,flag,alfa])
```

# Parameters

Sys
: `syslin` list (linear system)

J
: nx x ny constant matrix (output injection matrix)

flag
: character strings (`'pp'` or `'st'` (default))

alfa
: location of closed-loop poles (optional parameter, default=-1)

Obs
: linear system (`syslin` list), the observer

U
: orthogonal matrix (see `dt_ility`)

m
: integer (dimension of unstable unobservable (`st`) or unobservable (`pp`) subspace)

# Description

`Obs=observer(Sys,J)` returns the observer `Obs=syslin(td,A+J*C,[B+J*D,-J],eye(A))` obtained from `Sys` by a `J` output injection. (td is the time domain of `Sys`). More generally, `observer` returns in `Obs` an observer for the observable part of linear system `Sys`: `dotx=A x + Bu, y=Cx + Du` represented by a `syslin` list. `Sys` has `nx` state variables, `nu` inputs and `ny` outputs. `Obs` is a linear system with matrices `[Ao,Bo,Identity]`, where `Ao` is `no x no`, `Bo` is `no x (nu+ny)`, `Co` is `no x no` and `no=nx-m`.

Input to `Obs` is `[u,y]` and output of `Obs` is:

xhat=estimate of x modulo unobservable subsp. (case `flag='pp'`) or

xhat=estimate of x modulo unstable unobservable subsp. (case `flag='st'`)

case `flag='st'`: z=H*x can be estimated with stable observer iff `H*U(:,1:m)=0` and assignable poles of the observer are set to `alfa(1),alfa(2),...`

case `flag='pp'`: z=H*x can be estimated with given error spectrum iff `H*U(:,1:m)=0` all poles of the observer are assigned and set to `alfa(1),alfa(2),...`

If H satifies the constraint: `H*U(:,1:m)=0` (ker(H) contains unobs-subsp. of Sys) one has H*U=[0,H2] and the observer for z=H*x is H2*Obs with H2=H*U(:,m+1:nx) i.e. Co, the C-matrix of the observer for H*x, is Co=H2.

In the particular case where the pair `(A,C)` of `Sys` is observable, one has `m=0` and the linear system `U*Obs` (resp. `H*U*Obs`) is an observer for `x` (resp. `Hx`). The error spectrum is `alpha(1),alpha(2),...,alpha(nx)`.

# Examples

```
nx=5;nu=1;ny=1;un=3;us=2;Sys=ssrand(ny,nu,nx,list('dt',us,us,un));
//nx=5 states, nu=1 input, ny=1 output,
//un=3 unobservable states, us=2 of them unstable.
[Obs,U,m]=observer(Sys);  //Stable observer (default)
W=U';H=W(m+1:nx,:);[A,B,C,D]=abcd(Sys);  //H*U=[0,eye(no,no)];
Sys2=ss2tf(syslin('c',A,B,H))  //Transfer u-->z
Idu=eye(nu,nu);Sys3=ss2tf(H*U(:,m+1:$)*Obs*[Idu;Sys])
//Transfer u-->[u;y=Sys*u]-->Obs-->xhat-->HUxhat=zhat  i.e. u-->output of Obs
//this transfer must equal Sys2, the u-->z transfer  (H2=eye).

//Assume a Kalman model
//dotx = A x + B u + G w
// y   = C x + D u + H w + v
//with Eww' = QN, Evv' = RN, Ewv' = NN
//To build a Kalman observer:
//1-Form BigR = [G*QN*G'         G*QN*H'+G*NN;
//               H*QN*G'+NN*G'   H*QN*H'+RN];
//the covariance matrix of the noise vector [Gw;Hw+v]
//2-Build the plant P21 : dotx = A x + B1 e ; y = C2 x + D21 e
//with e a unit white noise.
// [W,Wt]=fullrf(BigR);
//B1=W(1:size(G,1),:);D21=W(($+1-size(C,1)):$,:);
//C2=C;
//P21=syslin('c',A,B1,C2,D21);
//3-Compute the Kalman gain
//L = lqe(P21);
//4- Build an observer for the plant [A,B,C,D];
//Plant = syslin('c',A,B,C,D);
//Obs = observer(Plant,L);
//Test example:
A=-diag(1:4);
B=ones(4,1);
C=B'; D= 0; G=2*B; H=-3; QN=2;
RN=5; NN=0;
BigR = [G*QN*G'         G*QN*H'+G*NN;
        H*QN*G'+NN*G'   H*QN*H'+RN];
[W,Wt]=fullrf(BigR);
B1=W(1:size(G,1),:);D21=W(($+1-size(C,1)):$,:);
C2=C;
P21=syslin('c',A,B1,C2,D21);
L = lqe(P21);
Plant = syslin('c',A,B,C,D);
Obs = observer(Plant,L);
spec(Obs.A)
```

# See Also

dt_ility , unobs , stabil

# Authors

F.D.

# Name

obsv_mat — observability matrix

```
[O]=obsv_mat(A,C)
[O]=obsv_mat(sl)
```

# Parameters

A,C,O
: real matrices

sl
: `syslin` list

# Description

`obsv_mat` returns the observability matrix:

```
O=[C; CA; CA^2;...; CA^(n-1) ]
```

# See Also

contrss , obsvss , obs_gram

# Name

obsvss — observable part

```
[Ao,Bo,Co]=obsvss(A,B,C [,tol])
[slo]=obsvss(sl [,tol])
```

# Parameters

A,B,C,Ao,Bo,Co
: real matrices

sl,slo
: `syslin` lists

tol
: real (threshold) (default value `100*%eps`)

# Description

`slo=(Ao,Bo,Co)` is the observable part of linear system `sl=(A,B,C)` (`syslin` list)

`tol` threshold to test controllability (see `contr`); default value = `100*%eps`

# See Also

contr , contrss , obsv_mat , obs_gram

# Name

p_margin — phase margin and associated crossover frequency

```
[phm,fr] = p_margin(h)
phm=p_margin(h)
```

# Parameters

h
: a SISO linear system (see :syslin).

phm
: a number, the phase margin in degree if it exists or an empty matrix.

fr
: a number, the corresponding frequency ( in hz) or an empty matrix.

# Description

Given a SISO linear system in continuous or discrete time, `p_margin` returns `phm`, the phase margin in degree of `h` and `fr`, the achieved corresponding frequency in hz.

The phase margin is the values of the phase at frequency points where the nyquist plot of `h` crosses the unit circle. In other words the phase margin is the difference between the phase of the frequency response of `h` and -180° when the gain of `h` is 1.

The algorithm uses polynomial root finder to solve the equations:

h(s)*h(-s)=1
: for the continuous time case.

h(z)*h(1/z)=1
: for the discrete time case.

# Examples

```
//continuous case
h=syslin('c',-1+%s,3+2*%s+%s^2)
[p,fr]=p_margin(h)
[p,fr]=p_margin(h+0.7)
show_margins(h+0.7,'nyquist')

//discrete case
h = syslin(0.1,0.04798*%z+0.0464,%z^2-1.81*%z+0.9048);//ok
[p ,f]=p_margin(h)
show_margins(h,'nyquist')
```

# See Also

p_margin, show_margins, repfreq, black, bode, chart, nyquist

# Authors

Serge Steer, INRIA

# Name

parrot — Parrot's problem

```
K=parrot(D,r)
```

# Parameters

D,K
: matrices

r
: 1X2 vector (dimension of the 2,2 part of `D`)

# Description

Given a matrix `D` partionned as `[D11 D12; D21 D22]` where `size(D22)=r=[r1,r2]` compute a matrix `K` such that largest singular value of `[D11 D12; D21 D22+K]` is minimal (Parrot's problem)

# See Also

h_inf_st

# Name

pfss — partial fraction decomposition

```
elts=pfss(Sl)
elts=pfss(Sl,rmax)
elts=pfss(Sl,'cord')
elts=pfss(Sl,rmax,'cord')
```

# Parameters

Sl

`syslin` list (state-space or transfer linear system) rmax : real number controlling the conditioning of block diagoanalization cord : character string `'c'` or `'d'`.

# Description

Partial fraction decomposition of the linear system `Sl` (in state-space form, transfer matrices are automatically converted to state-space form by `tf2ss`):

`elts` is the list of linear systems which add up to `Sl` i.e. `elts=list(S1,S2,S3,...,Sn)` with:

`Sl = S1 + S2 +... +Sn`.

Each `Si` contains some poles of S according to the block-diagonalization of the `A` matrix of `S`.

For non proper systems the polynomial part of `Sl` is put in the last entry of `elts`.

If `Sl` is given in transfer form, it is first converted into state-space and each subsystem `Si` is then converted in transfer form.

The A matrix is of the state-space is put into block diagonal form by function `bdiag`. The optional parameter `rmax` is sent to `bdiag`. If `rmax` should be set to a large number to enforce block-diagonalization.

If the optional flag `cord='c'` is given the elements in `elts` are sorted according to the real part (resp. magnitude if `cord='d'`) of the eigenvalues of A matrices.

# Examples

```
W=ssrand(1,1,6);
elts=pfss(W);
W1=0;for k=1:size(elts), W1=W1+ss2tf(elts(k));end
clean(ss2tf(W)-W1)
```

# See Also

pbig , bdiag , coffg , dtsi

# Authors

F.D.;

# Name

phasemag — phase and magnitude computation

```
[phi,db]=phasemag(z [,mod])
```

# Parameters

z
: matrix or row vector of complex numbers.

mod
: character string

  mod='c'
  : "continuous" representation between -infinity and +360 degrees (default)

  mod='m'
  : representation between -360 and 0 degrees

phi
: phases (in degree) of `z`.

db
: magnitude (in Db)

# Description

`phasemag` computes the phases and magnitudes of the entries of a complex matrix. For `mod='c'phasemag` computes `phi(:,i+1)` to minimize the distance with `phi(:,i)`, i.e. it tries to obtain a "continuous representation" of the phase.

To obtain the phase between -%pi and %pi use `phi=atan(imag(z),real(z))`

# Examples

```
s=poly(0,'s');
h=syslin('c',1/((s+5)*(s+10)*(100+6*s+s*s)*(s+.3)));
[frq,rf]=repfreq(h,0.1,20,0.005);
scf();
plot2d(frq',phasemag(rf,'c')');
scf();
plot2d(frq',phasemag(rf,'m')');
```

# See Also

repfreq , gainplot , atan , bode

# Name

ppol — pole placement

```
[K]=ppol(A,B,poles)
```

# Parameters

A,B
: real matrices of dimensions `nxn` and `nxm`.

poles
: real or complex vector of dimension `n`.

K
: real matrix (negative feedback gain)

# Description

`K=ppol(A,B,poles)` returns a mxn gain matrix `K` such that the eigenvalues of `A-B*K` are `poles`. The pair `(A,B)` must be controllable. Complex number in `poles` must appear in conjugate pairs.

An output-injection gain `F` for `(A,C)` is obtained as follows:

```
Ft=ppol(A',C',poles); F=Ft'
```

The algorithm is by P.H. Petkov.

# Examples

```
A=rand(3,3);B=rand(3,2);
F=ppol(A,B,[-1,-2,-3]);
spec(A-B*F)
```

# See Also

canon , stabil

# Name

prbs_a — pseudo random binary sequences generation

```
[u]=prbs_a(n,nc,[ids])
```

# Description

generation of pseudo random binary sequences `u=[u0,u1,...,u_(n-1)]` u takes values in {-1,1} and changes at most nc times its sign. ids can be used to fix the date at which u must change its sign ids is then an integer vector with values in `[1:n]`.

# Examples

```
u=prbs_a(50,10);
plot2d2("onn",(1:50)',u',1,"151",' ',[0,-1.5,50,1.5]);
```

# Name

projsl — linear system projection

```
[slp]=projsl(sl,Q,M)
```

# Parameters

sl,slp
: `syslin` lists

Q,M
: matrices (projection factorization)

# Description

`slp`= projected model of `sl` where `Q*M` is the full rank factorization of the projection.

If `(A,B,C,D)` is the representation of `sl`, the projected model is given by `(M*A*Q,M*B,C*Q,D)`.

Usually, the projection `Q*M` is obtained as the spectral projection of an appropriate auxiliary matrix `W` e.g. `W` = product of (weighted) gramians or product of Riccati equations.

# Examples

```
rand('seed',0);sl=ssrand(2,2,5);[A,B,C,D]=abcd(sl);poles=spec(A)
[Q,M]=pbig(A,0,'c');  //keeping unstable poles
slred=projsl(sl,Q,M);spec(slred('A'))
sl('D')=rand(2,2);  //making proper system
trzeros(sl)  //zeros of sl
wi=inv(sl);  //wi=inverse in state-space
[q,m]=psmall(wi('A'),2,'d');  //keeping small zeros (poles of wi) i.e. abs(z)<2
slred2=projsl(sl,q,m);
trzeros(slred2)  //zeros of slred2 = small zeros of sl
//  Example keeping second order modes
A=diag([-1,-2,-3]);
sl=syslin('c',A,rand(3,2),rand(2,3));[nk2,W]=hankelsv(sl)
[Q,M]=pbig(W,nk2(2)-%eps,'c');    //keeping 2 eigenvalues of W
slr=projsl(sl,Q,M);  //reduced model
hankelsv(slr)
```

# See Also

pbig

# Authors

F. D.;

# Name

reglin — Linear regression

```
[a,b,sig]=reglin(x,y)
```

# Description

solve the regression problem y=a*x+ b in the least square sense. `sig` is the standard deviation of the residual. `x` and `y` are two matrices of size `x(p,n)` and `y(q,n)`, where `n` is the number of samples.

The estimator `a` is a matrix of size `(q,p)` and `b` is a vector of size `(q,1)`

```
// simulation of data for a(3,5) and b(3,1)
x=rand(5,100);
aa=testmatrix('magi',5);aa=aa(1:3,:);
bb=[9;10;11]
y=aa*x +bb*ones(1,100)+ 0.1*rand(3,100);
// identification
[a,b,sig]=reglin(x,y);
maxi(abs(aa-a))
maxi(abs(bb-b))
// an other example : fitting a polynom
f=1:100; x=[f.*f; f];
y= [ 2,3]*x+ 10*ones(f) + 0.1*rand(f);
[a,b]=reglin(x,y)
```

# See Also

pinv , leastsq , qr

# Name

repfreq — frequency response

```
[ [frq,] repf]=repfreq(sys,fmin,fmax [,step])
[ [frq,] repf]=repfreq(sys [,frq])
[ frq,repf,splitf]=repfreq(sys,fmin,fmax [,step])
[ frq,repf,splitf]=repfreq(sys [,frq])
```

# Parameters

sys
: `syslin` list : SIMO linear system

fmin,fmax
: two real numbers (lower and upper frequency bounds)

frq
: real vector of frequencies (Hz)

step
: logarithmic discretization step

splitf
: vector of indexes of critical frequencies.

repf
: vector of the complex frequency response

# Description

`repfreq` returns the frequency response calculation of a linear system. If `sys(s)` is the transfer function of `Sys`, `repf(k)` equals `sys(s)` evaluated at `s= %i*frq(k)*2*%pi` for continuous time systems and at `exp(2*%i*%pi*dt*frq(k))` for discrete time systems (`dt` is the sampling period).

`db(k)` is the magnitude of `repf(k)` expressed in dB i.e. `db(k)=20*log10(abs(repf(k)))` and `phi(k)` is the phase of `repf(k)` expressed in degrees.

If `fmin,fmax,step` are input parameters, the response is calculated for the vector of frequencies `frq` given by: `frq=[10.^((log10(fmin)):step:(log10(fmax))) fmax];`

If `step` is not given, the output parameter `frq` is calculated by `frq=calfrq(sys,fmin,fmax)`.

Vector `frq` is splitted into regular parts with the `split` vector. `frq(splitf(k):splitf(k+1)-1)` has no critical frequency. `sys` has a pole in the range `[frq(splitf(k)),frq(splitf(k)+1)]` and no poles outside.

# Examples

```
A=diag([-1,-2]);B=[1;1];C=[1,1];
Sys=syslin('c',A,B,C);
frq=0:0.02:5;w=frq*2*%pi; //frq=frequencies in Hz ;w=frequencies in rad/sec;
[frq1,rep] =repfreq(Sys,frq);
[db,phi]=dbphi(rep);
Systf=ss2tf(Sys)    //Transfer function of Sys
x=horner(Systf,w(2)*sqrt(-1))    // x is Systf(s) evaluated at s = i w(2)
```

```
rep=20*log(abs(x))/log(10)   //magnitude of x in dB
db(2)    // same as rep
ang=atan(imag(x),real(x));   //in rad.
ang=ang*180/%pi              //in degrees
phi(2)
repf=repfreq(Sys,frq);
repf(2)-x
```

# See Also

bode , freq , calfrq , horner , nyquist , dbphi

# Authors

S. S.

# Name

ric_desc — Riccati equation

```
X=ric_desc(H [,E))
[X1,X2,zero]=ric_desc(H [,E])
```

# Parameters

H,E
: real square matrices

X1,X2
: real square matrices

zero
: real number

# Description

Riccati solver with hamiltonian matrices as inputs.

In the continuous time case calling sequence is `ric_descr(H)` (one input):

Riccati equation is:

```
(Ec)   A'*X + X*A + X*R*X -Q = 0.
```

Defining the hamiltonian matrix `H` by:

```
H = [A  R;
     Q -A']
```

with the calling sequence `[X1,X2,zero]=ric_descr(H)`, the solution `X` is given by `X=X1/X2`.

`zero` = L1 norm of rhs of (`Ec`)

The solution `X` is also given by `X=riccati(A,Q,R,'c'))`

In the discrete-time case calling sequence is `ric_descr(H,E)` (two inputs):

The Riccati equation is:

```
(Ed)  A'*X*A-(A'*X*B*(R+B'*X*B)^-1)*(B'*X*A)+C-X = 0.
```

Defining `G=B/R*B'` and the hamiltonian pencil `(E,H)` by:

```
E=[eye(n,n),G;                H=[A, 0*ones(n,n);
   0*ones(n,n),A']              -C, eye(n,n)];
```

with the calling sequence `[X1,X2,err]=ric_descr(H,E)`, the solution `X` is given by `X=X1/X2`.

`zero`= L1 norm of rhs of (`Ed`)

The solution `X` is also given by `X=riccati(A,G,C,'d')` with `G=B/R*B'`

# See Also

riccati

# Name

ricc — Riccati equation

```
[X,RCOND,FERR]=ricc(A,B,C,"cont""method")
[X,RCOND,FERR]=ricc(F,G,H,"disc","method")
```

# Parameters

A,B,C
: real matrices of appropriate dimensions

F,G,H
: real matrices of appropriate dimensions

X
: real matrix

"cont","disc"'
: imposed string (flag for continuous or discrete)

method
: 'schr' or 'sign' for continuous-time systems and 'schr' or 'invf' for discrete-tyme systems

# Description

Riccati solver.

Continuous time:

```
X=ricc(A,B,C,'cont')
```

gives a solution to the continuous time ARE

```
A'*X+X*A-X*B*X+C=0 .
```

`B` and `C` are assumed to be nonnegative definite. `(A,G)` is assumed to be stabilizable with `G*G'` a full rank factorization of `B`.

`(A,H)` is assumed to be detectable with `H*H'` a full rank factorization of `C`.

Discrete time:

```
X=ricc(F,G,H,'disc')
```

gives a solution to the discrete time ARE

```
X=F'*X*F-F'*X*G1*((G2+G1'*X*G1)^-1)*G1'*X*F+H
```

`F` is assumed invertible and `G = G1*inv(G2)*G1'`.

One assumes `(F,G1)` stabilizable and `(C,F)` detectable with `C'*C` full rank factorization of `H`. Use preferably `ric_desc`.

C, D are symmetric .It is assumed that the matrices A, C and D are such that the corresponding matrix pencil has N eigenvalues with moduli less than one.

Error bound on the solution and a condition estimate are also provided. It is assumed that the matrices A, C and D are such that the corresponding Hamiltonian matrix has N eigenvalues with negative real parts.

# Examples

```
//Standard formulas to compute Riccati solutions
A=rand(3,3);B=rand(3,2);C=rand(3,3);C=C*C';R=rand(2,2);R=R*R'+eye();
B=B*inv(R)*B';
X=ricc(A,B,C,'cont');
norm(A'*X+X*A-X*B*X+C,1)
H=[A -B;-C -A'];
[T,d]=schur(eye(H),H,'cont');T=T(:,1:d);
X1=T(4:6,:)/T(1:3,:);
norm(X1-X,1)
[T,d]=schur(H,'cont');T=T(:,1:d);
X2=T(4:6,:)/T(1:3,:);
norm(X2-X,1)
//       Discrete time case
F=A;B=rand(3,2);G1=B;G2=R;G=G1/G2*G1';H=C;
X=ricc(F,G,H,'disc');
norm(F'*X*F-(F'*X*G1/(G2+G1'*X*G1))*(G1'*X*F)+H-X)
H1=[eye(3,3) G;zeros(3,3) F'];
H2=[F zeros(3,3);-H eye(3,3)];
[T,d]=schur(H2,H1,'disc');T=T(:,1:d);X1=T(4:6,:)/T(1:3,:);
norm(X1-X,1)
Fi=inv(F);
Hami=[Fi Fi*G;H*Fi F'+H*Fi*G];
[T,d]=schur(Hami,'d');T=T(:,1:d);
Fit=inv(F');
Ham=[F+G*Fit*H -G*Fit;-Fit*H Fit];
[T,d]=schur(Ham,'d');T=T(:,1:d);X2=T(4:6,:)/T(1:3,:);
norm(X2-X,1)
```

# See Also

riccati , ric_desc , schur

# Authors

P. Petkov

# Used Functions

See SCI/modules/cacsd/src/slicot/riccpack.f

# Name

riccati — Riccati equation

```
X=riccati(A,B,C,dom,[typ])
[X1,X2]=riccati(A,B,C,dom,[typ])
```

# Parameters

A,B,C
: real matrices nxn, `B` and `C` symetric.

dom
: `'c'` or `'d'` for the time domain (continuous or discrete)

typ
: string : `'eigen'` for block diagonalization or `schur'` for Schur method.

X1,X2,X
: square real matrices (X2 invertible), X symmetric

# Description

`X=riccati(A,B,C,dom,[typ])` solves the Riccati equation:

```
A'*X+X*A-X*B*X+C=0
```

in continuous time case, or:

```
A'*X*A-(A'*X*B1/(B2+B1'*X*B1))*(B1'*X*A)+C-X
```

with `B=B1/B2*B1'` in the discrete time case. If called with two output arguments, `riccati` returns `X1,X2` such that `X=X1/X2`.

# See Also

ricc , ric_desc

# Name

rowinout — inner-outer factorization

```
[Inn,X,Gbar]=rowinout(G)
```

# Parameters

G
: linear system (`syslin` list) `[A,B,C,D]`

Inn
: inner factor (`syslin` list)

Gbar
: outer factor (`syslin` list)

X
: row-compressor of `G` (`syslin` list)

# Description

Inner-outer factorization (and row compression) of (`l`x`p`) `G =[A,B,C,D]` with `l>=p`.

`G` is assumed to be tall (`l>=p`) without zero on the imaginary axis and with a `D` matrix which is full column rank.

`G` must also be stable for having `Gbar` stable.

`G` admits the following inner-outer factorization:

```
 G = [ Inn ] | Gbar |
             |  0   |
```

where `Inn` is square and inner (all pass and stable) and `Gbar` square and outer i.e: Gbar is square bi-proper and bi-stable (Gbar inverse is also proper and stable);

Note that:

```
      [ Gbar ]
X*G = [  -   ]
      [  0   ]
```

is a row compression of `G` where `X` = `Inn` inverse is all-pass i.e:

```
 T
X (-s) X(s) = Identity
```

(for the continous time case).

# See Also

syslin , colinout

# Name

rowregul — removing poles and zeros at infinity

```
[Stmp,Ws]=rowregul(Sl,alfa,beta)
```

# Parameters

Sl,Stmp
: `syslin` lists

alfa,beta
: real numbers (new pole and zero positions)

# Description

computes a postfilter `Ws` such that `Stmp=Ws*Sl` is proper and with full rank `D` matrix.

Poles at infinity of `Sl` are moved to `alfa`;

Zeros at infinity of `Sl` are moved to `beta`;

`Sl` is a assumed to be a right invertible linear system (`syslin` list) in state-space representation with possibly a polynomial `D` matrix.

This function is the dual of colregul (see function code).

# Examples

```
s=%s;
w=[1/s,0;s/(s^3+2),2/s];
Sl=tf2ss(w);
[Stmp,Ws]=rowregul(Sl,-1,-2);
Stmp('D')     // D matrix of Stmp
clean(ss2tf(Stmp))
```

# See Also

invsyslin , colregul

# Authors

F. D. , R. N. ;

# Name

rtitr — discrete time response (transfer matrix)

```
[y]=rtitr(Num,Den,u [,up,yp])
```

# Parameters

Num,Den
: polynomial matrices (resp. dimensions : `nxm` and `nxn`)

u
: real matrix (dimension `mx(t+1)`

up,yp
: real matrices (`up` dimension `mx(maxi(degree(Den)))` (default values=`0`) , `yp` dimension `nx (maxi(degree(Den)))`)

y
: real matrix

# Description

`y=rtitr(Num,Den,u [,up,yp])` returns the time response of the discrete time linear system with transfer matrix `Den^-1 Num` for the input `u`, i.e `y` and `u` are such that `Den y = Num u` at t=0,1,...

If `d1=maxi(degree(Den))`, and `d2=maxi(degree(Num))` the polynomial matrices `Den(z)` and `Num(z)` may be written respectively as:

```
D(z) = D_0  + D_1  z + ... + D_d1   z^d1
N(z) = N_0  + N_1  z + ... + N_d2   z^d2
```

and `Den y = Num u` is interpreted as the recursion:

```
D(0)y(t)+D(1)y(t+1)+...+ D(d1)y(t+d1)= N(0) u(t) +....+ N(d2) u(t+d2)
```

It is assumed that `D(d1)` is non singular.

The columns of u are the inputs of the system at t=0,1,...,T:

```
u=[u(0) , u(1),...,u(T)]
```

The outputs at `t=0,1,...,T+d1-d2` are the columns of the matrix `y`:

```
y = [y(0), y(1),  .... y(T+d1-d2)]
```

`up` and `yp` define the initial conditions for t < 0 i.e

```
up = [u(-d1), ..., u(-1)  ]
```

```
yp = [y(-d1), ...  y(-1)  ]
```

Depending on the relative values of `d1` and `d2`, some of the leftmost components of `up`, `yp` are ignored. The default values of `up` and `yp` are zero: `up = 0*ones(m,d1), yp=0*ones(n,d1)`

# Examples

```
z=poly(0,'z');
Num=1+z;Den=1+z;u=[1,2,3,4,5];
rtitr(Num,Den,u)-u
//Other examples
//siso
//causal
n1=1;d1=poly([1 1],'z','coeff');       // y(j)=-y(j-1)+u(j-1)
r1=[0 1 0 1 0 1 0 1 0 1 0];
r=rtitr(n1,d1,ones(1,10));norm(r1-r,1)
//hot restart
r=rtitr(n1,d1,ones(1,9),1,0);norm(r1(2:11)-r)
//non causal
n2=poly([1 1 1],'z','coeff');d2=d1;    // y(j)=-y(j-1)+u(j-1)+u(j)+u(j+1)
r2=[2 1 2 1 2 1 2 1 2];
r=rtitr(n2,d2,ones(1,10));norm(r-r2,1)
//hot restart
r=rtitr(n2,d2,ones(1,9),1,2);norm(r2(2:9)-r,1)
//
//MIMO example
//causal
d1=d1*diag([1 0.5]);n1=[1 3 1;2 4 1];r1=[5;14]*r1;
r=rtitr(n1,d1,ones(3,10));norm(r1-r,1)
//
r=rtitr(n1,d1,ones(3,9),[1;1;1],[0;0]);
norm(r1(:,2:11)-r,1)
//polynomial n1  (same ex.)
n1(1,1)=poly(1,'z','c');r=rtitr(n1,d1,ones(3,10));norm(r1-r,1)
//
r=rtitr(n1,d1,ones(3,9),[1;1;1],[0;0]);
norm(r1(:,2:11)-r,1)
//non causal
d2=d1;n2=n2*n1;r2=[5;14]*r2;
r=rtitr(n2,d2,ones(3,10));norm(r2-r)
//
r=rtitr(n2,d2,ones(3,9),[1;1;1],[10;28]);
norm(r2(:,2:9)-r,1)
//
//  State-space or transfer
a = [0.21 , 0.63 , 0.56 , 0.23 , 0.31
     0.76 , 0.85 , 0.66 , 0.23 , 0.93
     0 , 0.69 , 0.73 , 0.22 , 0.21
     0.33 , 0.88 , 0.2 , 0.88 , 0.31
     0.67 , 0.07 , 0.54 , 0.65 , 0.36];
b = [0.29 , 0.5 , 0.92
     0.57 , 0.44 , 0.04
     0.48 , 0.27 , 0.48
     0.33 , 0.63 , 0.26
     0.59 , 0.41 , 0.41];
c = [0.28 , 0.78 , 0.11 , 0.15 , 0.84
```

```
     0.13 , 0.21 , 0.69 , 0.7 , 0.41];
d = [0.41 , 0.11 , 0.56
     0.88 , 0.2 , 0.59];
s=syslin('d',a,b,c,d);
h=ss2tf(s);num=h('num');den=h('den');den=den(1,1)*eye(2,2);
u=1;u(3,10)=0;r3=flts(u,s);
r=rtitr(num,den,u);norm(r3-r,1)
```

# See Also

ltitr , exp , flts

# Name

sensi — sensitivity functions

```
[Se,Re,Te]=sensi(G,K)
[Si,Ri,Ti]=sensi(G,K,flag)
```

# Parameters

G
: standard plant (`syslin` list)

K
: compensator (`syslin` list)

flag
: character string `'o'` (default value) or `'i'`

Se
: output sensitivity function `(I+G*K)^-1`

Re
: `K*Se`

Te
: `G*K*Se` (output complementary sensitivity function)

# Description

`sensi` computes sensitivity functions. If `G` and `K` are given in state-space form, the systems returned are generically minimal. Calculation is made by `lft`, e.g., `Se` can be given by the commands `P = augment(G,'S')`, `Se=lft(P,K)`. If `flag` = `'i'`, `[Si,Ri,Ti]=sensi(G,K,'i')` returns the input sensitivity functions.

```
[Se;Re;Te]= [inv(eye()+G*K);K*inv(eye()+G*K);G*K*inv(eye()+G*K)];
[Si;Ri;Ti]= [inv(eye()+K*G);G*inv(eye()+K*G);K*G*inv(eye()+K*G)];
```

# Examples

```
G=ssrand(1,1,3);K=ssrand(1,1,3);
[Se,Re,Te]=sensi(G,K);
Se1=inv(eye()+G*K);  //Other way to compute
ss2tf(Se)    //Se seen in transfer form
ss2tf(Se1)
ss2tf(Te)
ss2tf(G*K*Se1)
[Si,Ri,Ti]=sensi(G,K,'i');
w1=[ss2tf(Si);ss2tf(Ri);ss2tf(Ti)]
w2=[ss2tf(inv(eye()+K*G));ss2tf(G*inv(eye()+K*G));ss2tf(K*G*inv(eye()+K*G))];
clean(w1-w2)
```

# See Also

augment , lft , h_cl

# Name

show_margins — display gain and phase margin and associated crossover frequencies

```
show_margins(h)
show_margins(h,'bode')
show_margins(h,'nyquist')
```

# Parameters

h
: a SISO linear system (see :syslin).

# Description

Given a SISO linear system in continuous or discrete time, `show_margins` display gain and phase margin and associated crossover frequencies on a bode (the defaut) or nyquist representation of the frequency response of the system.

# Examples

```
//continuous case
h=syslin('c',0.02909+0.11827*%s+0.12823*%s^2+0.35659*%s^3+0.256*%s^4+0.1*%s^5,.
             0.0409+0.1827*%s+1.28225*%s^2+3.1909*%s^3+2.56*%s^4+%s^5);
show_margins(h)
show_margins(h,'nyquist')

//discrete case
h = syslin(0.1,0.01547+0.01599*%z ,%z^2-1.81*%z+0.9048)
show_margins(h)
show_margins(h,'nyquist')
```

# See Also

p_margin, g_margin, bode, nyquist

# Authors

Serge Steer, INRIA

# Name

sident — discrete-time state-space realization and Kalman gain

```
[(A,C)(,B(,D))(,K,Q,Ry,S)(,rcnd)] = sident(meth,job,s,n,l,R(,tol,t,Ai,
Ci,printw))
```

# Parameters

meth
: integer option to determine the method to use:

=
: 1 : MOESP method with past inputs and outputs;

=
: 2 : N4SID method;

=
: 3 : combined method: A and C via MOESP, B and D via N4SID.

job
: integer option to determine the calculation to be performed:

=
: 1 : compute all system matrices, A, B, C, D;

=
: 2 : compute the matrices A and C only;

=
: 3 : compute the matrix B only;

=
: 4 : compute the matrices B and D only.

s
: the number of block rows in the processed input and output block Hankel matrices. s > 0.

n
: integer, the order of the system

l
: integer, the number of the system outputs

R
: the 2*(m+l)*s-by-2*(m+l)*s part of R contains the processed upper triangular factor R from the QR factorization of the concatenated block-Hankel matrices, and further details needed for computing system matrices.

tol
: (optional) tolerance used for estimating the rank of matrices. If tol > 0, then the given value of tol is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than 1/tol is considered to be of full rank. Default: m*n*epsilon_machine where epsilon_machine is the relative machine precision.

t
: (optional) the total number of samples used for calculating the covariance matrices. Either t = 0, or t >= 2*(m+l)*s. This parameter is not needed if the covariance matrices and/or the Kalman predictor gain matrix are not desired. If t = 0, then K, Q, Ry, and S are not computed. Default: t = 0.

Ai
real matrix

Ci
real matrix

printw
(optional) switch for printing the warning messages.

=
1: print warning messages;

=
0: do not print warning messages.

Default: printw = 0.

A
real matrix

C
real matrix

B
real matrix

D
real matrix

K
real matrix, kalman gain

Q
(optional) the n-by-n positive semidefinite state covariance matrix used as state weighting matrix when computing the Kalman gain.

RY
(optional) the l-by-l positive (semi)definite output covariance matrix used as output weighting matrix when computing the Kalman gain.

S
(optional) the n-by-l state-output cross-covariance matrix used as cross-weighting matrix when computing the Kalman gain.

rcnd
(optional) vector of length lr, containing estimates of the reciprocal condition numbers of the matrices involved in rank decisions, least squares, or Riccati equation solutions, where lr = 4, if Kalman gain matrix K is not required, and lr = 12, if Kalman gain matrix K is required.

# Description

SIDENT function for computing a discrete-time state-space realization (A,B,C,D) and Kalman gain K using SLICOT routine IB01BD.

```
              [A,C,B,D] = sident(meth,1,s,n,l,R)
[A,C,B,D,K,Q,Ry,S,rcnd] = sident(meth,1,s,n,l,R,tol,t)
                  [A,C] = sident(meth,2,s,n,l,R)
                      B = sident(meth,3,s,n,l,R,tol,0,Ai,Ci)
      [B,K,Q,Ry,S,rcnd] = sident(meth,3,s,n,l,R,tol,t,Ai,Ci)
```

```
                  [B,D] = sident(meth,4,s,n,l,R,tol,0,Ai,Ci)
    [B,D,K,Q,Ry,S,rcnd] = sident(meth,4,s,n,l,R,tol,t,Ai,Ci)
```

SIDENT computes a state-space realization (A,B,C,D) and the Kalman predictor gain K of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP, N4SID, or their combination).

The model structure is :

```
x(k+1) = Ax(k) + Bu(k) + Ke(k),   k >= 1,
y(k)   = Cx(k) + Du(k) + e(k),
```

where x(k) is the n-dimensional state vector (at time k),

u(k) is the m-dimensional input vector,

y(k) is the l-dimensional output vector,

e(k) is the l-dimensional disturbance vector,

and A, B, C, D, and K are real matrices of appropriate dimensions.

# Comments

1. The n-by-n system state matrix A, and the p-by-n system output matrix C are computed for job <= 2.

2. The n-by-m system input matrix B is computed for job <> 2.

3. The l-by-m system matrix D is computed for job = 1 or 4.

4. The n-by-l Kalman predictor gain matrix K and the covariance matrices Q, Ry, and S are computed for t > 0.

# Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

S = 15;
N = 3;
METH=1;
[R,N1] = findR(S,Y',U',METH);
[A,C,B,D,K] = sident(METH,1,S,N,1,R);
SYS1=syslin(1,A,B,C,D);
SYS1.X0 = inistate(SYS1,Y',U');
```

```
Y1=flts(U,SYS1);
clf();plot2d((1:nsmp)',[Y',Y1'])

METH = 2;
[R,N1,SVAL] = findR(S,Y',U',METH);
tol = 0;
t = size(U',1)-2*S+1;

[A,C,B,D,K] = sident(METH,1,S,N,1,R,tol,t)
SYS1=syslin(1,A,B,C,D)
SYS1.X0 = inistate(SYS1,Y',U');

Y1=flts(U,SYS1);
clf();plot2d((1:nsmp)',[Y',Y1'])
```

# See Also

findBD , sorder

# Authors

V. Sima, Research Institute for Informatics, Bucharest, Oct. 1999. Revisions: May 2000, July 2000.

# Name

sm2des — system matrix to descriptor

```
[Des]=sm2des(Sm);
```

# Parameters

Sm
: polynomial matrix (pencil system matrix)

Des
: descriptor system (`list('des',A,B,C,D,E)`)

# Description

Utility function: converts the system matrix:

```
Sm = [-sE + A   B;
     [    C     D]
```

to descriptor system `Des=list('des',A,B,C,D,E)`).

# See Also

ss2des , sm2ss

# Name

sm2ss — system matrix to state-space

```
[Sl]=sm2ss(Sm);
```

# Parameters

Sm
: polynomial matrix (pencil system matrix)

Sl
: linear system (`syslin` list)

# Description

Utility function: converts the system matrix:

```
Sm = [-sI + A   B;
     [    C     D]
```

to linear system in state-space representation (`syslin`) list.

# See Also

ss2des

# Name

sorder — computing the order of a discrete-time system

```
[Ro(,n,sval,rcnd)] = sorder(meth,alg,jobd,batch,conct,s,Y(,U,tol,
printw,ldwork,Ri))
```

# Parameters

meth
: integer option to determine the method to use:

=
: 1 : MOESP method with past inputs and outputs;

=
: 2 : N4SID method.

alg
: integer option to determine the algorithm for computing the triangular factor of the concatenated block-Hankel matrices built from the input-output data:

=
: 1 : Cholesky algorithm on the correlation matrix;

=
: 2 : fast QR algorithm;

=
: 3 : standard QR algorithm.

jobd
: integer option to specify if the matrices B and D should later be computed using the MOESP approach:

=
: 1 : the matrices B and D should later be computed using the MOESP approach;

=
: 2 : the matrices B and D should not be computed using the MOESP approach.

This parameter is not relevant for meth = 2.

batch
: integer option to specify whether or not sequential data processing is to be used, and, for sequential processing, whether or not the current data block is the first block, an intermediate block, or the last block, as follows:

=
: 1 : the first block in sequential data processing;

=
: 2 : an intermediate block in sequential data processing;

=
: 3 : the last block in sequential data processing;

=
: 4 : one block only (non-sequential data processing).

conct

integer option to specify whether or not the successive data blocks in sequential data processing belong to a single experiment, as follows:

=

1 : the current data block is a continuation of the previous data block and/or it will be continued by the next data block;

=

2 : there is no connection between the current data block and the previous and/or the next ones.

This parameter is not used if batch = 4.

s

the number of block rows in the input and output block Hankel matrices to be processed. s > 0

Y

the t-by-l output-data sequence matrix. Column j of Y contains the t values of the j-th output component for consecutive time increments.

U

(optional) the t-by-m input-data sequence matrix. Column j of U contains the t values of the j-th input component for consecutive time increments. Default: U = [].

tol

(optional) vector of length 2 containing tolerances: tol(1) - tolerance used for estimating the rank of matrices. If tol(1) > 0, then the given value of tol(1) is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than 1/tol(1) is considered to be of full rank. If tol(1) <= 0, then a default value m*n*epsilon_machine is used, where epsilon_machine is the relative machine precision.

tol(2) - tolerance used for determining an estimate of the system order. If tol(2) >= 0, the estimate is indicated by the index of the last singular value greater than or equal to tol(2). (Singular values less than tol(2) are considered as zero.) When tol(2) = 0, an internally computed default value, tol(2) = s*epsilon_machine*sval(1), is used, where sval(1) is the maximal singular value, and epsilon_machine the relative machine precision. When tol(2) < 0, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor.

Default: tol(1:2) = [0,-1].

printw

(optional) switch for printing the warning messages.

=

1: print warning messages;

=

0: do not print warning messages.

Default: printw = 0.

ldwork

(optional) the workspace size. Default : computed by the formulas

```
nr = 2*( m + l )*s
LDWORK = ( t - 2*s + 3 + 64 )*nr
if ( CSIZE > MAX( nr*nr + t*( m + l ) + 16, 2*nr ) ) then
   LDWORK = MIN( LDWORK, CSIZE - nr*nr - t*( m + l ) - 16 )
else
   LDWORK = MIN( LDWORK, MAX( 2*nr, CSIZE/2 ) )
```

```
end if
```

LDWORK = MAX( minimum workspace size needed, LDWORK ) where CSIZE is the cache size in double precision words.

If LDWORK is specified less than the minimum workspace size needed, that minimum value is used instead.

Ri
: (optional) if batch = 2 or 3, the 2*(m+l)*s-by-2*(m+l)*s (upper triangular, if alg <> 2) part of R must contain the (upper triangular) matrix R computed at the previous call of this mexfile in sequential data processing. If conct = 1, R has an additional column, also set at the previous call.

If alg = 2, R has m+l+1 additional columns, set at the previous call.

This parameter is not used for batch = 1 or batch = 4.

Ro
: if batch = 3 or 4, the 2*(m+l)*s-by-2*(m+l)*s part of R contains the processed upper triangular factor R from the QR factorization of the concatenated block-Hankel matrices, and further details needed for computing system matrices. If batch = 1 or 2, then R contains intermediate results needed at the next call of this mexfile. If batch = 1 or 2 and conct = 1, R has an additional column, also set before return. If batch = 1 or 2 and alg = 2, R has m+l+1 additional columns, set before return.

n
: the order of the system.

sval
: (optional) the singular values used for estimating the order of the system.

rcnd
: (optional) if meth = 2, vector of length 2 containing the reciprocal condition numbers of the matrices involved in rank decisions or least squares solutions.

# Description

sorder - function for computing the order of a discrete-time system using SLICOT routine IB01AD.

For one block (data sequences Y, U): [R,n,sval,rcnd] = sorder(meth,alg,jobd,4,conct,s,Y,U);

For f blocks (data sequences Yj, Uj, j = 1 : f):

```
R = sorder(meth,alg,jobd,1,conct,s,Y1,U1);
for j = 2 : f - 1
   R = sorder(meth,alg,jobd,2,conct,s,Yj,Uj,tol,printw,ldwork,R)
end
[R,n,sval,rcnd] = sorder(meth,alg,jobd,3,conct,s,Yf,Uf,tol);
```

sorder preprocesses the input-output data for estimating the matrices of a linear time-invariant dynamical system, using Cholesky or (fast) QR factorization and subspace identification techniques (MOESP and N4SID), and then estimates the order of a discrete-time realization.

The model structure is :

```
x(k+1) = Ax(k) + Bu(k) + w(k),   k >= 1,
```

```
y(k)   = Cx(k) + Du(k) + e(k),
```

where x(k) is the n-dimensional state vector (at time k),

u(k) is the m-dimensional input vector,

y(k) is the l-dimensional output vector,

w(k) is the n-dimensional state disturbance vector,

e(k) is the l-dimensional output disturbance vector,

and A, B, C, and D are real matrices of appropriate dimensions.

# Comments

1. The Cholesy or fast QR algorithms can be much faster (for large data blocks) than QR algorithm, but they cannot be used if the correlation matrix, H'*H, is not positive definite. In such a case, the code automatically switches to the QR algorithm, if sufficient workspace is provided and batch = 4.

2. If ldwork is specified, but it is less than the minimum workspace size needed, that minimum value is used instead.

# See Also

findBD , sident

# Authors

V. Sima, Research Institute for Informatics, Bucharest, Oct. 1999.; ; Revisions:; V. Sima, May 2000, July 2000.

# Name

specfact — spectral factor

```
[W0,L]=specfact(A,B,C,D)
```

# Description

Given a spectral density matrix `phi(s)`:

```
            -1                    -1
R + C*(s*I-A) * B  +  B'*(-s*I-A') * C'  with R=D+D' > 0
```

`specfact` computes `W0` and `L` such that `W(s)=W0+L*(s*I-A)^-1*B` is a spectral factor of of `PHI(s)`, i.e.

```
phi(s)=W'(-s)*W(s)
```

# Examples

```
A=diag([-1,-2]);B=[1;1];C=[1,1];D=1;s=poly(0,'s');
W1=syslin('c',A,B,C,D);
phi=gtild(W1,'c')+W1;
phis=clean(ss2tf(phi))
clean(phis-horner(phis,-s)');   //check this is 0...
[A,B,C,D]=abcd(W1);
[W0,L]=specfact(A,B,C,D);
W=syslin('c',A,B,L,W0)
Ws=ss2tf(W);
horner(Ws,-s)*Ws
```

# See Also

gtild , sfact , fspecg

# Authors

F. D.

# Name

ss2des — (polynomial) state-space to descriptor form

```
S=ss2des(Sl)
S=ss2des(Sl,flag)
```

# Parameters

Sl
: `syslin` list: proper or improper linear system.

flag
: character string `"withD"`

S
: list

# Description

Given the linear system in state-space representation `Sl` (`syslin` list), with a `D` matrix which is either polynomial or constant, but not zero `ss2des` returns a descriptor system as `list('des',A,B,C,0,E)` such that:

```
Sl=C*(s*E-A)^(-1)*B
```

If the flag `"withD"` is given, `S=list('des',A,B,C,D,E)` with a `D` matrix of maximal rank.

# Examples

```
s=poly(0,'s');
G=[1/(s+1),s;1+s^2,3*s^3];Sl=tf2ss(G);
S=ss2des(Sl)
S1=ss2des(Sl,"withD")
Des=des2ss(S);Des(5)=clean(Des(5))
Des1=des2ss(S1)
```

# See Also

pol2des , tf2des , des2ss

# Authors

F. D.; ;

# Name

ss2ss — state-space to state-space conversion, feedback, injection

```
[Sl1,right,left]=ss2ss(Sl,T, [F, [G , [flag]]])
```

# Parameters

Sl
: linear system (`syslin` list) in state-space form

T
: square (non-singular) matrix

Sl1, right, left
: linear systems (syslin lists) in state-space form

F
: real matrix (state feedback gain)

G
: real matrix (output injection gain)

# Description

Returns the linear system `Sl1=[A1,B1,C1,D1]` where `A1=inv(T)*A*T, B1=inv(T)*B, C1=C*T, D1=D`.

Optional parameters `F` and `G` are state feedback and output injection respectively.

For example, `Sl1=ss2ss(Sl,T,F)` returns `Sl1` with:

$$\mathrm{sl1}=\begin{pmatrix} T^{-1}*(A+B*F)*T & T^{-1}*B \\ (C+D*F)*T & D \end{pmatrix}$$

and `right` is a non singular linear system such that `Sl1=Sl*right`.

`Sl1*inv(right)` is a factorization of `Sl`.

`Sl1=ss2ss(Sl,T,0*F,G)` returns `Sl1` with:

$$\mathrm{sl1}=\begin{pmatrix} T^{-1}*(A+G*C)*T & T^{-1}*(B+G*D) \\ C*T & D \end{pmatrix}$$

and `left` is a non singular linear system such that `Sl1=left*Sl` (`right=Id` if `F=0`).

When both `F` and `G` are given, `Sl1=left*Sl*right`.

- When `flag` is used and `flag=1` an output injection as follows is used

$$\begin{pmatrix} T^{-1}*(A+G*C)*T & T^{-1}*(B+G*D,-G) \\ C*T & (D,0) \end{pmatrix}$$

  and then a feedback is performed, `F` must be of size `(m+p,n)`

$$x\in \mathbb{R}^{n}, y\in \mathbb{R}^{p}, u\in \mathbb{R}^{m}$$

`right` and `left` have the following property:

```
Sl1 = left*sysdiag(sys,eye(p,p))*right
```

- When `flag` is used and `flag=2` a feedback (`F` must be of size `(m,n)`) is performed and then the above output injection is applied. `right` and `left` have the following property:

```
 Sl1 = left*sysdiag(sys*right,eye(p,p)))
```

# Examples

```
Sl=ssrand(2,2,5); trzeros(Sl)       // zeros are invariant:
Sl1=ss2ss(Sl,rand(5,5),rand(2,5),rand(5,2));
trzeros(Sl1), trzeros(rand(2,2)*Sl1*rand(2,2))
// output injection [ A + GC, (B+GD,-G)]
//                  [   C   , (D   , 0)]
p=1,m=2,n=2; sys=ssrand(p,m,n);

// feedback (m,n)  first and then output injection.

F1=rand(m,n);
G=rand(n,p);
[sys1,right,left]=ss2ss(sys,rand(n,n),F1,G,2);

// Sl1 equiv left*sysdiag(sys*right,eye(p,p)))

res=clean(ss2tf(sys1) - ss2tf(left*sysdiag(sys*right,eye(p,p))))

// output injection then feedback (m+p,n)
F2=rand(p,n); F=[F1;F2];
[sys2,right,left]=ss2ss(sys,rand(n,n),F,G,1);

// Sl1 equiv left*sysdiag(sys,eye(p,p))*right

res=clean(ss2tf(sys2)-ss2tf(left*sysdiag(sys,eye(p,p))*right))

// when F2= 0; sys1 and sys2 are the same
F2=0*rand(p,n);F=[F1;F2];
[sys2,right,left]=ss2ss(sys,rand(n,n),F,G,1);

res=clean(ss2tf(sys2)-ss2tf(sys1))
```

# See Also

projsl, feedback

# Name

ss2tf — conversion from state-space to transfer function

```
[h]=ss2tf(sl)
[Ds,NUM,chi]=ss2tf(sl)

[h]=ss2tf(sl,"b")
[Ds,NUM,chi]=ss2tf(sl,"b")


[h]=ss2tf(sl,rmax)
[Ds,NUM,chi]=ss2tf(sl,rmax)
```

# Parameters

sl
: linear system (`syslin` list)

h
: transfer matrix

# Description

Called with three outputs `[Ds,NUM,chi]=ss2tf(sl)` returns the numerator polynomial matrix `NUM`, the characteristic polynomial `chi` and the polynomial part `Ds` separately i.e.:

```
h = NUM/chi + Ds
```

Method:

One uses the characteristic polynomial and `det(A+Eij)=det(A)+C(i,j)` where `C` is the adjugate matrix of `A`.

With `rmax` or `"b"` argument uses a block diagonalization of sl.A matrix and applies "Leverrier" algorithm on blocks. If given, `rmax` controls the conditionning (see bdiag).

# Examples

```
s=poly(0,'s');
h=[1,1/s;1/(s^2+1),s/(s^2-2)]
sl=tf2ss(h);
h=clean(ss2tf(sl))
[Ds,NUM,chi]=ss2tf(sl)
```

# See Also

tf2ss , syslin , nlev , glever

# Name

st_ility — stabilizability test

```
[ns, [nc, [,U [,Slo] ]]]=st_ility(Sl [,tol])
```

# Parameters

Sl
: `syslin` list (linear system)

ns
: integer (dimension of stabilizable subspace)

nc
: integer (dimension of controllable subspace `nc <= ns`)

U
: basis such that its `ns` (resp. `nc`) first components span the stabilizable (resp. controllable) subspace

Slo
: a linear system (`syslin` list)

tol
: threshold for controllability detection (see contr)

# Description

`Slo=( U'*A*U, U'*B, C*U, D, U'*x0 )` (`syslin` list) displays the stabilizable form of `Sl`. Stabilizability means `ns=nx` (dim. of `A` matrix).

```
         [*,*,*]            [*]
U'*A*U = [0,*,*]     U'*B = [0]
         [0,0,*]            [0]
```

where `(A11,B1)` (dim(A11)= `nc`) is controllable and `A22` (dim(A22)=`ns-nc`) is stable. "Stable" means real part of eigenvalues negative for a continuous linear system, and magnitude of eigenvalues lower than one for a discrete-time system (as defined by `syslin`).

# Examples

```
A=diag([0.9,-2,3]);B=[0;0;1];Sl=syslin('c',A,B,[]);
[ns,nc,U]=st_ility(Sl);
U'*A*U
U'*B
[ns,nc,U]=st_ility(syslin('d',A,B,[]));
U'*A*U
U'*B
```

# See Also

dt_ility , contr , stabil , ssrand

# Authors

S. Steer INRIA 1988

# Name

stabil — stabilization

```
F=stabil(A,B,alfa)
K=stabil(Sys,alfa,beta)
```

# Parameters

A
: square real matrix (`nx x nx`)

B
: real matrix (`nx x nu`)

alfa, beta
: real or complex vector (in conjugate pairs) or real number.

F
: real matrix (`nx x nu`)

Sys
: linear system (`syslin` list) (`m` inputs, `p` outputs).

K
: linear system (`p` inputs, `m` outputs)

# Description

`F=stabil(A,B,alfa)` returns a gain matrix `F` such that `A+B*F` is stable if pair `(A,B)` is stabilizable. Assignable poles are set to `alfa(1),alfa(2),...`. If `(A,B)` is not stabilizable a warning is given and assignable poles are set to `alfa(1),alfa(2),...`. If `alfa` is a number all eigenvalues are set to this `alfa` (default value is `alfa=-1`).

`K=stabil(Sys,alfa,beta)` returns `K`, a compensator for `Sys` such that `(A,B)`-controllable eigenvalues are set to `alfa` and `(C,A)`-observable eigenvalues are set to `beta`.

All assignable closed loop poles (which are given by the eigenvalues of `Aclosed=h_cl(Sys,K)` are set to `alfa(i)`'s and `beta(j)`'s.

# Examples

```
// Gain:
Sys=ssrand(0,2,5,list('st',2,3,3));
A=Sys('A');B=Sys('B');F=stabil(A,B);
spec(A) //2 controllable modes 2 unstable uncontrollable modes
//and one stable uncontrollable mode
spec(A+B*F) //the two controllable modes are set to -1.
// Compensator:
Sys=ssrand(3,2,5,list('st',2,3,3)); //3 outputs, 2 inputs, 5 states
//2 controllables modes, 3 controllable or stabilizable modes.
K=stabil(Sys,-2,-3);  //Compensator for Sys.
spec(Sys('A'))
spec(h_cl(Sys,K))   //K Stabilizes what can be stabilized.
```

# See Also

st_ility , contr , ppol

# Name

svplot — singular-value sigma-plot

```
[SVM]=svplot(sl,[w])
```

# Parameters

sl
: `syslin` list (continuous, discrete or sampled system)

w
: real vector (optional parameter)

# Description

computes for the system `sl=(A,B,C,D)` the singular values of its transfer function matrix:

```
              G(jw) = C(jw*I-A)B^-1+D
or
              G(exp(jw)) = C(exp(jw)*I-A)B^-1+D
or
              G(exp(jwT)) = C(exp(jw*T)*I-A)B^-1+D
```

evaluated over the frequency range specified by `w`. (T is the sampling period, `T=sl('dt')` for sampled systems).

`sl` is a `syslin` list representing the system `[A,B,C,D]` in state-space form. `sl` can be continous or discrete time or sampled system.

The `i`-th column of the output matrix `SVM` contains the singular values of `G` for the `i`-th frequency value `w(i)`.

```
SVM = svplot(sl)
```

is equivalent to

```
SVM = svplot(sl,logspace(-3,3))  (continuous)
```

```
SVM = svplot(sl,logspace(-3,%pi)) (discrete)
```

# Examples

```
x=logspace(-3,3);
y=svplot(ssrand(2,2,4),x);
clf();plot2d1("oln",x',20*log(y')/log(10));
xgrid(12)
xtitle("Singular values plot","(Rd/sec)", "Db");
```

# Authors

F.D; ;

# Name

sysfact — system factorization

```
[S,Series]=sysfact(Sys,Gain,flag)
```

# Parameters

Sys
: `syslin` list containing the matrices `[A,B,C,D]`.

Gain
: real matrix

flag
: string `'post'` or `'pre'`

S
: `syslin` list

Series
: `syslin` list

# Description

If `flag` equals `'post'`, `sysfact` returns in `S` the linear system with ABCD matrices `(A +B*Gain, B , Gain, I)`, and `Series`, a minimal realization of the series system `Sys*S`. If `flag` equals `'pre'`, `sysfact` returns the linear system `(A+Gain*C, Gain , C, I)` and `Series`, a minimal realization of the series system `S*Sys`.

# Examples

```
//Kalman filter
Sys=ssrand(3,2,4);Sys('D')=rand(3,2);
S=sysfact(Sys,lqr(Sys),'post');
ww=minss(Sys*S);
ss2tf(gtild(ww)*ww),Sys('D')'*Sys('D')
//Kernel
Sys=ssrand(2,3,4);
[X,d,F,U,k,Z]=abinv(Sys);
ss2tf(Sys*Z)
ss2tf(Sys*sysfact(Sys,F,'post')*U)
```

# See Also

lqr , lqe

# Authors

F.D.

# Name

syssize — size of state-space system

```
[r,nx]=syssize(Sl)
```

# Parameters

Sl
: linear system (`syslin` list) in state-space

r
: 1 x 2 real vector

nx
: integer

# Description

returns in `r` the vector [number of outputs, number of inputs] of the linear system `Sl`. `nx` is the number of states of `Sl`.

# See Also

size

# Name

tf2des — transfer function to descriptor

```
S=tf2des(G)
S=tf2des(G,flag)
```

# Parameters

G
: linear system (`syslin` list) with possibly polynomial `D` matrix

flag
: character string `"withD"`

S
: list

# Description

Transfer function to descriptor form: `S=list('d',A,B,C,D,E)`

```
E*xdot = A*x+B*u
y = C*x + D*u
```

Note that `D=0` if the optional parameter flag=`"withD"` is not given. Otherwise a maximal rank `D` matrix is returned in the fifth entry of the list `S`

# Examples

```
s=poly(0,'s');
G=[1/(s-1),s;1,2/s^3];
S1=tf2des(G);des2tf(S1)
S2=tf2des(G,"withD");des2tf(S2)
```

# See Also

pol2des , tf2ss , ss2des , des2tf

# Name

tf2ss — transfer to state-space

```
sl=tf2ss(h [,tol])
```

# Parameters

h
: rational matrix

tol
: may be the constant rtol or the 2 vector `[rtol atol]`

  rtol
  : tolerance used when evaluating observability.

  atol
  : absolute tolerance used when evaluating observability.

sl
: linear system (`syslin` list `sl=[A,B,C,D(s)]`)

# Description

transfer to state-space conversion:

`h=C*(s*eye()-A)^-1*B+D(s)`

# Examples

```
s=poly(0,'s');
H=[2/s,(s+1)/(s^2-5)];
Sys=tf2ss(H)
clean(ss2tf(Sys))
```

# See Also

ss2tf , tf2des , des2tf

# Name

time_id — SISO least square identification

```
[H [,err]]=time_id(n,u,y)
```

# Parameters

n
: order of transfer

u
: one of the following

  u1
  : a vector of inputs to the system

  "impuls"
  : if y is an impulse response

  "step"
  : if y is a step response.

y
: vector of response.

H
: rational function with degree n denominator and degree n-1 numerator if y(1)==0 or rational function with degree n denominator and numerator if y(1)<>0.

err
: `||y - impuls(H,npt)||^2`, where `impuls(H,npt)` are the `npt` first coefficients of impulse response of `H`

# Description

Identification of discrete time response. If `y` is strictly proper (`y(1)=0`) then `time_id` computes the least square solution of the linear equation: `Den*y-Num*u=0` with the constraint `coeff(Den,n):=1`. if `y(1)~=0` then the algorithm first computes the proper part solution and then add y(1) to the solution

# Examples

```
z=poly(0,'z');
h=(1-2*z)/(z^2-0.5*z+5)
rep=[0;ldiv(h('num'),h('den'),20)]; //impulse response
H=time_id(2,'impuls',rep)
//  Same example with flts and u
u=zeros(1,20);u(1)=1;
rep=flts(u,tf2ss(h));        //impulse response
H=time_id(2,u,rep)
//  step response
u=ones(1,20);
rep=flts(u,tf2ss(h));     //step response.
H=time_id(2,'step',rep)
H=time_id(3,u,rep)    //with u as input and too high order required
```

# See Also

imrep2ss , arl2 , armax , frep2tf

# Authors

Serge Steer INRIA

# Name

trzeros — transmission zeros and normal rank

```
[tr]=trzeros(Sl)
[nt,dt,rk]=trzeros(Sl)
```

# Parameters

Sl
: linear system (`syslin` list)

nt
: complex vectors

dt
: real vector

rk
: integer (normal rank of Sl)

# Description

Called with one output argument, `trzeros(Sl)` returns the transmission zeros of the linear system `Sl`.

`Sl` may have a polynomial (but square) `D` matrix.

Called with 2 output arguments, `trzeros` returns the transmission zeros of the linear system `Sl` as `tr=nt./dt`;

(Note that some components of `dt` may be zeros)

Called with 3 output arguments, `rk` is the normal rank of `Sl`

Transfer matrices are converted to state-space.

If `Sl` is a (square) polynomial matrix `trzeros` returns the roots of its determinant.

For usual state-space system `trzeros` uses the state-space algorithm of Emami-Naeni and Van Dooren.

If `D` is invertible the transmission zeros are the eigenvalues of the "`A` matrix" of the inverse system : `A - B*inv(D)*C`;

If `C*B` is invertible the transmission zeros are the eigenvalues of `N*A*M` where `M*N` is a full rank factorization of `eye(A)-B*inv(C*B)*C`;

For systems with a polynomial `D` matrix zeros are calculated as the roots of the determinant of the system matrix.

Caution: the computed zeros are not always reliable, in particular in case of repeated zeros.

# Examples

```
W1=ssrand(2,2,5);trzeros(W1)    //call trzeros
roots(det(systmat(W1)))         //roots of det(system matrix)
s=poly(0,'s');W=[1/(s+1);1/(s-2)];W2=(s-3)*W*W';[nt,dt,rk]=trzeros(W2);
```

```
St=systmat(tf2ss(W2));[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(St);
St1=Q*St*Z;rowf=(Qd(1)+Qd(2)+1):(Qd(1)+Qd(2)+Qd(3));
colf=(Zd(1)+Zd(2)+1):(Zd(1)+Zd(2)+Zd(3));
roots(St1(rowf,colf)), nt./dt     //By Kronecker form
```

# See Also

gspec , kroneck

# Name

ui_observer — unknown input observer

```
[UIobs,J,N]=ui_observer(Sys,reject,C1,D1)
[UIobs,J,N]=ui_observer(Sys,reject,C1,D1,flag,alfa,beta)
```

# Parameters

Sys
: `syslin` list containing the matrices `(A,B,C2,D2)`.

reject
: integer vector, indices of inputs of `Sys` which are unknown.

C1
: real matrix

D1
: real matrix. `C1` and `D1` have the same number of rows.

flag
: string `'ge'` or `'st'` (default) or `'pp'`.

alfa
: real or complex vector (loc. of closed loop poles)

beta
: real or complex vector (loc. of closed loop poles)

# Description

Unknown input observer.

`Sys: (w,u) --> y` is a `(A,B,C2,D2)` syslin linear system with two inputs w and u, w being the unknown input. The matrices B and D2 of Sys are (implicitly) partitioned as: `B=[B1,B2]` and `D2=[D21,D22]` with `B1=B(:,reject)` and `D21=D2(:,reject)` where reject = indices of unknown inputs. The matrices `C1` and `D1` define `z = C1 x + D1 (w,u)`, the to-be-estimated output.

The matrix `D1` is (implicitly) partitioned as `D1=[D11,D12]` with `D11=D(:,reject)`

The data (Sys, reject,C1, D1) define a 2-input 2-output system:

```
xdot =  A x + B1  w + B2  u
   z = C1 x + D11 w + D12 u
   y = C2 x + D21 w + D22 u
```

An observer `(u,y) --> zhat` is looked for the output z.

`flag='ge'` no stability constraints `flag='st'` stable observer (default) `flag='pp'` observer with pole placement alfa,beta = desired location of closed loop poles (default -1, -2) J=y-output to x-state injection. N=y-output to z-estimated output injection.

UIobs = linear system (u,y) --> zhat such that: The transfer function: (w,u) --> z equals the composed transfer function: [0,I; UIobs Sys] (w,u) -----> (u,y) -----> zhat i.e. transfer function of system {A,B,C1,D1} equals transfer function `UIobs*[0,I; Sys]`

Stability (resp. pole placement) requires detectability (resp. observability) of (A,C2).

# Examples

```
A=diag([3,-3,7,4,-4,8]);
B=[eye(3,3);zeros(3,3)];
C=[0,0,1,2,3,4;0,0,0,0,0,1];
D=[1,2,3;0,0,0];
rand('seed',0);w=ss2ss(syslin('c',A,B,C,D),rand(6,6));
[A,B,C,D]=abcd(w);
B=[B,matrix(1:18,6,3)];D=[D,matrix(-(1:6),2,3)];
reject=1:3;
Sys=syslin('c',A,B,C,D);
N1=[-2,-3];C1=-N1*C;D1=-N1*D;
nw=length(reject);nu=size(Sys('B'),2)-nw;
ny=size(Sys('C'),1);nz=size(C1,1);
[UIobs,J,N]=ui_observer(Sys,reject,C1,D1);

W=[zeros(nu,nw),eye(nu,nu);Sys];UIobsW=UIobs*W;
//(w,u) --> z=UIobs*[0,I;Sys](w,u)
clean(ss2tf(UIobsW));
wu_to_z=syslin('c',A,B,C1,D1);clean(ss2tf(wu_to_z));
clean(ss2tf(wu_to_z)-ss2tf(UIobsW),1.d-7)
/////2nd example//////
nx=2;ny=3;nwu=2;Sys=ssrand(ny,nwu,nx);
C1=rand(1,nx);D1=[0,1];
UIobs=ui_observer(Sys,1,C1,D1);
```

# See Also

cainv , ddp , abinv

# Authors

F.D.

# Name

unobs — unobservable subspace

```
[n,[U]]=unobs(A,C,[tol])
```

# Parameters

A, C
: real matrices

tol
: tolerance used when evaluating ranks (QR factorizations).

n
: dimension of unobservable subspace.

U
: orthogonal change of basis which puts `(A,B)` in canonical form.

# Description

`[n,[U]]=unobs(A,C,[tol])` gives the unobservable form of an `(A,C)` pair. The `n` first columns of `U` make a basis for the unobservable subspace.

The `(2,1)` block (made of last `nx-n` rows and `n` first columns) of `U'*A*U` is zero and and the `n` first columns of `C*U` are zero.

# Examples

```
A=diag([1,2,3]);C=[1,0,0];
unobs(A,C)
```

# See Also

contr , contrss , canon , cont_mat , spantwo , dt_ility

# Name

zeropen — zero pencil

```
[Z,U]=zeropen(Sl)
```

# Parameters

Sl
: a linear system (`syslin` list in state-space form `[A,B,C,D]`)

Z
: matrix pencil `Z=s*E-A`

U
: square orthogonal matrix

# Description

`Z = sE - F` is the zero pencil of the linear system `Sl` with matrices `[A,B,C,D]`. Utility function.

With `U` row compression of `[B;D]` i.e, `U*[B;D]=[0;*]`; one has:

```
U*[-sI+A |B;   [ Z |0;
      C  |D] = [ * |*]
```

The zeros of `Z` are the zeros of `Sl`.

# See Also

systmat , kroneck

# Parte X. Estruturas de Dados

# Name

cell — cria um cell (célula) de matrizes vazias

```
c=cell()
c=cell(m1)
c=cell(m1, m2)
c=cell(m1, m2, ..., mn)
c=cell(x)
```

# Parâmetros

x
: vetor contendo as dimensões do cell a ser criado

m1, m2,..
: dimensões do cell a ser criado

# Descrição

retorna um cell de matrizes vazias

cell()
: retorna um cell `(0,0)` de matrizes vazias.

cell(m1)
: retorna um cell `(m1,m1)` de matrizes vazias.

cell(m1,m2)
: retorna um cell `(m1,m2)` de matrizes vazias.

cell(m1,m2,..,mn)
: retorna um cell `(m1,m2,..,mn)` de matrizes vazias.

cell(x)
: retorna um cell de matrizes vazias com: a primeira dimensão sendo `x(1)`, a segunda sendo `x(2), ...`

# Observações

`cell(x)` não possui o mesmo tamanho que `x`.

`cell()` é equivalente a `cell(0)`.

Se A for um cell, você pode acessar os conteúdos de um elemento de A utilizando `A(m1, m2, ..., mn).entries`. A expressão `A(1,1) = zeros(2,2)` não é válida, a sintaxe correta é `A(1,1).entries = zeros(2,2)`.

Se A é um cell, você pode obter suas dimensões através de `A.dims`.

# Exemplos

```
a=cell(3)
b=cell(3,1)
c=cell([2,3,4])
```

```
// atribuição em entradas do cell
b=cell(3,1);

// atribuindo o primeiro elemento de b utilizando o campo 'entries'
b(1).entries=1:3

// atribuindo o segundo elemento de b utilizando o campo 'entries'
b(2).entries='Scilab'

// atribuindo o terceiro elemento de b utilizando o campo 'entries'
b(3).entries=poly(1:3,'s')

// atribuição de sub-cells
X=cell(3,2);
X(:,1)=b

// extraindo um sub-cell: o resultado é um cell
b(1)
b(1:2)

// extraindo o valor de um sub-cell: o resultado é um array
b(1).entries

// dimensões de b
b.dims
```

# Ver Também

eye, ones, zeros

# Name

definedfields — retorna o índice dos campos definidos de uma lista

```
k=definedfields(l)
```

# Parâmetros

l
: uma lista, tlist ou mlist

k
: um vetor de índices .

# Descrição

Se `l` é do tipo `list` `tlist` `mlist` `k=definedfields(l)` retorna em `k` os índices dos campos definidos da lista. Esta função é útil porque a indexação de campos indefinidos acarreta erros.

# Exemplos

```
l=list(1);l(3)=5
k=definedfields(l)

t=tlist('x');t(5)=4
definedfields(t)

m=mlist(['m','a','b']);m.b='sdfgfgd'
definedfields(m)
```

# Ver Também

list, tlist, mlist, insertion, extraction

# Name

getfield — extração de campos de listas

```
[x,...]=getfield(i,l)
```

# Parâmetros

x
: matriz de qualquer tipo possível

l
: lista, tlist ou mlist

i
: índice de campo, ver "extraction" para mais detalhes.

# Descrição

Esta função é equivalente à sintaxe `[x,...]=l(i)` para extração de campos com a única diferença de que também se aplica a objetos `mlist`.

# Exemplos

```
l=list(1,'qwerw',%s)
[a,b]=getfield([3 2],l)

a=hypermat([2,2,2],rand(1:2^3));// hipermatrizes são codificadas usando-se mlis
a(1) // a entrada a(1,1,1)
getfield(1,a) // o primeiro campo do mlist
```

# Ver Também

extraction

# Name

hypermat — inicializa matrizes n-dimensionais

```
M=hypermat(dims [,v])
```

# Parâmetros

dims
: vetor de dimensões da hipermatriz

v
: vetor de entradas da hipermatriz (valor padrão `zeros(prod(dims),1)`)

# Descrição

Incializa uma hipermatriz cujas dimensões são dadas pelo vetor dims e as entradas são dadas pelo argumento opcional `v`

A estrutura de dados de M contém o vetor de dimensões da matriz `M('dims')` e o vetor de entradas `M('entries')` tais que os índices subescritos mais à esquerda variam primeiro: `[M(1,1,..);..;M(n1,1,..);...;M(1,n2,..);..;M(n1,n2,..);...]`

# Exemplos

```
M=hypermat([2 3 2 2],1:24)
```

# Name

hypermatrices — objeto do Scilab: matrizes n-dimensionais

# Descrição

O tipo hipermatriz ("hypermatrix") permite manipular arrays multidimensionais.

Hipermatrizes podem ser definidas por uma extensão das matrizes 2D como segue: a=[1 2;3 4];a(:,:,2)=rand(2,2)

ou usando a função `hypermat` diretamente.

As entradas podem ser números reais ou complexos, polinômios, razões de polinômios, strings ou valores booleanos.

Hipermatrizes são `mlists`: `mlist(['hm','dims','entries'],sz,v)` onde `sz` é o vetor linha de dimensões e `v` é o vetor coluna de entradas (as primeiras dimensões são armazenadas primeiro)

NOTAS: O número de dimensões de hipermatrizes com tamanhos mais à direita iguais a 1 é automaticamente reduzido. Uma hipermatriz com apenas duas dimensões é automaticamente trocada por uma matriz regular (tipo 1).

# Exemplos

```
a(1,1,1,1:2)=[1 2]
a=[1 2;3 4];a(:,:,2)=rand(2,2)
a(1,1,:)

size(a)

a(:,:,1) //redução de dimensionalidade
type(a(:,:,1))

[a a]
```

# Ver Também

hypermat

# Name

iscell — verifica se um array é um cell (célula)

```
bool = iscell(x)
```

# Parâmetros

x
: variável Scilab

bool
: um booleano

# Descrição

`iscell(x)` retorna %t (verdadeiro) se x é um cell e %f (falso) em caso contrário.

# Exemplos

```
iscell(1)

iscell(cell())

c = cell(1,2);
c(1).entries="Scilab";
c(2).entries=datenum();
iscell(c)
```

# Ver Também

cell, isstruct

# Autor

V.C.

# Name

iscellstr — verifica se uma variável é um cell array de strings

```
bool = iscellstr(x)
```

# Parâmetros

x
: variável Scilab

bool
: um booleano

# Descrição

`iscellstr(x)` retorna verdadeiro se x é um cell array de strings (ou um cell array vazio) e falso em caso contrário.

# Exemplos

```
iscellstr(1)

iscellstr(cell())

iscellstr(cell(3))

strcell = cell(3,1);
strcell(1).entries="Scilab";
strcell(2).entries="iscellstr";
strcell(3).entries="help";
iscellstr(strcell)
```

# Ver Também

cell, iscell, isstruct

# Autor

V.C.

# Name

isstruct — checa se uma variável é um struct array (estrutura)

```
bool = isstruct(x)
```

# Parâmetros

x
: variável Scilab

bool
: um booleano

# Descrição

`isstruct(x)` retorna %t (verdadeiro) se x é um struct array e %f em caso contrário.

# Exemplos

```
isstruct(1)

isstruct(cell())

isstruct(struct("name","Scilab", "version", getversion()))

info.name="Scilab";
info.function="isstruct";
info.module="help";
isstruct(info)
```

# Ver Também

struct, iscell

# Autor

V.C.

# Name

list — objeto do Scilab e definição de lists ("listas")

```
list(a1,....an)
```

# Descrição

Cria um `list` com elementos `ai`'s que são elementos Scilab arbitrários (`matrix, list,...`). O tipo de objetos `list` é 15. `list()` cria o `list` vazio (0 elementos).

# Operações em Listas

extração
: `[x,y,z...]=L(v)` onde `v` é um vetor de índices; `[x,y,z]=L(:)` extrai todos os elementos.

inserção no índice i
: `L(i)=a` (note que não é errado usar `L(i)=a` com *i > 1 + size(L)* mas algumas entradas de lista estão então indefinidas e sua extração levanta um erro).

anexar elemento no fim da lista
: `L($+1)=e`.

anexar elementos no início da lista
: `L(0)=e`. (note que após esta operação e está no índice 1, os elementos iniciais foram movidos para a direita).

deleção
: `L(i)=null()` remove o i-ésimo elemento da lista `L`.

concatenação de duas listas
: `L3 = lstcat(L1,L2)`.

número de elementos de uma lista
: você pode usar tanto `nb_elm = size(L)` ou `nb_elm = length(L)`.

iterações com uma lista
: é possível usar uma lista `L` com um laço ("loop") for: `for e=L,...,end` é um laço com `length(L)` iterações, a variável de laço `e` sendo igual a `L(i)` na i-ésima iteração.

# Observações

O Scilab também possui outros tipos de listas, o tipo tlist (lista com tipo) e o tipo mlist que são úteis para definir um novo tipo de dado com o operador overloading facilities (hypermatrices (hipermatrizes), que são arrays multi-dimensionais no Scilab, são, na verdade, *mlist*).

*struct* do Matlab também estão disponíveis.

# Exemplos

```
l = list(1,["a" "b"])
l(0) = "foo"
l($+1) = "olá"
l(2) = "toto"
l(3) = rand(1,2)
l(3) = null()
lbis = list("gewurtz", "caipirina" ,"debug")
```

```
lter = lstcat(l,lbis)
size(lter) - size(lbis) - size(l)  // deve ser zero
```

## Ver Também

null, lstcat, tlist, insertion, extraction, size, length

# Name

lsslist — definição de função linear de espaço de estados do Scilab

```
lsslist()
lsslist(a1,....an)
```

# Descrição

`lsslist(a1,....an)` é um atalho para `tlist(['lss','A';'B';'C';'X0','dt'], a1,....an)`

Cria um `tlist` com `['lss','A';'B';'C';'X0','dt']` como primeiras entradas e os `ai`'s como próximas entradas, se tiver algum. Não é feita verificação de tipo ou tamanho nos `ai`'s.

# Ver Também

tlist, syslin

# Name

lstcat — concatenação de listas

```
lc=lstcat(l1,..ln)
```

# Parâmetros

li
: lista ou qualquer outro tipo de variável

lc
: uma lista

# Descrição

`lc=lstcat(l1,..ln)` concatena os componentes das `li` listas em uma única lista. Se os `li` são outros tipos de variáveis, eles simplesmente são adicionados à lista resultante.

# Exemplos

```
lstcat(list(1,2,3),33,list('foo',%s))
lstcat(1,2,3)
```

# Ver Também

list

# Name

mlist — objeto Scilab, definição de listas com tipos orientadas matricialmente

```
mlist(typ,a1,....an )
```

# Parâmetros

typ
: vector of character strings

ai
: any Scilab object (`matrix, list,string...`).

# Descrição

Objetos mlist são bastante semelhantes a objetos tlist objects. A única diferença concerne às sintaxes de extração e inserção (extraction e insertion): se `M` é uma mlist, para qualquer índice `i` que não é um campo de nome, `M(i)` não é mais o `i`-ésimo campo da lista

A semântica da sintaxe de extração e inserção deve ser dada por uma função de (sobrecarga) overloading.

A função de overloading para sintaxe de extração `b=a(i1,...,in)` tem a seguinte seqüência de chamamento: `b=%<tipo_de_a>_e_(i1,...,in,a)`

e a sintaxe `[x1,..,xm]=a(i1,...,in)` tem a seguinte seqüência de chamamento: `[x1,..,xm]=%<type_de_a>_e_(i1,...,in,a)`

A função de overloading associada à sintaxe de inserção `a(i1,...,in)=b` tem a seguinte seqüência de chamamento: `a=%<type_de_b>_i_<type_de_a>(i1,...,in,b,a)`.

Os campos de um mlist devem, então, ser designados por seus nomes. Eles também podem ser manipulados usando-se as funções `getfield` e `setfield`.

# Exemplos

```
M=mlist(['V','name','value'],['a','b';'c' 'd'],[1 2; 3 4]);
//Definindo exibição
function %V_p(M),disp(M.name+':'+string(M.value)),endfunction

//Definindo operação de extração
function r=%V_e(varargin)
  M=varargin($)
  r=mlist(['V','name','value'],M.name(varargin(1:$-1)),M.value(varargin(1:$-1))
endfunction
M(2,:) // a segunda entrada do vetor M
M.value

//Definindo operações de inserção
function M=%V_i_V(varargin)
  M=varargin($)
  N=varargin($-1)
  M.value(varargin(1:$-2))=N.value
  M.name(varargin(1:$-2))=N.name
```

```
endfunction
M(1,1)=M(2,2)

function M=%s_i_V(varargin) //inserção de uma matriz regular em uma matriz V
  M=varargin($)
  N=varargin($-1)
  M.value(varargin(1:$-2))=N
  M.name(varargin(1:$-2))=emptystr(N)
endfunction
M(1,1)=44

//caso de tlists
M=tlist(['V','name','value'],['a','b';'c' 'd'],[1 2; 3 4]);
M(2)
M(2)='a'+string([1 2;3 4])

M('name')
```

# Ver Também

tlist, list, overloading, getfield, setfield

# Name

rlist — definição de função racional do Scilab

```
rlist()
rlist(a1,....an)
```

# Descrição

`rlist(a1,....an)` é um atalho para `tlist(['r','num';'den','dt'], a1,....an)`

Cria um `tlist` com `['r','num';'den','dt']` como primeira entrada e os `ai`'s como próximas entradas, se tiver algum. Nenhuma verificação de tipo ou tamanho é feita nos `ai`'s.

# Ver Também

tlist, syslin

# Name

setfield — inserção de campos de listas

```
setfield(i,x,l)
```

# Parâmetros

x
: matriz de qualquer tipo

l
: lista, tlist ou mlist

i
: índice de campo, ver "insertion" para mais detalhes.

# Descrição

Esta função é equivalente à sintaxe `l(i)=x` spara inserção de campos, com a única diferença de que também se aplica a objetos `mlist`.

# Exemplos

```
l=list(1,'qwerw',%s)
l(1)='Modificado'
l(0)='Adicionado'
l(6)=['mais um';'adicionado']

a=hypermat([2,2,2],rand(1:2^3));// hipermatrizes são codificadas usando-se mlis
setfield(3,1:8,a);a // ajusta o valor de campo para 1:8
```

# Ver Também

insertion

# Name

struct — criando um struct (estrutura)

```
st=struct(field1,value1,field2,value2...)
```

# Parâmetros

field1, field2, ..
: strings, representam os nomes dos campos

value1, value2, ..
: todos os tipos de dados (double, char, int, ...), representam os valores dos campos

# Descrição

Esta função retorna uma estrutura com campos de nomes `fields1`, `field2`, ..., e valores correspondentes `value1`, `value2`, ...

# Exemplos

```
// criando um dado struct
date=struct('dia',25,'mes' ,'DEZ','ano',2006)

//mudando o mes
date.mes='AGO';

// mudando o ano
date.ano=1973;

//mudando o dia
date.dia=19;

// adicionando um novo campo
date.semana=32
```

# Ver Também

cell

# Name

tlist — objeto do Scilab e definição de listas com tipos ("typed lists")

```
tlist(typ,a1,....an )
```

# Parâmetros

typ
: string ou vetor de strings

ai
: qualquer objeto Scilab (`matrix, list,string...`).

# Descrição

Cria um `typed-list` com elementos `ai`'s. O argumento `typ` especifica o tipo da lista. Tais `typed-list` permitem ao usuário definir novas operações trabalhando nestes objetos através de funções do Scilab. A única diferença entre um `typed-list` e um `list` (lista) é o valor do tipo (16 ao invés de 15).

`typ(1)` espeifica o tipo de lista (string usado para definir operações "soft-coded").

Se especificado, `typ(i)` pode fornecer o nome formal do `i+1` -ésimo elemento.

Operações padrões em `list` operam de modo semelhante para `typed-list`:

Extração : `[x,y,z...]=l(v)` onde `v` é o vetor de índices; `[x,y,z]=l(:)` extrai todos os elementos.

Inserção: `l(i)=a`

Deleção: `l(i)=null()` remove o i-ésimo elemento do `tlist l`.

Exibição.

Além disso, se `typ(2:n+1)` forem especificados, o usuário pode apontar elementos por seus nomes.

Abaixo fornecemos exemplos de onde tlists são usados:

Sistemas lineares são representados por `typed-list` específicos, ex.: um sistema linear `[A,B,C,D]` é representado pelo tlist `Sys=tlist(['lss';'A';'B';'C';'D';'X0';'dt'],A,B,C,D,x0,'c')` e esta lista específica pode ser criada pela função `syslin`.

Sys(2), Sys('A') ou Sys.A é a estado-matriz e Sys('dt') ou Sys.dt é o domínio de tempo

Uma matriz de razões de polinômios `H` é representada por um `typed-list` `H=tlist(['r';'num';'den';'dt'],Num,Den,[])` onde `Num` e `Den` são duas matrizes de polinômios e um sistema linear (ex.: de tempo contínuo) com matriz de transferência `H` possivelmente criada por `syslin('c',H)`.

H(2), H('num') ou H.num é o numerador da matriz de transferência

# Ver Também

null, percent, syslin, list

# Name

fieldnames — returns the tlist, mlist or struct field names

```
f=fieldnames(x)
```

# Parameters

x
: a tlist or an mlist or a struct.

f
: column vector of strings

# Description

This function returns the tlist, mlist, cell or struct field names.

# Examples

```
clear t;
t.a=1;
t.b=2;
fieldnames(t)
fieldnames(1/%s)
fieldnames(tf2ss(1/%s))
```

# See Also

extraction , getfield , tlist , mlist , struct

# Name

isfield — Checks if the given fieldname exists in the structure

```
bool = isfield(s,fieldname)
```

# Parameters

s
: A struct array

fieldname
: A matrix of strings

bool
: A matrix of boolean.

# Description

This function returns true if the specified structure "s" includes the field "field", regardless of the corresponding value.

# Examples

```
s = struct("field_1",123,"field_2",456,"field_4",789)

// Single Fieldname Syntax
isfield( s , "field_1"  )

// Multiple Fieldname Syntax
isfield( s , [ "field_1" "field_2" ; "field_3" "field_4" ] )
```

# See Also

struct , getfield , definedfields

# Parte XI. Shell

# Name

clc — Clear Command Window

```
clc([nblines])
```

# Parameters

nblines
: a double value

# Description

`clc()` clears all input and output from the Command Window.

After using `clc()`, you cannot use the scroll bar to see the history of functions, but still can use the up arrow to recall statements from the command history.

`clc(nblines)` clears `nblines` above cursor current line and move cursor up to this line.

Note that `clc([nblines])` cannot be used under Unix/Linux platforms when Scilab used in no window mode.

# See Also

tohome

# Authors

V.C.

# Name

lines — rows and columns used for display

```
lines([nl [,nc]])
ncl=lines()
```

# Parameters

nl : an integer, the number of lines for vertical paging control. If 0
no vertical paging control is done.

nc
an integer, the number of column of output. Used for formatting output

ncl
a 1x2 vector [nc,nl]

# Description

`lines` handles Scilab display paging.

`lines()` returns the vector `[# columns, # rows]` currently used by Scilab for displaying the results.

`lines(nl)` sets the number of displayed lines (before user is asked for more) to `nl`.

`lines(0)` disables vertical paging

`lines(nl,nc)` changes also the size of the output to `nc` columns.

When Scilab is launched without -nw option, the `lines` parameters are automatically set according to the output window size, these parameters are also automatically modified when the window is resized.

# See Also

disp , print

# Name

prompt — Get/Set current prompt

```
currentprompt = prompt()
[currentprompt,pauselevel] = prompt()
prompt(userprompt)
```

# Parameters

currentprompt
: String: current prompt returned as a character string.

pauselevel
: integer: current pause level.

userprompt
: String: prompt to display for next user input. Then current prompt will be used again.

# Description

`currentprompt = prompt()` gets the current prompt.

`prompt(userprompt)` sets the prompt.

# See Also

pause, input

# Authors

A.C.

# Name

tohome — Move the cursor to the upper left corner of the Command Window

```
tohome()
```

# Description

`tohome()` moves the cursor to the upper-left corner of the Command Window and clears the screen.

You can use the scroll bar to see the history of previous functions.

Note that `tohome()` cannot be used under Windows platforms when Scilab used in no window mode.

# See Also

clc

# Authors

V.C.

# Parte XII. Console

# Name

console — Keyboard Shortcuts in the Console Window

# Description

| | |
|---|---|
| UP or Ctrl+P | recall previous line. |
| DOWN or Ctrl+N | recall next line. |
| F1 | call help. |
| F2 | clear console. |
| F12 | open console box only on Windows. |
| Ctrl+space or TAB | completion : scilab displays a list of all names that start with somes characters. |
| Ctrl + A or HOME | move to beginning of current line. |
| Ctrl + B or LEFT | moves the cursor one character to the left. |
| Ctrl + C | interrupts Scilab if nothing selected in the console, else text selected is sent to clipboard. |
| Ctrl + D or DELETE | deletes the current character. |
| Ctrl + E or END | moves the cursor to the end of command line. |
| Ctrl + F or RIGHT | moves the cursor one character to the right. |
| Ctrl + H or BACKSPACE | deletes the previous character. |
| Ctrl + K | kills command line from cursor to the end. |
| Ctrl + S | select all. |
| Ctrl + U | delete the whole command line. |
| Ctrl + V | do a paste from clipboard. |
| Ctrl + W | delete the last word of the command line. |
| Ctrl + X | Interrupt Scilab |
| Ctrl + LEFT | move left one word. |
| Ctrl + RIGHT | move right one word. |
| Shift + HOME | select from cursor to beginning of statement. |
| Shift + END | select from cursor to end of statement. |
| Double-click | select current word. |

# Parte XIII. Complementação

# Name

completion — returns words that start with the text you pass as parameter.

```
r = completion(beginning_of_a_word)
r = completion(beginning_of_a_word,dictionary)
[functions, commands, variables, macros, graphic_properties, files] = completio
[functions, commands, variables, macros, graphic_properties] = completion(begin
[functions, commands, variables, macros] = completion(beginning_of_a_word)
[functions, commands, variables] = completion(beginning_of_a_word)
[functions, commands] = completion(beginning_of_a_word)
```

# Parameters

r
: a string matrix

beginning_of_a_word
: a string

dictionary
: a string ("functions", "commands", "variables", "macros", "graphic_properties", "files")

functions, commands, variables, macros, graphic_properties, files
: a string matrix

# Description

returns words that start with the text you pass as parameter.

functions: a string matrix of functions name (C gateways). see 'what'.

commands: a string matrix of command words (reserved). see 'what'.

variables: a string matrix of variables names. see 'who'.

macros: a string matrix of macros names. see 'who'.

graphic_properties: a string matrix

files: a string matrix

# Examples

```
r = completion('w')
r = completion('w','functions')
r = completion('w','commands')
r = completion('w','variables')
r = completion('w','macros')
r = completion('w','graphic_properties')
r = completion('w','files')

[functions,commands,variables,macros,graphic_properties,files] = completion('w'
[functions,commands,variables,macros,graphic_properties] = completion('w')
[functions,commands,variables,macros] = completion('w')
[functions,commands,variables] = completion('w')
[functions,commands] = completion('w')
```

## See Also

getscilabkeywords, who, what, libraryinfo, librarieslist

# Parte XIV. Gerenciador de histórico

# Name

addhistory — add lines to current history.

```
addhistory(string)
addhistory(string_matrix)
```

# Parameters

string
: a string

string_matrix
: a string matrix

# Description

add lines to current history.

# Examples

```
addhistory('hello')
addhistory(['hello','Scilab'])
```

# Authors

A.C

# Name

displayhistory — displays current scilab history

```
displayhistory()
```

# Description

displays current scilab history.

# See Also

gethistory

# Authors

A.C

# Name

gethistory — returns current scilab history in a string matrix

```
matstr=gethistory()
line=gethistory(N)
```

# Parameters

matstr
: a string matrix

N
: Nth line in scilab's history

line
: a string

# Description

returns current scilab history in a string matrix.

# See Also

savehistory , loadhistory , resethistory

# Authors

A.C

# Name

gethistoryfile — get filename used for scilab's history

```
filename = gethistoryfile()
```

# Parameters

filename
: file name used for history

# Description

get filename for scilab's history

# Examples

```
gethistoryfile()
```

# Authors

A.C

# Name

historymanager — enable or disable history manager

```
state1=historymanager(state2)
state1=historymanager()
```

# Parameters

state1
: returns history manager state 'on' or 'off'

state2
: 'on' or 'off' set history manager's state

# Description

enable or disable history manager.

# Examples

```

displayhistory()
backupstate=historymanager()
historymanager('off')
displayhistory()
historymanager('on')
loadhistory()
displayhistory()
historymanager(backupstate)
```

# Authors

A.C

# Name

historysize — get number of lines in history

```
nb=historysize()
```

# Parameters

nb
: number of lines in history.

# Description

get number of lines in history.

# Examples

```
historysize()
```

# Authors

A.C

# Name

loadhistory — load a history file

```
loadhistory()
loadhistory(f)
```

# Parameters

f
: file pathname

# Description

load a history file

by default, history filename is SCIHOME+'/.history.scilab'

# Examples

```
loadhistory(SCI+'/session.scilab')
```

# See Also

savehistory , resethistory , gethistory

# Authors

A.C

# Name

removelinehistory — remove the Nth line in history.

```
removelinehistory(N)
```

# Parameters

N
: a line number

# Description

remove the Nth line in history.

# Examples

```
displayhistory()
removelinehistory(historysize()-2)
displayhistory()
```

# Authors

A.C

# Name

resethistory — Deletes all entries in the scilab history.

```
resethistory()
```

# Description

Deletes all entries in the current scilab history.

# See Also

savehistory , loadhistory

# Authors

A.C

# Name

saveafterncommands — Save the history file after n statements are added to the file.

```
saveafterncommands(n)
v = saveafterncommands()
```

# Parameters

n
: a integer, n statements

v
: current value

# Description

Save the history file after n statements are added to the file.

For example, when you select the option and set n to 5, after every 5 statements are added, the history file is automatically saved.

Use this option if you don't want to risk losing entries to the saved history because of an abnormal termination, such as a power failure.

saveafterncommands() returns current value.

0 is default value.

# Examples

```
saveafterncommands(3)
```

# Authors

A.C

# Name

saveconsecutivecommands — Save consecutive duplicate commands.

```
saveconsecutivecommands(boolean_in)
boolean_out = saveconsecutivecommands()
```

# Parameters

boolean_in
: a boolean (%t or %f)

boolean_out
: current value

# Description

Save consecutive duplicate commands.

saveconsecutivecommands(%t) if you want consecutive executions of the same statement to be saved to the history file.

# Examples

```
saveconsecutivecommands()
saveconsecutivecommands(%t)
1
1
2
saveconsecutivecommands(%f)
1
1
2
```

# Authors

A.C

# Name

savehistory — save the current history in a file

```
savehistory()
savehistory(f)
```

# Parameters

f
: file pathname

# Description

save the current history in a file.

by default, history filename is SCIHOME+'/.history.scilab'

# Examples

```
savehistory(SCI+'/session.scilab')
```

# See Also

loadhistory , resethistory , gethistory

# Authors

A.C

# Name

sethistoryfile — set filename for scilab history

```
sethistoryfile(filename)
sethistoryfile()
```

# Parameters

filename
: filename for history

# Description

set filename for scilab history.

sethistoryfile() without parameters will use the default filename (SCIHOME/history.scilab)

# Examples

```
gethistoryfile()
sethistoryfile(gethistoryfile())
```

# Authors

A.C

# Parte XV. IGU

# Índice

1. Tree ..... 1142
uiCreateNode ..... 1143
uiCreateTree ..... 1144
uiDisplayTree ..... 1145
uiDumpTree ..... 1146

# Capítulo 1. Tree

# Name

uiCreateNode — Creation of node (for Scilab Tree)

```
myNode = uiCreateNode(label[, icon[, callback]])
```

# Input parameters

label
: a string matrix which gives the label of the nodes.

icon (optional)
: a string matrix which gives the icon image of the nodes.

callback (optional)
: a string matrix which gives the callback instruction of the nodes.

# Output parameters

myNode
: a node of type Tree

# Description

Creates a node(a node or a leaf) of type Tree.

# Examples

```
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')
```

# See Also

uiCreateTree , uiDumpTree

# Name

uiCreateTree — Creation of a Tree

```
myTree = uiCreateTree(myParentTree, mySubTree1, mySubTree2,...,mySubTreeN)
```

# Input parameters

myParentTree
: a Tree.

mySubTree(s)
: one or many trees

# Output parameters

myTree
: a Tree

# Description

Creates a Tree in which myParentTree will have children(mySubTree1, mySubTree2,...,mySubTreeN).

# Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')

treeNode1 = uiCreateTree(node1, leaf11, leaf12)
treeNode3 = uiCreateTree(node3, leaf31, leaf32)
treeRoot = uiCreateTree( root, treeNode1, node2, treeNode3)

uiDisplayTree(treeRoot)
```

# See Also

uiCreateNode , uiDumpTree

# Name

uiDisplayTree — Printing a Tree in GUI mode

```
uiDisplayTree(tree)
```

# Input parameters

tree
: a Tree.

# Description

Display a tree into the a graphic window.

# Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1')
leaf12 = uiCreateNode('leaf 1.2')
leaf31 = uiCreateNode('leaf 3.1')
leaf32 = uiCreateNode('leaf 3.2')
node1 = uiCreateNode('Node 1')
node2 = uiCreateNode('Node 2')
node3 = uiCreateNode('Node 3')
root = uiCreateNode('Root')

treeNode1 = uiCreateTree(node1, leaf11, leaf12)
treeNode3 = uiCreateTree(node3, leaf31, leaf32)

treeRoot = uiCreateTree( root, treeNode1, node2, treeNode3)

uiDisplayTree(treeRoot)
```

# See Also

uiCreateNode , uiCreateTree , uiDumpTree

# Name

uiDumpTree — Printing a Tree in the console (text mode)

```
uiDumpTree(tree[,b])
```

# Input parameters

tree
: a Tree.

b(optional)
: display features of each node of the tree. By default b is '%F'.

# Description

Display a tree into the console(text mode).

# Examples

```
// We should create nodes(subTrees) before creating trees
leaf11 = uiCreateNode('leaf 1.1', 'iconLeaf1.1', 'callbackLeaf1.1')
leaf12 = uiCreateNode('leaf 1.2', 'iconLeaf1.2', 'callbackLeaf1.2')
leaf31 = uiCreateNode('leaf 3.1', 'iconLeaf3.1', 'callbackLeaf3.1')
leaf32 = uiCreateNode('leaf 3.2', 'iconLeaf3.2', 'callbackLeaf3.2')
node1 = uiCreateNode('Node 1', 'iconNode1', 'callbackNode1')
node2 = uiCreateNode('Node 2', 'iconNode2', 'callbackNode2')
node3 = uiCreateNode('Node 3', 'iconNode3', 'callbackNode3')
root = uiCreateNode('Root', 'iconRoot', 'callbackRoot')

treeNode1 = uiCreateTree(node1, leaf11,leaf12)
treeNode3 = uiCreateTree(node3, leaf31,leaf32)
treeRoot = uiCreateTree(root, node1,node2,node3)

uiDumpTree(treeRoot)
```

# See Also

uiCreateNode , uiCreateTree

# Name

about — show "about scilab" dialog box

```
about()
```

# Description

show "about scilab" dialog box.

# Examples

```
about()
```

# Authors

Allan CORNET

# Name

addmenu — interactive button or menu definition

```
addmenu(button [,submenus] [,action])
addmenu(gwin,button [,submenus] [,action])
```

# Parameters

button
: a character string. The button name. An & can be placed before the character in the name to be used for keyboard shortcut; this character will be underlined on the GUI. Under MacOSX, a sub-menu with the same name is automatically added (no button can be added to the menu bar).

submenus
: a vector of character string. The sub_menus items names

action
: a list with 2 elements action=list(flag,proc_name)

  flag
  : an integer (default value is 0)

  flag==0
  : the action is defined by a scilab instruction

  flag==1
  : the action is defined by a C or Fortran procedure

  flag==2
  : the action is defined by a scilab function

  proc_name
  : a character string which gives the name of scilab variable containing the instruction or the name of procedure to call.

gwin
: integer. The number of graphic window where the button is required to be installed

# Description

The function allows the user to add new buttons or menus in the main window or graphics windows command panels.

If
: `action` argument is not given the action associated with a button must be defined by a scilab instruction given by the character string variable which name is

  + `button` for a main window command

  + `button_gwin` for a graphic window command

If
: `action` argument is set to 0 `proc_name` should be the name of a Scilab string vector. Actions associated with the kth sub_menu must be defined by scilab instructions stored in the kth element of the character string variable.

If
: `action` argument is set to 1 `proc_name` designes a C or Fortran procedure, this procedure may be interfaced in Fortran subroutine default/fbutn.f or dynamically linked with scilab using the `link` function. The C calling sequence is: `(char* button_name, int* gwin,int *k)`

If

`action` argument is set to 2 `proc_name` designes a Scilab function. This function calling sequence should be:

+ `proc_name(k)` for a main window command

+ `proc_name(k,gwin)` for a graphic window command or a main window command

# Examples

```
if (getscilabmode() == "STD") then
  addmenu('foo');
  foo = 'disp(''hello'')';

  addmenu('Hello',['Franck';'Peter'])
  Hello = ['disp(''hello Franck'')';'disp(''hello Peter'')'];

  addmenu('Bye',list(0,'French_Bye'));
  French_Bye = 'disp(''Au revoir'')';
else
  mprintf('This example requires to use scilab with GUI mode.\n');
end

addmenu(0,'Hello',['Franck';'Peter']);
Hello_0 = ['disp(''hello Franck'')';'disp(''hello Peter'')'];

//C defined Callback
// creating Callback code
code=[ '#include ""machine.h""'
       '#include ""sciprint.h""'
       'void foo(char *name, int *win, int *entry)'
       '{'
       '  if (*win==-1) '
       '    sciprint(""menu %s(%i) in Scilab window selected.\n"", name, *entry
       '  else'
       '    sciprint(""menu %s(%i) in window %i selected.\n"", name, *entry+1,
       '}'];
//creating foo.c file
current_dir = pwd();
chdir(TMPDIR);
mputl(code, TMPDIR+'/foo.c');
//creating Makefile
ilib_for_link('foo','foo.c',[],'c');
exec('loader.sce');
chdir(current_dir);
//add menu
addmenu(0,'foo',['a','b','c'],list(1,'foo'));
```

# See Also

setmenu, unsetmenu, delmenu

# Name

clipboard — Copy and paste strings to and from the system clipboard.

```
clipboard("copy",data)
str=clipboard("paste")
clipboard("do","paste")
clipboard("do","copy")
clipboard("do","empty")
clipboard(winnum,"EMF")
clipboard(winnum,"DIB")
```

# Parameters

data
: Scilab variable or data to set as the clipboard contents.

str
: The clipboard contents returned as a Scilab character string.

winnum
: Number of the graphic window to set as the clipboard contents.

# Description

`clipboard("copy",data)` sets the clipboard contents to data. If data is not a character array, the clipboard uses `sci2exp` to convert it to a string.

`str = clipboard("paste")` returns the current contents of the clipboard as a string or as an empty string (""), if the current clipboard contents cannot be converted to a string.

`clipboard("do","paste")`, `clipboard("do","copy")`, `clipboard("do","empty")` performs a paste, copy or empty clipboard.

`clipboard(winnum,"EMF")` copy a graphic window identified by his window's number in the clipboard to EMF format.

`clipboard(winnum,"DIB")` copy a graphic window identified by his window's number in the clipboard to DIB format.

Note that `clipboard` function works only when Scilab used in window mode.

# Authors

A.C.

# Name

close — close a figure

# Parameters

h

integer Handle of the window to close.

# Description

This routine close a tksci figure (toplevel window). If a handle is given, the figure corresponding to this handle is closed. Otherwise, the current (active) figure is closed.

# Examples

```
h=figure();
// creates  figure number 1.
uicontrol( h, 'style','text', ...
              'string','scilab is great', ...
              'position',[50 70 100 100], ...
              'fontsize',15);
// put a clever text in figure 1
figure();
// create figure 2
uicontrol( 'style','text', ...
           'string','Really great', 'position',[50 70 100 100], 'fontsize',15);
// put a text in figure 2
close();
// close the current graphic window (ie fig. 2)
close(h);
// close figure 1
```

# See Also

figure , gcf

# Authors

Bertrand Guiheneuf

# Name

delmenu — interactive button or menu deletion

```
delmenu(button)
delmenu(gwin,button)
```

# Parameters

button
: a character string. The button name. On Windows operating systems (not X_window), an & should be placed before the character in the name used for keyboard shortcut; this character is underlined on the GUI.

gwin
: integer. The number of graphic window where the button is required to be installed

# Description

The function allows the user to delete buttons or menus create by `addmenu` in the main or graphics windows command panels. Predefined buttons on Scilab graphic windows can also be deleted.

If possible, it is better to delete first the latest created button for a given window to avoid gaps in command panels.

# Examples

```
addmenu('foo')
delmenu('foo')
```

# See Also

setmenu , unsetmenu , addmenu

# Name

exportUI — Call the file export graphical interface

```
exportUI(figId)
exportUI(fig)
```

# Parameters

figId
: integer, Id of the figure to export.

fig
: Figure handle, handle of the figure to export.

# Description

`exportUI` routine call the graphical interface dedicated in exporting a graphic window into an image file.

# See Also

xs2jpg , xs2eps , xs2png , xs2svg , xs2pdf

# Authors

Jean-Baptiste Silvy

# Name

figure — create a figure

```
f = figure(num);
f = figure("PropertyName1", Propertyvalue1, ..., ..., "PropertyNameN", Prop
```

# Description

This routine creates a figure. If an ID is given, the figure corresponding to this ID is created. Otherwise, the window is created with the first free ID, that is the lowest integer not already used by a window.

# Parameters

num
: ID of the window to create. If not specified, the first free ID is used.

PropertyName{1, ..., N}
: character string name of a property to set. One of the property names listed below.

PropertyValue{1, ..., N}
: scilab object value to give to the corresponding property.

f
: handle of the newly created window.

# Properties

BackgroundColor
: [1,3] real vector or string Background color of the figure. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B"

Figure_name
: character string, allows to set the title of the figure.

ForegroundColor
: [1,3] real vector or string Foreground color of the figure. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B"

Position
: allows to control the geometrical aspect of the figure. It is a [1,4] real vector [x y width height] where the letters stand for the x location of the top left corner, the y location of the top left corner, the width and the height of the virtual graphics window (the part of the figure which contains uicontrols and graphics). See the **axes_size** property description in figure properties help page. One can also set this property by giving a string where the fields are separated by a "|", ie "x|y|width|height".

Tag
: string this property is generally used to identify the figure. It allows to give it a "name". Mainly used in conjontion with findobj().

Userdata
: this can be used to associate some Scilab objects to a fugure.

# Examples

```
// Create figure having figure_id==3
h=figure(3);
// Add a text uicontrol in figure 3
uicontrol(h, "style", "text", ...
             "string", "This is a figure", ...
             "position", [50 70 100 100], ...
             "fontsize",15);

// Create figure having figure_id==1
figure();
// Add a text uicontrol in figure 1
uicontrol("style", "text", ...
          "string", "Another figure", ...
          "position", [50 70 100 100], ...
          "fontsize", 15);

// Close current figure (ie figure 1)
close();
// close figure 3
close(h);
```

# See Also

close , gcf

# Authors

Bertrand Guiheneuf

V.C.

# Name

findobj — find an object with specified property

```
h =  findobj(propertyName, propertyValue)
```

# Parameters

propertyName
: string character Name of the property to test (case unsensitive).

propertyValue
: string character specify the value the tested propoerty should be equal to (case sensitive).

h
: handle of the found object.

# Description

This routine is currently used to find objects knowing their 'tag' property. It returns handle of the first found object which property *propertyName* is equal to *propertyValue*. If such an object does not exist, the function returns an empty matrix.

# Examples

```
// Create a figure
h=figure();
// Put a text in the figure
uicontrol(h, "style","text", ...
             "string","This is a figure", ...
             "position",[50 70 100 100], ...
             "fontsize",15, ...
             "tag","Alabel");
// Find the object which "tag" value is "Alabel"
lab=findobj("tag","Alabel");
disp("The text of the label is """+lab.string+""""");
// Close the figure
close();
```

# See Also

uicontrol , uimenu , set , get

# Authors

Bertrand Guiheneuf

V.C.

# Name

gcbo — Handle of the object whose callback is executing.

```
gcbo
```

# Description

`gcbo` is a Scilab variable automatically created each time a callback in executed. This variable is initialised using getcallbackobject.

`gcbo` does not exists in Scilab environment if no callback is currently executed.

You can use `gcbo` in callback functions particularly if you write a single callback function for multiple objects, it helps you to know which object received a user action.

# See Also

getcallbackobject

# Authors

Vincent COUVERT

# Name

getcallbackobject — Return the handle of the object whose callback is executing.

```
h = getcallbackobject()
```

# Parameters

h

Handle: the handle of the object whose callback is executing.

# Description

`getcallbackobject` is used to automatically create Scilab variable called gcbo each time a callback in executed.

`getcallbackobject` returns [] if no callback is currently executed.

# See Also

gcbo

# Authors

Vincent COUVERT

# Name

getinstalledlookandfeels — returns a string matrix with all Look and Feels.

```
lnf=getinstalledlookandfeels()
```

# Parameters

lnf
: a string matrix.

# Description

returns a string matrix with all Look and Feels that you can use.

# Examples

```
getinstalledlookandfeels()
```

# See Also

setlookandfeel , getlookandfeel

# Authors

Allan CORNET

# Name

getlookandfeel — gets the current default look and feel.

```
lnf=getlookandfeel()
```

# Parameters

lnf
: a string with current look and feel.

bok
: a boolean.

# Description

Gets the current default look and feel.

# Examples

```
currentlnf = getlookandfeel();

// Look and feel CDE/Motif
setlookandfeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel")

sleep(3000);

// Look and feel métal
setlookandfeel("javax.swing.plaf.metal.MetalLookAndFeel")

sleep(3000);

setlookandfeel(currentlnf)
```

# See Also

getinstalledlookandfeels , setlookandfeel

# Authors

Allan CORNET

# Name

getvalue — xwindow dialog for data acquisition

```
[ok,x1,..,x14]=getvalue(desc,labels,typ,ini)
```

# Parameters

desc
: column vector of strings, dialog general comment

labels
: n column vector of strings, `labels(i)` is the label of the ith required value

typ
: `list(typ_1,dim_1,..,typ_n,dim_n)`

  typ_i
  : defines the type of the ith value, may have the following values:

    "mat"
    : for constant matrix

    "col"
    : for constant column vector

    "row"
    : for constant row vector

    "vec"
    : for constant vector

    "str"
    : for string

    "lis"
    : for list

  dim_i
  : defines the size of the ith value it must be a integer or a 2-vector of integer, -1 stands for undefined dimension

ini
: n column vector of strings, `ini(i)` gives the suggested response for the ith required value

ok
: boolean ,%t if ok button pressed, %f if cancel button pressed

xi
: contains the ith value if ok=%t. If left hand side has one more `xi` than required values the last `xi` contains the vector of answered strings.

# Description

This function encapsulate `x_mdialog` function with error checking, evaluation of numerical response, ...

# Remarks

All valid expressions can be used as answers; for matrices and vectors `getvalues` automatically adds [ ] around the given answer before numeric evaluation.

# Examples

```
labels=["magnitude";"frequency";"phase    "];
[ok,mag,freq,ph]=getvalue("define sine signal",labels,...
     list("vec",1,"vec",1,"vec",1),["0.85";"10^2";"%pi/3"])
```

# See Also

x_mdialog , x_matrix , x_dialog

# Authors

S. Steer ; ;

# Name

messagebox — Open a message box.

```
[btn] = messagebox(msg)
[btn] = messagebox(msg, msgboxtitle)
[btn] = messagebox(msg, msgboxtitle, msgboxicon)
[btn] = messagebox(msg, msgboxtitle, msgboxicon)
[btn] = messagebox(msg, msgboxtitle, msgboxicon, buttons)
[btn] = messagebox(msg, msgboxtitle, msgboxicon, buttons, ismodal)
```

# Parameters

msg
: Matrix of strings: the message box displays each entry of this matrix (one entry per line).

msgboxtitle
: String: the title of the message box (default value is "Scilab Message").

msgboxicon
: String: the name of the icon to be displayed in the message box, its possible values are:

- "error"
- "hourglass"
- "info"
- "passwd"
- "question"
- "warning"
- "scilab": default icon

buttons
: 1xn vector of strings: the names of the buttons to be displayed in the message box. By default, only one button is displayed with label "OK".

modal
: String: "modal" to create a modal dialog, any other string to create a non-modal dialog. Please note that "modal" can replace any of the other input arguments except msg (See examples).

btn
: Scalar: number of the button that the user pressed (1 is the leftmost button) for a modal dialog, 0 else.

# Description

Creates a dialog window to display a message waiting or not for a user action.

# Examples

```
// Simple example
messagebox("Single line message")
```

```
// Multi line message with title
messagebox(["Multi-line" "message"], "User defined title")

// Icon specified by th euser
messagebox("An error message", "Error", "error")

// Buttons labels + "modal" replaces title
messagebox("Have you seen this beautiful message", "modal", "info", ["Yes" "No"

// "modal" given as fifth input argument
messagebox("An error message", "Error", "error", ["Continue" "Stop"], "modal")
```

# Authors

Vincent COUVERT

# Name

printfigure — Opens a printing dialog and prints a figure.

```
printfigure(figid)
status = printfigure(figid)
```

# Parameters

figid
: Real: the id of the figure to be printed.

status
: Boolean: *%T* if the printing succeeds, *%F* otherwise.

# Description

This function opens a dialog to select a printer, printing options... and then prints the figure.

# Examples

```
plot2d();
printfigure(get(gcf(), "figure_id"));
```

# See Also

toprint , printsetupbox

# Authors

V.C.

# Name

printsetupbox — Display print dialog box.

```
printsetupbox()
status=printsetupbox()
```

# Parameters

status
: Boolean: *%T* if the user clicked on the `OK` button, *%F* otherwise.

# Description

Displays the built-in printing dialogbox and configure the printer.

# See Also

toprint , printfigure

# Authors

A.C

# Name

progressionbar — Draw a progression bar

```
winId=progressionbar(mes)
progressionbar(winId[,mes])
```

# Parameters

mes
: string, message to display.

winId
: integer greater than 0, window identificator.

# Description

`progressionbar(mes)` create a new progression bar, return window identificator.

`progressionbar(winId[,mes])` update the progression bar identificated as winId.

# Examples

```
winId=progressionbar('Do something');
realtimeinit(0.3);
for j=0:0.1:1,
  realtime(3*j);
  progressionbar(winId);
end
winclose(winId);
```

# Authors

Jaime Urzua

# Name

root_properties — description of the root object properties.

# Description

The root object is a virtual object used to get the computer screen properties. Use `get` function with *0* as first argument to access its properties.

Root properties

screensize_px:
: The screen size in pixels.

screensize_pt:
: The screen size in points.

screensize_mm:
: The screen size in millimeters.

screensize_cm:
: The screen size in centimeters.

screensize_in:
: The screen size in inches.

screensize_norm:
: The normalized screen size.

screendepth:
: The number of bits used to encode colors.

# Examples

```
get(0, "screensize_px")
get(0, "screendepth")
```

# See Also

get

# Author

Vincent COUVERT

# Name

setlookandfeel — sets the current default look and feel.

```
bok=setlookandfeel()
bok=setlookandfeel(lnf)
```

# Parameters

lnf
: a string with a look and feel.

bok
: a boolean.

# Description

Sets the current default Look and Feel.

setlookandfeel() without parameter set system default look and feel.

# Examples

```
currentlnf = getlookandfeel();

// Look and feel Windows Classic
setlookandfeel("com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel")

// Look and feel Windows
setlookandfeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel")

sleep(3000);

// Look and feel CDE/Motif
setlookandfeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel")

sleep(3000);

// Look and feel GTK+
setlookandfeel("com.sun.java.swing.plaf.gtk.GTKLookAndFeel")

sleep(3000);

// Look and feel métal
setlookandfeel("javax.swing.plaf.metal.MetalLookAndFeel")

sleep(3000);

// Look and feel Macintosh
setlookandfeel("it.unitn.ing.swing.plaf.macos.MacOSLookAndFeel")

// System default look and feel

setlookandfeel()

sleep(3000);
```

```
// restore previous look and feel
setlookandfeel(currentlnf)
```

## See Also

getinstalledlookandfeels , getlookandfeel

## Authors

Allan CORNET

# Name

setmenu — interactive button or menu activation

```
setmenu(button [,nsub])
setmenu(gwin,button [,nsub])
```

# Parameters

button
: a character string. The button name

gwin
: integer. The number of graphic window where the button is installed

nsub
: integer. The number of submenu to de-activate (if any). If button has no sub-menu, `nsub` is ignored

# Description

The function allows the user to make active buttons or menus created by `addmenu` in the main or graphics windows command panels.

# Examples

```
addmenu('foo')   // New button made in main scilab window
unsetmenu('foo') // button foo cannot be activated (grey string)
setmenu('foo')   // button foo can be activated (black string)
```

# See Also

delmenu , unsetmenu , addmenu

# Name

toolbar — show or hide a toolbar

```
state1=toolbar(winnum,state2)
state1=toolbar(winnum)
```

# Parameters

state1
: returns toolbar's state 'on' or 'off'

winum
: window's number (-1: Scilab console window)

state2
: 'on' or 'off' set toolbar's state

# Description

show or hide a toolbar.

# Examples

```
toolbar(-1,'off')
state=toolbar(-1,'on')

plot3d();
h=gcf();
toolbar(h.figure_id,'off')
```

# Authors

Allan CORNET
Vincent COUVERT

# Name

toprint — Send text or figure to the printer.

```
toprint(filename)
toprint(linestoprint,pageheader)
toprint(figid)
toprint(figid,output)
status = toprint(filename)
status = toprint(linestoprint,pageheader)
status = toprint(figid)
status = toprint(figid,output)
```

# Parameters

filename
: String: path of the text file to be printed.

linestoprint
: String matrix: text to be printed, each entry is a line in printed pages.

pageheader
: String: header of printed pages.

figid
: Real: the id of the figure to be printed.

output
: String: printing output type, must be *"pos"* for PostScript or *"gdi"* for Bitmap format (*"gdi"* by default).

status
: Boolean: *%T* if the printing succeeds, *%F* otherwise.

# Description

Prints a text file, Scilab character strings or figure.

# Examples

```
toprint(SCI+"/etc/scilab.start");
toprint(['Test','toprint primitive'],'Scilab page header');
scf(4);
plot();
toprint(4);
toprint(4,"pos");
```

# See Also

printfigure , printsetupbox

# Authors

A.C.

V.C.

# Name

uicontrol — create a Graphic User Interface object

```
h = uicontrol(PropertyName,PropertyValue,...)
h = uicontrol(parent,PropertyName,PropertyValue,...)
h = uicontrol(uich)
```

# Description

This routine creates an object in a figure.

If the handle of the figure is given (as the first parameter), the uicontrol is created in this figure. If no handle is given, the uicontrol is created in the current figure (which may be obtained with a call to `gcf()`). If there is no current figure, then one is created before the creation of the uicontrol.

Then when the control is created, the properties given as parameters are set with the corresponding values. It is equivalent to create the uicontrol, and then set its properties with the `set()` command. Nevertheless, it generally more efficient to set the properties in the call to `uicontrol()`. This is particularly true concerning the "Style" property. Indeed, the default value for this property is "Pushbutton". So if you do not set it at creation time, a button will be created, and will be transformed to another uicontrol when you call the `set(h, "Style", ... )` instruction. Scilab and all the graphic objects communicate through the property mechanism. Thus, to create adapted uicontrol, one has to know the use of the property fields.

`h = uicontrol(PropertyName, PropertyValue,...)` creates an uicontrol and assigns the specified properties and values to it. It assigns the default values to any properties you do not specify. The default uicontrol style is a "Pushbutton". The default parent is the current figure. See the Properties section for information about these and other properties.

`h = uicontrol(parent, PropertyName, PropertyValue,...)` creates a uicontrol in the object specified by the handle, parent. If you also specify a different value for the Parent property, the value of the Parent property takes precedence. parent is the handle of a figure.

`h = uicontrol(uich)` gives focus to the uicontrol specified by uich.

# Properties

BackgroundColor
: [1,3] real vector or string

  Background color of the uicontrol. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B".

Callback
: String

  Instruction evaluated by the Scilab interpreter when an uicontrol is activated. (for example when you click on a button).

Enable
: {on} | off

  Enable or disable the uicontrol. If this property is set to `"on"` (default), the uicontrol is operational, but if this property is set to `"off"`, the uicontrol will not respond to the mouse actions and will be grayed out.

FontAngle
: {normal} | italic | oblique

For a control containing some text, this property sets the slant of the font.

FontSize
: Scalar

For a control containing some text, this property sets the size of the font in FontUnits.

FontUnits
: {points} | pixels | normalized

For a control containing some text, this property sets the units with which the FontSize is specified.

FontWeight
: light | {normal} | demi | bold

For a control containing some text, this property sets the weight of the used font.

FontName
: String

Used to choose the name of the font selected to display the text of the control.

ForegroundColor
: [1,3] real vector or string

Foreground color of the uicontrol. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B".

HorizontAlalignment
: left | {center} | right

Set text horizontal alignment in the uicontrol. This property has only effect with Text, Edit and Check Boxes.

ListboxTop
: Scalar

For a ListBox, this property tells which item of the list appears on the first line of the visible area of the list.

Max
: Scalar

Specifies the largest value the "Value" property can be set to. It has however different meaning on each uicontrol:

- CheckBoxes: Max is the value the "Value" property take when control is checked.
- Sliders: Maximum value of the slider.
- ListBoxes: if (Max-Min)>1 the list allows multiple selection, Otherwise not.

Min
: Scalar

Specifies the lowest value the "Value" property can be set to. It has however different meaning on each uicontrol:

- CheckBoxes: Min is the value the "Value" property take when control is unchecked.
- Sliders: Minimum value of the slider.
- ListBoxes: if (Max-Min)>1 the list allows multiple selection, Otherwise not.

Parent
: Handle

Handle of the uicontrol parent. Changing this property allows to move a control from a figure to another.

Path
: This property is no more supported.

Position
: [1,4] real vector or string.

This property is used to set or get the geometrical configuration of a control. It is a vector [x y w h] where the letters stand for the x location of the left bottom corner, the y location of the left bottom corner, the width and the height of the uicontrol or a character string where each value is separated by a "|", ie "x|y|w|h". The units are determined by the "Units" property.

The width and height values determine the orientation of sliders. If width is greater than height, then the slider is oriented horizontally, otherwise the slider is oriented vertically.

Relief
: flat | groove | raised | ridge | solid | sunken

Appearance of the border of the uicontrol:

- PushButtons: the default value for "Relief" property is "raised".
- Edits: the default value for "Relief" property is "sunken".
- Other styles: the default value for "Relief" property is "flat".

SliderStep
: [1,2] real vector

[small big], the small step represents the movement achieved when clicking on the slider trough or tapping on the keyboard arrows (when the slider has focus); the big step is the amount moved when using Ctrl-keyboard-arrows. If the big step is omitted, it is defaulted to 1/10 of the scale.

String
: String.

This property represents the text appearing in a uicontrol (Except for Frame and Slider styles). For ListBoxes and PopupMenus, the value can be a vector of string or a string where the items are separated by a "|". For Text uicontrols, this string can contain HTML code to format the text.

Style
: {pushbutton} | radiobutton | checkbox | edit | text | slider | frame | listbox | popupmenu

Style of the uicontrol. Here is a short description of each one:

- Pushbutton: a rectangular button generally used to run a callback.
- Radiobutton: a button with to states. RadioButtons are intended to be mutually exclusive (Your code must implement mutually exclusive behavior).
- Checkbox: a button with to states (Used for multiple independent choices).

- Edit: an editable string zone.
- Text: a text control (generally static).
- Slider: a scale control, that is a scrollbar use to set values between in range with the mouse.
- Frame: a control representing a zone used to group related controls.
- Listbox: a control representing a list of items that can be scrolled. The items can be selected with the mouse.
- Popupmenu: a button which make a menu appear when clicked.

Tag
: String

This property is generally used to identify the control. It allows to give it a "name". Mainly used in conjunction with `findobj()`.

Units
: {points} | pixels | normalized

Set the units used to specify the "Position" property.

Userdata
: Scilab data

This can be used to associate some Scilab objects (string,string matrix, matrix mxn) to an uicontrol.

Value
: Scalar or vector

Value of the uicontrol. The exact meaning depends on the style of the uicontrol:

- CheckBoxes, Radio buttons: value is set to Max (see above) when on and Min when off.
- ListBoxes, PopupMenus: value is a vector of indexes corresponding to the indexes of the selected entries in the list. 1 is the first item of the list.
- Sliders: value indicated by the slider bar.

Verticalalignment
: top | {middle} | bottom

Set text vertical alignment in the uicontrol. This property has only effect with Text and CheckBoxes styles.

Visible
: {on} | off

Set the visibility of the uicontrol. If this property is set to `"on"` (default), the uicontrol is visible, but if this property is set to `"off"`, the uicontrol will not appear in its parent figure.

# Examples

```
f=figure();
// create a figure
h=uicontrol(f,'style','listbox', ...
 'position', [10 10 150 160]);
```

```
// create a listbox
set(h, 'string', "item 1|item 2|item3");
// fill the list
set(h, 'value', [1 3]);
// select item 1 and 3 in the list
close(f);
// close the figure
```

# See Also

figure, set, get, uimenu

# Authors

Bertrand Guiheneuf

Vincent Couvert

# Name

uigetcolor — Opens a dialog for selecting a color.

```
uigetcolor()
RGB = uigetcolor([title])
RGB = uigetcolor([title,] defaultRGB)
RGB = uigetcolor([title,] defaultRed, defaultGreen, defaultBlue)
[R, G, B] = uigetcolor([title])
[R, G, B] = uigetcolor([title,] defaultRGB)
[R, G, B] = uigetcolor([title,] defaultRed, defaultGreen, defaultBlue)
```

# Parameters

title
: String: Optional argument, the title to display in the dialog. Default value is "Color Chooser".

defaultRGB
: 1x3 vector: the default values for Red, Green and Blue values given as a vector [red, green, blue].

defaultRed
: Scalar: the default value for red.

defaultGreen
: Scalar: the default value for green.

defaultBlue
: Scalar: the default value for blue.

RGB
: 1x3 vector: the values for Red, Green and Blue values given as a vector [red, green, blue] or [] if the user cancels.

R
: Scalar: the value for red or [] if the user cancels.

G
: Scalar: the value for green or [] if the user cancels.

B
: Scalar: the value for blue or [] if the user cancels.

# Description

Creates a dialog window for selecting a color. All (default and returned) values must be in the interval [0 255].

# Examples

```
uigetcolor()
[R, G, B] = uigetcolor([255 128 0])
RBG = uigetcolor(0, 128, 255)
RBG = uigetcolor("My color chooser", 0, 128, 255)
```

## See Also

getcolor

## Authors

Vincent COUVERT

# Name

uigetdir — dialog for selecting a directory

```
directory = uigetdir()
directory = uigetdir(start_path [,title])
```

# Parameters

start_path
: a character string which gives the initial directory used for search. By default uigetdir uses current working directory.

title
: the title for the uigetdir window.

directory
: is the user selected directory if user answers "Ok" or the " " string if user cancels.

# Description

Creates a dialog window for selecting a directory

# Examples

```
uigetdir()
uigetdir("SCI/modules/")
uigetdir("SCI/modules/", "Choose a directory")
```

# See Also

uigetfile , uiputfile

# Name

uigetfile — dialog window to get a file(s) name(s), path and filter index

```
[FileName[,PathName[,FilterIndex]]]=uigetfile([file_mask[,dir[,boxTitle[,multip
PathFileName=uigetfile([file_mask[,dir[,boxTitle[,multiple]]]])
```

# Input parameters

file_mask
: a string matrix which gives the file masks to use for file selection. `file_mask` is written with Unix convention. The default value is '*'.

  we can also add descriptions for masks, for example ["*.x*","X files";"*.bin","BIN files"].

dir
: a character string which gives the initial directory used for file search. By default uigetfile uses the previously selected directory.

boxTitle
: a character string which gives the title of the uigetfile window. By default uigetfile's title is 'uigetfile'.

multipleSelection
: a boolean which allows to load only one file if it is at '%f' (false) or multiple files if it is at '%t" (true). By default uigetfile's multiple file selection is not enable.

# Output parameters

FileName
: matrix of string which give the user selected file(s) (path + file(s) name(s)) if user answers "Ok" or the " " string if user answers "Cancel".

PathName
: is the user selected file(s) path if user answers "Ok" or the " " string if user answers "Cancel".

FilterIndex
: is the user selected filter index on the list box if user answers "Ok" or '0' string if user answers "Cancel"

# Description

Creates a dialog window for file(s) selection.

# Comments

On Windows, java component used by uigetfile browse also .zip archive then it is very slow with big .zip files.

To disable, this feature:

if MSDOS then unix("REGSVR32 /u %WINDIR%\System32\zipfldr.dll") ;end

To re-enable,

if MSDOS then unix("REGSVR32 %WINDIR%\System32\zipfldr.dll") ;end

# Examples

```
uigetfile(["*.bin";"*.sce";"*.cos*"])
uigetfile(["*.sci";"*.bin"],"SCI/modules/gui/macros/")
uigetfile(["*.sc*";"*.bin"],"SCI/modules/gui/macros/")
uigetfile(["*.x*","X files";"*.bin","BIN files"],"SCI/modules/gui/macros/")
uigetfile(["*.sce";"*.bin"],"SCI/modules/gui/macros/", "Choose a file name"", %
uigetfile(["*.sce";"*.bin"],"SCI/modules/gui/macros/", "Choose a file name"", %
```

# See Also

uiputfile, uigetdir, x_dialog, file, read, write, exec

# Name

uigetfont — Opens a dialog for selecting a font.

```
uigetfont()

[fontname [,fontsize [,bold [,italic]]]] = uigetfont([defaultfontname [,defaultf

[fontname ,fontsize ,bold ,italic] = uigetfont(defaultfontname ,defaultfontsize
```

# Parameters

defaultfontname
: String: the default font name to select in the dialog.

defaultfontsize
: Scalar: the default font size to select in the dialog.

defaultbold
: Boolean: the default bold attribute in the dialog (%T for bold font, %F otherwise).

defaultitalic
: Boolean: the default italic attribute in the dialog (%T for bold font, %F otherwise).

fontname
: The selected font name ("" if the user cancels).

fontsize
: The selected font size ([] if the user cancels).

bold
: %T if bold attribute has been selected, %F otherwise ([] if the user cancels).

italic
: %T if italic attribute has been selected, %F otherwise ([] if the user cancels).

# Description

Creates a dialog window for selecting a font.

# Examples

```
uigetfont()
uigetfont("arial")
uigetfont("arial", 24)
uigetfont("arial", 24, %T)
uigetfont("arial", 24, %T, %F)
```

# See Also

getfont

# Authors

Vincent COUVERT

# Name

uimenu — Create a menu or a submenu in a figure

```
h=uimenu([prop1,val1] [,prop2, val2] ...)
h=uimenu(parent,[prop1, val1] [,prop2, val2] ...)
```

# Parameters

parent
: integer Handle of menu's parent

prop{1, 2 ...}
: string character name of a property to set up

val{1, 2 ...}
: scilab object value to affect to the corresponding property

h
: integer handle of the corresponding menu

# Description

This allows to create menus in a figure. If `parent` is a figure, then the menu item will be added to the menu bar of the figure. If `parent` is a menu item , then the new item will be added to the parent item, allowing to create cascaded submenu. To create a customized menu, you can use the properties listed below:

# Properties

Callback
: String

  Instruction evaluated by the Scilab interpreter when the menu is activated. Under MacOSX, the callback will not be executed for a "button menu" (a menu without children), you must specify at least a child.

Enable
: {on} | off

  Enable or disable the menu. If this property is set to `"on"` (default), the menu is operational, but if this property is set to `"off"`, the menu will not respond to the mouse actions and will be grayed out.

Checked
: {on} | off

  Menu check indicator. Setting this property to "on" (respectively "off") places (respectively removes) a check mark next to the corresponding menu item. This option can be used to create menus that indicate the state of a particular option.

  Please note that a standard menu (without the option "Checked" set) has no mechanism to become a menu which will be checked by a user action and conversely.

  This property is ignored for parent menus.

ForegroundColor
: [1,3] real vector or string

Foreground color of the uimenu (font color). A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B".

Label
: String.

This property represents the text appearing in the menu.

Tag
: String

This property is generally used to identify the menu. It allows to give it a "name". Mainly used in conjunction with `findobj()`.

Visible
: {on} | off

Set the visibility of the uimenu. If this property is set to `"on"` (default), the uimenu is visible, but if this property is set to `"off"`, the uimenu will not appear in its parent figure.

# Examples

```
f=figure('position', [10 10 300 200]);
// create a figure
m=uimenu(f,'label', 'windows');
// create an item on the menu bar
m1=uimenu(m,'label', 'operations');
m2=uimenu(m,'label', 'quit scilab', 'callback', "exit");
//create two items in the menu "windows"
m11=uimenu(m1,'label', 'new window', 'callback',"xselect()");
m12=uimenu(m1,'label', 'clear  window', 'callback',"clf()");
// create a submenu to the item "operations"
close(f);
// close the figure
```

# See Also

figure, uicontrol, set, get

# Authors

Bertrand Guiheneuf

# Name

uiputfile — Open standard dialog box for selecting and saving file.

```
[FileName[,PathName[,FilterIndex]]]=uiputfile([file_mask[,dir[,boxTitle]]])
PathFileName=uiputfile([file_mask[,dir[,boxTitle]]])
```

# Input parameters

file_mask
: a string matrix which gives the file masks to use for file selection. `file_mask` is written with Unix convention. The default value is '*'.

  we can also add descriptions for masks, for example ["*.x*","X files";"*.bin","BIN files"].

dir
: a character string which gives the initial directory used for file search. By default uiputfile uses the previously selected directory.

boxTitle
: a character string which gives the title of the uiputfile window. By default uiputfile's title is 'uiputfile'.

# Output parameters

FileName
: string which give the user selected file (path + file name) if user answers "Ok" or the " " string if user answers "Cancel".

PathName
: is the user selected file path if user answers "Ok" or the " " string if user answers "Cancel".

FilterIndex
: is the user selected filter index on the list box if user answers "Ok" or '0' string if user answers "Cancel"

# Description

Creates a dialog window for file saving.

# Examples

```
uiputfile(["*.bin";"*.sce";"*.cos*"])
uiputfile(["*.sci";"*.bin"],"SCI/modules/gui/macros/")
uiputfile(["*.sc*";"*.bin"],"SCI/modules/gui/macros/")
uiputfile(["*.x*","X files";"*.bin","BIN files"],"SCI/modules/gui/macros/")
uiputfile(["*.sce";"*.bin"],"SCI/modules/gui/macros/", "Choose a file name");
```

# See Also

uigetfile , uigetdir

# Name

unsetmenu — interactive button or menu or submenu de-activation

```
unsetmenu(button,[nsub])
unsetmenu(gwin,button,[nsub])
```

# Parameters

button
: a character string. The button name

gwin
: integer. The number of graphic window where the button is installed

nsub
: integer. The number of submenu to de-activate (if any). If button has no sub-menu, `nsub` is ignored

# Description

The function allows the user to desactivate buttons or menus created by `addmenu` in the main or graphics windows command panels.

# Examples

```
addmenu('foo')
unsetmenu('foo')
unsetmenu('File',2)
```

# See Also

delmenu , setmenu , addmenu

# Name

usecanvas — Get/Set the main component used for Scilab graphics.

```
[canvasused] = usecanvas([usecanvasfordisplay]);
```

# Parameters

canvasused
: Boolean:

- `%T` if a "GLCanvas" is used for graphics display (Mixing uicontrols and graphics **not available**).
- `%F` if a "GLJPanel" is used for graphics display (Mixing uicontrols and graphics available).

usecanvasfordisplay
: Boolean:

- `%T` to use a "GLCanvas" for graphics display (Mixing uicontrols and graphics **not available**).
- `%F` to use a "GLJPanel" for graphics display (Mixing uicontrols and graphics available).

# Description

Scilab uses a "GLJPanel" (a Swing OpenGL component) to display graphics (plot3d, plot, ...). This component uses some high level OpenGL primitives which are not correctly supported on some platforms (depending on the operating system, video cards, drivers ...)

"GLCanvas" (AWT + OpenGL) is an alternative component provided by the Java Framework. Scilab can use it to render graphics. **However, using this component disables some capabilities such as mixing plots and uicontrols (see demo GUI/UIcontrol2). That is why it is not the default behavior.**

In some particular cases, the use of the "GLCanvas" component is forced when Scilab starts (a warning message is displayed when a graphics function is used for the first time), here is a list of these cases:

| **Operating System** | **Video Card** | **Details** |
|---|---|---|
| 64-bits Windows | All | When Scilab is used in a remote session. |
| Linux | NVIDIA Card | With free drivers. |
| ATI Card | With free drivers or ATI-drivers with version < 8.52.3 (Installer version < 8.8 / OpenGL version < 2.1.7873). | |
| INTEL Card | With Direct Rendering activated. | |

You can also dynamically activate this component through Scilab using `usecanvas`:

- `usecanvas(%T)` will use "GLCanvas" for plot rendering.
- `usecanvas(%F)` will use "GLJPanel" for plot rendering. If your configuration is known as a one having problems with "GLJPanel" (See table above), a warning message will be displayed.

If you believe your configuration is able to use the "GLJPanel" and Scilab automatically forces the use of "GLCanvas", you can test your configuration by swithing to "GLJPanel" (`usecanvas(%F)`) and

try to plot something (`plot3d()` for example). If Scilab graphics work, please inform us about it by sending an email to `scilab.support@scilab.org` and giving us your Operating System/Video Card/Video Card driver version: this will help use to improve future versions of Scilab.

# Technical Aspects

Since version 5.0, Scilab is doing an advanced use of JOGL (the Java Binding for the OpenGL), which is using the Java2D OpenGL Pipeline. For performance reasons, we use the Java2D OpenGL Pipeline. From a more technical aspect, it uses the internal buffer of the graphic cards called `pbuffer`.

Problems may occur when the driver of the graphic card does not support properly this approach. As far as we know, there is no free driver under Linux handling this feature. In the proprietary world, the situation is as follows:

- `NVIDIA`: Nvidia provides the appropriate proprietary drivers. Scilab's graphics work without any problem with most NVIDIA drivers.
- `ATI`: From the driver version 8.8, most ATI graphics supports the pbuffer under Linux.
- `Intel`: This is the big drawback of using the pbuffer. There is currently no support of pbuffer by any official Intel drivers under Linux.

There is a workaround for Linux to tackle this issue, but a solution is to use a software accelerated driver. To do it, in `/etc/X11/xorg.conf`, look for the *Section "Device"* and change the option *Driver* to *vesa*:

```
Section "Device"
        Identifier      "Your Graphic card"
        Driver  "vesa"
[...]
EndSection
```

Unfortunately, this solution makes Scilab pretty slow.

Under Windows, video cards manufacturers update regularly and `pbuffers` are managed. Please download recent drivers at:

- For `ATI cards`: http://ati.amd.com/support/driver.html
- For `Intel cards`: http://www.intel.com/support/graphics/
- For `Matrox cards`: http://www.matrox.com/graphics/en/support/drivers/
- For `NVIDIA cards`: http://www.nvidia.com/content/drivers/drivers.asp
- For `S3 cards`: http://www.s3graphics.com/en/resources/drivers/index.jsp
- For `SiS cards`: http://www.sis.com/download/
- For `VIA cards`: http://www.viaarena.com/default.aspx?PageID=2

Some troubles can also occur when using Windows 2000 (video drivers are no more updated and no more supported).

In the cases where pBuffer create a problem, waiting for a working `pbuffer` is not a solution indeed: *The OpenGL community is moving away from pbuffers and toward the frame buffer object extension, which is a more portable and higher-performance solution for offscreen rendering than pbuffers.* [https://jogl.dev.java.net/issues/show_bug.cgi?id=163]. The JOGL team is working to fix this issue.

For more information about this problem, please refer to:

- `JoGL` bug database: Bug #366 [https://jogl.dev.java.net/issues/show_bug.cgi?id=366]
- `Scilab` bug database: Bug #3525 [http://bugzilla.scilab.org/show_bug.cgi?id=3525]
- `Debian` bug database: Bug #501799 [http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=501799]
- `Freedesktop` bug database: Bug #17603 [https://bugs.freedesktop.org/show_bug.cgi?id=17603]

# Examples

```
// Example using GLJPanel (Mixing uicontrols and graphics is available)
usecanvas(%F);
plot2d();
uicontrol("String", "Close the window", "Position", [10 10 100, 25], "Callback"
messagebox("You can see the button on the figure.", "Usecanvas example", "info"

// Example using GLCanvas (Mixing uicontrols and graphics is not available, uic
usecanvas(%T);
plot2d();
uicontrol("String", "Close the window", "Position", [10 10 100, 25], "Callback"
messagebox("You can''t see any button on the figure.", "Usecanvas example", "in
```

# Authors

Vincent COUVERT

# Name

waitbar — Draw a waitbar

```
winId=waitbar(x)
winId=waitbar(x,mes)
winId=waitbar(mes)
waitbar(x,winId)
waitbar(mes,winId)
waitbar(x,mes,winId)
```

# Parameters

x
: real, fraction to display.

mes
: string, message to display.

winId
: integer greater than 0, window identificator.

# Description

`waitbar(x)` create a new waitbar displaying a fraction x, return window identificator.

`waitbar(x,mes)` create a new waitbar displaying a fraction x and message mes, return window identificator.

`waitbar(mes)` create a new waitbar displaying a fraction 0 and message mes, return window identificator.

`waitbar(x,mes)` create a new waitbar displaying a fraction 0 and message mes, return window identificator.

`waitbar(x,winId)`, `waitbar(mes,winId)` and `waitbar(x,mes,winId)` update waitbar with window identificator winId.

# Examples

```
winId=waitbar('This is an example');
realtimeinit(0.3);
for j=0:0.1:1,
  realtime(3*j);
  waitbar(j,winId);
end
winclose(winId);
```

# Authors

Jaime Urzua

# Name

x_choices — interactive Xwindow choices through toggle buttons

```
rep=x_choices(title,items)
```

# Parameters

title
: vector of strings, title for the popup window.

items
: a list of items `items=list(item1,...,itemn)`, where each `item` is also a list of the following type : `item=list('label',default_choice,choices)`. `default_choice` is an integer which gives the default toggle on entry and `choices` is a row vector of strings which gives the possible choices.

rep
: an integer vector which gives for each item the number of the selected toggle. If user exits dialog with "cancel" button `rep` is set to `[]`.

# Description

Select items through toggle lists and return in `rep` the selected items

Type `x_choices()` to see an example.

# Examples

```
l1  = list('choice 1',1,['toggle c1','toggle c2','toggle c3']);
l2  = list('choice 2',2,['toggle d1','toggle d2','toggle d3']);
l3  = list('choice 3',3,['toggle e1','toggle e2']);
rep = x_choices('Toggle Menu',list(l1,l2,l3));
```

# Name

x_choose — interactive window choice (modal dialog)

```
[num]=x_choose(items,title [,button])
```

# Parameters

items
: column vector of string, items to choose

title
: column vector of string, comment for the dialog

button
: string, text to appear in the button. Default value is 'Cancel'

num
: integer, choosen item number or 0 if dialog resumed with "Cancel" button

# Description

Returns in `num` the number of the chosen item.

WARNING: this dialog was not modal before Scilab 5.0, please use x_choose_modeless for ascendant compatibility.

# Examples

```
n = x_choose(['item1';'item2';'item3'],['that is a comment';'for the dialog'])
n = x_choose(['item1';'item2';'item3'],['that is a comment'],'Return')
```

# See Also

x_choose_modeless , x_choices , x_mdialog , getvalue , unix_g

# Name

x_choose_modeless — interactive window choice (not modal dialog)

```
[num]=x_choose_modeless(items,title [,button])
```

# Parameters

items
: column vector of string, items to choose

title
: column vector of string, comment for the dialog

button
: string, text to appear in the button. Default value is 'Cancel'

num
: integer, choosen item number or 0 if dialog resumed with "Cancel" button

# Description

Returns in `num` the number of the chosen item.

# Examples

```
n = x_choose_modeless(['item1';'item2';'item3'],['that is a comment';'for the d
n = x_choose_modeless(['item1';'item2';'item3'],['that is a comment'],'Return')
```

# See Also

x_choose , x_choices , x_mdialog , getvalue , unix_g

# Name

x_dialog — Dialog for interactive multi-lines input.

```
result=x_dialog(labels,valueini)
```

# Parameters

labels
: column vector of strings, comment for dialog

valueini
: n column vector of strings, initial value suggested

result
: User answer: n column vector of strings if returned with "Ok" button or [] if returned with "Cancel" button

# Description

Opens a dialog for interactive multi-lines input.

# Examples

```
gain=evstr(x_dialog('value of gain ?','0.235'))
x_dialog(['Method';'enter sampling period'],'1')
m=evstr(x_dialog('enter a  3x3 matrix ',['[0 0 0';'0 0 0';'0 0 0]']))
```

# See Also

x_mdialog , x_matrix , evstr , execstr

# Name

x_matrix — Xwindow editing of matrix

```
[result]=x_matrix(label,matrix-init)
```

# Parameters

label
: character string (name of matrix)

matrix-init
: real matrix

# Description

For reading or editing a matrix .

# Examples

```
m=evstr(x_matrix('enter a  3x3 matrix ',rand(3,3)))
```

# See Also

x_mdialog , x_dialog

# Name

x_mdialog — Dialog for interactive vector/matrix input.

```
result=x_mdialog(title,labels,default_inputs_vector)
result=x_mdialog(title,labelsv,labelsh,default_input_matrix)
```

# Parameters

title
: column vector of strings, dialog general comment

labels
: n column vector of strings, `labels(i)` is the label of the ith required value

default_input_vector
: n column vector of strings, `default_input_vector(i)` is the initial value of the ith required value

labelsv
: n vector of strings, `labelsv(i)` is the label of the ith line of the required matrix

labelsh
: m vector of strings, `labelsh(j)` is the label of the jth column of the required matrix

default_input_matrix
: n x m matrix of strings, `default_input_matrix(i,j)` is the initial value of the (i,j) element of then required matrix

result
: n x m matrix of string if returned with "Ok" button or [] if returned with "Cancel" button

# Description

Opens a dialog for interactive vector/matrix input.

# Examples

```
txt = ['magnitude';'frequency';'phase    '];
sig = x_mdialog('enter sine signal',txt,['1';'10';'0'])
mag = evstr(sig(1))
frq = evstr(sig(2))
ph  = evstr(sig(3))

rep = x_mdialog(['System Simulation';'with PI regulator'],...
                 ['P gain';'I gain '],[' ';' '])
```

# See Also

editvar , x_dialog , x_choose , messagebox , getvalue , evstr , execstr , editvar

# Name

x_message_modeless — X window modeless message

```
x_message_modeless(strings)
```

# Parameters

strings
: vector of characters strings to be displayed

# Description

for displaying a message (information, user-guide ...). The function returns immediately. The message window is killed when "Ok" button is clicked.

# Examples

```
x_message_modeless(['This is a modeless message'
                    'Scilab may continue computation'
                    ' '
                    'Click on ""Ok"" to close the message'])
x_message_modeless('Now two message windows are opened')
```

# See Also

x_dialog , x_mdialog , messagebox

# Parte XVI. Link Dinâmico/incremental

# Name

call — chamada a rotinas de usuário FORTRAN ou C

```
// forma longa: 'out' está presente
[y1,...,yk]=call("ident",x1,px1,"tx1",...,xn,pxn,"txn",
"out",[ny1,my1],py1,"ty1",...,[nyl,myl],pyl,"tyl")
// forma curta : nenhum parâmetro 'out'
[y1,....,yk]=call("ident",x1,...,xn)
```

# Parâmetros

"ident"
: string.

xi
: real matrix or string

pxi, pyi
: inteiro

txi, tyi
: string `"d"`, `"r"`, `"i"` ou `"c"`.

# Descrição

Chamada interativa ao programa do usuário FORTRAN (ou C) do Scilab. A rotina deve estar previamente linkada ("ligada") ao Scilab. Este link pode ser feito:

- Com o comando "`link`" do Scilab (linkagem "suave" ("soft") incremental) durante a sessão Scilab. (ver `link`)

Há duas formas de sintaxe de chamamento, uma curta e uma longa. A curta fornece um código mais rápido e uma sintaxe de chamamento mais fácil, mas deve-se escrever uma pequena interface (C ou FORTRAN) para que esta forma seja possível. A forma longa torna possível uma chamada a uma rotina FORTRAN (ou C) sem modificação do código, mas a sintaxe é mais complexa e o código interpretado mais devagar.

O significado de cada parâmetro é descrito abaixo:

"ident"
: é o nome da subrotina chamada.

x1,...,xn
: são as variáveis de entrada (strings ou matrizes de reais) enviados à rotina

px1,...,pxn
: são as respectivas posições destas variáveis na seqüência de chamamento da rotina `"ident"` e

tx1,...,txn
: são seus tipos (`"r"`, `"i"`, `"d"` e `"c"` para ponto flutuante real , inteiro, dupla precisão e strings)

"out"
: é a palavra-chave usada para separar variáveis de entrada das variáveis de saída. Quando esta palavra-chave está presente, ela indica que a forma longa será usada e quando não está presente, indica que a forma curta será usada.

[ny1, my1]
: são os tamanhos (número de linhas e colunas. Para argumentos 'c', `m1*n1` é o número de caracteres) das variáveis de saída e

py1, ...
: são as posições das variáveis de saída (possivelmente iguais a `pxi` ) na seqüência de chamamento da rotina. Os inteiros `pyi`'s devem estar em ordem crescente.

"ty1", ...
: são os tipos FORTRAN das variáveis de saída. As `k` primeiras variáveis de saída são postas em `y1,..., yk`.

Se uma variável de saída coincide com uma variável de entrada (i.e., `pyi=pxj` ) pode-se apenas passar sua posição `pyi` . O tamanho e tipo de `yi` são então os mesmos que os de`xi`. Se uma variável de saída coincide com uma variável de entrada e são especificadas as dimensões da variável de saída, `[myl,nyl]` deve satisfazer a condição de compatibilidade `mxk*nxk >= myl*nyl`.

# Exemplos

```
//Exemplo 1 com um código C simples
f1=['#include &lt;math.h&gt;'
    'void fooc(c,a,b,m,n)'
    'double a[],*b,c[];'
    'int *m,*n;'
    '{'
    '   int i;'
    '   for ( i =0 ; i &lt; (*m)*(*n) ; i++) '
    '     c[i] = sin(a[i]) + *b; '
    '}'];

mputl(f1,'fooc.c')

//criando a biblioteca compartilhada (um gateway (ligação), um Makefile e um lo
//gerados.)

ilib_for_link('fooc','fooc.c',[],"c")

// carregando a biblioteca compartilhada

exec loader.sce

//usando a nova primitiva
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);

// Entradas:
// a está na posição 2 e double
// b                3     double
// n                4     integer
// m                5     integer
// Saídas:
// c está na posição 1 e double com tamanho [m,n]
c=call("fooc",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");

//Exemplo 2 com um código FORTRAN simples
f1=['      subroutine foof(c,a,b,n,m)'
    '      integer n,m'
    '      double precision a(*),b,c(*)'
    '      do 10 i=1,m*n '
    '        c(i) = sin(a(i))+b'
    '   10 continue'
```

```
    '      end'];
mputl(f1,'foof.f')

//criando a biblioteca compartilhada (um gateway, um Makefile e um loader são
//gerados.)

ilib_for_link('foof','foof.f',[],"f")

// carrega a biblioteca compartilhada

exec loader.sce

//usando a nova primitiva
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
c=call("foof",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");
```

# Ver Também

link, c_link, intersci, addinter

# Name

G_make — call make or nmake

```
Rfiles=G_make(files,dllname)
```

# Parameters

files
: a character string or a vector of character string.

dllname
: a character string.

Rfiles
: vector of character string. `Rfiles` can be used as a first argument when calling `addinter` function.

# Description

On Unix like systems `G_make` calls the `make` utility for building target `files` and returns the value of `files` in the variable `Rfiles`. On windows platforms, `G_make` calls the `nmake` utility for building target `dllname` and it returns the value of `dllname` in the variable `Rfiles`. Of course `G_make` will work if apropriate Makefiles are provided in the current Scilab directory.

`G_make` can be used to provide OS independant call to addinter.

# Examples

```
if MSDOS then
  txt = ['ex1c.dll:',
         '  @echo ------------------------------------------',
         '  @echo From Makefile.mak',
         '  @echo ------------------------------------------',
         ' '];
  mputl(txt,TMPDIR+'/makefile.mak')
  current_dir = pwd();
  cd TMPDIR
  files=G_make([TMPDIR+'/ex1cI.o',TMPDIR+'/ex1c.o'],'ex1c.dll');// compilation
  //
  //addinter(files,'foobar','foubare'); // link
  cd(current_dir);
end
```

# See Also

addinter

# Name

VCtoLCCLib — converts Ms VC libs to LCC-Win32 libs.

```
VCtoLCCLib()
```

# Description

converts Ms VC libs to LCC-Win32 libs.

# Examples

```
bOK=chooselcccompiler();
VCtoLCCLib()
```

# Authors

Allan CORNET

# Name

addinter — new functions interface dynamic link at run time.

```
addinter(files,spname,fcts)
```

# Parameters

files
: a character string or a vector of character string contain object files used to define the new Scilab interface routine (interface code, user routines or libraries, system libraries).

spname
: a character string. Name of interface routine entry point

fcts
: vector of character strings. The name of new Scilab function implemented in the new interface (in `fin` the order ).

# Description

`addinter` performs dynamic link of a compiled C or Fortran new Scilab interface routine and define corresponding scilab functions.

You can use the command `link('show')` to get the number of the shared libraries. And to reload a new version of an interface a call to `ulink` is necessary to get rid of the old version.

See `link` for more precision on use.

Number of 'addinter' in a scilab session can be limited by the operating system. On Windows, you cannot load more than 80 dynamic libraries at the same time.

Number of functions implemented in a interface is limited to 1000.

# Examples

```
if haveacompiler() then

chdir(TMPDIR);
mkdir('example_addinter');
chdir('example_addinter');

src = ['#include <math.h>'
'#include <stack-c.h>'
'#include <api_scilab.h>'
'#include <Scierror.h>'
'#include <localization.h>'
''
'static double fun2(double x);'
''
'void fun1(double *x,double *y) '
'{'
'  *y=fun2(*x)/(*x); '
'} '
''
'static double fun2(double x)'
'{ '
'  return( sin(x+1));'
'}'
```

```
''
'int sci_fun1(char *fname)'
'{'
'  int iType1 = 0;'
'  SciErr sciErr;'
'  int m1 = 0, n1 = 0;'
'  double *pdVarOne = NULL;'
'  int *piAddressVarOne = NULL;'
''
'  CheckRhs(1,1);'
'  CheckLhs(1,1);'
''
'  sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddressVarOne);'
'  if(sciErr.iErr)'
'  {'
' printError(&sciErr, 0);'
' return 0;'
'  }'
''
'  sciErr = getVarType(pvApiCtx, piAddressVarOne, &iType1);'
'  if(sciErr.iErr)'
'  {'
' printError(&sciErr, 0);'
' return 0;'
'  }'
''
'  if (iType1 != sci_matrix)'
'  {'
'  Scierror(999,_(""%s: Wrong type for input argument #%d: A string expected.\n
'  return 0;'
'  }'
''
'  sciErr = getMatrixOfDouble(pvApiCtx, piAddressVarOne, &m1, &n1, &pdVarOne);'
'  if(sciErr.iErr)'
'  {'
' printError(&sciErr, 0);'
' return 0;'
'  }'
''
'  fun1(pdVarOne, pdVarOne);'
'  LhsVar(1) = 1;'
'  return 0;'
'}'];


mputl(src,TMPDIR + '/example_addinter/example_addinter.c');
files=['example_addinter.c'];
ilib_build('addinter',['fun1_in_scilab','sci_fun1'],files,[]);
disp(mgetl('loader.sce'));
exec loader.sce;
fun1_in_scilab(%pi)

end // if haveacompiler()
```

# See Also

link, intersci, newfun, clearfun

# Authors

Allan CORNET

# Name

c_link — check incremental/dynamic link

```
c_link(routine-name)
[test,ilib]=c_link(routine-name)
test=c_link(routine-name,num)
```

# Parameters

routine-name
: a character string

num

test
: boolean, indicates if there is a shared library which contains `routine-name`.

ilib
: a scalar, the number of the shared library which contains `routine-name`

# Description

`c_link` is a boolean function which checks if the routine `routine-name` is currently linked. This function returns a boolean value true or false. When used with two return values, the function `c_link` returns a boolean value in `test` and the number of the shared library which contains `routine-name` in `ilib` (when `test` is true).

# See Also

link , fort

# Name

chooselcccompiler — choose LCC-Win32 as the default C Compiler.

```
bOK=chooselcccompiler()
```

# Parameters

bOK
returns %T if LCC-Win32 is the default C Compiler.

# Description

choose LCC-Win32 as the default C Compiler.

# Examples

```
bOK=chooselcccompiler()
```

# Authors

Allan CORNET

# Name

configure_lcc — set environments variables for LCC-Win32 C Compiler.

```
bOK=configure_lcc()
```

# Parameters

bOK
: returns %T if environments variables for LCC-Win32 C Compiler are OK.

# Description

set environments variables for LCC-Win32 C Compiler.

# Examples

```
bOK=configure_lcc()
```

# Authors

Allan CORNET

# Name

configure_ifort — set environments variables for Intel Fortran Compiler (Windows).

```
bOK=configure_msifort()
```

# Parameters

bOK
returns %T if environments variables for Intel fortran (9 or 10) Compiler are OK.

# Description

set environments variables for Intel fortran (9 or 10) Compiler.

# Examples

```
bOK = configure_msifort()
```

# Authors

Allan CORNET

# Name

configure_msvc — set environments variables for Microsoft C Compiler.

```
bOK=configure_msvc()
```

# Parameters

bOK
: returns %T if environments variables for Ms C Compiler are OK.

# Description

set environments variables for Microsoft C Compiler.

# Examples

```
bOK=configure_msvc()
```

# Authors

Allan CORNET

# Name

dllinfo — provides information about the format and symbols provided in executable and DLL files (Windows).

```
infolist = dllinfo(filename,option)
```

# Parameters

filename
: a string : a filename .dll or .exe file

option
: a string : 'machine' , 'exports' , 'imports'

infolist
: a list :

infolist(1) : a string : name of dll or executable.

infolist(2) : a string matrix : symbols (imported or exported) or machine type (x86 or x64).

# Description

This tool provides information about the format and symbols (imported or exported) provided in executable and DLL files.

This tool is based on dumpbin.exe. A tool provided with Visual studio SDK.

# Examples

```
if MSDOS then
  filename = SCI+'\bin\libscilab.dll';

  dllinfolist = dllinfo(filename,'machine');
  printf('Machine destination of %s: %s\n',dllinfolist(1),dllinfolist(2));

  dllinfolist = dllinfo(filename,'imports');
  printf('Dlls dependencies of %s:\n',filename);
  for i=1:size(dllinfolist)
    printf('%s\n',dllinfolist(i)(1));
  end

  dllinfolist = dllinfo(filename,'exports');
  printf('Dll exports of %s:\n',filename);
  disp(dllinfolist);
end
```

# See Also

addinter, link, ilib_compile, ilib_gen_Make, ilib_gen_gateway, ilib_gen_loader, ilib_for_link

# Authors

Allan CORNET

# Name

findlcccompiler — detects LCC-Win32 C Compiler

```
ret=findlcccompiler()
```

# Parameters

ret
: returns %T or %F

# Description

detects LCC-Win32 C Compiler.

# Examples

```
ret=findlcccompiler()
```

# Authors

Allan CORNET

# Name

findmsifortcompiler — detects Intel fortran Compiler

```
ifortv=findmsifortcompiler()
```

# Parameters

ifortv
: returns 'ifort90','ifort10','ifort11','unknown'

# Description

detects Intel fortran Compiler (Windows).

# Examples

```
ifortv = findmsifortcompiler()
```

# Authors

Allan CORNET

# Name

findmsvccompiler — detects Microsoft C Compiler

```
msvc=findmsvccompiler()
```

# Parameters

msvc
returns
'msvc70','msvc71','msvc80express','msvc80std,'msvc80pro','msvc90express','msvc90std,'msvc90pro','unknown'

# Description

detects Microsoft C Compiler.

# Examples

```
msvc=findmsvccompiler()
```

# Authors

Allan CORNET

# Name

fort — Fortran or C user routines call

```
// long form 'out' is present
[y1,...,yk]=fort("ident",x1,px1,"tx1",...,xn,pxn,"txn",
"out",[ny1,my1],py1,"ty1",...,[nyl,myl],pyl,"tyl")
// short form : no 'out' parameter
[y1,....,yk]=fort("ident",x1,...,xn)
```

# Parameters

"ident"
: string.

xi
: real matrix or string

pxi, pyi
: integers

txi, tyi
: character string `"d"`, `"r"`, `"i"` or `"c"`.

# Description

Interactive call of Fortran (or C) user program from Scilab. The routine must be previously linked with Scilab. This link may be done:

- with Scilab "`link`" command (dynamic link) during the Scilab session.(see `link`)

There are two forms of calling syntax, a short one and a long one. The short one will give faster code and an easier calling syntax but one has to write a small (C or Fortran) interface in order to make the short form possible. The long one make it possible to call a Fortran routine (or a C one) whitout modification of the code but the syntax is more complex and the interpreted code slower.

The meaning of each parameter is described now:

"ident"
: is the name of the called subroutine.

x1,...,xn
: are input variables (real matrices or strings) sent to the routine,

px1,...,pxn
: are the respective positions of these variables in the calling sequence of the routine `"ident"` and

tx1,...,txn
: are their types (`"r"`, `"i"`, `"d"` and `"c"` for real (float) , integer, double precision and strings)

"out"
: is a keyword used to separate input variables from output variables. when this key word is present it is assumes that the long form will be used and when it is not prsent, the short form is used.

[ny1, my1]
: are the size (number of rows and columns. For 'c' arguments,`m1*n1` is the number of charaters ) of output variables and

py1, ...
: are the positions of output variables (possibly equal to `pxi` ) in the calling sequence of the routine. The `pyi`'s integers must be in increasing order.

"ty1", ...
: are the Fortran types of output variables. The `k` first output variables are put in `y1, ..., yk`.

If an output variable coincides with an input variable (i.e. `pyi=pxj` ) one can pass only its position `pyi` . The size and type of `yi` are then the same as those of `xi`. If an output variable coincides with an input variable and one specify the dimensions of the output variable `[myl,nyl]` must follow the compatibility condition `mxk*nxk >= myl*nyl`.

For example the following program:

```
subroutine foof(c,a,b,n,m)
  integer n,m
  double precision a(*),b,c(*)
  do 10 i=1,m*n
    c(i) = sin(a(i))+b
  10 continue
end
```

```
link("foof"+getdynlibext(),"foof")
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
// Inputs:
// a is in position 2 and double
// b                3     double
// n                4     integer
// m                5     integer
// Outputs:
// c is in position 1 and double with size [m,n]
c=fort("foof",a,2,"d",b,3,"d",n,4,"i",m,5,"i","out",[m,n],1,"d");
```

returns the matrix `c=2*a+b`.

The same example coded in C:

```
void fooc(c,a,b,m,n)
double a[],*b,c[];
int *m,*n;
{
   double sin();
   int i;
   for ( i =0 ; i &lt; (*m)*(*n) ; i++)
       c[i] = sin(a[i]) + *b;
}
```

```
link("fooc"+getdynlibext(),"fooc","C") // note the third argument
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
c=fort("fooc",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");
```

# See Also

call, link, c_link, intersci, addinter

# Name

getdynlibext — get the extension of dynamic libraries on your operating system.

```
ret=getdynlibext()
```

# Description

get the extension of dynamic libraries on your operating system.

ret=getdynlibext() returns (.so on linux,.sl HP-UX,.dll on Windows, ...).

# Examples

```
getdynlibext()
```

# Authors

Allan CORNET

# Name

haveacompiler — detect if you have a C compiler.

```
bOK=haveacompiler()
```

# Parameters

bOK
returns %T if you have a C compiler.

# Description

detect if you have a C compiler.

# Examples

```
bOK = haveacompiler();
```

# See Also

findlcccompiler , findmsvccompiler

# Name

ilib_build — utility for shared library management

```
ilib_build(lib_name,table,files,libs [,makename,ldflags,cflags,fflags,ismex, cc
```

# Parameters

lib_name
: a character string, the generic name of the library without path and extension.

table
: 2 column string matrix giving the table of pairs 'scilab-name', 'interface name'

files
: string matrix giving source (from Scilab 5.0) or object files needed for shared library creation

libs
: string matrix giving extra libraries needed for shared library creation

makename
: character string. The path of the Makefile file without extension.

ldflags,cflags,fflags
: character strings to provide options for the loader, the C compiler and the Fortran compiler.

ismex
: Internal variable to specify if we are working with mex or not.

cc
: Provide the name of the C compiler.

# Description

This tool is used to create shared libraries and to generate a loader file which can be used to dynamically load the shared library into Scilab with `addinter`

Many examples are provided in `SCI/modules/dynamic_link/examples` directory. They are all released into the public domain.

**Note that a compiler must be available on the system to use this function.**

**Languages handle by this function are: C, C++, Fortran and Fortran 90.**

# Examples (C code)

```
//Here with give a complete example on adding new primitive to Scilab
//create the procedure files

cd TMPDIR;
mkdir('example_ilib_build_c');
cd('example_ilib_build_c');

f1=['extern double fun2();'
    'void fun1(double *x, double *y)'
    '{*y=fun2(*x)/(*x);}'];

mputl(f1,TMPDIR + '/example_ilib_build_c/fun1.c');

f2=['#include <math.h>'
    'double fun2(double x)'
```

```
    '{ return( sin(x+1.));}'];
mputl(f2,TMPDIR + '/example_ilib_build_c/fun2.c');

//creating the interface file
i=['#include <stdlib.h>'
   '#include <stack-c.h>'
   '#include <api_scilab.h>'
   '#include <Scierror.h>'
   '#include <localization.h>'
   ''
   'extern int fun1 ( double *x, double *y);'
   ''
   'int sci_fun1(char *fname)'
   '{'
   '  int iType1 = 0;'
   '  SciErr sciErr;'
   '  int m1 = 0, n1 = 0;'
   '  double *pdVarOne = NULL;'
   '  int *piAddressVarOne = NULL;'
   ''
   '  CheckRhs(1,1);'
   '  CheckLhs(1,1);'
   ''
   '  sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddressVarOne);'
   '  if(sciErr.iErr)'
   '  {'
   '    printError(&sciErr, 0);'
   '    return 0;'
   '  }'
   ''
   '  sciErr = getVarType(pvApiCtx, piAddressVarOne, &iType1);'
   '  if(sciErr.iErr)'
   '  {'
   '    printError(&sciErr, 0);'
   '    return 0;'
   '  }'
   ''
   '  if (iType1 != sci_matrix)'
   '  {'
   '     Scierror(999,_(""%s: Wrong type for input argument #%d: A string expec
   '     return 0;'
   '  }'
   ''
   '  sciErr = getMatrixOfDouble(pvApiCtx, piAddressVarOne, &m1, &n1, &pdVarOne
   '  if(sciErr.iErr)'
   '  {'
   '    printError(&sciErr, 0);'
   '    return 0;'
   '  }'
   ''
   '  fun1(pdVarOne, pdVarOne);'
   '  LhsVar(1) = 1;'
   '  return 0;'
   '}'];
mputl(i,TMPDIR + '/example_ilib_build_c/sci_fun1.c');

//creating the shared library (a gateway, a Makefile and a loader are
//generated.
```

```
files=['fun1.c','fun2.c','sci_fun1.c'];
ilib_build('build_c',['fun1','sci_fun1'],files,[]);

// load the shared library

exec loader.sce;

//using the new primitive
fun1(33)
```

## Examples (C code - previous Scilab API < 5.2)

```
cd TMPDIR;
mkdir('example_ilib_build_c_old');
cd('example_ilib_build_c_old');

//Here with give a complete example on adding new primitive to Scilab
//create the procedure files
f1=['extern double fun2();'
    'void fun1(double *x, double *y)'
    '{*y=fun2(*x)/(*x);}'];

mputl(f1,'fun1.c')

f2=['#include <math.h>'
    'double fun2(double x)'
    '{ return( sin(x+1.));}'];
mputl(f2,'fun2.c');

//creating the interface file
i=['#include ""stack-c.h""'
   '#include ""stackTypeVariable.h""'
   '#include ""version.h""'
   '#if SCI_VERSION_MAJOR <= 5'
   '#if SCI_VERSION_MINOR <  2'
   '   #error ""This example is obsolete see help ilib_buid""'
   '#endif'
   '#endif'
   ''
   'extern int fun1 ( double *x, double *y);'
   'int intfun1(char *fname)'
   '{'
   '  int m1,n1,l1;'
   '  CheckRhs(1,1);'
   '  CheckLhs(1,1);'
   '  GetRhsVar(1, MATRIX_OF_DOUBLE_DATATYPE, &m1, &n1, &l1);'
   '  fun1(stk(l1),stk(l1));'
   '  LhsVar(1) = 1;'
   '  return 0;'
   '}'];
mputl(i,'intfun1.c')

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

files=['fun1.c','fun2.c','intfun1.c'];
ilib_build('ilib_c_old',['scifun1','intfun1'],files,[]);
```

```
// load the shared library

exec loader.sce

//using the new primitive
scifun1(33)
```

# Examples (C++ code)

```
cd TMPDIR;
mkdir('example_ilib_build_cpp');
cd('example_ilib_build_cpp');

i=['#include <string>'
   'extern ""C"" {'
   '#include <stdlib.h>'
   '#include <stack-c.h>'
   '#include <api_scilab.h>'
   '#include <localization.h>'
   '#include <Scierror.h>'
   ''
   'int sci_cppfind(char *fname)'
   '{'
   ''
   ' SciErr sciErr;'
   ' int *piAddressVarOne = NULL;'
   ' char *pStVarOne = NULL;'
   ' int iType1 = 0;'
   ' int lenStVarOne = 0;'
   ' int m1 = 0, n1 = 0;'
   ''
   ' int *piAddressVarTwo = NULL;'
   ' char *pStVarTwo = NULL;'
   ' int iType2 = 0;'
   ' int lenStVarTwo = 0;'
   ' int m2 = 0, n2 = 0;'
   ''
   ' int m_out = 0;'
   ' int n_out = 0;'
   ''
   ' sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddressVarOne);'
   ' if(sciErr.iErr)'
   ' {'
   '   printError(&sciErr, 0);'
   '   return 0;'
   ' }'
   ''
   ' sciErr = getVarType(pvApiCtx, piAddressVarOne, &iType1);'
   ' if(sciErr.iErr)'
   ' {'
   '   printError(&sciErr, 0);'
   '   return 0;'
   ' }'
   ''
   ' if (iType1 != sci_strings)'
   ' {'
   '   Scierror(999,_(""%s: Wrong type for input argument #%d: A string expecte
```

```
'   return 0;'
' }'
''
' sciErr = getVarAddressFromPosition(pvApiCtx, 2, &piAddressVarTwo);'
' if(sciErr.iErr)'
' {'
'  printError(&sciErr, 0);'
'  return 0;'
' }'
''
' sciErr = getVarType(pvApiCtx, piAddressVarTwo, &iType2);'
' if(sciErr.iErr)'
' {'
'   printError(&sciErr, 0);'
'   return 0;'
' }'
''
' if (iType2 != sci_strings)'
' {'
'   Scierror(999,_(""%s: Wrong type for input argument #%d: A string expecte
'   return 0;'
' }'
''
' sciErr = getMatrixOfString(pvApiCtx, piAddressVarOne, &m1, &n1, &lenStVarO
' if(sciErr.iErr)'
' {'
'   printError(&sciErr, 0);'
'   return 0;'
' }'
''
' pStVarOne = new char[lenStVarOne + 1];'
' if (pStVarOne == NULL)'
' {'
'   Scierror(999,_(""%s : Memory allocation error.\n""),fname);'
'   return 0;'
' }'
''
' sciErr = getMatrixOfString(pvApiCtx, piAddressVarTwo, &m2, &n2, &lenStVarT
' if(sciErr.iErr)'
' {'
'   printError(&sciErr, 0);'
'   return 0;'
' }'
''
' pStVarTwo = new char[lenStVarTwo + 1];'
' if (pStVarTwo == NULL)'
' {'
'   Scierror(999,_(""%s : Memory allocation error.\n""),fname);'
'   return 0;'
' }'
''
' sciErr = getMatrixOfString(pvApiCtx, piAddressVarOne, &m1, &n1, &lenStVarO
' if(sciErr.iErr)'
' {'
'   printError(&sciErr, 0);'
'   return 0;'
' }'
''
```

```
   ' sciErr = getMatrixOfString(pvApiCtx, piAddressVarTwo, &m2, &n2, &lenStVarT
   ' if(sciErr.iErr)'
   ' {'
   '   printError(&sciErr, 0);'
   '   return 0;'
   ' }'
   ''
   ' std::string myMessage(pStVarOne);'
   ' std::string search(pStVarTwo);'

   ' delete pStVarTwo;'
   ' delete pStVarOne;'

   ' double dOut = 0.0;'
   ''
   ' if (myMessage.find(search) != std::string::npos) {'
   '     dOut = myMessage.find(search); /* The actual operation */'
   ' } else {'
   '     dOut = -1; /* Substring not found */'
   ' }'

   ' m_out = 1;'
   ' n_out = 1;'
   ' sciErr = createMatrixOfDouble(pvApiCtx, Rhs + 1, m_out, n_out, &dOut);'
   ' if(sciErr.iErr)'
   ' {'
   '   printError(&sciErr, 0);'
   '   return 0;'
   ' }'
   ''
   ' LhsVar(1) = Rhs + 1;'
   ' return 0;'
   '} /* extern ""C"" */'
   '}'];

mputl(i,TMPDIR + '/example_ilib_build_cpp/sci_cppfind.cxx');

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

files = ['sci_cppfind.cxx'];
ilib_build('ilib_build_cpp',['cppfind','sci_cppfind'],files,[]);

// load the shared library

exec loader.sce;

// Small test to see if the function is actually working.
if cppfind("my very long string","long") <> 8 pause, end
if cppfind("my very long string","very") <> 3 pause, end
if cppfind("my very long string","short") <> -1 pause, end
```

## Examples (C++ code - previous Scilab API < 5.2)

```
cd TMPDIR;
mkdir('example_ilib_build_cpp_old');
cd('example_ilib_build_cpp_old');
```

```
i=['#include <string>'
   'extern ""C"" {'
   '#include ""stack-c.h""'
   '#include ""version.h""'
   '#if SCI_VERSION_MAJOR <= 5'
   '#if SCI_VERSION_MINOR <  2'
   '   #error ""This example is obsolete see help ilib_buid""'
   '#endif'
   '#endif'
   ''
   'int sci_cppfind(char *fname) {'
   '  int m1 = 0, n1 = 0, l1;'
   '  char *inputString1, *inputString2;'
   '  int m2 = 0, n2 = 0, l2;'
   '  int m3 = 0, n3 = 0;'
   '  double *position = NULL; /* Where we will store the position */'
   '  CheckRhs(2,2); /* Check the number of input argument */'
   '  CheckLhs(1,1); /* Check the number of output argument */'
   '  GetRhsVar(1, ""c"", &m1, &n1, &l1); /* Retrieve the first input argument
   '  inputString1=cstk(l1);'
   '  GetRhsVar(2, ""c"", &m2, &n2, &l2); /* Retrieve the second input argument
   '  inputString2=cstk(l2);'
   '  std::string myMessage (inputString1);'
   '  std::string search (inputString2);'
   '  m3=1;n3=1;'
   '  position = new double[1];'
   '  if (myMessage.find(search) != std::string::npos) {'
   '    position[0] = myMessage.find(search); /* The actual operation */'
   '  } else {'
   '    position[0] = -1; /* Substring not found */'
   '  }'
   '  CreateVarFromPtr(Rhs+1,""d"",&m3,&n3,&position); /* Create the output arg
   '  LhsVar(1) = Rhs+1;'
   '  delete[] position;'
   '  return 0;'
   '}'
   '}'];

mputl(i,'sci_cppfind.cxx');

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

files=['sci_cppfind.cxx'];
ilib_build('foo_old',['cppfind','sci_cppfind'],files,[]);

// load the shared library

exec loader.sce

// Small test to see if the function is actually working.
if cppfind("my very long string","long") <> 8 pause, end
if cppfind("my very long string","very") <> 3 pause, end
if cppfind("my very long string","short") <> -1 pause, end
```

# Examples (Fortran 90 code)

```
cd TMPDIR;
mkdir('example_ilib_build_f90');
cd('example_ilib_build_f90');

sourcecode=['subroutine incrdoublef90(x,y)'
            '  implicit none'
            '  double precision, intent(in) :: x'
            '  double precision, intent(out) :: y'
            '  y=x+1'
            'end subroutine incrdoublef90'];
mputl(sourcecode,'incrdoublef90.f90');
libpath=ilib_for_link('incrdoublef90','incrdoublef90.f90',[],'f');
exec loader.sce
n=1.;
m=call("incrdoublef90",n,1,"d","out",[1,1],2,"d");
if abs(m-2.)>%eps then pause,end
n=2.;
m=call("incrdoublef90",n,1,"d","out",[1,1],2,"d");
if abs(m-3.)>%eps then pause,end
```

# See Also

addinter, link, ilib_compile, ilib_gen_Make, ilib_gen_gateway, ilib_gen_loader, ilib_for_link

# Author

Allan CORNET

# Name

ilib_compile — ilib_build utility: executes the Makefile produced by ilib_gen_Make

```
libn=ilib_compile(lib_name,makename [,files,ldflags,cflags,fflags,cc])
```

# Parameters

lib_name
: a character string, the generic name of the library without path and extension.

makename
: character string. The path of the Makefile file without extension.

files
: optionnal vector of character strings. If files is given the make is performed on each target contained in files then a whole make is performed

libn
: character string. The path of the actual generated shared library file.

ldflags,cflags,fflags,cc
: character strings to provide options/flags for the loader, the C compiler, the Fortran compiler. cc provides the name of the compiler.

# Description

Utility function used by `ilib_build`

This executes the Makefile produced by `ilib_gen_Make`, compiles the C and fortran files and generates the shared library.

Shared libraries can then be used with the `link` and `addinter` Scilab function for incremental/dynamic link.

**Note that a compiler must be available on the system to use this function.**

# See Also

addinter , link , ilib_build , ilib_gen_Make , ilib_gen_gateway , ilib_gen_loader , ilib_for_link

# Name

ilib_for_link — utility for shared library management with link

```
libn=ilib_for_link(names,files,libs,flag [,makename [,loadername [,libname [,ld
```

# Parameters

names
: a string matrix giving the entry names which are to be linked.

files
: string matrix giving source (from Scilab 5.0) or object files needed for shared library creation

libs
: string matrix giving extra libraries needed for shared library creation

flag
: a string flag ("c" or "f") for C or Fortran entry points.

makename
: character string. The pathname of the Makefile file without extension (default value `Makelib`).

loadername
: character string. The pathname of the loader file (default value is `loader.sce`).

libname
: optional character string. The name of the generated shared library (default value is '', and in this case the name is derived from `names(1)`).

ldflags
: optional character string. It can be used to add specific linker options in the generated Makefile. Default value is ''

cflags
: optional character string. It can be used to add specific C compiler options in the generated Makefile. Default value is ''

fflags
: optional character string. It can be used to add specific Fortran compiler options in the generated Makefile. Default value is ''

cc
: optional character string. It can be used to specify a C compiler. Default value is ''

libn
: character string. The path of the really generated shared library file.

# Description

This tool is used to create shared libraries and to generate a loader file which can be used to dynamically load the shared library into Scilab with the `link` function. New entry points given by `names` are then accessible through the `call` function or with non linear tools `ode`, `optim`,...

The file to compile are supposed to be located given by `makename`. If `makename` sets a path different to the current directory, `loader` script must be located in the same directory using the `loadername` variable.

Many examples are provided in `SCI/modules/dynamic_link/examples` directory. They are all released into the public domain.

**Note that a compiler must be available on the system to use this function.**

**Languages handle by this function are: C, C++, Fortran and Fortran 90.**

# Examples (C code)

```
if haveacompiler() then

chdir(TMPDIR)
f1=['int ext1c(int *n, double *a, double *b, double *c)'
    '{int k;'
    '  for (k = 0; k < *n; ++k) '
    '      c[k] = a[k] + b[k];'
    '  return(0);}'];

mputl(f1,'fun1.c')

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

ilib_for_link('ext1c','fun1.c',[],"c")

// load the shared library

exec loader.sce

//using the new primitive
a=[1,2,3];b=[4,5,6];n=3;
c=call('ext1c',n,1,'i',a,2,'d',b,3,'d','out',[1,3],4,'d');
if norm(c-(a+b)) > %eps then pause,end

end
```

# See Also

addinter , link , ilib_compile , ilib_gen_Make , ilib_gen_gateway , ilib_gen_loader , ilib_for_link

# Name

ilib_gen_Make — utility for ilib_build: produces a Makefile for building shared libraries

```
Makename=ilib_gen_Make(name,files,libs,makename [,with_gateway,ldflags,cflags,f
```

# Parameters

lib_name
: a character string, the generic name of the library without path and extension.

files
: a vector of character string. The names of the C or Fortran files without the extension and the path part.

libs
: a vector of character string. additionnal libraries paths or [].

makename
: character string. The path of the Makefile file.

with_gateway
: a boolean. If true a file with name <lib_name>_gateway is added. Default value is %t

ldflags
: a string. It can be used to add specific linker options in the generated Makefile. Default value is "

cflags
: a string. It can be used to add specific C compiler options in the generated Makefile. Default value is "

fflags
: a string. It can be used to add specific Fortran compiler options in the generated Makefile. Default value is "

cc
: a string. The name of the C compiler. Default value is the C compiler detected on the host.

Makename
: character string. The path of the really generated Makefile file.

# Description

Utility function used by `ilib_build`

This function generates a Makefile adapted to the Operating System for building shared libraries to be loaded in Scilab. Proper options and paths are set.

Shared libraries can then be used with the `link` and `addinter` scilab function for incremental/dynamic linking.

The shared library is build from a set of C or Fortran routines stored in a directory and if required from a set of external libraries.

Files are not required to exist, when Makefile is generated, but of course are required for executing the Makefile.

**Only use this function is you know what you are doing (it is a semi-private function).**

# See Also

addinter , link , ilib_build , ilib_compile , ilib_gen_gateway , ilib_gen_loader , ilib_for_link

# Name

ilib_gen_cleaner — utility for ilib_build: generates a cleaner file

```
ilib_gen_cleaner(makename,[loadername],[files])
```

# Parameters

makename
: character string. The pathname of the Makefile file without extension (default value `Makelib`).

loadername
: character string. The pathname of the loader file (default value is `loader.sce`).

files
: matrix of character string. files to delete.

# Description

Utility function used by `ilib_build and ilib_for_link` This function generates a cleaner file.

# See Also

ilib_gen_loader , ilib_build , ilib_for_link

# Name

ilib_gen_gateway — utility for ilib_build, generates a gateway file.

```
ilib_gen_gateway(name,table)
```

# Parameters

name
: a character string, the generic name of the library without path and extension.

table
: 2 column string matrix giving the table of pairs 'scilab-name' 'interface name'

# Description

Utility function used by `ilib_build` This function generates a gateway file used by `addinter`.

if WITHOUT_AUTO_PUTLHSVAR variable is defined and equals to %t, PutLhsVar(); will need to manage PutLhsVar in each interface as internal scilab functions.

In another case (default, for compatibility with previous version) , PutLhsVar(); is added in each interface.

You can also see SCI/contrib/toolbox_skeleton/sci_gateway/c/builder_gateway_c.sce (as example about WITHOUT_AUTO_PUTLHSVAR)

# Example about WITHOUT_AUTO_PUTLHSVAR = %t

```
cd TMPDIR
WITHOUT_AUTO_PUTLHSVAR = %t;
name = 'gw_example1';
table = ['sci_func1', 'func1']
ilib_gen_gateway(name,table)

// generated gateway
mgetl(TMPDIR+'/gw_example1.c')
```

```
int sci_func1(char *fname)
{
  // ... your C code
  // you need to add a explicit PutLhsVar();
  // as internal all gateways of scilab

  PutLhsVar();
  return 0;
}
```

# Example about WITHOUT_AUTO_PUTLHSVAR = %f (default)

```
cd TMPDIR
WITHOUT_AUTO_PUTLHSVAR = %f;
```

```
name = 'gw_example2';
table = ['sci_func2', 'func2']
ilib_gen_gateway(name,table)

// generated gateway
mgetl(TMPDIR+'/gw_example2.c')
```

```
int sci_func2(char *fname)
{
  // ... your code
  // you do NOT need to add a explicit PutLhsVar();
  // added by scilab after call to sci_func2
  // default mode in scilab 4

  return 0;
}
```

# See Also

addinter, link, ilib_build, ilib_compile, ilib_gen_Make, ilib_gen_loader, ilib_for_link

# Name

ilib_gen_loader — utility for ilib_build: generates a loader file

```
ilib_gen_loader(name,table,[libs])
```

# Parameters

name
: a character string, the generic name of the library without path and extension.

table
: 2 column string matrix giving the table of pairs 'scilab-name' 'interface name'

libs
: a string matrix, externals dynamic libraries filenames to load by loader file (optional).

# Description

Utility function used by `ilib_build` This function generates a loader file.

# See Also

addinter , link , ilib_build , ilib_compile , ilib_gen_Make , ilib_gen_loader , ilib_for_link

# Name

ilib_mex_build — utility for mex library management

```
ilib_mex_build(lib_name,table,files,libs [,makename,ldflags,cflags,fflags,cc])
```

# Parameters

lib_name
: a character string, the generic name of the library without path and extension.

table
: 3 column string matrix giving the table of 'scilab-name', 'interface name', 'cmex' or 'fmex'

files
: string matrix giving objects files needed for shared library creation

libs
: string matrix giving extra libraries needed for shared library creation

makename
: character string. The path of the Makefile file without extension.

ldflags,cflags,fflags,cc
: character strings to provide options/flags for the loader, the C compiler, the Fortran compiler. cc provides the name of the compiler.

# Description

This function is used to create mex libraries and to generate a loader file which can be used to dynamically load the mex shared library.

Note that the file name containing the mex code can be set in the third input argument (`files`) or the second value of the `table` input argument.

**Note that a compiler must be available on the system to use this function.**

# Examples

```
cd(TMPDIR);

mputl(['#include ""mex.h""'
       'void mexFunction(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[])'
       '{'
       '  int *dims = mxGetDimensions(prhs[0]);'
       '  sciprint(""%d %d %d\n"",dims[0],dims[1],dims[2]);'
       '}'],'mexfunction16.c');
ilib_mex_build('libmex',['mexf16','mexfunction16','cmex'],[],[],'Makelib','',''

exec(TMPDIR+'/loader.sce');
mexf16(rand(2,3,2));
```

# See Also

addinter , link , ilib_compile , ilib_gen_Make , ilib_gen_gateway , ilib_gen_loader , ilib_for_link

# Name

ilib_verbose — set level of display used by dynamic link functions.

```
level = ilib_verbose()
ilib_verbose(level)
```

# Parameters

level :

level of verbose for dynamic link functions.

0 : no message

1 : default level (as previous version of scilab)

2 : maximum verbose level (configure , makefile, debug information, ...)

# Description

"ilib_verbose" set level of display used by dynamic link functions.

All dynamic functions in dynamic link module check this value and display or not some informations.

# Examples

```
if haveacompiler() then
  cur_verbose = ilib_verbose();
  ilib_verbose(0);

  chdir(TMPDIR);
  f1=['int ext1c(int *n, double *a, double *b, double *c)'
      '{int k;'
      '  for (k = 0; k &lt; *n; ++k) '
      '      c[k] = a[k] + b[k];'
      '  return(0);}'];

  mputl(f1,'fun1.c');

  ilib_for_link('ext1c','fun1.c',[],"c");
  exec loader.sce;

  //using the new primitive
  a=[1,2,3];b=[4,5,6];n=3;
  c = call('ext1c',n,1,'i',a,2,'d',b,3,'d','out',[1,3],4,'d');
  if norm(c-(a+b)) &gt; %eps then pause,end

  ilib_verbose(1);

  f2=['int ext2c(int *n, double *a, double *b, double *c)'
      '{int k;'
      '  for (k = 0; k &lt; *n; ++k) '
      '      c[k] = a[k] + b[k];'
      '  return(0);}'];
```

```
  mputl(f2,'fun2.c');

  ilib_for_link('ext2c','fun2.c',[],"c")
  exec loader.sce;

  //using the new primitive
  a = [1,2,3]; b = [4,5,6];n = 3;
  c = call('ext2c',n,1,'i',a,2,'d',b,3,'d','out',[1,3],4,'d');
  if norm(c-(a+b)) &gt; %eps then pause,end

  ilib_verbose(cur_verbose);

end
```

# See Also

mode, link, ilib_compile, ilib_build, ilib_for_link

# Authors

Allan CORNET

# Name

link — dynamic linker

```
x=link(files [, sub-names,flag]);
link(x , sub-names [, flag]);
ulink(x)
lst=link('show')
lst=link()
```

# Parameters

files
: a character string or a vector of character strings, the files names used to define the new entry point (compiled routines, user libraries, system libraries,..)

sub-names
: a character string or a vector of character strings . Name of the entry points in `files` to be linked.

x
: an integer which gives the id of a shared library linked into Scilab with a previous call to `link`.

flag
: character string `'f'` or `'c'` for Fortran (default) or C code.

# Description

`link` is a incremental/dynamic link facility: this command allows to add new compiled Fortran or C routines to Scilab executable code. Linked routines can be called interactively by the function `call`. Linked routines can also be used as "external" for e.g. non linear problem solvers (`ode`, `optim`, `intg`, `dassl`...).

`link()` returns a string matrix with linked functions.

a call to `link` returns an integer which gives the id of the shared library which is loaded into Scilab. This number can then be used as the first argument of the link function in order to link additional function from the linked shared library. The shared library is removed with the `ulink` command.

A routine can be unlinked with `ulink`. If the linked function has been modified between two links, it is required to `ulink` the previous instance before the new link.

`link('show')` returns the current linked routines.

To be able to link routines in a system independent way, it is convenient to use the `ilib_for_link` utility function instead of `link`.

(Experienced) users may also `link` a new Scilab interface routine to add a set of new functions. See ilib_build and addinter functions.

Number of 'link' in a scilab session can be limited by the operating system. On Windows, you cannot load more than 80 dynamic libraries at the same time.

# Examples

```
//Example of the use of ilib_for_link with  a simple C code
cd TMPDIR
f1=['#include &lt;math.h&gt;'
```

```
    'void fooc(double c[],double a[],double *b,int *m,int *n)'
    '{'
    '   int i;'
    '   for ( i =0 ; i &lt; (*m)*(*n) ; i++) '
    '     c[i] = sin(a[i]) + *b; '
    '}'];

mputl(f1,'fooc.c');

//creating the shared library: a Makefile and a loader are
//generated, the code is compiled and a shared library built.
ilib_for_link('fooc','fooc.c',[],"c")

// display the loader.sce file which calls link
mprintf('%s\n',mgetl('loader.sce'))
// load the shared library
exec loader.sce;

link('show')
// call the new linked entry point
a=linspace(0,%pi,10);b=5;
y1=call('fooc',a,2,'d',b,3,'d',size(a,1),4,'i',size(a,2),5,'i','out',size(a),1,
// check
y1-(sin(a)+b)
exec cleaner.sce;
```

# See Also

call, external, c_link, addinter, ilib_for_link, ilib_build

# Name

ulink — unlink a dynamically linked shared object

```
ulink(x)
ulink()
```

# Description

see `link`

If you plan to use `valgrind` to profile your toolbox, you must first set the environment variable `PROFILE_SCILAB_DYNAMIC_LINK` before starting scilab:

```
# under bash shell:
export PROFILE_SCILAB_DYNAMIC_LINK=1
export SCILAB_VALGRIND_OPT="--db-attach=no --show-below-main=yes  --log-fd=2 --
scilab -nwni -profile
```

This environment variable force Scilab not to release the loaded dynamic libraries. This will allow `valgrind` to perform its sum-up analysis.

# See Also

link

# Name

with_lcc — returns if LCC-Win32 is the default C Compiler.

```
bOK=with_lcc()
```

# Parameters

bOK
returns %T if LCC-Win32 is the default C Compiler.

# Description

checks if LCC-Win32 is the default C Compiler.

# Examples

```
bOK=with_lcc()
```

# Authors

Allan CORNET

# Parte XVII. Inteiros

# Name

iconvert — conversão para representação inteira de 1 a 4 bytes

```
y=iconvert(X,itype)
```

# Parâmetros

X
: matriz de floats ou inteiros

y
: matriz de inteiros codificados em 1, 2 ou 4 bytes

# Descrição

Converte e armazena dados para inteiros de 1, 2 ou 4 bytes.

itype=0
: retorna números em ponto flutuante

itype=1
: retorna números int8 no intervalo [-128,127]

itype=11
: retorna números uint8 no intervalo [0,255]

itype=2
: retorna números int16 no intervalo [-32768,32767]

itype=12
: retorna números uint16 no intervalo [0, 65535]

itype=4
: retorna números int32 no intervalo [-2147483648,2147483647]

itype=14
: retorna números uint32 no intervalo [0, 4294967295]

# Exemplos

```
b=int32([1 -120 127 312])
y=iconvert(b,1)
```

# Ver Também

double, inttype

# Name

int8 — conversão para representação de inteiro de 1 byte
int16 — conversão para representação de inteiro de 2 bytes
int32 — conversão para representação de inteiro de 1 byte sem sinal
uint8 — conversão para representação de inteiro de 2 byte sem sinal
uint16 — conversão para representação de inteiro de 4 byte sem sinal
uint32 — conversão para representação de inteiro de 4 byte sem sinal

```
y=int8(X)
y=int16(X)
y=int32(X)
y=uint8(X)
y=uint16(X)
y=uint32(X)
```

# Parâmetros

X
: matriz de números em ponto flutuante ou inteiros

y
: matriz de inteiros codificados em 1, 2 ou 4 bytes.

# Descrição

Converte e armazena dados em interos de 1, 2 ou 4 bytes. Estes tipos de dados são especialmente úteis para armazenar objetos grandes como imagens, sinais longos,...

y=int8(X)
: retorna números no intervalo [-128,127]

y=uint8(X)
: retorna números no intervalo [0,255]

y=int16(X)
: retorna números no intervalo [-32768,32767]

y=uint16(X)
: retorna números no intervalo [0, 65535]

y=int32(X)
: retorna números no intervalo [-2147483648,2147483647]

y=uint32(X)
: retorna números no intervalo [0, 4294967295]

# Exemplos

```
int8([1 -120 127 312])
uint8([1 -120 127 312])

x=int32(-200:100:400)
int8(x)
```

# Ver Também

double, inttype, iconvert

# Name

inttype — tipos de dados inteiros

```
[i]=inttype(x)
```

# Parâmetros

x
: uma matriz de inteiros (ver int8,..)

i
: inteiro

# Descrição

`inttype(x)` retorna um inteiro que indica o tipo das entradas de `x` como segue :

1 : representação de inteiros de um byte

2 : representação de inteiros de dois bytes

4 : representação de inteiros de quatro bytes

11 : representação de inteiros de um byte sem sinal

12 : representação de inteiros de dois bytes sem sinal

14 : representação de inteiros de quatro bytes sem sinal

# Exemplos

```
x=uint16(1:10);
inttype(x)
```

# Ver Também

int8

# Parte XVIII. Interpolação

# Name

bsplin3val — função de avaliação de derivada arbitrária de spline 3d

```
[dfp]=bsplin3val(xp,yp,zp,tl,der)
```

# Parâmetros

xp, yp, zp
: vetores ou matrizes de reais de tamanhos iguais

tl
: tlist de tipo "splin3d", definindo um tensor spline 3d (chamado de `s` a seguir)

der
: vetor com três componentes `[ox,oy,oz]` definindo qual derivada de `s` computar.

dfp
: vetor ou matriz de mesmo formato que `xp`, `yp` e `zp`, avaliação elemento a elemento da derivada especificada de `s` nesses pontos.

# Descrição

Enquanto a função interp3d pode computar apenas o spline `s` e suas primeiras derivadas, `bsplin3val` pode computar qualquer derivada de `s`. A derivada a ser computada é especificada pelo argumento `der=[ox,oy,oz]` :

```
            ox   oy   oz
           d    d    d
dfp(i) =  ---  ---  ---   s (xp(i),yp(i),zp(i))
            ox   oy   oz
          dx   dy   dz
```

Então, `der=[0 0 0]` corresponde a *s*, `der=[1 0 0]` to *ds/dx*, `der=[0 1 0]` to *ds/dy*, `der=[1 1 0]` to *d2s/dxdy*, etc...

Para um ponto com coordenadas *(xp(i),yp(i),zp(i))* fora do grid, a função retorna 0.

# Exemplos

```
deff("v=f(x,y,z)","v=cos(x).*sin(y).*cos(z)");
deff("v=fx(x,y,z)","v=-sin(x).*sin(y).*cos(z)");
deff("v=fxy(x,y,z)","v=-sin(x).*cos(y).*cos(z)");
deff("v=fxyz(x,y,z)","v=sin(x).*cos(y).*sin(z)");
deff("v=fxxyz(x,y,z)","v=cos(x).*cos(y).*sin(z)");
n = 20;  // n x n x n  pontos de interpolação
x = linspace(0,2*%pi,n); y=x; z=x; // grid de interpolação
[X,Y,Z] = ndgrid(x,y,z); V = f(X,Y,Z);
tl = splin3d(x,y,z,V,[5 5 5]);

// computando f e algumas derivadas em um ponto
// e comparando com o spline interpolante
xp = grand(1,1,"unf",0,2*%pi);
yp = grand(1,1,"unf",0,2*%pi);
zp = grand(1,1,"unf",0,2*%pi);
```

```
f_e = f(xp,yp,zp)
f_i = bsplin3val(xp,yp,zp,tl,[0 0 0])

fx_e = fx(xp,yp,zp)
fx_i = bsplin3val(xp,yp,zp,tl,[1 0 0])

fxy_e = fxy(xp,yp,zp)
fxy_i = bsplin3val(xp,yp,zp,tl,[1 1 0])

fxyz_e = fxyz(xp,yp,zp)
fxyz_i = bsplin3val(xp,yp,zp,tl,[1 1 1])

fxxyz_e = fxxyz(xp,yp,zp)
fxxyz_i = bsplin3val(xp,yp,zp,tl,[2 1 1])
```

# Ver Também

splin3d, interp3d

# Autores

R.F. Boisvert, C. De Boor (código da biblioteca FORTRAN CMLIB)
B. Pincon (interface Scilab)

# Name

cshep2d — bidimensional cubic shepard (scattered) interpolation

```
tl_coef = cshep2d(xyz)
```

# Parâmetros

xyz
: uma matriz n x 3 dos pontos de interpolação interpolação (sem grid), a i-ésima linha dadas as coordenadas (x,y) e então a altitude z do i-ésimo ponto de interpolação

tl_coef
: uma estrutura tlist do Scilab (do tipo cshep2d)

# Descrição

Esta função é útil para definir uma função de interpolação 2d quando os pontos não estão em um grid (você pode usá-la para este caso, mas splin2d é melhor para este propósito). O interpolante é do tipo cúbico de Shepard e é uma função de duas variáveis de classe C2 (duas vezes continuamente diferenciável) *s(x,y)* tal que : *s(xi,yi)=zi* para todo *i=1,..,n* ( *(xi,yi,zi)* sendo a i-ésima linha de `xyz`).

A avaliação de s em alguns pontos deve ser feita pela função eval_cshep2d.

# Observação

A função operea se **n >= 10**, se os nós não são todos colineares (i.e. as coordenadas *(x,y)* dos pontos de interpolação não estão na mesma reta) e se não há nós duplicados (i.e. dois ou mais pontos de interpolação com as mesmas coordenadas *(x,y)* ). Há erro se essas condições não são respeitadas.

# Exemplos

```
// interpolação de cos(x)cos(y) com pontos de interpolação escolhidos aleatoriam
n = 150; // número de pontos de interpolação
xy = grand(n,2,"unf",0,2*%pi);
z = cos(xy(:,1)).*cos(xy(:,2));
xyz = [xy z];
tl_coef = cshep2d(xyz);

// avaliação em um grid
m = 30;
xx = linspace(0,2*%pi,m);
[X,Y] = ndgrid(xx,xx);
Z = eval_cshep2d(X,Y, tl_coef);
xbasc()
plot3d(xx,xx,Z,flag=[2 6 4])
param3d1(xy(:,1),xy(:,2),list(z,-9), flag=[0 0])
xtitle("Interpolação cúbica de Shepard de cos(x)cos(y) com pontos de interpolaç
legends("pontos de interpolação",-9,1)
xselect()
```

# Ver Também

splin2d, eval_cshep2d

# Autores

Robert J. Renka
B. Pincon (interface do Scilab)

# Name

eval_cshep2d — avaliação por interpolação cúbica bidimensional de Shepard

```
[zp [,dzpdx, dzpdy [,d2zpdxx,d2zpdxy,d2zpdyy]]] = eval_cshep2d(xp, yp, tl_coef)
```

# Parâmetros

xp, yp
: dois vetores (ou matrizes) de mesmo tamanho

tl_coef
: uma estrutura Scilab tlist (de tipo cshep2d) definindo uma função de interpolação cúbica de Shepard (chamada `S` a partir daqui)

zp
: vetor (ou matrizes) de mesmo tamanho que `xp` e `yp`, avaliação do interpolante `S` nesses pontos

dzpdx,dzpdy
: vetores (ou matrizes) de mesmo tamanho que `xp` e `yp`, avaliação das derivadas primeiras de `S` nestes pontos

d2zpdxx,d2zpdxy,d2zpdyy
: vetores (ou matrizes) de mesmo tamanho que `xp` e `yp`, avaliação das derivadas segundas de `S` nestes pontos

# Descrição

Esta é a rotina de avaliação para a função de interpolação cúbica de Shepard obtida com cshep2d, isto é :

```
zp(i) = S(xp(i),yp(i))
dzpdx(i) = dS/dx(xp(i),yp(i))
dzpdy(i) = dS/dy(xp(i),yp(i))
d2zpdxx(i) = d2S/dx2(xp(i),yp(i))
d2zpdxy(i) = d2S/dxdy(xp(i),yp(i))
d2zpdyy(i) = d2S/dy2(xp(i),yp(i))
```

# Observação

O interpolante S é C2 (duas vezes continuamente diferenciável) mas também é estendido por zero para *(x,y)* suficientemente longe dos pontos de interpolação. Isto leva a uma descontinuidade em regiões exteriores muito longe dos pontos de interpolação e, assim, a avaliação não se torna incoveniente na prática (de um modo geral, avaliação fora dos pontos de interpolação (i.e. extrapolação) leva a resultados muito inacurados).

# Exemplos

```
// ver seção de exemplos em cshep2d

// este exemplo mostra o comportamento longe dos pontos de interpolação...
deff("z=f(x,y)","z = 1+ 50*(x.*(1-x).*y.*(1-y)).^2")
x = linspace(0,1,10);
[X,Y] = ndgrid(x,x);
```

```
X = X(:); Y = Y(:); Z = f(X,Y);
S = cshep2d([X Y Z]);
// avaliação dentro e fora do quadrado [0,1]x[0,1]
m = 40;
xx = linspace(-1.5,0.5,m);
[xp,yp] = ndgrid(xx,xx);
zp = eval_cshep2d(xp,yp,S);
// computando faceta (para definir uma cor para a região de extrapolação
// e outra para a região de interpolação)
[xf,yf,zf] = genfac3d(xx,xx,zp);
color = 2*ones(1,size(zf,2));
// índices correspondentes à faceta na região de interpolação
ind=find( mean(xf,"r")>0 & mean(xf,"r")<1 & mean(yf,"r")>0 & mean(yf,"r")<1 );
color(ind)=3;
xbasc();
plot3d(xf,yf,list(zf,color), flag=[2 6 4])
legends(["região de extrapolação","região de interpolação"],[2 3],1)
xselect()
```

# Ver Também

cshep2d

# Autores

Robert J. Renka
B. Pincon (interface Scilab)

# Name

interp — função de avaliação de spline cúbico

```
[yp [,yp1 [,yp2 [,yp3]]]]=interp(xp, x, y, d [, out_mode])
```

# Parâmetros

xp
: vetor ou matriz de reais

x,y,d
: vetores de reais de mesmo tamanho definindo uma função de spline cúbico ou sub-spline (chamado `s` a partir daqui)

out_mode
: (opcional) string definido a avaliação de `s` fora do intervalo [x1,xn]

yp
: vetor ou matriz de mesmo tamanho que `xp`, avaliação elemento a elemento de `s` em `xp` (yp(i)=s(xp(i) ou yp(i,j)=s(xp(i,j))

yp1, yp2, yp3
: vetores (ou matrizes) de mesmo tamanho que `xp`, avaliação elemento a elemento das derivadas sucessivas de `s` em `xp`

# Descrição

Dados três vetores `(x,y,d)` ddefinindo uma função de spline cúbico ou sup-spline (ver splin) com `yi=s(xi), di = s'(xi)` esta função avalia `s` (e `s', s'', s'''` se necessário) em `xp(i)` :

```
yp(i) = s(xp(i))    ou  yp(i,j) = s(xp(i,j))
yp1(i) = s'(xp(i))   ou  yp1(i,j) = s'(xp(i,j))
yp2(i) = s''(xp(i))   ou  yp2(i,j) = s''(xp(i,j))
yp3(i) = s'''(xp(i))   ou  yp3(i,j) = s'''(xp(i,j))
```

O parâmetro `out_mode` ajusta a regra de avaliação para extrapolação, i.e., para `xp(i)` fora de [x1,xn] :

"by_zero"
: uma extrapolação por zero é feita

"by_nan"
: extrapolação por NaN

"C0"
: a extrapolação é definida como segue :

```
s(x) = y1  para  x < x1
s(x) = yn  para  x > xn
```

"natural"
: a extrapolação é definida como segue (p_i sendo o polinômio que define `s` em [x_i,x_{i+1}]) :

```
s(x) = p_1(x)      para  x < x1
s(x) = p_{n-1}(x)  para  x > xn
```

"linear"
a extrapolação é definida como segue :

```
s(x) = y1 + s'(x1)(x-x1)  para  x < x1
s(x) = yn + s'(xn)(x-xn)  para  x > xn
```

"periodic"
s é estendido por periodicidade.

# Exemplos

```
// veja os exemplos de splin e lsq_splin

// um exemplo exibindo as continuidades C2 e C1 de um spline e um sub-spline
a = -8; b = 8;
x = linspace(a,b,20)';
y = sinc(x);
dk = splin(x,y);  // not_a_knot
df = splin(x,y, "fast");
xx = linspace(a,b,800)';
[yyk, yy1k, yy2k] = interp(xx, x, y, dk);
[yyf, yy1f, yy2f] = interp(xx, x, y, df);
xbasc()
subplot(3,1,1)
plot2d(xx, [yyk yyf])
plot2d(x, y, style=-9)
legends(["spline não é um nó","sub-spline rápido","pontos de interpolação"],...
        [1 2 -9], "ur",%f)
xtitle("interpolação por spline")
subplot(3,1,2)
plot2d(xx, [yy1k yy1f])
legends(["spline não é um nó","sub-spline rápido"], [1 2], "ur",%f)
xtitle("interpolação por spline (derivadas)")
subplot(3,1,3)
plot2d(xx, [yy2k yy2f])
legends(["spline não é um nó","sub-spline rápido"], [1 2], "lr",%f)
xtitle("interpolação por splines (segundas derivadas)")

// aqui está um exemplo mostrando as diferentes possibilidades de extrapolação
x = linspace(0,1,11)';
y = cosh(x-0.5);
d = splin(x,y);
xx = linspace(-0.5,1.5,401)';
yy0 = interp(xx,x,y,d,"C0");
yy1 = interp(xx,x,y,d,"linear");
yy2 = interp(xx,x,y,d,"natural");
yy3 = interp(xx,x,y,d,"periodic");
xbasc()
plot2d(xx,[yy0 yy1 yy2 yy3],style=2:5,frameflag=2,leg="C0@linear@natural@period
xtitle(" Modos diferentes de avaliar um spline fora de seu domínio")
```

# Ver Também

splin, lsq_splin

# Autor

B. Pincon

# Name

interp1 — função de interpolação 1d

```
[yp]=interp1(x, y,xp [, method,[extrapolation]])
```

# Parâmetros

xp
: escalar real, vetor ou matriz (ou hipermatriz) de reais

x
: vetor de reais

method
: (opcional) string definindo o método de interpolação

extrapolation
: (opcional) string ou valor real defindo os componentes yp(j) para os valores xp(j) fora do intervalo [x1,xn].

yp
: vetor ou matriz (ou hipermatriz)

# Descrição

Dados `(x,y,xp)`, esta função faz corresponder os yp componentes aos xp por interpolação (linear por padrão) definida por x e y.

Se `yp` é um vetor, então o comprimento de xp é igual ao comprimento de `yp`. Se `yp` é uma matriz, então `xp` tem o mesmo comprimento que cada uma das colunas de yp. Se `yp` é uma hipermatriz, então o comprimento de `xp` é o mesmo da primeira dimensão de `yp.`

Se size(y)=[C,M1,M2,M3,...,Mj] e size(xp)= [N1,N2,N3,...,Nk] então size(yp)= [N1,N2,..,Nk,M1,M2,...Mj] e o comprimento de x deve ser igual a size(y,1)

O parâmetro `method` ajusta a regra avaliação para interpolação.

"linear"
: a interpolação é definida pelo método linear (ver interpln)

"spline"
: definição de interpolação por spline cúbico (ver splin , interp)

"nearest"
: para cada valor xp(j), yp(j) toma o valor ou y(i) correspondente ao x(i), o vízinho mais próximo de xp(j)

O parâmetro `extrapolation` ajusta a regra de avaliação para extrapolação, i.e para `xp(i)` fora do intervalo [x1,xn]

"extrap"
: a extrapolação é realizada pelo método definido. yp=interp1(x,y,xp,method,"extrap")

valor real
: você pode escolher um valor real para extrapolação. Deste modo, yp(i) toma este valor para xp(i) fora do intervalo [x1,xn], por exemplo 0 (mas também nan ou inf). yi=interp1(x,y,xp,method, 0)

por padrão
a extrapolação é realizada pelo método definido (para método spline), e por nan para os métodos "linear" e "nearest". yp=interp1(x,y,xp,method)

# Exemplos

```
x=linspace(0,3,20);
y=x^2;
xx=linspace(0,3,100);
yy1=interp1(x,y,xx,'linear');
yy2=interp1(x,y,xx,'spline');
yy3=interp1(x,y,xx,'nearest');
plot(xx,[yy1;yy2;yy3],x,y,'*')
xtitle('interpolation of square function')
legend(['linear','spline','nearest'],a=2)
```

# Ver Também

interp, interpln, splin

# Autor

F.B

# Name

interp2d — função de avaliação spline bicúbica (2d)

`[zp[,dzpdx,dzpdy[,d2zpdxx,d2zpdxy,d2zpdyy]]]=interp2d(xp,yp,x,y,C [,out_mode])`

# Parâmetros

xp, yp
: vetores ou matrizes de reais de mesmo tamanho

x,y,C
: vetores de reais definindo uma função de spline bicúbico ou sub-spline (chamada `s` daqui em diante)

out_mode
: (opcional) string definindo a avaliação de `s` fora de [x(1),x(nx)]x[y(1),y(ny)]

zp
: vetor ou matriz com o mesmo formato que `xp` e `yp`, avaliação elemento a elemento de `s` nestes pontos.

dzpdx, dzpdy
: vetores (ou matrizes) de mesmo formato que `xp` e `yp`, avaliação elemento a elemento das derivadas primeiras de `s` nesses pontos.

d2zpdxx, d2zpdxy, d2zpdyy
: vetores (ou matrizes) de mesmo formato que `xp` e `yp`, eavaliação elemento a elemento das derivadas segundas de `s` nesses pontos.

# Descrição

Dados três vetores `(x,y,C)` definindo uma função de sub-spline ou spline bicúbico (ver splin2d) esta função avalia s (e *ds/dx*, *ds/dy*, *d2s/dxx*, *d2s/dxy*, *d2s/dyy* se necessário) em *(xp(i),yp(i))* :

```
zp(i) = s(xp(i),yp(i))
dzpdx(i) = ds/dx(xp(i),yp(i))
dzpdy(i) = ds/dy(xp(i),yp(i))
d2zpdxx(i) = d2s/dx2(xp(i),yp(i))
d2zpdxy(i) = d2s/dxdy(xp(i),yp(i))
d2zpdyy(i) = d2s/dy2(xp(i),yp(i))
```

O parâmetro `out_mode` define a regra de avaliação para extrapolação, i.e., para *(xp(i),yp(i)) fora de [x(1),x(nx)]x[y(1),y(ny)]*:

"by_zero"
: uma extrapolação por zero é realizada

"by_nan"
: extrapolação por NaN

"C0"
: Extrapolação definida como segue :

```
s(x,y) = s(proj(x,y)) onde  proj(x,y) é o ponto mais próximo
                      em [x(1),x(nx)]x[y(1),y(ny)] de (x,y)
```

"natural"
a extrapolação é realizada utilizando o elemento de área bicúbico mais próximo de (x,y).

"periodic"
s é estendida por periodicidade.

# Exemplos

```
// veja os exemplos de splin2d

// este exemplo mostra características de extrapolações diferentes
// interpolação de cos(x)cos(y)
n = 7;  // um grid de interpolação n x n
x = linspace(0,2*%pi,n); y = x;
z = cos(x')*cos(y);
C = splin2d(x, y, z, "periodic");

// agora avaliando em um domínio maior que [0,2pi]x [0,2pi]
m = 80; // parâmetro de discretização do grid de avaliação
xx = linspace(-0.5*%pi,2.5*%pi,m); yy = xx;
[XX,YY] = ndgrid(xx,yy);
zz1 = interp2d(XX,YY, x, y, C, "C0");
zz2 = interp2d(XX,YY, x, y, C, "by_zero");
zz3 = interp2d(XX,YY, x, y, C, "periodic");
zz4 = interp2d(XX,YY, x, y, C, "natural");
xbasc()
subplot(2,2,1)
plot3d(xx, yy, zz1, flag=[2 6 4])
xtitle("Extrapolação com o outmode C0")
subplot(2,2,2)
plot3d(xx, yy, zz2, flag=[2 6 4])
xtitle("Extrapolação com o outmode by_zero")
subplot(2,2,3)
plot3d(xx, yy, zz3, flag=[2 6 4])
xtitle("Extrapolação com o outmode periodic")
subplot(2,2,4)
plot3d(xx, yy, zz4, flag=[2 6 4])
xtitle("Extrapolação com o outmode natural")
xselect()
```

# Ver Também

splin2d

# Autor

B. Pincon

# Name

interp3d — função de avaliação spline 3d

```
[fp[,dfpdx,dfpdy,dfpdz]]=interp3d(xp,yp,zp,tl,out_mode)
```

# Parâmetros

xp, yp, zp
: vetores ou matrizes de reais de mesmo tamanho

tl
: tlist do tipo "splin3d", definindo um tensor spline 3d (chamado `s` a partir daqui)

out_mode
: (opcional) string definindo a avaliação de `s` fora do grid ([xmin,xmax]x[ymin,ymax]x[zmin,zmax])

fp
: vetor ou matriz de mesmo formato que `xp`, `yp` e `zp`, avaliação elemento a elemento de `s` onesses pontos.

dfpdx, dfpdy, dfpdz
: vetores (ou matrizes) de mesmo formato que `xp`, `yp` e `zp`, avaliação elemento a elemento das primeiras derivadas de `s` nesses pontos.

# Descrição

Dada uma tlist `tl` definindo uma função spline 3d (ver splin3d) esta função avalia *s* (e *ds/dx*, *ds/dy*, *ds/dz* , se necessário) em *(xp(i),yp(i),zp(i))* :

```
zp(i) = s(xp(i),yp(i))
dzpdx(i) = ds/dx(xp(i),yp(i),zp(i))
dzpdy(i) = ds/dy(xp(i),yp(i),zp(i))
dzpdz(i) = ds/dz(xp(i),yp(i),zp(i))
```

O parâmetro `out_mode`define a regra de avaliação para extrapolação, i.e. para *(xp(i),yp(i),zp(i)) fora de [xmin,xmax]x[ymin,ymax]x[zmin,zmax]*:

"by_zero"
: uma extrapolação por zero é feita

"by_nan"
: extrapolação por NaN ("não é número")

"C0"
: a extrapolação definida como segue :

```
s(x,y) = s(proj(x,y)) onde  proj(x,y) é o ponto mais próximo
                       em [x(1),x(nx)]x[y(1),y(ny)] de (x,y)
```

"periodic"
: `s` é estendido por periodicidade.

# Exemplos

```
// veja exemplos na página da função splin3d
```

# Ver Também

splin3d, bsplin3val

# Autores

R.F. Boisvert, C. De Boor (código da biblioteca FORTRAN CMLIB)
B. Pincon (interface Scilab)

# Name

interpln — interpolação linear

```
[y]=interpln(xyd,x)
```

# Parâmetros

xyd
: matriz de duas linhas (coordenadas xy dos pontos)

x
: vetor (de abscissas)

y
: vetor (de ordenadas)

# Descrição

Dado `xyd` um conjunto de pontos no plano xy de abscissas crescentes e `x` aum conjunto de abscissas, esta função computa `y` , os valores correspondentes às ordenadas, por interpolação linear.

# Exemplos

```
x=[1 10 20 30 40];
y=[1 30 -10 20 40];
plot2d(x',y',[-3],"011"," ",[-10,-40,50,50]);
yi=interpln([x;y],-4:45);
plot2d((-4:45)',yi',[3],"000");
```

# Ver Também

splin, interp, smooth

# Name

intsplin — integração de dados experimentais por interpolação por spline

```
v = intsplin([x,] s)
```

# Parâmetros

x
: vetor de dados de coordenadas x crescentes. O valor padrão é `1:size(y,'*')`

s
: vetor de dados de coordenadas y

v
: valor da integral

# Descrição

Computa :

Onde f é uma função descrita por um conjunto de valores experimentais:

`s(i)=f(x(i))` e `x0=x(1)`, `x1=x(n)`

Entre os pontos da malha a função é interpolada usando-se splines.

# Exemplos

```
t=0:0.1:%pi
intsplin(t,sin(t))
```

# Ver Também

intg, integrate, inttrap, splin

# Name

linear_interpn — interpolação linear n-dimensional

```
vp = linear_interpn(xp1,xp2,..,xpn, x1, ..., xn, v [,out_mode])
```

# Parâmetros

xp1, xp2, .., xpn
: vetores de reais (ou matrizes) de mesmo tamanho

x1 ,x2, ..., xn
: vetores linhas estritamente crescentes (com pelo menos 2 elementos) definindo o grid de interpolação n-dimensional

v
: vetor (caso n=1), matriz (caso n=2) ou hipermatriz (caso n > 2) com valores da função subjacente interpolada nos pontos do grid.

out_mode
: (opcional) string definindo a avaliação fora do grid (extrapolação)

vp
: vetor ou matriz de mesmo tamanho que `xp1, ..., xpn`

# Descrição

Dado um grid n-dimensional definido pelos n vetores `x1 ,x2, ..., xn` e os valores `v` de uma função (aqui nomeada *f*) nos pontos do grid :

```
v(i1,i2,...,in) = f(x1(i1),x2(i2), ..., xn(in))
```

esta função computa o interpolante linear de *f* do grid (chamado de s a partir daqui) nos pontos nos quais as coordenadas são definidas pelos vetores (ou matrizes) `xp1, xp2, ..., xpn`:

```
vp(i) = s(xp1(i),xp2(i), ..., xpn(i))

or vp(i,j) = s(xp1(i,j),xp2(i,j), ..., xpn(i,j)) caso xpk sejam matrizes
```

O parâmetro `out_mode` ajusta a regra para extrapolação: se notarmos *Pi=(xp1(i),xp2(i),...,xpn(i))* , então `out_mode` define a regra de avaliação quando:

```
P(i) está fora de  [x1(1) x1($)] x [x2(1) x2($)] x ... x [xn(1) xn($)]
```

As escolhas são:

"by_zero"
: uma extrapolação por zero é feita

"by_nan"
: uma extrapolação por NaN

"C0"
: a extrapolação é definida como segue:

```
s(P) = s(proj(P)) onde  proj(P) é o ponto mais próximo de P
                  localizado na fronteira do grid.
```

"natural"
a extrapolação é feita usando o remendo n-linear mais próximo do ponto.

"periodic"
s é estendido por periodicidade.

# Exemplos

```
// exemplo 1 : interpolação linear 1d
x = linspace(0,2*%pi,11);
y = sin(x);
xx = linspace(-2*%pi,4*%pi,400)';
yy = linear_interpn(xx, x, y, "periodic");
xbasc()
plot2d(xx,yy,style=2)
plot2d(x,y,style=-9, strf="000")
xtitle("Interpolação linear de sin(x) com 11 pontos de interpolação")

// exemplo 2 : interpolação bilinear
n = 8;
x = linspace(0,2*%pi,n); y = x;
z = 2*sin(x')*sin(y);
xx = linspace(0,2*%pi, 40);
[xp,yp] = ndgrid(xx,xx);
zp = linear_interpn(xp,yp, x, y, z);
xbasc()
plot3d(xx, xx, zp, flag=[2 6 4])
[xg,yg] = ndgrid(x,x);
param3d1(xg,yg, list(z,-9*ones(1,n)), flag=[0 0])
xtitle("Interpolação bilinear de 2sin(x)sin(y)")
legends("pontos de interpolação",-9,1)
xselect()

// exemplo 3 : interpolação bilinear e experimentação
//             com todos os tipos de outmode
nx = 20; ny = 30;
x = linspace(0,1,nx);
y = linspace(0,2, ny);
[X,Y] = ndgrid(x,y);
z = 0.4*cos(2*%pi*X).*cos(%pi*Y);
nxp = 60 ; nyp = 120;
xp = linspace(-0.5,1.5, nxp);
yp = linspace(-0.5,2.5, nyp);
[XP,YP] = ndgrid(xp,yp);
zp1 = linear_interpn(XP, YP, x, y, z, "natural");
zp2 = linear_interpn(XP, YP, x, y, z, "periodic");
zp3 = linear_interpn(XP, YP, x, y, z, "C0");
zp4 = linear_interpn(XP, YP, x, y, z, "by_zero");
zp5 = linear_interpn(XP, YP, x, y, z, "by_nan");
xbasc()
subplot(2,3,1)
   plot3d(x, y, z, leg="x@y@z", flag = [2 4 4])
   xtitle("função inicial 0.4 cos(2 pi x) cos(pi y)")
```

```
subplot(2,3,2)
   plot3d(xp, yp, zp1, leg="x@y@z", flag = [2 4 4])
   xtitle("Natural")
subplot(2,3,3)
   plot3d(xp, yp, zp2, leg="x@y@z", flag = [2 4 4])
   xtitle("Periodic")
subplot(2,3,4)
   plot3d(xp, yp, zp3, leg="x@y@z", flag = [2 4 4])
   xtitle("C0")
subplot(2,3,5)
   plot3d(xp, yp, zp4, leg="x@y@z", flag = [2 4 4])
   xtitle("by_zero")
subplot(2,3,6)
   plot3d(xp, yp, zp5, leg="x@y@z", flag = [2 4 4])
   xtitle("by_nan")
xselect()

// exemplo 4 : interpolação trilinear (ver ajuda de splin3d
//             que tem os mesmos exemplos com
//             interpolação por spline tricúbico)
exec("SCI/demos/interp/interp_demo.sci")
func =  "v=(x-0.5).^2 + (y-0.5).^3 + (z-0.5).^2";
deff("v=f(x,y,z)",func);
n = 5;
x = linspace(0,1,n); y=x; z=x;
[X,Y,Z] = ndgrid(x,y,z);
V = f(X,Y,Z);
// computando (e exibindo) o interpolante linear em algumas fatias
m = 41;
dir = ["z="  "z="  "z="  "x="  "y="];
val = [ 0.1   0.5   0.9   0.5   0.5];
ebox = [0 1 0 1 0 1];

XF=[]; YF=[]; ZF=[]; VF=[];
for i = 1:length(val)
   [Xm,Xp,Ym,Yp,Zm,Zp] = slice_parallelepiped(dir(i), val(i), ebox, m, m, m);
   Vm = linear_interpn(Xm,Ym,Zm, x, y, z, V);
   [xf,yf,zf,vf] = nf3dq(Xm,Ym,Zm,Vm,1);
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
   Vp =  linear_interpn(Xp,Yp,Zp, x, y, z, V);
   [xf,yf,zf,vf] = nf3dq(Xp,Yp,Zp,Vp,1);
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
end
nb_col = 128;
vmin = min(VF); vmax = max(VF);
color = dsearch(VF,linspace(vmin,vmax,nb_col+1));
xset("colormap",jetcolormap(nb_col));
xbasc()
xset("hidden3d",xget("background"))
colorbar(vmin,vmax)
plot3d(XF, YF, list(ZF,color), flag=[-1 6 4])
xtitle("Interpolação trilinear de"+func)
xselect()
```

# Ver Também

interpln, splin, splin2d, splin3d

# Autor

B. Pincon

# Name

lsq_splin — ajuste ponderado por spline cúbico de mínimos quadrados

```
[y, d] = lsq_splin(xd, yd [, wd], x)
```

# Parâmetros

xd, yd
: vetores de mesmo tamanho, dados a serem ajustados por um spline cúbico

wd
: (opcional) um vetor de mesmo formato que `xd` e `yd`, pesos dos ajustes de mínimos quadrados.

x
: vetor (linha ou coluna) estritamente crescente, pontos de interrupção do spline cúbico

y, d
: vetores de mesmo formato que `x`, a tripla(x,y,d) define o spline cúbico aproximado.

# Descrição

Esta função computa um spline cúbico aproximado *s* para os dados `xd, yd, wd` (a partir daqui *m* é suposto como o comprimento desses vetores) e, de uma escolha dos pontos de interrupção do spline, o vetor `x` (por exemplo, se você deseja n pontos de interrupção uniformemente escolhidos você pode usar `x=linspace(min(xd),max(xd),n)`). Se *S* é o espaço de todas as funções spline cúbicas com pontos de interrupção *x1 < x2 < ... < xn* então, o spline resultante s é tal que:

```
_m_                                 _m_
\                            2      \                            2
/   wd(k) (s(xd(k)) - yd(k))  <=   /   wd(k) (f(xd(k)) - yd(k))
---                                 ---
k=1                                 k=1
```

para todo *f em S*, i.e., realiza o mínimo da soma de todos os erros quadrados sobre todas as funções de *S*.

O spline *s* é completamente definido pela tripla `(x,y,d)` (`y` e `d` são os vetores das ordenadas dos splines e das derivadas primeiras nos xi 's : *yi=s(xi)* e *di=s'(xi)*) e sua avaliação em alguns pontos deve ser feita pela função interp.

# Observações

Quando `wd` não é dado, todos os pontos têm o mesmo peso 1

Um ponto *(xd(k),yd(k))* é considerado no ajuste se *xd(k) em [x1,xn] e wd(k) > 0*. Em particular, você pode colocar um peso nulo (ou mesmo negativo) a todos os pontos que você deseja que sejam ignorados no ajuste. Quando o número total de pontos levados em conta no "procedure" de ajuste é (estritamente) menor que 4, ocorre um erro.

O vetor `xd` não necessita estar em ordem crescente

Dependendo do número e das posições dos *xd(k)* 's e da escolha dos *x(i)* 's podem haver várias soluções, mas apenas uma é selecionada. Quando isso ocorre, um aviso é exibido na janela de comando do Scilab. Esta função foi feita para ser usada quando *m* é muito maior que *n* e neste caso, tal tipo de problema não ocorre.

# Exemplos

```
// este é um exemplo artificial, onde os dados xd e yd
// são contruídos a partir de uma função seno perturbada
a = 0; b = 2*%pi;
sigma = 0.1;  // desvio padrão do ruído gaussiano
m = 200;       // número de pontos experimentais
xd = linspace(a,b,m)';
yd = sin(xd) + grand(xd,"nor",0,sigma);

n = 6; // número de pontos de interrupção
x = linspace(a,b,n)';

// computando o spline
[y, d] = lsq_splin(xd, yd, x);  // usando pesos iguais

// plotando
ye = sin(xd);
ys = interp(xd, x, y, d);
xbasc()
plot2d(xd,[ye yd ys],style=[2 -2 3], ...
       leg="função exata@medidas experimentais (perturbação gaussiana)@spline a
xtitle("Um spline de mínimos quadrados")
xselect()
```

# Ver Também

interp, splin

# Autores

C. De Boor, A.H. Morris (código da biblioteca FORTRAN NSWC)
B. Pincon (interface Scilab e ligeiras modificações)

# Name

smooth — suavização por funções splines

```
[pt]=smooth(ptd [,step])
```

# Parâmetros

ptd
: vetor de reais (2xn)

step
: real (passo de discretização das abscissas)

pt
: vetor de reais (2xn)

# Descrição

Esta função computa a interpolação por funções splines para um dado conjunto de pontos no plano. As coordenadas são `(ptd(1,i),ptd(2,i))`. Os componentes `ptd(1,:)` devem estar em ordem crescente. O valor padrão para o para step é `abs(maxi(ptd(1,:))-mini(ptd(1,:)))/100`

# Exemplos

```
x=[1 10 20 30 40];
y=[1 30 -10 20 40];
plot2d(x',y',[3],"011"," ",[-10,-40,50,50]);
yi=smooth([x;y],0.1);
plot2d(yi(1,:)',yi(2,:)',[1],"000");
```

# Ver Também

splin, interp, interpln

# Name

splin — interpolação por spline cúbico

```
d = splin(x, y [,spline_type [, der]])
```

# Parâmetros

x
: um vetor (linha ou coluna) estritamente crescente (x deve ter pelo menos dois componentes)

y
: um vetor com o mesmo formato que `x`

spline_type
: (opcional) um string selecionando o tipo de spline a ser computado

der
: (optional) um vetor com dois componentes, com as derivadas nas extremidades (a ser fornecido quando spline_type="clamped")

d
: vetor com o mesmo formato que `x` (`di` é a derivada do spline em `xi`)

# Descrição

Esta função computa o spline cúbico ou sub-spline *s* que interpola os pontos *(xi,yi)* i.e., temos *s(xi)=yi* para todo *i=1,..,n*. O spline resultante s é completamente definido pela tripla `(x,y,d)` onde `d` é o vetor com as derivadas nos `xi`: *s'(xi)=di* (esta forma é chamada de forma de *Hermite ou hermitiana* ). A avaliação do spline em alguns pontos deve ser feita pela função interp. Vários tipos de splines podem ser computados selecionando o parâmetro `spline_type` apropriado:

"not_a_knot"
: este é o caso padrão, o spline cúbico é computado sob as seguintes condições (considerando n pontos x1,...,xn):

```
s'''(x2-) = s'''(x2+)
s'''(x{n-1}-) = s'''(x{n-1}+)
```

"clamped"
: neste caso, o spline cúbico é computado usando derivadas nas extremidades do intervalo que devem ser fornecidas como último argumento `der`:

```
s'(x1) = der(1)
s'(xn) = der(2)
```

"natural"
: o spline cúbico é computado sob as seguintes condições:

```
s''(x1) = 0
s''(xn) = 0
```

"periodic"
: um spline cúbico periódico é computado (`y` deve verificar *y1=yn*) sob as seguintes condições:

```
s'(x1) = s'(xn)
s''(x1) = s''(xn)
```

"monotone"
: neste caso, um sub-spline (*s* é apenas uma vez continuamente diferenciável) é computado usando um esquema local para os *di* tais que s é monótono em cada intervalo:

```
if y(i) <= y(i+1)  s é crescente em [x(i), x(i+1)]
if y(i) >= y(i+1)  s é decrescente em [x(i), x(i+1)]
```

"fast"
: neste caso, um sub-spline também é computado usando um esquema local simples para os *di* : d(i) é a derivada em x(i) da interpolação polinomial (x(i-1),y(i-1)), (x(i),y(i)),(x(i+1),y(i+1)), exceto pelas extremidades (d1 sendo computado a partir dos 3 pontos mais à esquerda e dn dos 3 pontos mais à direita).

"fast_periodic"
: é o mesmo que o anterior, mas também usa uma fórmula centrada para *d1 = s'(x1) = dn = s'(xn)* através da periodicidade da função subjacente (`y` deve verificar *y1=yn*).

# Observações

De um ponto de vista de precisão, use essencialmente o tipo **clamped** se você conhece as derivadas nas extremidades, de outro modo, use **not_a_knot**. Mas se a função subjacente aproximada é periódica, use o tipo **periodic** Sob boas suposições, estes tipos de spline têm um comportamento assintótico `O(h^4)` do erro. Não use o tipo **natural** a não ser que a função subjacente possua zero derivadas segundas nas extremidades.

Os tipos **monotone**, **fast** (ou **fast_periodic**) podem ser úteis e alguns caso, por exemplo, para limitar oscilações (estes tipos de sub-splines têm um comportamento assintótico `O(h^3)` do erro).

Se *n=2* (e `spline_type` não é **clamped**) é usada interpolação linear. Se *n=3* e `spline_type` é **not_a_knot**, então, um sub-spline tipo **fast** é computado, na verdade.

# Exemplos

```
// exemplo 1
deff("y=runge(x)","y=1 ./(1 + x.^2)")
a = -5; b = 5; n = 11; m = 400;
x = linspace(a, b, n)';
y = runge(x);
d = splin(x, y);
xx = linspace(a, b, m)';
yyi = interp(xx, x, y, d);
yye = runge(xx);
xbasc()
plot2d(xx, [yyi yye], style=[2 5], leg="interpolação por@função exata")
plot2d(x, y, -9)
xtitle("Intepolação da função de Runge")

// exemplo 2 : mostra comportamento de splines diferentes em dados aleatórios
a = 0; b = 1;        // intervalo de interpolação
n = 10;              // número de pontos de interpolação
```

```
m = 800;             // discretização para avaliação
x = linspace(a,b,n)'; // abscissas dos pontos de interpolação
y = rand(x);          // ordenadas dos pontos de interpolação
xx = linspace(a,b,m)';
yk = interp(xx, x, y, splin(x,y,"not_a_knot"));
yf = interp(xx, x, y, splin(x,y,"fast"));
ym = interp(xx, x, y, splin(x,y,"monotone"));
xbasc()
plot2d(xx, [yf ym yk], style=[5 2 3], strf="121", ...
       leg="fast@monotone@spline not a knot")
plot2d(x,y,-9, strf="000")  // para mostrar pontos de interpolação
xtitle("Vários splines e sub-splines em dados aleatórios")
xselect()
```

# Ver Também

interp, lsq_splin

# Autores

B. Pincon
F. N. Fritsch (rotina pchim.f Slatec é usada para interpolação monótona)

# Name

splin2d — interpolação por spline bicúbico em grides 2d

```
C = splin2d(x, y, z, [,spline_type])
```

# Parâmetros

x,y
: vetores linhas estritamente crescentes (com pelo menos dois componentes) definindo o grid de interpolação

z
: nmatriz nx x ny (nx sendo o comprimento de `x` e ny o comprimento de `y`)

spline_type
: (opcional) um string selecionando o tipo de spline bicúbico a ser computado

C
: um vetor grande com coeficientes dos elementos de área bicúbicos (veja detalhes em "Observações")

# Descrição

Esta função computa um spline ou sub-spline bicúbico *s* que interpola os pontos *(xi,yj,zij)* i.e., temos *s(xi,yj)=zij* para todo *i=1,..,nx* e *j=1,..,ny*. O spline resultante s é definido pela tripla `(x,y,C)` onde `C` é o vetor (com comprimento 16(nx-1)(ny-1)) com os coeficientes de cada um dos (nx-1)(ny-1) elementos de área bicúbicos : em *[x(i) x(i+1)]x[y(j) y(j+1)]*, *s* é definido por :

```
          _4_   _4_
          \     \                    k-1       l-1
  s(x,y) = /     /    C (k,l) (x - xi)  (y - yj)
          ---   ---   ij
          k=1   l=1
```

A avaliação de s em alguns pontos deve ser feita pela função interp2d Vários tipos de splines podem ser computados selecionando o parâmetro `spline_type` apropriado. O método usada para computar os splines (ou sub-spline) bicúbicos é o mesmo do antigo, i.e., computar em cada ponto do grid *(xi,yj)* uma aproximação das derivadas primeiras *ds/dx(xi,yj)* e *ds/dy(xi,yj)* e das derivadas cruzadas *d2s/dxdy(xi,yj)*. Estas derivadas são computadas pelo modo dos esquemas do spline 1d levando a uma função de classe C2 (s é duas vezes continuamente diferenciável) ou através de um esquema de aproximação local, levando a uma função de classe C1 apenas. Este esquema é selecionado através do parâmetro `spline_type` (ver splin para detalhes) :

"not_a_knot"
: é o caso padrão

"periodic"
: usado se a função subjacente é periódica: deve-se ter *z(1,j) = z(nx,j) para todo j em [1,ny] e z(i,1) = z(i,ny) para i em [1,nx]* # mas isto não é verificado pela interface.

# Observações

De um ponto de vista de precisão, use essencialmente o tipo **not_a_knot** ou **periodic** se a função interpolada subjacente é periódica.

Os tipos **natural**, **monotone**, **fast** (ou **fast_periodic**) podem ser úteis em alguns casos, por exemplo para limitar oscilações (**monotone** é o mais poderoso para isso).

Para obter coeficientes dos remendos bicúbicos de um modo mais amigável você pode usar `c = hypermat([4,4,nx-1,ny-1],C)` então o coeficiente *(k,l)* do remendo *(i,j)* (ver equação aqui antes) é armazenado em `c(k,l,i,j)`. Não obstante, a função interp2d espera pelo vetor grande `C` e não pela hipermatriz `c` (note que se pode recuperar facilmente `C` de `c` com `C=c(:)`).

# Exemplos

```
// exemplo 1 : interpolação de cos(x)cos(y)
n = 7;  // um grid regular com n x n pontos de interpolação
        // será usado
x = linspace(0,2*%pi,n); y = x;
z = cos(x')*cos(y);
C = splin2d(x, y, z, "periodic");
m = 50; // parâmetro de discretização do grid de avaliação
xx = linspace(0,2*%pi,m); yy = xx;
[XX,YY] = ndgrid(xx,yy);
zz = interp2d(XX,YY, x, y, C);
emax = max(abs(zz - cos(xx')*cos(yy)));
xbasc()
plot3d(xx, yy, zz, flag=[2 4 4])
[X,Y] = ndgrid(x,y);
param3d1(X,Y,list(z,-9*ones(1,n)), flag=[0 0])
str = msprintf(" com %d x %d pontos de interpolação. ermax = %g",n,n,emax)
xtitle("Interpolação por spline de cos(x)cos(y)"+str)

// exemplo 2 : diferentes funções de interpolação em dados aleatórios
n = 6;
x = linspace(0,1,n); y = x;
z = rand(n,n);
np = 50;
xp = linspace(0,1,np); yp = xp;
[XP, YP] = ndgrid(xp,yp);
ZP1 = interp2d(XP, YP, x, y, splin2d(x, y, z, "not_a_knot"));
ZP2 = linear_interpn(XP, YP, x, y, z);
ZP3 = interp2d(XP, YP, x, y, splin2d(x, y, z, "natural"));
ZP4 = interp2d(XP, YP, x, y, splin2d(x, y, z, "monotone"));
xset("colormap", jetcolormap(64))
xbasc()
subplot(2,2,1)
plot3d1(xp, yp, ZP1, flag=[2 2 4])
xtitle("not_a_knot")
subplot(2,2,2)
plot3d1(xp, yp, ZP2, flag=[2 2 4])
xtitle("bilinear interpolation")
subplot(2,2,3)
plot3d1(xp, yp, ZP3, flag=[2 2 4])
xtitle("natural")
subplot(2,2,4)
plot3d1(xp, yp, ZP4, flag=[2 2 4])
xtitle("monotone")
xselect()

// exemplo 3 : spline not_a_knot e sub-spline monotone
//             em uma função de degraus
```

```
a = 0; b = 1; c = 0.25; d = 0.75;
// criando grid de interpolação
n = 11;
x = linspace(a,b,n);
ind = find(c &lt;= x &amp; x &lt;= d);
z = zeros(n,n); z(ind,ind) = 1;  // um degrau dentro de um quadrado
// criando grid de avaliação
np = 220;
xp = linspace(a,b, np);
[XP, YP] = ndgrid(xp, xp);
zp1 = interp2d(XP, YP, x, x, splin2d(x,x,z));
zp2 = interp2d(XP, YP, x, x, splin2d(x,x,z,"monotone"));
// plot
xbasc()
xset("colormap",jetcolormap(128))
subplot(1,2,1)
plot3d1(xp, xp, zp1, flag=[-2 6 4])
xtitle("spline (not_a_knot)")
subplot(1,2,2)
plot3d1(xp, xp, zp2, flag=[-2 6 4])
xtitle("subspline (monotone)")
```

# Ver Também

cshep2d, linear_interpn, interp2d

# Autor

B. Pincon

# Name

splin3d — interpolação spline em grides 3d

```
tl = splin3d(x, y, z, v, [order])
```

# Parâmetros

x,y,z
: vetores linhas estritamente crescentes (cada um com pelo menos três componentes) definindo o grid de interpolação 3d

v
: uma hipermatriz nx x ny x nz (nx, ny, nz sendo de comprimentos de `x`, `y` e `z`, respectivamente)

order
: (opcional) um vetor 1x3 [kx,ky,kz] dada a ordem do tensor spline em cada direção (o padrão é [4,4,4], i.e. spline tricúbico)

tl
: um tlist do tipo splin3d definindo o spline

# Descrição

Esta função computa um spline tensor 3d *s* que interpola os pontos *(xi,yj,zk,vijk)*, i.e., temos *s(xi,yj,zk)=vijk* para todo *i=1,..,nx*, *j=1,..,ny* e *k=1,..,nz*. O spline resultante *s* é definido por `tl` que consiste em uma representação B-tensor-spline de *s*. A avaliação de s em alguns pontos deve ser feita pela função interp3d (para computar s e suas derivadas primeiras) ou pela função bsplin3val (para computar uma derivada arbitrária de s) . Vários tipos de splines podem ser computados selecionando a ordem dos splines em cada direção `order=[kx,ky,kz]`.

# Observação

Esta função funciona sob as condições:

```
nx, ny, nz <= 3
2 <= kx < nx
2 <= ky < ny
2 <= kz < nz
```

Há erro quando estas condições não são respeitadas.

# Exemplos

```
// exemplo 1
// ==========================================================================

func =  "v=cos(2*%pi*x).*sin(2*%pi*y).*cos(2*%pi*z)";
deff("v=f(x,y,z)",func);
n = 10;  // n x n x n  pontos de interpolação
x = linspace(0,1,n); y=x; z=x; // grid de interpolação
[X,Y,Z] = ndgrid(x,y,z);
V = f(X,Y,Z);
tl = splin3d(x,y,z,V,[5 5 5]);
```

```
m = 10000;
// computando um erro aproximado
xp = grand(m,1,"def"); yp = grand(m,1,"def"); zp = grand(m,1,"def");
vp_exact = f(xp,yp,zp);
vp_interp = interp3d(xp,yp,zp, tl);
er = max(abs(vp_exact - vp_interp))
// tente agora com n=20 e veja o erro

// exemplo 2 (veja a página de ajuda de linear_interpn que contém o
//            mesmo exemplo com interpolação trilinear)
// ==============================================================================

exec("SCI/modules/interpolation/demos/interp_demo.sci")
func =  "v=(x-0.5).^2 + (y-0.5).^3 + (z-0.5).^2";
deff("v=f(x,y,z)",func);
n = 5;
x = linspace(0,1,n); y=x; z=x;
[X,Y,Z] = ndgrid(x,y,z);
V = f(X,Y,Z);
tl = splin3d(x,y,z,V);
// computando e exibindo o spline interpolante 3d em algumas fatias
m = 41;
dir = ["z="  "z="  "z="  "x="  "y="];
val = [ 0.1   0.5   0.9   0.5   0.5];
ebox = [0 1 0 1 0 1];
XF=[]; YF=[]; ZF=[]; VF=[];
for i = 1:length(val)
   [Xm,Xp,Ym,Yp,Zm,Zp] = slice_parallelepiped(dir(i), val(i), ebox, m, m, m);
   Vm = interp3d(Xm,Ym,Zm, tl);
   [xf,yf,zf,vf] = nf3dq(Xm,Ym,Zm,Vm,1);
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
   Vp = interp3d(Xp,Yp,Zp, tl);
   [xf,yf,zf,vf] = nf3dq(Xp,Yp,Zp,Vp,1);
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
end
nb_col = 128;
vmin = min(VF); vmax = max(VF);
color = dsearch(VF,linspace(vmin,vmax,nb_col+1));
xset("colormap",jetcolormap(nb_col));
xbasc(); xset("hidden3d",xget("background"));
colorbar(vmin,vmax)
plot3d(XF, YF, list(ZF,color), flag=[-1 6 4])
xtitle("interpolação por spline 3d da função "+func)
xselect()
```

# Ver Também

linear_interpn, interp3d, bsplin3val

# Autores

R.F. Boisvert, C. De Boor (código da biblioteca FORTRAN CMLIB)
B. Pincon (interface Scilab)

# Parte XIX. Funções de Entrada/Saída

# Name

file — Gerenciamento de arquivos

```
[unit [,err]]=file('open', file-name [,status] [,access [,recl]] [,format])
file(action,unit)
[units [,typ [,nams [,mod [,swap]]]]] = file([unit])
```

# Parâmetros

file-name
: string, nome o arquivo a ser aberto

status
: string, o status do arquivo a ser aberto

"new"
: o arquivo não deve existir, novo arquivo (padrão)

"old"
: o arquivo já deve existir

"unknown"
: status desconhecido

"scratch"
: o arquivo deve ser deletado ao fim da sessão

access
: string, o tipo de acesso ao arquivo

"sequential"
: acesso seqüencial (padrão)

"direct"
: acesso direto

format
: string,

"formatted"
: para um arquivo formatado (padrão)

"unformatted"
: registro binário

recl
: inteiro, é o tamanho de registros quando `access="direct"`

unit
: inteiro, descritor de unidade lógica do arquivo aberto

units
: vetor de inteiros, descritores de unidades lógicas dos arquivos abertos. As unidades 5 e 6 são reservadas pelo sistema para dispositivos de entrada e saída.

typs
: vetor de strings, tipo (C ou Fortran) dos arquivos abertos.

nams
: vetor de strings, endereços dos arquivos abertos.

mod
: modo de abertura de arquivo. Formado por três dígitos abc

  Arquivos Fortran

  a
  : 0 significa formatado e 1 não formatado (binário)

  b
  : 0 significa acesso sequencial e 1 acesso direto

  c
  : 0 significa "new", 1 significa "old", 2 significa "scratch" e 3 significa "unknown"

  Arquivos C

  a
  : é 1 se o arquivo foi aberto pelo modo "b" (binário)

  b
  : é 1 se o arquivo foi aberto pelo modo "+" (atualização)

  c
  : 1 significa "r" (leitura), 2 significa "w" (escrita) e 3 significa "a" (anexação)

swap
: alavanca de troca automática. swap=1 se a troca automática estiver ativada. swap é sempre 0 para arquivos Fortran.

err
: inteiro, número de mensagem de erro (ver error), se a abertura falha. Se err for omitido, uma mensagem de erro aparece.

action
: é um dos seguintes strings:

  "close"
  : fecha os arquivos fornecidos pelos descritores de unidades lógicas fornecidos em `units`

  "rewind"
  : coloca o ponteiro no início do arquivo

  "backspace"
  : coloca o ponteiro no início do último registro.

  "last"
  : coloca o ponteiro após o último registro.

# Descrição

Seleciona uma unidade lógica `unit` e gerencia o arquivo `file-name`.

`[unit [,err]]=file('open', file-name [,status] [,access [,recl]][,format])` permite abrir um arquivo com propriedades especificadas e obter o número de unidade associado `unit`. Este número de unidade pode ser utilizado para ações futuras no arquivo, ou como descritor de arquivo em chamadas às funções `read`, `write`, `readb`, `writb`,`save`, `load`.

`file(action,unit)` permite fechar o arquivo, ou mover o ponteiro do arquivo corrente.

`file()` retorna os descritores de unidades lógicas dos arquivos abertos. So `file('close',file() )` fecha todos os arquivos do usuário abertos (tipo C ou Fortran).

# Exemplos

```
u=file('open',TMPDIR+'/foo','unknown')
for k=1:4
  a=rand(1,4)
  write(u,a)
end
file('rewind',u)
x=read(u,2,4)
file('close',u)
//
u1=file('open',TMPDIR+'/foo','unknown')
u2=mopen(TMPDIR+'/foo1','wb')
[units,typs,nams]=file()
file('close',u1);
mclose(u2);
```

# Ver Também

save, load, write, read, writb, readb, uigetfile, mopen, mclose

# Name

getenv — retorna o valor de uma variável de ambiente

```
env=getenv(str [, rep] )
```

# Parâmetros

str
: string especificando o nome da variável de ambiente; rep : um string opcional. Quando este valor opcional é usado, a função `getenv` retorna o valor `rep` quando a variável de ambiente `str` não é encontrada.

env
: string que contém o valor da variável de ambiente

# Descrição

Retorna o valor de uma variável de ambiente, se existir.

# Exemplos

```
getenv('SCI')
getenv('FOO','foo')
```

# Name

getio — retorno de unidades lógicas de entrada/saída do Scilab

```
ios=getio()
```

# Parâmetros

ios
: um vetor [rio rte wio wte]

rio
: unidade lógica corrente para leitura de instruções

rte
: unidade lógica designada para entrada na janela do Scilab

wio
: unidade lógica relativa ao arquivo de diário, se houer. wio=0 significa que nenhum arquivo de diário foi aberto

wte
: unidade lógica designada para saída na janela do Scilab

# Descrição

`getio` retorna unidades lógicas designadas para entrada e saída principais no Scilab.

# Ver Também

file, exec

# Name

getpid — Retorna o identificador do processo Scilab

```
id=getpid()
```

# Descrição

Retorna retorna um inteiro correspondente ao identificador do processo Scilab.

# Exemplos

```
d='SD_'+string(getpid())+'_'
```

# Name

getscilabkeywords — retorna uma lista com todas as palavras-chave do Scilab

```
list_keywords=getscilabkeywords()
```

# Parâmetros

list_keywords
uma lista

# Descrição

list_keywords(1) : primitivas

list_keywords(2) : comandos

list_keywords(3) : variáveis predefinidas

list_keywords(4) : funções Scilab

list_keywords(5) : funções Scicos

# Autores

A.C, adaptado do código de Enrico Segre's no Scipad.

# Name

halt — para execução

```
halt()
halt('a message')
```

# Descrição

Para execução até a entrada de algo no teclado.

# Exemplos

```

halt('Press a key')

halt()
```

# Ver Também

pause, return, exec

# Name

host — execução de comandos do Unix ou DOS

```
stat=host(command-name)
```

# Parâmetros

command-name
: um string contendo uma instrução sh Unix

stat
: um flag ("indicador") inteiro

# Descrição

Envia um string `command-name` para o Unix para execução pelo interpretador de comando (sh em Unix, ou command.com em DOS). As saídas e os erros padrões do comando do shell são escritos no shell chamado. `stat` fornece -1 se o "host" (significa "anfitrião") não pode ser chamado (memória disponível do sistema insuficiente) ou se o interpretador de comando retorna o código.

# Exemplos

```
//criando uma função getdir baseda no host
function wd=getdir()
  if MSDOS then
    host('cd>'+TMPDIR+'\path');
  else
    host('pwd>'+TMPDIR+'/path');
  end
  wd=read(TMPDIR+'/path',1,1,'(a)')
endfunction
//chamando-a
wd=getdir()
```

# Ver Também

edit, manedit, unix_g, unix_s, unix_w, unix_x

# Name

input — prompt para entrada do usuário

```
x = input(message [, "string"])
```

# Parâmetros

message
: string

"string"
: string `"string"` (pode ser abreviado para `"s"`)

x
: número real (ou string se `"string"` estiver na seqüência de chamamento)

# Descrição

`input(message)` fornece ao usuário o prompt no string de texto e então espera por entrada no teclado. A entrada pode ser a expressão avaliada por evstr. Se apenas um retorno-de-carro for fornecido no prompt, `input(message)` retorna uma matriz vazia.

Chamado com dois argumentos, a saída é um string que é a expressão fornecida pelo teclado. Se apenas um retorno de carro for fornecido, `input(message)` retorna um único espaço em branco `" "`.

# Exemplos

```
//x=input("Quantas iterações")
//x=input("Qual o seu nome?","string")
```

# Ver Também

evstr, x_dialog, x_mdialog

# Name

load — carrega variável salva

```
load(filename [,x1,...,xn])
load(fd [,x1,...,xn])
```

# Parâmetros

filename
: string contendo o endereço do arquivo

fd
: descritor de arquivo fornecido por uma chamada a mopen

xi
: nome(s) de variáveis Scilab arbitrárias fornecidos como strings

# Descrição

O comando `load` pode ser utilizado para recarregar no Scilab variáveis de sessão previamente gravadas com o comando `save`. Se o arquivo contém variáveis correspondentes a manipuladores gráficos, as graphics_entities (entidades gráficas) correspondentes são desenhados.

Desde o Scilab 5.0, todos os manipuladores uimenu ou uicontrol também são desenhados.

`load(filename)` carrega variáveis no arquivo dado pelo endereço `filename`.

`load(fd)` carrega variáveis no arquivo fornecido pelo seu descritor `fd`.

`load(filename,'x','y')` ou `load(fd,'x','y')` carrega apenas as variáveis `x,y`.

Mesmo que o formato de arquivo binário tenha mudado na versão 2.5, `load(filename,...)` é capaz de ler formatos antigos. Formatos antigos podem ser acessados por enquanto utilizando as funções oldsave e oldload.

# Exemplos

```
a=eye(2,2);b=ones(a);
save('vals.dat',a,b);
clear a
clear b
load('vals.dat','a','b');
```

# Ver Também

save, listvarinfile, save_format, exec, mopen

# Name

oldload — carrega variáveis salvas em formato 2.4.1 e anteriores

```
oldload('file-name' [,x1,...,xn])
```

# Parâmetros

file-name
: string

xi
: nome(s) de variável(eis) arbitrárias do Scilab fornecidos como strings

# Descrição

A função `oldload` está obsoleta e esta sendo retida apenas para propósitos de compatibilidade.

O comando `oldload` pode ser usado para recarregar no Scilab variáveis de sessão previamente salvas em um arquivo através do comando `save`.

`oldload('file-name')` carrega as variáveis em um arquivo salvo `'file-name'`.

`oldload('file-name','x','y',...,'z')` carrega apenas variáveis `x,y,...,z` armazenadas no arquivo `'file-name'`.

# Exemplos

```
a=eye(2,2);b=ones(a);
oldsave(TMPDIR+'/vals.dat',a,b);
clear a
clear b
oldload(TMPDIR+'/vals.dat','a','b');
```

# Ver Também

save, exec

# Name

oldsave — gravação de variáveis em formato 2.4.1 e anteriores

```
oldsave(filename [,x1,x2,...,xn])
```

# Parâmetros

filename
: string ou unidade lógica retornada por file('open',...)

xi
: variável(is) Scilab arbtrária(s)

# Descrição

A função `oldsave` está obsoleta e retida apenas por compatibilidade.

O comando `oldsave` pode ser usado para salvar variáveis Scilab correntes em forma binária num arquivo.

`oldsave(filename)` salva todas as variáveis correntes no arquivo definido por `filename`.

`oldsave(file-name,x,y)` salva apenas as variáveis nomeadas `x` e `y`.

Variáveis salvas podem ser recarregadas através dos comandos `load` ou oldload.

# Exemplos

```
a=eye(2,2);b=ones(a);
oldsave('TMPDIR/val.dat',a,b);
clear a
clear b
oldload('TMPDIR/val.dat','a','b');
```

# Ver Também

load, file

# Name

read — Leitura de matrizes

```
[x]=read(file-desc,m,n,[format])
[x]=read(file-desc,m,n,k,format)
```

# Parâmetros

file-desc
: string especificando o nome do arquivo ou valor inteiro especificando uma unidade lógica (ver file).

m, n
: inteiros (dimensões da matriz x). Ajuste m=-1 se você desconhece o número de linhas, para que todo o arquivo seja lido.

format
: string, especifica um formato "Fortran". Este string deve começar com um parêntese direito e terminar com um parêntese esquerdo. Formatos não podem misturar pontos flutuantes, inteiros ou modo de edição de caractere

k
: inteiro ou vetor de inteiros

# Descrição

Lê linha após linha a matriz x `mxn` (`n=1` para cadeia de caracteres) no arquivo `file-desc` (string ou inteiro). Cada linha da matriz `x` começa em uma nova linha do arquivo `file-desc`. Dependendo de `format`, uma dada linha da matriz `x` pode ser lida de mais de uma linha do arquivo `file-desc`.

O tipo do resultado dependerá do formato especificado. Se format contém apenas os descritores `(d,e,f,g)` a função tenta ler dados numéricos (o resultado é uma matriz de números reais).

Se format contém apenas o descritor `a` a função tenta ler strings (o resultado é um vetor coluna de strings). Neste caso, n deve ser igual a 1. Aviso: Os strings serão truncados se tiverem tamanho maior que 4093.

Exemplos para `format`:

```
(1x,e10.3,5x,3(f3.0))
(10x,a20)
```

Quando format é o omitido, os dados são lidos utilizando-se o formato numérico livre: espaços em branco, vírgulas e barras podem ser utilizados como separadores de dados, n*v pode ser utilizado para representar n ocorrências do valor n.

Um arquivo de acesso direto pode ser utilzado na presença de um parâmetro `k` que é um vetor de números de gravações a serem lidas (uma gravação por linha), logo `m` deve ser `m=prod(size(k))`.

Para ler no teclado use `read(%io(1),...)`.

# Observação

A última linha de cada arquivo de dados devemser terminada por uma nova linha (newline) a ser levada em conta.

# Exemplos

```
if MSDOS then unix('del foo');
else unix('rm -f foo'); end
A=rand(3,5); write('foo',A);
B=read('foo',3,5)
B=read('foo',-1,5)
read(%io(1),1,1,'(a)')  // espera por entrada do usuário
```

# Ver Também

file, readb, write, x_dialog, mscanf, mfscanf, msscanf, fscanfMat

# Name

read4b — leitura de arquivo FORTRAN binário

```
x=read4b(file-name,m,n [,rec])
```

# Parâmetros

file-name
: string ou inteiro

m, n
: inteiros (dimensões da matriz x). Ajuste m=-1 se você não sabe o número de linhas, então todo o arquivo será lido

rec
: vetor de inteiros positivos, os registros selecionados para acesso direto. O tamanho desejado deve ter o mesmo tamanho de `x` desejado.

# Descrição

Leitura binária da matriz `x` no arquivo `file-name`. Supõe-se que as entradas da matriz foram armazenadas em palavras de quatro bytes.

Para acesso a registros direto, o arquivo deve ter sido previamente aberto com a função `file` para se ajustar o comprimento do registro. `file-name` deve ser o resultado da função `file`.

# Ver Também

file, write, writb, mget, write4b

# Name

readb — leitura de arquivo FORTRAN binário

```
x=readb(file-name,m,n [,rec])
```

# Parâmetros

file-name
: string ou inteiro

m, n
: inteiros (dimensões da matriz x). Ajuste m=-1 se o número de linhas não for conhecido, então todo arquivo será lido

rec
: vetor de inteiros positivos, os registros selecionados para acesso direto. O tamanho deste vetor deve ser igual ao número de linhas de `x` desejado.

# Descrição

Leitura binária da matriz `x` no arquivo `file-name`. Supõe-se que as entradas da matriz tenham sido armazenadas em palavras de 8 bytes.

Para acesso direto a registros, o arquivo deve ter sido aberto previamente com a função `file` para ajustar o comprimento de registro. `file-name` deve ser o resultado da função `file`.

# Ver Também

file, write, writb, mget, read4b

# Name

readc_ — lê um string

```
[c]=readc_(unit)
[c]=readc_()
```

# Descrição

`readc_` lê um string. Esta função permite interromper um arquivo exec sem `pause`; o arquivo exec pára até que seja dado um retorno de carro.

# Ver Também

read

# Name

save — salvando variáveis em arquivos binários

```
save(filename [,x1,x2,...,xn])
save(fd [,x1,x2,...,xn])
```

# Parâmetros

filename
: string contendo o endereço do arquivo

fd
: descritor do arquivo fornecido por uma chamada a mopen

xi
: variável(is) Scilab arbitrárias

# Descrição

O comando `save` pode ser usado para salvar as variáveis Scilab correntes em um arquivo. Se uma variável é um manipulador gráfico, a função `save` salva todas as definições de graphics_entities (entidades gráficas) correspondentes.

Desde o Scilab 5.0, todos os manipuladores de uimenu ou uicontrol também são gravados por esta função.

O arquivo pode ser fornecido tanto pelo seu endereço quanto por seu descritor dado previamente por `mopen`.

`save(filename)` salva todas as variáveis correntes definidas por `filename`.

`save(fd)` salva todas as variáveis correntes definidas pelo descritor `fd`.

`save(filename,x,y)` ou `save(fd,x,y)` salva apenas as variáveis chamadas `x` e `y`.

Variáveis salvas podem ser recarregadas através do comando `load`.

# Exemplos

```
a=eye(2,2);b=ones(a);
save('val.dat',a,b);
clear a
clear b
load('val.dat','a','b');

// gravação seqüêncial em um arquivo
fd=mopen('TMPDIR/foo','wb')
for k=1:4, x=k^2;save(fd,x,k),end
mclose(fd)
fd=mopen('TMPDIR/foo','rb')
for i=1:4, load(fd,'x','k');x,k,end
mclose(fd)

// anexando variáveis a um arquivo gravado antigo
fd=mopen('TMPDIR/foo','r+')
mseek(0,fd,'end')
```

```
lst=list(1,2,3)
save(fd,lst)
mclose(fd)
```

# Ver Também

load, save_format, mopen

# Name

setenv — ajusta o valor de uma variável de ambiente

```
rep=setenv(name, value )
```

# Parâmetros

name
: aponta para o nome de uma variável de ambiente. (name é um string)

value
: aponta para o valor a ser atribuído à variável de ambiente (value é um string)

rep
: Retorna %T, se estiver tudo correto ou %F, em caso contrário

# Descrição

Ajusta o valor de uma variável de ambiente.

# Exemplos

```
setenv('toto','example')
getenv('toto')
```

# Ver Também

getenv

# Autor

Allan CORNET

# Name

unix — execução de comando shell (sh)

```
stat=unix(command-name)
```

# Parâmetros

command-name
: string contendo uma instrução Unix sh

stat
: flag inteiro

# Descrição

Envia um string `command-name` ao Unix para execução com o shell sh. Saída e erros padrões do shell de comando são escritos no shell chamado. `stat` retorna -1 se o Unix não pode ser chamado (memória do sistema disponível insuficiente) ou o código de retorno sh.

# Exemplos

```
if ~MSDOS then
  unix("ls $SCI/demos");
end

function wd=directory()
  if MSDOS then
    unix('cd&gt;'+TMPDIR+'\path');
  else
    unix('pwd&gt;'+TMPDIR+'/path');
  end
  wd=read(TMPDIR+'/path',1,1,'(a)');
endfunction

wd=directory()
```

# Ver Também

edit, manedit, unix_g, unix_s, unix_w, unix_x, host

# Name

unix_g — execução de um comando shell (sh), saída redirecionada a uma variável

```
rep=unix_g(cmd)
[rep,stat]=unix_g(cmd)
[rep,stat,stderr]=unix_g(cmd)
```

# Parâmetros

cmd
: string

rep
: vetor coluna de strings (saída padrão)

stat
: inteiro o status de erro. stat=0 se nenhum erro tiver ocorrido

err
: vetor coluna de strings (erro padrão)

# Descrição

Envia um string `cmd` ao Unix para execução pelo comando shell (sh). A saída padrão é redirecionada para a variável Scilab `rep`. O erro padrão é redirecionado à variável Scilab err ou é exibido se você forneceu apenas dois argumentos de saída. Erros de execução Unix são detectados. Note que apenas o último erro de comando shell é reportado quando uma lista de comandos separados por ";" é enviada. Isto não é recomentdado.

# Exemplos

```
function d=DIR(path)
  path=pathconvert(path,%t,%t)
  if MSDOS then
    d=unix_g('dir '+path)
  else
    d=unix_g('ls '+path)
  end
endfunction

DIR('SCI/etc')
```

# Ver Também

unix_s, unix_w, unix_x, unix

# Name

unix_s — execução do comando shell (sh), sem saída

```
unix_s(cmd)
```

# Parâmetros

cmd
: string

# Descrição

Envia um string `cmd` ao Unix para execução pelo shell sh. A saída padrão é redirecionada para /dev/null. Erros de execução Unix são detectados; Note que apenas o último erro de comando shell é reportado quando uma lista de comandos separados por ";" é enviada: isto não é recomendado.

# Exemplos

```
if MSDOS then
  unix_s("del foo");
else
  unix_s("rm -f foo");
end
```

# Ver Também

edit, manedit, unix_g, unix_w, unix_x, unix

# Name

unix_w — execução de comando shell (sh), saída redirecionada à janela do Scilab

```
unix_w(cmd)
```

# Parâmetros

cmd
: string

# Descrição

Envia um string `cmd` ao Unix para execução com o comando shell (sh). A saída é redirecionada à janela do Scilab. Erros de execução Unix são detectados. Note que apenas o último erro de comando shell é reportado quando uma lista de comandos separados por ";" é enviada: isto não é recomandado.

# Exemplos

```
if MSDOS then
  unix_w("dir "+'"'"+WSCI+"\modules"+'"');
else
  unix_w("ls $SCI/modules");
end
```

# Ver Também

edit, manedit, unix_g, unix_s, unix_x, unix

# Name

unix_x — execução do comando shell (sh), saída redirecionada a uma janela

```
unix_x(cmd)
```

# Parâmetros

cmd
string

# Descrição

Envia um string `cmd` ao Unix para uma execução através do shel sh. A saída padrão é redirecionada a uma janela. Erros de execução Unix são detectados. Note que apenas o último erro de comando de shell é reportado quando uma lista de comandos separados por";" é enviada. Isto não é recomendado.

# Exemplos

```
if MSDOS then
  unix_x("dir "+"""+WSCI+"modules\graphics\demos"+""");
else
  unix_x("ls $SCI/modules/graphics/demos");
end
```

# Ver Também

edit, manedit, unix_g, unix_s, unix_w, unix

# Name

writb — escrita binária de arquivo FORTRAN

```
writb(file-name,a [,rec])
```

# Parâmetros

file-name
: string ou inteiro

rec
: vetor de inteiros positivos, os registros selecionados para acesso direto. O tamanho deste vetor deve ser igual ao número de linhas de `a`

# Descrição

Escreve em formato binário a matriz `a` no arquivo `'filename'`. As entradas da matriz são armazenadas em palavras de quatro bytes.

Para registro de acesso direto, o arquivo deve ter sido previamente aberto através da função `file` para se ajustar o comprimento do registro. `file-name` deve ser o resultado da função `file`.

# Ver Também

file, readb, write, mput, write4b

# Name

write — Escrita em arquivo formatado

```
write(file-desc,a,[format])
write(file-desc,a,k,format)
```

# Parâmetros

file-desc
: string especificando o nome do arquivo ou valor inteiro especificando a unidade lógica (ver file).

a
: matriz de reais ou vetor coluna de strings

format
: string, especifica um formato "Fortran". Este string deve começar com um parêntese direito e terminar com um parêntese esquerdo. Formatos não podem misturar pontos flutuantes, inteiros ou modo de edição de caractere.

k
: integer vector

# Descrição

Escreve linha por linha uma matriz de reais ou um vetor coluna de strings em um arquivo formatado. Cada linha do argumento `a` começa em uma nova linha do arquivo `file-desc`. Dependendo de `format` uma dada linha do argumento `a` pode ser escrita em mais de uma linha do arquivo `file-desc`.

Exemplos de formatos : `(1x,e10.3,5x,3(f3.0))` , `(10x,a20)` ;

Veja um livro de Fortran para maior precisão.

Arquivos de acesso direto : `x=write(file_desc,a,k,format)`. Aqui `k` é o vetor de gravações (uma gravação por linha, i.e. `m=prod(size(k))`)

`write(%io(2),....)` escreve em uma janela do Scilab. Note que neste caso `format` deve produzir uma linha de saída por linha da matriz. Se esta restrição não for respeitada, um erro imprevisível acontece.

# Exemplos

```
if MSDOS then unix('del asave');
else unix('rm -f asave'); end
A=rand(5,3); write('asave',A); A=read('asave',5,3);
write(%io(2),A,'('' | '',3(f10.3,'' | ''))')
write(%io(2),string(1:10))
write(%io(2),strcat(string(1:10),','))
write(%io(2),1:10,'(10(i2,3x))')

if MSDOS then unix('del foo');
else unix('rm -f foo'); end
write('foo',A)
```

# Ver Também

file, fileinfo, writb, read, print, string, mfprintf, mprintf, msprintf, fprintfMat

# Name

write4b — escrita binária de arquivo FORTRAN

```
write4b(file-name,a [,rec])
```

# Parâmetros

file-name
: string ou inteiro

rec
: vetor de inteiros positivos, os registros selecionados para acesso direto. O tamanho deste vetor deve ter o mesmo número de linhas que `a`

# Descrição

Escreve em formato binário a matriz `a` no arquivo `'filename'`. As entradas da matriz são armazenadas em palavras de 8 bytes.

Para acesso direto, o arquivo deve ter sido previamente aberto através da função `file` para se ajustar o comprimento do registro. `file-name` deve ser o resultado da função `file`.

# Ver Também

file, readb, write, mput, read4b

# Parte XX. Funções de Saída

# Name

disp — exibição de variáveis

```
disp(x1,[x2,...xn])
```

# Descrição

Exibe `xi` com o formato corrente. `xi`'s são objetos arbitrários (matrizes de constantes, strings, funções, listas, ...)

A exibição de objetos definidos por `tlist`'s pode ser sobrecarregada pela definição de uma função. Esta função não deve ter argumento de saída, um único argumento de entrada e seu nome deve ser formado como segue: `%<tlist_type>_p` onde `%<tlist_type>` é a primeira entrada do componente de tipo tlist.

A função lines pode ser usada para controlar a saída.

# Exemplos

```
disp([1 2],3)
deff('[]=%t_p(l)','disp(l(3),l(2))')
disp(tlist('t',1,2))
```

# Ver Também

lines, write, read, print, string, tlist

# Name

print — imprime variáveis em um arquivo

```
print('file-name',x1,[x2,...xn])
```

# Descrição

Imprime `xi` no arquivo `'file-name'` com o formato corrente, i.e., o formato utilizado pelo Scilab para exibir variáveis. Todos os tipos de variáveis podem ser impressos.

Note que `xi` deve ser uma variável nomeada. Com expressões, a exibição da parte do nome da variável é imprevisível.

`print(%io(2),...)` imprime na janela do Scilab. Esta sintaxe pode ser utilizada para exibir variáveis dentro de um macro.

# Exemplos

```
a=rand(3,3);p=poly([1,2,3],'s');l=list(1,'asdf',[1 2 3]);
print(%io(2),a,p,l)
write(%io(2),a)
```

# Ver Também

write, read, format, printf, disp

# Name

printf — emulador da função da linguagem C printf

```
printf(format,value_1,..,value_n)
```

# Parâmetros

format
: string Scilab. especifica um string combinando caracteres literais com especificações de conversão.

value_i
: especifica os dados a serem convertidos de acordo com o parâmetro de formato.

str
: vetor coluna de strings

file
: string Scilab especificando um arquivo ou um número de unidade lógica (ver `file`)

# Descrição

A função `printf` converte, formata e escreve seus parâmetros `value` sob controle do parâmetro `format` para a saída padrão.

O parâmetro `format` é um string que contém dois tipos de objetos:

Caracteres literais
: que são copiados no fluxo de saída

Especificações de conversão
: cada um dos quais faz com que zero ou mais itens sejam buscados da lista de parâmetros `value`. Ver printf_conversion para detalhes.

Se algum item de `value`s sobrar após todo o `format` ter sido processado, ele é ignorado.

# Exemplos

```
printf('O resultado é:\nalpha=%f',0.535)
```

# Ver Também

string, print, write, format, disp, file, fprintf, sprintf, printf_conversion

# Name

printf_conversion — Especificações de conversão de printf, sprintf, fprintf

# Descrição

Cada especificação de conversão no parâmetro format de `printf` , `sprintf` , `fprintf` possui a seguinte sintaxe:

- Um sinal % (porcento).
- Zero ou mais `options` (opções), que modificam o significado da especificação de conversão. A lista seguinte contem os caracteres `option` e seus significados:
- Alinhe à esquerda, dentro do campo, o resultado da conversão.
- Comece o resultado de uma conversão com sinal com um sinal (+ ou -).
- Prefixe um caractere de espaço ao resultado se o primeiro caractere de uma conversão com sinal não for um sinal. Se ambas as opções (espaço) e + aparecerem, a opção (space) é ignorada.
- Converta o valor para uma forma alternativa. Para as conversões `c`, `d`, `i`, `s`, e `u`, a opção # não possui efeito. Para a conversão `o`, # aumenta a precisão para forçar o primeiro dígito do resultado a ser 0 (zero). Para as conversões `x` e `X`, um valor não-nulo possui 0x ou 0X prefixado a ele. Para as conversões `e, E, f, g,` e `G`, o resultado sempre contém ponto decimal, Mesmo que nenhum dígito o siga. Para as conversões `g` e `G`, zeros por último não são removidos.
- Aumente a largura do campo utilizando zeros à esquerda (seguindo qualquer indicação de sinal ou base) para as conversões `d`, `i`, `o`, `u`, `x`, `X`, `e`, `E`, `f`, `g`, e `G`; nenhum aumento de espaço é realizado. Se ambos os indicadores `0` e `\-` (barra) aparecerem, o indicador `0` é ignorado. Para as conversões `d`, `i`, `o` `u`, `x`, e `X`, se uma precisão for especificada, o indicador `0` também é ignorado.

Um string de dígitos decimais opcional que especifica a largura mínima do campo. Se o valor convertido tiver menos caracteres que a largura do campo, o campo é aumentado à esquerda até o comprimento especificado pela largura do campo. Se a opção de ajuste à esquerda for especificada, o campo é aumentado pela direita.

Uma precisão opcional. A precisão é um `.` (ponto) seguido por um string de dígito decimal. Se nenhuma precisão for fornecida, o parâmetro é tratado como 0 (zero). A precisão especifica:

- O número mínimo de dígitos a aparecerem nas conversões `d`, `u`, `o`, `x`, ou `X`
- O número de dígitos a aparecerem após o ponto decimal nas conversões `e`, `E`, e `f`
- O número máximo de dígitos significativos para as conversões `g` e `G`
- O número máximo de caracteres a serem impressos a partir de um string em uma convesão `s`
- O caractere que indica o tipo de conversão a ser aplicada:
- Não realiza conversão. Exibe %.
- Aceita um valor inteiro e o converte para notação decimal com sinal. A precisão especifica o número mínimo de dígitos a aparecer. Se o valor sendo convertido puder ser representado em menos dígitos, ele é expandido com zeros à esquerda. A precisão padrão é 1. O resultado de se converter um valor zero com uma precisão de zero é um string nulo. A especificação de uma largura de campo com zero como caractere mais à esquerda faz com que o valor da largura do campo seja preenchido com zeros à esquerda.
- Aceita um valor inteiro e o converte para a notação decimal sem sinal. A precisão especifica o número mínimo de dígitos a aparecer. Se o valor sendo convertido puder ser representado em menos

dígitos, ele é expandido com zeros à esquerda. A precisão padrão é 1. O resultado de se converter um valor zero com uma precisão de zero é um string nulo. A especificação de uma largura de campo com zero como caractere mais à esquerda faz com que o valor da largura do campo seja preenchido com zeros à esquerda.

- Aceita um valor inteiro e o converte para a sua notação octal sem sinal. A precisão especifica o número mínimo de dígitos a aparecer. Se o valor sendo convertido puder ser representado em menos dígitos, ele é expandido com zeros à esquerda. A precisão padrão é 1. O resultado de se converter um valor zero com uma precisão de zero é um string nulo. A especificação de uma largura de campo com zero como caractere mais à esquerda faz com que o valor da largura do campo seja preenchido com zeros à esquerda. Não é implicado um valor octal para a largura do campo.

- Aceita um valor inteiro e o converte para a sua notação hexadecimal sem sinal. As letras ``abcdef'' são utilizadas para a conversão `x`; as letras ``ABCDEF'' são utilizadas para a conversão `X`. A precisão especifica o número mínimo de dígitos a aparecer. Se o valor sendo convertido puder ser representado em menos dígitos, ele é expandido com zeros à esquerda. A precisão padrão é 1. O resultado de se converter um valor zero com uma precisão de zero é um string nulo. A especificação de uma largura de campo com zero como caractere mais à esquerda faz com que o valor da largura do campo seja preenchido com zeros à esquerda.

- Aceita um valor float ou double e o converte para a sua notação decimal no formato %[\-]`ddd.ddd`. O número de dígitos após o ponto decimal é igual à especificação de precisão.

- Se nenhuma precisão for especificada, a saída possui seis dígitos

- Se a precisão for zero, nenhum ponto decimal aparece e o sistema imprime na saída o valor inteiro mais próximo de `value`.

- Se a saída possui um ponto decimal, pelo menos um dígito é posto antes dele.

- Aceita um real e o converte para a sua forma exponencial %[\-]`d.ddde`+/\-`dd`. Há um dígito antes do ponto decimal, e o número de dígitos após o ponto decimal é igual à especificação de precisão.

- Se nenhuma precisão for especificada, a saída são seis dígitos

- Se a precisão for zero, nenhum ponto decimal aparece.

- A caractere de conversão `E` produz um número com o caractere 'E', ao invés de 'e' antes do expoente. O expoente sempre contém pelo menos dois dígitos. Se o valor for zero, o expoente é zero.

- Aceita um real e o converte no estilo dos caracteres de conversão `e`, `E`, ou `f`, com a precisão especificando o número de dígitos significativos. Zeros por último são removidos. Um ponto decimal aparece apenas se for seguido de um dígito. O estilo depende do valor convertido. O resultado é o estilo `e` (`E`, se `G` é o indicador utilizado) apenas se o expoente resultante da conversão for menor do que -4, ou se for maior do que ou igual à precisão.

- Aceita e exibe um valor inteiro convertido em um caractere.

- Aceita um valor string e exibe caracteres do string até o fim ou até que o número de caracteres indicados pela precisão seja alcançado. Se nenhuma precisão for especificada, todos os caracteres até o fim são exibidos.

Uma largura de campo ou uma precisão podem ser indicadas por `*` (asterísco) ao invés de um string de dígito. Neste caso, um parâmetro de valor inteiro provê a largura do campo ou a precisão. O parâmetro de valor convertido para saída não é buscado até que a letra de conversão seja alcançada, então os parâmetros especificando o comprimento de campo ou precisão devem aparecer antes do valor a ser convertido (se houver algum).

Se o resultado da conversão for maior que a largura do campo, o campo é expandido para conter o resultado convertido.

A representação do sinal de mais depende da opção de formatação especificada, se + ou (espaço).

## Ver Também

printf, fprintf, sprintf

# Name

sprintf — emulador da função da linguagem C sprintf

```
str=sprintf(format,value_1,..,value_n)
```

# Parâmetros

format
: string Scilab. especifica um string combinando caracteres literais com especificações de conversão.

value_i
: especifica os dados a serem convertidos de acordo com o parâmetro de formato.

str
: vetor coluna de strings

# Descrição

A função `sprintf` converte, formata e escreve seus parâmetros `value` sob controle do parâmetro `format` para a saída padrão.

O parâmetro `format` é um string que contém dois tipos de objetos:

Literal characters
: que são copiados no fluxo de saída

Conversion specifications
: cada um dos quais faz com que zero ou mais itens sejam buscados da lista de parâmetros `value` parameter list. Ver printf_conversion para detalhes.

Se não há itens suficientes para `format` na lista de parâmetros `value`, `sprintf` gera um erro. Se algum valor de `value` sobrar após todo `format` ter sido processado, ele é ignorado.

Nota: `sprintf` é obsoleto, use `msprintf` ao invés.

# Exemplos

```
fahr=120
sprintf('%3d Fahrenheit = %6.1f Celsius',fahr,(5/9)*(fahr-32))
```

# Ver Também

string, print, write, format, disp, file, printf, fprintf, msprintf, printf_conversion

# Name

ssprint — embelezador de sistemas lineares

```
ssprint(sl [,out])
```

# Parâmetros

sl
: lista (lista `syslin`)

out
: saída (valor padrão `out=%io(2)`)

# Descrição

Representação embelezada de uma sistema linear em lista syslin de forma de espaço de estados `sl=(A,B,C,D)`.

# Exemplos

```
a=[1 1;0 1];b=[0 1;1 0];c=[1,1];d=[3,2];
ssprint(syslin('c',a,b,c,d))
ssprint(syslin('d',a,b,c,d))
```

# Ver Também

texprint

# Name

diary — diary of session

```
diary(filename)
[id,filename] = diary(filename, ['new'|'append'])

[ids, filenames] = diary()
[ids, filenames] = diary([], 'list')

diary([], 'close')
diary(0)
diary(filename, 'close')
diary(id, 'close')

diary([], 'pause'|'off')
diary(filename, 'pause'|'off')
diary(id, 'pause'|'off')

diary([], 'resume'|'on')
diary(filename, 'resume'|'on')
diary(id, 'resume'|'on')

diary(filenames, 'exists')
diary(ids, 'exists')

diary(filename, 'new'|'append', 'prefix=YYYY-MM-DD hh:mm:ss')
diary(filename, 'new'|'append', 'prefix=U')
diary(filename, 'new'|'append', [ 'prefix=YYYY-MM-DD hh:mm:ss' ; 'prefix-
```

# Parameters

filename
: a character string, give the full file name path.

id
: a scalar to identify a diary.

# Description

`diary(f)` function creates a log of keyboard input and the resulting text output.

# Start a diary session

`[id, filename] = diary(filename, ['new'|'append'])`

returns :

* id : a positive integer (>= 1) which is the diary session identifier.

* filename : A string, the absolute path of the effective written file.

The first input argument is a string that contain the path of the diary file. This can be a relative path or a absolute path.

The 2nd input argument controls if a new file is created ('new') or if diary() adds new content at the end of the file ('append'). If the 2nd input argument is not present, the default value is 'new'.

When diary() is called with 'new' mode : If 'filename' already exists and is not empty, an effective filename 'base(filemane)+_#+extension(filename)' is built, used, and returned by diary(filename) as a second output argument (beside id). The rank # would be set as the smallest integer for which the resultant filename does not yet exists.

# List diary sessions

```
[ids, filenames] = diary()
```

```
[ids, filenames] = diary([],'list')
```

returns a column vector of integer : identifiers list of opened diary sessions. a column vector of strings : absolute paths of the files associated with opened diary sessions.

# Close diary session(s)

```
diary([],'close')
```

```
diary(0)
```

```
diary(filename,'close')
```

```
diary(id,'close')
```

The first and second syntaxes close all opened diary sessions.

The third syntax closes diary session(s) identified by 'filename'.

The fourth syntax closes the diary session identified by “id” which is a positive integer or a vector of positive integers.

Remark : diary(0) is retained as backwards compatibility.

# Pause/Resume diary session(s)

```
diary([] ,'pause'|'off')
```

```
diary(filename,'pause'|'off')
```

```
diary(id ,'pause'|'off')
```

The first syntax suspends all opened diary sessions.

The second syntax suspend diary session(s) identified by 'filename'. 'filename' can be a single string or a character string array.

The third syntax suspend the diary session identified by 'id' which is a positive integer or a vector of positive integers.

```
diary([] ,'resume'|'on')
```

```
diary(filename,'resume'|'on')
```

```
diary(id ,'resume'|'on')
```

The first syntax resume all opened diary sessions.

The second syntax resume diary session(s) identified by 'filename'. 'filename' can be a single string or a character string array.

The third syntax resume the diary session identified by 'id' which is a positive integer or a vector of positive integers.

# Does a diary session exists ?

`diary(filename,'exists')`

`diary(id,'exists')`

return true if a diary session is currently opened with the file 'filename', if not false.

# Diary and time-stamp

`diary(filename,'new','prefix=YYYY-MM-DD hh:mm:ss')`

`diary(filename,'new','prefix=U')`

`diary(filename,'new',[ 'prefix=YYYY-MM-DD hh:mm:ss' ; 'prefix-only-commands' ] );`

'prefix=YYYY-MM-DD hh:mm:ss' add date & hour

'prefix=U' add UNIX time epoch

'prefix-only-commands' add time-stamp only as prefix for commands

# Filtering diary

`diary(filename,”new”,”filter=command”)` Log only the input commands.

`diary(filename,”new”,”filter=output”)` Log only the text output.

# Examples

```
d1 = diary(TMPDIR + '/diary1.txt')
d2 = diary(TMPDIR + '/diary2.txt')

// some Scilab instructions …
cd TMPDIR
dir

// returns infos about opened diary
[ids, filenames]  = diary()

// close diary d1
diary(d1,'close')
[ids, filenames]  = diary()

// close diary d2
diary(TMPDIR + '/diary2.txt')
[ids, filenames]  = diary()
// closes all diary
diary([],'close')
[ids, filenames]  = diary()
```

# Name

mprintf — converts, formats, and writes data to the main scilab window

```
mprintf(format,a1,...,an);
```

# Parameters

format
: a Scilab string describing the format to use to write the remaining operands. The format operand follows, as close as possible, the C printf format operand syntax.

a1,...,an
: Specifies the data to be converted and printed according to the format parameter.

# Description

The `mprintf` function is a interface for C-coded version of `printf` function.

The `mprintf` function writes formatted operands to the standard Scilab output (i.e the Scilab window). The argument operands are formatted under control of the format operand.

# Examples

```
mprintf('At iteration %i, Result is:\nalpha=%f',33,0.535)
```

# See Also

disp

# Name

msprintf — converts, formats, and writes data in a string

```
str=msprintf(format,a1,...,an);
```

# Parameters

format
: a Scilab string describing the format to use to write the remaining operands.

str
: a character string.

a1,...,an
: Specifies the data to be converted and printed according to the format parameter.

# Description

The `msprintf` writes formatted operands in its returned value (a Scilab string). The argument operands are formatted under control of the format operand.

Note that, in this case, the escape sequences (`"\n"`) split string to a matrix of string (see example)

# Examples

```
msprintf('%5.3f %5.3f',123,0.732)
msprintf('%5.3f\n%5.3f',123,0.732)
msprintf('--%s-\n-%d--',"hello",3)
```

# See Also

mprintf, printf_conversion

# Name

prettyprint — From any Scilab datatype and provide a representation to the TeX, LaTeX or MathML formats

```
str = prettyprint(a) // Show the variable a with the default format (LaTe
str = prettyprint(a,exportFormat) // Show the variable a a with the speci
str = prettyprint(a,exportFormat, delim) // As above but change the delim
str = prettyprint(a,exportFormat, delim, processByElement) // As above bu
str = prettyprint(a,exportFormat, delim, processByElement, isWrapped) // 
```

# Parameters

a:
is a Scilab variable

exportFormat:
is the format, if omitted 'latex' is used by default, it can be 'latex', 'tex' or 'mathml'.

delimiter:
is a string indicating the delimiter to use for the resulting matrix, it's only used if isWrapped is true. The delimiter can be '(', '{', '[', '|', '||' or ')'

processByElement:
is a boolean to indicate if the resulting matrix must be converted into a single string.

isWrapped:
is a boolean to indicate if the result must be wrapped inside delimiters ('$' for latex and tex or nothing for mathml) to be used with xstring or xtitle

str:
the representation of the variable a

# Description

Taking a variable, the prettyprint function will provide a formated representation of it. Formats can be TeX, LaTeX or MathML. They can be used in third party applications but also within Scilab with the most of the Scilab graphic features. The following types are handled by this function:

- Real / Complex matrices
- Polynomial types
- Boolean
- Integer
- String
- Tlist
- Rationnal
- Cell

# Examples

```
str = prettyprint(rand(3,3)) // Return the LaTeX representation of a 3,3 matrix
xstring(0.2,0.2,str) // Show the representation in a graphic Windows

prettyprint(rand(3,4),"mathml") // Return the MathML representation of a 3,4 ma
prettyprint(rand(3,4),"mathml","[") // Return the MathML representation of a 3,

s=poly(0,'s'); G=[1,s;1+s^2,3*s^3];
xstring(0.2,0.2,prettyprint(G*s-1)); // Show a polynom through a LaTeX represen
```

# See also

math_rendering_features_in_graphic, xtitle, axes_properties, label_properties, legend_properties, text_properties, xstringb, xstringl, xstring

# Authors

Calixte Denizet

# Parte XXI. Intersci

# Name

intersci — scilab tool to interface C of Fortran functions with scilab

# Description

All scilab primitive functions are defined in a set of interface routines. For each function the interfacing code checks first number of rhs and lhs arguments. Then it get pointers on input arguments in the Scilab data base and checks their types. After that it calls procedure associated with Scilab functions, checks returned errors flags and set the results in the data base.

`intersci`\ is a program which permits to interface automatically FORTRAN subroutines or C functions to Scilab

With `intersci`, a user can group all his FORTRAN or C code into a same set, called an interface, and use them in Scilab as Scilab functions. The interfacing is made by creating a FORTRAN subroutine which has to be linked to Scilab together with the user code. This complex FORTRAN subroutine is automatically generated by `intersci`\ from a description file of the interface.

Refer to intersci documentation for more details.

# See Also

fort , external , addinter

# Parte XXII. JVM

# Name

javaclasspath — set and get dynamic Java class path

```
res=javaclasspath()
javaclasspath(path)
```

# Parameters

res
: a string matrix

# Description

set and get the dynamic Java path to one or more directory or file specifications given in path.

# Examples

```
res=javaclasspath();
javaclasspath(SCI);
javaclasspath([SCI,SCI+'/java']);
```

# Authors

A.C

# Name

javalibrarypath — set and get dynamic java.library.path

```
res=javalibrarypath()
javalibrarypath(path)
```

# Parameters

res
: a string matrix

# Description

set and get the dynamic Java Library path to one or more directory given in path.

When you use java classes with native methods, you need to define path where is dynamic library.

# Examples

```
res=javalibrarypath();
javalibrarypath(SCI);
javalibrarypath([SCI,SCI+'/libs']);
```

# See Also

javaclasspath

# Authors

A.C

# Name

jre_path — returns Java Runtime Environment used by Scilab

```
p=jre_path()
```

# Parameters

p
: a string path of JRE

# Description

returns Java Runtime Environment used by Scilab.

# See Also

system_getproperty

# Authors

A.C

# Name

system_getproperty — gets the system property indicated by a specified key.

```
res=system_getproperty(key)
```

# Parameters

res
: a string value

key
: a string

# Description

gets the system property indicated by a specified key.

| | |
|---|---|
| java.version | Java Runtime Environment version |
| java.vendor | Java Runtime Environment vendor |
| java.vendor.url | Java vendor URL |
| java.home | Java installation directory |
| java.vm.specification.version | Java Virtual Machine specification version |
| java.vm.specification.vendor | Java Virtual Machine specification vendor |
| java.vm.specification.name | Java Virtual Machine specification name |
| java.vm.version | Java Virtual Machine implementation version |
| java.vm.vendor | Java Virtual Machine implementation vendor |
| java.vm.name | Java Virtual Machine implementation name |
| java.specification.version | Java Runtime Environment specification version |
| java.specification.vendor | Java Runtime Environment specification vendor |
| java.specification.name | Java Runtime Environment specification name |
| java.class.version | Java class format version number |
| java.class.path | Java class path |
| java.library.path | List of paths to search when loading libraries |
| java.io.tmpdir | Default temp file path |
| java.compiler | Name of JIT compiler to use |
| java.ext.dirs | Path of extension directory or directories |
| os.name | Operating system name |
| os.arch | Operating system architecture |
| os.version | Operating system version |
| file.separator | File separator ("/" on UNIX) |
| path.separator | Path separator (":" on UNIX) |
| line.separator Line separator | ("\n" on UNIX) |
| user.name | User's account name |
| user.home | User's home directory |
| user.dir | User's current working directory |

# Examples

```
system_getproperty('awt.toolkit')
system_getproperty('file.encoding')
system_getproperty('file.encoding.pkg')
system_getproperty('java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment')
system_getproperty('java.awt.printerjob=sun.awt.windows.WPrinterJob')
system_getproperty('java.class.path')
system_getproperty('java.class.version')
system_getproperty('java.endorsed.dirs')
system_getproperty('java.ext.dirs')
system_getproperty('java.home')
system_getproperty('java.io.tmpdir')
system_getproperty('java.library.path')
system_getproperty('java.runtime.name')
system_getproperty('java.runtime.version')
system_getproperty('java.specification.name')
system_getproperty('java.specification.vendor')
system_getproperty('java.specification.version')
system_getproperty('java.vendor')
system_getproperty('java.vendor.url')
system_getproperty('java.vendor.url.bug')
system_getproperty('java.version')
system_getproperty('java.vm.info')
system_getproperty('java.vm.name')
system_getproperty('java.vm.specification.name')
system_getproperty('java.vm.specification.vendor')
system_getproperty('java.vm.specification.version')
system_getproperty('java.vm.vendor')
system_getproperty('java.vm.version')
system_getproperty('line.separator')
system_getproperty('os.arch')
system_getproperty('os.name')
system_getproperty('os.version')
system_getproperty('path.separator')
system_getproperty('sun.arch.data.model')
system_getproperty('sun.boot.class.path')
system_getproperty('sun.boot.library.path')
system_getproperty('sun.cpu.endian')
system_getproperty('sun.cpu.isalist')
system_getproperty('sun.desktop')
system_getproperty('sun.io.unicode.encoding')
system_getproperty('sun.jnu.encoding')
system_getproperty('sun.management.compiler')
system_getproperty('sun.os.patch.level')
system_getproperty('user.country')
system_getproperty('user.dir')
system_getproperty('user.home')
system_getproperty('user.language')
system_getproperty('user.name')
system_getproperty('user.timezone')
system_getproperty('user.variant')
```

# Authors

A.C

# Name

system_setproperty — set a system property indicated by a specified key and value.

```
prev = system_setproperty(key,value)
```

# Parameters

prev
: a string previous value or []

key
: a string

value
: a string

# Description

Sets the system property indicated by the specified key.

Warning : change property with precaution.

# Examples

```
system_getproperty('myproperty')
system_setproperty('myproperty','hello')
system_getproperty('myproperty')
```

# Authors

A.C

# Name

with_embedded_jre — checks if scilab uses a embedded JRE

```
res=with_embedded_jre()
```

# Parameters

res
: a boolean

# Description

checks if scilab uses a embedded JRE.

# Examples

```
res=with_embedded_jre();
```

# Authors

A.C

# Parte XXIII. Álgebra Linear

# Name

aff2ab — Conversão de uma função linear (afim) para forma A,b

```
[A,b]=aff2ab(afunction,dimX,D [,flag])
```

# Parâmetros

afunction
: uma função do Scilab `Y =fct(X,D)` onde `X, D, Y` são `lists` de matrizes

dimX
: uma matriz de inteiros p x 2 (`p` é o número de matrizes em X)

D
: uma `list` de matrizes de reais (ou qualquer outro objeto Scilab válido).

flag
: parâmetro opcional (`flag='f'` ou `flag='sp'`)

A
: uma matriz de reais

b
: um vetor de reais tendo a mesma dimensão de linha que `A`

# Descrição

`aff2ab` retorna a representação matricial de uma função afim (na base canônica).

`afunction` é uma função com sintaxe imposta: `Y=afunction(X,D)` onde `X=list(X1,X2,...,Xp)` é uma lista de p matrizes de reais, e `Y=list(Y1,...,Yq)` é uma lista de q matrizes reais que dependem linearmente das `Xi`'s. A entrada (opcional) `D` contém parâmetros necessários para computar Y como uma função de X (geralmente é uma lista de matrizes).

`dimX` é uma matriz p x 2: `dimX(i)=[nri,nci]` é o número real de linhas e colunas da matriz `Xi`. Estas dimensões determinam `na`, a dimensão de coluna da matriz resultante `A`: `na=nr1*nc1 +...+ nrp*ncp`.

Se o parâmetro opcional `flag='sp'` a matriz resultante A é retornada como uma esparsa.

Esta função é útil para resolver um sistema de equações lineares onde as incógnitas são matrizes.

# Exemplos

```
// solucionador de equação de Lyapunov (uma incógnita, uma restrição)
deff('Y=lyapunov(X,D)','[A,Q]=D(:);Xm=X(:); Y=list(A''*Xm+Xm*A-Q)')
A=rand(3,3);Q=rand(3,3);Q=Q+Q';D=list(A,Q);dimX=[3,3];
[Aly,bly]=aff2ab(lyapunov,dimX,D);
[Xl,kerA]=linsolve(Aly,bly); Xv=vec2list(Xl,dimX); lyapunov(Xv,D)
Xm=Xv(:); A'*Xm+Xm*A-Q

// solucionador de equação de Lyapunov com restrição redundante X=X'
// (uma variável, uma restrição) D é variável global
deff('Y=ly2(X,D)','[A,Q]=D(:);Xm=X(:); Y=list(A''*Xm+Xm*A-Q,Xm''-Xm)')
A=rand(3,3);Q=rand(3,3);Q=Q+Q';D=list(A,Q);dimX=[3,3];
[Aly,bly]=aff2ab(ly2,dimX,D);
```

```
[Xl,kerA]=linsolve(Aly,bly); Xv=vec2list(Xl,dimX); ly2(Xv,D)

// equações de Francis
// Achando matrizes X1 e X2 tais que:
// A1*X1 - X1*A2 + B*X2 -A3 = 0
// D1*X1 -D2 = 0
deff('Y=bruce(X,D)','[A1,A2,A3,B,D1,D2]=D(:),...
[X1,X2]=X(:);Y=list(A1*X1-X1*A2+B*X2-A3,D1*X1-D2)')
A1=[-4,10;-1,2];A3=[1;2];B=[0;1];A2=1;D1=[0,1];D2=1;
D=list(A1,A2,A3,B,D1,D2);
[n1,m1]=size(A1);[n2,m2]=size(A2);[n3,m3]=size(B);
dimX=[[m1,n2];[m3,m2]];
[Af,bf]=aff2ab(bruce,dimX,D);
[Xf,KerAf]=linsolve(Af,bf);Xsol=vec2list(Xf,dimX)
bruce(Xsol,D)

// Achando todas as X que comutam com A
deff('y=f(X,D)','y=list(D(:)*X(:)-X(:)*D(:))')
A=rand(3,3);dimX=[3,3];[Af,bf]=aff2ab(f,dimX,list(A));
[Xf,KerAf]=linsolve(Af,bf);[p,q]=size(KerAf);
Xsol=vec2list(Xf+KerAf*rand(q,1),dimX);
C=Xsol(:); A*C-C*A
```

# Ver Também

linsolve

# Name

balanc — balanceamento de uma matriz ou de um feixe de matrizes

```
[Ab,X]=balanc(A)
[Eb,Ab,X,Y]=balanc(E,A)
```

# Parâmetros

A:
: matriz quadrada de reais

X:
: matriz quadrada de reais invertível

E:
: matriz quadrada de reais (com mesma dimensão que `A`)

Y:
: uma matriz quadrada de reais invertível

# Descrição

Balanceia uma matriz quadrada para melhorar seu número de condicionamento.

`[Ab,X] = balanc(A)` acha uma transformação de similaridade `X` tal que

`Ab = inv(X)*A*X` tem aproximadamente normas iguais de linha e de coluna.

Para feixes de matrizes, o balancemento é feito para melhorar o problema do autovalor generalizado.

`[Eb,Ab,X,Y] = balanc(E,A)` retorna transformações esquerda e direita `X` e `Y` tais que `Eb=inv(X)*E*Y, Ab=inv(X)*A*Y`

# Observação

O balanceamento é feito nas funções `bdiag` e `spec`.

# Exemplos

```
A=[1/2^10,1/2^10;2^10,2^10];
[Ab,X]=balanc(A);
norm(A(1,:))/norm(A(2,:))
norm(Ab(1,:))/norm(Ab(2,:))
```

# Ver Também

bdiag, spec, schur

# Name

bdiag — diagonalização em blocos, autovetores generalizados

```
[Ab [,X [,bs]]]=bdiag(A [,rmax])
```

# Parâmetros

A
: matriz quadrada de reais ou complexos

rmax
: número real

Ab
: matriz quadrada de reais ou complexos

X
: matriz de reais ou complexos não-singular

bs
: vetor de inteiros

# Descrição

```
[Ab [,X [,bs]]]=bdiag(A [,rmax])
```

realiza a diagonalização em blocos da matriz `A`. bs fornece a estrutura dos blocos (respectivos tamanhos dos blocos). `X` é a matriz mudança de base i.e `Ab = inv(X)*A*X` é diagonal em blocos.

`rmax` controla o condicionamento de `X`; o valor padrão é a norma L1 de `A`.

Para encontrar a forma diagonal (se existir) escolha um valor suficientemente grande para `rmax` (`rmax=1/%eps` , por exemplo). Genericamente, (para uma matriz A de reais aleatória) os blocos são (1x1) e (2x2) e `X` é a matriz de autovetores.

# Exemplos

```
//No caso de reais: blocos 1x1 e 2x2
a=rand(5,5);[ab,x,bs]=bdiag(a);ab

//No caso de complexos: blocos 1x1 complexos
[ab,x,bs]=bdiag(a+%i*0);ab
```

# Ver Também

schur, sylv, spec

# Name

chfact — fatoração esparsa de Cholesky

```
spcho=chfact(A)
```

# Parâmetros

A
: uma matriz simétrica, positiva e esparsa

spcho
: lista contendo os fatores de Cholesky em forma codificada

# Descrição

`spcho=chfact(A)` computa os fatores esparsos de Cholesky da matriz esparsa `A`, assumida simétrica e positiva definida. A função é baseada nos programas Ng-Peyton (ORNL). Ver os programas FORTRAN para uma completa descrição das variáveis em `spcho`. Esta função deve ser usada com a função `chsolve`.

# Ver Também

chsolve, sparse, lufact, luget, spchol

# Name

chol — Cholesky factorization

```
[R]=chol(X)
```

# Parâmetros

X
: uma matriz simétrica e positiva definida de reais ou complexos.

# Descrição

Se `X` é positiva definida, então `R = chol(X)` produz uma matriz triangular superior `R` tal que `R'*R = X`.

`chol(X)` usa apenas a diagonal e o triângulo superior de `X`. O triângulo inferior é assumido como sendo a transposta (ou complexo conjugado) da superior.

# Referências

A decomposição de Cholesky é baseada nas rotinas de Lapack DPOTRF para matrizes de reais e ZPOTRF no caso de matrizes de complexos.

# Exemplos

```
W=rand(5,5)+%i*rand(5,5);
X=W*W';
R=chol(X);
norm(R'*R-X)
```

# Ver Também

spchol, qr, svd, bdiag, fullrf

# Name

chsolve — solucionador esparso de Cholesky

```
sol=chsolve(spcho,rhs)
```

# Parâmetros

spcho
: lista contendo os fatores de Cholesky na forma codificada retornados por chfact

rhs, sol
: vetores colunas cheios

# Descrição

`sol=chsolve(spcho,rhs)` computa a solução de `rhs=A*sol`, com `A` uma matriz simétrica e positiva definida. Esta função é baseada nos programas Ng-Peyton (ORNL). Veja os programas FORTRAN para uma descrição completa das variáveis em `spcho`.

# Exemplos

```
A=sprand(20,20,0.1);
A=A*A'+eye();
spcho=chfact(A);
sol=(1:20)';rhs=A*sol;
spcho=chfact(A);
chsolve(spcho,rhs)
```

# Ver Também

chfact, sparse, lufact, luget, spchol

# Name

classmarkov — classes transientes e recorrentes da matriz de Markov

```
[perm,rec,tr,indsRec,indsT]=classmarkov(M)
```

# Parâmetros

M
: matriz de Markov N x N de reais. A soma das entradas em cada linha deve ser acrescida em uma unidade

perm
: vetor de permutação de inteiros

rec, tr
: vetor de inteiros, número (número de estados em cada classe recorrente, número de estados transientes)

indsRec,indsT
: vetor de inteiros (índices dos estados recorrentes e transientes)

# Descrição

Retorna um vetor de permutação `perm` tal que

```
M(perm,perm) = [M11 0 0 0 0   0]
               [0 M22 0 0     0]
               [0 0 M33       0]
               [      ...      ]
               [0 0       Mrr 0]
               [* *         *  Q]
```

Cada `Mii` é uma matriz de Markov de dimensão `rec(i) i=1,..,r`. `Q` é uma submatriz de Markov de dimensão `tr`. Estados de 1 a sum(rec) são recorrentes e estados de r+1 a n são transientes. Tem-se `perm=[indsRec,indsT]` onde indsRec é um vetor de tamanho sum(rec) e indsT é um vetor de tamanho tr.

# Exemplos

```
//P tem 2 classes recorrentes (com 2 e 1 estados) e 2 estados transientes
P=genmarkov([2,1],2,'perm')
[perm,rec,tr,indsRec,indsT]=classmarkov(P);
P(perm,perm)
```

# Ver Também

genmarkov, eigenmarkov

# Name

cmb_lin — combinação linear simbólica

```
[x]=cmb_lin(alfa,x,beta,y)
```

# Descrição

Avalia `alfa*x-beta*y`. `alfa, beta, x, y` são strings. (rotina de baixo-nível).

# Ver também

mulf, addf

# Name

coff — resolvente (método do cofator)

```
[N,d]=coff(M [,var])
```

# Parâmetros

M
: matriz quadrada de reais

var
: string

N
: matriz de polinômios (com o mesmo tamanho que `M`)

d
: polinômio ( polinômio característico `poly(A,'s')`)

# Descrição

`coff` computa R=`(s*eye()-M)^-1` para `M` uma matriz de reais. R é dado por `N/d`.

`N` = matriz de polinômios como numerador.

`d` = denominador comum.

`var` string ( ('`s`' se omitido)

# Exemplos

```
M=[1,2;0,3];
[N,d]=coff(M)
N/d
inv(%s*eye()-M)
```

# Ver Também

coffg, ss2tf, nlev, poly

# Name

colcomp — compressão de colunas, núcleo

```
[W,rk]=colcomp(A [,flag] [,tol])
```

# Parâmetros

A
: matriz de reais ou complexos

flag
: string

tol
: número real

W
: matriz quadrada não-singular (mudança de base)

rk
: inteiro (posto de A)

# Descrição

Compressão de colunas de A: `Ac = A*W` é de colunas comprimidas ,i.e.,

`Ac=[0,Af]` com o posto-coluna de `Af` cheio, posto(`Af`) = posto(A) = `rk`.

`flag` e `tol` são parâmetros opcionais: `flag = 'qr'` ou `'svd'` (o padrão é `'svd'`).

`tol` = parâmetro de tolerância (de ordem `%eps` como valor padrão).

As `ma-rk` primeiras colunas de `W` geram o núcleo de `A` quando `size(A)=(na,ma)`

# Exemplos

```
A=rand(5,2)*rand(2,5);
[X,r]=colcomp(A);
norm(A*X(:,1:$-r),1)
```

# Ver Também

rowcomp, fullrf, fullrfk, kernel

# Autor

F.D.;

# Name

companion — matriz companheira

```
A=companion(p)
```

# Parâmetros

p
: polinômio ou vetor de polinômios

A
: matriz quadrada

# Descrição

Retorna uma matriz quadrada `A` com o polinômio característico igual a `p` se `p` é mônico. Se `p` não é mônico, o polinômio característico de `A` é igual a `p/c` onde `c` é o coeficiente do termo de maior grau em `p`.

Se `p` é um vetor de polinômios mônicos, `A` é diagonal em blocos, e o polinômio característico do i-ésimo bloco é `p(i)`.

# Exemplos

```
s=poly(0,'s');
p=poly([1,2,3,4,1],'s','c')
det(s*eye()-companion(p))
roots(p)
spec(companion(p))
```

# Ver Também

spec, poly, randpencil

# Autor

F.D.

# Name

cond — número de condicionamento de uma matriz

```
cond(X)
```

# Parâmetros

X
: rmatriz quadrada de reais ou complexos

# Descrição

Número de condicionamento em norma-2. `cond(X)` é a razão entre o maior e o menor valor singular de `X`.

# Exemplos

```
A=testmatrix('hilb',6);
cond(A)
```

# Ver Também

rcond, svd

# Name

det — determinante

```
det(X)
[e,m]=det(X)
```

# Parâmetros

X
: matriz quadrada de reais ou complexos, matriz de polinômios ou de razões de polinômios

m
: número real ou complexo, a mantissa de base 10 do determinante

e
: inteiro, o expoente de base 10 do determinante

# Descrição

`det(X)` ( `m*10^e` é o determinante da matriz quadrada `X)`.

Para uma matriz de polinômios, `det(X)` é equivalente a `determ(X)`.

Para matrizes de razões de polinômios `det(X)` é equivalente a `detr(X)`.

# Referências

As computações da função det são baseadas nas rotinas do LAPACK DGETRF para matrizes de reais e ZGETRF para o caso de matrizes de complexos.

# Exemplos

```
x=poly(0,'x');
det([x,1+x;2-x,x^2])
w=ssrand(2,2,4);roots(det(systmat(w))),trzeros(w)   //zeros do sistema linear
A=rand(3,3);
det(A), prod(spec(A))
```

# Ver Também

detr, determ

# Name

eigenmarkov — Autovetores esquerdo e direito normalizados de Markov

```
[M,Q]=eigenmarkov(P)
```

# Parâmetros

P
: matriz de Markov N x N de reais. A soma das entradas de cada linha deve ser acrescida de uma unidade

M
: matriz de reais de N colunas

Q
: matriz de reais de N linhas

# Descrição

Retorna os autovetores esquerdo e direito normalizados associados ao autovalor 1 da matriz P de transição de Markov. Se a multiplicidade deste autovalor é m e P é N x N, M é uma matriz m x N e Q é uma matriz N x m. M(k,:) é o vetor de distribuição de probabilidade associado ao k-ésimo conjunto ergódico (classe recorrente). M(k,x) é zero se x não está na k-ésima classe recorrente. Q(x,k) é a probabilidade de se terminar na k-ésima classe recorrente começando de x. Se `P^k` converge para `k` (sem autovalores no círculo unitário, exceto 1), então o limite é `Q*M` (auto-projeção).

# Exemplos

```
//P tem duas classes recorrentes (com 2 e 1 estados) e 2 estados transientes
P=genmarkov([2,1],2)
[M,Q]=eigenmarkov(P);
P*Q-Q
Q*M-P^20
```

# Ver Também

genmarkov, classmarkov

# Name

ereduc — computa de forma escada de colunas da matriz por transformações qz

```
[E,Q,Z [,stair [,rk]]]=ereduc(X,tol)
```

# Parâmetros

X
: matriz m x n de entradas reais

tol
: escalar real positivo

E
: matriz em forma escada de colunas

Q
: matriz unitária m x m

Z
: matriz unitária n x n

stair
: vetor de índices,

  * `ISTAIR(i) = + j` se o elemento da borda `E(i,j)` é uma quina.
  * `ISTAIR(i) = - j` se o elemento da borda `E(i,j)` não é uma quina.

  `(i=1,...,M)`

rk
: inteiro, posto estimado da matriz

# Descrição

Dada uma matriz X `m x n` (não necessariamente regular), a função ereduc calcula a matriz unitária transformada `E=Q*X*Z` que está na forma escada de colunas (forma trapezoidal). Ainda, o posto da matriz `X` é determinado.

# Exemplos

```
X=[1 2 3;4 5 6]
[E,Q,Z ,stair ,rk]=ereduc(X,1.d-15)
```

# Ver Também

fstair

# Autores

Th.G.J. Beelen (Philips Glass Eindhoven). SLICOT

# Name

expm — exponencial de matriz quadrada (matriz exponencial)

```
expm(X)
```

# Parâmetros

X
: square matrix with real or complex entries.

# Descrição

`X` é uma matriz quadrada e `expm(X)` é a matriz

```
expm(X) = I + X + X^2 /2 + ...
```

A computação é realizada primeiro diagonalizando em blocos a matriz `X` e, em seguida, aplicando uma aproximação de Pade em cada bloco.

# Exemplos

```
X=[1 2;3 4]
expm(X)
logm(expm(X))
```

# Ver Também

logm, bdiag, coff, log, exp

# Name

fstair — computa a forma escada de feixe de colunas por transformações qz

```
[AE,EE,QE,ZE,blcks,muk,nuk,muk0,nuk0,mnei]=fstair(A,E,Q,Z,stair,rk,tol)
```

# Parâmetros

A
: matriz m x n com entradas reais

tol
: escalar real positivo

E
: matriz de forma escada de colunas

Q
: matriz unitária m x m

Z
: matriz unitária n x n

stair
: vetor de índices (ver ereduc)

rk
: inteiro, posto estimado da matriz

AE
: matriz m x n com entradas reais

EE
: matriz de forma escada de colunas

QE
: matriz unitária m x m

ZE
: matriz unitária n x n

nblcks
: é o número de submatrizes com posto linha completo >= 0 detectado na matriz A

muk:
: array (vetor ou matriz) de inteiros de dimensão (n). Contém as dimensões de coluna mu(k) (k=1,...,nblcks) das submatrizes com posto coluna cheio no feixe sE(eps)-A(eps)

nuk:
: array de inteiros de dimensão (m+1). Contém as dimensões de linha nu(k) (k=1,...,nblcks) das submatrizes com posto linha cheio no feixe sE(eps)-A(eps)

muk0:
: array de inteiros de dimensão (n). Contém as dimensões de coluna mu(k) (k=1,...,nblcks) das submatrizes com o posto-coluna cheio no feixe sE(eps,inf)-A(eps,inf)

nuk:
: array de inteiros de dimensão (m+1). Contém as dimensões de linha nu(k) (k=1,...,nblcks) das submatrizes com posto-linha cheio no feixe sE(eps,inf)-A(eps,inf)

mnei:
array de inteiros dimensão (4). mnei(1) = dimensão de linha de sE(eps)-A(eps)

# Descrição

Dado o feixe `sE-A` onde a matriz `E` está na forma escada de colunas, a função `fstair` computa, de acordo com as necessidades do usuário, um feixe unitário transformado `QE(sEE-AE)ZE` que é mais ou menos similar à forma generalizada de Schur do feixe `sE-A`. A função também produz parte da estrutura de Kronecker para um dado feixe.

`Q,Z` são as matrizes unitárias usadas para computar o feixe onde E está na forma escada de colunas (ver ereduc)

# Ver Também

quaskro, ereduc

# Autor

Th.G.J. Beelen (Philips Glass Eindhoven). SLICOT

# Name

fullrf — fatoração de posto completo (ou cheio)

```
[Q,M,rk]=fullrf(A,[tol])
```

# Parâmetros

A
: matriz de reais ou complexos

tol
: número real (limiar para determinação do posto)

Q,M
: matriz de reais ou complexos

rk
: inteiro(posto de A)

# Descrição

Fatoração de posto cheio : `fullrf` retorna `Q` e `M` tais que `A = Q*M` com Im(Q)=Im(A) e Nuc(M)=Nuc(A), `Q` de posto-coluna cheio, `M` de posto-linha cheio e `rk = rank(A) = #columns(Q) = #rows(M)`.

`tol` é um parâmetro real opcional (valor real padrão é `sqrt(%eps)`). O posto `rk` de `A` é definido como o número de valores singulares maiores que `norm(A)*tol`.

Se A é simétrica, `fullrf` retorna `M=Q'`.

# Exemplos

```
A=rand(5,2)*rand(2,5);
[Q,M]=fullrf(A);
norm(Q*M-A,1)
[X,d]=rowcomp(A);Y=X';
svd([A,Y(:,1:d),Q])        //span(Q) = span(A) = span(Y(:,1:2)) (span = "gerado
```

# Ver Também

svd, qr, fullrfk, rowcomp, colcomp

# Autor

F.D.;

# Name

fullrfk — fatoração de posto completo de A^k

```
[Bk,Ck]=fullrfk(A,k)
```

# Parâmetros

A
: matriz de reais ou de complexos

k
: inteiro

Bk,Ck
: matrizes de reais ou de complexos

# Descrição

Esta função calcula a fatoração de posto completo (ou cheio) de `A^k` i.e. `Bk*Ck=A^k` onde `Bk` é de posto-coluna cheio e `Ck` de posto-linha cheio. Tem-se Im(`Bk`)=Im(`A^k`) e Nuc(`Ck`)=Nuc(`A^k`).

Para `k=1`, `fullrfk` é equivalente a `fullrf`.

# Exemplos

```
A=rand(5,2)*rand(2,5);[Bk,Ck]=fullrfk(A,3);
norm(Bk*Ck-A^3,1)
```

# Ver Também

fullrf, range

# Autor

F.D (1990);

# Name

genmarkov — gera uma matriz de Markov aleatória com classes recorrentes e transientes

```
M=genmarkov(rec,tr)
M=genmarkov(rec,tr,flag)
```

# Parâmetros

rec
: vetor linha de inteiros (sua dimensão é o número de classes recorrentes)

tr
: inteiro (número de estados transientes)

M
: matriz de Markov de reais. A soma das entradas de cada linha deve ser acrecsida de uma unidade

flag
: string `'perm'`. Se fornecido, uma permutação dos estados é feita.

# Descrição

Retorna em M uma matriz de probabilidade de transição de Markov aleatória com `rec(1),...rec($)` entradas respectivamente e tr estados transientes.

# Exemplos

```
//P tem duas classes recorrentes (com 2 e 1 states) e 2 estados transientes
P=genmarkov([2,1],2,'perm')
[perm,rec,tr,indsRec,indsT]=classmarkov(P);
P(perm,perm)
```

# Ver Também

classmarkov, eigenmarkov

# Name

givens — transformação de Givens

```
U=givens(xy)
U=givens(x,y)
[U,c]=givens(xy)
[U,c]=givens(x,y)
```

# Parâmetros

x,y
: dois números reais ou complexos

xy
: vetor coluna de reais ou complexos de tamanho 2

U
: matriz 2x2 unitária

c
: vetor coluna de reais ou complexos de tamanho 2

# Descrição

`U = givens(x, y)` ou `U = givens(xy)` com `xy = [x;y]` retorna uma matriz unitária `2x2` `U` tal que:

`U*xy=[r;0]=c`.

Note que `givens(x,y)` e `givens([x;y])` são equivalentes.

# Exemplos

```
A=[3,4;5,6];
U=givens(A(:,1));
U*A
```

# Ver Também

qr

# Name

glever — inverso do feixe de matrizes

```
[Bfs,Bis,chis]=glever(E,A [,s])
```

# Parâmetros

E, A
: duas matrizes de reais quadradas de igual dimensão

s
: string (o valor padrão é 's')

Bfs,Bis
: duas matrizes de polinômios

chis
: polinômio

# Descrição

Computação de

`(s*E-A)^-1`

pelo algoritmo generalizado de Leverrier para um feixe de matrizes.

```
(s*E-A)^-1 = (Bfs/chis) - Bis.
```

`chis` = polinômio característico (até uma constante multiplicativa).

`Bfs` = nmatriz de polinômios como numerador.

`Bis` = matriz de polinômios ( - expansão de `(s*E-A)^-1` ao infinito).

Note o sinal - antes de `Bis`.

# Cuidado

Esta função usa `cleanp` para simplificar `Bfs,Bis` e `chis`.

# Exemplos

```
s=%s;F=[-1,s,0,0;0,-1,0,0;0,0,s-2,0;0,0,0,s-1];
[Bfs,Bis,chis]=glever(F)
inv(F)-((Bfs/chis) - Bis)
```

# Ver Também

rowshuff, det, invr, coffg, pencan, penlaur

# Autor

F. D. (1988)

# Name

gschur — forma de Schur generalizada (função obsoleta)

```
[As,Es]=gschur(A,E)
[As,Es,Q,Z]=gschur(A,E)
[As,Es,Z,dim] = gschur(A,E,flag)
[As,Es,Z,dim]= gschur(A,E,extern)
```

# Descrição

Esta função é obsoleta e agora está inclusa na função `schur` function. Na maior parte dos casos, a função `gschur` irá ainda trabalhar como antes, mas será removida em um lançamento futuro do Scilab.

As três sintaxes podem ser substituídas por

```
[As,Es]=schur(A,E)
[As,Es,Q,Z]=schur(A,E);Q=Q' //NOTE A TRANSPOSIÇÃO AQUI
[As,Es,Z,dim] = schur(A,E,flag)
```

A última sintaxe requer algumas adaptações a mais:

if
: é uma função do Scilab, a nova seqüência de chamamento deve ser `[As,Es,Z,dim]= schur(A,E,Nextern)` com Nextern definido como segue:

```
function t=Nextern(R)
if R(2)==0 then
  t=extern([1,R(1),R(3)])==1
else
  c=(R(1)+%i*R(2))/R(3)
  t=extern([2,real(c+c'),real(c*c')])==1
end
endfunction
```

if
: é o nome de uma função externa codificada em FORTRAN ou C, a nova seqüência de chamamento deve ser `[As,Es,Z,dim]= schur(A,E,'nextern')` com nextern definido como segue:

```
logical function nextern(ar,ai,beta)
double precision ar,ai,beta
integer r,extern
if (ai.eq.0.0d0) then
  r=extern(1,ar,beta,0.0d0,0.0d0)
else
  r=extern(2,0.0d0,0.0d0,2.0d0*ar,ar*ar+ai*ai)
endif
nextern=r.eq.1
end
```

# Ver Também

external, schur

# Name

gspec — autovalores de feixe de matrizes (função obsoleta)

```
[al,be]=gspec(A,E)
[al,be,Z]=gspec(A,E)
```

# Descrição

Esta função está agora inclusa na função `spec` . A seqüência de chamamento deve ser substituida por

```
[al,be]=spec(A,E)
[al,be,Z]=spec(A,E)
```

# Ver Também

spec

# Name

hess — forma de Hessenberg

```
H = hess(A)
[U,H] = hess(A)
```

# Parâmetros

A
: matriz quadrada de reais ou complexos

H
: matriz quadrada de reais ou complexos

U
: matriz quadrada ortogonal ou unitária

# Descrição

`[U,H] = hess(A)` produz uma matriz unitária `U` e uma matriz de Hessenberg `H` tais que `A = U*H*U'` e `U'*U` = Identidade. Por si só, `hess(A)` retorna `H`.

A forma de Hessenberg de uma matriz é zero abaixo da primeira subdiagonal. Se a matriz é simetrica ou Hermitiana, a forma é tridiagonal.

# Referências

A função hess é baseada nas rotinas Lapack DGEHRD, DORGHR para matrizes de e ZGEHRD, ZORGHR para matrizes de complexos.

# Exemplos

```
A=rand(3,3);[U,H]=hess(A);
and( abs(U*H*U'-A)<1.d-10 )
```

# Ver Também

qr, contr, schur

# Name

householder — matriz de reflexão ortogonal de Householder

```
u=householder(v [,w])
```

# Parâmetros

v
: vetor coluna de reais ou complexos

w
: vetor coluna de reais ou complexos com o mesmo tamanho que `v`. Valor padrão é `eye(v)`

u
: vetor coluna de reais ou complexos

# Descrição

Dados dois vetores coluna `v`, `w` de mesmo tamanho, `householder(v,w)` retorna um vetor coluna unitário `u`, tal que `(eye()-2*u*u')*v` é proporcional a `w`. `(eye()-2*u*u')` é a matriz de reflexão ortogonal de Householder .

O valor padrão de `w` é `eye(v)`. Neste caso, o vetor `(eye()-2*u*u')*v` é o vetor `eye(v)*norm(v)`.

# Ver Também

qr, givens

# Name

im_inv — imagem inversa

```
[X,dim]=im_inv(A,B [,tol])
[X,dim,Y]=im_inv(A,B, [,tol])
```

# Parâmetros

A,B
: duas matirzes de reais ou complexos com igual número de colunas

X
: matriz quadrada ortogonal ou unitária de ordem igual ao número de colunas de `A`

dim
: inteiro (dimensão do subespaço)

Y
: matriz ortogonal de ordem igual ao número de linhas de `A` e `B`.

# Descrição

`[X,dim]=im_inv(A,B)` computa`(A^-1)(B)` ,i.e, vetores cujas imagens através de `A` estão em Im(`B`)

As `dim` primeiras colunas de `X` geram `(A^-1)(B)`

`tol` é um limiar usado para testar a inclusão de subespaço ; o valor padrão é `tol = 100*%eps`. Se `Y` é retornado, então `[Y*A*X,Y*B]` é particionado como segue: `[A11,A12;0,A22]`,`[B1;0]`

onde `B1` tem posto-linha cheio (igual a posto`(B)`) e `A22` tem posto-coluna cheio e tem `dim` colunas.

# Exemplos

```
A=[rand(2,5);[zeros(3,4),rand(3,1)]];B=[[1,1;1,1];zeros(3,2)];
W=rand(5,5);A=W*A;B=W*B;
[X,dim]=im_inv(A,B)
svd([A*X(:,1:dim),B])   //vetores A*X(:,1:dim) pertencem a range(B) (Imagem de
[X,dim,Y]=im_inv(A,B);[Y*A*X,Y*B]
```

# Ver Também

rowcomp, spaninter, spanplus, linsolve

# Autor

F. Delebecque INRIA

# Name

inv — inversa de uma matriz

```
inv(X)
```

# Parâmetros

X
: matriz quadrada de reais ou complexos, matriz de polinômios, matriz de razões de polinômios em representação de transferência ou espaço de estados

# Descrição

`inv(X)` é a inversa da matriz quadrada `X`. Uma aviso é impresso na tela se `X` possui má escala ou é quase singular.

Para matrizes de polinômios ou matrizes razões de polinômios em representação de transferência, `inv(X)` é equivalente a `invr(X)`.

Para sistemas lineares na representação de espaço de estados (lista `syslin`), `invr(X)` é equivalente a `invsyslin(X)`.

# Referências

A função inv para matrizes de números é baseada nas rotinas de Lapack DGETRF, DGETRI para matrizes de reais e ZGETRF, ZGETRI para o caso de matrizes de complexos. Para matrizes de polinômios e matrizes de funções racionais, `inv` é baseado na função `invr` do Scilab.

# Exemplos

```
A=rand(3,3);inv(A)*A

x=poly(0,'x');
A=[x,1,x;x^2,2,1+x;1,2,3];inv(A)*A

A=[1/x,2;2+x,2/(1+x)]
inv(A)*A

A=ssrand(2,2,3);
W=inv(A)*A
clean(ss2tf(W))
```

# Ver Também

slash, backslash, pinv, qr, lufact, lusolve, invr, coff, coffg

# Name

kernel — núcleo de uma matriz

```
W=kernel(A [,tol,[,flag])
```

# Parâmetros

A
: matriz de reais ou complexos completa ou matriz de reais esparsa

flag
: string `'svd'` (padrão) ou `'qr'`

tol
: número real

W
: matriz de posto-coluna completo

# Descrição

`W=kernel(A)` retorna o núcleo (espaço nulo) de `A`. Se A tem posto-coluna completo, então uma matriz vazia [] é retornada.

`flag` e `tol` são parâmetros opcionais: `flag = 'qr'` ou `'svd'` (o padrão é `'svd'`).

`tol` = parâmetro de tolerância (de ordem `%eps` como valor padrão).

# Exemplos

```
A=rand(3,1)*rand(1,3);
A*kernel(A)
A=sparse(A);
clean(A*kernel(A))
```

# Ver Também

colcomp, fullrf, fullrfk, linsolve

# Autor

F.D.;

# Name

kroneck — forma de Kronecker de feixe de matrizes

```
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(F)
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(E,A)
```

# Parâmetros

F
: feixe de matrizes de reais `F=s*E-A`

E,A
: duas matrizes de reais de mesma dimensão

Q,Z
: duas matrizes quadradas ortogonais

Qd,Zd
: dois vetores de inteiros

numbeps,numeta
: dois vetores de inteiros

# Descrição

Forma de Kronecker de feixe de matrizes: `kroneck` computa duas matrizes ortogonais `Q, Z` que põem o feixe `F=s*E -A` na forma triangular superior:

```
           | sE(eps)-A(eps) |        X       |      X     |      X        |
           |----------------|----------------|------------|---------------|
           |        0       | sE(inf)-A(inf) |      X     |      X        |
Q(sE-A)Z = |--------------------------------|----------------------------|
           |                |                |            |               |
           |        0       |       0        | sE(f)-A(f) |      X        |
           |-------------------------------------------------------------|
           |                |                |            |               |
           |        0       |       0        |      0     | sE(eta)-A(eta)|
```

As dimensões dos quatro blocos são dadas por:

`eps=Qd(1) x Zd(1)`, `inf=Qd(2) x Zd(2)`, `f = Qd(3) x Zd(3)`, `eta=Qd(4)xZd(4)`

O bloco `inf` contém modos infinitos de feixes.

O bloco `f` contém modos finitos de feixes.

A estrutura dos blocos epsilon e eta é dada por

`numbeps(1)` = # de blocos eps de tamanho 0 x 1

`numbeps(2)` = # de blocos eps de tamanho 1 x 2

`numbeps(3)` = # de blocos eps de tamanho 2 x 3 etc...

`numbeta(1)` = # de blocos eta de tamanho 1 x 0

`numbeta(2)` = # de blocos eta de tamanho 2 x 1

`numbeta(3)` = # de blocos eta de tamanho 3 x 2 etc...

O código foi retirado de T. Beelen (Slicot-WGS group).

# Exemplos

```
F=randpencil([1,1,2],[2,3],[-1,3,1],[0,3]);
Q=rand(17,17);Z=rand(18,18);F=Q*F*Z;
//feixe aleatório com eps1=1,eps2=1,eps3=1; 2 blocos J @ infty (infinito)
//com dimensões 2 e
//3 autovalores finitos em -1,3,1 e eta1=0,eta2=3
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(F);
[Qd(1),Zd(1)]    //parte eps. é sum(epsi) x (sum(epsi) + número de epsi) (sum="
[Qd(2),Zd(2)]    //parte infinita
[Qd(3),Zd(3)]    //parte finita
[Qd(4),Zd(4)]    //parte eta é (sum(etai) + number(eta1)) x sum(etai) (number=n
numbeps
numbeta
```

# Ver Também

gschur, gspec, systmat, pencan, randpencil, trzeros

# Name

linsolve — solucionador de equações lineares

```
[x0,kerA]=linsolve(A,b [,x0])
```

# Parâmetros

A
: uma matriz `na x ma` de reais (possivelmente esparsa)

b
: um vetor `na x 1` (mesma dimensão de linha de `A`)

x0
: um vetor de reais

kerA
: uma matriz `ma x k` de reais

# Descrição

`linsolve` computa todas as soluções para `A*x+b=0`.

`x0` é uma solução particular (se houver) e `kerA=` núcleo de `A`. Qualquer `x=x0+kerA*w` com `w` arbitrário satisfaz `A*x+b=0`.

Se um compatible `x0` compatível é dado na entrada, `x0` é retornado. Senão, um `x0`, compatível é retornado, se houver.

# Exemplos

```
A=rand(5,3)*rand(3,8);
b=A*ones(8,1);[x,kerA]=linsolve(A,b);A*x+b   //b comatível
b=ones(5,1);[x,kerA]=linsolve(A,b);A*x+b   //b incompatível
A=rand(5,5);[x,kerA]=linsolve(A,b), -inv(A)*b  //x é único

// A benchmark of sparse linear solver

[A,descr,ref,mtype] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa"

b = 0*ones(size(A,1),1);

tic();
res = umfpack(A,'\',b);
printf('\ntime needed to solve the system with umfpack: %.3f\n',toc());

tic();
res = linsolve(A,b);
printf('\ntime needed to solve the system with linsolve: %.3f\n',toc());

tic();
res = A\b;
printf('\ntime needed to solve the system with the backslash operator: %.3f\n',
```

# Ver Também

inv, pinv, colcomp, im_inv, backslash, umfpack

# Name

lsq — problemas de mínimos quadrados lineares

```
X=lsq(A,B [,tol])
```

# Parâmetros

A
: matriz de reais ou de complexos (m x n)

B
: matriz de reais ou de complexos (m x p)

tol
: escalar positivo, usado para determinar o posto efetivo de A (definido como sendo a ordem da maior submatriz triangular R11 regente na fatoração QR com pivoteamento de A, cujo número de condicionamento estimado <= 1/tol. O valor padrão de tol é `sqrt(%eps)`.

X
: matriz de reais ou complexos (n x p)

# Descrição

`X=lsq(A,B)` computa a solução de mínimo quadrado de menor norma da equação `A*X=B`, enquanto `X=A \ B` computa uma solução de mínimo quadrado com no máximo `posto(A)` componentes não-nulos por coluna.

# Referências

`lsq` é baseado nas funções Lapack DGELSY para matrizes de reais e ZGELSY para matrizes de complexos.

# Exemplos

```
//Construindo os dados
x=(1:10)';

y1=3*x+4.5+3*rand(x,'normal');
y2=1.8*x+0.5+2*rand(x,'normal');
plot2d(x,[y1,y2],[-2,-3])
//Achando a regressão linear
A=[x,ones(x)];B=[y1,y2];
X=lsq(A,B);

y1e=X(1,1)*x+X(2,1);
y2e=X(1,2)*x+X(2,2);
plot2d(x,[y1e,y2e],[2,3])

//Diferença entre lsq(A,b) e A\b
A=rand(4,2)*rand(2,3);//uma matriz de posto 2
b=rand(4,1);
X1=lsq(A,b)
X2=A\b
[A*X1-b, A*X2-b] //os resíduos são os mesmos
```

## Ver Também

backslash, inv, pinv, rank

# Name

lu — fatores LU de eliminação Gaussiana

```
[L,U]= lu(A)
[L,U,E]= lu(A)
```

# Parâmetros

A
: matriz de reais ou complexos (m x n)

L
: matriz de reais ou complexos (m x min(m,n))

U
: matriz de reais ou complexos (min(m,n) x n )

E
: uma matriz de permutação (n x n)

# Descrição

`[L,U]= lu(A)` produz duas matrizes `L` e `U` tais que `A = L*U` com `U` triangular superior e `E*L` triangular infeiror para uma matriz de permutação `E`.

Se `A` tem posto `k`, as linhas de `k+1` a `n` de `U` são zeros.

`[L,U,E]= lu(A)` produz três matrizes `L`, `U` e `E` tais que `E*A = L*U` com `U` triangular superior e `E*L` l triangular inferior para uma matriz de permutação `E`.

Se `A` é uma matriz de reais, usando as funções `lufact` e `luget` é possível obter as matrizes de permutação e, também, quando `A` não é de posto cheio, a compressão de colunas da matriz `L`.

# Exemplos

```
a=rand(4,4);
[l,u]=lu(a)
norm(l*u-a)

[h,rk]=lufact(sparse(a))  // função lufact com matrizes esparsas
[P,L,U,Q]=luget(h);
ludel(h)
P=full(P);L=full(L);U=full(U);Q=full(Q);
norm(P*L*U*Q-a) // P e Q são as matrizes de permutação
```

# Ver Também

lufact, luget, lusolve, qr, svd

# Função Usada

As decomposições de lu são baseadas nas rotinas de Lapack DGETRF para matrizes reais e ZGETRF para o caso de matrizes complexas.

# Name

lyap — equação de Lyapunov

```
[X]=lyap(A,C,'c')
[X]=lyap(A,C,'d')
```

# Parâmetros

A, C
: matrizes quadradas de reais, `C` deve ser simétrica

# Descrição

`X = lyap(A,C,flag)` resolve as equações matriciais de tempo contínuo ou de tempo discreto de Lyapunov:

```
A'*X + X*A = C       ( flag='c' )
A'*X*A - X = C       ( flag='d' )
```

Perceba que existe uma única solução se e só se um autovalor de `A` não é um autovalor de `-A` (`flag='c'`) ou 1 sobre um autovalor de `A` (`flag='d'`).

# Exemplos

```
A=rand(4,4);C=rand(A);C=C+C';
X=lyap(A,C,'c');
A'*X + X*A -C
X=lyap(A,C,'d');
A'*X*A - X -C
```

# Ver Também

sylv, ctr_gram, obs_gram

# Name

nlev — algoritmo de Leverrier

```
[num,den]=nlev(A,z [,rmax])
```

# Parâmetros

A
: matriz quadrada de reais

z
: string

rmax
: parâmetro opcional (ver `bdiag`)

# Descrição

`[num,den]=nlev(A,z [,rmax])` computa `(z*eye()-A)^(-1)`

por diagonalização por blocos de A seguido pelo algoritmo de Leverrier em cada bloco.

Este algoritmo é melhor que o algoritmo usual de Leverrier, mas ainda não está perfeito!

# Exemplos

```
A=rand(3,3);x=poly(0,'x');
[NUM,den]=nlev(A,'x')
clean(den-poly(A,'x'))
clean(NUM/den-inv(x*eye()-A))
```

# Ver Também

coff, coffg, glever, ss2tf

# Autores

F. Delebecque., S. Steer INRIA;

# Name

orth — base ortogonal

```
Q=orth(A)
```

# Parâmetros

A
: matriz de reais ou complexos

Q
: matriz de reais ou complexos

# Descrição

`Q=orth(A)` retorna `Q`, uma base ortogonal para o gerado de `A`. Im(`Q`) = Im(`A`) e `Q'*Q=eye`.

O número de colunas de `Q` é o posto de `A` como determinado pelo algoritmo QR.

# Exemplos

```
A=rand(5,3)*rand(3,4);
[X,dim]=rowcomp(A);X=X';
svd([orth(A),X(:,1:dim)])
```

# Ver Também

qr, rowcomp, colcomp, range

# Name

pbig — autoprojeção

```
[Q,M]=pbig(A,thres,flag)
```

# Parâmetros

A
: matriz quadrada de reais

thres
: número real

flag
: string (`'c'` ou `'d'`)

Q,M
: matrizes de reais

# Descrição

Projeção sobre um auto-subespaço associado aos autovalores de parte real>= `thres` (`flag='c'`) ou com magnitude >= `thres` (`flag='d'`).

A projeção é definida por `Q*M`, `Q` tem posto-coluna cheio, `M` tem posto-linha cheio e `M*Q=eye`.

Se `flag='c'`, os autovalores de `M*A*Q` = autovalores de `A` com parte real >= `thres`.

Se `flag='d'`, os autovalores de `M*A*Q` = autovalores de `A` com magnitude >= `thres`.

Se `flag='c'` e se `[Q1,M1]` = fatoração em posto cheio (`fullrf`) de `eye()-Q*M` então os autovalores de `M1*A*Q1` = autovalores de `A` com parte real < `thres`.

Se `flag='d'` e se `[Q1,M1]` =fatoração em posto cheio (`fullrf`) de `eye()-Q*M` então os autovalores de `M1*A*Q1` = autovalores de `A` com magnitude < `thres`.

# Exemplos

```
A=diag([1,2,3]);X=rand(A);A=inv(X)*A*X;
[Q,M]=pbig(A,1.5,'d');
spec(M*A*Q)
[Q1,M1]=fullrf(eye()-Q*M);
spec(M1*A*Q1)
```

# Ver Também

psmall, projspec, fullrf, schur

# Autor

F. D. (1988); ;

# Função Usada

`pbig` é baseada na forma ordenada de Schur (função do Scilab `schur`).

# Name

pencan — forma canônica de feixe de matrizes

```
[Q,M,i1]=pencan(Fs)
[Q,M,i1]=pencan(E,A)
```

# Parâmetros

Fs
: um feixe regular `s*E-A`

E,A
: duas matrizes quadradas de reais

Q,M
: duas matrizes não-singulares de reais

i1
: inteiro

# Descrição

Dado o feixe regular `Fs=s*E-A`, `pencan` retorna as matrizes `Q` e `M` tais que `M*(s*E-A)*Q` está na forma "canônica".

`M*E*Q` é uma matriz de blocos

```
[I,0;
 0,N]
```

com `N` nilpotente e `i1` = tamanho da matriz acima `I`.

`M*A*Q` é uma matriz de blocos:

```
[Ar,0;
 0,I]
```

# Exemplos

```
F=randpencil([],[1,2],[1,2,3],[]);
F=rand(6,6)*F*rand(6,6);
[Q,M,i1]=pencan(F);
W=clean(M*F*Q)
roots(det(W(1:i1,1:i1)))
det(W($-2:$,$-2:$))
```

# Ver Também

glever, penlaur, rowshuff

# Autor

F. D.; ;

# Name

penlaur — Laurent coefficients of matrix pencil

```
[Si,Pi,Di,order]=penlaur(Fs)
[Si,Pi,Di,order]=penlaur(E,A)
```

# Parâmetros

Fs
: um feixe regular `s*E-A`

E, A
: duas matrizes quadradas de reais

Si,Pi,Di
: três matrizes quadradas de reais

order
: inteiro

# Descrição

`penlaur` computa os primeiros coeficientes de Laurent de `(s*E-A)^-1` no infinito.

`(s*E-A)^-1 = ... + Si/s - Pi - s*Di + ...` em `s` = infinito.

`order` = ordem da singularidade (ordem=índice-1).

O feixe de matrizes `Fs=s*E-A` deve ser invertível.

Para um feixe de índice 0, `Pi, Di,...` são zero e `Si=inv(E)`.

Para um feixe de índice 1 (order=0),`Di` =0.

Para feixes de índices maiores, os termos `-s^2 Di(2), -s^3 Di(3),...` são dados por:

`Di(2)=Di*A*Di`, `Di(3)=Di*A*Di*A*Di` (até `Di(order)`).

# Observação

Versão experimental: há problemas quando se tem mal-condicionamento de`so*E-A`

# Exemplos

```
F=randpencil([],[1,2],[1,2,3],[]);
F=rand(6,6)*F*rand(6,6);[E,A]=pen2ea(F);
[Si,Pi,Di]=penlaur(F);
[Bfs,Bis,chis]=glever(F);
norm(coeff(Bis,1)-Di,1)
```

# Ver Também

glever, pencan, rowshuff

# Autor

F. Delebecque INRIA(1988,1990) ;

# Name

pinv — pseudo-inversa

```
pinv(A,[tol])
```

# Parâmetros

A
: matriz de reais ou complexos

tol
: número real

# Descrição

`X = pinv(A)` produz uma matriz `X` de mesma dimensão que `A'` tal que:

`A*X*A = A, X*A*X = X` e ambas `A*X` e `X*A` são Hermitianas.

A computação é baseada em SVD e qualquer valor singular abaixo da tolerância é tratado como zero: esta tolerância é acessada por `X=pinv(A,tol)`.

# Exemplos

```
A=rand(5,2)*rand(2,4);
norm(A*pinv(A)*A-A,1)
```

# Ver Também

rank, svd, qr

# Função Usada

`pinv` é baseada na decomposição em valores singulares (função do Scilab `svd`).

# Name

polar — forma polar

```
[Ro,Theta]=polar(A)
```

# Parâmetros

A
: matriz quadrada de reais ou complexos

Ro,
: matriz de reais

Theta,
: matriz de reais ou complexos

# Descrição

`[Ro,Theta]=polar(A)` retorna a forma polar de `A` i.e. `A=Ro*expm(%i*Theta)Ro` simétrico >= 0 e `Theta` hermitiano >=0.

# Exemplos

```
A=rand(5,5);
[Ro,Theta]=polar(A);
norm(A-Ro*expm(%i*Theta),1)
```

# Ver Também

expm, svd

# Autor

F. Delebecque INRIA; ;

# Name

proj — projeção

```
P = proj(X1,X2)
```

# Parâmetros

X1,X2
: duas matrizes reais com igual número de colunas

P
: matriz de projeção de real (`P^2=P`)

# Descrição

`P` é a projeção sobre `X2` paralela a `X1`.

# Ver Também

```
X1=rand(5,2);X2=rand(5,3);
P=proj(X1,X2);
norm(P^2-P,1)
trace(P)    // Este é dim(X2)
[Q,M]=fullrf(P);
svd([Q,X2])   // span(Q) = span(X2)
```

# See Also

projspec, orth, fullrf

# Autor

F. D.; ;

# Name

projspec — operadores espectrais

```
[S,P,D,i]=projspec(A)
```

# Parâmetros

A
: matriz quadrada

S, P, D
: matrizes quadradas

i
: inteiro (índice do autovalor zero de A).

# Descrição

Características espectrais de A em 0.

S = resolvente reduzido em 0 (S = -Inverso_de_Drazin(A)).

P = projeção espectral em 0.

D = operador nilpotente em 0.

index = índice do autovalor 0.

Tem-se `(s*eye()-A)^(-1) = D^(i-1)/s^i +... + D/s^2 + P/s - S - s*S^2 -...` ao redor da singularidade s=0.

# Exemplos

```
deff('j=jdrn(n)','j=zeros(n,n);for k=1:n-1;j(k,k+1)=1;end')
A=sysdiag(jdrn(3),jdrn(2),rand(2,2));X=rand(7,7);
A=X*A*inv(X);
[S,P,D,index]=projspec(A);
index   //tamanho do bloco J
trace(P)  //soma das dimensões dos blocos J
A*S-(eye()-P)
norm(D^index,1)
```

# Ver Também

coff

# Autor

F. D.; ;

# Name

psmall — projeção espectral

```
[Q,M]=psmall(A,thres,flag)
```

# Parameters

A
: matriz quadrada de reais

thres
: número real

flag
: string (`'c'` ou `'d'`)

Q,M
: matrizes de reais

# Description

Projeção sobre auto-subespaço associado com autovalores com parte real < `thres` (`flag='c'`) ou com módulo < `thres` (`flag='d'`).

A projeção é definda por `Q*M`, `Q` é de posto-coluna cheio, `M` é de posto-linha cheio e `M*Q=eye`.

Se `flag='c'`, os autovalores de `M*A*Q` = autovalores de `A` com parte real < `thres`.

Se `flag='d'`, os autovalores de `M*A*Q` = autovalores de `A` com magnitude < `thres`.

Se `flag='c'` e se `[Q1,M1]` = fatoração em posto cheio (`fullrf`) de `eye()-Q*M`então os autovalores de `M1*A*Q1` = autovalores de `A` com parte real >= `thres`.

Se `flag='d'` e se `[Q1,M1]` = fatoração em posto cheio (`fullrf`) de `eye()-Q*M` então os autovalores de `M1*A*Q1` = autovalores de `A` com magnitude >= `thres`.

# Examples

```
A=diag([1,2,3]);X=rand(A);A=inv(X)*A*X;
[Q,M]=psmall(A,2.5,'d');
spec(M*A*Q)
[Q1,M1]=fullrf(eye()-Q*M);
spec(M1*A*Q1)
```

# See Also

pbig, proj, projspec

# Authors

F. Delebecque INRIA. (1988);

# Used Functions

Esta função é baseada na forma de Schur ordenada (Função do Scilab `schur`).

# Name

qr — QR decomposição

```
[Q,R]=qr(X [,"e"])
[Q,R,E]=qr(X [,"e"])
[Q,R,rk,E]=qr(X [,tol])
```

# Parâmetros

X
: matriz de reais ou complexos

tol
: número real não-negativo

Q
: matriz quadrada ortogonal ou unitária

R
: matriz com as mesmas dimensões de `X`

E
: matriz de permutação

rk
: inteiro (posto QR de `X`)

# Descrição

[Q,R] = qr(X)
: pproduz uma matriz triangular superior `R` de mesma dimensão que `X` e uma matriz ortogonal (unitária no caso de matriz de complexos) `Q` tais que `X = Q*R`. `[Q,R] = qr(X,"e")` produz um "economia de tamanho": Se `X` é m-por-n com m > n, então, apenas as primeiras n colunas de `Q` são computadas assim como as primeiras n linhas de `R`.

  De `Q*R = X` , segue que a k-ésima coluna da matriz `X`, é expressa como combinação linear das k primeiras colunas de `Q` (com coeficientes `R(1,k), ..., R(k,k)` ). As k primeiras colunas de `Q` formam uma base ortogonal para o subespaço gerado pelas k priemiras colunas de `X`. Se a coluna `k` de `X` (i.e. `X(:,k)` ) é uma combinação linear das `p` primeiras colunas de `X`, então, as entradas de `R(p+1,k), ..., R(k,k)` são zeros. Neste caso, `R` é trapezoidal superior. Se `X` tem posto `rk`, as linhas `R(rk+1,:), R(rk+2,:), ...` são zeros.

[Q,R,E] = qr(X)
: produz uma matriz de permutação (de colunas) `E`, uma matriz triangular superior `R` com elementos na diagonal decrescentes e uma matriz ortogonal (ou unitaria) `Q` tais que `X*E = Q*R`. Se `rk` é o posto de `X`, as `rk` primeiras entradas ao longo da diagonal de `R`, i.e. `R(1,1), R(2,2), ..., R(rk,rk)` são todas diferentes de zero. `[Q,R,E] = qr(X,"e")` produz uma "economia de tamanho": Se `X` ié m-por-n com m > n, então, apenas as n primeiras colunas de `Q` são computadas tanto quanto as n priemiras linhas de `R`.

[Q,R,rk,E] = qr(X ,tol)
: retorna `rk` = estimativa do posto de `X` i.e. `rk` é o número elementos da diagonal de `R` que são maiores que um dado limiar `tol`.

[Q,R,rk,E] = qr(X)
: retorna `rk` = estimativa do posto de `X` i.e. `rk` é o número de elementos da diagonal de `R` que são maiores que `tol=R(1,1)*%eps*max(size(R))`. Veja `rankqr` para uma fatoração QR que revela o posto usando o número de condicionamento de `R`.

# Exemplos

```
// fatoração QR, caso genérico
// X é alta (posto cheio)
X=rand(5,2);[Q,R]=qr(X); [Q'*X R]

//X é gorda (posto cheio)
X=rand(2,3);[Q,R]=qr(X); [Q'*X R]

//coluna 4 de X é uma combinação linear das colunas 1 e 2:
X=rand(8,5);X(:,4)=X(:,1)+X(:,2); [Q,R]=qr(X); R, R(:,4)

//X tem posto 2, linhas 3 a $ de R são zero:
X=rand(8,2)*rand(2,5);[Q,R]=qr(X); R

//Avaliando o posto rk: pivotação por colunas ==&gt; rk primeiras
//As entradas diagonais de R são não-nulas:
A=rand(5,2)*rand(2,5);
[Q,R,rk,E] = qr(A,1.d-10);
norm(Q'*A-R)
svd([A,Q(:,1:rk)])    //span(A) =span(Q(:,1:rk)) (span="gerado")
```

# Ver Também

rankqr, rank, svd, rowcomp, colcomp

# Funções Usadas

A decomposição qr é baseada nas rotinas de Lapack DGEQRF, DGEQPF, DORGQR para as matrizes de reais ZGEQRF, ZGEQPF, ZORGQR para as matrizes de complexos.

# Name

quaskro — forma quasi-Kronecker

```
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(F)
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(E,A)
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(F,tol)
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(E,A,tol)
```

# Parâmetros

F
: feixe de matrizes de reais `F=s*E-A` (`s=poly(0,'s')`)

E,A
: duas matrizes reais de iguais dimensões

tol
: número real (tolerância, valor padrão=1.d-10)

Q,Z
: duas matrizes quadradas ortogonais

Qd,Zd
: dois vetores de inteiros

numbeps
: vetor de inteiros

# Descrição

Forma quasi-Kronecker de um feixe de matrizes: `quaskro` computa duas matrizes ortogonais `Q, Z` que põem o feixe `F=s*E -A` na forma triangular superior:

```
           | sE(eps)-A(eps) |        X       |      X     |
           |----------------|----------------|------------|
           |        0       | sE(inf)-A(inf) |      X     |
Q(sE-A)Z = |================================|============|
           |                                |            |
           |                0               | sE(r)-A(r) |
```

As dimensões dos blocos são dadas por:

`eps=Qd(1) x Zd(1)`, `inf=Qd(2) x Zd(2)`, `r = Qd(3) x Zd(3)`

O bloco `inf` contém os modos infinitos do feixe.

O bloco `f` contém os modos finitos do feixe

A estrutura dos blocos epsilon é dada por:

`numbeps(1)` = # de blocos eps de tamanho 0 x 1

`numbeps(2)` = # de blocos eps de tamanho 1 x 2

`numbeps(3)` = # de blocos eps de tamanho 2 x 3 etc...

A forma completa (de quatro blocos) de Kronecker é dada pela função `kroneck` que chama a função `quaskro` sobre o feixe (pertransposto) `sE(r)-A(r)`.

O código é retirado de T. Beelen.

# Ver Também

kroneck, gschur, gspec

# Name

randpencil — feixe aleatório

```
F=randpencil(eps,infi,fin,eta)
```

# Parâmetros

eps
: vetor de inteiros

infi
: vetor de inteiros

fin
: vetor de reais, ou polinômio mônico, ou vetor de polinômios mônicos

eta
: vetor de inteiros

F
: feixe de matrizes de reais `F=s*E-A(s=poly(0,'s'))`

# Descrição

Função utilitária. `F=randpencil(eps,infi,fin,eta)` retorna um feixe aleatório `F` com dada estrutura de Kronecker. A estrutura é dada por: `eps=[eps1,...,epsk]`: estrutura de blocos epsilon (tamanho eps1x(eps1+1),....) `fin=[l1,...,ln]` conjunto de autovalores finitos (assumidos como reais) (possivelmente []) `infi=[k1,...,kp]` tamanho de blocos J no infinito `ki>=1` (infi=[] se não há blocos J). `eta=[eta1,...,etap]`: estrutura dos blocos eta (size eta1+1)xeta1,...)

`epsi`'s devem ser >=0, `etai`'s devem ser >=0, `infi`'s devem ser >=1.

Se `fin` é um polinômio (mônico), o bloco finito admite raízes de `fin` como autovalores.

Se `fin` é um vetor de polinômios, eles são os divisores elementares finitos de `F` ,i.e., as raízes de `p(i)` são autovalores finitos de `F`.

# Exemplos

```
F=randpencil([0,1],[2],[-1,0,1],[3]);
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(F);
Qd, Zd
s=poly(0,'s');
F=randpencil([],[1,2],s^3-2,[]); //feixe regular
det(F)
```

# Ver Também

kroneck, pencan, penlaur

# Name

range — Imagem (gerado) de A^k

```
[X,dim]=range(A,k)
```

# Parâmetros

A
: matriz de reais ou complexos que se assume quadrada se k>1

k
: inteiro

X
: matriz ortonormal

dim
: inteiro (dimensão de subespaço)

# Descrição

Computação da imagem de `A^k` ; as primeiras dim colunas de `X` geram a imagem de `A^k`. As últimas linhas de `X` geram o complemento ortogonal da imagem. `X*X'` é a matriz identidade.

# Exemplos

```
A=rand(4,2)*rand(2,4);   // 4 vetores colunas, 2 independentes.
[X,dim]=range(A,1);dim   // computando a imagem

y1=A*rand(4,1);          //um vetor que está na imagem de A
y2=rand(4,1);            //um vetor que não está na imagem de A
norm(X(dim+1:$,:)*y1)    //as últimas entradas são zeros, y1 está na imagem de A
norm(X(dim+1:$,:)*y2)    //as últimas entradas não são zeros

I=X(1:dim,:)'            //I é uma base para a imagem
coeffs=X(1:dim,:)*y1     //componentes de y1 relativos à base I

norm(I*coeffs-y1)        //verificando
```

# Ver Também

fullrfk, rowcomp

# Autor

F. D. INRIA ;

# Função Usada

A função `range` é baseada na função rowcomp que usa decomposição svd (decomposição em valores singulares).

# Name

rank — posto

```
[i]=rank(X)
[i]=rank(X,tol)
```

# Parâmetros

X
: matriz de reais ou complexos

tol
: número real não-negativo

# Descrição

`rank(X)` é o posto numérico de `X` i.e. o número de valores singulares de X que são maiores que `norm(size(X),'inf') * norm(X) * %eps`.

`rank(X,tol)` é o número de valores singulares de `X` que são maiores que `tol`.

Note que o valor padrão de `tol` é proporcional a `norm(X)`. Como conseqüência, `rank([1.d-80,0;0,1.d-80])` é 2 !.

# Exemplos

```
rank([1.d-80,0;0,1.d-80])
rank([1,0;0,1.d-80])
```

# Ver Também

svd, qr, rowcomp, colcomp, lu

# Name

rankqr — fatoração QR com revelação do posto

```
[Q,R,JPVT,RANK,SVAL]=rankqr(A, [RCOND,JPVT])
```

# Parâmetros

A
: matriz de reais ou complexos

RCOND
: número real usado para determinar o posto efetivo de A, que é definido como sendo a ordem da maior submatriz regente triangular R11 na fatoração QR com pivoteamento de A, cujo número de condicionamento estimado é < 1/RCOND.

JPVT
: vetor de inteiros nas entradas, se JPVT(i) não é 0, a i-ésimo coluna de A é permtutada para a frente de AP, senão, a coluna i é uma coluna livre. Na saída, se JPVT(i) = k, então a i-ésima coluna de A*P era a k-ésima coluna de A.

RANK
: posto efetivo de A, i.e., a ordem da submatriz R11. É o mesmo que a ordem da submatriz T1 na fatoração ortogonal completa de A.

SVAL
: vetor de reais com 3 componentes; as estimativas de alguns dos valores singulares do fator triangular R.

  SVAL(1) é o maior valor singular de R(1:RANK,1:RANK);

  SVAL(2) é o menor valor singular de R(1:RANK,1:RANK);

  SVAL(3) é o menor valor singular de R(1:RANK+1,1:RANK+1), se RANK < MIN(M,N), ou de R(1:RANK,1:RANK), caso contrário.

# Descrição

Computa (opcionalmente) uma fatoração QR com revelação do posto de uma matriz de reais geral M-por-N, ou de complexos A, que pode ser deficiente de posto, e estima seu posto efetivo usando estimativa de condição incremental.

A rotina usa uma fatoração QR com pivoteamento de colunas:

```
A * P = Q * R,  onde  R = [ R11 R12 ],
                          [  0  R22 ]
```

com R11 definida como a maior submatriz regente cujo número de condição estimado é menor que 1/RCOND. A ordem de R11, RANK, é o posto efetivo deA.

Se a fatoração triangular revela o posto (que será o caso se as colunas regentes forem bem condicionadas), então SVAL(1) também será uma estimativa para o maior valor singular de A, e SVAL(2) e SVAL(3) serão estimativas para o RANK-ésimo e (RANK+1)-ésimo valores singulares de A, respectivamente.

Examinando-se estes valores, pode-se confirmar que o posto é bem definido a respeito do valor escolhido de RCOND. A razão SVAL(1)/SVAL(2) é uma estimativa do número de condicionamento de R(1:RANK,1:RANK).

# Exemplos

```
A=rand(5,3)*rand(3,7);
[Q,R,JPVT,RANK,SVAL]=rankqr(A,%eps)
```

# Ver Também

qr, rank

# Funções Usadas

Rotinas da biblioteca Slicot MB03OD, ZB03OD.

# Name

rcond — número de condicionamento inverso

```
rcond(X)
```

# Parâmetros

X
: matriz quadrada de reais ou complexos

# Descrição

`rcond(X)` é uma estimativa para a recíproca da condição de `X` na norma-1.

Se `X` é bem condicionada, `rcond(X)` é próximo a 1. Senão, `rcond(X)` é próximo a 0.

`[r,z]=rcond(X)` ajusta `r` a `rcond(X)` e retorna `z` tal que `norm(X*z,1) = r*norm(X,1)*norm(z,1)`

Portanto, se `rcond` é pequeno, `z` é um vetor do núcleo.

# Exemplos

```
A=diag([1:10]);
rcond(A)
A(1,1)=0.000001;
rcond(A)
```

# Ver Também

svd, cond, inv

# Name

rowcomp — compressão de linhas, imagem

```
[W,rk]=rowcomp(A [,flag [,tol]])
```

# Parâmetros

A
: matriz de reais ou de complexos

flag
: string opcional, com valores possíveis `'svd'` ou `'qr'`. O valor padrão é `'svd'`.

tol
: número real não-negativo opcional. O valor padrão é `sqrt(%eps)*norm(A,1)`.

W
: matriz quadrada não-singular (matriz mudança de base)

rk
: inteiro (posto de `A`)

# Descrição

Compressão de linhas de `A`. `Ac = W*A` é uma matriz de linhas comprimidas, i.e. `Ac=[Af;0]` com `Af` de posto-linha cheio.

`flag` e `tol` são parâmetros opcionais: `flag='qr'` ou `'svd'` (o padrão é `'svd'`).

`tol` é um parâmetro de tolerância.

As `rk` primeiras colunas de `W'` geram a imagem de `A`.

As `rk` primeiras linhas (do topo) de `W` geram a imagem de linha de `A`.

Um vetor não nulo `x` pertence à Im(`A`) se,e só se, `W*x` é de linhas comprimidas de acordo com `Ac` i.e, a norma de seus últimos componentes é pequena com relação a dos seus primeiros componentes.

# Exemplos

```
A=rand(5,2)*rand(2,4);              //  4 vetores colunas, 2 independentes
[X,dim]=rowcomp(A);Xp=X';
svd([Xp(:,1:dim),A])                //span(A) = span(Xp(:,1:dim) (span="gerado"
x=A*rand(4,1);                      //x pertence a span(A)
y=X*x
norm(y(dim+1:$))/norm(y(1:dim))     // pequeno
```

# Ver Também

colcomp, fullrf, fullrfk

# Autor

F. D.; INRIA

# Função Usada

A função `rowcomp` é baseada nas decomposições svd (decomposição em valores singulares) ou qr .

# Name

rowshuff — algoritmo de embaralhamento

```
[Ws,Fs1]=rowshuff(Fs, [alfa])
```

# Parâmetros

Fs
: feixe quadrado de reais `Fs = s*E-A`

Ws
: matriz de polinômios

Fs1
: feixe quadrado de reais `F1s = s*E1 -A1` com `E1` não-singular

alfa
: número real (`alfa = 0` é o valor padrão)

# Descrição

Algoritmo de embaralhamento: dado o feixe `Fs=s*E-A`, retorna Ws=W(s) (matriz quadrada de polinômios) tal que:

`Fs1 = s*E1-A1 = W(s)*(s*E-A)` é um feixe com matriz `E1` não-singular.

Isto é possível se, e só se, o feixe `Fs = s*E-A` é regular (i.e., invertível). O grau de `Ws` é igual ao índice do feixe.

Os pólos no infinito de `Fs` asão colocados para `alfa` e os zeros de `Ws` estão em `alfa`.

Note que `(s*E-A)^-1 = (s*E1-A1)^-1 * W(s) = (W(s)*(s*E-A))^-1 *W(s)`

# Exemplos

```
F=randpencil([],[2],[1,2,3],[]);
F=rand(5,5)*F*rand(5,5);   // feixe regular  5 x 5 com três avaliações em 1,2,3
[Ws,F1]=rowshuff(F,-1);
[E1,A1]=pen2ea(F1);
svd(E1)           //E1 não-singular
roots(det(Ws))
clean(inv(F)-inv(F1)*Ws,1.d-7)
```

# Ver Também

pencan, glever, penlaur

# Autor

F. D.; ; ; ; ;

# Name

rref — computa a matriz-linha reduzida a forma escada por transformações de LU

```
R=rref(A)
```

# Parâmetros

A
: matriz m x n com entradas escalares

R
: matriz m x n, forma escada de A

# Descrição

`rref` computa a forma escada de linhas reduzidas da matriz dada pela decomposição esquerda LU. Se for necessária a transformação usada, basta chamar `X=rref([A,eye(m,m)])` A forma escada de linhas reduzidas `R` é `X(:,1:n)` e a transformação esquerda `L` ié dada por `X(:,n+1:n+m)` tal como `L*A=R`

# Exemplos

```
A=[1 2;3 4;5 6];
X=rref([A,eye(3,3)]);
R=X(:,1:2)
L=X(:,3:5);L*A
```

# Ver Também

lu, qr

# Name

schur — decomposição (ordenada) de Schur de matrizes e feixes

```
[U,T] = schur(A)
[U,dim [,T] ]=schur(A,flag)
[U,dim [,T] ]=schur(A,extern1)

[As,Es [,Q,Z]]=schur(A,E)
[As,Es [,Q],Z,dim] = schur(A,E,flag)
[Z,dim] = schur(A,E,flag)
[As,Es [,Q],Z,dim]= schur(A,E,extern2)
[Z,dim]= schur(A,E,extern2)
```

# Parâmetros

A
: matriz quadrada de reais ou complexos

E
: matriz quadrada de reais ou complexos com as mesmas dimensões de `A`.

flag
: string (`'c'` ou `'d'`)

extern1
: uma ``external'' (função externa), veja abaixo

extern2
: uma ``external'', veja abaixo

U
: matriz quadrada ortogonal ou unitária

Q
: matriz quadrada ortogonal ou unitária

Z
: matriz quadrada ortogonal ou unitária

T
: matriz quadrada triangular superior ou quasi-triangular

As
: matriz quadrada triangular superior ou quasi-triangular

Es
: matriz quadrada triangular superior

dim
: inteiro

# Descrição

Formas de Schur, formas ordenadas de Schur de matrizes e feixes

FORMA DE SCHUR MATRICIAL

Forma de Schur usual:

`[U,T] = schur(A)` produz uma matriz de Schur `T` e uma matriz unitária `U` tais que `A = U*T*U'` e `U'*U = eye(U)`. Por si mesmo, schur(`A`) retorna `T`. Se `A` é de complexos, a

Forma de Schur Complexa é retornada na matriz `T`. A Forma de Schur Complexa é triangular superior com os autovalores de `A` na diagonal. Se `A` é de reais, a Forma de Schur Real é retornada. A Forma de Schur Real tem autovalores reais na diagonal e os autovalores complexos em blocos 2-por-2 na diagonal.

Formas de Schur ordenadas

`[U,dim]=schur(A,'c')` rretorna uma matriz unitária `U` que transforma `A` em uma forma de Schur. Ainda, as primeiras dim colunas de `U` formam uma base para o autoespaço de `A` associado aos autovalores com partes reais negativas (autoespaço de "tempo contínuo" estável).

`[U,dim]=schur(A,'d')` retorna uma matriz unitária `U` que transforma `A` em uma forma de Schur. Ainda, as primeiras `dim` colunas de `U` geram uma base do autoespaço de `A` associado aos autovalores de magnitude menor que 1 (autoespaço de "tempo discreto" estável).

`[U,dim]=schur(A,extern1)` retorna uma matriz unitária `U` que transforma `A` em uma forma de Schur. Ainda, as `dim` primeiras colunas de `U` geram uma base para o autoespaço de `A` associado aos autovalores que são selecionados pela "external" `extern1` (veja "external" para detalhes). Esta "external" pode ser descrita por uma função do Scilab ou por um "procedure" de C ou FORTRAN:

Uma Função do Scilab

Se `extern1` é descrita por uma função do Scilab, deve ter a seguinte seqüência de chamamento: `s=extern1(Ev)`, onde `Ev` é um autovalor e `s` um booleano.

Um "Procedure" C ou FORTRAN

Se `extern1` é descrita por uma função de C ou FORTRAN, deve ter a seguinte seqüência de chamamento: `int extern1(double *EvR, double *EvI)` onde `EvR` e `EvI` são partes real e complexa de autovalor. Valor verdadeiro ou diferente de zero retornado significa autovalor selecionado.

FORMAS DE SCHUR DE FEIXES

Forma de Schur de Feixe Usual

`[As,Es] = schur(A,E)` produz uma matriz `As` quasi-triangular e uma matriz triangular `Es` que são a forma generalizada de Schur do par `A, E`.

`[As,Es,Q,Z] = schur(A,E)` retorna, ainda, duas matrizes unitárias `Q` e `Z` tais que `As=Q'*A*Z` e `Es=Q'*E*Z`.

Formas de Schur Ordenadas:

`[As,Es,Z,dim] = schur(A,E,'c')` retorna a forma real generalizada de Schur do feixe `s*E-A`. Ainda, as primeiras dim colunas de `Z` geram uma base para o autoespaço direito associado aos autovalores com partes reais negativas (autoespaço de "tempo contínuo" generalizado).

```
[As,Es,Z,dim] = schur(A,E,'d')
```

retorna a forma real generalizada de Schur do feixe `s*E-A`. Ainda, as dim primeiras colunas de `Z` formam uma base para o autoespaço direito associado aos autovalores de magnitude menor que 1 (autoespaço de "tempo discreto" generalizado).

```
[As,Es,Z,dim] = schur(A,E,extern2)
```

retorna a forma real generalizada de Schur do feixe `s*E-A`. Ainda, as dim primeiras colunas de `Z` formam uma base para o autoespaço direito associado aos autovalores do feixe que são selecionados de acordo com a regra que é dada pela "external" `extern2`. (veja "external" para detalhes). Esta external pode ser descrita por uma função do Scilab ou por um "procedure" de C ou FORTRAN.

Função do Scilab
: Se `extern2` é descrita por uma função do Scilab, deve ter a seqüência de chamamento: `s=extern2(Alpha,Beta)`, onde `Alpha` e `Beta` definem um autovalor generalizado e `s` um booleano.

Um "Procedure" C ou FORTRAN
: Se a "external" `extern2` é descrita por um "procedure" C ou FORTRAN, deve ter a seqüência de chamamento:

```
int extern2(double *AlphaR, double *AlphaI, double *Beta)
```

se A e E são matrizes de reais e

```
int extern2(double *AlphaR, double *AlphaI, double *BetaR,
double *BetaI)
```

se A ou E é matriz de complexos. `Alpha`, e `Beta` definem o autovalor generalizado. Um valor verdadeiro ou diferente de zero siginfica autovalor generalizado selecionado.

# Referências

As computações da forma de Schur matricial são baseadas nas rotinas de Lapack DGEES e ZGEES.

As computações da forma de Schur de feixes são baseadas nas rotinas de Lapack DGGES e ZGGES.

# Exemplos

```
//FORMA SCHUR DE UMA MATRIZ
//----------------------
A=diag([-0.9,-2,2,0.9]);X=rand(A);A=inv(X)*A*X;
[U,T]=schur(A);T

[U,dim,T]=schur(A,'c');
T(1:dim,1:dim)      //autovalores estáveis contínuos

function t=mytest(Ev),t=abs(Ev)<0.95,endfunction
[U,dim,T]=schur(A,mytest);
T(1:dim,1:dim)

// A mesma função em C (um compilador é requerido)
C=['int mytest(double *EvR, double *EvI) {' //o codigo C
   'if (*EvR * *EvR + *EvI * *EvI < 0.9025) return 1;'
   'else return 0; }';]
mputl(C,TMPDIR+'/mytest.c')


//construindo e ligando
lp=ilib_for_link('mytest','mytest.c',[],'c',TMPDIR+'/Makefile');
link(lp,'mytest','c');

//executando
[U,dim,T]=schur(A,'mytest');
//FORMA SCHUR DE UM FEIXE
//----------------------
F=[-1,%s, 0,   1;
    0,-1,5-%s, 0;
    0, 0,2+%s, 0;
```

```
    1, 0, 0, -2+%s];
A=coeff(F,0);E=coeff(F,1);
[As,Es,Q,Z]=schur(A,E);
Q'*F*Z //isto é As+%s*Es

[As,Es,Z,dim] = schur(A,E,'c')
function t=mytest(Alpha,Beta),t=real(Alpha)&lt;0,endfunction
[As,Es,Z,dim] = schur(A,E,mytest)

//a mesma função em FORTRAN (um compilador é requerido)
ftn=['funcao inteira mytestf(ar,ai,b)' //codigo FORTRAN
      'double precision ar,ai,b'
      'mytestf=0'
      'if(ar.lt.0.0d0) mytestf=1'
      'end']
mputl('      '+ftn,TMPDIR+'/mytestf.f')

//construíndo e ligando
lp=ilib_for_link('mytestf','mytestf.f',[],'F',TMPDIR+'/Makefile');
link(lp,'mytestf','f');

//executando

[As,Es,Z,dim] = schur(A,E,'mytestf')
```

# Ver Também

spec, bdiag, ricc, pbig, psmall

# Name

spaninter — interseção de subespaços

```
[X,dim]=spaninter(A,B [,tol])
```

# Parâmetros

A, B
: duas matrizes de reais ou de complexos com igual número de linhas

X
: matriz quadrada ortogonal ou unitária

dim
: inteiro, dimensão do subesbaço Im(A) inter Im(B)

# Descrição

Computa a intereseção de Im(A) e Im(B).

As primeiras `dim` colunas de `X` geram esta interseção i.e. `X(:,1:dim)` é uma base ortogonal para

Im(A) inter Im(B)

Na base `X`, `A` e `B` são respectivamente representados por:

`X'*A` e `X'*B`.

`tol` é um limiar (`sqrt(%eps)` é o valor padrão).

# Exemplos

```
A=rand(5,3)*rand(3,4);     // A é 5 x 4, rank=3
B=[A(:,2),rand(5,1)]*rand(2,2);
[X,dim]=spaninter(A,B);
X1=X(:,1:dim);    //A interseção
svd(A),svd([X1,A])   // X1 no gerado(A) (span(A))
svd(B),svd([B,X1])   // X1 no gerado(B) (span(B))
```

# Ver Também

spanplus, spantwo

# Autor

F. D.; ;

# Name

spanplus — soma de subespaços

```
[X,dim,dima]=spanplus(A,B[,tol])
```

# Parâmetros

A, B
: duas matrizes de reais ou complexos com igual número de linhas

X
: matriz quadrada ortogonal ou unitária

dim, dima
: inteiros, dimensões de subespaços

tol
: número real não-negativo

# Descrição

Computa a base X tal que:

as primeiras `dima` colunas de `X` geram Im(`A`) e as (`dim-dima`) colunas seguintes formam uma base de `A+B` em relação a `A`.

As `dim` primeiras colunas de `X` formam uma base para `A+B`.

Tem-se a seguinte forma canônica para `[A,B]`:

```
         [*,*]    (dima rows)
X'*[A,B]=[0,*]    (dim-dima rows)
         [0,0]
```

`tol` é um argumento opcional (ver código da função).

# Exemplos

```
A=rand(6,2)*rand(2,5);      // rank(A)=2 (posto(A)=2)
B=[A(:,1),rand(6,2)]*rand(3,3);   //dois vetores adicionais independentes
[X,dim,dimA]=spanplus(A,B);
dimA
dim
```

# Ver Também

spaninter, im_inv, spantwo

# Autor

F. D.; ;

# Name

spantwo — soma e interseção de subespaços

```
[Xp,dima,dimb,dim]=spantwo(A,B, [tol])
```

# Parâmetros

A, B
: duas matrizes de reais ou complexos com igual número de linhas

Xp
: matriz quadrada não-singular

dima, dimb, dim
: inteiros, dimensões dos subespaços

tol
: número real não-negativo

# Descrição

Dadas duas matrizes `A` e `B` com o mesmo número de linhas, retorna uma matriz quadrada `Xp` (não-singular, mas não necessariamente ortogonal) tal que :

```
         [A1, 0]    (dim-dimb linhas)
Xp*[A,B]=[A2,B2]    (dima+dimb-dim linhas)
         [0, B3]    (dim-dima linha)
         [0 , 0]
```

As primeiras `dima` colunas de `inv(Xp)` geram Im(`A`).

As colunas de `dim-dimb+1` até `dima` de `inv(Xp)` geram a interseção de Im(A) e Im(B).

As primeiras `dim` colunas de `inv(Xp)` geram Im(`A`)+Im(`B`).

As colunas de `dim-dimb+1` até `dim` de `inv(Xp)` geram Im(`B`).

A matrix `[A1;A2]` tem posto-linha cheio (= posto(A)), a matrix `[B2;B3]` tem posto-linha cheio (=posto(B)), a matriz `[A2,B2]` tem posto-linha (=posto(A inter B)) e a matriz `[A1,0;A2,B2;0,B3]` tem posto-linha cheio (=posto(A+B)).

# Exemplos

```
A=[1,0,0,4;
   5,6,7,8;
   0,0,11,12;
   0,0,0,16];
B=[1,2,0,0]';C=[4,0,0,1];
Sl=ss2ss(syslin('c',A,B,C),rand(A));
[no,X]=contr(Sl('A'),Sl('B'));CO=X(:,1:no);  //Parte controlável
[uo,Y]=unobs(Sl('A'),Sl('C'));UO=Y(:,1:uo);  //Parte inobservável
[Xp,dimc,dimu,dim]=spantwo(CO,UO);    //Decomposição de Kalman
Slcan=ss2ss(Sl,inv(Xp));
```

# Ver Também

spanplus, spaninter

# Autor

F. D.

# Name

spec — autovalores de matrizes e feixes

```
evals=spec(A)
[R,diagevals]=spec(A)

evals=spec(A,B)
[alpha,beta]=spec(A,B)
[alpha,beta,Z]=spec(A,B)
[alpha,beta,Q,Z]=spec(A,B)
```

# Parâmetros

A
: matriz quadrada de reais ou complexos

B
: matriz quadrada de reais ou complexos com as mesmas dimensões que `A`

evals
: vetor de reais ou complexos, os autovalores

diagevals
: matriz diagonal de reais ou complexos (autovalores ao longo da diagonal)

alpha
: vetor de reais ou complexos, al./be fornece os autovalores

beta
: vetor de reais, al./be fornece os autovalores

R
: matriz quadrada de reais ou complexos invertível, autovetores direitos da matriz

Q
: matriz quadrada de reais ou complexos invertível, autovetores esquerdos do feixe

Z
: matriz quadrada de reais ou complexos invertível, autovetores direitos do feixe

# Descrição

evals=spec(A)
: retorna no vetor `evals` os autovalores.

[R,diagevals] =spec(A)
: retorna na matriz diagonal `evals` os autovalores e em `R` os autovetores direitos.

evals=spec(A,B)
: retorna o espectro do feixe de matrizes A - s B, i.e. as raízes da matriz de polinômios s B - A.

[alpha,beta] = spec(A,B)
: retorna o espectro do feixe de matrizes `A - s B`, i.e. as raízes da matriz de polinômios `A - s B`. Auto valores generalizados alpha e beta são tais que a matriz `A - alpha./beta B` é uma matriz singular. Os autovalores são dados por `al./be` e se `beta(i) = 0` o i-ésimo autovalor está no infinito. (Para `B = eye(A), alpha./beta` é `spec(A)`). É usualmente representado pelo par (alpha,beta), já que há uma interpretação razoável para beta=0, e até mesmo para os dois sendo zero.

[alpha,beta,R] = spec(A,B)
retorna, ainda, a matriz `R` de autovetores direitos generalizados do feixe.

[al,be,Q,Z] = spec(A,B)
rretorna ainda a matriz `Q` e `Z` de autovetores esquerdos e direitos generalizados do feixe.

# Referências

As computações de autovalores de matrizes são baseadas nas rotinas Lapack

- DGEEV e ZGEEV quando as matrizes não são simétricas,
- DSYEV e ZHEEV quando as matrizes são simétricas.

Uma matriz de complexos simétrica tem termos fora da diagonal conjugados e termos diagonais reais.

As computações de autovalores de feixes são baseadas nas rotinas Lapack DGGEV e ZGGEV.

# Matrizes de reais e de complexos

Deve-se notar que o tipo das variáveis de saída, tais como evals ou R por exemplo, não é necessariamente o mesmo das que das matrizes de entrada A e B. No parágrafo seguinte, análisamos o tipo das variáveis de saída no caso onde nos casos onde se computa os autovalores e autovetores de uma única matriz A.

- Matriz A de reais
  - Simétrica

    Os autovetores e autovalores são reais.
  - Não simétrica

    Os autovetores e autovalores são complexos.
- Matriz A de complexos
  - Simétrica

    Os autovalores são reais, mas os autovetores são complexos.
  - Não simétrica

    Os autovetores e autovalores são complexos.

# Exemplos

```
// AUTOVALORES DA MATRIZ
A=diag([1,2,3]);
X=rand(3,3);
A=inv(X)*A*X;
spec(A)
//
x=poly(0,'x');
pol=det(x*eye()-A)
roots(pol)
//
[S,X]=bdiag(A);
```

```
clean(inv(X)*A*X)

// AUTOVALORES DO FEIXE
A=rand(3,3);
[al,be,R] = spec(A,eye(A));
al./be
clean(inv(R)*A*R)  //exibindo os autovalores (matriz genérica)
A=A+%i*rand(A);
E=rand(A);
roots(det(A-%s*E))   //caso de complexos
```

# Ver Também

poly, det, schur, bdiag, colcomp

# Name

sqroot — fatoração hermitiana W*W'

```
sqroot(X)
```

# Parâmetros

X
: matriz simétrica, não-negativa definida de reais ou complexos

# Descrição

Retorna W tal que `X=W*W'` (usa SVD).

# Exemplos

```
X=rand(5,2)*rand(2,5);X=X*X';
W=sqroot(X)
norm(W*W'-X,1)
//
X=rand(5,2)+%i*rand(5,2);X=X*X';
W=sqroot(X)
norm(W*W'-X,1)
```

# Ver Também

chol, svd

# Name

squeeze — remoção de dimensões singletons

```
hypOut = squeeze(hypIn)
```

# Parâmetros

hypIn
: hipermatriz ou matriz do tipo constante.

hypOut
: hipermatriz ou matriz do tipo constante.

# Descrição

Remove dimensões singletons de uma hipermatriz, i.e., qualquer dimensão para a qual o tamanho é 1. Se a entrada é uma matriz, ela não é afetada.

# Ver Também

hypermat, hypermatrices

# Autores

Eric Dubois, Jean-Baptiste Silvy

# Name

sva — aproximação em valores singulares

```
[U,s,V]=sva(A,k)
[U,s,V]=sva(A,tol)
```

# Parâmetros

A
: matriz de reais ou complexos

k
: inteiro

tol
: número real não-negativo

# Descrição

Aproximação em valores singulares.

`[U,S,V]=sva(A,k)` com `k` inteiro >=1, retorna `U,S` e `V` tais que `B=U*S*V'` é a melhor aproximação L2 de `A` com posto(`B`)=`k`.

`[U,S,V]=sva(A,tol)` com `tol` real retorna `U,S` e `V` tais que `B=U*S*V'` e a norma-L2 de `A-B` é, no máximo, `tol`.

# Exemplos

```
A=rand(5,4)*rand(4,5);
[U,s,V]=sva(A,2);
B=U*s*V';
svd(A)
svd(B)
clean(svd(A-B))
```

# Ver Também

svd

# Name

svd — decomposição em valores singulares

```
s=svd(X)
[U,S,V]=svd(X)
[U,S,V]=svd(X,0) (obsolete)
[U,S,V]=svd(X,"e")
[U,S,V,rk]=svd(X [,tol])
```

# Parâmetros

X
: matriz de reais ou complexos

s
: vetor de reais (valores singulares)

S
: matriz diagonal de reais (valores singulares)

U,V
: matrizes quadradas ortogonais ou unitárias (vetores singulares)

tol
: número real

# Descrição

`[U,S,V] = svd(X)` produz uma matriz diagonal `S` , com dimensão igual a de `X` e com elementos da diagonal não-negativos em ordem decrescente, e matrizes unitárias `U` e `V` tais que `X = U*S*V'`.

`[U,S,V] = svd(X,0)` produz a decomposição com "economia de tamanho". Se `X` é m-por-n com m > n, então apenas as primeiras n colunas de `U` são computadas e `S` é n-por-n.

`s = svd(X)` por si mesmo retorna um vetor `s` contendo os valores singulares.

`[U,S,V,rk]=svd(X,tol)` fornece também `rk`, o posto numérico de `X` i.e. i.e. o número de valores singulares maiores que `tol`.

O valor default de `tol` é o mesmo que em `rank`.

# Exemplos

```
X=rand(4,2)*rand(2,4)
svd(X)
sqrt(spec(X*X'))
```

# Ver Também

rank, qr, colcomp, rowcomp, sva, spec

# Função Usada

Decomposições svd são baseadas nas rotinas Lapack DGESVD para matrizes de reais e ZGESVD no caso de matrizes de complexos.

# Name

sylv — equação de Sylvester

```
sylv(A,B,C,flag)
```

# Parâmetros

A,B,C
: três matrizes de reais de dimensões apropriadas

flag
: string (`'c'` ou `'d'`)

# Descrição

`X = sylv(A,B,C,'c')` computa `X`, solução da equação de "tempo contínuo" de Sylvester.

```
A*X+X*B=C
```

`X=sylv(A,B,C,'d')` computa `X`, solução da equação de "tempo discreto" de Sylvester.

```
A*X*B-X=C
```

# Exemplos

```
A=rand(4,4);C=rand(4,3);B=rand(3,3);
X = sylv(A,B,C,'c');
norm(A*X+X*B-C)
X=sylv(A,B,C,'d')
norm(A*X*B-X-C)
```

# Ver Também

lyap

# Name

trace — traço de uma matriz

```
trace(X)
```

# Parâmetros

X
: matriz de reais ou complexos, matriz de polinômios ou de razões de polinômios.

# Descrição

`trace(X)` é o traço da matriz `X`.

É o mesmo que `sum(diag(X))`.

# Exemplos

```
A=rand(3,3);
trace(A)-sum(spec(A))
```

# Ver Também

det

# Parte XXIV. Localização

# Name

dgettext — get text translated into the current locale and a specific domain domain.

```
msg=dgettext(domain, myString)
```

# Parameters

domain
: The name of the message domain

string
: the message to be translated

# Description

`dgettext` get the translation of a string to the current locale in a specified message domain.

# Examples

```
dgettext('scilab','Startup execution:')
```

# See Also

gettext

# Authors

Sylvestre Ledru

# Name

getdefaultlanguage — getdefaultlanguage() returns the default language used by Scilab.

```
getdefaultlanguage()
```

# Description

`getdefaultlanguage()` returns the default language used by Scilab. By default, this function should return en_US.

# Examples

```
getdefaultlanguage()
```

# See Also

setlanguage getlanguage

# Authors

Sylvestre Ledru

# Name

getlanguage — getlanguage() returns current language used by Scilab.

```
getlanguage()
```

# Description

`getlanguage()` returns current language used by Scilab.

# Examples

```
setlanguage('en_US')
getlanguage()
```

# See Also

setlanguage

# Authors

A.C.
Sylvestre Ledru

# Name

gettext — get text translated into the current locale and domain.

```
msg=gettext(myString)
```

# Parameters

string
: the message to be translated

# Description

`gettext` get the translation of a string to the current locale in the current domain.

# Examples

```
gettext('Startup execution:')
```

# See Also

dgettext

# Authors

Sylvestre Ledru

# Name

LANGUAGE — Variable defining the language (OBSOLETE)

# Description

LANGUAGE is obsolete. If you need LANGUAGE, add LANGUAGE=getlanguage();

# See Also

getlanguage

# Name

setdefaultlanguage — sets and saves the internal LANGUAGE value.

```
setdefaultlanguage(language)
```

# Parameters

language
: with language='fr', 'en', 'ru_RU', 'zh_TW', ...

# Description

`setdefaultlanguage(language)` changes current language and save this value in scilab.

You need to restart scilab, if you want to use menus.

setdefaultlanguage('') resets language to the system value.

setdefaultlanguage is used only Windows. On others operating systems , it returns always %f.

# Examples

```
setdefaultlanguage('en_US')

// restart scilab
getlanguage()
setdefaultlanguage('fr_FR')

// restart scilab
getlanguage()
setdefaultlanguage('')

// restart scilab
```

# See Also

getlanguage, setlanguage

# Authors

A.C.

# Name

setlanguage — Sets the internal LANGUAGE value.

```
setlanguage(language)
```

# Parameters

language
: with language='fr' or 'en', ...

# Description

`setlanguage(language)` changes current language in scilab.

# Examples

```
setlanguage('en_US')
getlanguage()
setlanguage('en')
getlanguage()
setlanguage('fr')
getlanguage()
setlanguage('fr_FR')
getlanguage()
```

# See Also

getlanguage

# Authors

A.C.

# Parte XXV. Otimização e Simulação

# Índice

2. Neldermead ..... 1432
fminsearch ..... 1433
neldermead ..... 1442
overview ..... 1463
nmplot ..... 1467
optimget ..... 1478
optimplotfunccount ..... 1480
optimplotfval ..... 1481
optimplotx ..... 1482
optimset ..... 1483
3. Optimization base ..... 1487
optimbase ..... 1488
4. Optimization simplex ..... 1502
optimsimplex ..... 1503

# Capítulo 2. Neldermead

# Nom

fminsearch — Computes the unconstrained minimimum of given function with the Nelder-Mead algorithm.

```
x = fminsearch ( costf , x0 )
x = fminsearch ( costf , x0 , options )
[x,fval,exitflag,output] = fminsearch ( costf , x0 , options )
```

# Description

This function searches for the unconstrained minimum of a given cost function.

The provided algorithm is a direct search algorithm, i.e. an algorithm which does not use the derivative of the cost function. It is based on the update of a simplex, which is a set of k>=n+1 vertices, where each vertex is associated with one point and one function value. This algorithm is the Nelder-Mead algorithm.

# Design

This function is based on a specialized use of the more general neldermead component. Users which want to have a more flexible solution based on direct search algorithms should consider using the neldermead component instead of the fminsearch function.

# Parameters

costf
: The cost function.

x0
: The initial guess.

options
: A struct which contains configurable options of the algorithm (see below for details).

x
: The minimum.

fval
: The minimum function value.

exitflag
: The flag associated with exist status of the algorithm.

  The following values are available.

  -1
  : The maximum number of iterations has been reached.

  0
  : The maximum number of function evaluations has been reached.

  1
  : The tolerance on the simplex size and function value delta has been reached. This signifies that the algorithm has converged, probably to a solution of the problem.

output
: A struct which stores detailed information about the exit of the algorithm. This struct contains the following fields.

output.algorithm
: A string containing the definition of the algorithm used, i.e. 'Nelder-Mead simplex direct search'.

output.funcCount
: The number of function evaluations.

output.iterations
: The number of iterations.

output.message
: A string containing a termination message.

# Options

In this section, we describe the options input argument which have an effect on the algorithm used by fminsearch.

The options input argument is a data structure which drives the behaviour of fminsearch. It allows to handle several options in a consistent and simple interface, without the problem of managing many input arguments.

These options must be set with the optimset function. See the optimset help for details of the options managed by this function.

The fminsearch function is sensitive to the following options.

options.MaxIter
: The maximum number of iterations. The default is 200 * n, where n is the number of variables.

options.MaxFunEvals
: The maximum number of evaluations of the cost function. The default is 200 * n, where n is the number of variables.

options.TolFun
: The absolute tolerance on function value. The default value is 1.e-4.

options.TolX
: The absolute tolerance on simplex size. The default value is 1.e-4.

options.Display
: The verbose level.

options.OutputFcn
: The output function, or a list of output functions. The default value is empty.

options.PlotFcns
: The plot function, or a list of plotput functions. The default value is empty.

# Termination criteria

In this section, we describe the termination criteria used by fminsearch.

The criteria is based on the following variables:

ssize
: the current simplex size,

shiftfv
: the absolute value of the difference of function value between the highest and lowest vertices.

If both the following conditions

```
ssize < options.TolX
```

and

```
shiftfv < options.TolFun
```

are true, then the iterations stop.

The size of the simplex is computed using the "sigmaplus" method of the optimsimplex component. The "sigmamplus" size is the maximum length of the vector from each vertex to the first vertex. It requires one loop over the vertices of the simplex.

# The initial simplex

The fminsearch algorithm uses a special initial simplex, which is an heuristic depending on the initial guess. The strategy chosen by fminsearch corresponds to the -simplex0method flag of the neldermead component, with the "pfeffer" method. It is associated with the -simplex0deltausual = 0.05 and -simplex0deltazero = 0.0075 parameters. Pfeffer's method is an heuristic which is presented in "Global Optimization Of Lennard-Jones Atomic Clusters" by Ellen Fan. It is due to L. Pfeffer at Stanford. See in the help of optimsimplex for more details.

# The number of iterations

In this section, we present the default values for the number of iterations in fminsearch.

The *options* input argument is an optionnal data structure which can contain the *options.MaxIter* field. It stores the maximum number of iterations. The default value is 200n, where n is the number of variables. The factor 200 has not been chosen by chance, but is the result of experiments performed against quadratic functions with increasing space dimension.

This result is presented in "Effect of dimensionality on the nelder-mead simplex method" by Lixing Han and Michael Neumann. This paper is based on Lixing Han's PhD, "Algorithms in Unconstrained Optimization". The study is based on numerical experiment with a quadratic function where the number of terms depends on the dimension of the space (i.e. the number of variables). Their study shows that the number of iterations required to reach the tolerance criteria is roughly 100n. Most iterations are based on inside contractions. Since each step of the Nelder-Mead algorithm only require one or two function evaluations, the number of required function evaluations in this experiment is also roughly 100n.

# Output and plot functions

The optimset function can be used to configure one or more output and plot functions.

The output or plot function is expected to have the following definition:

```
function myfun ( x , optimValues , state )
```

The input arguments `x`, `optimValues` and `state` are described in detail in the optimset help page. The `optimValues.procedure` field represent the type of step used during the current iteration and can be equal to one of the following strings

- "" (the empty string),
- "initial simplex",
- "expand",
- "reflect",
- "contract inside",
- "contract outside".

# Example

In the following example, we use the fminsearch function to compute the minimum of the Rosenbrock function. We first define the function "banana", and then use the fminsearch function to search the minimum, starting with the initial guess [-1.2 1.0]. In this particular case, 85 iterations are performed and 159 function evaluations are

```
function y = banana (x)
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
[x, fval, exitflag, output] = fminsearch ( banana , [-1.2 1] );
x
fval
exitflag
output
```

# Example with customized options

In the following example, we configure the absolute tolerance on the size of the simplex to a larger value, so that the algorithm performs less iterations. Since the default value of "TolX" for the fminsearch function is 1.e-4, we decide to use 1.e-2. The optimset function is used to create an optimization data structure and the field associated with the string "TolX" is set to 1.e-2. The `opt` data structure is then passed to the `fminsearch` function as the third input argument. In this particular case, the number of iterations is 70 with 130 function evaluations.

```
function y = banana (x)
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "TolX" , 1.e-2 );
[x , fval , exitflag , output] = fminsearch ( banana , [-1.2 1] , opt );
x
fval
exitflag
output
```

# Example with a pre-defined plot function

In the following example, we want to produce a graphic of the progression of the algorithm, so that we can include that graphic into a report without defining a customized plot function. The `fminsearch` function comes with the following 3 pre-defined functions :

- optimplotfval, which plots the function value,
- optimplotx, which plots the current point `x`,
- optimplotfunccount, which plots the number of function evaluations.

In the following example, we use the three pre-defined functions in order to create one graphic, representing the function value depending on the number of iterations.

```
function y = banana (x)
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "PlotFcns" , optimplotfval );
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

In the previous example, we could replace the "optimplotfval" function with "optimplotx" or "optimplotfunccount" and obtain different results. In fact, we can get all the figures at the same time, by setting the "PlotFcns" to a list of functions, as in the following example.

```
function y = banana (x)
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
myfunctions = list ( optimplotfval , optimplotx , optimplotfunccount );
opt = optimset ( "PlotFcns" , myfunctions );
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

# Example with an customized output function

In the following example, we want to produce intermediate outputs of the algorithm. We define the `outfun` function, which takes the current point `x` as input argument. The function plots the current point into the current graphic window with the `plot` function. We use the "OutputFcn" feature of the optimset function and set it to the output function. Then the option data structure is passed to the `fminsearch` function. At each iteration, the output function is called back, which creates and update an interactive plot. While this example creates a 2D plot, the user may customized the output function so that it writes a message in the console, write some data into a data file, etc... The user can distinguish between the output function (associated with the "OutputFcn" option) and the plot function (associated with the "PlotFcns" option). See the optimset for more details on this feature.

```
function y = banana (x)
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
function outfun ( x , optimValues , state )
```

```
  plot( x(1),x(2),'.');
endfunction
opt = optimset ( "OutputFcn" , outfun);
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

# Example with customized "Display" option

The "Display" option allows to get some input about the intermediate steps of the algorithm as well as to be warned in case of a convergence problem.

In the following example, we present what happens in case of a convergence problem. We set the number of iterations to 10, instead of the default 400 iterations. We know that 85 iterations are required to reach the convergence criteria. Therefore, the convergence criteria is not met and the maximum number of iterations is reached.

```
function y = banana (x)
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "MaxIter" , 10 );
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

Since the default value of the "Display" option is "notify", a message is generated, which warns the user about a possible convergence problem. The previous script produces the following output.

```
Exiting: Maximum number of iterations has been exceeded
         - increase MaxIter option.
         Current function value: 4.1355598
```

Notice that if the "Display" option is now set to "off", no message is displayed at all. Therefore, the user should be warned that turning the Display "off" should be used at your own risk...

In the following example, we present how to display intermediate steps used by the algorithm. We simply set the "Display" option to the "iter" value.

```
ffunction y = banana (x)
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "Display" , "iter" );
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

The previous script produces the following output. It allows to see the number of function evaluations, the minimum function value and which type of simplex step is used for the iteration.

```
 Iteration   Func-count     min f(x)         Procedure
     0            3             24.2
     1            3            20.05         initial simplex
```

```
    2            5         5.161796         expand
    3            7         4.497796         reflect
    4            9         4.497796         contract outside
    5           11        4.3813601         contract inside
    etc...
```

# Example with customized output option

In this section, we present an example where all the fields from the `optimValues` data structure are used to print a message at each iteration.

```
function outfun ( x , optimValues , state )
  fc = optimValues.funccount;
  fv = optimValues.fval;
  it = optimValues.iteration;
  pr = optimValues.procedure;
  mprintf ( "%d %e %d -%s-\n" , fc , fv , it , pr )
endfunction
opt = optimset ( "OutputFcn" , outfun );
[x fval] = fminsearch ( banana , [-1.2 1] , opt );
```

The previous script produces the following output.

```
3 2.420000e+001 0 --
3 2.005000e+001 1 -initial simplex-
5 5.161796e+000 2 -expand-
7 4.497796e+000 3 -reflect-
9 4.497796e+000 4 -contract outside-
11 4.381360e+000 5 -contract inside-
13 4.245273e+000 6 -contract inside-
[...]
157 1.107549e-009 84 -contract outside-
159 8.177661e-010 85 -contract inside-
159 8.177661e-010 85 --
```

# Some advices

In this section, we present some practical advices with respect to the Nelder-Mead method. A deeper analysis is provided in the bibliography at the end of this help page, as well as in the "Nelder-Mead User's Manual" provided on Scilab's Wiki. The following is a quick list of tips to overcome problems that may happen with this algorithm.

- We should consider the optim function before considering the `fminsearch` function. Because optim uses the gradient of the function and uses this information to guess the local curvature of the cost function, the number of iterations and function evaluations is (much) lower with optim, when the function is sufficiently smooth. If the derivatives of the function are not available, it is still possible to use numerical derivatives combined with the optim function (this feature is provided by the derivative and numdiff functions). If the function has discontinuous derivatives, the optim function provides the `nd` solver which is very efficient. Still, there are situations where the cost function is discontinuous or "noisy". In these situations, the `fminsearch` function can perform well.

- We should not expected a fast convergence with many parameters, i.e. more that 10 to 20 parameters. It is known that the efficiency of this algorithm decreases rapidly when the number of parameters increases.
- The default tolerances are set to pretty loose values. We should not reduce the tolerances in the goal of getting very accurate results. Because the convergence rate of Nelder-Mead's algorithm is low (at most linear), getting a very accurate solution will require a large number of iterations. Instead, we can most of the time expect a "good reduction" of the cost function with this algorithm.
- Although the algorithm practically converges in many situations, the Nelder-Mead algorithm is not a provably convergent algorithm. There are several known counter-examples where the algorithm fails to converge on a stationnary point and, instead, converge to a non-stationnary point. This situation is often indicated by a repeated application of the contraction step. In that situation, we simply restart the algorithm with the final point as the new initial guess. If the algorithm converges to the same point, there is a good chance that this point is a "good" solution.
- Taking into account for bounds constraints or non-linear inequality constraints can be done by penalization methods, i.e. setting the function value to a high value when the constraints are not satisfied. While this approach works in some situations, it may also fail. In this case, users might be interested in Box's complex algorithm, provided by Scilab in the `neldermead` component. If the problem is really serious, Box's complex algorithm will also fail and a more powerful solver is necessary.

# Bibliography

"Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation", Spendley, W. and Hext, G. R. and Himsworth, F. R., American Statistical Association and American Society for Quality, 1962

"A Simplex Method for Function Minimization", Nelder, J. A. and Mead, R., The Computer Journal, 1965

"Iterative Methods for Optimization", C. T. Kelley, SIAM Frontiers in Applied Mathematics, 1999

"Algorithm AS47 - Function minimization using a simplex procedure", O'Neill, R., Applied Statistics, 1971

"Effect of dimensionality on the nelder-mead simplex method", Lixing Han and Michael Neumann, Optimization Methods and Software, 21, 1, 1--16, 2006.

"Algorithms in Unconstrained Optimization", Lixing Han, Ph.D., The University of Connecticut, 2000.

"Global Optimization Of Lennard-Jones Atomic Clusters" Ellen Fan, Thesis, February 26, 2002, McMaster University

# TODO

- implement the 'interrupt' state of the algorithm
- add a demo with an interactive output function, which draws the plot during the optimization.
- implement the stop of an optimization via the stop output argument of the output function

# Authors

Michael Baudin - INRIA - 2008-2009

Michael Baudin - Digiteo - 2009

# Acknowledgements

Michael Baudin would like to thank Lixing Han, who kindly sent his PhD thesis.

# See Also

optim, neldermead, optimset, optimget, optimplotfval, optimplotx, optimplotfunccount

# Nom

neldermead — Provides several direct search optimization algorithms based on the simplex method.

```
newobj = neldermead_new ()
this = neldermead_destroy (this)
this = neldermead_configure (this,key,value)
value = neldermead_cget (this,key)
neldermead_display ( this )
value = neldermead_get ( this , key )
this = neldermead_search ( this )
this = neldermead_restart ( this )
[ this , result ] = neldermead_function ( this , x )
```

# Description

This class provides several direct search optimization algorithms based on the simplex method.

The optimization problem to solve is the minimization of a cost function, with bounds and nonlinear constraints

```
min f(x)
l_i <= x_i <= h_i, i = 1,n
g_i(x) <= 0, i = 1,nbineq
```

where

n
: number of variables

nbineq
: number of inequality constraints

The provided algorithms are direct search algorithms, i.e. algorithms which do not use the derivative of the cost function. They are based on the update of a simplex, which is a set of k>=n+1 vertices, where each vertex is associated with one point and one function value.

The following algorithms are available :

Spendley, Hext and Himsworth fixed size simplex method
: This algorithm solves an unconstrained optimization problem with a fixed sized simplex made of k=n+1 vertices.

Nelder and Mead variable size simplex method
: This algorithm solves an unconstrained optimization problem with a variable sized simplex made of k=n+1 vertices.

Box complex method
: This algorithm solves an constrained optimization problem with a variable sized simplex made of an arbitrary k number of vertices (k=2n is recommended by Box).

# Design

The neldermead component is built on top of the optimbase and optimsimplex components.

# Functions

The following functions are available.

newobj = neldermead_new ()
Creates a new neldermead object.

newobj
The new object.

this = neldermead_destroy (this)
Destroy the given object.

this
The current object.

this = neldermead_configure (this,key,value)
Configure the current object with the given value for the given key.

this
The current object.

key
the key to configure. The following keys are available.

-verbose
set to 1 to enable verbose logging. (default is 0)

-verbosetermination
set to 1 to enable verbose termination logging. (default is 0)

-x0
the initial guess, as a n x 1 column vector, where n is the number of variables.

-maxfunevals
the maximum number of function evalutations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxfuneval".

-maxiter
the maximum number of iterations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxiter".

-tolfunabsolute
the absolute tolerance for the function value (default is 0.0).

-tolfunrelative
the relative tolerance for the function value (default is %eps).

-tolfunmethod
the method used for the tolerance on function value in the termination criteria.

The following values are available : %t, %f (default is %f). If this criteria is triggered, the status of the optimization is set to "tolf".

-tolxabsolute
the absolute tolerance on x (default is 0.0).

-tolxrelative
the relative tolerance on x (default is %eps).

-tolxmethod
the method used for the tolerance on x in the termination criteria.

The following values are available : %t, %f (default is %t). If this criteria is triggered, the status of the optimization is set to "tolx".

-function
the objective function, which computes the value of the cost and the non linear constraints, if any.

See below for the details of the communication between the optimization system and the cost function.

-costfargument
an additionnal argument, passed to the cost function.

-outputcommand
a command which is called back for output.

See below for the details of the communication between the optimization system and the output command function.

-outputcommandarg
an additionnal argument, passed to the output command.

-numberofvariables
the number of variables to optimize (default is 0).

-storehistory
set to 1 to enable the history storing (default is 0).

-boundsmin
the minimum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-boundsmax
the maximum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-nbineqconst
the number of inequality constraints (default is 0)

-method
the name of the algorithm to use. The following methods are available :

"fixed"
the Spendley et al. fixed simplex shape algorithm. This algorithm is for unconstrained problems (i.e. bounds and non linear constraints are not taken into account)

"variable"
the Nelder-Mead variable simplex shape algorithm. This algorithm is for unconstrained problems (i.e. bounds and non linear constraints are not taken into account)

"box"
the Box complex algorithm. This algorithm takes into account bounds and nonlinear inequality constraints.

"mine"
the user-defined algorithm, associated with the `-mymethod`option. See below for details.

-simplex0method
the method to use to compute the initial simplex. The first vertex in the simplex is always the initial guess associated with the -x0 option. The following methods are available :

"given"
the coordinates associated with the -coords0 option are used to compute the initial simplex, with arbitrary number of vertices.

This allow the user to setup the initial simplex by a specific method which is not provided by the current component (for example with a simplex computed from a design of experiments). This allows also to configure the initial simplex so that a specific behaviour of the algorithm an be reproduced (for example the Mac Kinnon test case).

The given matrix is expected to have n rows and k columns, where n is the dimension of the problem and k is the number of vertices.

"axes"
the simplex is computed from the coordinate axes and the length associated with the -simplex0length option.

"spendley"
the simplex is computed so that it is regular with the length associated with the -simplex0length option (i.e. all the edges have the same length).

"pfeffer"
the simplex is computed from an heuristic, in the neighborhood of the initial guess. This initial simplex depends on the -simplex0deltausual and -simplex0deltazero.

"randbounds"
the simplex is computed from the bounds and a random number. This option is available only if bounds are available : if bounds are not available, an error is generated. This method is usually associated with Box's algorithm. The number of vertices in the simplex is taken from the -boxnbpoints option.

-coords0
the coordinates of the vertices of the initial simplex. If the -simplex0method option is set to "given", these coordinates are used to compute the initial simplex. This matrix is expected to have shape nbve x n where nbve is the number of vertices and n is the number of variables.

-simplex0length
the length to use when the initial simplex is computed with the "axes" or "spendley" methods. If the initial simplex is computed from "spendley" method, the length is expected to be a scalar value. If the initial simplex is computed from "axes" method, it may be either a scalar value or a vector of values, with rank n, where n is the number of variables.

-simplex0deltausual
the relative delta for non-zero parameters in "pfeffer" method. The default value is 0.05.

-simplex0deltazero
the absolute delta for non-zero parameters in "pfeffer" method. The default value is 0.0075.

-rho
the reflection coefficient. This parameter is used when the -method option is set to "fixed" or "variable". The default value is 1.0.

-chi
the expansion coefficient. This parameter is used when the -method option is set to "variable". The default value is 2.0.

-gamma
the contraction coefficient. This parameter is used when the -method option is set to "variable". The default value is 0.5.

-sigma
: the shrinkage coefficient. This parameter is used when the -method option is set to "fixed" or "variable". The default value is 0.5.

-tolsimplexizemethod
: set to %f to disable the tolerance on the simplex size. The default value is %t. If this criteria is triggered, the status of the optimization is set to "tolsize".

  When this criteria is enabled, the values of the options -tolsimplexizeabsolute and -tolsimplexizerelative are used in the termination criteria. The method to compute the size is the "sigmaplus" method.

-tolsimplexizeabsolute
: the absolute tolerance on the simplex size. The default value is 0.0.

-tolsimplexizerelative
: the relative tolerance on the simplex size. The default value is %eps.

-tolssizedeltafvmethod
: set to %t to enable the termination criteria based on the size of the simplex and the difference of function value in the simplex. The default value is %f. If this criteria is triggered, the status of the optimization is set to "tolsizedeltafv".

  This termination criteria uses the values of the options -tolsimplexizeabsolute and -toldeltafv. This criteria is identical to Matlab's fminsearch.

-toldeltafv
: the absolute tolerance on the difference between the highest and the lowest function values.

-tolvarianceflag
: set to %t to enable the termination criteria based on the variance of the function value. If this criteria is triggered, the status of the optimization is set to "tolvariance".

  This criteria is suggested by Nelder and Mead.

-tolabsolutevariance
: the absolute tolerance on the variance of the function values of the simplex.

-tolrelativevariance
: the relative tolerance on the variance of the function values of the simplex.

-kelleystagnationflag
: set to %t to enable the termination criteria using Kelley's stagnation detection, based on sufficient decrease condition. The default value is %f. If this criteria is triggered, the status of the optimization is set to "kelleystagnation".

-kelleynormalizationflag
: set to %f to disable the normalization of the alpha coefficient in Kelley's stagnation detection, i.e. use the value of the option -kelleystagnationalpha0 as is. Default value is %t, i.e. the simplex gradient of the initial simplex is taken into account in the stagnation detection.

-kelleystagnationalpha0
: the parameter used in Kelley's stagnation detection. The default value is 1.e-4.

-restartflag
: set to %t to enable the automatic restart of the algorithm. Default value is %f.

-restartdetection
: the method to detect if the automatic restart must be performed. The following methods are available :

"oneill"
the factorial local optimality test by O'Neill is used. If the test finds a local point which is better than the computed optimum, a restart is performed.

"kelley"
the sufficient decrease condition by O'Neill is used. If the test finds that the status of the optimization is "kelleystagnation", a restart is performed. This status may be generated if the -kelleystagnationflag option is set to %t.

The default method is "oneill".

-restartmax
the maximum number of restarts, when automatic restart is enabled via the -restartflag option. Default value is 3.

-restarteps
the absolute epsilon value used to check for optimality in the factorial O'Neill restart detection. The default value is %eps.

-restartstep
the absolute step length used to check for optimality in the factorial O'Neill restart detection. The default value is 1.0.

-restartsimplexmethod
the method to compute the initial simplex after a restart. The following methods are available.

"given"
the coordinates associated with the -coords0 option are used to compute the initial simplex, with arbitrary number of vertices.

This allow the user to setup the initial simplex by a specific method which is not provided by the current component (for example with a simplex computed from a design of experiments). This allows also to configure the initial simplex so that a specific behaviour of the algorithm an be reproduced (for example the Mc Kinnon test case).

The given matrix is expected to have n rows and k columns, where n is the dimension of the problem and k is the number of vertices.

"axes"
the simplex is computed from the coordinate axes and the length associated with the -simplex0length option.

"spendley"
the simplex is computed so that it is regular with the length associated with the -simplex0length option (i.e. all the edges have the same length).

"pfeffer"
the simplex is computed from an heuristic, in the neighborhood of the initial guess. This initial simplex depends on the -simplex0deltausual and -simplex0deltazero.

"randbounds"
the simplex is computed from the bounds and a random number. This option is available only if bounds are available : if bounds are not available, an error is generated. This method is usually associated with Box's algorithm. The number of vertices in the simplex is taken from the -boxnbpoints option.

"oriented"
the simplex is computed so that it is oriented, as suggested by C.T. Kelley.

The default method is "oriented".

-scalingsimplex0
the algorithm used to scale the initial simplex into the nonlinear constraints. The following two algorithms are provided :

- "tox0": scales the vertices toward the initial guess.
- "tocentroid": scales the vertices toward the centroid, as recommended by Box.

If the centroid happens to be unfeasible, because the constraints are not convex, the scaling of the initial simplex toward the centroid may fail. Since the initial guess is always feasible, scaling toward the initial guess cannot fail. The default value is "tox0".

-boxnbpoints
the number of points in the initial simplex, when the -simplex0method is set to `"randbounds"`. The value of this option is also use to update the simplex when a restart is performed and the `-restartsimplexmethod` option is set to `"randbounds"`. The default value is so that the number of points is twice the number of variables of the problem.

-boxineqscaling
the scaling coefficient used to scale the trial point for function improvement or into the constraints of Box's algorithm. Default value is 0.5.

-guinalphamin
the minimum value of alpha when scaling the vertices of the simplex into nonlinear constraints in Box's algorithm. Default value is 1.e-5.

-boxreflect
the reflection factor in Box's algorithm. Default value is 1.3.

-boxtermination
set to %t to enable Box termination criteria. Default value is %f.

-boxtolf
the absolute tolerance on difference of function values in the simplex, suggested by Box. This tolerance is used if the -boxtermination option is set to %t. Default value is 1.e-5.

-boxnbmatch
the number of consecutive match of Box termination criteria. Default value is 5.

-boxboundsalpha
the parameter used to project the vertices into the bounds in Box's algorithm. Default value is 0.000001.

-mymethod
a user-derined simplex algorithm. See below for details (default is empty).

-myterminate
a user-defined terminate function. See below for details (default is empty).

-myterminateflag
set to %t to enable the user-defined terminate function (default is %f).

value
the value.

value = neldermead_cget (this,key)
Get the value for the given key. If the key is unknown, generates an error.

this
The current object.

key
the name of the key to quiery. The list of available keys is the same as for the neldermead_configure function.

value = neldermead_get ( this , key )
Get the value for the given key. If the key is unknown, generates an error.

this
The current object.

key
the key to get.

The following keys are available :

-funevals
the number of function evaluations

-iterations
the number of iterations

-xopt
the x optimum, as a n x 1 column vector, where n is the number of variables.

-fopt
the optimum cost function value

-historyxopt
an array, with nbiter values, containing the history of x during the iterations.

This array is available after optimization if the history storing was enabled with the -storehistory option.

-historyfopt
an array, with nbiter values, containing the history of the function value during the iterations.

This array is available after optimization if the history storing was enabled with the -storehistory option.

-fx0
the function value for the initial guess

-status
a string containing the status of the optimization. See below for details about the optimization status.

-historysimplex
a matrix containing the history of the simplex during the iterations. This matrix has rank nbiter x nbve x n, where nbiter is the number of iterations, nbve is the number of vertices in the simplex and n is the number of variables.

-simplex0
the initial simplex. This is a simplex object, which is suitable for processing with the optimsimplex component.

-simplexopt
the optimum simplex. This is a simplex object, which is suitable for processing with the optimsimplex component.

-restartnb
the number of actual restarts performed.

Most fields are available only after an optimization has been performed with one call to the neldermead_search method.

neldermead_display ( this )
Display the current settings in the console.

this
The current object.

this = neldermead_search ( this )
Performs the optimization associated with the method associated with the -method option and find the optimum.

this
The current object.

If the -restartflag option is enabled, automatic restarts are performed, based on the -restartdetection option.

this = neldermead_restart ( this )
Restarts the optimization by updating the simplex and performing a new search.

this
The current object.

[ this , result ] = neldermead_function ( this , x )
Call the cost function and return the value.

this
The current object.

x
the point where the function is to be evaluated

index
optionnal, a flag to pass to the cost function (default = 1). See the section "The cost function" for available values of index.

# The cost function

The option `-function` allows to configure the cost function. The cost function is used to compute the objective function value `f`. If the `-nbineqconst` option is configured to a non-zero value, the cost function must also compute the value of the nonlinear, positive, inequality constraints `c`. The cost function can also take as input/output an additional argument, if the `-costfargument` option is configured. Depending of these options, the cost function can have one of the following headers :

```
//  Without additionnal data
   [ f , index ] = costf ( x , index )
   [ f , c , index ] = costf ( x , index )
//  With -costfargument
   [ f , index , data ] = costf ( x , index , data )
   [ f , c , index , data ] = costf ( x , index , data )
```

where

x
the current point, as a column vector

index
optional, an integer representing the value to compute

f
: the value of the cost function

c
: the value of the non-linear, positive, inequality constraints

data
: an user-provided input/output argument

The index input parameter tells to the cost function what is expected in the output arguments. It has the following meaning

index = 2
: compute `f`

index = 5
: compute `c`

index = 6
: compute `f` and `c`

In the most simplex case, there is no additionnal cost function argument and no nonlinear constraints. In this case, the cost function is expected to have the following header

```
function [ f , index ]= myfunction ( x , index )
```

The `data` argument is both input and output and the following paragraph explains for what reason and how it works.

This feature may be used in the situation where the cost function has to update its environment from call to call. Its simplest use is to count the number of calls to the cost function, but this feature is already available directly. Consider the more practical situation where the optimization requires the execution of an underlying Newton method (a chemical solver for example). This Newton method requires an initial guess x0. If the initial guess for this underlying Newton method is kept constant, the Newton method may have problems to converge when the current optimization point get far away from the its initial point. If the `-costfargument` option is provided, the initial guess for the Newton method can be updated so that it gets the value of the previous call. This way, the Newton method will have less problems to converge and the cost function evaluation may be faster.

We now present how the feature works. Everytime the cost function is called back, the `-costfargument` option is passed to the cost function as an input argument. If the cost function modifies its content in the output argument, the content of the `-costfargument` option is updated accordingly. Once the optimization is performed, the user may call the `neldermead_cget` function and get back an updated `-costfargument` content.

# The output function

The option -outputcommand allows to configure a command which is called back at the start of the optimization, at each iteration and at the end of the optimization.

The output function must have the following header

```
function outputcmd(state, data, myobj)
```

where

state
: a string representing the current state of the algorithm. Available values are "init", "iter", "done".

data
: a tlist containing at least the following entries

  x
  : the current optimum

  fval
  : the current function value

  iteration
  : the current iteration index

  funccount
  : the number of function evaluations

  simplex
  : the current simplex

  step
  : the previous step in the algorithm. The following values are available : "init", "done", "reflection", "expansion", "insidecontraction", "outsidecontraction", "reflectionnext", "shrink".

myobj
: a user-defined parameter.

  This input parameter is defined with the -outputcommandarg option.

The output function may be used when debugging the specialized optimization algorithm, so that a verbose logging is produced. It may also be used to write one or several report files in a specialized format (ASCII, LaTeX, Excel, Hdf5, etc...). The user-defined parameter may be used in that case to store file names or logging options.

The data tlist argument may contain more fields than the current presented ones. These additionnal fields may contain values which are specific to the specialized algorithm, such as the simplex in a Nelder-Mead method, the gradient of the cost function in a BFGS method, etc...

# Termination

The current component takes into account for several generic termination criterias.

The following termination criterias are enabled by default :

- -maxiter,
- -maxfunevals,
- -tolxmethod.
- -tolsimplexizemethod.

The optimization_terminate function uses a set of rules to compute if the termination occurs, which leads to an optimization status which is equal to one of the following : "continue", "maxiter", "maxfunevals", "tolf", "tolx", "tolsize", "tolsizedeltafv", "kelleystagnation", "tolboxf", "tolvariance". The value of the status may also be a user-defined string, in the case where a user-defined termination function has been set.

The following set of rules is examined in this order.

- By default, the status is `"continue"` and the `terminate` flag is %f.
- The number of iterations is examined and compared to the `-maxiter` option : if the following condition

```
iterations >= maxiter
```

is true, then the status is set to "maxiter" and terminate is set to %t.

- The number of function evaluations and compared to the `-maxfunevals` option is examined : if the following condition

```
funevals >= maxfunevals
```

is true, then the status is set to `"maxfuneval"` and `terminate` is set to %t.

- The tolerance on function value is examined depending on the value of the `-tolfunmethod`.

%f
: then the criteria is just ignored.

%t
: if the following condition

```
abs(currentfopt) < tolfunrelative * abs(previousfopt) + tolfunabsolute
```

is true, then the status is set to "tolf" and terminate is set to %t.

The relative termination criteria on the function value works well if the function value at optimum is near zero. In that case, the function value at initial guess fx0 may be used as `previousfopt`.

This criteria is sensitive to the `-tolfunrelative` and `-tolfunabsolute` options.

The absolute termination criteria on the function value works if the user has an accurate idea of the optimum function value.

- The tolerance on x is examined depending on the value of the -tolxmethod.

%f
: then the criteria is just ignored.

%t
: if the following condition

```
norm(currentxopt - previousxopt) < tolxrelative * norm(currentxopt) + tolxa
```

is true, then the status is set to `"tolx"` and `terminate` is set to %t.

This criteria is sensitive to the `-tolxrelative` and `-tolxabsolute` options.

The relative termination criteria on x works well if x at optimum is different from zero. In that case, the condition measures the distance between two iterates.

The absolute termination criteria on x works if the user has an accurate idea of the scale of the optimum x. If the optimum x is near 0, the relative tolerance will not work and the absolute tolerance is more appropriate.

- The tolerance on simplex size is examined depending on the value of the `-tolsimplexize-method` option.

%f
then the criteria is just ignored.

%t
if the following condition

```
ssize < tolsimplexizerelative * simplexsize0 + tolsimplexizeabsolute
```

is true where `simplexsize0` is the size of the simplex at iteration 0, then the `status` is set to `"tolsize"` and `terminate` is set to %t.

The size of the simplex is computed from the "sigmaplus" method of the `optimsimplex` component. This criteria is sensitive to the `-tolsimplexizeabsolute` and the `-tolsimplexizerelative` options.

- The absolute tolerance on simplex size and absolute difference of function value is examined depending on the value of the -tolssizedeltafvmethod option.

  %f
  then the criteria is just ignored.

  %t
  if both the following conditions

  ```
  ssize < tolsimplexizeabsolute
  ```

  ```
  shiftfv < toldeltafv
  ```

  is true where `ssize` is the current simplex size and `shiftfv` is the absolute value of the difference of function value between the highest and lowest vertices, then the status is set to `"tolsizedeltafv"` and `terminate` is set to %t.

- The stagnation condition based on Kelley sufficient decrease condition is examined depending on the value of the `-kelleystagnationflag` option.

  %f
  then the criteria is just ignored.

  %t
  if the following condition

  ```
  newfvmean <= oldfvmean - alpha * sg' * sg
  ```

  is true where `newfvmean` (resp. `oldfvmean`) is the function value average in the current iteration (resp. in the previous iteration), then the status is set to "kelleystagnation" and terminate is set to %t. Here, `alpha` is a non-dimensional coefficient and `sg` is the simplex gradient.

- The termination condition suggested by Box is examined depending on the value of the -boxtermination option.

  %f
  then the criteria is just ignored.

%t
if both the following conditions

```
shiftfv < boxtolf
```

```
boxkount == boxnbmatch
```

is true where `shiftfv` is the difference of function value between the best and worst vertices, and `boxkount` is the number of consecutive iterations where this criteria is met, then the status is set to "tolboxf" and terminate is set to %t. Here, the `boxtolf` parameter is the value associated with the `-boxtolf` option and is a user-defined absolute tolerance on the function value. The `boxnbmatch` parameter is the value associated with the `-boxnbmatch` option and is the user-defined number of consecutive match.

- The termination condition based on the variance of the function values in the simplex is examined depending on the value of the `-tolvarianceflag` option.

%f
then the criteria is just ignored.

%t
if the following condition

```
var < tolrelativevariance * variancesimplex0 + tolabsolutevariance
```

is true where `var` is the variance of the function values in the simplex, then the status is set to "tolvariance" and terminate is set to %t. Here, the `tolrelativevariance` parameter is the value associated with the `-tolrelativevariance` option and is a user-defined relative tolerance on the variance of the function values. The `tolabsolutevariance` parameter is the value associated with the `-tolabsolutevariance` option and is the user-defined absolute tolerance of the variance of the function values.

- The user-defined termination condition is examined depending on the value of the `-mytermina-teflag` option.

%f
then the criteria is just ignored.

%t
if the `term` output argument boolean returned by the termination function is true, then the status is set to the user-defined status and `terminate` is set to %t.

# Kelley's stagnation detection

The stagnation detection criteria suggested by Kelley is based on a sufficient decrease condition, which requires a parameter alpha > 0 to be defined. The -kelleynormalizationflag option allows to configure the method to use to compute this alpha parameter : two methods are available, where each method corresponds to a different paper by Kelley :

constant
In "Detection and Remediation of Stagnation in the Nelder--Mead Algorithm Using a Sufficient Decrease Condition", Kelley uses a constant alpha, with the suggested value 1.e-4, which is is typical choice for line search method.

normalized
in "Iterative Methods for Optimization", Kelley uses a normalized alpha, computed from the following formula

```
alpha = alpha0 * sigma0 / nsg
```

where sigma0 is the size of the initial simplex and nsg is the norm of the simplex gradient for the initial guess point.

# O'Neil factorial optimality test

In "Algorithm AS47 - Function minimization using a simplex procedure", R. O'Neil presents a fortran 77 implementation of the simplex method. A factorial test is used to check if the computed optimum point is a local minimum. If the -restartdetection option is set to "oneill", that factorial test is used to see if a restart should be performed.

# Spendley et al. implementation notes

The original paper may be implemented with several variations, which might lead to different results. This section defines what algorithmic choices have been used.

The paper states the following rules.

- "Rule 1. Ascertain the lowest reading y, of yi ... yk+1 Complete a new simplex Sp by excluding the point Vp corresponding to y, and replacing it by V* defined as above."
- "Rule 2. If a result has occurred in (k + 1) successive simplexes, and is not then eliminated by application of Rule 1, do not move in the direction indicated by Rule 1, or at all, but discard the result and replace it by a new observation at the same point."
- "Rule 3. If y is the lowest reading in So , and if the next observation made, y* , is the lowest reading in the new simplex S , do not apply Rule 1 and return to So from Sp . Move out of S, by rejecting the second lowest reading (which is also the second lowest reading in So)."

We implement the following "rules" of the Spendley et al. method.

- Rule 1 is strictly applied, but the reflection is done by reflection the high point, since we minimize a function instead of maximizing it, like Spendley.
- Rule 2 is NOT implemented, as we expect that the function evaluation is not subject to errors.
- Rule 3 is applied, ie reflection with respect to next to high point.

The original paper does not mention any shrink step. When the original algorithm cannot improve the function value with reflection steps, the basic algorithm stops. In order to make the current implementation of practical value, a shrink step is included, with shrinkage factor sigma. This perfectly fits into to the spirit of the original paper. Notice that the shrink step make the rule #3 (reflection with respect to next-to-worst vertex) unnecessary. Indeed, the minimum required steps are the reflection and shrinkage. Never the less, the rule #3 has been kept in order to make the algorithm as close as it can be to the original.

# Nelder-Mead implementation notes

The purpose of this section is to analyse the current implementation of Nelder-Mead's algorithm.

The algorithm that we use is described in "Iterative Methods for Optimization" by C. T. Kelley.

The original paper uses a "greedy" expansion, in which the expansion point is accepted whatever its function value. The current implementation, as most implementations, uses the expansion point only if it improves over the reflection point, that is,

- if fe<fr, then the expansion point is accepted,
- if not, the reflection point is accepted.

The termination criteria suggested by Nelder and Mead is based on an absolute tolerance on the standard deviation of the function values in the simplex. We provide this original termination criteria with the `-tolvarianceflag` option, which is disabled by default.

# Box's complex algorithm implementation notes

In this section, we analyse the current implementation of Box's complex method.

The initial simplex can be computed as in Box's paper, but this may not be safe. In his paper, Box suggest that if a vertex of the initial simplex does not satisfy the non linear constraints, then it should be "moved halfway toward the centroid of those points already selected". This behaviour is available when the `-scalingsimplex0` option is set to `"tocenter"`. It may happen, as suggested by Guin, that the centroid is not feasible. This may happen if the constraints are not convex. In this case, the initial simplex cannot be computed. This is why we provide the `"tox0"` option, which allows to compute the initial simplex by scaling toward the initial guess, which is always feasible.

In Box's paper, the scaling into the non linear constraints is performed "toward" the centroid, that is, by using a scaling factor equal to 0.5. This default scaling factor might be sub-optimal in certain situations. This is why we provide the `-boxineqscaling` option, which allows to configure the scaling factor.

In Box's paper, whether we are concerned with the initial simplex or with the simplex at a given iteration, the scaling for the non linear constraints is performed without end. This is because Box's hypothesis is that "ultimately, a satisfactory point will be found". As suggested by Guin, if the process fails, the algorithm goes into an infinite loop. In order to avoid this, we perform the scaling until a minimum scaling value is reached, as defined by the `-guinalphamin` option.

We have taken into account for the comments by Guin, but it should be emphasized that the current implementation is still as close as possible to Box's algorithm and is not Guin's algorithm. More precisely, during the iterations, the scaling for the non linear constraints is still performed toward the centroid, be it feasible or not.

# User-defined algorithm

The `-mymethod` option allows to configure a user-defined simplex-based algorithm. The reason for this option is that many simplex-based variants of Nelder-Mead's algorithm have been developped over the years, with specific goals. While it is not possible to provide them all, it is very convenient to use the current structure without being forced to make many developments.

The value of the `-mymethod` option is expected to be a Scilab function with the following header

```
function this = myalgorithm ( this )
endfunction
```

where `-this` is the current object.

In order to use the user-defined algorithm, the `-method` option must be set to "mine". In this case, the component performs the optimization exactly as if the user-defined algorithm was provided by the component.

The user interested in that feature may use the internal scripts provided in the distribution as templates and tune his own algorithm from that point. There is of course no warranty that the user-defined algorithm improves on the standard algorithm, so that users use this feature at their own risks.

# User-defined termination

Many termination criteria are found in the bibliography. Users which aim at reproducing the results exhibited in a particular paper may find that that none of the provided termination criteria match the one which is used in the paper. It may also happen that the provided termination criteria are not suitable for the specific test case. In those situation the `-myterminate` option allows to configure a user-defined termination function.

The value of the `-myterminate` option is expected to be a Scilab function with the following header

```
function [ this , terminate , status ] = mystoppingrule ( this , simplex )
endfunction
```

where `-this` is the current object and `-simplex` is the current simplex. The `terminate` output argument is a boolean which is false if the algorithm must continue and true if the algorithm must stop. The `status` output argument is a string which is associated with the current termination criteria.

In order to enable the use of the user-defined termination function, the value of the `-mytermina-teflag` must be set to true. At each iteration, if the -myterminateflag option has been set to true, the user-defined termination is called. If the `terminate` output argument is true, then the algorithm is stopped. In that case, the value of the `-status` option of the `neldermead_get` is the value of the `status` output argument of the user-defined termination function.

# Example #1 : basic use

In the following example, we solve a simple quadratic test case. We begin by defining the cost function, which takes 2 input arguments and returns the objective. The classical starting point [-1.2 1.0] is used. The neldermead_new creates a new neldermead object. Then we use the neldermead_configure method to configure the parameters of the problem. We use all default settings and perform the search for the optimum. The neldermead_display function is used to display the state of the optimization and the neldermead_get is used to retrieve the optimum parameters.

```
function [ f , index ] = quadratic ( x , index )
  f = x(1)^2 + x(2)^2;
endfunction
x0 = [1.0 1.0].';
nm = neldermead_new ();
nm = neldermead_configure(nm,"-numberofvariables",2);
nm = neldermead_configure(nm,"-function",quadratic);
nm = neldermead_configure(nm,"-x0",x0);
nm = neldermead_search(nm);
neldermead_display(nm);
xopt = neldermead_get(nm,"-xopt");
nm = neldermead_destroy(nm);
```

# Example #2 : customized use

In the following example, we solve the Rosenbrock test case. We begin by defining the Rosenbrock function, which takes 2 input arguments and returns the objective. The classical starting point [-1.2 1.0] is used. The neldermead_new creates a new neldermead object. Then we use the neldermead_configure method to configure the parameters of the problem. The initial simplex is computed from the axes and the single length 1.0 (this is the default, but is explicitely written here as an example). The variable simplex algorithm by Nelder and Mead is used, which corresponds to the -method "variable" option. The neldermead_search function performs the search for the minimum. Once the minimum is found, the neldermead_contour allows to compute the data required by the contour function. This is possible since our problem involves only 2 parameters. This function uses the cost function previously configured to compute the required data. The contour plot is directly drawn from the data provided by neldermead_contour. Then we plot the initial guess on the contour plot as a blue dot. The neldermead_get function is used to get the optimum, which is associated with the -xopt option. The optimum is plot on the contour plot as a red dot.

```
mprintf("Defining Rosenbrock function...\n");
function [ f , index ] = rosenbrock ( x , index )
  y = 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
endfunction
x0 = [-1.2 1.0]';
mprintf("x0=%s\n",strcat(string(x0)," "));
mprintf("Creating object...\n");
nm = neldermead_new ();
mprintf("Configuring object...\n");
nm = neldermead_configure(nm,"-numberofvariables",2);
nm = neldermead_configure(nm,"-function",rosenbrock);
nm = neldermead_configure(nm,"-x0",x0);
nm = neldermead_configure(nm,"-maxiter",200);
nm = neldermead_configure(nm,"-maxfunevals",300);
nm = neldermead_configure(nm,"-tolfunrelative",10*%eps);
nm = neldermead_configure(nm,"-tolxrelative",10*%eps);
nm = neldermead_configure(nm,"-simplex0method","axes");
nm = neldermead_configure(nm,"-simplex0length",1.0);
nm = neldermead_configure(nm,"-method","variable");
nm = neldermead_configure(nm,"-verbose",0);
nm = neldermead_configure(nm,"-verbosetermination",0);
mprintf("Searching for minimum...\n");
nm = neldermead_search(nm);
mprintf("Plot contour...\n");
xmin = -2.0 ; xmax = 2.0 ; ymin = -2.0 ; ymax = 2.0 ; nx = 100 ; ny = 100;
stepx = (xmax - xmin)/nx
xdata = xmin:stepx:xmax;
stepy = (ymax - ymin)/ny
ydata = ymin:stepy:ymax;
for ix = 1:length(xdata)
  for iy = 1:length(ydata)
    experiment = [xdata(ix) ydata(iy)];
    [ nm , fiexp ] = neldermead_function ( nm , experiment );
    zdata ( ix , iy ) = fiexp;
  end
end
wnum = 100001;
my_handle             = scf(wnum);
contour ( xdata , ydata , zdata , [1 10 100 500 1000 2000] )
// Plot starting point
```

```
mprintf("x0 : blue dot\n");
plot(x0(1),x0(2));
my_handle.children.children(1).children.mark_mode="on";
my_handle.children.children(1).children.mark_size = 5;
my_handle.children.children(1).children.mark_foreground = 2;
mprintf("xopt : red dot\n");
xopt = neldermead_get(nm,"-xopt");
plot(xopt(1),xopt(2));
my_handle.children.children(1).children.mark_mode="on";
my_handle.children.children(1).children.mark_size = 5;
my_handle.children.children(1).children.mark_foreground = 5;

nm = neldermead_destroy(nm);
```

The -verbose option allows to get detailed informations about the current optimization process. The following is a sample output for an optimization based on the Nelder and Mead variable-shape simplex algorithm. Only the output corresponding to the iteration #156 is displayed. In order to display specific outputs (or to create specific output files and graphics), the -outputcommand option should be used.

```
==================================================================
Iteration #156 (total = 156)
Function Eval #297
Xopt : 1 1
Fopt : 6.871176e-027
DeltaFv : 2.880999e-026
Center : 1 1
Size : 2.548515e-013
Vertex #1/3 : fv=0.000000, x=1.000000 1.000000
Vertex #2/3 : fv=0.000000, x=1.000000 1.000000
Vertex #3/3 : fv=0.000000, x=1.000000 1.000000
nmplot_outputcmd (1)
Reflect
xbar=1 1
Function Evaluation #298 is [1.155D-25] at [1 1]
xr=[1 1], f(xr)=0.000000
Contract - inside
Function Evaluation #299 is [6.023D-27] at [1 1]
xc=1 1, f(xc)=0.000000
  > Perform Inside Contraction
Sort
```

# Example #3 : use output function

In the following example, we show how to use the output command to create specialized outputs. These outputs may be used to create specific data files or make interactive graphic outputs.

We define the function "myoutputcmd", which takes the current state as its first argument. The state is a string which can contain "init", "iter" or "done", depending on the status of the optimization. The data input argument is a tlist, which contains the data associated with the current iteration. In this case, we use the fields to print a message in the console. As another example of use, we could format the message so that it uses LaTeX formatting rules, which may allow the user to directly copy and paste the output into a LaTeX report.

```
function [ f , index ] = rosenbrock ( x , index )
  f = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction


//
// myoutputcmd --
//  This command is called back by the Nelder-Mead
//  algorithm.
// Arguments
//  state : the current state of the algorithm
//    "init", "iter", "done"
//  data : the data at the current state
//    This is a tlist with the following entries:
//    * x : the optimal vector of parameters
//    * fval : the minimum function value
//    * simplex : the simplex, as a simplex object
//    * iteration : the number of iterations performed
//    * funccount : the number of function evaluations
//    * step : the type of step in the previous iteration
//
function myoutputcmd ( state , data )
  iter = data.iteration
  if ( state == "init" ) then
    mprintf ( "================================\n");
    mprintf ( "Initialization\n");
  elseif ( state == "done" ) then
    mprintf ( "================================\n");
    mprintf ( "End of Optimization\n");
  end
  fc = data.funccount
  fval = data.fval
  x = data.x
  simplex = data.simplex
  // Simplex is a data structure, which can be managed
  // by the optimsimplex class.
  ssize = optimsimplex_size ( simplex )
  mprintf ( "Iteration #%d, Feval #%d, Fval = %e, x = %s, Size = %e\n", iter, f
endfunction


nm = neldermead_new ();
nm = neldermead_configure(nm,"-numberofvariables",2);
nm = neldermead_configure(nm,"-function",rosenbrock);
nm = neldermead_configure(nm,"-x0",[-1.2 1.0]');
nm = neldermead_configure(nm,"-maxiter",200);
nm = neldermead_configure(nm,"-maxfunevals",300);
nm = neldermead_configure(nm,"-tolfunrelative",10*%eps);
nm = neldermead_configure(nm,"-tolxrelative",10*%eps);
nm = neldermead_configure(nm,"-simplex0method","axes");
nm = neldermead_configure(nm,"-simplex0length",1.0);
nm = neldermead_configure(nm,"-method","variable");
nm = neldermead_configure(nm,"-verbose",0);
nm = neldermead_configure(nm,"-verbosetermination",0);
nm = neldermead_configure(nm,"-outputcommand",myoutputcmd);
nm = neldermead_search(nm);
nm = neldermead_destroy(nm);
```

The previous script produces the following output.

```
==================================
Initialization
Iteration #0, Feval #4, Fval = 2.420000e+001, x = -1.2 1, Size = 1.000000e+000
Iteration #1, Feval #4, Fval = 2.420000e+001, x = -1.2 1, Size = 1.000000e+000
Iteration #2, Feval #6, Fval = 2.420000e+001, x = -1.2 1, Size = 1.000000e+000
Iteration #3, Feval #8, Fval = 2.420000e+001, x = -1.2 1, Size = 1.000000e+000
Iteration #4, Feval #10, Fval = 9.999182e+000, x = -1.0125 0.78125, Size = 5.97
...
Iteration #155, Feval #296, Fval = 2.024754e-026, x = 1 1, Size = 4.601219e-013
Iteration #156, Feval #298, Fval = 6.871176e-027, x = 1 1, Size = 2.548515e-013
Iteration #157, Feval #300, Fval = 6.023002e-027, x = 1 1, Size = 2.814328e-013
==================================
End of Optimization
Iteration #157, Feval #300, Fval = 6.023002e-027, x = 1 1, Size = 2.814328e-013
```

# Bibliography

"Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation", Spendley, W. and Hext, G. R. and Himsworth, F. R., American Statistical Association and American Society for Quality, 1962

"A Simplex Method for Function Minimization", Nelder, J. A. and Mead, R., The Computer Journal, 1965

"A New Method of Constrained Optimization and a Comparison With Other Methods", M. J. Box, The Computer Journal 1965 8(1):42-52, 1965 by British Computer Society

"Discussion and correspondence: modification of the complex method of constrained optimization", J. A. Guin, The Computer Journal, 1968

"Detection and Remediation of Stagnation in the Nelder--Mead Algorithm Using a Sufficient Decrease Condition", Kelley C. T., SIAM J. on Optimization, 1999

"Iterative Methods for Optimization", C. T. Kelley, SIAM Frontiers in Applied Mathematics, 1999

"Algorithm AS47 - Function minimization using a simplex procedure", O'Neill, R., Applied Statistics, 1971

# Authors

Michael Baudin - INRIA - 2008-2009

Michael Baudin - Digiteo - 2009

# See Also

optimbase, optimsimplex, nmplot

# Nom

overview — An overview of the Nelder-Mead toolbox.

# Purpose

The goal of this toolbox is to provide a Nelder-Mead direct search optimization method. That Nelder-Mead algorithm may be used in the following optimization context :

- there is no need to provide the derivatives of the objective function,
- the number of parameters is small (up to 10-20),
- there are bounds and/or non linear constraints.

# Design

This package provides the following components :

- `neldermead` provides various Nelder-Mead variants and manages for Nelder-Mead specific settings, such as the method to compute the initial simplex, the specific termination criteria,
- `fminsearch` provides a simplified Nelder-Mead algorithm. Specific terminations criteria, initial simplex and auxiliary settings are automatically configured.
- `optimset`, `optimget` provide Scilab commands to emulate their Matlab counterparts.
- `optimplotfunccount`, `optimplotx` and `optimplotfval` provide plotting features for the `fminsearch` function.
- `nmplot` provides a high-level component which provides directly output pictures for Nelder-Mead algorithm.

The current component is based on the following components

- `optimbase` provides an abstract class for a general optimization component, including the number of variables, the minimum and maximum bounds, the number of non linear inequality constraints, the loggin system, various termination criteria, the cost function, etc...
- `optimsimplex` provides a class to manage a simplex made of an arbitrary number of vertices, including the computation of a simplex by various methods (axes, regular, Pfeffer's, randomized bounds), the computation of the size by various methods (diameter, sigma +, sigma-, etc...),

# Features

The following is a list of features the Nelder-Mead prototype algorithm currently provides :

- Provides 3 algorithms, including
  - Spendley et al. fixed shaped algorithm,
  - Nelder-Mead variable shape algorithm,
  - Box "complex" algorithm managing bounds and nonlinear inequality constraints based on arbitrary number of vertices in the simplex.
- Manage various simplex initializations
  - initial simplex given by user,
  - initial simplex computed with a length and along the coordinate axes,

  - initial regular simplex computed with Spendley et al. formula
  - initial simplex computed by a small perturbation around the initial guess point
- Manage cost function
  - optionnal additionnal argument
  - direct communication of the task to perform : cost function or inequality constraints
- Manage various termination criteria, including maximum number of iterations, tolerance on function value (relative or absolute),
  - tolerance on x (relative or absolute),
  - tolerance on standard deviation of function value (original termination criteria in [3]),
  - maximum number of evaluations of cost function,
  - absolute or relative simplex size,
- Manage the history of the convergence, including
  - history of function values,
  - history of optimum point,
  - history of simplices,
  - history of termination criterias,
- Provide a plot command which allows to graphically see the history of the simplices toward the optimum,
- Provide query features for the status of the optimization process number of iterations, number of function evaluations, status of execution, function value at initial point, function value at optimal point, etc...
- Kelley restart based on simplex gradient,
- O'Neill restart based on factorial search around optimum,

# Example : Optimizing the Rosenbrock function

In the following example, one searches the minimum of the 2D Rosenbrock function. One begins by defining the function "rosenbrock" which computes the Rosenbrock function. The traditionnal initial guess [-1.2 1.0] is used. The initial simplex is computed along the axes with a length equal to 0.1. The Nelder-Mead algorithm with variable simplex size is used. The verbose mode is enabled so that messages are generated during the algorithm. After the optimization is performed, the optimum is retrieved with quiery features.

```
function y = rosenbrock (x)
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction

nm = neldermead_new ();
nm = neldermead_configure(nm,"-x0",[-1.2 1.0]');
nm = neldermead_configure(nm,"-simplex0method","axes");
nm = neldermead_configure(nm,"-simplex0length",0.1);
```

```
nm = neldermead_configure(nm,"-method","variable");
nm = neldermead_configure(nm,"-verbose",1);
nm = neldermead_configure(nm,"-function",rosenbrock);
nm = neldermead_search(nm);
xopt = neldermead_get(nm,"-xopt");
fopt = neldermead_get(nm,"-fopt");
historyfopt = neldermead_get(nm,"-historyfopt");
iterations = neldermead_get(nm,"-iterations");
historyxopt = neldermead_get(nm,"-historyxopt");
historysimplex = neldermead_get(nm,"-historysimplex");
fx0 = neldermead_get(nm,"-fx0");
status = neldermead_get(nm,"-status");
nm = neldermead_destroy(nm);
```

# Authors

Michael Baudin, 2008-2009

# Bibliography


“Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation”, Spendley, W. and Hext, G. R. and Himsworth, F. R., American Statistical Association and American Society for Quality, 1962

“A Simplex Method for Function Minimization”, Nelder, J. A. and Mead, R., The Computer Journal, 1965

"A New Method of Constrained Optimization and a Comparison With Other Methods", M. J. Box, The Computer Journal 1965 8(1):42-52, 1965 by British Computer Society

“Convergence Properties of the Nelder--Mead Simplex Method in Low Dimensions”, Jeffrey C. Lagarias and James A. Reeds and Margaret H. Wright and Paul E. Wright, SIAM Journal on Optimization, 1998

“Compact numerical methods for computers : linear algebra and function minimisation”, Nash, J. C., Hilger, Bristol, 1979

“Iterative Methods for Optimization”, C. T. Kelley, 1999

“Iterative Methods for Optimization: Matlab Codes”, http://www4.ncsu.edu/~ctk/matlab_darts.html

“Sequential Simplex Optimization: A Technique for Improving Quality and Productivity in Research, Development, and Manufacturing”, Walters, Fred H. and Jr, Lloyd R. and Morgan, Stephen L. and Deming, Stanley N., 1991

“Numerical Recipes in C, Second Edition”, W. H. Press and Saul A. Teukolsky and William T. Vetterling and Brian P. Flannery, 1992

“Detection and Remediation of Stagnation in the Nelder--Mead Algorithm Using a Sufficient Decrease Condition”, SIAM J. on Optimization, Kelley,, C. T., 1999

Matlab – fminsearch , http://www.mathworks.com/access/helpdesk/help/techdoc/index.html?/access/helpdesk/help/techdoc/ref/fminsearch.html

GAMS, A19A20 - description, http://gams.nist.gov/serve.cgi/Module/NASHLIB/A19A20/11238/

asa047.f, http://people.sc.fsu.edu/~burkardt/f77_src/asa047/asa047.f

optim1.f, http://www.stat.uconn.edu/~mhchen/survbook/example51/optim1.f

as47,f, http://lib.stat.cmu.edu/apstat/47

“Algorithm AS47 - Function minimization using a simplex procedure, O'Neill, R., 1971, Applied Statistics

# TODO

- include Multi-Directionnal Search algorithm in the release
- include parabolic interpolation in the release
- provide a neldermead_outputfunction to make nice and easy outputs
- provide a stopping feature to the cost function and manage the `stop` output argument to the output functions of `fminsearch`
- provide a (quadratic) penalized version of `neldermead` for bound constrained problems

# Nom

nmplot — Provides several direct search optimization algorithms based on the simplex method.

```
newobj = nmplot_new ()
this = nmplot_destroy (this)
this = nmplot_configure (this,key,value)
value = nmplot_cget (this,key)
this = nmplot_display ( this )
value = nmplot_get ( this , key )
this = nmplot_search ( this )
this = nmplot_restart ( this )
[ this , xdata , ydata , zdata ] = nmplot_contour ( this , xmin , xmax , ymin ,
this = nmplot_historyplot ( this , datafile  , mytitle , myxlabel , myylabel )
this = nmplot_simplexhistory ( this , colorforeground , markforeground , markst
```

# Description

This class provides several direct search optimization algorithms based on the simplex method.

The goal of this class is to provide a neldermead component with plotting features. It enables to make fast plots of the algorithm progress through the iterations.

It is a specialized neldermead class, with a specific output command. This output function allows to store the history of several datas through the iterations of the algorithm. These datas are :

- the history of the coordinates of the simplex ,
- the history of the function value (averaged on the vertices),
- the history of the minimum function value in the simplex,
- the history of the size of the simplex (as computed with the sigma+ method).

These data are stored into several data files during the optimization process. Several methods allows to plot the data stored into these data files.

# Design

The nmplot component is built on top of the neldermead component. The -outputcommand option (of the neldermead class) is not available since the nmplot class uses its own output function. Additionnal options -simplexfn, -fbarfn, -foptfn and -sigmafn are provided, which allows to configure the file names where the data is stored.

The nmplot class can be considered as a sample test case of the -outputcommand option of the neldermead class. It gives an example of the situation where the user wants to get specialized outputs out of the neldermead class.

# Functions

The following functions are available.

newobj = nmplot_new ()
: Creates a new nmplot object.

  newobj
  : The new object.

this = nmplot_destroy (this)
: Destroy the given object.

this
The current object.

this = nmplot_configure (this,key,value)
Configure the current object with the given value for the given key.

this
The current object.

key
the key to configure. The following keys are available.

-verbose
set to 1 to enable verbose logging. (default is 0)

-verbosetermination
set to 1 to enable verbose termination logging. (default is 0)

-x0
the initial guess, as a n x 1 column vector, where n is the number of variables.

-maxfunevals
the maximum number of function evalutations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxfuneval".

-maxiter
the maximum number of iterations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxiter".

-tolfunabsolute
the absolute tolerance for the function value (default is 0.0).

-tolfunrelative
the relative tolerance for the function value (default is %eps).

-tolfunmethod
the method used for the tolerance on function value in the termination criteria.

The following values are available : "absolute+relative", "relative", "absolute", "disabled" (default is "disabled"). If this criteria is triggered, the status of the optimization is set to "tolf".

-tolxabsolute
the absolute tolerance on x (default is 0.0).

-tolxrelative
the relative tolerance on x (default is %eps).

-tolxmethod
the method used for the tolerance on x in the termination criteria.

The following values are available : "relative", "absolute", "disabled" (default is "relative"). If this criteria is triggered, the status of the optimization is set to "tolx".

-function
the objective function, which computes the value of the cost and the non linear constraints, if any.

See below for the details of the communication between the optimization system and the cost function.

-costfargument
an additionnal argument, passed to the cost function.

-outputcommand
: a command which is called back for output.

  See below for the details of the communication between the optimization system and the output command function.

-outputcommandarg
: an additionnal argument, passed to the output command.

-numberofvariables
: the number of variables to optimize (default is 0).

-storehistory
: set to 1 to enable the history storing (default is 0).

-boundsmin
: the minimum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-boundsmax
: the maximum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-nbineqconst
: the number of inequality constraints (default is 0)

-method
: the name of the algorithm to use. The following methods are available :

  "fixed"
  : the Spendley et al. fixed simplex shape algorithm. This algorithm is for unconstrained problems (i.e. bounds and non linear constraints are not taken into account)

  "variable"
  : the Nelder-Mead variable simplex shape algorithm. This algorithm is for unconstrained problems (i.e. bounds and non linear constraints are not taken into account)

  "box"
  : the Box complex algorithm. This algorithm takes into account bounds and nonlinear inequality constraints.

-simplex0method
: the method to use to compute the initial simplex. The first vertex in the simplex is always the initial guess associated with the -x0 option. The following methods are available :

  "given"
  : the coordinates associated with the -coords0 option are used to compute the initial simplex, with arbitrary number of vertices.

    This allow the user to setup the initial simplex by a specific method which is not provided by the current component (for example with a simplex computed from a design of experiments). This allows also to configure the initial simplex so that a specific behaviour of the algorithm an be reproduced (for example the Mac Kinnon test case).

    The given matrix is expected to have n rows and k columns, where n is the dimension of the problem and k is the number of vertices.

  "axes"
  : the simplex is computed from the coordinate axes and the length associated with the -simplex0length option.

"spendley"
the simplex is computed so that it is regular with the length associated with the -simplex0length option (i.e. all the edges have the same length).

"pfeffer"
the simplex is computed from an heuristic, in the neighborhood of the initial guess. This initial simplex depends on the -simplex0deltausual and -simplex0deltazero.

"randbounds"
the simplex is computed from the bounds and a random number. This option is available only if bounds are available : if bounds are not available, an error is generated. This method is usually associated with Box's algorithm. The number of vertices in the simplex is taken from the -boxnbpoints option.

-coords0
the coordinates of the vertices of the initial simplex. If the -simplex0method option is set to "given", these coordinates are used to compute the initial simplex. This matrix is expected to have shape nbve x n where nbve is the number of vertices and n is the number of variables.

-simplex0length
the length to use when the initial simplex is computed with the "axes" or "spendley" methods. If the initial simplex is computed from "spendley" method, the length is expected to be a scalar value. If the initial simplex is computed from "axes" method, it may be either a scalar value or a vector of values, with rank n, where n is the number of variables.

-simplex0deltausual
the relative delta for non-zero parameters in "pfeffer" method. The default value is 0.05.

-simplex0deltazero
the absolute delta for non-zero parameters in "pfeffer" method. The default value is 0.0075.

-rho
the reflection coefficient. This parameter is used when the -method option is set to "fixed" or "variable". The default value is 1.0.

-chi
the expansion coefficient. This parameter is used when the -method option is set to "variable". The default value is 2.0.

-gamma
the contraction coefficient. This parameter is used when the -method option is set to "variable". The default value is 0.5.

-sigma
the shrinkage coefficient. This parameter is used when the -method option is set to "fixed" or "variable". The default value is 0.5.

-tolfstdeviationmethod
set to "enabled" to enable the termination criteria based on the standard deviation of the function values in the simplex. The default value is "disabled". If this criteria is triggered, the status of the optimization is set to "tolfstdev".

This criteria is suggested by Nelder and Mead.

-tolfstdeviation
the absolute tolerance on standard deviation. The default value is 0.0.

-tolsimplexizemethod
set to "disabled" to disable the tolerance on the simplex size. The default value is "enabled". If this criteria is triggered, the status of the optimization is set to "tolsize".

When this criteria is enabled, the values of the options -tolsimplexizeabsolute and -tolsimplexizerelative are used in the termination criteria. The method to compute the size is the "sigmaplus" method.

-tolsimplexizeabsolute
the absolute tolerance on the simplex size. The default value is 0.0.

-tolsimplexizerelative
the relative tolerance on the simplex size. The default value is %eps.

-tolssizedeltafvmethod
set to "enabled" to enable the termination criteria based on the size of the simplex and the difference of function value in the simplex. The default value is "disabled". If this criteria is triggered, the status of the optimization is set to "tolsizedeltafv".

This termination criteria uses the values of the options -tolsimplexizeabsolute and -toldeltafv. This criteria is identical to Matlab's fminsearch.

-toldeltafv
the absolute tolerance on the difference between the highest and the lowest function values.

-kelleystagnationflag
set to 1 to enable the termination criteria using Kelley's stagnation detection, based on sufficient decrease condition. The default value is 0. If this criteria is triggered, the status of the optimization is set to "kelleystagnation".

-kelleynormalizationflag
set to 0 to disable the normalization of the alpha coefficient in Kelley's stagnation detection, i.e. use the value of the option -kelleystagnationalpha0 as is. Default value is 1, i.e. the simplex gradient of the initial simplex is taken into account in the stagnation detection.

-kelleystagnationalpha0
the parameter used in Kelley's stagnation detection. The default value is 1.e-4.

-restartflag
set to 1 to enable the automatic restart of the algorithm. Default value is 0.

-restartdetection
the method to detect if the automatic restart must be performed. The following methods are available :

"oneill"
the factorial local optimality test by O'Neill is used. If the test finds a local point which is better than the computed optimum, a restart is performed.

"kelley"
the sufficient decrease condition by O'Neill is used. If the test finds that the status of the optimization is "kelleystagnation", a restart is performed. This status may be generated if the -kelleystagnationflag option is set to 1.

The default method is "oneill".

-restartmax
the maximum number of restarts, when automatic restart is enabled via the -restartflag option. Default value is 3.

-restarteps
the absolute epsilon value used to check for optimality in the factorial O'Neill restart detection. The default value is %eps.

-restartstep
: the absolute step length used to check for optimality in the factorial O'Neill restart detection. The default value is 1.0.

-restartsimplexmethod
: the method to compute the initial simplex after a restart. The following methods are available.

"given"
: the coordinates associated with the -coords0 option are used to compute the initial simplex, with arbitrary number of vertices.

This allow the user to setup the initial simplex by a specific method which is not provided by the current component (for example with a simplex computed from a design of experiments). This allows also to configure the initial simplex so that a specific behaviour of the algorithm an be reproduced (for example the Mac Kinnon test case).

The given matrix is expected to have n rows and k columns, where n is the dimension of the problem and k is the number of vertices.

"axes"
: the simplex is computed from the coordinate axes and the length associated with the -simplex0length option.

"spendley"
: the simplex is computed so that it is regular with the length associated with the -simplex0length option (i.e. all the edges have the same length).

"pfeffer"
: the simplex is computed from an heuristic, in the neighborhood of the initial guess. This initial simplex depends on the -simplex0deltausual and -simplex0deltazero.

"randbounds"
: the simplex is computed from the bounds and a random number. This option is available only if bounds are available : if bounds are not available, an error is generated. This method is usually associated with Box's algorithm. The number of vertices in the simplex is taken from the -boxnbpoints option.

"oriented"
: the simplex is computed so that it is oriented, as suggested by C.T. Kelley.

The default method is "oriented".

-boxnbpoints
: the number of points in the initial simplex, when the -restartsimplexmethod option is set to "randbounds". The default value is so that the number of points is twice the number of variables of the problem.

-nbineqloops
: the number of loops to perform in Box and Box-Guin algorithms to scale the trial point for function improvement or into the constraints. Default value is 10.

-ineqscaling
: the scaling coefficient used to scale the trial point for function improvement or into the constraints. Default value is 0.5

-simplexfn
: the name of the file containing the history of the simplex. Default value is the empty string.

-fbarfn
the name of the file containing the history of the function value, averaged on the vertices of the simplex. Default value is the empty string.

-foptfn
the name of the file containing the history of the minimum function value in the simplex. Default value is the empty string.

-sigmafn
the name of the file containing the history of the size of the simplex. Default value is the empty string.

value
the value.

value = nmplot_cget (this,key)
Get the value for the given key. If the key is unknown, generates an error.

this
The current object.

key
the name of the key to quiery. The list of available keys is the same as for the nmplot_configure function.

value = nmplot_get ( this , key )
Get the value for the given key. If the key is unknown, generates an error.

this
The current object.

key
the key to get.

The following keys are available :

-funevals
the number of function evaluations

-iterations
the number of iterations

-xopt
the x optimum, as a n x 1 column vector, where n is the number of variables.

-fopt
the optimum cost function value

-historyxopt
an array, with nbiter values, containing the history of x during the iterations.

This array is available after optimization if the history storing was enabled with the -storehistory option.

-historyfopt
an array, with nbiter values, containing the history of the function value during the iterations.

This array is available after optimization if the history storing was enabled with the -storehistory option.

-fx0
the function value for the initial guess

-status
a string containing the status of the optimization. See below for details about the optimization status.

-historysimplex
a matrix containing the history of the simplex during the iterations. This matrix has rank nbiter x nbve x n, where nbiter is the number of iterations, nbve is the number of vertices in the simplex and n is the number of variables.

-simplexopt
the optimum simplex. This is a simplex object, which is suitable for processing with the simplex interface.

-restartnb
the number of actual restarts performed.

Most fields are available only after an optimization has been performed with one call to the neldermead_search method.

this = nmplot_display ( this )
Display the current settings in the console.

this
The current object.

this = nmplot_search ( this )
Performs the optimization associated with the method associated with the -method option and find the optimum.

this
The current object.

If the -restartflag option is enabled, automatic restarts are performed, based on the -restartdetection option.

this = nmplot_restart ( this )
Restarts the optimization by updating the simplex and performing a new search.

this
The current object.

[ this , xdata , ydata , zdata ] = nmplot_contour ( this , xmin , xmax , ymin , ymax , nx , ny )
Plot the contours of the cost function. The cost function must be a function with two parameters.

this
The current object.

xmin , xmax , ymin , ymax
the bounds for the contour plot

nx , ny
the number of points in the directions x, y

xdata , ydata , zdata
vectors of data, as required by the contour function

nmplot_simplexhistory ( this , colorforeground , markforeground , markstyle )
Plots the simplex history on the current graphic window. The colorforeground , markforeground , markstyle options are provided to produce fast plots. Specific settings can still be applied with the usual graphic features.

this
The current object.

colorforeground
: the color of the foreground for the simplices. Default value is 5.

markforeground
: the foreground mark for the simplices. Default value is 3.

markstyle
: the mark style for the simplices. Default value is 9.

nmplot_historyplot ( this , datafile , mytitle , myxlabel , myylabel )
: Plots the history from the given data file on the current graphic window. The mytitle, myxlabel, myylabel options are provided as a way to produce plots faster. Specific settings can still be applied with the usual graphic features.

this
: The current object.

datafile
: the data file which contains the history. The file is expected to be formatted in a way similar to the files associated with the -fbarfn, -foptfn and -sigmafn files. The default value is the value of the -foptfn option.

mytitle
: the title of the plot. Default value is the empty string.

myxlabel
: the x label for the plot. Default value is the empty string.

myylabel
: the y label for the plot. Default value is the empty string.

[ this , result ] = nmplot_function ( this , x , index )
: Call the cost function and return the value.

this
: The current object.

x
: the point where the function is to be evaluated

index
: optionnal, a flag to pass to the cost function (default = 1). See the section "The cost function" of the neldermead component for available values of index.

# Example

In the following example, we use the fixed shape Spendley et al. simplex algorithm and find the minimum of a quadratic function. We begin by defining a quadratic function associated with 2 input variables. We then define an nmplot object and configure the object so that the "fixed" shape simplex algorithm is used with the regular initial simplex associated with the "spendley" key. Four files are configured, which will contain the history of the simplex, the history of fbar, fopt and sigma through the iterations. The search is performed by the nmplot_search function, which writes the 4 data files during the iterations. The nmplot_contour function is called in order to compute the arrays xdata, ydata and zdata which are required as input to the contour function. The nmplot_simplexhistory then uses the history of the simplex, as stored in the rosenbrock.fixed.history.simplex.txt data file, and plot the various simplices on the contour plot. The nmplot_historyplot is used with the files rosenbrock.fixed.history.fbar.txt, rosenbrock.fixed.history.fopt.txt and rosenbrock.fixed.history.sigma.txt, which produces 3 plots of the history of the optimization algorithm through the iterations.

```
mprintf("Defining quadratic function...\n");
function y = quadratic (x)
  y = x(1)^2 + x(2)^2 - x(1) * x(2);
endfunction

mprintf("Creating nmplot object...\n");
nm = nmplot_new ();
nm = nmplot_configure(nm,"-numberofvariables",2);
nm = nmplot_configure(nm,"-function",quadratic);
nm = nmplot_configure(nm,"-x0",[2.0 2.0]');
nm = nmplot_configure(nm,"-maxiter",100);
nm = nmplot_configure(nm,"-maxfunevals",300);
nm = nmplot_configure(nm,"-tolxrelative",1.e-8);
nm = nmplot_configure(nm,"-simplex0method","spendley");
nm = nmplot_configure(nm,"-method","fixed");
//
// Setup output files
//
nm = nmplot_configure(nm,"-simplexfn","rosenbrock.fixed.history.simplex.txt");
nm = nmplot_configure(nm,"-fbarfn","rosenbrock.fixed.history.fbar.txt");
nm = nmplot_configure(nm,"-foptfn","rosenbrock.fixed.history.fopt.txt");
nm = nmplot_configure(nm,"-sigmafn","rosenbrock.fixed.history.sigma.txt");
//
// Perform optimization
//
mprintf("Searching for minimum...\n");
nm = nmplot_search(nm);
nmplot_display(nm);
// Plot the contours of the cost function and the simplex history
mprintf("Plotting contour...\n");
[nm , xdata , ydata , zdata ] = nmplot_contour ( nm , xmin = -2.0 , xmax = 4.0
f = scf();
contour ( xdata , ydata , zdata , [0.1 1.0 2.0 5.0 10.0 15.0 20.0] )
nmplot_simplexhistory ( nm );
mprintf("Plotting history of fbar...\n");
f = scf();
nmplot_historyplot ( nm , "rosenbrock.fixed.history.fbar.txt" , ...
  mytitle = "Function Value Average" , myxlabel = "Iterations" );
mprintf("Plotting history of fopt...\n");
f = scf();
nmplot_historyplot ( nm , "rosenbrock.fixed.history.fopt.txt" , ...
  mytitle = "Minimum Function Value" , myxlabel = "Iterations" );
mprintf("Plotting history of sigma...\n");
f = scf();
nmplot_historyplot ( nm , "rosenbrock.fixed.history.sigma.txt" , ...
  mytitle = "Maximum Oriented length" , myxlabel = "Iterations" );
deletefile("rosenbrock.fixed.history.simplex.txt");
deletefile("rosenbrock.fixed.history.fbar.txt");
deletefile("rosenbrock.fixed.history.fopt.txt");
deletefile("rosenbrock.fixed.history.sigma.txt");
nm = nmplot_destroy(nm);
```

# TODO

- add an example

# Authors

Michael Baudin - INRIA - 2008-2009

Michael Baudin - Digiteo - 2009

# See Also

neldermead

# Nom

optimget — Queries an optimization data structure.

```
val = optimget ( options , key )
val = optimget ( options , key , value )
```

# Description

This function allows to make queries on an existing optimization data structure. This data structure must have been created and updated by the `optimset` function. The `optimget` allows to retrieve the value associated with a given key.

In the following, we analyse the various ways to call the optimget function.

The following calling sequence

```
val = optimget(options , key)
```

returns the value associated with the given `key`. The `key` is expected to be a string. We search the `key` among the list of all possible fields in the `options` data structure. In this search, the case is ignored. The `key` which matches a possible field is returned. Some letters of the field may be dropped, provided that the matching field is unique. For example, the key "MaxF" corresponds to the field "MaxFunEval". But the key "Tol" corresponds both to the "TolX" and "TolFun" fields and therefore will generate an error.

The following calling sequence

```
val = optimget(options, key, value)
```

allows to manage default value for optimization parameters. Indeed, if the field corresponding to the `key` is empty (i.e. has not been set by the user), the input argument `value` is returned. Instead, if the field corresponding the the key is not empty (i.e. has been set by the user), the parameter stored in the `options` argument is returned.

# Parameters

options
: A struct which contains the optimization fields.

key
: A string corresponding to a field of the optimization structure.

value
: A real default value.

val
: The real value corresponding to the key.

# Example #1

In the following example, we create an optimization structure and set the "TolX" field to 1.e-12. Then we use `optimget` to get back the value.

```
op = optimset();
op = optimset(op,'TolX',1.e-12);
val = optimget(op,'TolX'); // val is 1.e-12
```

# Example #2

In the following example, we create an empty optimization structure. Then we use `optimget` to get back the value corresponding to the "TolX" field, with 1.e-5 as the default value. Since the field is empty, we retrieve 1.e-5.

```
op = optimset();
val = optimget(op,'TolX' , 1.e-5); // val = 1.e-5
```

# Example #3

In the following example, we create an optimization structure and set the "TolX" field to 1.e-12. Then we use `optimget` to get back the value corresponding to the "TolX" field, with 1.e-5 as the default value. Since the field is not empty, we retrieve 1.e-12.

```
op = optimset();
op = optimset(op,'TolX',1.e-12);
val = optimget(op,'TolX' , 1.e-5); // val = 1.e-12
```

# Example #4

In the following example, we create an optimization structure and configure the maximum number of function evaluations to 1000. Then we quiery the data structure, giving only the "MaxF" key to the `optimget` function. Since that corresponds only to the "MaxFunEvals" field, there is only one match and the function returns 10000.

```
op = optimset();
op = optimset(op, 'MaxFunEvals' , 1000);
val = optimget(op, 'MaxF'); // val = 1000
```

# Authors

Michael Baudin - Digiteo - 2009

# See Also

optimset

# Nom

optimplotfunccount — Plot the number of function evaluations of an optimization algorithm

```
optimplotfunccount ( x , optimValues , state )
```

# Description

This function creates and updates a plot of the number of function evaluations, depending on the number of iterations. It is a pre-defined plot function which should be used as an option of the "PlotFcns" option.

# Example

In the following example, we use the `optimplotfunccount` function for use with the `fminsearch` function.

```
function y = rosenbrock (x)
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "PlotFcns" , optimplotfunccount );
[x fval] = fminsearch ( rosenbrock , [-1.2 1] , opt );
```

# Authors

Michael Baudin - Digiteo - 2009

# See Also

optimset , fminsearch , optimplotx , optimplotfval

# Nom

optimplotfval — Plot the function value of an optimization algorithm

```
optimplotfval ( x , optimValues , state )
```

# Description

This function creates and updates a plot of the function value, depending on the number of iterations. It is a pre-defined plot function which should be used as an option of the "PlotFcns" option.

# Example

In the following example, we use the `optimplotfval` function for use with the `fminsearch` function.

```
function y = rosenbrock (x)
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "PlotFcns" , optimplotfval );
[x fval] = fminsearch ( rosenbrock , [-1.2 1] , opt );
```

# Authors

Michael Baudin - Digiteo - 2009

# See Also

optimset , fminsearch , optimplotx , optimplotfunccount

# Nom

optimplotx — Plot the value of the parameters of an optimization algorithm

```
optimplotx ( x , optimValues , state )
```

# Description

This function creates and updates a plot of the value of the parameters depending on the number of iterations. It is a pre-defined plot function which should be used as an option of the "PlotFcns" option.

# Example

In the following example, we use the `optimplotx` function for use with the `fminsearch` function.

```
function y = rosenbrock (x)
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
opt = optimset ( "PlotFcns" , optimplotx );
[x fval] = fminsearch ( rosenbrock , [-1.2 1] , opt );
```

# Authors

Michael Baudin - Digiteo - 2009

# See Also

optimset , fminsearch , optimplotfval , optimplotfunccount

# Nom

optimset — Configures and returns an optimization data structure.

```
options = optimset ()
options = optimset ( funname )
options = optimset ( key , value )
options = optimset ( key1 , value1 , key2 , value2, ... )
options = optimset ( oldoptions , key , value )
```

# Description

This function creates or updates a data structure which can be used on modify the behaviour of optimization methods. The goal of this function is to manage the "options" data structure, which is a struct with a set of fields (for example, "MaxFunEvals", "MaxIter", etc...). The user can create a new structure with empty fields or create a new structure with default fields which correspond to a particular algorithm. The user can also configure each field and set it to a particular value. Finally, the user pass the structure to an optimization function so that the algorithm uses the options configured by the user.

In the following, we analyse the various ways to call the optimset function.

The following calling sequence

```
options = optimset ()
```

creates a new data structure where the fields have been set to the empty matrix (i.e. []).

The following calling sequence

```
options = optimset ( funname )
```

creates a new data structure where the default parameters which correspond to the "funname" function have been set. For example,

```
options = optimset ( "fminsearch" )
```

returns a new data structure where the default parameters which correspond to the "fminsearch" function have been set.

The following calling sequence

```
options = optimset ( oldoptions , key , value )
```

creates a new data structure where all fields from the "oldoptions" structure have been copied, except the field corresponding to the "key", which has been set to "value".

# Parameters

key
: a string. The following keys are available :

- "Display"
- "FunValCheck"
- "MaxFunEvals"
- "MaxIter"
- "OutputFcn"
- "PlotFcns"

value
: the value of the parameter

options
: A struct which contains the following fields. By default, all fields are empty. Specific settings are associated with a particular function name.

options.Display
: The verbose level. The default value is "notify". The following is a list of available verbose levels.

- options.Display = "off": the algorithm displays no message at all.
- options.Display = "notify": the algorithm displays message if the termination criteria is not reached at the end of the optimization. This may happen if the maximum number of iterations of the maximum number of function evaluations is reached and warns the user of a convergence problem.
- options.Display = "final": the algorithm displays a message at the end of the optimization, showing the number of iterations, the number of function evaluations and the status of the optimization. This option includes the messages generated by the "notify" option i.e. warns in case of a convergence problem.
- options.Display = "iter": the algorithm displays a one-line message at each iteration. This option includes the messages generated by the "notify" option i.e. warns in case of a convergence problem. It also includes the message generated by the "final" option.

options.FunValCheck
: A boolean to enable the checking of function values.

options.MaxFunEvals
: The maximum number of evaluations of the cost function.

options.MaxIter
: The maximum number of iterations.

options.OutputFcn
: A function which is called at each iteration to print out intermediate state of the optimization algorithm (for example into a log file).

options.PlotFcns
: A function which is called at each iteration to plot the intermediate state of the optimization algorithm (for example into a 2D graphic).

options.TolFun
: The absolute tolerance on function value.

options.TolX
: The absolute tolerance on the variable x.

funname
: A string containing the name of an optimization function which takes the options structure as input argument. Currently, the only possible value is "fminsearch".

# Design

Most optimization algorithms require many algorithmic parameters such as the number of iterations or the number of function evaluations. If these parameters are given to the optimization function as input parameters, this forces both the user and the developper to manage many input parameters. For example, the "optim" function provides more than 20 input arguments. The goal of the optimset function is to simplify the management of input arguments, by gathering all the parameters into a single data structure.

While the current implementation of the "optimset" function only supports the fminsearch function, it is designed to be extended to as many optimization function as required. Because all optimization algorithms do not require the same parameters, the data structure aims at remaining flexible. But, most of the time, most parameters are the same from algorithm to algorithm, for example, the tolerance parameters which drive the termination criteria are often the same, even if the termination criteria itself is not the same.

# Output and plot functions

The "OutputFcn" and "PlotFcns" options accept as argument a function (or a list of functions). In the client optimization algorithm, this output or plot function is called back once per iteration. It can be used by the user to display a message in the console, write into a file, etc...

The output or plot function is expected to have the following definition:

```
function myfun ( x , optimValues , state )
```

where the input parameters are:

- `x`: the current point
- `optimValues`: a `struct` which contains the following fields
  - `optimValues.funccount`: the number of function evaluations
  - `optimValues.fval`: the best function value
  - `optimValues.iteration`: the current iteration number
  - `optimValues.procedure`: the type of step performed. This string depends on the specific algorithm see fminsearch for details.
- `state`: the state of the algorithm. The following states are available : "init", "iter" and "done".
  - "init", when the algorithm is initializing,
  - "iter", when the algorithm is performing iterations,
  - "done", when the algorithm is terminated.

# Example #1

In the following example, we create an empty optimization structure.

```
op = optimset ()
```

# Example #2

In the following example, we create an optimization structure with all fields set to specific settings.

```
op = optimset ("Display","iter",...
               "FunValCheck","on",...
               "MaxFunEvals",100,...
               "MaxIter",110,...
               "OutputFcn",myoutputfun,...
               "PlotFcns",myplotfun,...
               "TolFun",1.e-12,...
               "TolX",1.e-13)
```

# Example #3

In the following example, we create an optimization structure with all fields set to the default settings for the "fminsearch" optimization function.

```
op = optimset ("fminsearch")
```

# Authors

Michael Baudin - INRIA - 2008-2009

Michael Baudin - Digiteo - 2009

# See Also

optimget

# Capítulo 3. Optimization base

# Nom

optimbase — Provides an abstract class for a general optimization component.

```
newobj = optimbase_new ()
this = optimbase_destroy (this)
this = optimbase_configure (this,key,value)
value = optimbase_cget (this,key)
value = optimbase_get (this,key)
this = optimbase_set ( this , key , value )
[ this , isok ] = optimbase_checkbounds ( this )
[ opt , isok ] = optimbase_checkx0 ( this )
this = optimbase_display ( this )
[ this , f , index [ , data ] ] = optimbase_function ( this , x , index [ , data
[ this , f , c , index [ , data ] ] = optimbase_function ( this , x , index [ ,
[ this , f , g , index [ , data ] ] = optimbase_function ( this , x , index [ ,
[ this , f , g , c , gc , index [ , data ] ] = optimbase_function ( this , x ,
[ this , hasbounds ] = optimbase_hasbounds ( this )
[ this , hascons ] = optimbase_hasconstraints ( this )
[ this , hasnlcons ] = optimbase_hasnlcons ( this )
value = optimbase_histget ( this , iter , key )
this = optimbase_histset ( this , iter , key , value )
this = optimbase_incriter ( this )
[ this , isfeasible ] = optimbase_isfeasible ( this , x )
this = optimbase_log (this,msg)
optimbase_outputcmd ( this , state , data )
data = optimbase_outstruct ( this )
[ this , p ] = optimbase_proj2bnds ( this ,  x )
this = optimbase_stoplog ( this , msg )
[this , terminate , status] = optimbase_terminate (this , previousfopt , curren
this = optimbase_checkcostfun ( this )
[ this , isfeasible ] = optimbase_isinbounds ( this , x )
[ this , isfeasible ] = optimbase_isinnonlinconst ( this , x )
```

# Purpose

The goal of this component is to provide a building block for optimization methods. The goal is to provide a building block for a large class of specialized optimization methods. This component manages

- the number of variables,
- the minimum and maximum bounds,
- the number of non linear inequality constraints,
- the cost function,
- the logging system,
- various termination criteria,
- etc...

# Design

This toolbox is designed with Oriented Object ideas in mind.

# Features

The following is a list of features the Optimbase toolbox currently provides :

- Manage cost function
  - optionnal additionnal argument
  - direct communication of the task to perform : cost function or inequality constraints
- Manage various termination criteria, including
  - maximum number of iterations,
  - tolerance on function value (relative or absolute),
  - tolerance on x (relative or absolute),
  - maximum number of evaluations of cost function,
- Manage the history of the convergence, including
  - history of function values,
  - history of optimum point.
- Provide query features for
  - the status of the optimization process,
  - the number of iterations,
  - the number of function evaluations,
  - function value at initial point,
  - function value at optimal point,
  - the optimum parameters,
  - etc...

# Description

This set of commands allows to manage an abstract optimization method. The goal is to provide a building block for a large class of specialized optimization methods. This component manages the number of variables, the minimum and maximum bounds, the number of non linear inequality constraints, the logging system, various termination criteria, the cost function, etc...

The optimization problem to solve is the following

```
min f(x)
l_i <= x_i <= h_i, i = 1,n
g_i(x) <= 0, i = 1,nbineq
```

where

n
: number of variables

nbineq
number of inequality constraints

# Functions

The following functions are available.

newobj = optimbase_new ()
Creates a new optimization object.

newobj
The new object.

this = optimbase_destroy (this)
Destroy the given object.

this
The current object.

this = optimbase_configure (this,key,value)
Configure the current object with the given value for the given key.

this
The current object.

key
the key to configure. The following keys are available.

-verbose
set to 1 to enable verbose logging. (default is 0)

-verbosetermination
set to 1 to enable verbose termination logging. (default is 0)

-x0
the initial guess.

-maxfunevals
the maximum number of function evalutations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxfuneval".

-maxiter
the maximum number of iterations (default is 100). If this criteria is triggered, the status of the optimization is set to "maxiter".

-tolfunabsolute
the absolute tolerance for the function value (default is 0.0).

-tolfunrelative
the relative tolerance for the function value (default is %eps).

-tolfunmethod
the method used for the tolerance on function value in the termination criteria.

The following values are available : %t, %f (default is %f). If this criteria is triggered, the status of the optimization is set to "tolf".

-tolxabsolute
the absolute tolerance on x (default is 0.0).

-tolxrelative
the relative tolerance on x (default is %eps).

-tolxmethod
the method used for the tolerance on x in the termination criteria.

The following values are available : %t, %f (default is %t). If this criteria is triggered, the status of the optimization is set to "tolx".

-function
the objective function, which computes the value of the cost and the non linear constraints, if any.

See below for the details of the communication between the optimization system and the cost function.

-costfargument
an additionnal argument, passed to the cost function.

-outputcommand
a command which is called back for output.

See below for the details of the communication between the optimization system and the output command function.

-outputcommandarg
an additionnal argument, passed to the output command.

-numberofvariables
the number of variables to optimize (default is 0).

-storehistory
set to %t to enable the history storing (default is %f).

-boundsmin
the minimum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-boundsmax
the maximum bounds for the parameters, as an array of values (default is empty, i.e. there are no bounds).

-nbineqconst
the number of inequality constraints (default is 0)

-logfile
the name of the log file

-withderivatives
set to %t if the algorithm uses derivatives. Default is %f.

value
the value.

value = optimbase_cget (this,key)
Get the value for the given key. If the key is unknown, generates an error.

this
The current object.

key
the name of the key to quiery. The list of available keys is the same as for the optimbase_configure function.

[ this , isok ] = optimbase_checkbounds ( this )

Check if the bounds are consistent and puts an error message if not. One could generate an error, but errors are not testable with the current system.

this

The current object.

[ opt , isok ] = optimbase_checkx0 ( this )

Returns %T if the initial guess is consistent with the bounds and the non linear inequality constraints, if any.

this

The current object.

this = optimbase_display ( this )

Display the current settings in the console.

this

The current object.

[ this , f , index [ , data ] ] = optimbase_function ( this , x , index [ , data ] ), [ this , f , c , index [ , data ] ] = optimbase_function ( this , x , index [ , data ] ), [ this , f , g , index [ , data ] ] = optimbase_function ( this , x , index [ , data ] ), [ this , f , g , c , gc , index [ , data ] ] = optimbase_function ( this , x , index [ , data ] )

Call the cost function and return the required results.

If a cost function additionnal argument is defined in current object, pass it to the function as the last argument. See below for details.

this

The current object.

x

the current point, as a column vector

index

what value to compute.

See below in the section "Cost function" for details on this index.

f

the value of the cost function

g

the gradient of the cost function

c

the nonlinear, positive, inequality constraints

gc

the gradient of the nonlinear, positive, inequality constraints

data

an optionnal user-provided argument

this = optimbase_set ( this , key , value )

Set the value for the given key. If the key is unknown, generates an error.

this

The current object.

key
the key to set

The following keys are available :

-funevals
the number of function evaluations

-iterations
the number of iterations

-xopt
the x optimum

-fopt
the optimum cost function value

-historyxopt
an array, with nbiter values, containing the history of x during the iterations.

This array is available after optimization if the history storing was enabled with the -storehistory option.

-historyfopt
an array, with nbiter values, containing the history of the function value during the iterations.

This array is available after optimization if the history storing was enabled with the -storehistory option.

-fx0
the function value for the initial guess

-status
a string containing the status of the optimization

value
the value to set

value = optimbase_get (this,key)
Get the value for the given key. If the key is unknown, generates an error. This command corresponds with options which are not available directly to the optimbase_configure function, but are computed internally.

this
The current object.

key
the name of the key to quiery.

The list of available keys is the same as the optimbase_set function.

[ this , hasbounds ] = optimbase_hasbounds ( this )
Returns %T if current problem has bounds.

this
The current object.

[ this , hascons ] = optimbase_hasconstraints ( this )
Returns %T if current problem has bounds constraints, linear constraints or non linear constraints.

this
The current object.

[ this , hasnlcons ] = optimbase_hasnlcons ( this )

Returns %T if current problem has non linear constraints.

this

The current object.

this = optimbase_histset ( this , iter , key , value )

Set the history value at given iteration for the given key. If the key is unknown, generates an error.

this

The current object.

iter

the iteration number to get

key

the name of the key to quiery.

The list of available keys is the following : "-xopt", "-fopt".

value

the value to set

value = optimbase_histget ( this , iter , key )

Returns the history value at the given iteration number for the given key. If the key is unknown, generates an error.

this

The current object.

iter

the iteration number to get

key

the name of the key to quiery.

The list of available keys is the same as the optimbase_histset function.

this = optimbase_incriter ( this )

Increments the number of iterations.

this

The current object.

[ this , isfeasible ] = optimbase_isfeasible ( this , x )

Returns 1 if the given point satisfies bounds constraints and inequality constraints. Returns 0 if the given point is not in the bounds. Returns -1 if the given point does not satisfies inequality constraints.

this

The current object.

x

the current point

this = optimbase_log (this,msg)

If verbose logging is enabled, prints the given message in the console. If verbose logging is disabled, does nothing. If the -lofgile option has been set, writes the message into the file instead of writing to the console. If the console is too slow, writing into a file can be a solution, since it is very fast.

this
The current object.

msg
the message to print

optimbase_outputcmd ( this , state , data )
Calls back user's output command.

this
The current object.

data = optimbase_outstruct ( this )
Returns a tlist with basic optimization fields. This tlist is suitable for use as an input argument of the output function. This tlist may be enriched by children (specialize) optimization methods.

this
The current object.

[ this , p ] = optimbase_proj2bnds ( this , x )
Returns a point, which is the projection of the given point into the bounds.

this
The current object.

x
the current point

this = optimbase_stoplog ( this , msg )
Prints the given stopping rule message if verbose termination is enabled. If verbose termination is disabled, does nothing.

this
The current object.

msg
the message to print

[this , terminate , status] = optimbase_terminate (this , previousfopt , currentfopt , previousxopt , currentxopt )
Returns 1 if the algorithm terminates. Returns 0 if the algorithm must continue. If the -verbose-termination option is enabled, messages are printed detailing the termination intermediate steps. The optimbase_terminate function takes into account the number of iterations, the number of evaluations of the cost function, the tolerance on x and the tolerance on f. See below in the section "Termination" for more details.

this
The current object.

previousfopt
the previous value of the cost function

currentfopt
the current value of the cost function

previousxopt
the previous x optimum

currentxopt
the current x optimum

terminate
%t if the algorithm must terminate, %f if the algorithm must continue

status
if terminate = %t, the detailed status of the termination, as a string. If terminate = %f, the status is "continue".

The following status are available :

"maxiter"
the maximum number of iterations, provided by the -maxiter option, is reached.

"maxfuneval"
the maximum number of function evaluations, provided by the -maxfunevals option, is reached

"tolf"
the tolerance on the function value is reached. This status is associated with the -tolfunmethod, -tolfunabsolute and -tolfunrelative options.

"tolx"
the tolerance on x is reached. This status is associated with the -tolxmethod, -tolxabsolute and -tolxrelative options.

this = optimbase_checkcostfun ( this )
Check that the cost function is correctly connected. Generate an error if there is one. Takes into account for the cost function at the initial guess x0 only. Checks that all values of the index argument are valid. If there are nonlinear constraints, check that the matrix has the correct shape.

This function requires at least one call to the cost function to make the necessary checks.

this
The current object.

[ this , isfeasible ] = optimbase_isinbounds ( this , x )
Returns isfeasible = %t if the given point satisfies bounds constraints. Returns isfeasible = %f if the given point is not in the bounds.

this
The current object.

isfeasible
a boolean

[ this , isfeasible ] = optimbase_isinnonlinconst ( this , x )
Returns isfeasible = %t if the given point satisfies the nonlinear constraints. Returns isfeasible = %f if the given point does not satisfy the nonlinear constraints.

this
The current object.

isfeasible
a boolean

# The cost function

The `-function` option allows to configure the cost function. The cost function is used, depending on the context, to compute the cost, the nonlinear inequality positive constraints, the gradient of the function and the gradient of the nonlinear inequality constraints.

The cost function can also be used to produce outputs and to terminate an optimization algorithm.

In the following, the variables are

- `f`: scalar, the objective,
- `g`: row matrix, the gradient of the objective,
- `c`: row matrix, the constraints,
- `gc`: matrix, the gradient of the constraints.

Each calling sequence of the `optimbase_function` function corresponds to a specific calling sequence of the user-provided cost function.

- If the -withderivatives is false and there is no nonlinear constraint, the calling sequence is

```
[ this , f , index ] = optimbase_function ( this , x , index )
```

  which corresponds to the cost functions

```
function [ f , index ] = costf ( x , index )
function [ f , index , data ] = costf ( x , index , data ) // if there is a -c
```

- If the -withderivatives is false and there are nonlinear constraints, the calling sequence is

```
[ this , f , c , index ] = optimbase_function ( this , x , index )
```

  which corresponds to the cost functions

```
function [ f , c , index ] = costf ( x , index )
function [ f , c , index , data ] = costf ( x , index , data ) // if there is
```

- If the -withderivatives is true and there is no nonlinear constraint, the calling sequence is

```
[ this , f , g , index ] = optimbase_function ( this , x , index )
```

  which corresponds to the cost functions

```
function [ f , g , index ] = costf ( x , index )
function [ f , g , index , data ] = costf ( x , index , data ) // if there is
```

- If the -withderivatives is true and there is are nonlinear constraints, the calling sequence is

```
[ this , f , g , c , gc , index ] = optimbase_function ( this , x , index )
```

  which corresponds to the cost functions

```
function [ f , g , c , gc , index ] = costf ( x , index )
```

```
function [ f , g , c , gc , index , data ] = costf ( x , index , data ) // if
```

Each calling sequence corresponds to a particular class of algorithms, including for example

- unconstrained, derivative-free algorithms,
- nonlinearily constrained, derivative-free algorithms,
- unconstrained, derivative-based algorithms,
- nonlinearilyconstrained, derivative-based algorithms,
- etc...

The current component is designed in order to be able to handle many situations.

The index input parameter has the following meaning.

- index = 1: nothing is to be computed, the user may display messages, for example
- index = 2: compute f
- index = 3: compute g
- index = 4: compute f and g
- index = 5: compute c
- index = 6: compute f and c
- index = 7: compute f, g, c and gc

The index output parameter has the following meaning.

- index > 0: everything is fine,
- index = 0: the optimization must stop,
- index < 0: one function could not be avaluated.

# The output function

The option -outputcommand allows to configure a command which is called back at the start of the optimization, at each iteration and at the end of the optimization.

The output function must have the following header

```
function outputcmd(state, data, myobj)
```

where

state
: a string representing the current state of the algorithm. Available values are "init", "iter", "done".

data
: a tlist containing at least the following entries

  x
  : the current optimum

fval
: the current function value

iteration
: the current iteration index

funccount
: the number of function evaluations

myobj
: a user-defined parameter.

This input parameter is defined with the -outputcommandarg option.

The output function may be used when debugging the specialized optimization algorithm, so that a verbose logging is produced. It may also be used to write one or several report files in a specialized format (ASCII, LaTeX, Excel, Hdf5, etc...). The user-defined parameter may be used in that case to store file names or logging options.

The data tlist argument may contain more fields than the current presented ones. These additionnal fields may contain values which are specific to the specialized algorithm, such as the simplex in a Nelder-Mead method, the gradient of the cost function in a BFGS method, etc...

# Termination

The current component takes into account for several generic termination criterias. Specialized termination criterias should be implemented in specialized optimization algorithms, by calling the optimbase_termination function and adding external criterias, rather than by modification of this function.

The optimbase_terminate function uses a set of rules to compute if the termination occurs, which leads to an optimization status which is equal to one of the following : "continue", "maxiter", "maxfunevals", "tolf", "tolx". The set of rules is the following.

- By default, the status is "continue" and the terminate flag is 0.
- The number of iterations is examined and compared to the -maxiter option : if the following condition

  ```
  iterations >= maxiter
  ```

  is true, then the status is set to "maxiter" and terminate is set to %t.
- The number of function evaluations and compared to the -maxfunevals option is examined : if the following condition

  ```
  funevals >= maxfunevals
  ```

  is true, then the status is set to "maxfuneval" and terminate is set to %t.
- The tolerance on function value is examined depending on the value of the -tolfunmethod.

  "disabled"
  : then the tolerance on f is just skipped.

  "enabled"
  : if the following condition

```
abs(currentfopt) < tolfunrelative * abs(previousfopt) + tolfunabsolute
```

is true, then the status is set to "tolf" and terminate is set to %t.

The relative termination criteria on the function value works well if the function value at optimum is near zero. In that case, the function value at initial guess fx0 may be used as previousfopt.

The absolute termination criteria on the function value works if the user has an accurate idea of the optimum function value.

- The tolerance on x is examined depending on the value of the -tolxmethod.

  %f
  then the tolerance on x is just skipped.

  %t
  if the following condition

```
norm(currentxopt - previousxopt) < tolxrelative * norm(currentxopt) + tolxa
```

  is true, then the status is set to "tolx" and terminate is set to %t.

  The relative termination criteria on x works well if x at optimum is different from zero. In that case, the condition measures the distance between two iterates.

  The absolute termination criteria on x works if the user has an accurate idea of the scale of the optimum x. If the optimum x is near 0, the relative tolerance will not work and the absolute tolerance is more appropriate.

# Example : Setting up an optimization

In the following example, one searches the minimum of the 2D Rosenbrock function. One begins by defining the function "rosenbrock" which computes the Rosenbrock function. The traditionnal initial guess [-1.2 1.0] is used. The initial simplex is computed along the axes with a length equal to 0.1. The Nelder-Mead algorithm with variable simplex size is used. The verbose mode is enabled so that messages are generated during the algorithm. After the optimization is performed, the optimum is retrieved with quiery features.

```
function [ f , index ] = rosenbrock ( x , index )
  f = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction

opt = optimbase_new();
opt = optimbase_configure(opt,"-numberofvariables",2);
nbvar = optimbase_cget(opt,"-numberofvariables");
opt = optimbase_configure(opt,"-function",rosenbrock);
[ this , f , index ] = optimbase_function ( opt , [0.0 0.0] , index );
opt = optimbase_destroy(opt);
```

# Authors

Michael Baudin, 2008-2009

# TODO

- manage equality constraints
- manage linear constraints
- manage quadratic objective
- manage linear objective
- manage linear inequality constraints
- manage non linear equality constraints
- manage linear equality constraints
- rename -outputcommand to -outputfunction

# Capítulo 4. Optimization simplex

# Nom

optimsimplex — Manage a simplex with arbitrary number of points.

```
[ newobj [, data] ] = optimsimplex_new ( coords , fun [, data] )
[ newobj [, data] ] = optimsimplex_new ( "axes" , x0 , fun , len [, data] )
[ newobj [, data] ] = optimsimplex_new ( "spendley" , x0 , fun , len [, data] )
[ newobj [, data] ] = optimsimplex_new ( "pfeffer" , x0 , fun , deltausual , de
[ newobj [, data] ] = optimsimplex_new ( "randbounds" , x0 , fun , boundsmin , 
[ newobj [, data] ] = optimsimplex_new ( "oriented" , simplex0 , fun [, data] )
this = optimsimplex_destroy (this)
this = optimsimplex_setall ( this , simplex )
this = optimsimplex_setallfv ( this , fv )
this = optimsimplex_setallx ( this , x )
this = optimsimplex_setfv ( this , ive , fv )
this = optimsimplex_setn ( this , n )
this = optimsimplex_setnbve ( this , nbve )
this = optimsimplex_setve ( this , ive , fv , x )
this = optimsimplex_setx ( this , ive , x )
simplex = optimsimplex_getall ( this )
fv = optimsimplex_getallfv ( this )
x = optimsimplex_getallx ( this )
fv = optimsimplex_getfv ( this , ive )
n = optimsimplex_getn ( this )
nbve = optimsimplex_getnbve ( this )
vertex = optimsimplex_getve ( this , ive )
x = optimsimplex_getx ( this , ive )
sicenter = optimsimplex_center ( this )
optimsimplex_check ( this )
[ this , data ] = optimsimplex_computefv ( this , fun , data )
df = optimsimplex_deltafv ( this )
dfm = optimsimplex_deltafvmax ( this )
m = optimsimplex_dirmat ( this )
m = optimsimplex_fvmean ( this )
sd = optimsimplex_fvstdev ( this )
v = optimsimplex_fvvariance ( this )
[ g , data ] = optimsimplex_gradientfv ( this , fun , method , data )
optimsimplex_print ( this )
[ r , data ] = optimsimplex_reflect ( this , fun , data )
[ this , data ] = optimsimplex_shrink ( this , fun , sigma , data )
ssize = optimsimplex_size ( this , method )
this = optimsimplex_sort ( this )
str = optimsimplex_tostring ( this )
cen = optimsimplex_xbar ( this , iexcl )
```

# Purpose

The goal of this component is to provide a building block for optimization algorithms based on a simplex. The optimsimplex package may be used in the following optimization methods :

- the Spendley et al. simplex method,
- the Nelder-Mead method,
- the Box algorithm for constrained optimization,
- the multi-dimensional search by Virginia Torczon,
- etc ...

# Design

This toolbox is designed with Oriented Object ideas in mind.

# Features

The following is a list of features the Nelder-Mead prototype algorithm currently provides :

- Manage various simplex initializations
  - initial simplex given by user,
  - initial simplex computed with a length and along the coordinate axes,
  - initial regular simplex computed with Spendley et al. formula,
  - initial simplex computed by a small perturbation around the initial guess point,
  - initial simplex computed from randomized bounds.
- sort the vertices by increasing function values,
- compute the standard deviation of the function values in the simplex,
- compute the simplex gradient with forward or centered differences,
- shrink the simplex toward the best vertex,
- etc...

# Description

This set of commands allows to manage a simplex made of k>=n+1 points in a n-dimensional space. This component is the building block for a class of direct search optimization methods such as the Nelder-Mead algorithm or Torczon's Multi-Dimensionnal Search.

A simplex is designed as a collection of k>=n+1 vertices. Each vertex is made of a point and a function value at that point.

The simplex can be created with various shapes. It can be configured and quieried at will. The simplex can also be reflected or shrinked. The simplex gradient can be computed with a order 1 forward formula and with a order 2 centered formula.

The optimsimplex_new function allows to create a simplex. If vertices coordinates are given, there are registered in the simplex. If a function is provided, it is evaluated at each vertex. The optimsimplex_destroy function destroys the object and frees internal memory. Several functions allow to create a simplex with special shapes, including axes-by-axes (optimsimplex_axes), regular (optimsimplex_spendley), randomized bounds simplex with arbitrary nbve vertices (optimsimplex_randbounds) and an heuristical small variation around a given point (optimsimplex_pfeffer).

In the following functions, simplices and vertices are, depending on the functions either input or output arguments. The following general principle have been used to manage the storing of the coordinates of the points.

- The vertices are stored row by row, while the coordinates are stored column by column. This implies the following rules.
- The coordinates of a vertex are stored in a row vector, i.e. a 1 x n matrix where n is the dimension of the space.

- The function values are stored in a column vector, i.e. a nbve x 1 matrix where nbve is the number of vertices.

# Functions

The following functions are available.

newobj = optimsimplex_new ( )

Creates a new simplex object. All input arguments are optional. If no input argument is provided, returns an empty simplex object.

The following is a complete list of available calling sequences.

```
newobj = optimsimplex_new ( coords )
newobj = optimsimplex_new ( coords , fun )
[ newobj , data ] = optimsimplex_new ( coords , fun , data )
```

newobj

The new object.

coords

optional, matrix of point coordinates in the simplex.

The coords matrix is expected to be a nbve x n matrix, where n is the dimension of the space and nbve is the number of vertices in the simplex, with nbve>= n+1.

fun

optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

where `x` is a row vector.

data

optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

this = optimsimplex_new ( "axes" , x0 )

Creates a new simplex object so that it is computed axis by axis, with the given length.

The following is a complete list of available calling sequences.

```
this = optimsimplex_new ( "axes" , x0 , fun )
this = optimsimplex_new ( "axes" , x0 , fun , len )
```

```
[ this , data ] = optimsimplex_new ( "axes" , x0 , fun , len , data )
```

this
: The current simplex object.

x0
: the initial point, as a row vector.

fun
: optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

len
: optional, the length of the simplex. The default length is 1.0. If length is a value, that unique length is used in all directions. If length is a vector with n values, each length is used with the corresponding direction.

data
: optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

this = optimsimplex_new ( "pfeffer" , x0 )
: Creates a new simplex so that it is computed from Pfeffer's method, i.e. a relative delta for non-zero values and an absolute delta for zero values.

The following is a complete list of available calling sequences.

```
this = optimsimplex_new ( "pfeffer" , x0 , fun )
this = optimsimplex_new ( "pfeffer" , x0 , fun , deltausual )
this = optimsimplex_new ( "pfeffer" , x0 , fun , deltausual , deltazero )
[ this , data ] = optimsimplex_new ( "pfeffer" , x0 , fun , deltausual , del
```

this
: The current simplex object.

x0
: the initial point, as a row vector.

fun
: optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

deltausual
optional, the absolute delta for non-zero values. The default value is 0.05.

deltazero
optional, the absolute delta for zero values. The default value is 0.0075.

data
optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

this = optimsimplex_new ( "randbounds" , x0 , fun , boundsmin , boundsmax , nbpoints )
Creates a new simplex so that it is computed by taking the bounds into account with random scaling. The number of vertices in the simplex is arbitrary.

The following is a complete list of available calling sequences.

```
[ this , data ] = optimsimplex_new ( "randbounds" , x0 , fun , boundsmin , b
```

this
The current simplex object.

x0
the initial point, as a row vector. It is the first vertex in the simplex.

fun
optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

boundsmin
array of minimum bounds

boundsmax
array of maximum bounds

Each component ix =1,n of the vertex #k = 2,nbve is computed from the formula :

```
x ( k , ix ) = boundsmin( ix ) + rand() * (boundsmax( ix ) - boundsmin( i
```

nbpoints
: total number of points in the simplex

data
: optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

this = optimsimplex_new ( "spendley" , x0 )
: Creates a new simplex so that it is computed from Spendley's et al. method, i.e. a regular simplex made of nbve = n+1 vertices.

The following is a complete list of available calling sequences.

```
this = optimsimplex_new ( "spendley" , x0 , fun )
this = optimsimplex_new ( "spendley" , x0 , fun , len )
[ this , data ] = optimsimplex_new ( "spendley" , x0 , fun , len, data )
```

this
: The current simplex object.

x0
: the initial point, as a row vector.

fun
: optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

len
: optional, the length of the simplex. The default length is 1.0.

data
: optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

this = optimsimplex_new ( "oriented" , simplex0 , fun )

Returns a new oriented simplex, in sorted order. The new simplex has the same sigma- length of the base simplex, but is "oriented" depending on the function value. The created simplex may be used, as Kelley suggests, for a restart of Nelder-Mead algorithm.

The following is a complete list of available calling sequences.

```
[ this , data ] = optimsimplex_new ( "oriented" , simplex0 , fun, data )
```

this

The current simplex object.

simplex0

the base simplex

fun

optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

data

optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

this = optimsimplex_destroy (this)

Destroy the given object.

this

The current simplex object.

this = optimsimplex_setall ( this , simplex )

Set all the coordinates and and the function values of all the vertices.

this

The current simplex object.

simplex

the simplex to set.

The given matrix is expected to be a nbve x n+1 matrix where n is the dimension of the space, nbve is the number of vertices and with the following content (where the data is organized by row with function value first, and x coordinates)

- simplex(k,1) is the function value of the vertex #k, with k = 1 , nbve

- simplex(k,2:n+1) is the coordinates of the vertex #k, with k = 1 , nbve

this = optimsimplex_setallfv ( this , fv )

Set all the function values of all the vertices. The vertex #k is expected to be stored in fv(k) with k = 1 , nbve. The fv input argument is expected to be a row vector.

this
: The current simplex object.

fv
: the array of function values

this = optimsimplex_setallx ( this , x )

Set all the coordinates of all the vertices. The vertex #k is expected to be stored in x(k,1:n) with k = 1 , nbve

this
: The current simplex object.

x
: the coordinates of the vertices.

this = optimsimplex_setfv ( this , ive , fv )

Set the function value at given index and // returns an updated simplex.

this
: The current simplex object.

ive
: vertex index

fv
: the function value

this = optimsimplex_setn ( this , n )

Set the dimension of the space of the simplex.

this
: The current simplex object.

n
: the dimension

this = optimsimplex_setnbve ( this , nbve )

Set the number of vertices of the simplex.

this
: The current simplex object.

nbve
: the number of vertices

this = optimsimplex_setve ( this , ive , fv , x )

Sets the coordinates of the vertex and the function value at given index in the current simplex.

this
: The current simplex object.

ive
: the vertex index

fv
: the function value

x
the coordinates of the point, as a row vector.

this = optimsimplex_setx ( this , ive , x )
Set the coordinates of the vertex at given index, as a row vector, into the current simplex.

this
The current simplex object.

ive
the vertex index

x
the coordinates of the point, as a row vector

simplex = optimsimplex_getall ( this )
Returns all the coordinates of all the vertices and the function values in the same matrix.

this
The current simplex object.

simplex
the simplex data.

The simplex matrix has size nbve x n+1, and is organized by row by row as follows :

- simplex(k,1) is the function value of the vertex #k, with k = 1 , nbve
- simplex(k,2:n+1) is the coordinates of the vertex #k, with k = 1 , nbve

fv = optimsimplex_getallfv ( this )
Returns all the function values of all the vertices, as a row vector.

this
The current simplex object.

fv
The array of function values. The function value of vertex #k is stored in fv(k) with k = 1 , nbve.

x = optimsimplex_getallx ( this )
Returns all the coordinates of all the vertices.

this
The current simplex object.

x
the coordinates.

The vertex #k is stored in x(k,1:n) with k = 1 , nbve.

fv = optimsimplex_getfv ( this , ive )
Returns the function value at given index

this
The current simplex object.

ive
the vertex index

n = optimsimplex_getn ( this )
Returns the dimension of the space of the simplex

this
The current simplex object.

nbve = optimsimplex_getnbve ( this )
Returns the number of vertices in the simplex.

this
The current simplex object.

vertex = optimsimplex_getve ( this , ive )
Returns the vertex at given index as a tlist, with fields n, x and fv

this
The current simplex object.

ive
the vertex index

x = optimsimplex_getx ( this , ive )
Returns the coordinates of the vertex at given index, as a row vector.

this
The current simplex object.

ive
the vertex index

sicenter = optimsimplex_center ( this )
Returns the center of the given simplex

this
The current simplex object.

optimsimplex_check ( this )
Check the consistency of the internal data. Generates an error if necessary.

this
The current simplex object.

[ this , data ] = optimsimplex_computefv ( this , fun , data )
Set the values of the function at vertices points.

this
The current simplex object.

fun
optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

data
optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

df = optimsimplex_deltafv ( this )
Returns the vector of difference of function values with respect to the function value at vertex #1.

this
The current simplex object.

dfm = optimsimplex_deltafvmax ( this )
Returns the difference of function value between the high and the low vertices. It is expected that the vertex #1 is associated with the smallest function value and that the vertex #nbve is associated with the highest function value. Since vertices are ordered, the high is greater than the low.

this
The current simplex object.

m = optimsimplex_dirmat ( this )
Returns the n x n matrix of simplex directions i.e. the matrix of differences of vertices coordinates with respect to the vertex #1.

this
The current simplex object.

m = optimsimplex_fvmean ( this )
Returns the mean of the function value on the simplex.

this
The current simplex object.

sd = optimsimplex_fvstdev ( this )
Returns the standard deviation of the function value on the simplex.

this
The current simplex object.

v = optimsimplex_fvvariance ( this )
Returns the variance of the function value on the simplex.

this
The current simplex object.

g = optimsimplex_gradientfv ( this , fun , method )
Returns the simplex gradient of the function.

The following is a complete list of available calling sequences.

```
g = optimsimplex_gradientfv ( this , fun , method )
[ g , data ] = optimsimplex_gradientfv ( this , fun , method , data )
```

this
The current simplex object.

fun
optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

method
optional, the method to use to compute the simplex gradient. Two methods are available : "forward" or "centered". The forward method uses the current simplex to compute the simplex gradient. The centered method creates an intermediate reflected simplex and computes the average.

If not provided, the default method is "forward".

data
optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

optimsimplex_print ( this )
Display the current simplex, with coordinates and function values.

this
The current simplex object.

[ r , data ] = optimsimplex_reflect ( this , fun , data )
Returns a new simplex by reflexion of current simplex, by reflection with respect to the first vertex in the simplex. This move is used in the centered simplex gradient.

this
The current simplex object.

fun
optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

data
optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing

the function value. This feature may be used, for example, to count the number of times that the function has been called.

[ this , data ] = optimsimplex_shrink ( this , fun , sigma , data )

Shrink the simplex with given coefficient sigma and returns an updated simplex. The shrink is performed with respect to the first point in the simplex.

this

The current simplex object.

fun

optional, the function to compute at vertices.

The function is expected to have the following input and output arguments :

```
function y = myfunction (x)
```

sigma

optional, the shrinkage coefficient. The default value is 0.5.

data

optional, user-defined data passed to the function.

If data is provided, it is passed to the callback function both as an input and output argument. In that case, the function must have the following header :

```
function [ y , data ] = myfunction ( x , data )
```

The data input parameter may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

ssize = optimsimplex_size ( this , method )

Returns the size of the simplex.

this

The current simplex object.

method

optional, the method to use to compute the size.

The available methods are the following :

- "sigmaplus" (this is the default)

  The sigmamplus size is the maximum 2-norm length of the vector from each vertex to the first vertex. It requires one loop over the vertices.

- "sigmaminus"

  The sigmaminus size is the minimum 2-norm length of the vector from each vertex to the first vertex. It requires one loop over the vertices.

- "Nash"

  The "Nash" size is the sum of the norm of the norm-1 length of the vector from the given vertex to the first vertex. It requires one loop over the vertices.

- "diameter"

  The diameter is the maximum norm-2 length of all the edges of the simplex. It requires 2 nested loops over the vertices.

this = optimsimplex_sort ( this )
: Sorts the simplex with increasing function value order so that the smallest function value is at vertex #1

this
: The current simplex object.

str = optimsimplex_tostring ( this )
: Returns the current simplex as a string.

this
: The current simplex object.

cen = optimsimplex_xbar ( this , iexcl )
: Returns the center of n vertices, by excluding the vertex with index iexcl. Returns a row vector.

this
: The current simplex object.

iexcl
: the index of the vertex to exclude in center computation. The default value of iexcl is the number of vertices : in that case, if the simplex is sorted in increasing function value order, the worst vertex is excluded.

# Example : Creating a simplex with given vertices coordinates

In the following example, one creates a simplex with known vertices coordinates. The function values at the vertices are unset.

```
coords = [
    0.    0.
    1.    0.
    0.    1.
];
s1 = optimsimplex_new ( coords );
computed = optimsimplex_getallx ( s1 );
computed = optimsimplex_getn(s1);
computed = optimsimplex_getnbve (s1);
s1 = optimsimplex_destroy(s1);
```

# Example : Creating a simplex with randomized bounds

In the following example, one creates a simplex with in the 2D domain [-5 5]^2, with [-1.2 1.0] as the first vertex. One uses the randomized bounds method to generate a simplex with 5 vertices. The function takes an additionnal argument mystuff, which is counts the number of times the function is called. After the creation of the simplex, the value of mystuff.nb is 5, which is the expected result because there is one function call by vertex.

```
function y = rosenbrock (x)
  y = 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
endfunction
function [ y , mystuff ] = mycostf ( x , mystuff )
  y = rosenbrock(x);
  mystuff.nb = mystuff.nb + 1
endfunction

mystuff = tlist(["T_MYSTUFF","nb"]);
mystuff.nb = 0;
s1 = optimsimplex_new ();
[ s1 , mystuff ] = optimsimplex_randbounds ( s1 , x0 = [-1.2 1.0], fun = mycost
  boundsmin = [-5.0 -5.0] , boundsmax = [5.0 5.0], nbve=5 , data = mystuff );
mprintf("Function evaluations: %d\n",mystuff.nb)
s1 = optimsimplex_destroy ( s1 );
```

# Initial simplex strategies

In this section, we analyse the various initial simplex which are provided in this component.

It is known that direct search methods based on simplex designs are very sensitive to the initial simplex. This is why the current component provides various ways to create such an initial simplex.

The first historical simplex-based algorithm is the one presented in "Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation" by W. Spendley, G. R. Hext and F. R. Himsworth. The "spendley" simplex creates the regular simplex which is presented in the paper.

The "randbounds" simplex is due to M.J. Box in "A New Method of Constrained Optimization and a Comparison With Other Methods".

Pfeffer's method is an heuristic which is presented in "Global Optimization Of Lennard-Jones Atomic Clusters" by Ellen Fan. It is due to L. Pfeffer at Stanford and it is used in fminsearch.

# TODO

- implement reflection and expansion as in multidimensional search by Torczon
- turn optimsimplex_reflect into a proper constructor, i.e. an option of the the optimsimplex_new function. Another possibility is to reflect "in place" as in the optimsimplex_shrink function (but in this case we must provide a "copy" constructor from current simplex before reflecting it).

# Authors

Michael Baudin - INRIA - 2008-2009

Michael Baudin - Digiteo - 2009

# Bibliography

“Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation”, Spendley, W. and Hext, G. R. and Himsworth, F. R., American Statistical Association and American Society for Quality, 1962

"A Simplex Method for Function Minimization", Nelder, J. A. and Mead, R. The Computer Journal, January, 1965, 308--313

"A New Method of Constrained Optimization and a Comparison With Other Methods", M. J. Box, The Computer Journal 1965 8(1):42-52, 1965 by British Computer Society

"Iterative Methods for Optimization", C.T. Kelley, 1999, Chapter 6., section 6.2

"Compact Numerical Methods For Computers - Linear Algebra and Function Minimization", J.C. Nash, 1990, Chapter 14. Direct Search Methods

"Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation", W. Spendley, G. R. Hext, F. R. Himsworth, Technometrics, Vol. 4, No. 4 (Nov., 1962), pp. 441-461, Section 3.1

"A New Method of Constrained Optimization and a Comparison With Other Methods", M. J. Box, The Computer Journal 1965 8(1):42-52, 1965 by British Computer Society

“Detection and Remediation of Stagnation in the Nelder--Mead Algorithm Using a Sufficient Decrease Condition”, SIAM J. on Optimization, Kelley,, C. T., 1999

" Multi-Directional Search: A Direct Search Algorithm for Parallel Machines", by E. Boyd, Kenneth W. Kennedy, Richard A. Tapia, Virginia Joanne Torczon,, Virginia Joanne Torczon, 1989, Phd Thesis, Rice University

"Grid Restrained Nelder-Mead Algorithm", Árpád B#rmen, Janez Puhan, Tadej Tuma, Computational Optimization and Applications, Volume 34 , Issue 3 (July 2006), Pages: 359 - 375

"A convergent variant of the Nelder-Mead algorithm", C. J. Price, I. D. Coope, D. Byatt, Journal of Optimization Theory and Applications, Volume 113 , Issue 1 (April 2002), Pages: 5 - 19,

"Global Optimization Of Lennard-Jones Atomic Clusters", Ellen Fan, Thesis, February 26, 2002, McMaster University

# Name

NDcost — generic external for optim computing gradient using finite differences

```
[f,g,ind]=NDcost(x,ind,fun,varargin)
```

# Parameters

x
: real vector or matrix

ind
: integer parameter (see optim)

fun
: Scilab function with calling sequence `F=fun(x,varargin)` varargin may be use to pass parameters `p1,...pn`

f
: criterion value at point `x` (see optim)

g
: gradient value at point `x` (see optim)

# Description

This function can be used as an external for `optim` to minimize problem where gradient is too complicated to be programmed. only the function `fun` which computes the criterion is required.

This function should be used as follow: `[f,xopt,gopt]=optim(list(NDcost,fun,p1,...pn),x0,...)`

# Examples

```
// example #1 (a simple one)
//function to minimize
function f=rosenbrock(x,varargin)
  p=varargin(1)
  f=1+sum( p*(x(2:$)-x(1:$-1)^2)^2 + (1-x(2:$))^2)
endfunction

x0=[1;2;3;4];
[f,xopt,gopt]=optim(list(NDcost,rosenbrock,200),x0)

// example #2: This example (by Rainer von Seggern) shows a quick (*) way to
//             identify the parameters of a linear differential equation with
//             the help of scilab.
//             The model is a simple damped (linear) oscillator:
//
//               x''(t) + c x'(t) + k x(t) = 0 ,
//
// and we write it as a system of two differential equations of first
// order with y(1) = x, and y(2) = x':
//
//     dy1/dt = y(2)
//     dy2/dt = -c*y(2) -k*y(1).
//
// We suppose to have m measurements of x (that is y(1)) at different times
```

```
// t_obs(1), ..., t_obs(m) called x_obs(1), ..., x_obs(m)  (in this example
// these measuresments will be simulated), and we want to identify the paramete
// c and k by minimizing the sum of squared errors between x_obs and y1(t_obs,p
//
// (*) This method is not the most efficient but it is easy to implement.
//
function dy = DEQ(t,y,p)
  // The rhs of our first order differential equation system.
  c =p(1);k=p(2)
  dy=[y(2);-c*y(2)-k*y(1)]
endfunction

function y=uN(p, t, t0, y0)
  // Numerical solution obtained with ode. (In this linear case an exact analyt
  // solution can easily be found, but ode would also work for "any" system.)
  // Note: the ode output must be an approximation of the solution at
  //       times given in the vector t=[t(1),...,t($)]
  y = ode(y0,t0,t,list(DEQ,p))
endfunction

function r = cost_func(p, t_obs, x_obs, t0, y0)
  // This is the function to be minimized, that is the sum of the squared
  // errors between what gives the model and the measuments.
  sol = uN(p, t_obs, t0, y0)
  e = sol(1,:) - x_obs
  r = sum(e.*e)
endfunction

// Data
y0 = [10;0]; t0 = 0; // Initial conditions y0 for initial time t0.
T = 30;  // Final time for the measurements.

// Here we simulate experimental data, (from which the parameters
// should be identified).
pe = [0.2;3];  // Exact parameters
m = 80;  t_obs = linspace(t0+2,T,m); // Observation times
// Noise: each measurement is supposed to have a (gaussian) random error
// of mean 0 and std deviation proportional to the magnitude
// of the value (sigma*|x_exact(t_obs(i))|).
sigma = 0.1;
y_exact = uN(pe, t_obs, t0, y0);
x_obs = y_exact(1,:) + grand(1,m,"nor",0, sigma).*abs(y_exact(1,:));

// Initial guess parameters
p0 = [0.5 ; 5];

// The value of the cost function before optimization:
cost0 = cost_func(p0, t_obs, x_obs, t0, y0);
mprintf("\n\r The value of the cost function before optimization = %g \n\r",...

// Solution with optim
[costopt,popt]=optim(list(NDcost,cost_func, t_obs, x_obs, t0, y0),p0,...
                                                        'ar',40,40,1e-3);

mprintf("\n\r The value of the cost function after optimization  = %g",costopt)
mprintf("\n\r The identified values of the parameters: c = %g, k = %g \n\r",...
                                                              popt(1),popt(2))
```

```
// A small plot:
t = linspace(0,T,400);
y = uN(popt, t, t0, y0);
clf();
plot2d(t',y(1,:)',style=5)
plot2d(t_obs',x_obs(1,:)',style=-5)
legend(["model","measurements"]);
xtitle("Least square fit to identify ode parameters")
```

# See Also

optim, external, derivative

# Name

aplat — Flattens a list.

```
[lf,ind] = aplat(l,r)
```

# Parameters

l
: a list

r
: an optional flat list

lf
: a flat list (a single hierachical level)

ind
: a list, each entry give the path of the corresponding lf entry in the original list

# Description

Creates a flat list, built with the initial l list leaves and if given prepended by the r list entries

# Examples

```
[lf,ind]=aplat(list(1,2,list([3,1],'xxx',list([3,2,1]))))
```

# See Also

recons

# Authors

F.D. and S.S., INRIA

# Name

datafit — Parameter identification based on measured data

```
[p,err]=datafit([imp,] G [,DG],Z [,W],[contr],p0,[algo],[df0,[mem]],
[work],[stop],['in'])
```

# Parameters

imp
: scalar argument used to set the trace mode. `imp=0` nothing (execpt errors) is reported, `imp=1` initial and final reports, `imp=2` adds a report per iteration, `imp>2` add reports on linear search. Warning, most of these reports are written on the Scilab standard output.

G
: function descriptor (e=G(p,z), e: ne x 1, p: np x 1, z: nz x 1)

DG
: partial of G wrt p function descriptor (optional; S=DG(p,z), S: ne x np)

Z
: matrix [z_1,z_2,...z_n] where z_i (nz x 1) is the ith measurement

W
: weighting matrix of size ne x ne (optional; defaut no ponderation)

contr
: `'b',binf,bsup` with `binf` and `bsup` real vectors with same dimension as `p0`. `binf` and `bsup` are lower and upper bounds on `p`.

p0
: initial guess (size np x 1)

algo
: `'qn'` or `'gc'` or `'nd'` . This string stands for quasi-Newton (default), conjugate gradient or non-differentiable respectively. Note that `'nd'` does not accept bounds on `x` ).

df0
: real scalar. Guessed decreasing of `f` at first iteration. (`df0=1` is the default value).

mem :
: integer, number of variables used to approximate the Hessian, (`algo='gc' or 'nd'`). Default value is around 6.

stop
: sequence of optional parameters controlling the convergence of the algorithm. `stop= 'ar',nap, [iter [,epsg [,epsf [,epsx]]]]`

  "ar"
  : reserved keyword for stopping rule selection defined as follows:

  nap
  : maximum number of calls to `fun` allowed.

  iter
  : maximum number of iterations allowed.

  epsg
  : threshold on gradient norm.

epsf
threshold controlling decreasing of `f`

epsx
threshold controlling variation of `x`. This vector (possibly matrix) of same size as `x0` can be used to scale `x`.

"in"
reserved keyword for initialization of parameters used when `fun` in given as a Fortran routine (see below).

p
Column vector, optimal solution found

err
scalar, least square error.

# Description

`datafit` is used for fitting data to a model. For a given function `G(p,z)`, this function finds the best vector of parameters `p` for approximating `G(p,z_i)=0` for a set of measurement vectors `z_i`. Vector `p` is found by minimizing `G(p,z_1)'WG(p,z_1)+G(p,z_2)'WG(p,z_2)+... +G(p,z_n)'WG(p,z_n)`

`datafit` is an improved version of `fit_dat`.

# Examples

```
//generate the data
function y=FF(x,p),y=p(1)*(x-p(2))+p(3)*x.*x,endfunction
X=[];Y=[];
pg=[34;12;14] //parameter used to generate data
for x=0:.1:3, Y=[Y,FF(x,pg)+100*(rand()-.5)];X=[X,x];end
Z=[Y;X];

//The criterion function
function e=G(p,z),
  y=z(1),x=z(2);
  e=y-FF(x,p),
endfunction

//Solve the problem
p0=[3;5;10]
[p,err]=datafit(G,Z,p0);

scf(0);clf()
plot2d(X,FF(X,pg),5) //the curve without noise
plot2d(X,Y,-1)  // the noisy data
plot2d(X,FF(X,p),12) //the solution

//the gradient of the criterion function
function s=DG(p,z),
  a=p(1),b=p(2),c=p(3),y=z(1),x=z(2),
  s=-[x-b,-a,x*x]
endfunction

[p,err]=datafit(G,DG,Z,p0);
```

```
scf(1);clf()
plot2d(X,FF(X,pg),5) //the curve without noise
plot2d(X,Y,-1)  // the noisy data
plot2d(X,FF(X,p),12) //the solution

// Add some bounds on the estimate of the parameters
// We want positive estimation (the result will not change)
[p,err]=datafit(G,DG,Z,'b',[0;0;0],[%inf;%inf;%inf],p0,algo='gc');
scf(1);clf()
plot2d(X,FF(X,pg),5) //the curve without noise
plot2d(X,Y,-1)  // the noisy data
plot2d(X,FF(X,p),12) //the solution
```

# See Also

lsqrsolve, optim, leastsq

# Name

derivative — approximate derivatives of a function

```
derivative(F,x)
[J [,H]] = derivative(F,x [,h ,order ,H_form ,Q])
```

# Parameters

F

a Scilab function F: `R^n --> R^m` or a `list(F,p1,...,pk)`, where F is a scilab function in the form `y=F(x,p1,...,pk)`, p1, ..., pk being any scilab objects (matrices, lists,...).

x

real column vector of dimension n.

h

(optional) real, the stepsize used in the finite difference approximations.

order

(optional) integer, the order of the finite difference formula used to approximate the derivatives (order = 1,2 or 4, default is order=2 ).

H_form

(optional) string, the form in which the Hessean will be returned. Possible forms are:

H_form='default'

H is a m x (`n^2`) matrix ; in this form, the k-th row of H corresponds to the Hessean of the k-th component of F, given as the following row vector :

$$\left( \frac{d(\mathrm{grad}(F_k))}{\mathrm{d}x_1}, \ldots, \frac{d(\mathrm{grad}(F_k))}{\mathrm{d}x_n} \right)$$

((grad(F_k) being a row vector).

H_form='blockmat' :

H is a (mxn) x n block matrix : the classic Hessean matrices (of each component of F) are stacked by row (H = [H1 ; H2 ; ... ; Hm] in scilab syntax).

H_form='hypermat' :

H is a n x n matrix for m=1, and a n x n x m hypermatrix otherwise. H(:,:,k) is the classic Hessean matrix of the k-th component of F.

Q

(optional) real matrix, orthogonal (default is eye(n,n)). Q is added to have the possibility to remove the arbitrariness of using the canonical basis to approximate the derivatives of a function and it should be an orthogonal matrix. It is not mandatory but better to recover the derivative as you need the inverse matrix (and so simply Q' instead of inv(Q)).

J

approximated Jacobian

H

approximated Hessian

# Description

Numerical approximation of the first and second derivatives of a function F: `R^n --> R^m` at the point x. The Jacobian is computed by approximating the directional derivatives of the components of

F in the direction of the columns of Q. (For m=1, v=Q(:,k) : grad(F(x))*v = Dv(F(x)).) The second derivatives are computed by composition of first order derivatives. If H is given in its default form the Taylor series of F(x) up to terms of second order is given by :

$$F(x+\mathrm{dx}) = F(x) + J(x)\cdot \mathrm{dx} + \frac{1}{2}\cdot \mathrm{dx}^{t}\cdot H(x)\cdot \mathrm{dx} + \cdots$$

(([J,H]=derivative(F,x,H_form='default'), J=J(x), H=H(x).)

# Performances

If the problem is correctly scaled, increasing the accuracy reduces the total error but requires more function evaluations. The following list presents the number of function evaluations required to compute the Jacobian depending on the order of the formula and the dimension of `x`, denoted by `n`:

- `order=1`, the number of function evaluations is `n+1`,
- `order=2`, the number of function evaluations is `2n`,
- `order=4`, the number of function evaluations is `4n`.

Computing the Hessian matrix requires square the number of function evaluations, as detailed in the following list.

- `order=1`, the number of function evaluations is `(n+1)^2`,
- `order=2`, the number of function evaluations is `4n^2`,
- `order=4`, the number of function evaluations is `16n^2`.

# Remarks

The step size h must be small to get a low error but if it is too small floating point errors will dominate by cancellation. As a rule of thumb, do not change the default step size. To work around numerical difficulties one may also change the order and/or choose different orthogonal matrices Q (the default is eye(n,n)), especially if the approximate derivatives are used in optimization routines. All the optional arguments may also be passed as named arguments, so that one can use calls in the form :

```
derivative(F, x, H_form = "hypermat")
derivative(F, x, order = 4) etc.
```

# Examples

```
function y=F(x)
  y=[sin(x(1)*x(2))+exp(x(2)*x(3)+x(1)) ; sum(x.^3)];
endfunction

function y=G(x,p)
  y=[sin(x(1)*x(2)*p)+exp(x(2)*x(3)+x(1)) ; sum(x.^3)];
endfunction

x=[1;2;3];
[J,H]=derivative(F,x,H_form='blockmat')

n=3;
```

```
// form an orthogonal matrix :
Q = qr(rand(n,n))
// Test order 1, 2 and 4 formulas.
for i=[1,2,4]
  [J,H]=derivative(F,x,order=i,H_form='blockmat',Q=Q);
  mprintf("order= %d \n",i);
  H,
end

p=1;
h=1e-3;
[J,H]=derivative(list(G,p),x,h,2,H_form='hypermat');
H
[J,H]=derivative(list(G,p),x,h,4,Q=Q);
H

// Taylor series example:
dx=1e-3*[1;1;-1];
[J,H]=derivative(F,x);
F(x+dx)
F(x+dx)-F(x)
F(x+dx)-F(x)-J*dx
F(x+dx)-F(x)-J*dx-1/2*H*(dx .*. dx)

// A trivial example
function y=f(x,A,p,w)
  y=x'*A*x+p'*x+w;
endfunction
// with Jacobian and Hessean given by J(x)=x'*(A+A')+p', and H(x)=A+A'.
A = rand(3,3);
p = rand(3,1);
w = 1;
x = rand(3,1);
[J,H]=derivative(list(f,A,p,w),x,h=1,H_form='blockmat')

// Since f(x) is quadratic in x, approximate derivatives of order=2 or 4 by fin
// differences should be exact for all h~=0. The apparent errors are caused by
// cancellation in the floating point operations, so a "big" h is choosen.
// Comparison with the exact matrices:
Je = x'*(A+A')+p'
He = A+A'
clean(Je - J)
clean(He - H)
```

# Accuracy issues

The `derivative` function uses the same step `h` whatever the direction and whatever the norm of `x`. This may lead to a poor scaling with respect to `x`. An accurate scaling of the step is not possible without many evaluations of the function. Still, the user has the possibility to compare the results produced by the `derivative` and the `numdiff` functions. Indeed, the `numdiff` function scales the step depending on the absolute value of `x`. This scaling may produce more accurate results, especially if the magnitude of `x` is large.

In the following Scilab script, we compute the derivative of an univariate quadratic function. The exact derivative can be computed analytically and the relative error is computed. In this rather extreme case, the `derivative` function produces no significant digits, while the `numdiff` function produces 6 significant digits.

```
 // Difference between derivative and numdiff when x is large
function y = myfunction (x)
  y = x*x;
endfunction
x = 1.e100;
fe = 2.0 * x;
fp = derivative(myfunction,x);
e = abs(fp-fe)/fe;
mprintf("Relative error with derivative: %e\n",e)
fp = numdiff(myfunction,x);
e = abs(fp-fe)/fe;
mprintf("Relative error with numdiff: %e\n",e)
```

The previous script produces the following output.

```
Relative error with derivative: 1.000000e+000
Relative error with numdiff: 7.140672e-006
```

In a practical situation, we may not know what is the correct numerical derivative. Still, we are warned that the numerical derivatives should be used with caution in this specific case.

# See Also

numdiff, derivat

# Authors

Rainer von Seggern, Bruno Pincon

# Name

fit_dat — Parameter identification based on measured data

```
[p,err]=fit_dat(G,p0,Z [,W] [,pmin,pmax] [,DG])
```

# Parameters

G
: Scilab function (e=G(p,z), e: nex1, p: npx1, z: nzx1)

p0
: initial guess (size npx1)

Z
: matrix [z_1,z_2,...z_n] where z_i (nzx1) is the ith measurement

W
: weighting matrix of size nexne (optional; default 1)

pmin
: lower bound on p (optional; size npx1)

pmax
: upper bound on p (optional; size npx1)

DG
: partial of G wrt p (optional; S=DG(p,z), S: nexnp)

# Description

`fit_dat` is used for fitting data to a model. For a given function G(p,z), this function finds the best vector of parameters p for approximating G(p,z_i)=0 for a set of measurement vectors z_i. Vector p is found by minimizing `G(p,z_1)'WG(p,z_1)+G(p,z_2)'WG(p,z_2)+...+G(p,z_n)'WG(p,z_n)`

# Examples

```
deff('y=FF(x)','y=a*(x-b)+c*x.*x')
X=[];Y=[];
a=34;b=12;c=14;for x=0:.1:3, Y=[Y,FF(x)+100*(rand()-.5)];X=[X,x];end
Z=[Y;X];
deff('e=G(p,z)','a=p(1),b=p(2),c=p(3),y=z(1),x=z(2),e=y-FF(x)')

[p,err]=fit_dat(G,[3;5;10],Z)

xset('window',0)
clf();
plot2d(X',Y',-1)
plot2d(X',FF(X)',5,'002')
a=p(1),b=p(2),c=p(3);plot2d(X',FF(X)',12,'002')

a=34;b=12;c=14;
deff('s=DG(p,z)','y=z(1),x=z(2),s=-[x-p(2),-p(1),x*x]')

[p,err]=fit_dat(G,[3;5;10],Z,DG)
```

```
xset('window',1)
clf();
plot2d(X',Y',-1)
plot2d(X',FF(X)',5,'002')
a=p(1),b=p(2),c=p(3);plot2d(X',FF(X)',12,'002')
```

# See Also

optim, datafit

# Name

fsolve — find a zero of a system of n nonlinear functions

```
[x [,v [,info]]]=fsolve(x0,fct [,fjac] [,tol])
```

# Parameters

x0
: real vector (initial value of function argument).

fct
: external (i.e function or list or string).

fjac
: external (i.e function or list or string).

tol
: real scalar. precision tolerance: termination occurs when the algorithm estimates that the relative error between x and the solution is at most tol. (`tol=1.d-10` is the default value).

x :
: real vector (final value of function argument, estimated zero).

v :
: real vector (value of function at x).

info
: termination indicator

  0
  : improper input parameters.

  1
  : algorithm estimates that the relative error between x and the solution is at most tol.

  2
  : number of calls to fcn reached

  3
  : tol is too small. No further improvement in the approximate solution x is possible.

  4
  : iteration is not making good progress.

# Description

find a zero of a system of n nonlinear functions in n variables by a modification of the powell hybrid method. Jacobian may be provided.

```
0 = fct(x) w.r.t x.
```

`fct` is an "external". This external returns `v=fct(x)` given `x`.

The simplest calling sequence for `fct` is:

```
```

```
[v]=fct(x).
```

If `fct` is a character string, it refers to a C or Fortran routine which must be linked to Scilab. Fortran calling sequence must be

```
fct(n,x,v,iflag)
integer n,iflag
double precision x(n),v(n)
```

and C Calling sequence must be

```
fct(int *n, double x[],double v[],int *iflag)
```

Incremental link is possible (help `link`).

`jac` is an "external". This external returns `v=d(fct)/dx (x)` given `x`.

The simplest calling sequence for `jac` is:

```
[v]=jac(x).
```

If `jac` is a character string, it refers to a to a C or Fortran routine which must be linked to Scilab calling sequences are the same as those for fct. Note however that v must be a nxn array.

# Examples

```
// A simple example with fsolve
a=[1,7;2,8];b=[10;11];

deff('[y]=fsol1(x)','y=a*x+b');
deff('[y]=fsolj1(x)','y=a');

[xres]=fsolve([100;100],fsol1);
a*xres+b

[xres]=fsolve([100;100],fsol1,fsolj1);
a*xres+b

// See routines/default/Ex-fsolve.f
[xres]=fsolve([100;100],'fsol1','fsolj1',1.e-7);
a*xres+b
```

For some starting points and some equations system, the fsolve method can fail. The fsolve method is a local search method. So, to have a good chance to find a solution to your equations system, you must ship, a good starting point to fsolve.

Here is an example on which fsolve can fail:

```
// Another example with fsolve
function F=feuler(x,r)
```

```
  F=x-r-dt*(x^2-x^3);
endfunction
function J=dFdx(x)  //definition de la derivee de F
   J=1-dt*(2*x-3*x^2);
endfunction

r = 0.04257794928862307 ;
dt = 10;

[x,v,info]=fsolve(r,list(feuler,r),dFdx); // fsolve don't find the solution
disp(v); // The residual
disp(info); // The termination indicator

[x,v,info]=fsolve(1,list(feuler,r),dFdx); // fsolve find the solution
disp(v); // The residual
disp(info); // The termination indicator

clf();x=linspace(0,1,1000);plot(x,feuler(x))
a=gca();a.grid=[5 5];
```

So, each time you use fsolve, be sure to check the termination indicator and the residual value to see if fsolve has converged.

# See Also

external, qpsolve, optim

# Name

karmarkar — karmarkar algorithm

```
[x1]=karmarkar(a,b,c,x0)
```

# Parameters

a
: matrix (n,p)

b
: n - vector

c
: p - vector

x0
: initial vector

eps
: threshold (default value : 1.d-5)

gamma
: descent step `0<gamma<1` , default value : 1/4

x1
: solution

crit
: value of c'*x1

# Description

Computes `x` which minimizes

$$\min c^t \cdot x$$
$$\text{with}\, a \cdot x = b$$
$$\text{and}\, x \geq 0$$

# Examples

```
n=10;p=20;
a=rand(n,p);
c=rand(p,1);
x0=abs(rand(p,1));
b=a*x0;
x1=karmarkar(a,b,c,x0);
```

# Name

leastsq — Solves non-linear least squares problems

```
[fopt,[xopt,[grdopt]]]=leastsq(fun, x0)
[fopt,[xopt,[grdopt]]]=leastsq(fun, dfun, x0)
[fopt,[xopt,[grdopt]]]=leastsq(fun, cstr, x0)
[fopt,[xopt,[grdopt]]]=leastsq(fun, dfun, cstr, x0)
[fopt,[xopt,[grdopt]]]=leastsq(fun, dfun, cstr, x0, algo)
[fopt,[xopt,[grdopt]]]=leastsq([imp], fun [,dfun] [,cstr],x0 [,algo],[df0,[mem]
```

# Parameters

fopt
: value of the function *f(x)=||fun(x)||^2* at `xopt`

xopt
: best value of `x` found to minimize *||fun(x)||^2*

grdopt
: gradient of *f* at `xopt`

fun
: a scilab function or a list defining a function from *R^n* to *R^m* (see more details in DESCRIPTION).

x0
: real vector (initial guess of the variable to be minimized).

dfun
: a scilab function or a string defining the Jacobian matrix of `fun` (see more details in DESCRIPTION).

cstr
: bound constraints on `x`. They must be introduced by the string keyword `'b'` followed by the lower bound `binf` then by the upper bound `bsup` (so `cstr` appears as `'b',binf,bsup` in the calling sequence). Those bounds are real vectors with same dimension than `x0` (-%inf and +%inf may be used for dimension which are unrestricted).

algo
: a string with possible values: `'qn'` or `'gc'` or `'nd'`. These strings stand for quasi-Newton (default), conjugate gradient or non-differentiable respectively. Note that `'nd'` does not accept bounds on `x`.

imp
: scalar argument used to set the trace mode. `imp=0` nothing (except errors) is reported, `imp=1` initial and final reports, `imp=2` adds a report per iteration, `imp>2` add reports on linear search. Warning, most of these reports are written on the Scilab standard output.

df0
: real scalar. Guessed decreasing of *||fun||^2* at first iteration. (`df0=1` is the default value).

mem
: integer, number of variables used to approximate the Hessean (second derivatives) of *f* when `algo='qn'`. Default value is around 6.

stop
: sequence of optional parameters controlling the convergence of the algorithm. They are introduced by the keyword `'ar'`, the sequence being of the form `'ar',nap, [iter [,epsg [,epsf [,epsx]]]]`

nap
: maximum number of calls to `fun` allowed.

iter
: maximum number of iterations allowed.

epsg
: threshold on gradient norm.

epsf
: threshold controlling decreasing of `f`

epsx
: threshold controlling variation of `x`. This vector (possibly matrix) of same size as `x0` can be used to scale `x`.

# Description

*fun* being a function from *R^n* to *R^m* this routine tries to minimize w.r.t. *x*, the function:

$$f(x) = \| \mathrm{fun}(x) \|^2 = \sum_{i=1}^{m} \mathrm{fun}_i^2(x)$$

which is the sum of the squares of the components of *fun*. Bound constraints may be imposed on `x`.

# How to provide fun and dfun

`fun` can be either a usual scilab function (case 1) or a fortran or a C routine linked to scilab (case 2). For most problems the definition of *fun* will need supplementary parameters and this can be done in both cases.

case 1:
: when `fun` is a Scilab function, its calling sequence must be: `y=fun(x [,opt_par1,opt_par2,...])`. When `fun` needs optional parameters it must appear as `list(fun,opt_par1,opt_par2,...)` in the calling sequence of `leastsq`.

case 2:
: when `fun` is defined by a Fortran or C routine it must appear as `list(fun_name,m [,opt_par1,opt_par2,...])` in the calling sequence of `leastsq`, `fun_name` (a string) being the name of the routine which must be linked to Scilab (see link). The generic calling sequences for this routine are:

```
In Fortran:    subroutine fun(m, n, x, params, y)
               integer m,n
               double precision x(n), params(*), y(m)

In C:          void fun(int *m, int *n, double *x, double *params, double *y
```

where `n` is the dimension of vector `x`, `m` the dimension of vector `y` (which must store the evaluation of *fun* at *x*) and `params` is a vector which contains the optional parameters *opt_par1*, *opt_par2*, ... (each parameter may be a vector, for instance if *opt_par1* has 3 components, the description of *opt_par2* begin from `params(4)` (fortran case), and from `params[3]` (C case), etc... Note that even if `fun` doesn't need supplementary parameters you must anyway write the fortran code with a `params` argument (which is then unused in the subroutine core).

In many cases it is adviced to provide the Jacobian matrix `dfun` (*dfun(i,j)=dfi/dxj*) to the optimizer (which uses a finite difference approximation otherwise) and as for `fun` it may be given as a usual scilab function or as a fortran or a C routine linked to scilab.

case 1:
when `dfun` is a scilab function, its calling sequence must be: `y=dfun(x [, optional parameters])` (notes that even if `dfun` needs optional parameters it must appear simply as `dfun` in the calling sequence of `leastsq`).

case 2:
when `dfun` is defined by a Fortran or C routine it must appear as `dfun_name` (a string) in the calling sequence of `leastsq` (`dfun_name` being the name of the routine which must be linked to Scilab). The calling sequences for this routine are nearly the same than for `fun`:

```
In Fortran:    subroutine dfun(m, n, x, params, y)
               integer m,n
               double precision x(n), params(*), y(m,n)

In C:          void fun(int *m, int *n, double *x, double *params, double *y
```

in the C case *dfun(i,j)=dfi/dxj* must be stored in `y[m*(j-1)+i-1]`.

# Remarks

Like datafit, `leastsq` is a front end onto the optim function. If you want to try the Levenberg-Marquard method instead, use lsqrsolve.

A least squares problem may be solved directly with the optim function ; in this case the function NDcost may be useful to compute the derivatives (see the NDcost help page which provides a simple example for parameters identification of a differential equation).

# Examples

```
// We will show different calling possibilities of leastsq on one (trivial) exa
// which is non linear but doesn't really need to be solved with leastsq (apply
// log linearizes the model and the problem may be solved with linear algebra).
// In this example we look for the 2 parameters x(1) and x(2) of a simple
// exponential decay model (x(1) being the unknow initial value and x(2) the
// decay constant):

function y = yth(t, x)
   y  = x(1)*exp(-x(2)*t)
endfunction

// we have the m measures (ti, yi):
m = 10;
tm = [0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5]';
ym = [0.79, 0.59, 0.47, 0.36, 0.29, 0.23, 0.17, 0.15, 0.12, 0.08]';
wm = ones(m,1); // measure weights (here all equal to 1...)

// and we want to find the parameters x such that the model fits the given
// datas in the least square sense:
//
//  minimize  f(x) = sum_i  wm(i)^2 ( yth(tm(i),x) - ym(i) )^2

// initial parameters guess
x0 = [1.5 ; 0.8];

// in the first examples, we define the function fun and dfun
// in scilab language
```

```
function e = myfun(x, tm, ym, wm)
   e = wm.*( yth(tm, x) - ym )
endfunction

function g = mydfun(x, tm, ym, wm)
   v = wm.*exp(-x(2)*tm)
   g = [v , -x(1)*tm.*v]
endfunction

// now we could call leastsq:

// 1- the simplest call
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),x0)

// 2- we provide the Jacobian
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),mydfun,x0)

// a small graphic (before showing other calling features)
tt = linspace(0,1.1*max(tm),100)';
yy = yth(tt, xopt);
clf()
plot2d(tm, ym, style=-2)
plot2d(tt, yy, style = 2)
legend(["measure points", "fitted curve"]);
xtitle("a simple fit with leastsq")

// 3- how to get some information (we use imp=1)
[f,xopt, gropt] = leastsq(1,list(myfun,tm,ym,wm),mydfun,x0)

// 4- using the conjugate gradient (instead of quasi Newton)
[f,xopt, gropt] = leastsq(1,list(myfun,tm,ym,wm),mydfun,x0,"gc")

// 5- how to provide bound constraints (not useful here !)
xinf = [-%inf,-%inf]; xsup = [%inf, %inf];
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),"b",xinf,xsup,x0) // without Jac
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),mydfun,"b",xinf,xsup,x0) // with

// 6- playing with some stopping parameters of the algorithm
//    (allows only 40 function calls, 8 iterations and set epsg=0.01, epsf=0.1)
[f,xopt, gropt] = leastsq(1,list(myfun,tm,ym,wm),mydfun,x0,"ar",40,8,0.01,0.1)


// 7 and 8: now we want to define fun and dfun in fortran then in C
//          Note that the "compile and link to scilab" method used here
//          is believed to be OS independant (but there are some requirements,
//          in particular you need a C and a fortran compiler, and they must
//          be compatible with the ones used to build your scilab binary).

// 7- fun and dfun in fortran

// 7-1/ Let 's Scilab write the fortran code (in the TMPDIR directory):
f_code = ["      subroutine myfun(m,n,x,param,f)"
          "*     param(i) = tm(i), param(m+i) = ym(i), param(2m+i) = wm(i)"
          "      implicit none"
          "      integer n,m"
          "      double precision x(n), param(*), f(m)"
          "      integer i"
          "      do i = 1,m"
```

```
          "         f(i) = param(2*m+i)*( x(1)*exp(-x(2)*param(i)) - param(m+i)
          "      enddo"
          "      end ! subroutine fun"
          ""
          "      subroutine mydfun(m,n,x,param,df)"
          "*     param(i) = tm(i), param(m+i) = ym(i), param(2m+i) = wm(i)"
          "      implicit none"
          "      integer n,m"
          "      double precision x(n), param(*), df(m,n)"
          "      integer i"
          "      do i = 1,m"
          "         df(i,1) =  param(2*m+i)*exp(-x(2)*param(i))"
          "         df(i,2) = -x(1)*param(i)*df(i,1)"
          "      enddo"
          "      end ! subroutine dfun"];

mputl(f_code,TMPDIR+'/myfun.f')

// 7-2/ compiles it. You need a fortran compiler !
names = ["myfun" "mydfun"]
flibname = ilib_for_link(names,"myfun.o",[],"f",TMPDIR+"/Makefile");

// 7-3/ link it to scilab (see link help page)
link(flibname,names,"f")

// 7-4/ ready for the leastsq call: be carreful don't forget to
//      give the dimension m after the routine name !
[f,xopt, gropt] = leastsq(list("myfun",m,tm,ym,wm),x0)  // without Jacobian
[f,xopt, gropt] = leastsq(list("myfun",m,tm,ym,wm),"mydfun",x0) // with Jacobia

// 8- last example: fun and dfun in C

// 8-1/ Let 's Scilab write the C code (in the TMPDIR directory):
c_code = ["#include <math.h>"
          "void myfunc(int *m,int *n, double *x, double *param, double *f)"
          "{"
          "  /*  param[i] = tm[i], param[m+i] = ym[i], param[2m+i] = wm[i] */"
          "  int i;"
          "  for ( i = 0 ; i &lt; *m ; i++ )"
          "    f[i] = param[2*(*m)+i]*( x[0]*exp(-x[1]*param[i]) - param[(*m)+i
          "  return;"
          "}"
          ""
          "void mydfunc(int *m,int *n, double *x, double *param, double *df)"
          "{"
          "  /*  param[i] = tm[i], param[m+i] = ym[i], param[2m+i] = wm[i] */"
          "  int i;"
          "  for ( i = 0 ; i &lt; *m ; i++ )"
          "    {"
          "      df[i] = param[2*(*m)+i]*exp(-x[1]*param[i]);"
          "      df[i+(*m)] = -x[0]*param[i]*df[i];"
          "    }"
          "  return;"
          "}"];

mputl(c_code,TMPDIR+'/myfunc.c')

// 8-2/ compiles it. You need a C compiler !
```

```
names = ["myfunc" "mydfunc"]
clibname = ilib_for_link(names,"myfunc.o",[],"c",TMPDIR+"/Makefile");

// 8-3/ link it to scilab (see link help page)
link(clibname,names,"c")

// 8-4/ ready for the leastsq call
[f,xopt, gropt] = leastsq(list("myfunc",m,tm,ym,wm),"mydfunc",x0)
```

# See Also

lsqrsolve, optim, NDcost, datafit, external, qpsolve

# Name

list2vec — Catenates list entries in a matrix.

```
[bigVector,varsizes] = list2vec(li)
```

# Parameters

li
: a list with n entries. The list entries must be 2D matrices with comptible types.

bigVector
: A column vector. Formed by the elements of the corresponding list entry.

varsizes
: An n by 3 matrix. Each row contains the dimensions of the corresponding list entry.

# Description

Catenates list entries in a column vector. The list entries are supposed to be matrices with compatible types with respect to catenation.

This function is a subsidiary for lmisolver

# Examples

```
list2vec(list(1,2,3))
list2vec(list([1 2 3],[4;5],[%s %s+1]))
```

# See Also

vec2list

# Authors

F.D. INRIA

# Name

lmisolver — linear matrix inequation solver

```
[XLISTF[,OPT]] = lmisolver(XLIST0,evalfunc [,options])
```

# Parameters

XLIST0
: a list of containing initial guess (e.g. `XLIST0=list(X1,X2,..,Xn)`)

evalfunc
: a Scilab function ("external" function with specific syntax)

The syntax the function `evalfunc` must be as follows:

`[LME,LMI,OBJ]=evalfunct(X)` where `X` is a list of matrices, `LME, LMI` are lists and `OBJ` a real scalar.

XLISTF
: a list of matrices (e.g. `XLIST0=list(X1,X2,..,Xn)`)

options
: optional parameter. If given, `options` is a real row vector with 5 components `[Mbound,abstol,nu,maxiters,reltol]`

# Description

`lmisolver` solves the following problem:

minimize `f(X1,X2,...,Xn)` a linear function of Xi's

under the linear constraints: `Gi(X1,X2,...,Xn)=0` for i=1,...,p and LMI (linear matrix inequalities) constraints:

`Hj(X1,X2,...,Xn) > 0` for j=1,...,q

The functions f, G, H are coded in the Scilab function `evalfunc` and the set of matrices Xi's in the list X (i.e. `X=list(X1,...,Xn)`).

The function `evalfun` must return in the list `LME` the matrices `G1(X),...,Gp(X)` (i.e. `LME(i)=Gi(X1,...,Xn),` i=1,...,p). `evalfun` must return in the list `LMI` the matrices `H1(X0),...,Hq(X)` (i.e. `LMI(j)=Hj(X1,...,Xn)`, j=1,...,q). `evalfun` must return in `OBJ` the value of `f(X)` (i.e. `OBJ=f(X1,...,Xn)`).

`lmisolver` returns in `XLISTF`, a list of real matrices, i. e. `XLIST=list(X1,X2,..,Xn)` where the Xi's solve the LMI problem:

Defining `Y,Z` and `cost` by:

`[Y,Z,cost]=evalfunc(XLIST)`, `Y` is a list of zero matrices, `Y=list(Y1,...,Yp)`, `Y1=0, Y2=0, ..., Yp=0`.

`Z` is a list of square symmetric matrices, `Z=list(Z1,...,Zq)` , which are semi positive definite `Z1>0, Z2>0, ..., Zq>0` (i.e. `spec(Z(j))` > 0),

`cost` is minimized.

`lmisolver` can also solve LMI problems in which the `Xi's` are not matrices but lists of matrices. More details are given in the documentation of LMITOOL.

# Examples

```
//Find diagonal matrix X (i.e. X=diag(diag(X), p=1) such that
//A1'*X+X*A1+Q1 < 0, A2'*X+X*A2+Q2 < 0 (q=2) and trace(X) is maximized
n  = 2;
A1 = rand(n,n);
A2 = rand(n,n);
Xs = diag(1:n);
Q1 = -(A1'*Xs+Xs*A1+0.1*eye());
Q2 = -(A2'*Xs+Xs*A2+0.2*eye());

deff('[LME,LMI,OBJ]=evalf(Xlist)','X   = Xlist(1); ...
                                   LME = X-diag(diag(X));...
                                   LMI = list(-(A1''*X+X*A1+Q1),-(A2''*X+X*A2+Q
                                   OBJ = -sum(diag(X))  ');

X=lmisolver(list(zeros(A1)),evalf);

X=X(1)
[Y,Z,c]=evalf(X)
```

# See Also

lmitool

# Name

lmitool — tool for solving linear matrix inequations

```
lmitool()

lmitool(filename)

txt=lmitool(probname,varlist,datalist)
```

# Parameters

filename
: a string referring to a `.sci` function

probname
: a string containing the name of the problem

varlist
: a string containing the names of the unknown matrices (separated by commas if there are more than one)

datalist
: a string containing the names of data matrices (separated by commas if there are more than one)

txt
: a string providing information on what the user should do next

# Description

`lmitool()` or `lmitool(filename)` is used to define interactively a LMI problem. In the non interactive mode, `txt=lmitool(probname,varlist,datalist)` generates a file in the current directory. The name of this file is obtained by adding `.sci` to the end of `probname`. This file is the skeleton of a solver function and the corresponding evaluation function needed by `lmisolver`.

# See Also

lmisolver

# Name

lsqrsolve — minimize the sum of the squares of nonlinear functions, levenberg-marquardt algorithm

```
[x [,v [,info]]]=lsqrsolve(x0,fct,m [,stop [,diag]])
[x [,v [,info]]]=lsqrsolve(x0,fct,m ,fjac [,stop [,diag]])
```

# Parameters

x0
: real vector of size `n` (initial estimate of the solution vector).

fct
: external (i.e function or list or string).

m
: integer, the number of functions. `m` must be greater than or equal to `n`.

fjac
: external (i.e function or list or string).

stop
: optional vector `[ftol,xtol,gtol,maxfev,epsfcn,factor]` the default value is `[1.d-8,1.d-8,1.d-5,1000,0,100]`

  ftol
  : A positive real number,termination occurs when both the actual and predicted relative reductions in the sum of squares are at most `ftol`. therefore, `ftol` measures the relative error desired in the sum of squares.

  xtol
  : A positive real number, termination occurs when the relative error between two consecutive iterates is at most `xtol`. therefore, `xtol` measures the relative error desired in the approximate solution.

  gtol
  : A nonnegative input variable. termination occurs when the cosine of the angle between `fct`(x) and any column of the jacobian is at most `gtol` in absolute value. therefore, `gtol` measures the orthogonality desired between the function vector and the columns of the jacobian.

  maxfev
  : A positive integer, termination occurs when the number of calls to `fct` is at least maxfev by the end of an iteration.

  epsfcn
  : A positive real number, used in determining a suitable step length for the forward-difference approximation. this approximation assumes that the relative errors in the functions are of the order of `epsfcn`. if `epsfcn` is less than the machine precision, it is assumed that the relative errors in the functions are of the order of the machine precision.

  factor
  : A positive real number, used in determining the initial step bound. this bound is set to the product of factor and the euclidean norm of `diag*x` if nonzero, or else to `factor` itself. in most cases `factor` should lie in the interval `(0.1,100)`. `100` is a generally recommended value.

diag
: is an array of length `n`. `diag` must contain positive entries that serve as multiplicative scale factors for the variables.

x :
real vector (final estimate of the solution vector).

v :
real vector (value of `fct(x)`).

info
termination indicator

0
improper input parameters.

1
algorithm estimates that the relative error between `x` and the solution is at most `tol`.

2
number of calls to `fcn` reached

3
`tol` is too small. No further improvement in the approximate solution `x` is possible.

4
iteration is not making good progress.

5
number of calls to `fcn` has reached or exceeded `maxfev`

6
`ftol` is too small. no further reduction in the sum of squares is possible.

7
`xtol` is too small. no further improvement in the approximate solution`x` is possible.

8
`gtol` is too small. `fvec` is orthogonal to the columns of the jacobian to machine precision.

# Description

minimize the sum of the squares of m nonlinear functions in n variables by a modification of the levenberg-marquardt algorithm. the user must provide a subroutine which calculates the functions. the jacobian is then calculated by a forward-difference approximation.

minimize `sum(fct(x,m).^2)` where `fct` is function from `R^n` to `R^m`

`fct` should be :

- a Scilab function whose calling sequence is `v=fct(x,m)` given `x` and `m`.
- a character string which refers to a C or Fortran routine which must be linked to Scilab.

  Fortran calling sequence should be `fct(m,n,x,v,iflag)` where `m`, `n`, `iflag` are integers, `x` a double precision vector of size `n` and `v` a double precision vector of size `m`.

  C calling sequence should be `fct(int *m, int *n, double x[],double v[],int *iflag)`

`fjac` is an external which returns `v=d(fct)/dx (x)`. it should be :

a Scilab function
whose calling sequence is `J=fjac(x,m)` given `x` and `m`.

a character string
: it refers to a C or Fortran routine which must be linked to Scilab.

  Fortran calling sequence should be `fjac(m,n,x,jac,iflag)` where `m`, `n`, `iflag` are integers, `x` a double precision vector of size `n` and `jac` a double precision vector of size `m*n`.

  C calling sequence should be `fjac(int *m, int *n, double x[],double v[],int *iflag)`

return -1 in iflag to stop the algoritm if the function or jacobian could not be evaluated.

# Examples

```
// A simple example with lsqrsolve
a=[1,7;
   2,8
   4 3];
b=[10;11;-1];

function y=f1(x,m)
  y=a*x+b;
endfunction

[xsol,v]=lsqrsolve([100;100],f1,3)
xsol+a\b

function y=fj1(x,m)
  y=a;
endfunction

[xsol,v]=lsqrsolve([100;100],f1,3,fj1)
xsol+a\b

// Data fitting problem
// 1 build the data
a=34;b=12;c=14;

deff('y=FF(x)','y=a*(x-b)+c*x.*x');
X=(0:.1:3)';Y=FF(X)+100*(rand()-.5);

//solve
function e=f1(abc,m)
  a=abc(1);b=abc(2),c=abc(3),
  e=Y-(a*(X-b)+c*X.*X);
endfunction

[abc,v]=lsqrsolve([10;10;10],f1,size(X,1));
abc
norm(v)
```

# See Also

external, qpsolve, optim, fsolve

# Used Functions

lmdif, lmder from minpack, Argonne National Laboratory.

# Name

numdiff — numerical gradient estimation

```
g=numdiff(fun,x [,dx])
```

# Parameters

fun
: an external, Scilab function or list. See below for calling sequence, see also external for details about external functions.

x
: vector, the argument of the function `fun`

dx
: vector, the finite difference step. Default value is `dx=sqrt(%eps)*(1+1d-3*abs(x))`

g
: vector, the estimated gradient

# Description

given a function `fun(x)` from `R^n` to `R^p` computes the matrix `g` such as

```
g(i,j) = (df_i)/(dx_j)
```

using finite difference methods.

Without parameters, the function fun calling sequence is `y=fun(x)`, and numdiff can be called as `g=numdiff(fun,x)`. Else the function fun calling sequence must be `y=fun(x,param_1,pararm_2,..,param_q)`. If parameters `param_1,param_2,..param_q` exist then `numdiff` can be called as follow `g=numdiff(list(fun,param_1,param_2,..param_q),x)`.

See the derivative with respect to numerical accuracy issues and comparison between the two algorithms.

# Examples

```
// example 1 (without parameters)
// myfun is a function from R^2 to R :   (x(1),x(2)) |--> myfun(x)
function f=myfun(x)
  f=x(1)*x(1)+x(1)*x(2)
endfunction

x=[5 8]
g=numdiff(myfun,x)

// The exact gradient (i.e derivate belong x(1) :first component and derivate b
exact=[2*x(1)+x(2)  x(1)]

//example 2 (with parameters)
// myfun is a function from R to R:  x(1) |--> myfun(x)
// myfun contains 3 parameters, a, b, c
```

```
function  f=myfun(x,a,b,c)
  f=(x+a)^c+b
endfunction

a=3; b=4; c=2;
x=1
g2=numdiff(list(myfun,a,b,c),x)

// The exact gradient, i.e derivate belong x(1), is :
exact2=c*(x+a)^(c-1)
```

# See Also

optim, derivative, external

# Name

optim — non-linear optimization routine

```
[f,xopt]=optim(costf,x0)
[f [,xopt [,gradopt [,work]]]]=optim(costf [,<contr>],x0 [,algo] [,df0 [,mem]]
```

# Parameters

costf
: external, i.e Scilab function list or string (`costf` is the cost function, that is, a Scilab script, a Fortran 77 routine or a C function.

x0
: real vector (initial value of variable to be minimized).

f
: value of optimal cost (`f=costf(xopt)`)

xopt
: best value of `x` found.

<contr>
: keyword representing the following sequence of arguments: `'b',binf,bsup` with `binf` and `bsup` are real vectors with same dimension as `x0`. `binf` and `bsup` are lower and upper bounds on `x`.

algo

- `'qn'` : quasi-Newton (this is the default solver)
- `'gc'` : conjugate gradient
- `'nd'` : non-differentiable.

  Note that the conjugate gradient solver does not accept bounds on `x`.

df0
: real scalar. Guessed decreasing of `f` at first iteration. (`df0=1` is the default value).

mem :
: integer, number of variables used to approximate the Hessian. Default value is 10. This feature is available for the Gradient-Conjugate algorithm "gc" without constraints and the non-smooth algorithm "nd" without constraints.

<stop>
: keyword representing the sequence of optional parameters controlling the convergence of the algorithm. `'ar',nap [,iter [,epsg [,epsf [,epsx]]]]`

  "ar"
  : reserved keyword for stopping rule selection defined as follows:

  nap
  : maximum number of calls to `costf` allowed (default is 100).

  iter
  : maximum number of iterations allowed (default is 100).

  epsg
  : threshold on gradient norm.

epsf
threshold controlling decreasing of `f`

epsx
threshold controlling variation of `x`. This vector (possibly matrix) of same size as `x0` can be used to scale `x`.

<params>
keyword representing the method to initialize the arguments `ti, td` passed to the objective function, provided as a C or Fortran routine. This option has no meaning when the cost function is a Scilab script. <params> can be set to only one of the following values.

- "in"

  That mode allows to allocate memory in the internal Scilab workspace so that the objective function can get arrays with the required size, but without directly allocating the memory. "in" stands for "initialization". In that mode, before the value and derivative of the objective function is to be computed, there is a dialog between the optim Scilab primitive and the objective function. In this dialog, the objective function is called two times, with particular values of the "ind" parameter. The first time, ind is set to 10 and the objective function is expected to set the nizs, nrzs and ndzs integer parameters of the "nird" common.

  ```
  common /nird/ nizs,nrzs,ndzs
  ```

  This allows Scilab to allocate memory inside its internal workspace. The second time the objective function is called, ind is set to 11 and the objective function is expected to set the ti, tr and tz arrays. After this initialization phase, each time it is called, the objective function is ensured that the ti, tr and tz arrays which are passed to it have the values that have been previously initialized.

- "ti",valti

  In this mode, valti is expected to be a Scilab vector variable containing integers. Whenever the objective function is called, the ti array it receives contains the values of the Scilab variable.

- "td", valtd

  In this mode, valtd is expected to be a Scilab vector variable containing double values. Whenever the objective function is called, the td array it receives contains the values of the Scilab variable.

- "ti",valti,"td",valtd

  This mode combines the two previous.

The `ti, td` arrays may be used so that the objective function can be computed. For example, if the objective function is a polynomial, the ti array may may be used to store the coefficients of that polynomial.

Users should choose carefully between the "in" mode and the "ti" and "td" mode, depending on the fact that the arrays are Scilab variables or not. If the data is available as Scilab variables, then the "ti", valti, "td", valtd mode should be chosen. If the data is available directly from the objective function, the "in" mode should be chosen. Notice that there is no "tr" mode, since, in Scilab, all real values are of "double" type.

If neither the "in" mode, nor the "ti", "td" mode is chosen, that is, if <params> is not present as an option of the optim primitive, the user may should not assume that the ti,tr and td arrays can be used : reading or writing the arrays may generate unpredictable results.

"imp=iflag"
: named argument used to set the trace mode. The possible values for iflag are 0,1,2 and >2. Use this option with caution : most of these reports are written on the Scilab standard output.

- iflag=0: nothing (except errors) is reported (this is the default),
- iflag=1: initial and final reports,
- iflag=2: adds a report per iteration,
- iflag>2: add reports on linear search.
- iflag<0: calls the cost function with ind=1 every -imp iterations.

gradopt
: gradient of `costf` at `xopt`

work
: working array for hot restart for quasi-Newton method. This array is automatically initialized by `optim` when `optim` is invoked. It can be used as input parameter to speed-up the calculations.

# Description

Non-linear optimization routine for programs without constraints or with bound constraints:

```
min costf(x) w.r.t x.
```

`costf` is an "external" i.e a Scilab function, a list or a string giving the name of a C or Fortran routine (see "external"). This external must return the value `f` of the cost function at the point `x` and the gradient `g` of the cost function at the point `x`.

- Scilab function case
: If `costf` is a Scilab function, the calling sequence for `costf` must be:

```
[f,g,ind]=costf(x,ind)
```

Here, `costf` is a function which returns `f`, value (real number) of cost function at `x`, and `g`, gradient vector of cost function at `x`. The variable `ind` is described below.

- List case
: If `costf` is a list, it should be of the form: `list(real_costf, arg1,...,argn)` with `real_costf` a Scilab function with calling sequence : `[f,g,ind]=costf(x,ind,arg1,... argn)`. The `x`, `f`, `g`, `ind` arguments have the same meaning that above. `argi` arguments can be used to pass function parameters.

- String case
: If `costf` is a character string, it refers to the name of a C or Fortran routine which must be linked to Scilab

* Fortran case
: The interface of the Fortran subroutine computing the objective must be :

```
subroutine costf(ind,n,x,f,g,ti,tr,td)
```

with the following declarations:

```
integer ind,n ti(*)
double precision x(n),f,g(n),td(*)
real tr(*)
```

The argument `ind` is described below.

If ind = 2, 3 or 4, the inputs of the routine are : `x, ind, n, ti, tr,td`.

If ind = 2, 3 or 4, the outputs of the routine are : `f` and `g`.

* C case
  The interface of the C function computing the objective must be :

```
void costf(int *ind, int *n, double *x, double *f, double *g, int *ti, fl
```

The argument `ind` is described below.

The inputs and outputs of the function are the same as in the fortran case.

The `ind` input argument is a message sent from the solver to the cost function so that the cost function knows what to compute.

- If `ind=2`, `costf` must provide `f`.
- If `ind=3`, `costf` must provide `g`.
- If `ind=4`, `costf` must provide `f` and `g`.
- If `ind=1`, `costf` can print messages.

When the `imp` option is set to a negative integer value, say `m` for example, the cost function is called back every `-m` iterations, with the `ind=1` parameter. This can be used to print messages, fill a log file or creating an interactive plot. See below for sample uses of this feature.

On output, `ind<0` means that `f` cannot be evaluated at `x` and `ind=0` interrupts the optimization.

# Example: Scilab function

The following is an example with a Scilab function. Notice, for simplifications reasons, the Scilab function "cost" of the following example computes the objective function f and its derivative no matter of the value of ind. This allows to keep the example simple. In practical situations though, the computation of "f" and "g" may raise performances issues so that a direct optimization may be to use the value of "ind" to compute "f" and "g" only when needed.

```
// External function written in Scilab
function [f,g,ind] = cost(x,ind)
  xref=[1;2;3];
  f=0.5*norm(x-xref)^2;
  g=x-xref;
endfunction

// Simplest call
```

```
x0=[1;-1;1];
[f,xopt]=optim(cost,x0)

// By conjugate gradient - you can use 'qn', 'gc' or 'nd'
[f,xopt,gopt]=optim(cost,x0,'gc')

//Seen as non differentiable
[f,xopt,gopt]=optim(cost,x0,'nd')

// Upper and lower bounds on x
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0)

// Upper and lower bounds on x and setting up the algorithm to 'gc'
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0,'gc')

// Bound on the number of call to the objective function
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0,'gc','ar',3)

// Set max number of call to the objective function (3)
// Set max number of iterations (100)
// Set stopping threshold on the value of f (1e-6),
// on the value of the norm of the gradient of the objective function (1e-6)
// on the improvement on the parameters x_opt (1e-6;1e-6;1e-6)
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0,'gc','ar',3,100,1e-6,1e-6,[1

// Additionnal messages are printed in the console.
[f,xopt]=optim(cost,x0,imp=3)
```

# Example: C function

The following is an example with a C function, where a C source code is written into a file, dynamically compiled and loaded into Scilab, and then used by the "optim" solver. The interface of the "rosenc" function is fixed, even if the arguments are not really used in the cost function. This is because the underlying optimization solvers must assume that the objective function has a known, constant interface. In the following example, the arrays ti and tr are not used, only the array "td" is used, as a parameter of the Rosenbrock function. Notice that the content of the arrays ti and td are the same that the content of the Scilab variable, as expected.

```
// External function written in C (C compiler required)
// write down the C code (Rosenbrock problem)
C=['#include <math.h>'
   'double sq(double x)'
   '{ return x*x;}'
   'void rosenc(int *ind, int *n, double *x, double *f, double *g, '
   '                                int *ti, float *tr, double *td)'
   '{'
   '  double p;'
   '  int i;'
   '  p=td[0];'
   '  if (*ind==2||*ind==4) {'
   '    *f=1.0;'
   '    for (i=1;i<*n;i++)'
   '      *f+=p*sq(x[i]-sq(x[i-1]))+sq(1.0-x[i]);'
   '  }'
   '  if (*ind==3||*ind==4) {'
```

```
   '    g[0]=-4.0*p*(x[1]-sq(x[0]))*x[0];'
   '    for (i=1;i<*n-1;i++)'
   '      g[i]=2.0*p*(x[i]-sq(x[i-1]))-4.0*p*(x[i+1]-sq(x[i]))*x[i]-2.0*(1.0-x[
   '    g[*n-1]=2.0*p*(x[*n-1]-sq(x[*n-2]))-2.0*(1.0-x[*n-1]);'
   '  }'
   '}'];
mputl(C,TMPDIR+'/rosenc.c')

// compile the C code
l=ilib_for_link('rosenc','rosenc.c',[],'c',TMPDIR+'/Makefile');

// incremental linking
link(l,'rosenc','c')

//solve the problem
x0=[40;10;50];
p=100;
[f,xo,go]=optim('rosenc',x0,'td',p)
```

# Example: Fortran function

The following is an example with a Fortran function.

```
// External function written in Fortran (Fortran compiler required)
// write down the Fortran  code (Rosenbrock problem)
F=[ '      subroutine rosenf(ind, n, x, f, g, ti, tr, td)'
    '      integer ind,n,ti(*)'
    '      double precision x(n),f,g(n),td(*)'
    '      real tr(*)'
    'c'
    '      double precision y,p'
    '      p=td(1)'
    '      if (ind.eq.2.or.ind.eq.4) then'
    '        f=1.0d0'
    '        do i=2,n'
    '          f=f+p*(x(i)-x(i-1)**2)**2+(1.0d0-x(i))**2'
    '        enddo'
    '      endif'
    '      if (ind.eq.3.or.ind.eq.4) then'
    '        g(1)=-4.0d0*p*(x(2)-x(1)**2)*x(1)'
    '        if(n.gt.2) then'
    '          do i=2,n-1'
    '            g(i)=2.0d0*p*(x(i)-x(i-1)**2)-4.0d0*p*(x(i+1)-x(i)**2)*x(i)'
    '     &           -2.0d0*(1.0d0-x(i))'
    '          enddo'
    '        endif'
    '        g(n)=2.0d0*p*(x(n)-x(n-1)**2)-2.0d0*(1.0d0-x(n))'
    '      endif'
    '      return'
    '      end'];

mputl(F,TMPDIR+'/rosenf.f')

// compile the Fortran code
l=ilib_for_link('rosenf','rosenf.f',[],'f',TMPDIR+'/Makefile');
```

```
// incremental linking
link(l,'rosenf','f')

//solve the problem
x0=[40;10;50];
p=100;
[f,xo,go]=optim('rosenf',x0,'td',p)
```

# Example: Fortran function with initialization

The following is an example with a Fortran function in which the "in" option is used to allocate memory inside the Scilab environment. In this mode, there is a dialog between Scilab and the objective function. The goal of this dialog is to initialize the parameters of the objective function. Each part of this dialog is based on a specific value of the "ind" parameter.

At the beginning, Scilab calls the objective function, with the ind parameter equals to 10. This tells the objective function to initialize the sizes of the arrays it needs by setting the nizs, nrzs and ndzs integer parameters of the "nird" common. Then the objective function returns. At this point, Scilab creates internal variables and allocate memory for the variable izs, rzs and dzs. Scilab calls the objective function back again, this time with ind equals to 11. This tells the objective function to initialize the arrays izs, rzs and dzs. When the objective function has done so, it returns. Then Scilab enters in the real optimization mode and calls the optimization solver the user requested. Whenever the objective function is called, the izs, rzs and dzs arrays have the values that have been previously initialized.

```
//
// Define a fortran source code and compile it (fortran compiler required)
//
fortransource=['      subroutine rosenf(ind,n,x,f,g,izs,rzs,dzs)'
               'C     -------------------------------------------'
               'c     Example of cost function given by a subroutine'
               'c     if n<=2 returns ind=0'
               'c     f.bonnans, oct 86'
               '      implicit double precision (a-h,o-z)'
               '      real rzs(1)'
               '      double precision dzs(*)'
               '      dimension x(n),g(n),izs(*)'
               '      common/nird/nizs,nrzs,ndzs'
               '      if (n.lt.3) then'
               '        ind=0'
               '        return'
               '      endif'
               '      if(ind.eq.10) then'
               '         nizs=2'
               '         nrzs=1'
               '         ndzs=2'
               '         return'
               '      endif'
               '      if(ind.eq.11) then'
               '         izs(1)=5'
               '         izs(2)=10'
               '         dzs(2)=100.0d+0'
               '         return'
               '      endif'
               '      if(ind.eq.2)go to 5'
               '      if(ind.eq.3)go to 20'
               '      if(ind.eq.4)go to 5'
```

```
                '      ind=-1'
                '      return'
                '5     f=1.0d+0'
                '      do 10 i=2,n'
                '        im1=i-1'
                '10      f=f + dzs(2)*(x(i)-x(im1)**2)**2 + (1.0d+0-x(i))**2'
                '      if(ind.eq.2)return'
                '20    g(1)=-4.0d+0*dzs(2)*(x(2)-x(1)**2)*x(1)'
                '      nm1=n-1'
                '      do 30 i=2,nm1'
                '        im1=i-1'
                '        ip1=i+1'
                '        g(i)=2.0d+0*dzs(2)*(x(i)-x(im1)**2)'
                '30      g(i)=g(i) -4.0d+0*dzs(2)*(x(ip1)-x(i)**2)*x(i) - '
                '     &        2.0d+0*(1.0d+0-x(i))'
                '      g(n)=2.0d+0*dzs(2)*(x(n)-x(nm1)**2) - 2.0d+0*(1.0d+0-x(n)
                '      return'
                '      end'];
mputl(fortransource,TMPDIR+'/rosenf.f')

// compile the C code
libpath=ilib_for_link('rosenf','rosenf.f',[],'f',TMPDIR+'/Makefile');

// incremental linking
linkid=link(libpath,'rosenf','f');

x0=1.2*ones(1,5);
//
// Solve the problem
//
[f,x,g]=optim('rosenf',x0,'in');
```

# Example: Fortran function with initialization on Windows with Intel Fortran Compiler

Under the Windows operating system with Intel Fortran Compiler, one must carefully design the fortran source code so that the dynamic link works properly. On Scilab's side, the optimization component is dynamically linked and the symbol "nird" is exported out of the optimization dll. On the cost function's side, which is also dynamically linked, the "nird" common must be imported in the cost function dll.

The following example is a re-writing of the previous example, with special attention for the Windows operating system with Intel Fortran compiler as example. In that case, we introduce additionnal compiling instructions, which allows the compiler to import the "nird" symbol.

```
fortransource=['subroutine rosenf(ind,n,x,f,g,izs,rzs,dzs)'
                'cDEC$ IF DEFINED (FORDLL)'
                'cDEC$ ATTRIBUTES DLLIMPORT:: /nird/'
                'cDEC$ ENDIF'
                'C     ------------------------------------------'
                'c     Example of cost function given by a subroutine'
                'c     if n<=2 returns ind=0'
                'c     f.bonnans, oct 86'
                '      implicit double precision (a-h,o-z)'
[etc...]
```

# Example: Logging features

The imp flag may take negative integer values, say k. In that case, the cost function is called once every -k iterations. This allows to draw the function value or write a log file.

In the following example, we solve the Rosenbrock test case. For each iteration of the algorithm, we print the value of x, f and g.

```
function [f,g,ind] = cost(x,ind)
  xref=[1;2;3];
  if ( ind == 1 | ind == 4 ) then
    f=0.5*norm(x-xref)^2;
  end
  if ( ind == 1 | ind == 4 ) then
    g=x-xref;
  end
  if ( ind == 1 ) then
    mprintf("===========\n")
    mprintf("x = %s\n", strcat(string(x)," "))
    mprintf("f = %e\n", f)
    g=x-xref;
    mprintf("g = %s\n", strcat(string(g)," "))
  end
endfunction
x0=[1;-1;1];
[f,xopt]=optim(cost,x0,imp=-1)
```

The previous script produces the following output.

```
===========
x = 1 -1 1
f = 6.500000e+000
g = 0 -3 -2
===========
x = 1 0 1.6666667
f = 2.888889e+000
g = 0 -2 -1.3333333
===========
x = 1 2 3
f = 9.860761e-031
g = 0 -4.441D-16 1.332D-15
===========
x = 1 2 3
f = 0.000000e+000
g = 0 0 0
```

In the following example, we solve the Rosenbrock test case. For each iteration of the algorithm, we plot the current value of x into a 2D graph containing the contours of Rosenbrock's function. This allows to see the progress of the algorithm while the algorithm is performing. We could as well write the value of x, f and g into a log file if needed.

```
// 1. Define rosenbrock
function [ f , g , ind ] = rosenbrock ( x , ind )
  if ((ind == 1) | (ind == 4)) then
    f = 100.0 *(x(2)-x(1)^2)^2 + (1-x(1))^2;
  end
  if ((ind == 1) | (ind == 4)) then
    g(1) = - 400. * ( x(2) - x(1)**2 ) * x(1) -2. * ( 1. - x(1) )
    g(2) = 200. * ( x(2) - x(1)**2 )
  end
  if (ind == 1) then
    plot ( x(1) , x(2) , "g." )
  end
endfunction
x0 = [-1.2 1.0];
xopt = [1.0 1.0];
// 2. Draw the contour of Rosenbrock's function
xmin = -2.0
xmax = 2.0
stepx = 0.1
ymin = -1.0
ymax = 2.0
stepy = 0.1
nx = 100
ny = 100
stepx = (xmax - xmin)/nx;
xdata = xmin:stepx:xmax;
stepy = (ymax - ymin)/ny;
ydata = ymin:stepy:ymax;
for ix = 1:length(xdata)
    for iy = 1:length(ydata)
      x = [xdata(ix) ydata(iy)];
      [ f , g , ind ] = rosenbrock ( x , 4 );
      zdata ( ix , iy ) = f;
    end
end
contour ( xdata , ydata , zdata , [1 10 100 500 1000])
plot(x0(1) , x0(2) , "b.")
plot(xopt(1) , xopt(2) , "r*")
// 3. Plot the optimization process, during optimization
[ fopt , xopt ] = optim ( rosenbrock , x0 , imp = -1)
```

# Example: Optimizing with numerical derivatives

It is possible to optimize a problem without an explicit knowledge of the derivative of the cost function. For this purpose, we can use the numdiff or derivative function to compute a numerical derivative of the cost function.

In the following example, we use the numdiff function to solve Rosenbrock's problem.

```
function f = rosenbrock ( x )
  f = 100.0 *(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
```

```
function [ f , g , ind ] = rosenbrockCost ( x , ind )
  if ((ind == 1) | (ind == 4)) then
    f = rosenbrock ( x );
  end
  if ((ind == 1) | (ind == 4)) then
    g= numdiff ( rosenbrock , x );
  end
endfunction
x0 = [-1.2 1.0];
[ fopt , xopt ] = optim ( rosenbrockCost , x0 )
```

In the following example, we use the derivative function to solve Rosenbrock's problem. Given that the step computation strategy is not the same in numdiff and derivative, this might lead to improved results.

```
function f = rosenbrock ( x )
  f = 100.0 *(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction
function [ f , g , ind ] = rosenbrockCost2 ( x , ind )
  if ((ind == 1) | (ind == 4)) then
    f = rosenbrock ( x );
  end
  if ((ind == 1) | (ind == 4)) then
    g= derivative ( rosenbrock , x.' , order = 4 );
  end
endfunction
x0 = [-1.2 1.0];
[ fopt , xopt ] = optim ( rosenbrockCost2 , x0 )
```

# Example: Counting function evaluations and number of iterations

The `imp` option can take negative values. If the `imp` is equal to `m` where `m` is a negative integer, then the cost function is evaluated every `-m` iterations, with the `ind` input argument equal to 1. The following example uses this feature to compute the number of iterations. The global variable `mydata` is used to store the number of function evaluations as well as the number of iterations.

```
xref=[1;2;3];
x0=[1;-1;1];
global _MYDATA_
_MYDATA_ = tlist ( ["T_MYDATA","niter","nfevals"])
_MYDATA_.niter = 0
_MYDATA_.nfevals = 0
function [f,g,ind] = cost(x,ind)
  global _MYDATA_
  disp(ind)
  if ( ind == 1 )
    _MYDATA_.niter = _MYDATA_.niter + 1
  end
  _MYDATA_.nfevals = _MYDATA_.nfevals + 1
```

```
  f=0.5*norm(x-xref)^2;
  g=x-xref;
endfunction
[f,xopt]=optim(cost,x0,imp=-1)
mprintf ( "Number of function evaluations:%d\n",_MYDATA_.nfevals)
mprintf ( "Number of iterations:%d\n",_MYDATA_.niter)
```

While the previous example perfectly works, there is a risk that the same variable `_MYDATA_` is used by some internal function used by `optim`. In this case, the value may be wrong. This is why a sufficiently weird variable name has been used.

# Example : Passing extra parameters

In most practical situations, the cost function depends on extra parameters which are required to evaluate the cost function. There are several methods to achieve this goal.

In the following example, the cost function uses 4 parameters `a, b, c` and `d`. We define the cost function with additionnal input arguments, which are declared after the index argument. Then we pass a list as the first input argument of the `optim` solver. The first element of the list is the cost function. The additionnal variables are directly passed to the cost function.

```
function [ f , g , ind ] = costfunction ( x , ind , a , b , c , d )
    f = a * ( x(1) - c ) ^2 + b * ( x(2) - d )^2
    g(1) = 2 * a * ( x(1) - c )
    g(2) = 2 * b * ( x(2) - d )
endfunction
x0 = [1 1];
a = 1.0;
b = 2.0;
c = 3.0;
d = 4.0;
costf = list ( costfunction , a , b , c, d );
[ fopt , xopt ] = optim ( costf , x0 , imp = 2 )
```

These coefficients are defined before the optimizer is called. There are directly used in the cost function.

```
// The example NOT to follow
function [ f , g , ind ] = costfunction ( x , ind )
    f = a * ( x(1) - c ) ^2 + b * ( x(2) - d )^2
    g(1) = 2 * a * ( x(1) - c )
    g(2) = 2 * b * ( x(2) - d )
endfunction
x0 = [1 1];
a = 1.0;
b = 2.0;
c = 3.0;
d = 4.0;
[ fopt , xopt ] = optim ( costfunction , x0 , imp = 2 )
```

While the previous example perfectly works, there is a risk that the same variables are used by some internal function used by `optim`. In this case, the value of the parameters are not what is expected and the optimization can fail or, worse, give a wrong result.

In the following example, we define the cost function with the classical header. Inside the function definition, we declare that the parameters `a, b, c` and `d` are global variables. Then we declare and set the global variables.

```
// Another example NOT to follow
function [ f , g , ind ] = costfunction ( x , ind )
    global a b c d
    f = a * ( x(1) - c ) ^2 + b * ( x(2) - d )^2
    g(1) = 2 * a * ( x(1) - c )
    g(2) = 2 * b * ( x(2) - d )
endfunction
x0 = [1 1];
global a b c d
a = 1.0;
b = 2.0;
c = 3.0;
d = 4.0;
[ fopt , xopt ] = optim ( costfunction , x0 , imp = 2 )
```

While the previous example perfectly works, there is a risk that the same variables are used by some internal function used by `optim`. In this case, the value of the parameters are not what is expected and the optimization can fail or, worse, give a wrong result.

# Example : Checking that derivatives are correct

Many optimization problem can be avoided if the derivatives are computed correctly. One common reason for failure in the step-length procedure is an error in the calculation of the cost function and its gradient. Incorrect calculation of derivatives is by far the most common user error.

In the following example, we give a false implementation of Rosenbrock's gradient. In order to check the computation of the derivatives, we use the `derivative` function. We define the `simplified` function, which delegates the computation of `f` to the rosenbrock function. The `simplified` function is passed as an input argument of the `derivative` function.

```
function [ f , g , index ] = rosenbrock ( x , index )
    f = 100.0 *(x(2)-x(1)^2)^2 + (1-x(1))^2;
    // Exact :
    g(1) = - 400. * ( x(2) - x(1)**2 ) * x(1) -2. * ( 1. - x(1) )
    // Wrong :
    g(1) = - 1200. * ( x(2) - x(1)**2 ) * x(1) -2. * ( 1. - x(1) )
    g(2) = 200. * ( x(2) - x(1)**2 )
endfunction
function f = simplified ( x )
  index = 1;
  [ f , g , index ] = rosenbrock ( x , index )
endfunction
x0 = [-1.2 1];
index = 1;
[ f , g , index ] = rosenbrock ( x0 , index );
```

```
gnd = derivative ( simplified , x0.' );
mprintf ( "Exact derivative:[%s]\n" , strcat ( string(g) , " " ));
mprintf ( "Numerical derivative:[%s]\n" , strcat ( string(gnd) , " " ));
```

The previous script produces the following output. Obviously, the difference between the two gradient is enormous, which shows that the wrong formula has been used in the gradient.

```
Exact derivative:[-638 -88]
Numerical derivative:[-215.6 -88]
```

# See Also

external, qpsolve, datafit, leastsq, numdiff, derivative, NDcost

# References

The following is a map from the various options to the underlying solvers, with some comments about the algorithm, when available.

"qn" without constraints
: n1qn1 : a quasi-Newton method with a Wolfe-type line search

"qn" with bounds constraints
: qnbd : a quasi-Newton method with projection

  RR-0242 - A variant of a projected variable metric method for bound constrained optimization problems, Bonnans Frederic, Rapport de recherche de l'INRIA - Rocquencourt, Octobre 1983

"gc" without constraints
: n1qn3 : a conjugate gradient method with BFGS.

"gc" with bounds constraints
: gcbd : a BFGS-type method with limited memory and projection

"nd" without constraints
: n1fc1 : a bundle method

"nd" with bounds constraints
: not available

# Author

The Modulopt library : J.Frederic Bonnans, Jean-Charles Gilbert, Claude Lemarechal

The interfaces to the Modulopt library : J.Frederic Bonnans

This help : Michael Baudin

# Name

qld — linear quadratic programming solver

```
[x,lagr]=qld(Q,p,C,b,ci,cs,me [,tol])
[x,lagr,info]=qld(Q,p,C,b,ci,cs,me [,tol])
```

# Parameters

Q
: real positive definite symmetric matrix (dimension `n x n`).

p
: real (column) vector (dimension `n`)

C
: real matrix (dimension `(me + md) x n`)

b
: RHS column vector (dimension `(me + md)`)

ci
: column vector of lower-bounds (dimension `n`). If there are no lower bound constraints, put `ci = []`. If some components of `x` are bounded from below, set the other (unconstrained) values of `ci` to a very large negative number (e.g. `ci(j) = -number_properties('huge')`.

cs
: column vector of upper-bounds. (Same remarks as above).

me
: number of equality constraints (i.e. `C(1:me,:)*x = b(1:me)`)

tol
: Floatting point number, required prÃ©cision.

x
: optimal solution found.

lagr
: vector of Lagrange multipliers. If lower and upper-bounds `ci,cs` are provided, `lagr` has `n + me + md` components and `lagr(1:n)` is the Lagrange vector associated with the bound constraints and `lagr (n+1 : n + me + md)` is the Lagrange vector associated with the linear constraints. (If an upper-bound (resp. lower-bound) constraint `i` is active `lagr(i)` is > 0 (resp. <0). If no bounds are provided, `lagr` has only `me + md` components.

info
: integer, return the execution status instead of sending errors.

  info==1 : Too many iterations needed

  info==2 : Accuracy insufficient to statisfy convergence criterion

  info==5 : Length of working array is too short

  info==10: The constraints are inconsistent

# Description

$$\min \frac{1}{2} \cdot x^t \cdot Q \cdot x + p^t \cdot x$$
$$\text{with } C(j,:) \cdot x = b(j), j = 1, \cdots, \text{me}$$
$$C(j,:) \cdot x \leq b(j), j = \text{me} + 1, \cdots, \text{me} + \text{md}$$
$$\text{ci} \leq x \leq \text{cs}$$

This function requires `Q` to be positive definite, if it is not the case, one may use the The contributed toolbox "**quapro**".

# Examples

```
//Find x in R^6 such that:
//C1*x = b1 (3 equality constraints i.e me=3)
C1= [1,-1,1,0,3,1;
    -1,0,-3,-4,5,6;
     2,5,3,0,1,0];
b1=[1;2;3];

//C2*x <= b2 (2 inequality constraints)
C2=[0,1,0,1,2,-1;
    -1,0,2,1,1,0];
b2=[-1;2.5];

//with  x between ci and cs:
ci=[-1000;-10000;0;-1000;-1000;-1000];cs=[10000;100;1.5;100;100;1000];

//and minimize 0.5*x'*Q*x + p'*x with
p=[1;2;3;4;5;6]; Q=eye(6,6);

//No initial point is given;
C=[C1;C2];
b=[b1;b2];
me=3;
[x,lagr]=qld(Q,p,C,b,ci,cs,me)
//Only linear constraints (1 to 4) are active (lagr(1:6)=0):
```

# See Also

qpsolve, optim

The contributed toolbox "quapro" may also be of interest, in particular for singular `Q`.

# Authors

K.Schittkowski
, University of Bayreuth, Germany

A.L. Tits and J.L. Zhou
, University of Maryland

# Used Functions

`ql0001.f` in `modules/optimization/src/fortran/ql0001.f`

# Name

qp_solve — linear quadratic programming solver builtin

```
[x [,iact [,iter [,f]]]]=qp_solve(Q,p1,C1,b,me)
```

# Parameters

Q
: real positive definite symmetric matrix (dimension `n x n`).

p
: real (column) vector (dimension `n`)

C
: real matrix (dimension `(me + md) x n`). This matrix may be dense or sparse.

b
: RHS column vector (dimension `m=(me + md)`)

me
: number of equality constraints (i.e. `x'*C(:,1:me) = b(1:me)'`)

x
: optimal solution found.

iact
: vector, indicator of active constraints. The first non zero entries give the index of the active constraints

iter
: 2x1 vector, first component gives the number of "main" iterations, the second one says how many constraints were deleted after they became active.

# Description

$$\min \frac{1}{2} \cdot x^t \cdot Q \cdot x + p^t \cdot x$$
$$\text{with} x^t \cdot C(:,j) = b(j), j = 1, \cdots, \text{me}$$
$$x^t \cdot C(:,j) \geq b(j), j = \text{me} + 1, \cdots, \text{me} + \text{md}$$

This function requires `Q` to be symmetric positive definite. If this hypothesis is not satisfied, one may use the contributed **quapro toolbox**.

# Examples

```
// Find x in R^6 such that:
// x'*C1 = b1 (3 equality constraints i.e me=3)
C1= [ 1,-1, 2;
     -1, 0, 5;
      1,-3, 3;
      0,-4, 0;
      3, 5, 1;
      1, 6, 0];
```

```
b1=[1;2;3];

// x'*C2 >= b2 (2 inequality constraints)
C2= [ 0 ,1;
     -1, 0;
      0,-2;
     -1,-1;
     -2,-1;
      1, 0];
b2=[ 1;-2.5];

// and minimize 0.5*x'*Q*x - p'*x with
p=[-1;-2;-3;-4;-5;-6]; Q=eye(6,6);

me=3;
[x,iact,iter,f]=qp_solve(Q,p,[C1 C2],[b1;b2],me)
// Only linear constraints (1 to 4) are active
```

# See Also

optim, qld, qpsolve

The contributed toolbox "quapro" may also be of interest, in particular for singular `Q`.

# Memory requirements

Let r be

```
r=min(m,n)
```

Then the memory required by qp_solve during the computations is

```
2*n+r*(r+5)/2 + 2*m +1
```

# Authors

S. Steer
INRIA (Scilab interface)

Berwin A. Turlach
School of Mathematics and Statistics (M019), The University of Western Australia, Crawley, AUSTRALIA (solver code)

# References

- Goldfarb, D. and Idnani, A. (1982). "Dual and Primal-Dual Methods for Solving Strictly Convex Quadratic Programs", in J.P. Hennart (ed.), Numerical Analysis, Proceedings, Cocoyoc, Mexico 1981, Vol. 909 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, pp. 226-239.
- Goldfarb, D. and Idnani, A. (1983). "A numerically stable dual method for solving strictly convex quadratic programs", Mathematical Programming 27: 1-33.
- QuadProg (Quadratic Programming Routines), Berwin A Turlach,http://www.maths.uwa.edu.au/~berwin/software/quadprog.html

# Used Functions

qpgen2.f and >qpgen1.f (also named QP.solve.f) developped by Berwin A. Turlach according to the Goldfarb/Idnani algorithm

# Name

qpsolve — linear quadratic programming solver

```
[x [,iact [,iter [,f]]]]=qpsolve(Q,p,C,b,ci,cs,me)
```

# Parameters

Q
: real positive definite symmetric matrix (dimension `n x n`).

p
: real (column) vector (dimension `n`)

C
: real matrix (dimension `(me + md) x n`). This matrix may be dense or sparse.

b
: RHS column vector (dimension `m=(me + md)`)

ci
: column vector of lower-bounds (dimension `n`). If there are no lower bound constraints, put `ci = []`. If some components of `x` are bounded from below, set the other (unconstrained) values of `ci` to a very large negative number (e.g. `ci(j) = -number_properties('huge')`.

cs
: column vector of upper-bounds. (Same remarks as above).

me
: number of equality constraints (i.e. `C(1:me,:)*x = b(1:me)`)

x
: optimal solution found.

iact
: vector, indicator of active constraints. The first non zero entries give the index of the active constraints

iter
: . 2x1 vector, first component gives the number of "main" iterations, the second one says how many constraints were deleted after they became active.

# Description

$$\min \frac{1}{2} \cdot x^t \cdot Q \cdot x + p^t \cdot x$$
$$\text{with } C(j,:) \cdot x = b(j), j = 1, \cdots, \text{me}$$
$$C(j,:) \cdot x \leq b(j), j = \text{me} + 1, \cdots, \text{me} + \text{md}$$
$$\text{ci} \leq x \leq \text{cs}$$

This function requires `Q` to be symmetric positive definite. If that hypothesis is not satisfied, one may use the quapro function, which is provided in the Scilab quapro toolbox.

The qpsolve solver is implemented as a Scilab script, which calls the compiled qp_solve primitive. It is provided as a facility, in order to be a direct replacement for the former quapro solver : indeed, the

qpsolve solver has been designed so that it provides the same interface, that is, the same input/output arguments. But the x0 and imp input arguments are available in quapro, but not in qpsolve.

# Examples

```
//Find x in R^6 such that:
//C1*x = b1 (3 equality constraints i.e me=3)
C1= [1,-1,1,0,3,1;
    -1,0,-3,-4,5,6;
     2,5,3,0,1,0];
b1=[1;2;3];

//C2*x <= b2 (2 inequality constraints)
C2=[0,1,0,1,2,-1;
    -1,0,2,1,1,0];
b2=[-1;2.5];

//with  x between ci and cs:
ci=[-1000;-10000;0;-1000;-1000;-1000];
cs=[10000;100;1.5;100;100;1000];

//and minimize 0.5*x'*Q*x + p'*x with
p=[1;2;3;4;5;6]; Q=eye(6,6);

//No initial point is given;
C=[C1;C2];
b=[b1;b2];
me=3;
[x,iact,iter,f]=qpsolve(Q,p,C,b,ci,cs,me)
//Only linear constraints (1 to 4) are active
```

# See Also

optim, qp_solve, qld

The contributed toolbox "quapro" may also be of interest, in particular for singular `Q`.

# Memory requirements

Let r be

```
r=min(m,n)
```

Then the memory required by qpsolve during the computations is

```
2*n+r*(r+5)/2 + 2*m +1
```

# Authors

S. Steer
INRIA (Scilab interface)

Berwin A. Turlach
School of Mathematics and Statistics (M019), The University of Western Australia, Crawley, AUSTRALIA (solver code)

# References


- Goldfarb, D. and Idnani, A. (1982). "Dual and Primal-Dual Methods for Solving Strictly Convex Quadratic Programs", in J.P. Hennart (ed.), Numerical Analysis, Proceedings, Cocoyoc, Mexico 1981, Vol. 909 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, pp. 226-239.
- Goldfarb, D. and Idnani, A. (1983). "A numerically stable dual method for solving strictly convex quadratic programs", Mathematical Programming 27: 1-33.
- QuadProg (Quadratic Programming Routines), Berwin A Turlach,http://www.maths.uwa.edu.au/~berwin/software/quadprog.html


# Used Functions

qpgen1.f (also named QP.solve.f) developped by Berwin A. Turlach according to the Goldfarb/Idnani algorithm

# Name

quapro — linear quadratic programming solver (obsolete)

# Description

This function is superseded by qpsolve.

Users who are still interested by quapro may consider the Scilab quapro toolbox which provide the same features as in older Scilab releases.

See at http://www.scilab.org/contrib/index_contrib.php?page=download.php.

# See Also

qpsolve

# Name

readmps — reads a file in MPS format

```
mps= readmps (file-name,bounds [,maxsizes]);
```

# Parameters

file-name
: a string, the name of the mps file

bounds
: 2-vector `[lowbound,upbound]` , default lower and upper bounds

maxsizes
: 3-vector `[maxm,maxn,maxnza]` Maximum number of contraints and variables, maximum number of nonzeros entries in the LP constraint matrix. If omitted readmps reads the file once just to compute these numbers.

mps
: tlist with following fields

  irobj
  : integer (index of the objective row).

  namec
  : character string (Name of the objective).

  nameb
  : character string (Name of the right hand side).

  namran
  : character string (Name of the ranges section).

  nambnd
  : character string (Name of the bounds section).

  name
  : character string (Name of the LP problem).

  rownames
  : character string column vector (Name of the rows).

  colnames
  : character string row vector (Name of the columns).

  rowstat
  : integer vector, row types:

    1
    : row type is "="

    2
    : row type is ">="

    3
    : row type is "<="

    4
    : objective row

5
: other free row

rowcode
: real matrix `[hdrowcd,lnkrow]` with

hdrowcd
: real vector (Header to the linked list of rows with the same codes).

lnkrow
: integer vector (Linked list of rows with the same codes).

colcode
: real matrix `[hdcolcd,lnkcol]` with

hdcolcd
: integer vector (Header to the linked list of columns with the same codes).

lnkcol
: integer vector (Linked list of columns with the same codes).

rownmbs
: integer vector (Row numbers of nonzeros in columns of matrix A.)

colpnts
: integer vector (Pointers to the beginning of columns of matrix A).

acoeff
: real vector (Array of nonzero elements for each column).

rhs
: real vector ( Right hand side of the linear program).

ranges
: real vector of constraint ranges.

bounds
: real matrix `[lbounds,ubounds]` with

ubounds
: full column vector of upper bounds

lbounds
: full column vector of lower bounds

stavar
: full column vector of variable status

0
: standard (non negative) variable

1
: upper bounded variable

2
: lower bounded variable

3
: lower and upper bounded variable

4
: minus infinity type variable i.e -inf<x<=u

5
: plus infinity type variable i.e l<=x< inf

6
: fixed type variable i.e l=x=u

-k
: free variable

# Description

Reads a file containing description of an LP problem given in MPS format and returns a `tlist` which describes the problem. It is an interface with the program `rdmps1.f` of hopdm (J. Gondzio). For a description of the variables, see the file rdmps1.f.

MPS format is a standard ASCII medium for LP codes. MPS format is described in more detail in Murtagh's book:

Murtagh B. (1981). Advanced Linear Programming, McGrew-Hill, New York, 1981.

# Examples

```
//Let the LP problem:
//objective:
//   min     XONE + 4 YTWO + 9 ZTHREE
//constraints:
//  LIM1:    XONE +   YTWO            < = 5
//  LIM2:    XONE +            ZTHREE > = 10
// MYEQN:         -   YTWO  +  ZTHREE   = 7
//Bounds
//  0 < = XONE < = 4
// -1 < = YTWO < = 1

//Generate MPS file
txt=['NAME          TESTPROB'
     'ROWS'
     ' N  COST'
     ' L  LIM1'
     ' G  LIM2'
     ' E  MYEQN'
     'COLUMNS'
     '    XONE      COST                 1   LIM1                 1'
     '    XONE      LIM2                 1'
     '    YTWO      COST                 4   LIM1                 1'
     '    YTWO      MYEQN               -1'
     '    ZTHREE    COST                 9   LIM2                 1'
     '    ZTHREE    MYEQN                1'
     'RHS'
     '    RHS1      LIM1                 5   LIM2                10'
     '    RHS1      MYEQN                7'
     'BOUNDS'
     ' UP BND1      XONE                 4'
     ' LO BND1      YTWO                -1'
     ' UP BND1      YTWO                 1'
     'ENDATA'];
mputl(txt,TMPDIR+'/test.mps')
```

```
//Read the MPS file
P=readmps(TMPDIR+'/test.mps',[0 10^30])
disp(P)
```

# Nom

recons — reciprocal function for aplat.

```
r = recons(fl,ind)
```

# Parameters

fl
: a "flat" list.

ind
: a list of paths

r
: a hierachical list build with the leaves of fl

# Description

Reciprocal function for aplat. Creates a hierachical list given a flat one and a list of paths.

utility function for vec2list and lmisolver.

# Examples

```
[lf,ind]=aplat(list(1,2,list([3,1],'xxx',list([3,2,1]))));
recons(lf,ind)
```

# See Also

aplat

# Authors

F.D and S.S. INRIA

# Name

semidef — semidefinite programming

```
[x,Z,ul,info]=semidef(x0,Z0,F,blck_szs,c,options)
```

# Parameters

x0
: m x 1 real column vector (must be strictly primal feasible, see below)

Z0
: L x 1 real vector (compressed form of a strictly feasible dual matrix, see below)

F
: L x (m+1) real matrix

blck_szs
: p x 2 integer matrix (sizes of the blocks) defining the dimensions of the (square) diagonal blocks `size(Fi(j)=blck_szs(j) j=1,...,m+1`.

c
: m x 1 real vector

options
: row vector with five entries `[nu,abstol,reltol,0,maxiters]`

ul
: row vector with two entries

# Description

`[x,Z,ul,info]=semidef(x0,Z0,F,blck_szs,c,options)` solves semidefinite program:

$$\min c^t \cdot x$$
$$\text{with} F_0 + x_1 \cdot F_1 + \cdots + x_m \cdot F_m \geq 0$$

and its dual:

$$\max - \text{trace}(F_0 \cdot Z)$$
$$\text{with} \text{trace}(F_i \cdot Z) = c_i, i = 1, \cdots, m$$
$$Z \geq 0$$

exploiting block structure in the matrices `F_i`.

It interfaces L. Vandenberghe and S. Boyd sp.c program.

The `Fi's` matrices are stored columnwise in `F` in compressed format: if `F_i^j`, i=0,..,m, j=1,...,L denote the jth (symmetric) diagonal block of `F_i`, then

$$F = \begin{pmatrix} \mathrm{pack}(F_0^1) & \mathrm{pack}(F_1^1) & \cdots & \mathrm{pack}(F_m^1) \\ \mathrm{pack}(F_0^2) & \mathrm{pack}(F_1^2) & \cdots & \mathrm{pack}(F_m^2) \\ \cdots & \cdots & \cdots & \cdots \\ \mathrm{pack}(F_0^L) & \mathrm{pack}(F_1^L) & \cdots & \mathrm{pack}(F_m^L) \end{pmatrix}$$

where `pack(M)`, for symmetric `M`, is the vector `[M(1,1);M(1,2);...;M(1,n);M(2,2);M(2,3);...;M(2,n);...;M(n,n)]` (obtained by scanning columnwise the lower triangular part of `M`).

`blck_szs` gives the size of block `j`, ie, `size(F_i^j)=blck_szs(j)`.

Z is a block diagonal matrix with L blocks `Z^0, ..., Z^{L-1}`. `Z^j` has size `blck_szs[j] times blck_szs[j]`. Every block is stored using packed storage of the lower triangular part.

The 2 vector `ul` contains the primal objective value `c'*x` and the dual objective value `-trace(F_0*Z)`.

The entries of `options` are respectively: `nu` = a real parameter which ntrols the rate of convergence. `abstol` = absolute tolerance. `reltol` = relative tolerance (has a special meaning when negative). `tv` target value, only referenced if `reltol < 0`. `iters` = on entry: maximum number of iterations >= 0, on exit: the number of iterations taken. Notice that the absolute tolerance cannot be lower than 1.0e-8, that is, the absolute tolerance used in the algorithm is the maximum of the user-defined tolerance and the constant tolerance 1.0e-8.

`info` returns 1 if maxiters exceeded, 2 if absolute accuracy is reached, 3 if relative accuracy is reached, 4 if target value is reached, 5 if target value is not achievable; negative values indicate errors.

Convergence criterion:

```
(1) maxiters is exceeded
(2) duality gap is less than abstol
(3) primal and dual objective are both positive and
    duality gap is less than (reltol * dual objective)
    or primal and dual objective are both negative and
    duality gap is less than (reltol * minus the primal objective)
(4) reltol is negative and
    primal objective is less than tv or dual objective is greater
    than tv
```

# Examples

```
F0=[2,1,0,0;
    1,2,0,0;
    0,0,3,1
    0,0,1,3];

F1=[1,2,0,0;
    2,1,0,0;
    0,0,1,3;
    0,0,3,1]
```

```
F2=[2,2,0,0;
    2,2,0,0;
    0,0,3,4;
    0,0,4,4];

blck_szs=[2,2];

F01=F0(1:2,1:2);F02=F0(3:4,3:4);
F11=F1(1:2,1:2);F12=F1(3:4,3:4);
F21=F2(1:2,1:2);F22=F2(3:4,3:4);

x0=[0;0]
Z0=2*F0;
Z01=Z0(1:2,1:2);Z02=Z0(3:4,3:4);
FF=[[F01(:);F02(:)],[F11(:);F12(:)],[F21(:);F22(:)]]
ZZ0=[[Z01(:);Z02(:)]];

c=[trace(F1*Z0);trace(F2*Z0)];
options=[10,1.d-10,1.d-10,0,50];

[x,Z,ul,info]=semidef(x0,pack(ZZ0),pack(FF),blck_szs,c,options)

w=vec2list(unpack(Z,blck_szs),[blck_szs;blck_szs]);Z=sysdiag(w(1),w(2))

c'*x+trace(F0*Z)
spec(F0+F1*x(1)+F2*x(2))
trace(F1*Z)-c(1)
trace(F2*Z)-c(2)
```

# References

L. Vandenberghe and S. Boyd, " Semidefinite Programming," Informations Systems Laboratory, Stanford University, 1994.

Ju. E. Nesterov and M. J. Todd, "Self-Scaled Cones and Interior-Point Methods in Nonlinear Programming," Working Paper, CORE, Catholic University of Louvain, Louvain-la-Neuve, Belgium, April 1994.

SP: Software for Semidefinite Programming, http://www.ee.ucla.edu/~vandenbe/sp.html

# Name

vec2list — list2vec reciprocal function

```
li=vec2list(bigVector,varsizes,ind)
```

# Parameters

bigVector
: An m by n matrix. Each column is used to generate the corresponding list entry.

varsizes
: An n by 2 matrix. Each row give the dimensions of the matrix to be built with the correponding column of bigVector.

ind
: a list with n entries. Each entry is a vector of positive integers which gives the hierchical path where the corresponding matrix has to be put.

li
: a list or a recursive list with n leaves. The list entries (or leaves) are 2D matrices built with the corresponding column of bigVector and size given by the corresponding row of varsizes.

# Description

If the ind argument is not given, this function creates a list. The list entries (or leaves) are 2D matrices built with the corresponding column of bigVector and size given by the corresponding row of varsizes.

If the ind argument is given, this function creates a hierachical list with n leaves. The leaves are 2D matrices built with the corresponding column of bigVector and size given by the corresponding row of varsizes. The hierachical path for each leaf if given by the corresponding entry of ind.

This function is a subsidiary for lmisolver

# Examples

```
vec2list(1:4,ones(4,2))
vec2list(1:4,[2 1;1 2])
vec2list(1:4,ones(4,2),list(1,2,[3,1],[3,2,1]))
```

# See Also

list2vec

# Authors

F.D., INRIA
S.S., INRIA

# Parte XXVI. Sobrecarga

# Name

overloading — capacidades de overloading ("sobrecarga") de exibições, funções e operadores

# Descrição

No Scilab, exibições, funções e operadores de variáveis podem ser definidos para novos objetos utilizando funções (codificadas no Scilab ou primitivas).

Exibição (Display)

a exibição de objetos definidos por uma estrutura `tlist` pode ser sobrecarregada (a exibição padrão é semelhante a de `list`'s). A função de sobrecarga não deve ter argumentos de saída e deve ter um único argumento de entrada. Seu nome é formado como segue: `%<tlist_type>_p` onde `%<tlist_type>` significa a primeira entrada do componente do tipo `tlist` truncado aos 9 primeiros caracteres.

Operadores (Operators)

cada operador que não está definido para dados tipos de operandos pode ser definido. A função de sobrecarga deve ter um único argumento de saída e um ou dois de entrada de acordo com o número de operandos. O nome da função é definido como segue:

para operadores binários: `%<tipo_do_primeiro_operando>_<código_do_operador>_<tipo_do_segundo_operando>`

para operadores unários: `%<tipo_do_operando>_<código_do_operador>`

Operadores de extração e inserção que são n-nários são descritos abaixo.

`<tipo_do_operando>`, `<tipo_do_primeiro_operando>`, `<tipo_do_segundo_operando>` são seqüências de caracteres associadas a cada tipo de dado como descrito na tabela seguinte:

| tipo de dado | código "char" | tipo de dado | código "char" |
|---|---|---|---|
| constante | `s` | booleano | `b` |
| string | `c` | bibilioteca | `f` |
| ponteiro de função | `fptr` | manipulador | `h` |
| inteiro | `i` | lista | `l` |
| função | `m` | função compilada | `mc` |
| polinômio | `p` | esparso | `sp` |
| esparso booleano | `spb` | tlist | `tlist_type` |
| polinômio de tamanho implícito | `ip` | matriz esparsa do Matlab | `msp` |
| mlist | `mlist_type` | ponteiro | `ptr` |

`<código_do_operador>` é um único caractere associado a cada operador como descrito na tabela seguinte:

| operador | código "char" | operador | código "char" |
|---|---|---|---|
| `'` | t | + | a |
| - | s | * | m |
| `/` | r | \ | l |
| ^ | p | .* | x |
| ./ | d | .\ | q |

| | | | |
|---|---|---|---|
| .*. | k | ./. | y |
| .\. | z | : | b |
| *. | u | /. | v |
| \. | w | [a,b] | c |
| [a;b] | f | () extraction | e |
| () insertion | i | == | o |
| <> | n | \| | g |
| & | h | .^ | j |
| ~ | 5 | .' | 0 |
| < | 1 | > | 2 |
| <= | 3 | >= | 4 |
| | | iext | 6 |

A função de sobrecarga para sintaxe de extração `b=a(i1,...,in)` tem a seguinte seqüência de chamamento: `b=%<tipo_de_a>_e_(i1,...,in,a)`

e a sintaxe`[x1,..,xm]=a(i1,...,in)` tem a seguinte seqüência de chamamento: `[x1,..,xm]=%<tipo_de_a>_e_(i1,...,in,a)`

A função de sobrecarga associada à sintaxe de inserção `a(i1,...,in)=b` tem a segunite seqüência de chamamento: `a=%<tipo_de_b>_i_<tipo_de_a>(i1,...,in,b,a)`.

O código "char" `6` pode ser usado para alguns algoritmos de inserção complexos como `x.b(2)=33` onde o campo `b` não está definido na estrutura `x`. A inserção é automaticamente decomposta em `temp=x.b; temp(2)=33; x.b=temp`. O código "char" `6` é usado para o primeiro passo desse algoritmo. A função de sobrecarga de `6` é muito semelhante à função de `e`'s.

Funções (Functions)

algumas funções primitivas básicas também podem ser sobrecarregadas para novos tipo de dados. Quando tal função não é definida para tipos de dados particulares, a função `%<tipo_de_um_argumento>_<nome_da_função>` é chamada. O usuário pode adicionar a esta função chamada a definição associada aos tipos de dados de entrada.

# Exemplos

```
//EXIBIÇÃO
deff('[]=%tab_p(l)','disp([['' '';l(3)] [l(2);string(l(4))]])')
tlist('tab',['a','b'],['x';'y'],rand(2,2))

//OPERADOR
deff('x=%c_a_s(a,b)','x=a+string(b)')
's'+1

//FUNÇÃO
deff('x=%c_sin(a)','x=''sin(''+a+'')''')
sin('2*x')
```

# Ver Também

tlist, disp, symbols

# Parte XXVII. Polinômios

# Name

bezout — equação de Bezout para polinômios ou inteiros

```
[thegcd,U]=bezout(p1,p2)
```

# Parâmetros

p1, p2
: dois polinômios reais ou dois escalares inteiros (tipo igual a 8)

# Descrição

`[thegcd,U]=bezout(p1,p2)` computa o MDC `thegcd` de `p1` e `p2` e também uma matriz (2x2) unimodular `U` tal quet:

`[p1,p2]*U = [thegcd,0]`

O MMC de `p1` e `p2` é dado por:

`p1*U(1,2)` (or `-p2*U(2,2)`)

# Exemplos

```
// caso polinomial
x=poly(0,'x');
p1=(x+1)*(x-3)^5;p2=(x-2)*(x-3)^3;
[thegcd,U]=bezout(p1,p2)
det(U)
clean([p1,p2]*U)
thelcm=p1*U(1,2)
lcm([p1,p2])

// caso inteiro
i1=int32(2*3^5); i2=int32(2^3*3^2);
[thegcd,U]=bezout(i1,i2)
V=int32([2^2*3^5, 2^3*3^2,2^2*3^4*5]);
[thegcd,U]=gcd(V)
V*U
lcm(V)
```

# Ver Também

poly, roots, simp, clean, lcm

# Autor

S. Steer INRIA

# Name

clean — limpa matrizes (arredonda para zero entradas pequenas)

```
B=clean(A [,epsa [,epsr]])
```

# Parâmetros

A
: uma matriz numérica (de escalares, de polinômios, esparsa...)

epsa,epsr
: números reais (valores padrões respectivos: 1.d-10 e 1.d-10)

# Descrição

Esta função elimina (i.e. ajusta para zero) todos os coeficientes com valores absolutos< `epsa` e valores relativos< `epsr` (relativo siginifica "em relação à norma-1 de coeficientes") em um polinômio (possivelmente uma matriz de polinômios ou matriz de razões de polinômios).

Os valores padrões são `epsa=1.d-10` e `epsr=1.d-10;`

Para uma matriz de constantes `clean(A,epsa)` ajusta para zero todas as entradas menores que `epsa`.

# Exemplos

```
x=poly(0,'x');
w=[x,1,2+x;3+x,2-x,x^2;1,2,3+x]/3;
w*inv(w)
clean(w*inv(w))
```

# Name

cmndred — forma de denominador comum

```
[n,d]=cmndred(num,den)
```

# Parâmetros

num, den
: duas matrizes de polinômios de dimensões iguais

# Descrição

`[n,d]=cmndred(num,den)` computa uma matriz de polinômios `n` e um polinômio denominador comum `d` tais que:

`n/d=num./den`

A matriz de razões de polinômios definida por `num./den` é `n/d`

# Ver Também

simp, clean

# Name

coeff — coeficientes de matrizes de polinômios

```
[C]=coeff(Mp [,v])
```

# Parâmetros

Mp
: matriz de polinômios

v
: vetor (linha ou coluna) de inteiros dos graus selecionados

C
: matriz grande dos coeficientes

# Descrição

`C=coeff(Mp)` retorna em uma matriz grande `C` os coeficientes da matriz de polinômios `Mp` . `C` é particionada como `C=[C0,C1,...,Ck]` onde os `Ci` estão dispostos em ordem crescente, `k = maxi(degree(Mp))`

`C=coeff(Mp,v)` retorna a matriz de coeficientes com graus em `v` . (`v` é um vetor linha ou coluna).

# Ver Também

poly, degree, inv_coeff

# Autor

S. Steer INRIA

# Name

coffg — matriz inversa de polinômios

```
[Ns,d]=coffg(Fs)
```

# Parâmetros

Fs
: matriz quadrada de polinômios

# Descrição

`coffg` computa `Fs^-1` onde `Fs` é uma matriz de polinômios, pelo método dos cofatores.

`Fs` inverse = `Ns/d`

`d` = denominador comum; `Ns` = numerador (uma matriz de polinômios)

(Para matrizes grandes, seja paciente... os resultados geralmente são confiáveis)

# Exemplos

```
s=poly(0,'s')
a=[ s, s^2+1; s  s^2-1];
[a1,d]=coffg(a);
(a1/d)-inv(a)
```

# Ver Também

determ, detr, invr, penlaur, glever

# Autor

F. D.; ;

# Name

colcompr — compressão de colunas de matrizes de polinômios

```
[Y,rk,ac]=colcompr(A);
```

# Parâmetros

A
: matriz de polinômios

Y
: matriz de polinômios quadrada (base unimodular direita)

rk
: posto normal de `A`

Ac
: `Ac=A*Y`, matriz de polinômios

# Descrição

Compressão de colunas da matriz de polinômios `A` (compressão para a esquerda).

# Exemplos

```
s=poly(0,'s');
p=[s;s*(s+1)^2;2*s^2+s^3];
[Y,rk,ac]=colcompr(p*p');
p*p'*Y
```

# Ver Também

rowcompr

# Name

degree — grau da matriz de polinômios

```
[D]=degree(M)
```

# Parâmetros

M
: matriz de polinômios

D
: matriz de inteiros

# Descrição

Retorna a matriz de graus mais elevados de `M`.

# Ver Também

poly, coeff, clean

# Name

denom — denominador

```
den=denom(r)
```

# Parâmetros

r
: matriz de polinômios, de razões de polinômios ou de constantes.

den
: matriz de polinômios

# Descrição

`den=denom(r)` retorna o denominador de uma matriz de razões de polinômios.

Desde que razões de polinômios são representadas internamente como `r=list(['r','num','den','dt'],num,den,[])`, `denom(r)` é o mesmo que `r(3)`, `r('den')` ou `r.den`

# Ver Também

numer

# Name

derivat — derivada de matriz de razões de polinômios

```
pd=derivat(p)
```

# Parâmetros

p
: matriz de polinômios ou de razões de polinômios

# Descrição

Computa a derivada da matriz de polinômios ou de funções racionais em relação à variável livre.

# Exemplos

```
s=poly(0,'s');
derivat(1/s)  // -1/s^2;
```

# Name

determ — determinante de matrizes de polinômios

```
res=determ(W [,k])
```

# Parâmetros

W
: matriz quadrada de polinômios reais

k
: inteiro (limite superior para o grau do determinante de W)

# Descrição

Retorna o determinante de uma matriz de polinômios reais (computação feita por Transformada Rápida de Fourier (TRF) se W possuir tamanho maior do que 2*2).

`res=determ(W [,k])` `k` é um inteiro maior do que o grau verdadeiro do determinante de `W`.

O valor padrão de `k` é a menor potência de 2 que é maior que `n*maxi(degree(W))`.

Método (apenas se o tamanho de W for maior que 2*2) : avalia o determinante de W para as freqüências de Fourier e aplica a TRF inversa para os coeficientes do determinante.

# Exemplos

```
s=poly(0,'s');
w=s*rand(10,10);
determ(w)
det(coeff(w,1))*s^10
```

# Ver Também

det, detr, coffg

# Autor

F.D.

# Name

detr — determinante polinomial

```
d=detr(h)
```

# Parâmetros

h
: matriz quadrada de polinômios ou de razões de polinômios

# Descrição

`d=detr(h)` retorna o determinante `d` da matriz de funções polinomiais ou racionais `h`. É baseado no algoritmo de Leverrier.

# Ver Também

det, determ

# Name

diophant — equação (de Bezout) diofantina

```
[x,err]=diophant(p1p2,b)
```

# Parâmetros

p1p2
: vetor de polinômios `p1p2 = [p1 p2]`

b
: polinômio

x
: vetor de polinômios [x1;x2]

# Descrição

`diophant` resolve a equação de Bezout:

`p1*x1+p2*x2=b` com `p1p2` um vetor de polinômios. Se a equação não for solúvel

de outro modo `err=0`

# Exemplos

```
s=poly(0,'s');p1=(s+3)^2;p2=(1+s);
x1=s;x2=(2+s);
[x,err]=diophant([p1,p2],p1*x1+p2*x2);
p1*x1+p2*x2-p1*x(1)-p2*x(2)
```

# Name

factors — fatoração numérica real

```
[lnum,g]=factors(pol [,'flag'])
[lnum,lden,g]=factors(rat [,'flag'])
rat=factors(rat,'flag')
```

# Parâmetros

pol
: polinômio real

rat
: razão de polinômios reais (`rat=pol1/pol2`)

lnum
: lista de polinômios (de graus 1 ou 2)

lden
: lista de polinômios (de graus 1 ou 2)

g
: número real

flag
: string `'c'` ou `'d'`

# Descrição

Retorna os fatores do polinômio `pol` na lista `lnum` e o "ganho" g.

Tem-se pol= g vezes o produto das entradas da lista `lnum` (se `flag` não for fornecido). Se `flag='c'` for dado, então tem-se `|pol(i omega)|` = `|g*prod(lnum_j(i omega)|`. Se `flag='d'` for dado, então tem-se `|pol(exp(i omega))|` = `|g*prod(lnum_i(exp(i omega))|`. Se o argumento de `factors` for uma razão `rat=pol1/pol2`, os fatores do numerador `pol1` e do denominador `pol2` são retornados nas listas `lnum` e `lden` respectivamente.

O "ganho" é retornado como `g`, i.e. tem-se: rat= g vezes (produto das entradas de lnum) / (produto das entradas de lden).

Se `flag` for `'c'` (respectivamente, `'d'`), as raízes `pol` serão refletidas em relação em relação ao eixo imaginário (respectivamente, ao círculo unitário), i.e. os fatores em `lnum` são polinômios estáveis.

A mesma coisa se `factors` for invocado com argumentos racionais: as entradas em `lnum` e `lden` são polinômios estáveis se `flag` for dado. `R2=factors(R1,'c')` ou `R2=factors(R1,'d')` com `R1` uma função racional ou lista `syslin` SISO, então a saída `R2` é uma transferência com com numerador e denominador estáveis e com a mesma magnitude de `R1` ao longo do eixo imaginário (`'c'`) ou do círculo unitário (`'d'`).

# Exemplos

```
n=poly([0.2,2,5],'z');
d=poly([0.1,0.3,7],'z');
R=syslin('d',n,d);
R1=factors(R,'d')
roots(R1('num'))
```

```
roots(R1('den'))
w=exp(2*%i*%pi*[0:0.1:1]);
norm(abs(horner(R1,w))-abs(horner(R,w)))
```

## Ver Também

simp

# Name

gcd — cálculo de máximo divisor comum

```
[pgcd,U]=gcd(p)
```

# Parâmetros

p
: vetor linha de polinômios `p=[p1,..,pn]` ou vetor linha de inteiros (tipo igual a 8)

# Descrição

Computa o MDC dos componentes de `p` e uma matriz unimodular (inversa de polinômios) `U`, com grau mínimo tais que

```
p*U=[0 ... 0 pgcd]
```

# Exemplos

```
//caso polinomial
s=poly(0,'s');
p=[s,s*(s+1)^2,2*s^2+s^3];
[pgcd,u]=gcd(p);
p*u

//caso inteiro
V=int32([2^2*3^5, 2^3*3^2,2^2*3^4*5]);
[thegcd,U]=gcd(V)
V*U
```

# Ver Também

bezout, lcm, hermit

# Name

hermit — forma hermitiana

```
[Ar,U]=hermit(A)
```

# Parâmetros

A
: matriz de polinômios

Ar
: matriz de polinômios triangular

U
: matriz de polinômios unimodular

# Descrição

Forma hermitiana: `U` é uma matriz unimodular tal que `A*U` está na forma triangular de Hermite:

A variável de saída é `Ar=A*U`.

Aviso: versão experimental

# Exemplos

```
s=poly(0,'s');
p=[s, s*(s+1)^2, 2*s^2+s^3];
[Ar,U]=hermit(p'*p);
clean(p'*p*U), det(U)
```

# Ver Também

hrmt, htrianr

# Name

horner — avaliação polinomial/racional

```
horner(P,x)
```

# Parâmetros

P
: matriz de polinômios ou de razões de polinômios

x
: número real, polinômio ou razão de polinômios

# Descrição

Avalia a matriz de polinômios ou de razões de polinômios `P = P(s)` quando a variável `s` do polinômio é substituida por `x`:

`horner(P,x)=P(x)`

Exemplo (transformação bilinear): admita que `P = P(s)` uma matriz de razões de polinômios, então a matriz de razões de polinômios `P((1+s)/(1-s))` é obtida por `horner(P,(1+s)/(1-s))`.

Para avaliar uma matriz de razões de polinômios em dadas freqüências use preferivelmente a primitiva `freq`.

# Exemplos

```
//avaliação de um polinômio para um vetor de números
P=poly(1:3,'x')
horner(P,[1 2 5])
horner(P,[1 2 5]+%i)

//avaliação de uma razão
s=poly(0,'s');M=[s,1/s];
horner(M,1)
horner(M,%i)
horner(M,1/s)

//avaliação de um polinômio para uma matriz de números
X= [1 2;3 4]
p=poly(1:3,'x','c')
m=horner(p, X)
1*X.^0+2*X.^1+3*X.^2
```

# Ver Também

freq, repfreq, evstr

# Name

hrmt — máximo divisor comum de polinômios

```
[pg,U]=hrmt(v)
```

# Parâmetros

v
: vetor linha de polinômios, i.e. matriz de polinômios `1xk`

pg
: polinômio

U
: matriz de polinômios unimodular

# Descrição

`[pg,U]=hrmt(v)` retorna uma matriz unimodular `U` e `pg` = MDC do vetor linha de polinômios `v` tais que `v*U = [pg,0]`.

# Exemplos

```
x=poly(0,'x');
v=[x*(x+1),x^2*(x+1),(x-2)*(x+1),(3*x^2+2)*(x+1)];
[pg,U]=hrmt(v);U=clean(U)
det(U)
```

# Ver Também

gcd, htrianr

# Autor

S. Steer INRIA

# Name

htrianr — triangularização de matrizes de polinômios

```
[Ar,U,rk]=htrianr(A)
```

# Parâmetros

A
: matriz de polinômios

Ar
: matriz de polinômios

U
: matriz unimodular de polinômios

rk
: inteiro, posto normal def `A`

# Descrição

Triangularização da matriz de polinômios `A`.

`A` is `[m,n]`, `m <= n`.

`Ar=A*U`

Aviso: há eliminação de termos "pequenos" (ver código da função).

# Exemplos

```
x=poly(0,'x');
M=[x;x^2;2+x^3]*[1,x-2,x^4];
[Mu,U,rk]=htrianr(M)
det(U)
M*U(:,1:2)
```

# Ver Também

hrmt, colcompr

# Name

invr — inversão de matrizes (de razões de polinômios)

```
F = invr(H)
```

# Parâmetros

H
: matriz de polinômios ou de razões de polinômios

F
: matriz de polinômios ou de razões de polinômios

# Descrição

Se `H` é uma matriz de polinômios ou de razões de polinômios, `invr` computa `H^(-1)` utilizando o algoritmo de Leverrier (ver código da função).

# Exemplos

```
s=poly(0,'s')
H=[s,s*s+2;1-s,1+s]; invr(H)
[Num,den]=coffg(H);Num/den
H=[1/s,(s+1);1/(s+2),(s+3)/s];invr(H)
```

# Ver Também

glever, coffg, inv

# Name

lcm — mínimo múltiplo comum

```
[pp,fact]=lcm(p)
```

# Parâmetros

p
: vetor de polinômios

fact
: vetor de polinômios ou inteiros (tipo igual a 8)

pp
: polinômio ou inteiro

# Descrição

`pp=lcm(p)` computa o MMC `pp` do vetor de polinômios `p`.

`[pp,fact]=lcm(p)` computa, ainda, o vetor `fact` tal que:

`p.*fact=pp*ones(p)`

# Exemplos

```
//caso polinomial
s=poly(0,'s');
p=[s,s*(s+1)^2,s^2*(s+2)];
[pp,fact]=lcm(p);
p.*fact, pp

//caso inteiro
V=int32([2^2*3^5, 2^3*3^2,2^2*3^4*5]);
lcm(V)
```

# Ver Também

gcd, bezout

# Name

lcmdiag — fatoração diagonal por MMC

```
[N,D]=lcmdiag(H)
[N,D]=lcmdiag(H,flag)
```

# Parâmetros

H
: matriz de razões de polinômios

N
: matriz de polinômios

D
: matriz de polinômios diagonal

flag
: string: `'row'` ou `'col'` (padrão)

# Descrição

`[N,D]=lcmdiag(H,'row')` computa uma fatoração `D*H=N`, i.e. `H=D^(-1)*N` onde D é uma matriz diagonal com D(k,k)=MMC da k-ésima linha de H('den').

`[N,D]=lcmdiag(H)` ou `[N,D]=lcmdiag(H,'col)` retorna `H=N*D^(-1)` com matriz diagonal D e D(k,k)=MMC da k-ésima coluna de H('den')

# Exemplos

```
s=poly(0,'s');
H=[1/s,(s+2)/s/(s+1)^2;1/(s^2*(s+2)),2/(s+2)];
[N,D]=lcmdiag(H);
N/D-H
```

# Ver Também

lcm, gcd, bezout

# Name

ldiv — divisão longa entre matrizes de polinômios

```
[x]=ldiv(n,d,k)
```

# Parâmetros

n,d
: duas matrizes de polinômios reais

k
: inteiro

# Descrição

`x=ldiv(n,d,k)` fornece os `k` primeiros coeficientes da divisão longa de `n` por `d` i.e. the a expansão de Taylor da matriz de razões de polinômios `[nij(z)/dij(z)]` aproximando-se do infinito.

Os coeficientes de expansão de `nij/dij` são armazenados em `x((i-1)*n+k,j) k=1:n`

# Exemplos

```
wss=ssrand(1,1,3);[a,b,c,d]=abcd(wss);
wtf=ss2tf(wss);
x1=ldiv(numer(wtf),denom(wtf),5)
x2=[c*b;c*a*b;c*a^2*b;c*a^3*b;c*a^4*b]
wssbis=markp2ss(x1',5,1,1);
wtfbis=clean(ss2tf(wssbis))
x3=ldiv(numer(wtfbis),denom(wtfbis),5)
```

# Ver Também

arl2, markp2ss, pdiv

# Name

numer — numerador

```
num=numer(R)
```

# Parâmetros

R
: matriz de polinômios, de razões de polinômios ou de constantes

num
: matriz de polinômios

# Descrição

Função utilitária. `num=numer(R)` retorna o numerador `num` de uma matriz de funções racionais `R` (`R` também pode ser uma matriz de polinômios ou de constantes). `numer(R)` é equivalente a `R(2)`, `R('num')` ou `R.num`

# Ver Também

denom

# Name

pdiv — divisão polinomial

```
[R,Q]=pdiv(P1,P2)
[Q]=pdiv(P1,P2)
```

# Parâmetros

P1
: matriz de polinômios

P2
: polinômio ou matriz de polinômios

R,Q
: duas matrizes de polinômios

# Descrição

Divisão euclidiana elemento a elemento da matriz de polinômios `P1` pelo polinômio `P2` ou pela matriz de polinômios `P2`. `Rij` é a matriz de restos, `Qij` é a matriz de quocientes e `P1ij = Qij*P2 + Qij` ou `P1ij = Qij*P2ij + Qij`.

# Exemplos

```
x=poly(0,'x');
p1=(1+x^2)*(1-x);p2=1-x;
[r,q]=pdiv(p1,p2)
p2*q-p1
p2=1+x;
[r,q]=pdiv(p1,p2)
p2*q+r-p1
```

# Ver Também

ldiv, gcd

# Name

pol2des — conversão de matrizes de polinômios para formas descritoras

```
[N,B,C]=pol2des(Ds)
```

# Parâmetros

Ds
: matriz de polinômios

N, B, C
: três matrizes de reais

# Descrição

Dada a matriz de polinômios `Ds=D_0 +D_1 s +D_2 s^2 +... +D_k s^k`, `pol2des` retorna três matrizes `N, B, C`, com sendo `N` nilpotente tais que:

```
Ds = C (s*N-eye())^-1 B
```

# Exemplos

```
s=poly(0,'s');
G=[1,s;1+s^2,3*s^3];[N,B,C]=pol2des(G);
G1=clean(C*inv(s*N-eye())*B),G2=numer(G1)
```

# Ver Também

ss2des, tf2des

# Autor

F.D.;

# Name

pol2str — conversão de polinômio para string

```
[str]=pol2str(p)
```

# Parâmetros

p
: polinômio real

str
: string

# Descrição

Converte um polinômio para um string (função utilitária).

# Ver Também

string, pol2tex

# Name

polfact — fatores mínimos

```
[f]=polfact(p)
```

# Parâmetros

p
: polinômio

f
: vetor `[f0 f1 ... fn]` tal que `p=prod(f)`

  f0
  : constante

  fi
  : polinômio

# Descrição

`f=polfact(p)` retorna os valores mínimos de p, i.e., `f=[f0 f1 ... fn]` tal que `p=prod(f)`

# Ver Também

lcm, cmndred, factors

# Autor

S. Steer INRIA

# Name

residu — resíduo

```
[V]=residu(P,Q1,Q2)
```

# Parâmetros

P, Q1, Q2
: polinômios ou matrizes de polinômios de coeficientes reais ou complexos.

# Descrição

`V=residu(P,Q1,Q2)` retorna a matriz `V` tal que `V(i,j)` é a soma dos resíduos da fração racional `P(i,j)/(Q1(i,j)*Q2(i,j))` calculada nos zeros de `Q1(i,j)`.

`Q1(i,j)` e `Q2(i,j)` não devem possuir raízes em comum.

# Exemplos

```
s=poly(0,'s');
H=[s/(s+1)^2,1/(s+2)];N=numer(H);D=denom(H);
w=residu(N.*horner(N,-s),D,horner(D,-s));  //N(s) N(-s) / D(s) D(-s)
sqrt(sum(w))  //This is H2 norm
h2norm(tf2ss(H))
//
p=(s-1)*(s+1)*(s+2)*(s+10);a=(s-5)*(s-1)*(s*s)*((s+1/2)**2);
b=(s-3)*(s+2/5)*(s+3);
residu(p,a,b)+531863/4410    //Exato
z=poly(0,'z');a=z^3+0.7*z^2+0.5*z-0.3;b=z^3+0.3*z^2+0.2*z+0.1;
atild=gtild(a,'d');btild=gtild(b,'d');
residu(b*btild,z*a,atild)-2.9488038   //Exato
a=a+0*%i;b=b+0*%i;
real(residu(b*btild,z*a,atild)-2.9488038) //Caso complexo
```

# Ver Também

pfss, bdiag, roots, poly, gtild

# Autor

F.Delebecque INRIA

# Name

roots — raízes de polinômios

```
[x]=roots(p)
[x]=roots(p,'e')
```

# Parâmetros

p
: polinômio com coeficientes reais ou complexos ou vetor dos coeficientes do polinômio em ordem de graus decrescentes (para compatibilidade com o Matlab).

# Descrição

`x=roots(p)` retorna no vetor de complexos `x` as raízes do polinômio `p`. Para polinômios reais de grau<= 100 o algoritmo rápido RPOLY é utilizado. Em outros casos, as raízes são computadas como os autovalores da matriz companheira associada. Use `x=roots(p,'e')` para forçar este algoritmo em qualquer caso.

# Exemplos

```
p=poly([0,10,1+%i,1-%i],'x');
roots(p)
A=rand(3,3);roots(poly(A,'x'))    // avaliações por polinômios característicos
spec(A)
```

# Ver Também

poly, spec, companion

# Autor

Serge Steer (INRIA)

# References

O algoritmo RPOLY é descrito em "Algorithm 493: Zeros of a Real Polynomial", ACM TOMS Volume 1, edição 2 (Junho 1975), pp. 178-189

Jenkins, M. A. e Traub, J. F. (1970), A Three-Stage Algorithm for Real Polynomials Using Quadratic Iteration, SIAM J. Numer. Anal., 7(1970), 545-566.

Jenkins, M. A. e Traub, J. F. (1970), Principles for Testing Polynomial Zerofinding Programs. ACM TOMS 1, 1 (Março1975), pp. 26-34

# Funções Utilizadas

O código fonte de rpoly.f pode ser achado no diretório routines/control de uma distribuição fonte do Scilab. A computação de autovalores é feita utilizando-se os códigos do LAPACK DGEEV e ZGEEV.

# Name

rowcompr — compressão de linhas de uma matriz de polinômios

```
[X,rk,Ac]=rowcompr(A)
```

# Parâmetros

A
: matriz de polinômios

Y
: matriz de polinômios quadrada (base unimodular esquerda)

rk
: posto normal de `A`

Ac
: `Ac=X*A`, matriz de polinômios

# Descrição

Compressão de linhas da matriz de polinômios `A`.

`X` é uma base polinomial unimodular esquerda que comprime as linhas de `A`. `rk` é o posto normal de A.

Aviso: há eliminação de termos "pequenos" (utilize com cuidado!).

# Ver Também

colcompr

# Name

sfact — fatoração espectral em tempo discreto

```
F=sfact(P)
```

# Parâmetros

P
: matriz de polinômios reais

# Descrição

Acha `F`, um fator espectral de `P`. `P` é uma matriz de polinômios tal que cada raiz de `P` possui uma imagem simétrica em relação ao círculo unitário. O problema é singular se uma raiz estiver no círculo unitário.

`sfact(P)` retorna uma matriz de polinômios `F(z)` que é anti-estável e tal que

```
P = F(z)* F(1/z) *z^n
```

Para polinômios escalares um algorimto específico é implementado. Os algoritmos são implementados do livro de Kucera.

# Exemplos

```
//exemplo polinomial
z=poly(0,'z');
p=(z-1/2)*(2-z)
w=sfact(p);
w*numer(horner(w,1/z))

//exemplo matricial
F1=[z-1/2,z+1/2,z^2+2;1,z,-z;z^3+2*z,z,1/2-z];
P=F1*gtild(F1,'d');  //P é simétrica
F=sfact(P)
roots(det(P))
roots(det(gtild(F,'d')))  //as raízes estáveis
roots(det(F))             //as raízes anti-estáveis
clean(P-F*gtild(F,'d'))

//exemplo de uso de tempo contínuo
s=poly(0,'s');
p=-3*(s+(1+%i))*(s+(1-%i))*(s+0.5)*(s-0.5)*(s-(1+%i))*(s-(1-%i));p=real(p);

//p(s) = polinômio em s^2 , procura por f estável tal que p=f(s)*f(-s)
w=horner(p,(1-s)/(1+s));  // transformação bilinear w=p((1-s)/(1+s))
wn=numer(w);              //tomando o numerador
fn=sfact(wn);f=numer(horner(fn,(1-s)/(s+1))); //fator e transformação de volta
f=f/sqrt(horner(f*gtild(f,'c'),0));f=f*sqrt(horner(p,0));      //normalização
roots(f)    //f é estável
clean(f*gtild(f,'c')-p)    //f(s)*f(-s) é p(s)
```

# Ver Também

gtild, fspecg

# Name

simp — simplificação racional

```
[N1,D1]=simp(N,D)
H1=simp(H)
```

# Parâmetros

N,D
: polinômios reais ou matrizes de polinômios reais

H
: matriz de razões de polinômios (i.e matriz com entradas `n/d` ,`n` e `d` polinômios reais)

# Descrição

`[n1,d1]=simp(n,d)` calcula dois polinômios `n1` e `d1` tais que `n1/d1 = n/d`.

Se `N` e `D` são matrizes de polinômios, os cálculos são feitos elemento a elemento.

`H1=simp(H)` também é válido (cada entrada de `H` é simplificada em `H1`).

Cuidado:

-Nenhum limiar é dado i.e. `simp` não pode forçar uma simplificação.

-Para sistemas lineares dinâmicos que incluem integrador(es), a simplificação modifica o ganho estático. (`H(0)` para sistemas contínuos `H(1)` para sistemas discretos)

-Para dados complexos, `simp` retorna sua(s) entrada(s).

-Uma simplificação racional é chamada após aproximadamente cada operação sobre razões de polinômios. É possível alternar o modo de simplificação entre "on" e "off" (ligado e desligado) utilizando a função `simp_mode`.

# Exemplos

```
s=poly(0,'s');
[n,d]=simp((s+1)*(s+2),(s+1)*(s-2))

simp_mode(%F);hns=s/s
simp_mode(%T);hns=s/s
```

# Ver Também

roots, trfmod, poly, clean, simp_mode

# Name

simp_mode — modifica o modo de simplificação racional

```
mod=simp_mode()
simp_mode(mod)
```

# Parâmetros

mod
: um booleano

# Descrição

A simplificação racional é chamada após quase cada operação sobre razões de polinômios. É possível alternar o modo de simplificação entre "on" e "off" (ligado e desligado) utilizando a função `simp_mode`.

`simp_mod(%t)` ajusta o modo de simplificação racional para "on"

`simp_mod(%f)` ajusta o modo de simplificação racional para "off"

`mod=simp_mod()` retorna em mod o modo de simplificação racional corrente

# Exemplos

```
s=poly(0,'s');
mod=simp_mode()
simp_mode(%f);hns=s/s
simp_mode(%t);hns=s/s
simp_mode(mod);
```

# Ver Também

simp

# Name

sylm — matriz de Sylvester

```
[S]=sylm(a,b)
```

# Parâmetros

a,b
: duas matrizes de polinômios

S
: matriz

# Descrição

`sylm(a,b)` fornece a matriz de Sylvester associada aos polinômios `a` e `b`, i.e. a matriz `S` tal que:

`coeff( a*x + b*y )' = S * [coeff(x)';coeff(y)']`.

A dimensão de `S` é igual a `degree(a)+degree(b)`.

Se `a` e `b` são polinômios primos entre si, então

`rank(sylm(a,b))=degree(a)+degree(b))` e as instruções

```
u = sylm(a,b) \ eye(na+nb,1)
x = poly(u(1:nb),'z','coeff')
y = poly(u(nb+1:na+nb),'z','coeff')
```

computam os fatores de Bezout `x` e `y` de graus mínimos tais que `a*x+b*y = 1`

# Name

systmat — matriz sistema

```
[Sm]=systmat(Sl);
```

# Parâmetros

Sl
: sistema linear (lista `syslin` ) ou sistema descritor

Sm
: feixe de matrizes

# Descrição

Matriz sistema do sistema linear `Sl` (lista `syslin`) em forma de espaço de estados (função utilitária).

```
Sm = [-sI + A   B;
     [    C     D]
```

Para um sistema descritor (`Sl=list('des',A,B,C,D,E)`), `systmat` retorna:

```
Sm = [-sE + A   B;
     [    C     D]
```

# Ver Também

ss2des, sm2des, sm2ss

# Parte XXVIII. Processamento de Sinais

# Índice

5. How to ..... 1626
How to design an elliptic filter ..... 1627

# Capítulo 5. How to

# Nome

How to design an elliptic filter — How to design an elliptic filter (analog and digital)

# Description

The goal is to design a simple analog and digital elliptic filter.

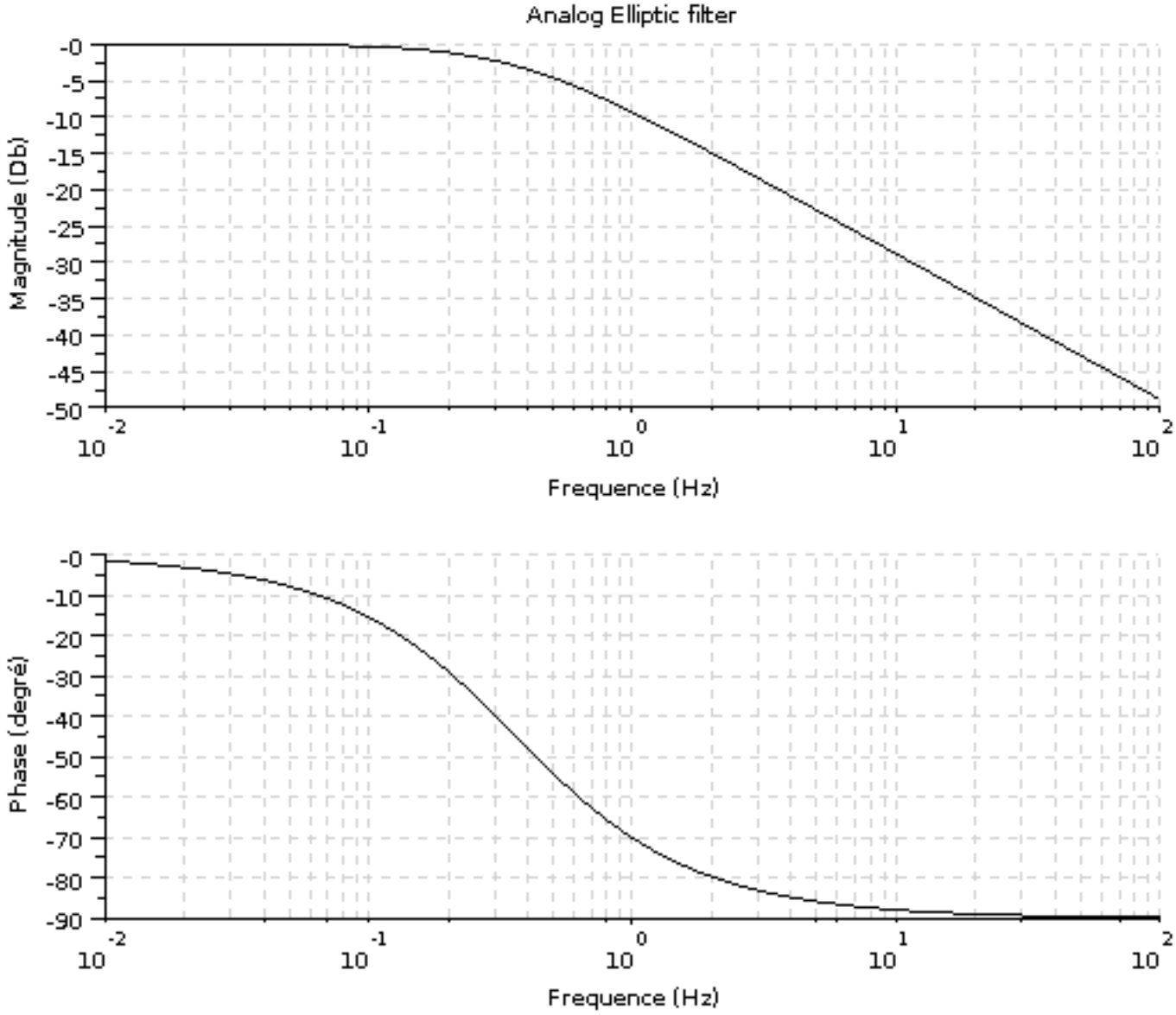
# Designing an analog elliptic filter

There are several possibilities to design an elliptic lowpass filter. We can use analpf or zpell. We will use zpell to produce the poles and zeros of the filter. Once we have got these poles and zeros, we will have to translate this representation into a syslin one.

And then, the filter can be represented in bode plot.

```
// analog elliptic (Bessel), order 2, cutoff 1 Hz
Epsilon = 3;  // ripple of filter in pass band (0<epsilon<1)
A       = 60; // attenuation of filter in stop band (A<1)
OmegaC  = 10; // pass band cut-off frequency in Hertz
OmegaR  = 50; // stop band cut-off frequency in Hertz

// Generate the filter
[_zeros,pols,gain] = zpell(3,60,10,50);

// Generate the equivalent linear system of the filter
num   = gain * real(poly(_zeros,'s'));;
den   = real(poly(pols,'s'));
elatf = syslin('c',num,den);

// Plot the resulting filter
bode(elatf,0.01,100);
title('Analog Elliptic filter');
```

Bode plot is only suited for analog filters.




If you want to design a highpass, bandpass or bandstop filter, you can first design a lowpass and then transfrom this lowpass filter using the trans function.

# Designing a digital elliptic filter

Now, let's focus on how to produce a digital lowpass elliptic filter.

We can produce two kinds of digital filters:

- an IIR (Infinite Impulse Response).

  To compute such a filter, we can use the following functions:

  - iir
  - eqiir

- a FIR (Finite Impulse Response).

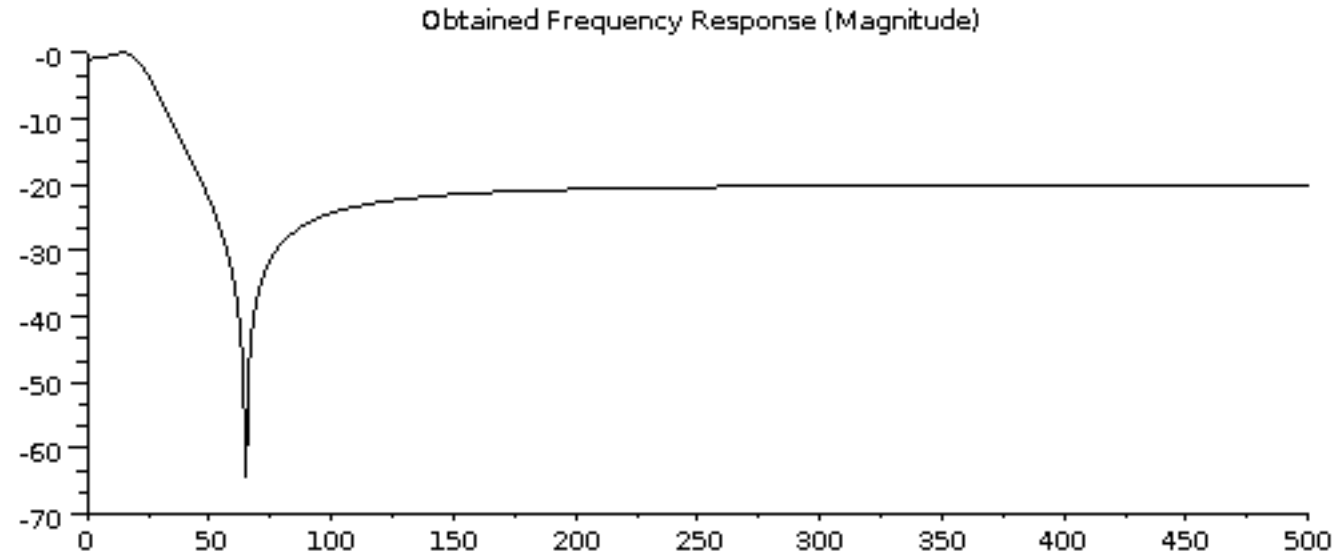
  To compute such a filter, we can use the following functions:

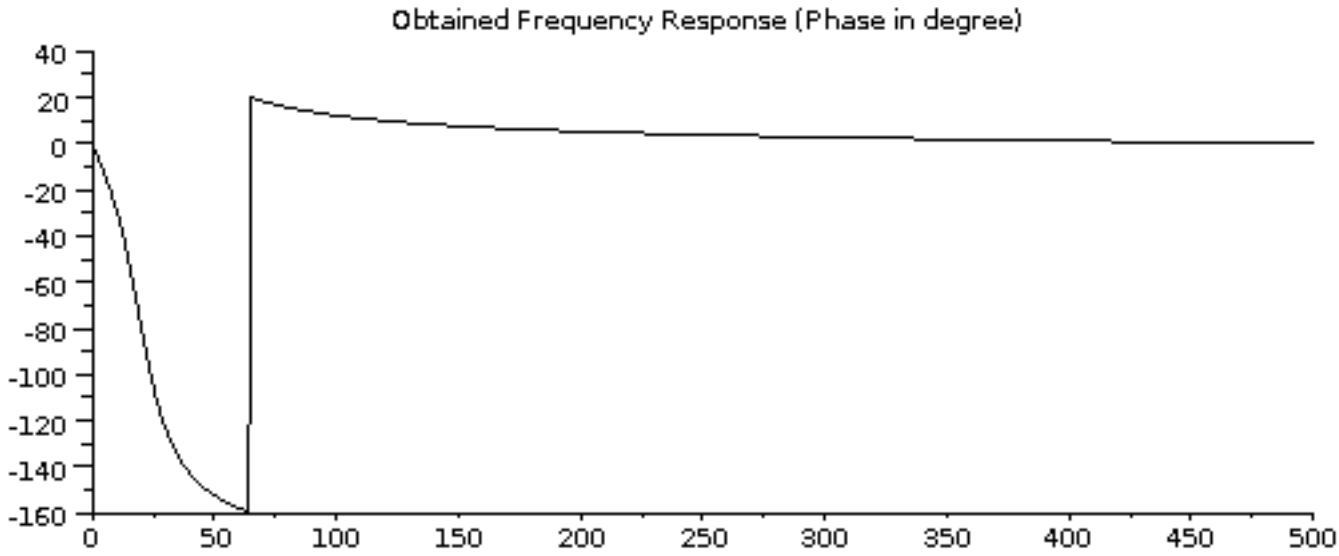
- eqfir
- ffilt
- wfir
- fsfirlin

For our demonstration, we will use the iir function.

```
Order   = 2; // The order of the filter
Fs      = 1000; // The sampling frequency
Fcutoff = 40;   // The cutoff frequency

// We design a low pass elliptic filter
hz = iir(Order,'lp','ellip',[Fcutoff/Fs/2 0],[0.1 0.1]);

// We compute the frequency response of the filter
[frq,repf]=repfreq(hz,0:0.001:0.5);
[db_repf, phi_repf] = dbphi(repf);

// And plot the bode like representation of the digital filter
subplot(2,1,1);
plot2d(Fs*frq,db_repf);
xtitle('Obtained Frequency Response (Magnitude)');
subplot(2,1,2);
plot2d(Fs*frq,phi_repf);
xtitle('Obtained Frequency Response (Phase in degree)');
```

Here is the representation of the digital elliptic filter.




To represent the filter in phase and magnitude, we need first to convert the discrete impulse response into magnitude and phase using the dbphi function. This convertion is done using a set of normalized frequencies.

# Filtering a signal using the digital filter

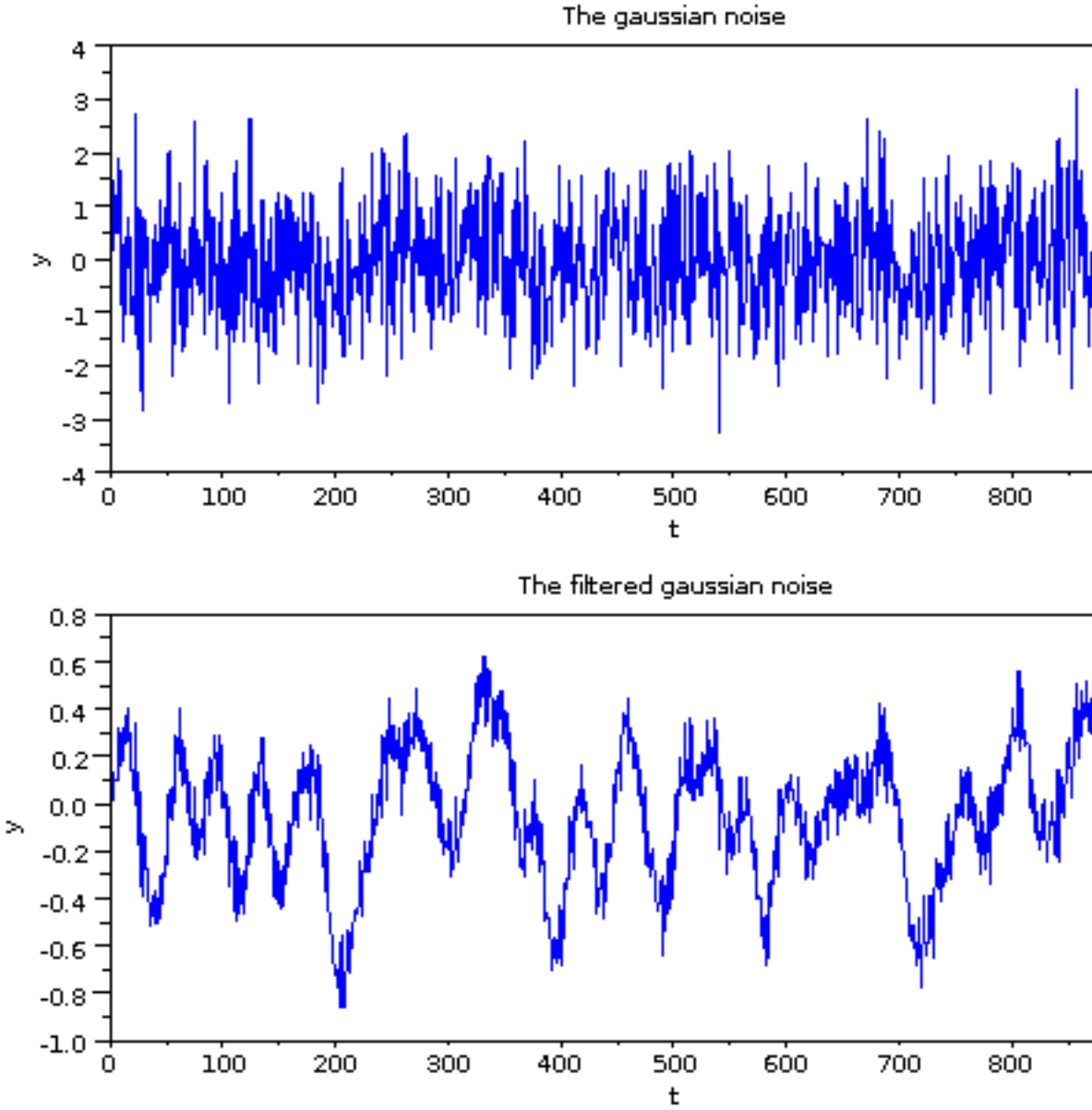
Designing a filter is a first step. Once done, this filter will be used to transform a signal. To get rid of some noise for example.

In the following examples, we will filter a gaussian noise.

```
rand('normal');
Input = rand(1,1000); // Produce a random gaussian noise
t     = 1:1000;

sl= tf2ss(hz); // From transfert function to syslin representation
y = flts(Input,sl); // Filter the signal
```

```
subplot(2,1,1);
plot(t,Input);
xtitle('The gaussian noise','t','y');
subplot(2,1,2);
plot(t,y);
xtitle('The filtered gaussian noise','t','y');
```

Here is the representation of the signal before and after filtering.




As we can see in the result, the high frequencies of the noise have been removed and it remains only the low frequencies. The signal is still noisy, but it contains mainly low frequencies.

# Filtering a signal using the analog filter

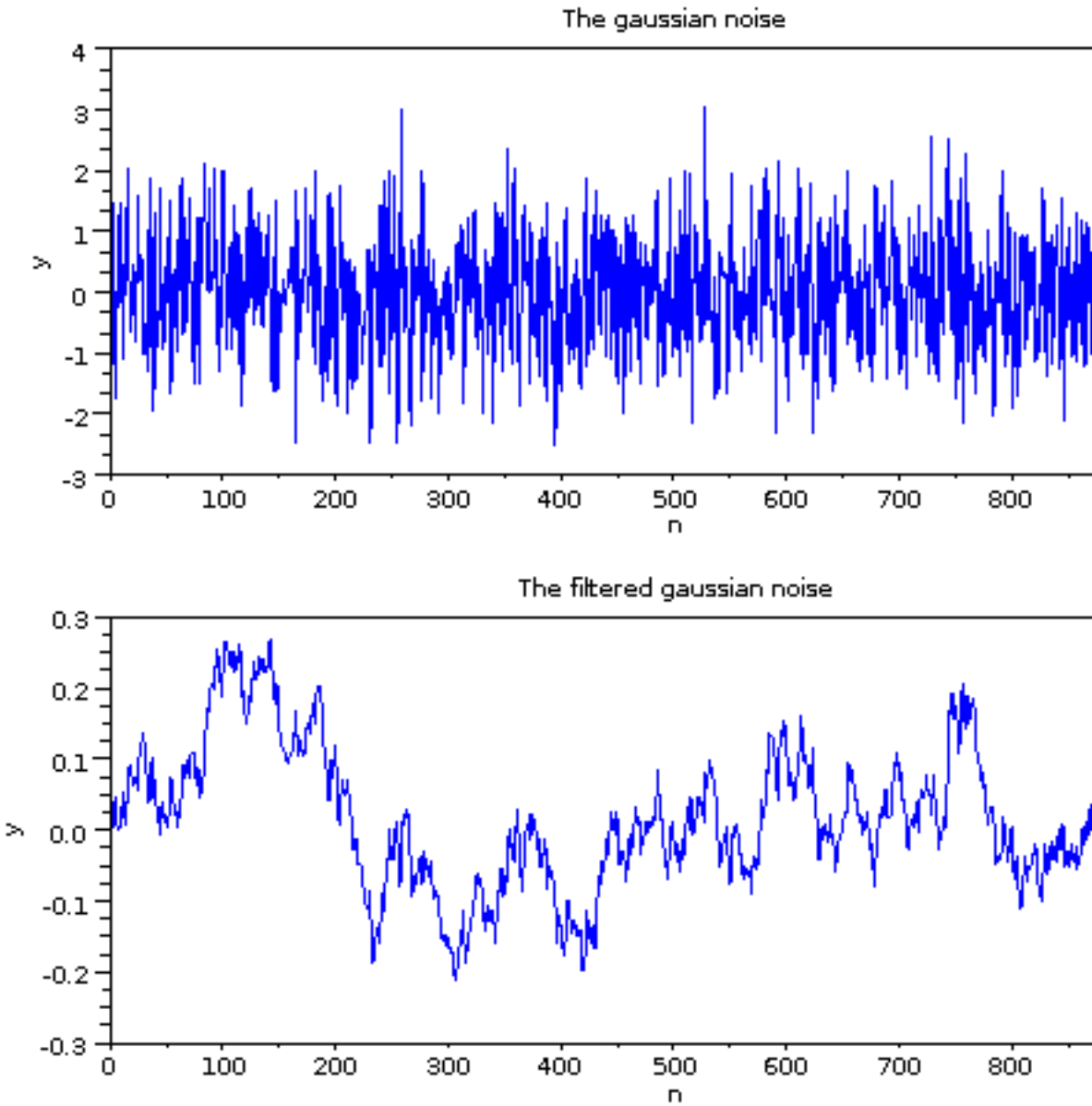
To filter a signal using an analog filter, we have two strategies:

- transform the analog filter into a discrete one using the dscr function
- apply the csim function to filter the signal

First, we try using the dscr + flts functions.

```
rand('normal');
Input = rand(1,1000); // Produce a random gaussian noise
n     = 1:1000; // The sample index

eldtf = dscr(elatf,1/100); // Discretization of the linear filter
y = flts(Input,eldtf); // Filter the signal

subplot(2,1,1);
plot(n,Input);
xtitle('The gaussian noise','n','y');
subplot(2,1,2);
plot(n,y);
xtitle('The filtered gaussian noise','n','y');
```

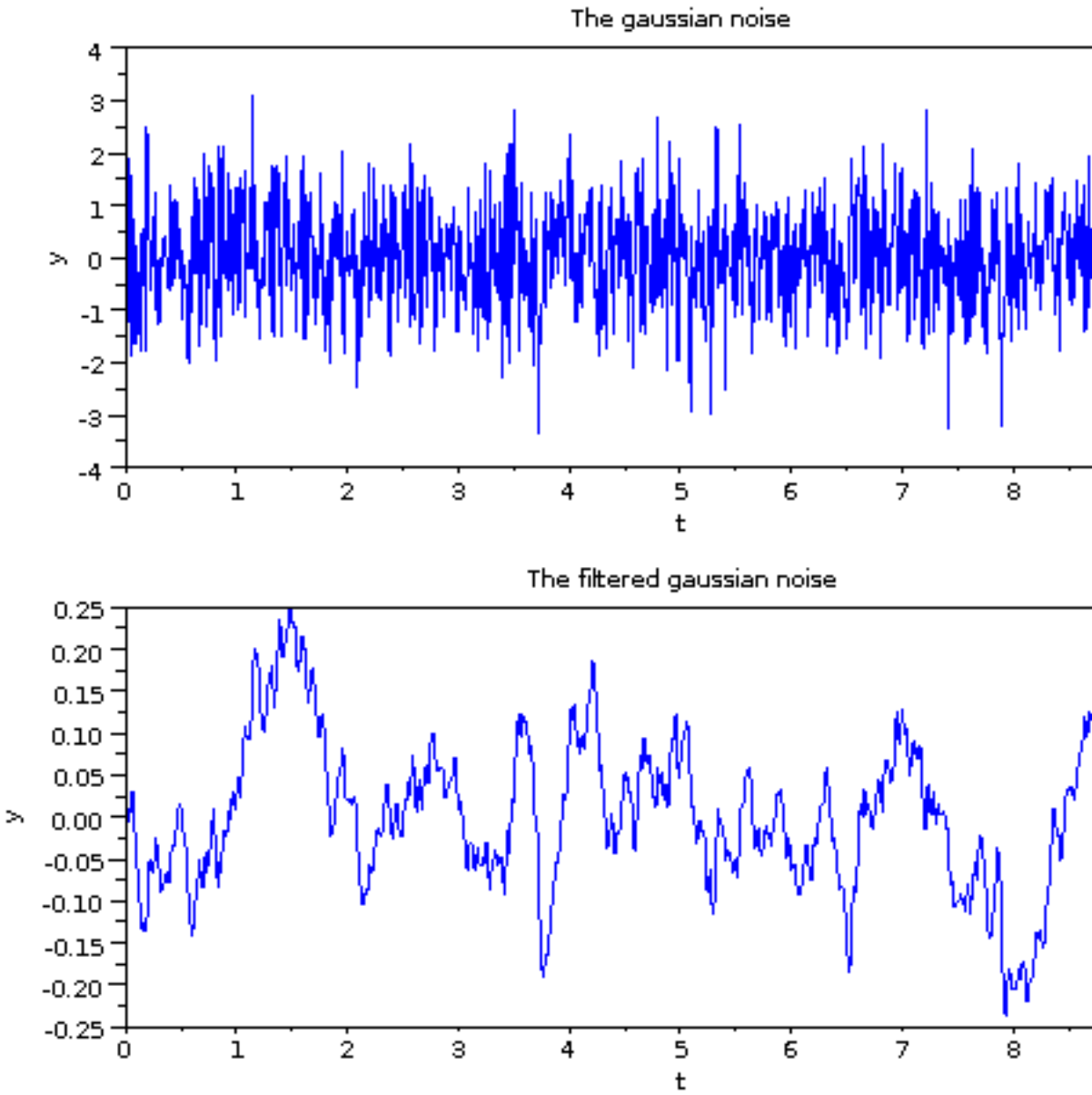
Here is the representation of the signal before and after filtering using the dscr + flts approach.



Next, we use the csim function.

```
rand('normal');
Input = rand(1,1000); // Produce a random gaussian noise
t     = 1:1000;
t     = t*0.01; // Convert sample index into time steps

y = csim(Input,t,elatf); // Filter the signal

subplot(2,1,1);
plot(t,Input);
xtitle('The gaussian noise','t','y');
subplot(2,1,2);
plot(t,y);
xtitle('The filtered gaussian noise','t','y');
```

Here is the representation of the signal before and after filtering using the csim approach.



The main difference between the dscr + flts approach and the csim approach: the dscr + flts uses samples whereas the csim functions uses time steps.

# See Also

bode, iir, poly, syslin, zpell, flts, tf2ss, dscr, csim, trans, analpf

# Name

Signal — Signal manual description

# Filters

analpf
: analog low-pass filter

buttmag
: squared magnitude response of a Butterworth filter

casc
: creates cascade realization of filter

cheb1mag
: square magnitude response of a type 1 Chebyshev filter

cheb2mag
: square magnitude response of a type 1 Chebyshev filter

chepol
: recursive implementation of Chebychev polynomial

convol
: convolution of 2 discrete series

ell1 mag
: squared magnitude of an elliptic filter

eqfir
: minimax multi-band, linear phase, FIR filter

eqiir
: design of iir filter

faurre
: optimal lqg filter.

lindquis
: optimal lqg filter lindquist algorithm

ffilt
: FIR low-pass,high-pass, band-pass, or stop-band filter

filter
: compute the filter model

find_freq
: parameter compatibility for elliptic filter design

findm
: for elliptic filter design

frmag
: magnitude of the frequency responses of FIR and IIR filters.

fsfirlin
: design of FIR, linear phase (frequency sampling technique)

fwiir
: optimum design of IIR filters in cascade realization,

iir
: designs an iir digital filter using analog filter designs.

iirgroup
: group delay of iir filter

iirlp
: Lp IIR filters optimization

group
: calculate the group delay of a digital filter

remezb
: minimax approximation of a frequency domain magnitude response.

kalm
: Kalman update and error variance

lev
: resolve the Yule-Walker equations :

levin
: solve recursively Toeplitz system (normal equations)

srfaur
: square-root algorithm for the algebraic Riccati equation.

srkf
: square-root Kalman filter algorithm

sskf
: steady-state Kalman filter

system
: generates the next observation given the old state

trans
: transformation of standardized low-pass filter into low-pass, high-pass, band-pass, stop-band.

wfir
: linear-phase windowed FIR low-pass, band-pass, high-pass, stop-band

wiener
: Wiener estimate (forward-backward Kalman filter formulation)

wigner
: time-frequency wigner spectrum of a signal.

window
: calculate symmetric window

zpbutt
: Butterworth analog filter

zpch1
: poles of a type 1 Chebyshev analog filter

zpch2
: poles and zeros of a type 2 Chebyshev analog filter

zpell
: poles and zeros of prototype lowpass elliptic filter

# Spectral estimation

corr
: correlation coefficients

cspect
: spectral estimation using the modified periodogram method.

czt
: chirp z-transform algorithm

intdec
: change the sampling rate of a 1D or 2D signal

mese
: calculate the maximum entropy spectral estimate

pspect
: auto and cross-spectral estimate

wigner
: Wigner-Ville time/frequency spectral estimation

# Transforms

dft
: discrete Fourier transform

fft
: fast flourier transform

hilb
: Hilbert transform centred around the origin.

hank
: hankel matrix of the covariance sequence of a vector process

mfft
: fft for a multi-dimensional signal

# Identification

lattn,lattp
: recursive solution of normal equations

phc
: State space realisation by the principal hankel component approximation method,

rpem
: identification by the recursive prediction error method

# Miscellaneous

sinc
: calculate the function sin(2*pi*fl*t)/(pi*t)

sincd
: calculates the function Sin(N*x)/Sin(x)

%k
: Jacobi's complete elliptic integral

%asn
: .TP the elliptic integral :

%sn
: Jacobi 's elliptic function with parameter m

bilt
: bilinear transform or biquadratic transform.

jmat
: permutes block rows or block columns of a matrix

# Name

analpf — create analog low-pass filter

```
[hs,pols,zers,gain]=analpf(n,fdesign,rp,omega)
```

# Parameters

n
: positive integer : filter order

fdesign
: string : filter design method : `'butt'` or `'cheb1'` or `'cheb2'` or `'ellip'`

rp
: 2-vector of error values for cheb1, cheb2 and ellip filters where only `rp(1)` is used for cheb1 case, only `rp(2)` is used for cheb2 case, and `rp(1)` and `rp(2)` are both used for ellip case. `0<rp(1),rp(2)<1`

- for cheb1 filters `1-rp(1)<ripple<1` in passband
- for cheb2 filters `0<ripple<rp(2)` in stopband
- for ellip filters `1-rp(1)<ripple<1` in passband `0<ripple<rp(2)` in stopband

omega
: cut-off frequency of low-pass filter in Hertz

hs
: rational polynomial transfer function

pols
: poles of transfer function

zers
: zeros of transfer function

gain
: gain of transfer function

# Description

Creates analog low-pass filter with cut-off frequency at omega.

`hs=gain*poly(zers,'s')/poly(pols,'s')`

# Examples

```
//Evaluate magnitude response of continuous-time system
hs=analpf(4,'cheb1',[.1 0],5)
fr=0:.1:15;
hf=freq(hs(2),hs(3),%i*fr);
hm=abs(hf);
plot(fr,hm)
```

# Authors

C. B.

# Name

bilt — bilinear or biquadratic transform SISO system given by a zero/poles representation

```
[npl,nzr,ngn] = bilt(pl,zr,gn,num,den)
```

# Parameters

pl
: a vector, the poles of the given system.

zr
: a vector, the zeros of the given system.

num
: a polynomial with degree equal to the degree of `den`, the numerator of the transform.

den
: a polynomial with degree 1 or 2, the denominator of the transform.

npl
: a vector, the poles of the transformed system.

nzr
: a vector, the zeros of the transformed system.

ngn
: a scalar, the gain of the transformed system.

# Description

function for calculating the gain poles and zeros which result from a bilinear transform or from a biquadratic transform. Used by the functions iir and trans.

# Examples

```
Hlp=iir(3,'lp','ellip',[0.1 0],[.08 .03]);
pl=roots(Hlp.den);
zr=roots(Hlp.num);
gn=coeff(Hlp.num,degree(Hlp.num))/coeff(Hlp.den,degree(Hlp.den));
z=poly(0,'z');
a=0.3;
num=z-a;
den=1-a*z;
[npl,nzr,ngn] = bilt(pl,zr,gn,num,den)

Hlpt=ngn*poly(nzr,'z','r')/poly(npl,'z','r')

//comparison with horner
horner(Hlp,num/den)
```

# Authors

Carey Bunks ;

## See Also

iir, trans, horner

# Name

buttmag — response of Butterworth filter

```
[h]=buttmag(order,omegac,sample)
```

# Parameters

order
: integer : filter order

omegac
: real : cut-off frequency in Hertz

sample
: vector of frequency where `buttmag` is evaluated

h
: Butterworth filter values at sample points

# Description

squared magnitude response of a Butterworth filter `omegac` = cutoff frequency ; `sample` = sample of frequencies

# Examples

```
//squared magnitude response of Butterworth filter
h=buttmag(13,300,1:1000);
mag=20*log(h)'/log(10);
plot2d((1:1000)',mag,[2],"011"," ",[0,-180,1000,20])
```

# Authors

F. D.

# Name

casc — cascade realization of filter from coefficients

```
[cels]=casc(x,z)
```

# Parameters

x
: (4xN)-matrix where each column is a cascade element, the first two column entries being the numerator coefficients and the second two column entries being the denominator coefficients

z
: string representing the cascade variable

cels
: resulting cascade representation

# Description

Creates cascade realization of filter from a matrix of coefficients (utility function).

# Examples

```
x=[1,2,3;4,5,6;7,8,9;10,11,12]
cels=casc(x,'z')
```

# Name

cepstrum — cepstrum calculation

```
fresp = cepstrum(w,mag)
```

# Parameters

w
: positive real vector of frequencies (rad/sec)

mag
: real vector of magnitudes (same size as `w`)

fresp
: complex vector

# Description

`fresp = cepstrum(w,mag)` returns a frequency response `fresp(i)` whose magnitude at frequency `w(i)` equals `mag(i)` and such that the phase of `freq` corresponds to a stable and minimum phase system. `w` needs not to be sorted, but minimal entry should not be close to zero and all the entries of `w` should be different.

# Examples

```
w=0.1:0.1:5;mag=1+abs(sin(w));
fresp=cepstrum(w,mag);
plot2d([w',w'],[mag(:),abs(fresp)])
```

# See Also

frfit

# Name

cheb1mag — response of Chebyshev type 1 filter

```
[h2]=cheb1mag(n,omegac,epsilon,sample)
```

# Parameters

n
: integer : filter order

omegac
: real : cut-off frequency

epsilon
: real : ripple in pass band

sample
: vector of frequencies where `cheb1mag` is evaluated

h2
: Chebyshev I filter values at sample points

# Description

Square magnitude response of a type 1 Chebyshev filter.

`omegac`=passband edge.

`epsilon`such that `1/(1+epsilon^2)`=passband ripple.

`sample`vector of frequencies where the square magnitude is desired.

# Examples

```
//Chebyshev; ripple in the passband
n=13;epsilon=0.2;omegac=3;sample=0:0.05:10;
h=cheb1mag(n,omegac,epsilon,sample);
plot2d(sample,h)
xtitle('','frequencies','magnitude')
```

# See Also

buttmag

# Name

cheb2mag — response of type 2 Chebyshev filter

```
[h2]=cheb2mag(n,omegar,A,sample)
```

# Parameters

n
: integer ; filter order

omegar
: real scalar : cut-off frequency

A
: attenuation in stop band

sample
: vector of frequencies where cheb2mag is evaluated

h2
: vector of Chebyshev II filter values at sample points

# Description

Square magnitude response of a type 2 Chebyshev filter.

`omegar` = stopband edge, `sample` = vector of frequencies where the square magnitude `h2` is desired.

# Examples

```
//Chebyshev; ripple in the stopband
n=10;omegar=6;A=1/0.2;sample=0.0001:0.05:10;
h2=cheb2mag(n,omegar,A,sample);
plot(sample,log(h2)/log(10),'frequencies','magnitude in dB')

//Plotting of frequency edges
minval=(-maxi(-log(h2)))/log(10);
plot2d([omegar;omegar],[minval;0],[2],"000");

//Computation of the attenuation in dB at the stopband edge
attenuation=-log(A*A)/log(10);
plot2d(sample',attenuation*ones(sample)',[5],"000")
```

# See Also

cheb1mag

# Name

chepol — Chebychev polynomial

```
[Tn]=chepol(n,var)
```

# Parameters

n
: integer : polynomial order

var
: string : polynomial variable

Tn
: polynomial in the variable `var`

# Description

Recursive implementation of Chebychev polynomial. `Tn=2*poly(0,var)*chepol(n-1,var)-chepol(n-2,var)` with `T0=1` and `T1=poly(0,var)`.

# Examples

```
chepol(4,'x')
```

# Authors

F. D.

# Name

convol — convolution

```
[y]=convol(h,x)
[y,e1]=convol(h,x,e0)
```

# Parameters

h
: a vector, first input sequence ("short" one)

x
: a vector, second input sequence ( "long" one)

e0
: a vector,old tail to overlap add (not used in first call)

y
: a vector, the convolution.

e1
: new tail to overlap add (not used in last call)

# Description

Calculates the convolution `y= h*x` of two discrete sequences by using the fft. The convolution is defined as follow:

$$y_k = \sum_j h_j * x_{k+1-j}$$

Overlap add method can be used.

USE OF OVERLAP ADD METHOD: For `x=[x1,x2,...,xNm1,xN]` First call is `[y1,e1]=convol(h,x1);` Subsequent calls : `[yk,ek]=convol(h,xk,ekm1)`; Final call : `[yN]=convol(h,xN,eNm1);` Finally `y=[y1,y2,...,yNm1,yN]`.

The algorithm based on the convolution definition is implemented for polynomial product: `y=convol(h,x)` is equivalent to `y=coeff(poly(h,'z','c')*poly(x,'z','c')` but much more efficient if `x` is a "long" array.

# Examples

```
x=1:3;
h1=[1,0,0,0,0];h2=[0,1,0,0,0];h3=[0,0,1,0,0];
x1=convol(h1,x),x2=convol(h2,x),x3=convol(h3,x),
convol(h1+h2+h3,x)
p1=poly(x,'x','coeff')
p2=poly(h1+h2+h3,'x','coeff')
p1*p2
```

# See Also

corr, fft, pspect

# Authors

F. D , C. Bunks Date 3 Oct. 1988; ;

# Name

corr — correlation, covariance

```
[cov,Mean]=corr(x,[y],nlags)
[cov,Mean]=corr('fft',xmacro,[ymacro],n,sect)

[w,xu]=corr('updt',x1,[y1],w0)
[w,xu]=corr('updt',x2,[y2],w,xu)
...
[wk]=corr('updt',xk,[yk],w,xu)
```

# Parameters

x
: a real vector

y
: a real vector, default value x.

nlags
: integer, number of correlation coefficients desired.

xmacro
: a scilab external (see below).

ymacro
: a scilab external (see below), default value xmacro

n
: an integer, total size of the sequence (see below).

sect
: size of sections of the sequence (see below).

xi
: a real vector

yi
: a real vector,default value xi.

cov
: real vector, the correlation coefficients

Mean
: real number or vector, the mean of x and if given y

# Description

Computes

```
         n - m
         ====
         \                                        1
cov(m) =  >   (x(k) - xmean) (y(m+k) - ymean) * ---
         /                                        n
         ====
         k = 1
```

for m=0,..,`nlag-1` and two vectors `x=[x(1),..,x(n)]` `y=[y(1),..,y(n)]`

Note that if x and y sequences are differents corr(x,y,...) is different with corr(y,x,...)

Short sequences
: `[cov,Mean]=corr(x,[y],nlags)` returns the first nlags correlation coefficients and Mean = `mean(x)` (mean of `[x,y]` if `y` is an argument). The sequence `x` (resp. `y`) is assumed real, and `x` and `y` are of same dimension n.

Long sequences
: `[cov,Mean]=corr('fft',xmacro,[ymacro],n,sect)` Here `xmacro` is either

- a function of type `[xx]=xmacro(sect,istart)` which returns a vector `xx` of dimension `nsect` containing the part of the sequence with indices from `istart` to `istart+sect-1`.
- a fortran subroutine or C procedure which performs the same calculation. (See the source code of `dgetx` for an example). `n` = total size of the sequence. `sect` = size of sections of the sequence. `sect` must be a power of 2. `cov` has dimension `sect`. Calculation is performed by FFT.

Updating method

```
[w,xu]=corr('updt',x1,[y1],w0)
[w,xu]=corr('updt',x2,[y2],w,xu)
 ...
wk=corr('updt',xk,[yk],w,xu)
```

With this calling sequence the calculation is updated at each call to `corr`.

```
w0 = 0*ones(1,2*nlags);
nlags = power of 2.
```

`x1,x2,...` are parts of `x` such that `x=[x1,x2,...]` and sizes of `xi` a power of 2. To get `nlags` coefficients a final fft must be performed `c=fft(w,1)/n`; `cov=c(1nlags)` (`n` is the size of `x (y)`). Caution: this calling sequence assumes that `xmean = ymean = 0`.

# Examples

```
x=%pi/10:%pi/10:102.4*%pi;
rand('seed');rand('normal');
y=[.8*sin(x)+.8*sin(2*x)+rand(x);.8*sin(x)+.8*sin(1.99*x)+rand(x)];
c=[];
for j=1:2,for k=1:2,c=[c;corr(y(k,:),y(j,:),64)];end;end;
c=matrix(c,2,128);cov=[];
for j=1:64,cov=[cov;c(:,(j-1)*2+1:2*j)];end;
rand('unif')

rand('normal');x=rand(1,256);y=-x;
deff('[z]=xx(inc,is)','z=x(is:is+inc-1)');
deff('[z]=yy(inc,is)','z=y(is:is+inc-1)');
[c,mxy]=corr(x,y,32);
x=x-mxy(1)*ones(x);y=y-mxy(2)*ones(y);  //centring
c1=corr(x,y,32);c2=corr(x,32);
```

```
norm(c1+c2,1)
[c3,m3]=corr('fft',xx,yy,256,32);
norm(c1-c3,1)
[c4,m4]=corr('fft',xx,256,32);
norm(m3,1),norm(m4,1)
norm(c3-c1,1),norm(c4-c2,1)
x1=x(1:128);x2=x(129:256);
y1=y(1:128);y2=y(129:256);
w0=0*ones(1:64);   //32 coeffs
[w1,xu]=corr('u',x1,y1,w0);w2=corr('u',x2,y2,w1,xu);
zz=real(fft(w2,1))/256;c5=zz(1:32);
norm(c5-c1,1)
[w1,xu]=corr('u',x1,w0);w2=corr('u',x2,w1,xu);
zz=real(fft(w2,1))/256;c6=zz(1:32);
norm(c6-c2,1)
rand('unif')

// test for Fortran or C external
//
deff('[y]=xmacro(sec,ist)','y=sin(ist:(ist+sec-1))');
x=xmacro(100,1);
[cc1,mm1]=corr(x,2^3);
[cc,mm]=corr('fft',xmacro,100,2^3);
[cc2,mm2]=corr('fft','corexx',100,2^3);
[maxi(abs(cc-cc1)),maxi(abs(mm-mm1)),maxi(abs(cc-cc2)),maxi(abs(mm-mm2))]

deff('[y]=ymacro(sec,ist)','y=cos(ist:(ist+sec-1))');
y=ymacro(100,1);
[cc1,mm1]=corr(x,y,2^3);
[cc,mm]=corr('fft',xmacro,ymacro,100,2^3);
[cc2,mm2]=corr('fft','corexx','corexy',100,2^3);
[maxi(abs(cc-cc1)),maxi(abs(mm-mm1)),maxi(abs(cc-cc2)),maxi(abs(mm-mm2))]
```

# See Also

fft

# Name

cspect — two sided cross-spectral estimate between 2 discrete time signals using the correlation method

```
[sm [,cwp]]=cspect(nlags,npoints,wtype,x [,y] [,wpar])
[sm [,cwp]]=cspect(nlags,npoints,wtype,nx [,ny] [,wpar])
```

# Parameters

x
: vector, the data of the first signal.

y
: vector, the data of the second signal. If `y` is omitted it is supposed to be equal to `x` (auto-correlation). If it is present, it must have the same numer of element than `x.`

nx
: a scalar : the number of points in the `x` signal. In this case the segments of the x signal are loaded by a user defined function named `getx` (see below).

ny
: a scalar : the number of points in the `y` signal. In this case the segments of the `y` signal are loaded by a user defined function named `gety` (see below). If present `ny` must be equal to `nx`.

nlags
: number of correlation lags (positive integer)

npoints
: number of transform points (positive integer)

wtype
: The window type

- `'re'`: rectangular
- `'tr'`: triangular
- `'hm'`: Hamming
- `'hn'` : Hanning
- `'kr'`: Kaiser,in this case the wpar argument must be given
- `'ch'`: Chebyshev, in this case the wpar argument must be given

wpar
: optional parameters for `Kaiser and Chebyshev windows:`

- 'kr': `wpar must be a strictly positive number`
- 'ch': `wpar` must be a 2 element vector `[main_lobe_width,side_lobe_height]with 0<main_lobe_width<.5`, and `side_lobe_height>0`

sm
: The power spectral estimate in the interval `[0,1]` of the normalized frequencies. It is a row array of size `npoints`. The array is real in case of auto-correlation and complex in case of cross-correlation.

cwp
: the unspecified Chebyshev window parameter in case of Chebyshev windowing, or an empty matrix.

# Description

Computes the cross-spectrum estimate of two signals `x` and `y` if both are given and the auto-spectral estimate of `x` otherwise. Spectral estimate obtained using the correlation method.

The cross spectrum of two signal x and y is defined to be

$$S_{xy}(\omega) = \frac{1}{N}\left(\sum_{n=0}^{N-1} x(n)e^{-i\omega n}\right)\left(\sum_{n=0}^{N-1} \bar{y}(n)e^{i\omega n}\right)$$

The correlation method calculates the spectral estimate as the Fourier transform of a modified estimate of the auto/cross correlation function. This auto/cross correlation modified estimate consist of repeatedly calculating estimates of the autocorrelation function from overlapping sub-segments if the data, and then averaging these estimates to obtain the result.

The number of points of the window is `2*nlags-1.`

For batch processing, the `x` and `y` data may be read segment by segment using the `getx` and `gety` user defined functions. These functions have the following calling sequence:

`xk=getx(ns,offset)` and `yk=gety(ns,offset)` where `ns` is the segment size and `offset` is the index of the first element of the segment in the full signal.

# Warning

For Scilab version up to 5.0.2 the returned value was the modulus of the current one.

# Reference

Oppenheim, A.V., and R.W. Schafer. Discrete-Time Signal Processing, Upper Saddle River, NJ: Prentice-Hall, 1999

# Examples

```
rand('normal');rand('seed',0);
x=rand(1:1024-33+1);
//make low-pass filter with eqfir
nf=33;bedge=[0 .1;.125 .5];des=[1 0];wate=[1 1];
h=eqfir(nf,bedge,des,wate);
//filter white data to obtain colored data
h1=[h 0*ones(1:maxi(size(x))-1)];
x1=[x 0*ones(1:maxi(size(h))-1)];
hf=fft(h1,-1);   xf=fft(x1,-1);yf=hf.*xf;y=real(fft(yf,1));
sm=cspect(100,200,'tr',y);
smsize=maxi(size(sm));fr=(1:smsize)/smsize;
plot(fr,log(sm))
```

# See Also

pspect, mese, corr

# Authors

C. Bunks INRIA

# Name

czt — chirp z-transform algorithm

```
[czx]=czt(x,m,w,phi,a,theta)
```

# Parameters

x
: input data sequence

m
: czt is evaluated at `m` points in z-plane

w
: magnitude multiplier

phi
: phase increment

a
: initial magnitude

theta
: initial phase

czx
: chirp z-transform output

# Description

chirp z-transform algorithm which calcultes the z-transform on a spiral in the z-plane at the points `[a*exp(j*theta)][w^kexp(j*k*phi)]` for `k=0,1,...,m-1`.

# Examples

```
a=.7*exp(%i*%pi/6);
[ffr,bds]=xgetech(); //preserve current context
rect=[-1.2,-1.2*sqrt(2),1.2,1.2*sqrt(2)];
t=2*%pi*(0:179)/179;xsetech([0,0,0.5,1]);
plot2d(sin(t)',cos(t)',[2],"012",' ',rect)
plot2d([0 real(a)]',[0 imag(a)]',[3],"000")
xsegs([-1.0,0;1.0,0],[0,-1.0;0,1.0])
w0=.93*exp(-%i*%pi/15);w=exp(-(0:9)*log(w0));z=a*w;
zr=real(z);zi=imag(z);
plot2d(zr',zi',[5],"000")
xsetech([0.5,0,0.5,1]);
plot2d(sin(t)',cos(t)',[2],"012",' ',rect)
plot2d([0 real(a)]',[0 imag(a)]',[-1],"000")
xsegs([-1.0,0;1.0,0],[0,-1.0;0,1.0])
w0=w0/(.93*.93);w=exp(-(0:9)*log(w0));z=a*w;
zr=real(z);zi=imag(z);
plot2d(zr',zi',[5],"000")
xsetech(ffr,bds); //restore context
```

# Authors

C. Bunks

# Name

detrend — remove constant, linear or piecewise linear trend from a vector

```
y = detrend(x)
y = detrend(x,flag)
y = detrend(x,flag,bp)
```

# Parameters

x
: vector or matrix of real or complex numbers (the signal to treat)

flag
: a string equal to "linear" (or "l") for linear or piecewise linear treatment or "constant" (or "c") for constant treatment.

bp
: the breakpoints to provide if you want a piecewise linear treatment.

y
: output, the signal x with the trend removed from it.

# Description

This function removes the constant or linear or piecewise linear trend from a vector `x`. In general this can be useful before a fourier analysis. If `x` is matrix this function removes the trend of each column of x.

When `flag` = "constant" or "c" `detrend` removes the constant trend (simply the mean of the signal) and when `flag` = "linear" or "l" the function removes the linear trend. By adding a third argument `bp` it is possible to remove a continuous *piecewise linear* trend. Note that the "instants" of the signal `x` goes from 0 to m-1 (m = length(x) if x is a vector and m = size(x,1) in case x is a matrix). So the breakpoints `bp(i)` must be reals in *[0 m-1]* (breakpoints outside are simply removed from `bp` vector).

The trend is got by a least square fit of `x` on the appropriate function space.

# Examples

```
// example #1
t = linspace(0,16*%pi,1000)';
x = -20 + t + 0.3*sin(0.5*t) + sin(t) + 2*sin(2*t) + 0.5*sin(3*t);
y = detrend(x);
clf()
plot2d(t,[x y],style=[2 5])
legend(["before detrend","after detrend"]);
xgrid()

// example #2
t = linspace(0,32*%pi,2000)';
x = abs(t-16*%pi) + 0.3*sin(0.5*t) + sin(t) + 2*sin(2*t) + 0.5*sin(3*t);
y = detrend(x,"linear",1000);
clf()
plot2d(t,[x y],style=[2 5])
legend(["before detrend","after detrend"]);
xgrid()
```

# Authors

Bruno Pincon

# Name

dft — discrete Fourier transform

```
[xf]=dft(x,flag);
```

# Parameters

x
: input vector

flag
: indicates dft (flag=-1) or idft (flag=1)

xf
: output vector

# Description

Function which computes dft of vector `x`.

# Examples

```
n=8;omega = exp(-2*%pi*%i/n);
j=0:n-1;F=omega.^(j'*j);  //Fourier matrix
x=1:8;x=x(:);
F*x
fft(x,-1)
dft(x,-1)
inv(F)*x
fft(x,1)
dft(x,1)
```

# See Also

fft

# Authors

C. B.

# Name

ell1mag — magnitude of elliptic filter

```
[v]=ell1mag(eps,m1,z)
```

# Parameters

eps
: passband ripple=`1/(1+eps^2)`

m1
: stopband ripple=`1/(1+(eps^2)/m1)`

z
: sample vector of values in the complex plane

v
: elliptic filter values at sample points

# Description

Function used for squared magnitude of an elliptic filter. Usually `m1=eps*eps/(a*a-1)`. Returns `v=real(ones(z)./(ones(z)+eps*eps*s.*s))` for `s=%sn(z,m1)`.

# Examples

```
deff('[alpha,BeTa]=alpha_beta(n,m,m1)',...
'if 2*int(n/2)==n then, BeTa=K1; else, BeTa=0;end;...
alpha=%k(1-m1)/%k(1-m);')
epsilon=0.1;A=10;  //ripple parameters
m1=(epsilon*epsilon)/(A*A-1);n=5;omegac=6;
m=find_freq(epsilon,A,n);omegar = omegac/sqrt(m)
%k(1-m1)*%k(m)/(%k(m1)*%k(1-m))-n   //Check...
[alpha,Beta]=alpha_beta(n,m,m1)
alpha*%asn(1,m)-n*%k(m1)      //Check
sample=0:0.01:20;
//Now we map the positive real axis into the contour...
z=alpha*%asn(sample/omegac,m)+Beta*ones(sample);
plot(sample,ell1mag(epsilon,m1,z))
```

# See Also

buttmag

# Name

eqfir — minimax approximation of FIR filter

```
[hn]=eqfir(nf,bedge,des,wate)
```

# Parameters

nf
: number of output filter points desired

bedge
: Mx2 matrix giving a pair of edges for each band

des
: M-vector giving desired magnitude for each band

wate
: M-vector giving relative weight of error in each band

hn
: output of linear-phase FIR filter coefficients

# Description

Minimax approximation of multi-band, linear phase, FIR filter

# Examples

```
hn=eqfir(33,[0 .2;.25 .35;.4 .5],[0 1 0],[1 1 1]);
[hm,fr]=frmag(hn,256);
plot(fr,hm),
```

# Authors

C. B.

# Name

eqiir — Design of iir filters

```
[cells,fact,zzeros,zpoles]=eqiir(ftype,approx,om,deltap,deltas)
```

# Parameters

ftype
: filter type (`'lp','hp','sb','bp'`)

approx
: design approximation (`'butt','cheb1','cheb2','ellip'`)

om
: 4-vector of cutoff frequencies (in radians) `om=[om1,om2,om3,om4]`, `0 <= om1 <= om2 <= om3 <= om4 <= pi`. When `ftype`='lp' or 'hp', `om3` and `om4` are not used and may be set to 0.

deltap
: ripple in the passband. `0<= deltap <=1`

deltas
: ripple in the stopband. `0<= deltas <=1`

cells
: realization of the filter as second order cells

fact
: normalization constant

zzeros
: zeros in the z-domain

zpoles
: poles in the z-domain

# Description

Design of iir filter based on syredi.

The filter obtained is `h(z)=fact`*product of the elements of `cells`.

That is `hz=fact*prod(cells.num)./prod(cells.den)`.

# Examples

```
[cells,fact,zzeros,zpoles]=eqiir('lp','ellip',[2*%pi/10,4*%pi/10],0.02,0.001)
h=fact*poly(zzeros,'z')/poly(zpoles,'z')
```

# See Also

eqfir, iir, syredi

# Name

faurre — filter computation by simple Faurre algorithm

```
[P,R,T]=faurre(n,H,F,G,R0)
```

# Parameters

n
: number of iterations.

H, F, G
: estimated triple from the covariance sequence of `y`.

R0
: E(yk*yk')

P
: solution of the Riccati equation after n iterations.

R, T
: gain matrix of the filter.

# Description

This function computes iteratively the minimal solution of the algebraic Riccati equation and gives the matrices `R` and `T` of the filter model. The algorithm tries to compute the solution P as the growing limit of a sequence of matrices Pn such that

```
                                          -1
Pn+1=F*Pn*F'+(G-F*Pn*h')*(R0-H*Pn*H')  *(G'-H*Pn*F')
       -1
P0=G*R0 *G'
```

Note that this method may not converge,especially when F has poles near the unit circle. Use preferably the srfaur function.

# See Also

srfaur , lindquist , phc

# Authors

G. Le V.

# Name

ffilt — coefficients of FIR low-pass

```
[x]=ffilt(ft,n,fl,fh)
```

# Parameters

ft
: filter type where `ft` can take the values

"lp"
: for low-pass filter

"hp"
: for high-pass filter

"bp"
: for band-pass filter

"sb"
: for stop-band filter

n
: integer (number of filter samples desired)

fl
: real (low frequency cut-off)

fh
: real (high frequency cut-off)

x
: vector of filter coefficients

# Description

Get `n` coefficients of a FIR low-pass, high-pass, band-pass, or stop-band filter. For low and high-pass filters one cut-off frequency must be specified whose value is given in `fl`. For band-pass and stop-band filters two cut-off frequencies must be specified for which the lower value is in `fl` and the higher value is in `fh`

# Authors

C. B.

# Name

fft — fast Fourier transform.
ifft — fast Fourier transform.

```
x=fft(a ,-1) or x=fft(a)
x=fft(a,1) or x=ifft(a)
x=fft(a,-1,dim,incr)
x=fft(a,1,dim,incr)
```

# Parameters

x
: real or complex vector. Real or complex matrix (2-dim fft)

a
: real or complex vector, matrix or multidimensionnal array.

dim
: integer

incr
: integer

# Description

Short syntax

direct
: `x=fft(a,-1)` or `x=fft(a)` gives a direct transform.

single variate
: If `a` is a vector a single variate direct FFT is computed that is:

x(k)=sum over m from 1 to n of a(m)*exp(-2i*pi*(m-1)*(k-1)/n)

for k varying from 1 to n (n=size of vector `a`).

(the `-1` argument refers to the sign of the exponent..., NOT to "inverse"),

multivariate
: If `a` is a matrix or or a multidimensionnal array a multivariate direct FFT is performed.

inverse
: `a=fft(x,1)` or `a=ifft(x)`performs the inverse transform normalized by `1/n`.

single variate
: If `a` is a vector a single variate inverse FFT is computed

multivariate
: If `a` is a matrix or or a multidimensionnal array a multivariate inverse FFT is performed.

Long syntax for multidimensional FFT
: `x=fft(a,-1,dim,incr)` allows to perform an multidimensional fft.

If a is a real or complex vector implicitly indexed by `j1,j2,..,jp` i.e. `a(j1,j2,..,jp)` where `j1` lies in `1:dim(1), j2` in `1:dim(2),...` one gets a p-variate FFT by calling p times `fft` as follows

```
incrk=1; x=a;
for k=1:p
  x=fft(x ,-1,dim(k),incrk)
  incrk=incrk*dim(k)
end
```

where `dimk` is the dimension of the current variable w.r.t which one is integrating and `incrk` is the increment which separates two successive `jk` elements in `a`.

In particular,if `a` is an mxn matrix, `x=fft(a,-1)` is equivalent to the two instructions:

`a1=fft(a,-1,m,1)` and `x=fft(a1,-1,n,m)`.

# Examples

```
//Comparison with explicit formula
//-----------------------------------
a=[1;2;3];n=size(a,'*');
norm(1/n*exp(2*%i*%pi*(0:n-1)'.*.(0:n-1)/n)*a -fft(a,1))
norm(exp(-2*%i*%pi*(0:n-1)'.*.(0:n-1)/n)*a -fft(a,-1))

//Frequency components of a signal
//-----------------------------------
// build a noides signal sampled at 1000hz  containing to pure frequencies
// at 50 and 70 Hz
sample_rate=1000;
t = 0:1/sample_rate:0.6;
N=size(t,'*'); //number of samples
s=sin(2*%pi*50*t)+sin(2*%pi*70*t+%pi/4)+grand(1,N,'nor',0,1);

y=fft(s);
//the fft response is symetric we retain only the first N/2 points
f=sample_rate*(0:(N/2))/N; //associated frequency vector
n=size(f,'*')
clf()
plot2d(f,abs(y(1:n)))
```

# See Also

corr

# Name

fft2 — two-dimension fast Fourier transform

```
y=fft2(x)
y=fft2(x,n,m)
```

# Parameters

x
: a vector/matrix/array (Real or Complex)

y
: a vector/matrix/array (Real or Complex)

m
: integer, number of rows.

n
: integer, number of columns.

# Description

```
This functions performs the two-dimension discrete Fourier transform.
```

`y=fft2(x)`y and x have the same size

`y=fft2(x,m,n):` If `m` (respectively `n`) is less than the rows number (respectively columns) of `x` then the `x` rows number (resp. columns) is truncated, else if m (resp. `n`) is more than the rows number (resp. columns) of `x` then `x` rows are completed by zero (resp. columns) .

if `x` is a matrix then `y` is a matrix, if `x` is a hypermatrix then `y` is a hypermatrix, with the size of the first dimension of `y` is equal to `m`, the size of the second dimension of `y` is equal to `n`, the size of the ith dimension of `y` (for i>2, case hypermatrix) equal to the size of the ith dimension of `x`. (i.e size(y,1)=m, size(y,2)=n and size(y,i)=size(x,i) for i>2)

# Examples

```
//Comparison with explicit formula
a=[1 2 3 ;4 5 6 ;7 8 9 ;10 11 12]
m=size(a,1)
n=size(a,2)

// fourier transform along the rows
for i=1:n
a1(:,i)=exp(-2*%i*%pi*(0:m-1)'.*.(0:m-1)/m)*a(:,i)
end

// fourier transform along the columns
for j=1:m
a2temp=exp(-2*%i*%pi*(0:n-1)'.*.(0:n-1)/n)*(a1(j,:)).'
a2(j,:)=a2temp.'
end
norm(a2-fft2(a))
```

## See Also

fft

# Name

fftshift — rearranges the fft output, moving the zero frequency to the center of the spectrum

```
y=fftshift(x [,job])
```

# Parameters

x
: real or complex vector or matrix.

y
: real or complex vector or matrix.

job
: integer, dimension selection, or string 'all'

# Description

if `x` results of an fft computation `y= fftshift(x)` or `y= fftshift(x,"all")` moves the zero frequency component to the center of the spectrum, which is sometimes a more convenient form.

If `x` is a vector of size n, y is the vector `x([n/2+1:n,1:n/2])`

If `x` is an `m` by `n` matrix `y` is the matrix `x([m/2+1:n,1:m/2],[n/2+1:n,1:n/2])`.

```
  [x11 x12]                   [x22 x21]
x=[       ]        gives   y=[       ]
  [x21 x22]                   [x12 x11]
```

`y= fftshift(x,n)` make the swap only along the `n`th dimension

# Examples

```
//make a signal
t=0:0.1:1000;
x=3*sin(t)+8*sin(3*t)+0.5*sin(5*t)+3*rand(t);
//compute the fft
y=fft(x,-1);

//display
clf();
subplot(2,1,1);plot2d(abs(y))
subplot(2,1,2);plot2d(fftshift(abs(y)))

//make a 2D image
t=0:0.1:30;
x=3*sin(t')*cos(2*t)+8*sin(3*t')*sin(5*t)+..
  0.5*sin(5*t')*sin(5*t)+3*rand(t')*rand(t);
//compute the fft
y=fft(x,-1);

//display
clf();
xset('colormap',hotcolormap(256))
```

```
subplot(2,1,1);Matplot(abs(y))
subplot(2,1,2);Matplot(fftshift(abs(y)))
```

## See Also

fft

# Name

filt_sinc — samples of sinc function

```
[x]=filt_sinc(n,fl)
```

# Parameters

n
: number of samples

fl
: cut-off frequency of the associated low-pass filter in Hertz.

x
: samples of the sinc function

# Description

Calculate n samples of the function `sin(2*pi*fl*t)/(pi*t)` for `t=-(n-1)/2:(n-1)/2` (i.e. centred around the origin).

# Examples

```
plot(filt_sinc(100,0.1))
```

# See Also

sincd

# Authors

C. B.;

# Name

filter — filters a data sequence using a digital filter

```
[y,zf] = filter(num,den,x [,zi])
```

# Parameters

num
: real vector : the coefficients of the filter numerator in decreasing power order, or a polynomial.

den
: real vector : the coefficients of the filter denominator in decreasing power order, or a polynomial.

x
: real row vector : the input signal

zi
: real row vector of length `max(length(a),length(b))-1`: the initial condition relative to a "direct form II transposed" state space representation. The default value is a vector filled with zeros.

y
: real row vector : the filtered signal.

zf
: real row vector : the final state. It can be used to filter a next batch of the input signal.

# Description

This function filters a data sequence using a digital filter using a "direct form II transposed" implementation

# References

Oppenheim, A. V. and R.W. Schafer. Discrete-Time Signal Processing, Englewood Cliffs, NJ: Prentice-Hall, 1989, pp. 311-312.

# See Also

flts , rtitr , ltitr

# Authors

Serge Steer, INRIA

# Name

find_freq — parameter compatibility for elliptic filter design

```
[m]=find_freq(epsilon,A,n)
```

# Parameters

epsilon
: passband ripple

A
: stopband attenuation

n
: filter order

m
: frequency needed for construction of elliptic filter

# Description

Search for m such that `n=K(1-m1)K(m)/(K(m1)K(1-m))` with

`m1=(epsilon*epsilon)/(A*A-1);`

If `m = omegar^2/omegac^2`, the parameters `epsilon,A,omegac,omegar` and `n` are then compatible for defining a prototype elliptic filter. Here, `K=%k(m)` is the complete elliptic integral with parameter `m`.

# See Also

%k

# Authors

F. D.

# Name

findm — for elliptic filter design

```
[m]=findm(chi)
```

# Description

Search for `m` such that `chi = %k(1-m)/%k(m)` (For use with `find_freq`).

# See Also

%k

# Authors

F. D.;

# Name

frfit — frequency response fit

```
sys=frfit(w,fresp,order)
[num,den]=frfit(w,fresp,order)
sys=frfit(w,fresp,order,weight)
[num,den]=frfit(w,fresp,order,weight)
```

# Parameters

w
: positive real vector of frequencies (Hz)

fresp
: complex vector of frequency responses (same size as `w`)

order
: integer (required order, degree of `den`)

weight
: positive real vector (default value `ones(w)`).

num,den
: stable polynomials

# Description

`sys=frfit(w,fresp,order,weight)` returns a bi-stable transfer function `G(s)=sys=num/den`, of of given `order` such that its frequency response `G(w(i))` matches `fresp(i)`, i.e. `freq(num,den,%i*w)` should be close to `fresp`. `weight(i)` is the weight given to `w(i)`.

# Examples

```
w=0.01:0.01:2;s=poly(0,'s');
G=syslin('c',2*(s^2+0.1*s+2), (s^2+s+1)*(s^2+0.3*s+1));
fresp=repfreq(G,w);
Gid=frfit(w,fresp,4);
frespfit=repfreq(Gid,w);
bode(w,[fresp;frespfit])
```

# See Also

frep2tf , factors , cepstrum , mrfit , freq , calfrq

# Name

frmag — magnitude of FIR and IIR filters

```
[xm,fr]=frmag(sys,npts)
[xm,fr]=frmag(num,den,npts)
```

# Parameters

sys
: a single input, single output discrete transfer function, or a polynomial or the vector of polynomial coefficients, the filter.

num
: a polynomial or the vector of polynomial coefficients, the numerator of the filter

den
: a polynomial or the vector of polynomial coefficients, the denominator of the filter (the default value is 1).

npts
: integer, the number of points in frequency response.

xm
: vector of magnitude of frequency response at the points `fr`.

fr
: points in the normalized frequency domain where magnitude is evaluated.

# Description

calculates the magnitude of the frequency responses of FIR and IIR filters. The filter description can be one or two vectors of coefficients, one or two polynomials, or a single output discrete transfer function.

the frequency discretisation is given by `fr=linspace(0,1/2,npts)`.

# Authors

Carey Bunks.

# Examples

```
hz=iir(3,'bp','cheb1',[.15 .25],[.08 .03]);
[hzm,fr]=frmag(hz,256);
plot(fr,hzm)
hz=iir(3,'bp','ellip',[.15 .25],[.08 .03]);
[hzm,fr]=frmag(hz,256);
plot(fr,hzm,'r')
```

# See Also

iir, eqfir, repfreq, calfrq, phasemag

# Name

fsfirlin — design of FIR, linear phase filters, frequency sampling technique

```
[hst]=fsfirlin(hd,flag)
```

# Parameters

hd
: vector of desired frequency response samples

flag
: is equal to 1 or 2, according to the choice of type 1 or type 2 design

hst
: vector giving the approximated continuous response on a dense grid of frequencies

# Description

function for the design of FIR, linear phase filters using the frequency sampling technique

# Examples

```
//
//Example of how to use the fsfirlin macro for the design
//of an FIR filter by a frequency sampling technique.
//
//Two filters are designed : the first (response hst1) with
//abrupt transitions from 0 to 1 between passbands and stop
//bands; the second (response hst2) with one sample in each
//transition band (amplitude 0.5) for smoothing.
//
hd=[zeros(1,15) ones(1,10) zeros(1,39)];//desired samples
hst1=fsfirlin(hd,1);//filter with no sample in the transition
hd(15)=.5;hd(26)=.5;//samples in the transition bands
hst2=fsfirlin(hd,1);//corresponding filter
pas=1/prod(size(hst1))*.5;
fg=0:pas:.5;//normalized frequencies grid
plot2d([1 1].*.fg(1:257)',[hst1' hst2']);

// 2nd example
hd=[0*ones(1,15) ones(1,10) 0*ones(1,39)];//desired samples
hst1=fsfirlin(hd,1);//filter with no sample in the transition
hd(15)=.5;hd(26)=.5;//samples in the transition bands
hst2=fsfirlin(hd,1);//corresponding filter
pas=1/prod(size(hst1))*.5;
fg=0:pas:.5;//normalized frequencies grid
n=prod(size(hst1))
plot(fg(1:n),hst1);
plot2d(fg(1:n)',hst2',[3],"000");
```

# See Also

ffilt , wfir

# Authors

G. Le Vey

# Name

group — group delay for digital filter

```
[tg,fr]=group(npts,a1i,a2i,b1i,b2i)
```

# Parameters

npts
: integer : number of points desired in calculation of group delay

a1i
: in coefficient, polynomial, rational polynomial, or cascade polynomial form this variable is the transfer function of the filter. In coefficient polynomial form this is a vector of coefficients (see below).

a2i
: in coeff poly form this is a vector of coeffs

b1i
: in coeff poly form this is a vector of coeffs

b2i
: in coeff poly form this is a vector of coeffs

tg
: values of group delay evaluated on the grid fr

fr
: grid of frequency values where group delay is evaluated

# Description

Calculate the group delay of a digital filter with transfer function h(z).

The filter specification can be in coefficient form, polynomial form, rational polynomial form, cascade polynomial form, or in coefficient polynomial form.

In the coefficient polynomial form the transfer function is formulated by the following expression

```
h(z)=prod(a1i+a2i*z+z**2)/prod(b1i+b2i*z+z^2)
```

# Examples

```
z=poly(0,'z');
h=z/(z-.5);
[tg,fr]=group(100,h);
plot(fr,tg)
```

# Authors

C. B.

# Name

hank — covariance to hankel matrix

```
[hk]=hank(m,n,cov)
```

# Parameters

m
: number of bloc-rows

n
: number of bloc-columns

cov
: sequence of covariances; it must be given as :[R0 R1 R2...Rk]

hk
: computed hankel matrix

# Description

this function builds the hankel matrix of size `(m*d,n*d)` from the covariance sequence of a vector process

# Examples

```
//Example of how to use the hank macro for
//building a Hankel matrix from multidimensional
//data (covariance or Markov parameters e.g.)
//
//This is used e.g. in the solution of normal equations
//by classical identification methods (Instrumental Variables e.g.)
//
//1)let's generate the multidimensional data under the form :
//  C=[c_0 c_1 c_2 .... c_n]
//where each bloc c_k is a d-dimensional matrix (e.g. the k-th correlation
//of a d-dimensional stochastic process X(t) [c_k = E(X(t) X'(t+k)], '
//being the transposition in scilab)
//
//we take here d=2 and n=64

c=rand(2,2*64)

//generate the hankel matrix H (with 4 bloc-rows and 5 bloc-columns)
//from the data in c

H=hank(4,5,c);
```

# See Also

toeplitz

# Authors

G. Le Vey

# Name

hilb — FIR approximation to a Hilbert transform filter

```
xh=hilb(n [,wtype [,par]])
```

# Parameters

n
: odd integer : number of points in filter

wtype
: string : window type `('re','tr','hn','hm','kr','ch')` (default `='re'`)

par
: window parameter for `wtype='kr' or 'ch'` default `par=[0 0]` see the function window for more help

xh
: Hilbert transform

# Description

Returns the first n points of an FIR approximation to a Hilbert transform filter centred around the origin.

The FIR filter is designed by appropraitely windowing the ideal impulse response `h(n)=(2/(n*pi))*(sin(n*pi/2))^2` for `n` not equal 0 and `h(0)=0`.

An approximation to an analytic signal generator can be built by designing an FIR (Finite Impulse Response) filter approximation to the Hilbert transform operator. The analytic signal can then be computed by adding the appropriately time-shifted real signal to the imaginary part generated by the Hilbert filter.

# References


`http://ieeexplore.ieee.org/iel4/78/7823/00330385.pdf?tp=&arnumber=330385&isnumber=7823`

A. Reilly, G. Frazer, and B. Boashash, "Analytic signal generation Tips and traps", IEEE Trans. Signal Processing, vol. 42, pp.3241-3245, Nov. 1994.


# See Also

window , hilbert

# Examples

```
plot(hilb(51))
```

# Authors

C. B.

# Name

hilbert — Discrete-time analytic signal computation of a real signal using Hilbert transform

```
x=hilbert(xr)
```

# Parameters

xr
: real vector : the real signal samples

x
: Complex vector: the discrete-time analytic signal.

# Description

Returns theanalytic signal, from a real data sequence.

The analytic signal `x = xr + i*xi` has a real part, `xr`, which is the original data, and an imaginary part, `xi`, which contains the Hilbert transform. The imaginary part is a version of the original real sequence with a 90° phase shift.

# References

`http://ieeexplore.ieee.org/iel5/78/16975/00782222.pdf?arnumber=782222`

Marple, S.L., "Computing the discrete-time analytic signal via FFT," IEEE Transactions on Signal Processing, Vol. 47, No.9 (September 1999), pp.2600-2603

# See Also

window , hil

# Examples

```
//compare the discrete-time analytic signal imaginary part of the impulse real
// with the FIR approximation of the Hilbert transform filter
m=25;
n=2*m+1;
y=hilbert(eye(n,1));
h=hilb(n)';
h=[h((m+1):$);h(1:m)];
plot([imag(y) h])
```

# Authors

C. B.

# Name

iir — iir digital filter

```
[hz]=iir(n,ftype,fdesign,frq,delta)
```

# Parameters

n
: positive number witn inteher value, the filter order.

ftype
: string specifying the filter type, the possible values are: `'lp'` for low-pass,`'hp'` for high pass,`'bp'` for band pass and `'sb'` for stop band.

fdesign
: string specifying the analog filter design, the possible values are: `'butt'`, `'cheb1'`, `'cheb2'` and `'ellip'`

frq
: 2-vector of discrete cut-off frequencies (i.e., `0<frq<.5`). For `'lp'` and `'hp'` filters only `frq(1)` is used. For `'bp'` and `'sb'` filters `frq(1)` is the lower cut-off frequency and `frq(2)` is the upper cut-off frequency

delta
: 2-vector of error values for `cheb1`, `cheb2`, and `ellip` filters where only `delta(1)` is used for `cheb1` case, only `delta(2)` is used for `cheb2` case, and `delta(1)` and `delta(2)` are both used for `ellip` case. `0<delta(1),delta(2)<1`

- for `cheb1` filters `1-delta(1)<ripple<1` in passband
- for `cheb2` filters `0<ripple<delta(2)` in stopband
- for `ellip` filters `1-delta(1)<ripple<1` in passband and `0<ripple<delta(2)` in stopband

# Description

function which designs an iir digital filter using analog filter designs and bilinear transformation .

# Examples

```
hz=iir(3,'bp','ellip',[.15 .25],[.08 .03]);
[hzm,fr]=frmag(hz,256);
plot2d(fr',hzm')
xtitle('Discrete IIR filter band pass  0.15&lt;fr&lt;0.25 ',' ',' ');
q=poly(0,'q');     //to express the result in terms of the delay operator q=z^-
hzd=horner(hz,1/q)
```

# See Also

eqfir, eqiir, analpf, bilt

# Authors

Carey Bunks

# Name

iirgroup — group delay Lp IIR filter optimization

```
[lt,grad]=iirgroup(p,r,theta,omega,wt,td)
[cout,grad,ind]=iirlp(x,ind,p,[flag],lambda,omega,ad,wa,td,wt)
```

# Parameters

r
: vector of the module of the poles and the zeros of the filters

theta
: vector of the argument of the poles and the zeros of the filters

omega
: frequencies where the filter specifications are given

wt
: weighting function for and the group delay

td
: desired group delay

lt, grad
: criterium and gradient values

# Description

optimization of IIR filters for the Lp criterium for the the group delay. (Rabiner & Gold pp270-273).

# Name

iirlp — Lp IIR filter optimization

```
[cost,grad,ind]=iirlp(x,ind,p,[flag],lambda,omega,ad,wa,td,wt)
```

# Parameters

x
: 1X2 vector of the module and argument of the poles and the zeros of the filters

flag
: string : `'a'` for amplitude, 'gd' for group delay; default case for amplitude and group delay.

omega
: frequencies where the filter specifications are given

wa,wt
: weighting functions for the amplitude and the group delay

lambda
: weighting (with `1-lambda`) of the costs (`'a'` and `'gd'` for getting the global cost.

ad, td
: desired amplitude and group delay

cost, grad
: criterium and gradient values

# Description

optimization of IIR filters for the Lp criterium for the amplitude and/or the group delay. (Rabiner & Gold pp270-273).

# Name

intdec — Changes sampling rate of a signal

```
[y]=intdec(x,lom)
```

# Parameters

x
: input sampled signal

lom
: For a 1D signal this is a scalar which gives the rate change. For a 2D signal this is a 2-Vector of sampling rate changes `lom`=(col rate change,row rate change)

y
: Output sampled signal

# Description

Changes the sampling rate of a 1D or 2D signal by the rates in lom

# Authors

C. B.

# Name

jmat — row or column block permutation

```
[j]=jmat(n,m)
```

# Parameters

n
: number of block rows or block columns of the matrix

m
: size of the (square) blocks

# Description

This function permutes block rows or block columns of a matrix

# Name

kalm — Kalman update

```
[x1,p1,x,p]=kalm(y,x0,p0,f,g,h,q,r)
```

# Parameters

f,g,h
: current system matrices

q, r
: covariance matrices of dynamics and observation noise

x0,p0
: state estimate and error variance at t=0 based on data up to t=-1

y
: current observation Output from the function is:

x1,p1
: updated estimate and error covariance at t=1 based on data up to t=0

x
: updated estimate and error covariance at t=0 based on data up to t=0

# Description

function which gives the Kalman update and error variance

# Authors

C. B.

# Name

lattn — recursive solution of normal equations

```
[la,lb]=lattn(n,p,cov)
```

# Parameters

n
: maximum order of the filter

p
: fixed dimension of the MA part. If `p= -1`, the algorithm reduces to the classical Levinson recursions.

cov
: matrix containing the `Rk`'s (`d*d` matrices for a d-dimensional process).It must be given the following way

la
: list-type variable, giving the successively calculated polynomials (degree 1 to degree n),with coefficients Ak

# Description

solves recursively on `n` (`p` being fixed) the following system (normal equations), i.e. identifies the AR part (poles) of a vector ARMA(n,p) process

where {`Rk;k=1,nlag`} is the sequence of empirical covariances

# Authors

G. Le V.

# Name

lattp — lattp

```
[la,lb]=lattp(n,p,cov)
```

# Description

see lattn

# Authors

G.Levey

# Name

lev — Yule-Walker equations (Levinson's algorithm)

```
[ar,sigma2,rc]=lev(r)
```

# Parameters

r
: correlation coefficients

ar
: auto-Regressive model parameters

sigma2
: scale constant

rc
: reflection coefficients

# Description

resolve the Yule-Walker equations

using Levinson's algorithm.

# Authors

C. B.

# Name

levin — Toeplitz system solver by Levinson algorithm (multidimensional)

```
[la,sig]=levin(n,cov)
```

# Parameters

n
: A scalar with integer value: the maximum order of the filter

cov
: A `(nlag*d) x d` matrix. It contains the `Rk` (`d x d` matrices for a `d`-dimensional process) stored in the following way :

$$\begin{pmatrix} R_0 \\ R_1 \\ R_2 \\ \vdots \\ R_{\text{nlags}} \end{pmatrix}$$

la
: A list, the successively calculated Levinson polynomials (degree 1 to `n`), with coefficients `Ak`

sig
: A list, the successive mean-square errors.

# Description

function which solves recursively on n the following Toeplitz system (normal equations)

$$\begin{pmatrix} I & -A_1 & \cdots & -A_n \end{pmatrix} * \begin{pmatrix} R_1 & R_2 & \cdots & R_n \\ R_0 & R_1 & \cdots & R_{n-1} \\ R_{-1} & R_0 & \cdots & R_{n-2} \\ \vdots & \vdots & \cdots & \vdots \\ R_{2-n} & R_{3-n} & \cdots & R_1 \\ R_{1-n} & R_{2-n} & \cdots & R_0 \end{pmatrix} = 0$$

where {`Rk;k=1:nlag`} is the sequence of `nlag` empirical covariances

# Examples

```
//We use the 'levin' macro for solving the normal equations
//on two examples: a one-dimensional and a two-dimensional process.
//We need the covariance sequence of the stochastic process.
//This example may usefully be compared with the results from
```

```
//the 'phc' macro (see the corresponding help and example in it)
//
//
//1) A one-dimensional process
//   -------------------------
//
//We generate the process defined by two sinusoids (1Hz and 2 Hz)
//in additive Gaussian noise (this is the observed process);
//the simulated process is sampled at 10 Hz (step 0.1 in t, underafter).

t1=0:.1:100;rand('normal');
y1=sin(2*%pi*t1)+sin(2*%pi*2*t1);y1=y1+rand(y1);plot(t1,y1);

//covariance of y1

nlag=128;
c1=corr(y1,nlag);
c1=c1';//c1 needs to be given columnwise (see the section PARAMETERS of this he

//compute the filter for a maximum order of n=10
//la is a list-type variable each element of which
//containing the filters of order ranging from 1 to n; (try varying n)
//in the d-dimensional case this is a matrix polynomial (square, d X d)
//sig gives, the same way, the mean-square error

n=15;
[la1,sig1]=levin(n,c1);

//verify that the roots of 'la' contain the
//frequency spectrum of the observed process y
//(remember that y is sampled -in our example
//at 10Hz (T=0.1s) so that we need to retrieve
//the original frequencies (1Hz and 2 Hz) through
//the log and correct scaling by the frequency sampling)
//we verify this for each filter order

for i=1:n, s1=roots(la1(i));s1=log(s1)/2/%pi/.1;

//now we get the estimated poles (sorted, positive ones only !)

s1=gsort(imag(s1));s1=s1(1:i/2);end;

//the last two frequencies are the ones really present in the observed
//process ---> the others are "artifacts" coming from the used model size.
//This is related to the rather difficult problem of order estimation.
//
//2) A 2-dimensional process
//   ------------------------
//(4 frequencies 1, 2, 3, and 4 Hz, sampled at 0.1 Hz :
//   |y_1|        y_1=sin(2*Pi*t)+sin(2*Pi*2*t)+Gaussian noise
// y=|   | with :
//   |y_2|        y_2=sin(2*Pi*3*t)+sin(2*Pi*4*t)+Gaussian noise

d=2;dt=0.1;
nlag=64;
t2=0:2*%pi*dt:100;
y2=[sin(t2)+sin(2*t2)+rand(t2);sin(3*t2)+sin(4*t2)+rand(t2)];
c2=[];
```

```
for j=1:2, for k=1:2, c2=[c2;corr(y2(k,:),y2(j,:),nlag)];end;end;
c2=matrix(c2,2,128);cov=[];
for j=1:64,cov=[cov;c2(:,(j-1)*d+1:j*d)];end;//covar. columnwise
c2=cov;

//in the multidimensional case, we have to compute the
//roots of the determinant of the matrix polynomial
//(easy in the 2-dimensional case but tricky if d>=3 !).
//We just do that here for the maximum desired
//filter order (n); mp is the matrix polynomial of degree n

[la2,sig2]=levin(n,c2);
mp=la2(n);determinant=mp(1,1)*mp(2,2)-mp(1,2)*mp(2,1);
s2=roots(determinant);s2=log(s2)/2/%pi/0.1;//same trick as above for 1D process
s2=gsort(imag(s2));s2=s2(1:d*n/2);//just the positive ones !

//There the order estimation problem is seen to be much more difficult !
//many artifacts ! The 4 frequencies are in the estimated spectrum
//but beneath many non relevant others.
```

# See Also

phc

# Authors

G. Le Vey

# Name

lindquist — Lindquist's algorithm

```
[P,R,T]=lindquist(n,H,F,G,R0)
```

# Parameters

n
: number of iterations.

H, F, G
: estimated triple from the covariance sequence of `y`.

R0
: E(yk*yk')

P
: solution of the Riccati equation after n iterations.

R, T
: gain matrices of the filter.

# Description

computes iteratively the minimal solution of the algebraic Riccati equation and gives the matrices `R` and `T` of the filter model, by the Lindquist's algorithm.

# See Also

srfaur , faurre , phc

# Authors

G. Le V.

# Name

mese — maximum entropy spectral estimation

```
[sm,fr]=mese(x [,npts]);
```

# Parameters

x
: Input sampled data sequence

npts
: Optional parameter giving number of points of `fr` and `sm` (default is 256)

sm
: Samples of spectral estimate on the frequency grid `fr`

fr
: npts equally spaced frequency samples in `[0,.5)`

# Description

Calculate the maximum entropy spectral estimate of `x`

# Authors

C. B.

# Name

mfft — multi-dimensional fft

```
[xk]=mfft(x,flag,dim)
```

# Parameters

x
: `x(i,j,k,...)` input signal in the form of a row vector whose values are arranged so that the i index runs the quickest, followed by the j index, etc.

flag
: (-1) FFT or (1) inverse FFT

dim
: dimension vector which gives the number of values of `x` for each of its indices

xk
: output of multidimensional fft in same format as for `x`

# Description

FFT for a multi-dimensional signal

For example for a three dimensional vector which has three points along its first dimension, two points along its second dimension and three points along its third dimension the row vector is arranged as follows

```
x=[x(1,1,1),x(2,1,1),x(3,1,1),
   x(1,2,1),x(2,2,1),x(3,2,1),
   x(1,1,2),x(2,1,2),x(3,1,2),
   x(1,2,2),x(2,2,2),x(3,2,2),
   x(1,1,3),x(2,1,3),x(3,1,3),
   x(1,2,3),x(2,2,3),x(3,2,3)]
```

and the `dim` vector is: `dim=[3,2,3]`

# Authors

C. B.

# Name

mrfit — frequency response fit

```
sys=mrfit(w,mag,order)
[num,den]=mrfit(w,mag,order)
sys=mrfit(w,mag,order,weight)
[num,den]=mrfit(w,mag,order,weight)
```

# Parameters

w
: positive real vector of frequencies (Hz)

mag
: real vector of frequency responses magnitude (same size as `w`)

order
: integer (required order, degree of `den`)

weight
: positive real vector (default value `ones(w)`).

num,den
: stable polynomials

# Description

`sys=mrfit(w,mag,order,weight)` returns a bi-stable transfer function `G(s)=sys=num/den`, of of given `order` such that its frequency response magnitude `abs(G(w(i)))` matches `mag(i)` i.e. `abs(freq(num,den,%i*w))` should be close to `mag`. `weight(i)` is the weigth given to `w(i)`.

# Examples

```
w=0.01:0.01:2;s=poly(0,'s');
G=syslin('c',2*(s^2+0.1*s+2),(s^2+s+1)*(s^2+0.3*s+1)); // syslin('c',Num,Den);
fresp=repfreq(G,w);
mag=abs(fresp);
Gid=mrfit(w,mag,4);
frespfit=repfreq(Gid,w);
plot2d([w',w'],[mag(:),abs(frespfit(:))])
```

# See Also

cepstrum , frfit , freq , calfrq

# Name

%asn — elliptic integral

```
[y]=%asn(x,m)
```

# Parameters

x
: upper limit of integral (`x>0`) (can be a vector)

m
: parameter of integral (`0<m<1`)

y
: value of the integral

# Description

Calculates the elliptic integral

If `x` is a vector, `y` is a vector of same dimension as `x`.

# Examples

```
m=0.8;z=%asn(1/sqrt(m),m);K=real(z);Ktilde=imag(z);
x2max=1/sqrt(m);
x1=0:0.05:1;x2=1:((x2max-1)/20):x2max;x3=x2max:0.05:10;
x=[x1,x2,x3];
y=%asn(x,m);
rect=[0,-Ktilde,1.1*K,2*Ktilde];
plot2d(real(y)',imag(y)',1,'011',' ',rect)

deff('y=f(t)','y=1/sqrt((1-t^2)*(1-m*t^2))');
intg(0,0.9,f)-%asn(0.9,m)  //Works for real case only!
```

# Authors

F. D.

# Name

%k — Jacobi's complete elliptic integral

```
[K]=%k(m)
```

# Parameters

m
: parameter of the elliptic integral `0<m<1` (`m` can be a vector)

K
: value of the elliptic integral from 0 to 1 on the real axis

# Description

Calculates Jacobi's complete elliptic integral of the first kind :

# References

Abramowitz and Stegun page 598

# Examples

```
m=0.4;
%asn(1,m)
%k(m)
```

# See Also

%asn

# Authors

F.D.

# Name

%sn — Jacobi 's elliptic function

```
[y]=%sn(x,m)
```

# Parameters

x
: a point inside the fundamental rectangle defined by the elliptic integral; `x` is a vector of complex numbers

m
: parameter of the elliptic integral (`0<m<1`)

y
: result

# Description

Jacobi 's sn elliptic function with parameter `m`: the inverse of the elliptic integral for the parameter `m`.

The amplitude am is computed in fortran and the addition formulas for elliptic functions are applied

# Examples

```
m=0.36;
K=%k(m);
P=4*K; //Real period
real_val=0:(P/50):P;
plot(real_val,real(%sn(real_val,m)))
clf();
KK=%k(1-m);
Ip=2*KK;
ima_val1=0:(Ip/50):KK-0.001;
ima_val2=(KK+0.05):(Ip/25):(Ip+KK);
z1=%sn(%i*ima_val1,m);z2=%sn(%i*ima_val2,m);
plot2d([ima_val1',ima_val2'],[imag(z1)',imag(z2)']);
xgrid(3)
```

# See Also

%asn , %k

# Authors

F. D.

# Name

phc — Markovian representation

```
[H,F,G]=phc(hk,d,r)
```

# Parameters

hk
: hankel matrix

d
: dimension of the observation

r
: desired dimension of the state vector for the approximated model

H, F, G
: relevant matrices of the Markovian model

# Description

Function which computes the matrices `H, F, G` of a Markovian representation by the principal hankel component approximation method, from the hankel matrix built from the covariance sequence of a stochastic process.

# Examples

```
//This example may usefully be compared with the results from
//the 'levin' macro (see the corresponding help and example)
//
//We consider the process defined by two sinusoids (1Hz and 2 Hz)
//in additive Gaussian noise (this is the observation);
//the simulated process is sampled at 10 Hz.

t=0:.1:100;rand('normal');
y=sin(2*%pi*t)+sin(2*%pi*2*t);y=y+rand(y);plot(t,y)

//covariance of y

nlag=128;
c=corr(y,nlag);

//hankel matrix from the covariance sequence
//(we can choose to take more information from covariance
//by taking greater n and m; try it to compare the results !

n=20;m=20;
h=hank(n,m,c);

//compute the Markov representation (mh,mf,mg)
//We just take here a state dimension equal to 4 :
//this is the rather difficult problem of estimating the order !
//Try varying ns !
//(the observation dimension is here equal to one)
```

```
ns=4;
[mh,mf,mg]=phc(h,1,ns);

//verify that the spectrum of mf contains the
//frequency spectrum of the observed process y
//(remember that y is sampled -in our example
//at 10Hz (T=0.1s) so that we need
//to retrieve the original frequencies through the log
//and correct scaling by the frequency sampling)

s=spec(mf);s=log(s);
s=s/2/%pi/.1;

//now we get the estimated spectrum
imag(s),
```

# See Also

levin

# Name

pspect — two sided cross-spectral estimate between 2 discrete time signals using the Welch's average periodogram method.

```
[sm [,cwp]]=pspect(sec_step,sec_leng,wtype,x [,y] [,wpar])
[sm [,cwp]]=pspect(sec_step,sec_leng,wtype,nx [,ny] [,wpar])
```

# Parameters

x
: vector, the time-domain samples of the first signal.

y
: vector, the time-domain samples of the second signal. If `y` is omitted it is supposed to be equal to `x` (auto-correlation). If it is present, it must have the same numer of element than `x`.

nx
: a scalar : the number of samples in the `x` signal. In this case the segments of the `x` signal are loaded by a user defined function named `getx` (see below).

ny
: a scalar : the number of samples in the `y` signal. In this case the segments of the y signal are loaded by a user defined function named `gety` (see below). If present `ny` must be equal to `nx`.

sec_step
: offset of each data window. The overlap `D` is given by sec_leng -sec_step. `if sec_step==sec_leng/2` 50% overlap is made. The overlap

sec_leng
: Number of points of the window.

wtype
: The window type

- `'re'`: rectangular
- `'tr'`: triangular
- `'hm'`: Hamming
- `'hn'` : Hanning
- `'kr'`: Kaiser,in this case the wpar argument must be given
- `'ch'`: Chebyshev, in this case the wpar argument must be given

wpar
: optional parameters for `Kaiser and Chebyshev windows:`

- 'kr': `wpar must be a strictly positive number`
- 'ch': `wpar` must be a 2 element vector `[main_lobe_width,side_lobe_height]with 0<main_lobe_width<.5`, and `side_lobe_height>0`

sm
: Two sided power spectral estimate in the interval `[0,1]` of the normalized frequencies. It is a row array with `sec_len` elements . The array is real in case of auto-correlation and complex in case of cross-correlation.

The associated normalized frequencies array is `linspace(0,1,sec_len)`.

cwp
: unspecified Chebyshev window parameter in case of Chebyshev windowing, or an empty matrix.

# Description

Computes the cross-spectrum estimate of two signals `x` and `y` if both are given and the auto-spectral estimate of `x` otherwise. Spectral estimate obtained using the modified periodogram method.

The cross spectrum of two signal `x` and `y` is defined to be

$$S_{xy}(\omega) = \frac{1}{N}\left(\sum_{n=0}^{N-1} x(n)e^{-i\omega n}\right)\left(\sum_{n=0}^{N-1} \bar{y}(n)e^{i\omega n}\right)$$

The modified periodogram method of spectral estimation repeatedly calculates the periodogram of windowed sub-sections of the data containes in `x` and `y` . These periodograms are then averaged together and normalized by an appropriate constant to obtain the final spectral estimate. It is the averaging process which reduces the variance in the estimate.

For batch processing, the `x` and `y` data may be read segment by segment using the `getx` and `gety` user defined functions. These functions have the following calling sequence:

`xk=getx(ns,offset)` and `yk=gety(ns,offset)` where `ns` is the segment size and `offset` is the index of the first element of the segment in the full signal.

# Reference

Oppenheim, A.V., and R.W. Schafer. Discrete-Time Signal Processing, Upper Saddle River, NJ: Prentice-Hall, 1999

# Examples

```
rand('normal');rand('seed',0);
x=rand(1:1024-33+1);

//make low-pass filter with eqfir
nf=33;bedge=[0 .1;.125 .5];des=[1 0];wate=[1 1];
h=eqfir(nf,bedge,des,wate);

//filter white data to obtain colored data
h1=[h 0*ones(1:maxi(size(x))-1)];
x1=[x 0*ones(1:maxi(size(h))-1)];
hf=fft(h1,-1); xf=fft(x1,-1);y=real(fft(hf.*xf,1));

//plot magnitude of filter
h2=[h 0*ones(1:968)];hf2=fft(h2,-1);hf2=real(hf2.*conj(hf2));
hsize=maxi(size(hf2));fr=(1:hsize)/hsize;plot(fr,log(hf2));

//pspect example
sm=pspect(100,200,'tr',y);smsize=maxi(size(sm));fr=(1:smsize)/smsize;
plot(fr,log(sm));
rand('unif');
```

# See Also

cspect, pspect, mese, window

# Authors

C. Bunks INRIA

# Name

remez — Remez exchange algorithm for the weighted chebyshev approximation of a continuous function with a sum of cosines.

```
an=remez(guess,mag,fgrid,weight)
```

# Parameters

guess
: real array of size `n+2 the` initial guess

fgrid
: real array of size `ng`: the grid of normalized frequency points in [0,.5[

mag
: real array of size `ng`: the desired magnitude on grid `fg`

weight
: real array of size `ng`: weighting function on error on grid `fg`

an
: real array of size `n`: cosine coefficients

# Description

Minimax approximation of a frequency domain magnitude response. The approximation takes the form

```
h = sum[a(i)*cos(weight)], i=1:n
```

An FIR, linear-phase filter can be obtained from the the output of `remez` by using the following commands:

```
hn(1:nc-1)=an(nc:-1:2)/2;
hn(nc)=an(1);
hn(nc+1:2*nc-1)=an(2:nc)/2;
```

This function is mainly intended to be called by the remezb function.

# Bibliography

E.W. Cheney, Introduction to Approximation Theory, McGraw-Hill, 1966

http://en.wikipedia.org/wiki/Remez_algorithm

# References

This function is based on the fortran code `remez.f` written by:

- James H. Mcclellan, department of electrical engineering and computer science, Massachusetts Institute of Technology, Cambridge, Massachussets. 02139
- Thomas W. Parks, department of electrical engineering, Rice university, Houston, Texas 77001

- Thomas W. Parks, department of electrical engineering, Rice university, Houston, Texas 77001

# Examples

```
nc=21;
ngrid=nc*250;
fgrid=.5*(0:(ngrid-1))/(ngrid-1);
mag(1:ngrid/2)=ones(1:ngrid/2);
mag(ngrid/2+1:ngrid)=0*ones(1:ngrid/2);
weight=ones(fgrid);
guess=round(1:ngrid/nc:ngrid);
guess(nc+1)=ngrid;
guess(nc+2)=ngrid;
an=remez(guess,mag,fgrid,weight);
```

# See Also

remezb, eqfir

# Name

remezb — Minimax approximation of magnitude response

```
[an]=remezb(nc,fg,ds,wt)
```

# Parameters

nc
: Number of cosine functions

fg
: Grid of frequency points in [0,.5)

ds
: Desired magnitude on grid `fg`

wt
: Weighting function on error on grid `fg`

an
: Cosine filter coefficients

# Description

Minimax approximation of a frequency domain magnitude response. The approximation takes the form `h = sum[a(n)*cos(wn)]` for n=0,1,...,nc. An FIR, linear-phase filter can be obtained from the the output of the function by using the following commands

```
hn(1:nc-1)=an(nc:-1:2)/2;
hn(nc)=an(1);
hn(nc+1:2*nc-1)=an(2:nc)/2;
```

# Examples

```
// Choose the number of cosine functions and create a dense grid
// in [0,.24) and [.26,.5)
nc=21;ngrid=nc*16;
fg=.24*(0:ngrid/2-1)/(ngrid/2-1);
fg(ngrid/2+1:ngrid)=fg(1:ngrid/2)+.26*ones(1:ngrid/2);

// Specify a low pass filter magnitude for the desired response
ds(1:ngrid/2)=ones(1:ngrid/2);
ds(ngrid/2+1:ngrid)=zeros(1:ngrid/2);

// Specify a uniform weighting function
wt=ones(fg);

// Run remezb
an=remezb(nc,fg,ds,wt)

// Make a linear phase FIR filter
hn(1:nc-1)=an(nc:-1:2)/2;
hn(nc)=an(1);
```

```
hn(nc+1:2*nc-1)=an(2:nc)/2;

// Plot the filter's magnitude response
plot(.5*(0:255)/256,frmag(hn,256));

// Choose the number of cosine functions and create a dense grid in [0,.5)
nc=21; ngrid=nc*16;
fg=.5*(0:(ngrid-1))/ngrid;

// Specify a triangular shaped magnitude for the desired response
ds(1:ngrid/2)=(0:ngrid/2-1)/(ngrid/2-1);
ds(ngrid/2+1:ngrid)=ds(ngrid/2:-1:1);

// Specify a uniform weighting function
wt=ones(fg);

// Run remezb
an=remezb(nc,fg,ds,wt)

// Make a linear phase FIR filter
hn(1:nc-1)=an(nc:-1:2)/2;
hn(nc)=an(1);
hn(nc+1:2*nc-1)=an(2:nc)/2;

// Plot the filter's magnitude response
plot(.5*(0:255)/256,frmag(hn,256));
```

# See Also

eqfir

# Authors

C. B.

# Name

rpem — Recursive Prediction-Error Minimization estimation

```
[w1,[v]]=rpem(w0,u0,y0,[lambda,[k,[c]]])
```

# Arguments

w0

`list(theta,p,l,phi,psi)` where:

theta

[a,b,c] is a real vector of order `3*n`

a,b,c

`a=[a(1),...,a(n)], b=[b(1),...,b(n)], c=[c(1),...,c(n)]`

p

(3*n x 3*n) real matrix.

phi,psi,l

real vector of dimension `3*n`

Applicable values for the first call:

```
theta=phi=psi=l=0*ones(1,3*n). p=eye(3*n,3*n)
```

u0

real vector of inputs (arbitrary size). (`u0($)` is not used by rpem)

y0

vector of outputs (same dimension as `u0`). (`y0(1)` is not used by rpem).

If the time domain is `(t0,t0+k-1)` the `u0` vector contains the inputs

`u(t0),u(t0+1),..,u(t0+k-1)` and `y0` the outputs

`y(t0),y(t0+1),..,y(t0+k-1)`

# Optional arguments

lambda

optional argument (forgetting constant) choosed close to 1 as convergence occur:

`lambda=[lambda0,alfa,beta]` evolves according to :

```
lambda(t)=alfa*lambda(t-1)+beta
```

with `lambda(0)=lambda0`

k

contraction factor to be chosen close to 1 as convergence occurs.

`k=[k0,mu,nu]` evolves according to:

```
k(t)=mu*k(t-1)+nu
```

with `k(0)=k0`.

c
: Large argument.(`c=1000` is the default value).

# Outputs:

w1
: Update for `w0`.

v
: sum of squared prediction errors on `u0, y0`.(optional).

In particular `w1(1)` is the new estimate of `theta`. If a new sample `u1, y1` is available the update is obtained by:

`[w2,[v]]=rpem(w1,u1,y1,[lambda,[k,[c]]])`. Arbitrary large series can thus be treated.

# Description

Recursive estimation of arguments in an ARMAX model. Uses Ljung-Soderstrom recursive prediction error method. Model considered is the following:

```
y(t)+a(1)*y(t-1)+...+a(n)*y(t-n)=
b(1)*u(t-1)+...+b(n)*u(t-n)+e(t)+c(1)*e(t-1)+...+c(n)*e(t-n)
```

The effect of this command is to update the estimation of unknown argument `theta=[a,b,c]` with

`a=[a(1),...,a(n)], b=[b(1),...,b(n)], c=[c(1),...,c(n)]`.

# Name

sincd — digital sinc function or Direchlet kernel

```
[s]=sincd(n,flag)
```

# Parameters

n
: integer

flag
: if `flag = 1` the function is centred around the origin; if `flag = 2` the function is delayed by `%pi/(2*n)`

s
: vector of values of the function on a dense grid of frequencies

# Description

function which calculates the function `Sin(N*x)/N*Sin(x)`

# Examples

```
plot(sincd(10,1))
```

# Authors

G. Le V.

# Name

srfaur — square-root algorithm

```
[p,s,t,l,rt,tt]=srfaur(h,f,g,r0,n,p,s,t,l)
```

# Parameters

h, f, g
: convenient matrices of the state-space model.

r0
: E(yk*yk').

n
: number of iterations.

p
: estimate of the solution after n iterations.

s, t, l
: intermediate matrices for successive iterations;

rt, tt
: gain matrices of the filter model after n iterations.

p, s, t, l
: may be given as input if more than one recursion is desired (evaluation of intermediate values of p).

# Description

square-root algorithm for the algebraic Riccati equation.

# Examples

```
//GENERATE SIGNAL
x=%pi/10:%pi/10:102.4*%pi;
rand('seed',0);rand('normal');
y=[1;1]*sin(x)+[sin(2*x);sin(1.9*x)]+rand(2,1024);

//COMPUTE CORRELATIONS
c=[];for j=1:2,for k=1:2,c=[c;corr(y(k,:),y(j,:),64)];end;end
c=matrix(c,2,128);

//FINDING H,F,G with 6 states
hk=hank(20,20,c);
[H,F,G]=phc(hk,2,6);

//SOLVING RICCATI EQN
r0=c(1:2,1:2);
[P,s,t,l,Rt,Tt]=srfaur(H,F,G,r0,200);

//Make covariance matrix exactly symetric
Rt=(Rt+Rt')/2
```

## See Also

phc , faurre , lindquist

# Name

srkf — square root Kalman filter

```
[x1,p1]=srkf(y,x0,p0,f,h,q,r)
```

# Parameters

f, h
: current system matrices

q, r
: covariance matrices of dynamics and observation noise

x0, p0
: state estimate and error variance at t=0 based on data up to t=-1

y
: current observation Output from the function is

x1, p1
: updated estimate and error covariance at t=1 based on data up to t=0

# Description

square root Kalman filter algorithm

# Authors

C. B.

# Name

sskf — steady-state Kalman filter

```
[xe,pe]=sskf(y,f,h,q,r,x0)
```

# Parameters

y
: data in form `[y0,y1,...,yn]`, `yk` a column vector

f
: system matrix dim(NxN)

h
: observations matrix dim(MxN)

q
: dynamics noise matrix dim(NxN)

r
: observations noise matrix dim(MxM)

x0
: initial state estimate

xe
: estimated state

pe
: steady-state error covariance

# Description

steady-state Kalman filter

# Authors

C. B.

# Name

syredi — Design of iir filters, syredi code interface

```
[fact,b2,b1,b0,c1,c0,zzeros,zpoles]=syredi(ityp,iapro,om,deltap,deltas)
```

# Parameters

itype
: integer, the filter type: 1 stands for low-pass, 2 for high-pass, 3 for band-pass, 4 for stop-band.

iappro
: integer, the design approximation type: 1 stands for butterworth, 2 for elliptic, 3 for Chebytchev1, 4 for Chebytchev2.

om
: 4-vector of cutoff frequencies (in radians) `om=[om1,om2,om3,om4]`,

  `0 <= om1 <= om2 <= om3 <= om4 <= pi`.

  When `ftype`='lp' or `'hp'`, `om3` and `om4` are not used and may be set to 0.

deltap
: a real scalar, the ripple in the passband. `0< deltap <1`

deltas
: a real scalar, the ripple in the stopband. `0< deltas <1`

gain
: scalar, the filter gain

b2
: real row vector, degree 2 coefficients of numerators.

b1
: real row vector, degree 1 coefficients of numerators.

b0
: real row vector, degree 0 coefficients of numerators.

c1
: real row vector, degree 1 coefficients of denominators.

c0
: real row vector, degree 0 coefficients of denominators.

zzeros
: complex row vector, filter zeros in the z-domain

zpoles
: complex row vector, filter poles in the z-domain

# Description

Computes iir filter approximation. The result is given as a set of second order transfer functions `Hi=(b0(i)+b1(i)*z+b2(i)*z^2)/(c0(i)+c1(i)*z+z^2)` and also as a poles, zeros, gain representation.

The filter obtained is `h=fact*H1*...*Hn`.

# Remark

This built-in function is mainly intended to be used by the eqiir function.

# References

The syredi code is derived from doredi package written by Guenter F. Dehner, Institut fuer Nachrichtentechnik Universitaet Erlangen-Nuernberg, Germany.

Dehner,G.F. 1979, DOREDI: Program for Design and Optimization of REcursive DIgital filters-Programs for Digital Signal Processing, ed:Digital Signal Processing committee of IEEE Acoustics, Speech and Signal Processing Society.

For DOREDI.f source code see http://michaelgellis.tripod.com/dsp/pgm25.html

# Examples

```
[fact,b2,b1,b0,c1,c0,zzeros,zpoles]=syredi(1,4,[2*%pi/10,4*%pi/10,0,0],0.02,0.0
h=fact*(b0+b1*%z+b2*%z^2)./(c0+c1*%z+%z^2)
```

# See Also

eqiir

# Name

system — observation update

```
[x1,y]=system(x0,f,g,h,q,r)
```

# Parameters

x0
: input state vector

f
: system matrix

g
: input matrix

h
: Output matrix

q
: input noise covariance matrix

r
: output noise covariance matrix

x1
: output state vector

y
: output observation

# Description

define system function which generates the next observation given the old state. System recursively calculated

```
x1=f*x0+g*u
y=h*x0+v
```

where `u` is distributed `N(0,q)` and `v` is distribute `N(0,r)`.

# Authors

C. B.

# Name

trans — low-pass to other filter transform

```
hzt=trans(hz,tr_type,frq)
hzt=trans(pd,zd,gd,tr_type,frq)
```

# Parameters

hz
: a single input single output discrete transfer function, the low pass filter

pd
: Vector of given filter poles

zd
: Vector of given filter zeros

gd
: scalar: the given filter gain

tr_type
: string, the type of transformation, see description for possible values

frq
: 2-vector of discrete cut-off frequencies (i.e.,`0<frq<.5`). see description for details.

hzt
: transformed filter transfert function.

# Description

function for transforming standardized low-pass filter given its poles-zeros_gain representation into one of the following filters:

tr_type='lp'
: low pass filter, the cutoff frequency is given by the first entry of `frq`, the second one is ignored.

tr_type='hp'
: high pass filter, the cutoff frequency is given by the first entry of `frq`, the second one is ignored.

tr_type='bp'
: band pass filter, the frequency range is given by `frq(1)` and `frq(2)`.

tr_type='sb'
: stop band filter, the frequency range is given by `frq(1)` and `frq(2)`.

# Used functions

bilt

# Examples

```
clf()
Hlp=iir(3,'lp','ellip',[0.1 0],[.08 .03]);
```

```
Hbp=trans(Hlp,'bp',[0.01 0.1]);
Hsb=trans(Hlp,'sb',[0.01 0.1])

clf();gainplot([Hlp;Hbp;Hsb],1d-3,0.48);
l=legend(['original low pass';'band pass';'stop band']);
l.legend_location="in_lower_left";
```

# Authors

Carey Bunks ;

# See Also

iir, bilt

# Name

wfir — linear-phase FIR filters

```
[wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)
```

# Parameters

ftype
: string : `'lp','hp','bp','sb'` (filter type)

forder
: Filter order (pos integer)(odd for `ftype='hp'` or `'sb'`)

cfreq
: 2-vector of cutoff frequencies (`0<cfreq(1),cfreq(2)<.5`) only `cfreq(1)` is used when `ftype='lp'` or `'hp'`

wtype
: Window type (`'re','tr','hm','hn','kr','ch'`)

fpar
: 2-vector of window parameters. Kaiser window `fpar(1)>0 fpar(2)=0`. Chebyshev window `fpar(1)>0, fpar(2)<0` or `fpar(1)<0, 0<fpar(2)<.5`

wft
: time domain filter coefficients

wfm
: frequency domain filter response on the grid fr

fr
: Frequency grid

# Description

Function which makes linear-phase, FIR low-pass, band-pass, high-pass, and stop-band filters using the windowing technique. Works interactively if called with no arguments.

# Authors

C. Bunks

# Name

wiener — Wiener estimate

```
[xs,ps,xf,pf]=wiener(y,x0,p0,f,g,h,q,r)
```

# Parameters

f, g, h
: system matrices in the interval `[t0,tf]`

  f
  : =`[f0,f1,...,ff]`, and `fk` is a nxn matrix

  g
  : =`[g0,g1,...,gf]`, and `gk` is a nxn matrix

  h
  : =`[h0,h1,...,hf]`, and `hk` is a mxn matrix

q, r
: covariance matrices of dynamics and observation noise

  q
  : =`[q0,q1,...,qf]`, and `qk` is a nxn matrix

  r
  : =`[r0,r1,...,rf]`, and `gk` is a mxm matrix

x0, p0
: initial state estimate and error variance

y
: observations in the interval `[t0,tf]`. `y=[y0,y1,...,yf]`, and `yk` is a column m-vector

xs
: Smoothed state estimate `xs= [xs0,xs1,...,xsf]`, and `xsk` is a column n-vector

ps
: Error covariance of smoothed estimate `ps=[p0,p1,...,pf]`, and `pk` is a nxn matrix

xf
: Filtered state estimate `xf= [xf0,xf1,...,xff]`, and `xfk` is a column n-vector

pf
: Error covariance of filtered estimate `pf=[p0,p1,...,pf]`, and `pk` is a nxn matrix

# Description

function which gives the Wiener estimate using the forward-backward Kalman filter formulation

# Authors

C. B.

# Name

wigner — 'time-frequency' wigner spectrum

```
[tab]=wigner(x,h,deltat,zp)
```

# Parameters

tab
: wigner spectrum (lines correspond to the time variable)

x
: analyzed signal

h
: data window

deltat
: analysis time increment (in samples)

zp
: length of FFT's. `%pi/zp` gives the frequency increment.

# Description

function which computes the 'time-frequency' wigner spectrum of a signal.

# Name

window — compute symmetric window of various type

```
win_l=window('re',n)
win_l=window('tr',n)
win_l=window('hn',n)
win_l=window('hm',n)
win_l=window('kr',n,alpha)
[win_l,cwp]=window('ch',n,par)
```

# Parameters

n
: window length

par
: parameter 2-vector `par=[dp,df])`, where `dp` (`0<dp<.5`) rules the main lobe width and `df` rules the side lobe height (`df>0`).

  Only one of these two value should be specified the other one should set equal to `-1`.

alpha
: kaiser window parameter `alpha >0`).

win
: window

cwp
: unspecified Chebyshev window parameter

# Description

function which calculates various symmetric window for Disgital signal processing

The Kaiser window is a nearly optimal window function. `alpha` is an arbitrary positive real number that determines the shape of the window, and the integer `n` is the length of the window.

By construction, this function peaks at unity for `k = n/2`, i.e. at the center of the window, and decays exponentially towards the window edges. The larger the value of `alpha`, the narrower the window becomes; `alpha = 0` corresponds to a rectangular window. Conversely, for larger `alpha` the width of the main lobe increases in the Fourier transform, while the side lobes decrease in amplitude. Thus, this parameter controls the tradeoff between main-lobe width and side-lobe area.

| alpha | window shape |
|---|---|
| 0 | Rectangular shape |
| 5 | Similar to the Hamming window |
| 6 | Similar to the Hanning window |
| 8.6 | Similar to the Blackman window |

The Chebyshev window minimizes the mainlobe width, given a particular sidelobe height. It is characterized by an equiripple behavior, that is, its sidelobes all have the same height.

The Hanning and Hamming windows are quite similar, they only differ in the choice of one parameter `alpha`: `w=alpha+(1 - alpha)*cos(2*%pi*x/(n-1))` `alpha` is equal to 1/2 in Hanning window and to 0.54 in Hamming window.

# Examples

```
// Hamming window
clf()
N=64;
w=window('hm',N);
subplot(121);plot2d(1:N,w,style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
subplot(122)
n=256;[W,fr]=frmag(w,n);
plot2d(fr,20*log10(W),style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))

//Kaiser window
clf()
N=64;
w=window('kr',N,6);
subplot(121);plot2d(1:N,w,style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
subplot(122)
n=256;[W,fr]=frmag(w,n);
plot2d(fr,20*log10(W),style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))

//Chebyshev window
clf()
N=64;
[w,df]=window('ch',N,[0.005,-1]);
subplot(121);plot2d(1:N,w,style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
subplot(122)
n=256;[W,fr]=frmag(w,n);
plot2d(fr,20*log10(W),style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
```

# See Also

wfir , frmag , ffilt

# Authors

Carey Bunks

# Bibliography

IEEE. Programs for Digital Signal Processing. IEEE Press. New York: John Wiley and Sons, 1979. Program 5.2.

# Name

yulewalk — least-square filter design

```
Hz = yulewalk(N,frq,mag)
```

# Parameters

N
: integer (order of desired filter)

frq
: real row vector (non-decreasing order), frequencies.

mag
: non negative real row vector (same size as frq), desired magnitudes.

Hz
: filter `B(z)/A(z)`

# Description

Hz = yulewalk(N,frq,mag) finds the N-th order iir filter

```
                  n-1         n-2
      B(z)   b(1)z     + b(2)z    + .... + b(n)
H(z)= ---- = ---------------------------------
                 n-1       n-2
      A(z)    z   + a(2)z    + .... + a(n)
```

which matches the magnitude frequency response given by vectors frq and mag. Vectors frq and mag specify the frequency and magnitude of the desired frequency response. The frequencies in frq must be between 0.0 and 1.0, with 1.0 corresponding to half the sample rate. They must be in increasing order and start with 0.0 and end with 1.0.

# Examples

```
f=[0,0.4,0.4,0.6,0.6,1];H=[0,0,1,1,0,0];Hz=yulewalk(8,f,H);
fs=1000;fhz = f*fs/2;
clf(0);xset('window',0);plot2d(fhz',H');
xtitle('Desired Frequency Response (Magnitude)')
[frq,repf]=repfreq(Hz,0:0.001:0.5);
clf(1);xset('window',1);plot2d(fs*frq',abs(repf'));
xtitle('Obtained Frequency Response (Magnitude)')
```

# Name

zpbutt — Butterworth analog filter

```
[pols,gain]=zpbutt(n,omegac)
```

# Parameters

n
: integer (filter order)

omegac
: real (cut-off frequency in Hertz)

pols
: resulting poles of filter

gain
: resulting gain of filter

# Description

computes the poles of a Butterworth analog filter of order `n` and cutoff frequency omegac transfer function H(s) is calculated by `H(s)=gain/real(poly(pols,'s'))`

# Authors

F. Delebecque INRIA

# Name

zpch1 — Chebyshev analog filter

```
[poles,gain]=zpch1(n,epsilon,omegac)
```

# Parameters

n
: integer (filter order)

epsilon
: real : ripple in the pass band (`0<epsilon<1`)

omegac
: real : cut-off frequency in Hertz

poles
: resulting filter poles

gain
: resulting filter gain

# Description

Poles of a Type 1 Chebyshev analog filter. The transfer function is given by :

```
H(s)=gain/poly(poles,'s')
```

# Authors

F.D.

# Name

zpch2 — Chebyshev analog filter

```
[zeros,poles,gain]=zpch2(n,A,omegar)
```

# Parameters

n
: integer : filter order

A
: real : attenuation in stop band (`A>1`)

omegar
: real : cut-off frequency in Hertz

zeros
: resulting filter zeros

poles
: resulting filter poles

gain
: Resulting filter gain

# Description

Poles and zeros of a type 2 Chebyshev analog filter gain is the gain of the filter

```
H(s)=gain*poly(zeros,'s')/poly(poles,'s')
```

# Authors

F.D.

# Name

zpell — lowpass elliptic filter

```
[zeros,poles,gain]=zpell(epsilon,A,omegac,omegar)
```

# Parameters

epsilon
: real : ripple of filter in pass band (`0<epsilon<1`)

A
: real : attenuation of filter in stop band (`A>1`)

omegac
: real : pass band cut-off frequency in Hertz

omegar
: real : stop band cut-off frequency in Hertz

zeros
: resulting zeros of filter

poles
: resulting poles of filter

gain
: resulting gain of filter

# Description

Poles and zeros of prototype lowpass elliptic filter. `gain` is the gain of the filter

# See Also

ell1mag , eqiir

# Authors

F.D.

# Parte XXIX. Matrizes Esparsas

# Name

full — conversão de matriz esparsa para cheia (completa)

```
X=full(sp)
```

# Parâmetros

sp
: matriz esparsa (ou cheia) de reais ou complexos

X
: matriz cheia (completa)

# Descrição

`X=full(sp)` converte a matriz esparsa `sp` em sua representação cheia (completa). (Se `sp` já é cheia, então `X` é igual a `sp`).

# Exemplos

```
sp=sparse([1,2;5,4;3,1],[1,2,3]);
A=full(sp)
```

# Ver Também

sparse, sprand, speye

# Name

ludel — função utilitária usada com lufact

```
ludel(hand)
```

# Parâmetros

hand
: manipulador para fatores lu esparsos (saída de lufact)

# Descrição

Esta função é usada de modo conjunto com `lufact`. Ela limpa o espaço de memória interna usado para guardar o resultado de `lufact`.

A seqüência de comandos `[p,r]=lufact(A);x=lusolve(p,b);ludel(p);` resolve o sistema linear esparso `A*x = b` e limpa `p`.

# Ver Também

sparse, lufact, luget

# Name

lufact — fatoração LU esparsa

```
[hand,rk]=lufact(A,prec)
```

# Parâmetros

A
: matriz quadrada esparsa

hand
: manipulador para fatores LU esparsos

rk
: inteiro (posto de A)

prec
: vetor de tamanho 2 `prec=[eps,reps]` fornecendo os limiares absoluto e relativo.

# Descrição

`[hand,rk]=lufact(A)` realiza a fatoração LU da matriz esparsa `A`. `hand` (sem exibição) é usado por `lusolve` (para resolver sistemas lineares) e `luget` (para retirar os fatores). `hand` deve ser limpo pelo comando: `ludel(hand);`

A matriz A não precisa ser de posto cheio, mas deve ser quadrada (desde que A é assumida como sendo esparsa, pode-se adicionar 0, se necessário, para quadrá-la).

eps :
: a magnitude absoluta que um elemento deve ter para ser considerado um candidato a pivô, exceto como último recurso. Este número deve ser posto de modo a ser significantemente menor que o menor elemento da diagonal que se espera estar localizado na matriz. O valor padrão é `%eps`.

reps :
: Este número determina qual será o limiar relativo do pivô. Deve estar entre 0 e 1. Se for 1, então o método de pivoteamento torna-se pivotação completa, que é muito lento e tende a completar a matriz. Se o número acertado é próximo de 0, o método de pivoteamento torna-se estritamente de Markowitz, sem limiar. O limiar de pivô é usado para eliminar candidatos a pivô que poderiam causar crescimento excessivo de elementos se fossem usados. Crescimento de elementos é a causa dos erros de arredondamento. Crescimento de elementos ocorre mesmo em matrizes bem condicionadas. Definir o reps como um número grande reduzirá o crescimento de elementos e os erros de arredondamento, mas colocá-lo muito grande aumentará muito o tempo de execução e resultará num grande número de preenchimentos. Se isto ocorrer, a precisão pode ficar prejudicada por causa do grande número de operações requeridas na matriz devido ao grande número de preenchimentos. 0.001 parece um bom valor, e é o valor default. O default é escolhido fornecendo-se um valor maior que 1 ou menor que ou igual a 0. Este valor deve ser aumentado e a matriz resolvida se o crescimento for excessivo. Mudar o limiar do pivô não melhora o desempenho em matrizes onde o crescimento é baixo, como é geralmente o caso de matrizes mal-condicionadas. reps foi escolhido para uso com matrizes quase diagonalmente dominantes como uma matriz de admissão de nó e nó modificado. Para estas matrizes, geralmente o melhor é usar pivotação diagonal. Para matrizes sem uma diagonal forte, geralmente é melhor usar um limiar maior, como 0.01 ou 0.1.

# Exemplos

```
a=rand(5,5);b=rand(5,1);A=sparse(a);
[h,rk]=lufact(A);
```

```
x=lusolve(h,b);a*x-b
ludel(h)
```

## Ver Também

sparse, lusolve, luget

# Name

luget — extração dos fatores LU esparsos

```
[P,L,U,Q]=luget(hand)
```

# Parâmetros

hand
: manipulador, saída de `lufact`

P
: matriz de permutação esparsa

L
: matriz esparsa, triangular infeiror se `hand` é obtida de uma matriz não-singular

U
: matriz quadrada triangular superior esparsa não-snigular preenchida com 1 ao longo da diagonal principal

Q
: matriz de permutação esparsa

# Descrição

`[P,L,U,Q]=luget(hand)` com `hand` obtida pelo comando `[hand,rk]=lufact(A)` com `A` uma matriz esparsa retorna quatro matrizes esparsas tais que `P*L*U*Q=A`.

A matriz A não precisa ser de posto cheio, mas deve ser quadrada (desde que A é assumida esparsa, pode-se adicionar 0, se necessário, para quadrar A).

Se `A` é singular, a matriz `L` é de colunas comprimidas (com `rk` colunas independentes não-nulas): a matriz não-singular esparsa `Q'*inv(U)` comprime em colunas `A`.

# Exemplos

```
a=rand(5,2)*rand(2,5);A=sparse(a);
[hand,rk]=lufact(A);[P,L,U,Q]=luget(hand);
full(L), P*L*U*Q-A
clean(P*L*U*Q-A)
ludel(hand)
```

# Ver Também

sparse, lusolve, luget, clean

# Name

lusolve — solucionador de sistemas lineares esparsos

```
lusolve(hand,b)
lusolve(A,b)
```

# Parâmetros

b
: matriz de reais completa

A
: matriz quadrada de reais esparsa e invertível

hand
: manipulador para fatores de lu esparsos previamente computados (saída de lufact)

# Descrição

`x=lusolve(hand,b)` resolve o sistema linear esparso `A*x = b`.

`[hand,rk]=lufact(A)` é a saída de lufact.

`x=lusolve(A,b)` resolve o sistema linear esparso `A*x = b`

# Exemplos

```
non_zeros=[1,2,3,4];rows_cols=[1,1;2,2;3,3;4,4];
sp=sparse(rows_cols,non_zeros);
[h,rk]=lufact(sp);x=lusolve(h,[1;1;1;1]);ludel(h)
rk,sp*x

non_zeros=[1,2,3,4];rows_cols=[1,1;2,2;3,3;4,4];
sp=sparse(rows_cols,non_zeros);
x=lusolve(sp,-ones(4,1));
sp*x
```

# Ver Também

sparse, lufact, slash, backslash

# Name

mtlb_sparse — converte matriz esparsa

```
Y=mtlb_sparse(X)
```

# Parâmetros

X
: matriz esparsa

Y
: matriz esparsa em formato Matlab

# Descrição

`Y=mtlb_sparse(X)` é usado para converter `X`, uma matriz esparsa Scilab, para formato Matlab. `Y` é uma variável de tipo 7, i.e., `type(Y)` é igual a 7. Esta função deve ser usada em mexfiles (um mexfile Matlab contendo matrizes esparsas pode ser usado apenas se as matrizes esparsas do Scilab forem convertidas para este formato). As funções `full` e `spget` funcionam com este formato.

Outras operações e funções usando este formato podem ficar sobrecarregadas com funções do Scilab usando o prefixo "%msp". Por exemplo, a função `%msp_p(x)` (ver diretório SCIDIR/macros/percent) é usada para exibir tais objetos "tipo 7".

# Exemplos

```
X=sparse(rand(2,2)); Y=mtlb_sparse(X);
Y, full(Y), [ij,v,mn]=spget(Y)
```

# Ver Também

full, spget

# Name

nnz — número de entradas não-nulas de uma matriz

```
n=nnz(X)
```

# Parâmetros

X
: matriz de reais ou complexos esparsa (ou cheia)

n
: inteiro, número de elementos não-nulos de X.

# Descrição

`nnz` conta o número de entradas não-nulas de uma matriz esparsa ou cheia.

# Exemplos

```
sp=sparse([1,2;4,5;3,10],[1,2,3]);
nnz(sp)
a=[1 0 0 0 2];
nnz(a)
```

# Ver Também

spget

# Name

sparse — definição de matriz esparsa

```
sp=sparse(X)
sp=sparse(ij,v [,mn])
```

# Parâmetros

X
: matriz completa (ou esparsa) de reais ou complexos

ij
: matriz de inteiros de duas colunas (índices das entradas não nulas)

v
: vetor

mn
: vetor de inteiros com duas entradas (dimensão de linha, dimensão de coluna)

sp
: matriz esparsa

# Descrição

`sparse` é usado para construir uma matriz esparsa. Apenas entradas não-nulas são armazenadas.

`sp = sparse(X)` converte uma matriz completa para sua forma esparsa retirando qualquer elemento nulo. (Se `X` já é esparsa `sp` é `X`).

`sp=sparse(ij,v [,mn])` constrói uma matriz esparsa `mn(1)`-por-`mn(2)` sparse matrix com `sp(ij(k,1),ij(k,2))=v(k)`. `ij` e `v` devem ter a mesma dimensão de coluna. Se o parâmetro opcional `mn` não for dado, as dimensões da matriz `sp` são os valores máximos de `ij(:,1)` e `ij(:,2)` respectivamente.

Operações (concatenação, adição, etc,) com matrizes esparsas são feitas usando a mesma sintaxe para matrizes completas.

Funções elementares também estão disponíveis (`abs,maxi,sum,diag,...`) para matrizes esparsas.

Operações mistas (completas-esparsas) são permitidas. Os resultados são completos ou esparsos dependendo das operações.

# Exemplos

```
sp=sparse([1,2;4,5;3,10],[1,2,3])
size(sp)
x=rand(2,2);abs(x)-full(abs(sparse(x)))
```

# Ver Também

full, spget, sprand, speye, lufact

# Name

spchol — fatoração esparsa de Cholesky

```
[R,P] = spchol(X)
```

# Parâmetros

X
: matriz simétrica, esparsa e positiva definida de reais

P
: matriz de permutação

R
: fator de Cholesky

# Descrição

`[R,P] = spchol(X)` produz uma matriz triângular inferior `R` tal que `P*R*R'*P' = X`.

# Exemplos

```
X=[
3.,  0.,  0.,  2.,  0.,  0.,  2.,  0.,  2.,  0.,  0. ;
0.,  5.,  4.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0. ;
0.,  4.,  5.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0. ;
2.,  0.,  0.,  3.,  0.,  0.,  2.,  0.,  2.,  0.,  0. ;
0.,  0.,  0.,  0. , 5.,  0.,  0.,  0.,  0.,  0.,  4. ;
0.,  0.,  0.,  0.,  0.,  4.,  0.,  3.,  0.,  3.,  0. ;
2.,  0.,  0.,  2.,  0.,  0.,  3.,  0.,  2.,  0.,  0. ;
0.,  0.,  0.,  0.,  0.,  3.,  0.,  4.,  0.,  3.,  0. ;
2.,  0.,  0.,  2.,  0.,  0.,  2.,  0.,  3.,  0.,  0. ;
0.,  0.,  0.,  0.,  0.,  3.,  0.,  3.,  0.,  4.,  0. ;
0.,  0.,  0.,  0.,  4.,  0.,  0.,  0.,  0.,  0.,  5.];
X=sparse(X);[R,P] = spchol(X);
max(P*R*R'*P'-X)
```

# Ver Também

sparse, lusolve, luget, chol

# Name

spcompack — converte uma representação de adjacência comprimida em representação de adjacência padrão

# Parâmetros

xadj
: vetor de inteiros de comprimento (n+1).

xlindx
: vetor de inteiros de comprimento n+1 (ponteiros).

lindx
: vetor de inteiros

adjncy
: vetor de inteiros

# Descrição

```
A função utilitária spcompak é usada para converter uma representação de adjacê
comprimida em uma representação de adjacência padrão.
```

# Exemplos

```
// A é a matriz esparsa:
A=[1,0,0,0,0,0,0;
   0,1,0,0,0,0,0;
   0,0,1,0,0,0,0;
   0,0,1,1,0,0,0;
   0,0,1,1,1,0,0;
   0,0,1,1,0,1,0;
   0,0,1,1,0,1,1];
A=sparse(A);
//Para esta matriz a representação padrão de adjacência é dada por:
xadj=[1,2,3,8,12,13,15,16];
adjncy=[1, 2, 3,4,5,6,7, 4,5,6,7, 5, 6,7, 7];
//(ver sp2adj).
// Incrementos no vetor xadj dão o número de entradas não nulas em cada coluna
// i.e., há 2-1=1 entrada na coluna 1
//    há 3-2=1 entrada na coluna 2
//    há 8-3=5 entradas na coluna 3
//              12-8=4                      4
//etc.
//O índice de linha dessas entradas é dado pelo vetor adjncy
// por exemplo,
// adjncy (3:7)=adjncy(xadj(3):xadj(4)-1)=[3,4,5,6,7]
// diz que as 5=xadj(4)-xadj(3) entradas na coluna 3 têm índices de linha
// 3,4,5,6,7.
//Na representação compacta, as seqüências repetidas em adjncy
//são eliminadas.
//Aqui em adjncy, as seqüências 4,5,6,7  e 7 são eliminadas.
//A estrutura padrão (xadj,adjncy) toma a forma comprimida (lindx,xlindx)
lindx=[1, 2, 3,4,5,6,7, 5, 6,7];
```

```
xlindx=[1,2,3,8,9,11];
//(colunas 4 e 7 de A são eliminadas).
//A pode ser reconstruída de (xadj,xlindx,lindx).
[xadj,adjncy,anz]= sp2adj(A);
adjncy-spcompack(xadj,xlindx,lindx)
```

# Ver Também

sp2adj, adj2sp, spget

# Name

spget — recupera entradas de matriz esparsa

```
[ij,v,mn]=spget(sp)
```

# Parâmetros

sp
: matriz esparsa de reais ou complexos

ij
: matriz de inteiros de duas colunas (índices das entradas não-nulas)

mn
: vetor de inteiros com duas entradas (dimensão de linha, dimensão de coluna)

v
: vetor coluna

# Descrição

`spget` é usado para converter a representação interna de matrizes esparsas na representação padrão `ij, v`.

Entradas não-nulas de `sp` estão localizadas em linhas e colunas com índices em `ij`.

# Exemplos

```
sp=sparse([1,2;4,5;3,10],[1,2,3])
[ij,v,mn]=spget(sp);
```

# Ver Também

sparse, sprand, speye, lufact

# Name

gmres — Generalized Minimum RESidual method

```
[x,flag,err,iter,res] = gmres(A,b,rstr,tol,maxi,M,x0)
```

# Parameters

A
: n-by-n matrix or function returning `A*x`

b
: right hand side vector

x0
: initial guess vector (default: zeros(n,1))

M
: preconditioner: matrix or function returning `M*x` (In the first case, default: eye(n,n))

rstr
: number of iterations between restarts (default: 10)

maxi
: maximum number of iterations (default: n)

tol
: error tolerance (default: 1e-6)

x
: solution vector

err
: final residual norm

iter
: number of iterations performed

flag

0 =
: `gmres` converged to the desired tolerance within `maxi` iterations

1 =
: no convergence given `maxi`

res
: residual vector

# Description

GMRES
: solves the linear system `Ax=b` using the Generalized Minimal residual method with restarts.

Details
: of this algorithm are described in :

"Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra, Eijkhout, Pozo, Romine, and Van der Vorst, SIAM Publications, 1993 (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).

"Iterative Methods for Sparse Linear Systems, Second Edition" Saad, SIAM Publications, 2003 (ftp ftp.cs.umn.edu; cd dept/users/saad/PS; get all_ps.zip).

# Examples

```
// GMRES call x=gmres(A,b);
```

# See Also

pcg , qmr

# Authors

Sage Group (IRISA, 2005)

# Name

pcg — precondioned conjugate gradient

```
[x, flag, err, iter, res] = pcg(A, b [, tol [, maxIter [, M [, M2 [, x0 [, verb
[x, flag, err, iter, res] = pcg(A, b [key=value,...])
```

# Parameters

A,
: a matrix, or a function, or a list computing `A*x` for each given `x`. The following is a description of the computation of A*x depending on the type of A.

- `matrix.` If A is a matrix, it can be dense or sparse
- `function.` If A is a function, it must have the following header :

```
function y = A ( x )
```

- `list.` If A is a list, the first element of the list is expected to be a function and the other elements in the list are the arguments of the function, from index 2 to the end. When the function is called, the current value of x is passed to the function as the first argument. The other arguments passed are the one given in the list.

b
: right hand side vector (size: nx1)

tol
: error relative tolerance (default: 1e-8). The termination criteria is based on the 2-norm of the residual r=b-Ax, divided by the 2-norm of the right hand side b.

maxIter
: maximum number of iterations (default: n)

M
: preconditioner: full or sparse matrix or function returning `M\x` (default: none)

M2
: preconditioner: full or sparse matrix or function returning `M2\x` for each `x` (default: none)

x0
: initial guess vector (default: zeros(n,1))

verbose
: set to 1 to enable verbose logging (default 0)

x
: solution vector

flag
: 0 if `pcg` converged to the desired tolerance within `maxi` iterations, 1 else

err
: final relative norm of the residual (the 2-norm of the right-hand side b is used)

iter
: number of iterations performed

res
vector of the residual relative norms

# Description

Solves the linear system `Ax=b` using the conjugate gradient method with or without preconditioning. The preconditionning should be defined by a symmetric positive definite matrix `M`, or two matrices `M1` and `M2` such that `M=M1*M2`. in the case the function solves `inv(M)*A*x = inv(M)*b` for `x`. `M`, `M1` and `M2` can be Scilab functions with calling sequence `y=Milx(x)` which computes the corresponding left division `y=Mi\x`.

The `A` matrix must be a symmetric positive definite matrix (full or sparse) or a function with calling sequence `y=Ax(x)` which computes `y=A*x`

# Example with well conditionned and ill conditionned problems

In the following example, two linear systems are solved. The first maxtrix has a condition number equals to ~0.02, which makes the algorithm converge in exactly 10 iterations. Since this is the size of the matrix, it is an expected behaviour for a gradient conjugate method. The second one has a low condition number equals to 1.d-6, which makes the algorithm converge in a larger 22 iterations. This is why the parameter maxIter is set to 30. See below for other examples of the "key=value" syntax.

```
//Well conditionned problem
A=[ 94  0   0   0    0   28  0   0   32  0
     0   59  13  5    0   0   0   10  0   0
     0   13  72  34   2   0   0   0   0   65
     0   5   34  114  0   0   0   0   0   55
     0   0   2   0    70  0   28  32  12  0
     28  0   0   0    0   87  20  0   33  0
     0   0   0   0    28  20  71  39  0   0
     0   10  0   0    32  0   39  46  8   0
     32  0   0   0    12  33  0   8   82  11
     0   0   65  55   0   0   0   0   11  100];

b=ones(10,1);
[x, fail, err, iter, res]=pcg(A,b,1d-12,15);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)

//Ill contionned one
A=[ 894     0   0     0   0   28  0   0   1000  70000
      0      5   13    5   0   0   0   0   0     0
      0      13  72    34  0   0   0   0   0     6500
      0      5   34    1   0   0   0   0   0     55
      0      0   0     0   70  0   28  32  12    0
      28     0   0     0   0   87  20  0   33    0
      0      0   0     0   28  20  71  39  0     0
      0      0   0     0   32  0   39  46  8     0
      1000   0   0     0   12  33  0   8   82    11
      70000  0   6500  55  0   0   0   0   11    100];

[x, fail, err, iter, res]=pcg(A,b,maxIter=30,tol=1d-12);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)
```

# Examples with A given as a sparse matrix, or function, or list

The following example shows that the method can handle sparse matrices as well. It also shows the case where a function, computing the right-hand side, is given to the "pcg" primitive. The final case shown by this example, is when a list is passed to the primitive.

```
//Well conditionned problem
A=[ 94  0   0   0    0   28  0   0   32  0
     0   59  13  5    0   0   0   10  0   0
     0   13  72  34   2   0   0   0   0   65
     0   5   34  114  0   0   0   0   0   55
     0   0   2   0    70  0   28  32  12  0
     28  0   0   0    0   87  20  0   33  0
     0   0   0   0    28  20  71  39  0   0
     0   10  0   0    32  0   39  46  8   0
     32  0   0   0    12  33  0   8   82  11
     0   0   65  55   0   0   0   0   11  100];
b=ones(10,1);

// Convert A into a sparse matrix
Asparse=sparse(A);
[x, fail, err, iter, res]=pcg(Asparse,b,maxIter=30,tol=1d-12);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)

// Define a function which computes the right-hand side.
function y=Atimesx(x)
  A=[ 94  0   0   0    0   28  0   0   32  0
       0   59  13  5    0   0   0   10  0   0
       0   13  72  34   2   0   0   0   0   65
       0   5   34  114  0   0   0   0   0   55
       0   0   2   0    70  0   28  32  12  0
       28  0   0   0    0   87  20  0   33  0
       0   0   0   0    28  20  71  39  0   0
       0   10  0   0    32  0   39  46  8   0
       32  0   0   0    12  33  0   8   82  11
       0   0   65  55   0   0   0   0   11  100];
  y=A*x
endfunction

// Pass the script Atimesx to the primitive
[x, fail, err, iter, res]=pcg(Atimesx,b,maxIter=30,tol=1d-12);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)

// Define a function which computes the right-hand side.
function y=Atimesxbis(x,A)
  y=A*x
endfunction

// Pass a list to the primitive
Alist = list(Atimesxbis,Asparse);
[x, fail, err, iter, res]=pcg(Alist,b,maxIter=30,tol=1d-12);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)
```

# Examples with key=value syntax

The following example shows how to pass arguments with the "key=value" syntax. This allows to set non-positionnal arguments, that is, to set arguments which are not depending on their order in the list of arguments. The available keys are the names of the optional arguments, that is : tol, maxIter, %M, %M2, x0, verbose. Notice that, in the following example, the verbose option is given before the maxIter option. Without the "key=value" syntax, the positionnal arguments would require that maxIter come first and verbose after.

```
// Example of an argument passed with key=value syntax
A=[100,1;1,10];
b=[101;11];
[xcomputed, flag, err, iter, res]=pcg(A,b,verbose=1);

// With key=value syntax, the order does not matter
[xcomputed, flag, err, iter, res]=pcg(A,b,verbose=1,maxIter=0);
```

# See Also

backslash, qmr, gmres

# Authors

Sage Group, IRISA, 2004

Serge Steer, INRIA, 2006

Michael Baudin, INRIA, 2008-2009

# References

"Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra, Eijkhout, Pozo, Romine, and Van der Vorst, SIAM Publications, 1993, ftp netlib2.cs.utk.edu/linalg/templates.ps

"Iterative Methods for Sparse Linear Systems, Second Edition", Saad, SIAM Publications, 2003, ftp ftp.cs.umn.edu/dept/users/saad/PS/all_ps.zip

# Name

qmr — quasi minimal resiqual method with preconditioning

```
[x,flag,err,iter,res] = qmr(A,b,x0,M1,M1p,M2,M2p,maxi,tol)
```

# Parameters

A
: matrix of size n-by-n or function returning `A*x`

b
: right hand side vector

x0
: initial guess vector (default: zeros(n,1))

M1
: left preconditioner: matrix or function returning `M1*x` (In the first case, default: eye(n,n))

M1p
: must only be provided when `M1` is a function. In this case `M1p` is the function which returns `M1'*x`

M2
: right preconditioner: matrix or function returning `M2*x` (In the first case, default: eye(n,n))

M2p
: must only be provided when `M2` is a function. In this case `M2p` is the function which returns `M2'*x`

maxi
: maximum number of iterations (default: n)

tol
: error tolerance (default: 1000*%eps)

x
: solution vector

flag

0 =
: `gmres` converged to the desired tolerance within `maxi` iterations

1 =
: no convergence given `maxi`

res
: residual vector

err
: final residual norm

iter
: number of iterations performed

# Description

Solves the linear system `Ax=b` using the Quasi Minimal Residual Method with preconditioning.

## See Also

gmres

## Authors

SAGE Group, IRISA 2005

# Parte XXX. Funções Especiais

# Name

besseli — funções modificadas de Bessel do primeiro tipo (I sub alfa).
besselj — funções de Bessel do primeiro tipo (J sub alpha).
besselk — funções modificadas de Bessel do segundo tipo (K sub alpha).
bessely — funções de Bessel do segundo tipo (Y sub alpha).
besselh — funções de Bessel do terceiro tipo (também conhecidas como funções de Hankel)

```
y = besseli(alpha,x [,ice])
y = besselj(alpha,x [,ice])
y = besselk(alpha,x [,ice])
y = bessely(alpha,x [,ice])
y = besselh(alpha,x)
y = besselh(alpha,K,x [,ice])
```

# Parâmetros

x
: vetor de reais ou complexos

alpha
: vetor de reais

ice
: flag (sinalizador) inteiro, com valor padrão 0

K
: inteiro, com valores possíveis 1 ou 2, a função do tipo de Hankel.

# Descrição

Aviso: a semântica destas funções varia entre o Scilab-3.0 e Scilab-3.1. A semântica antiga está disponível para compatibilidade usando as funções oldbesseli, oldbesselj, oldbesselk, oldbessely.

- `besseli(alpha,x)` computa as funções de Bessel modificadas do primeiro tipo (I sub alfa), para ordem real `alpha` e argumento `x`. `besseli(alpha,x,1)` computa `besseli(alpha,x).*exp(-abs(real(x)))`.
- `besselj(alpha,x)` computa as funções de Bessel do primeiro tipo (J sub alfa), para ordem real `alpha` e argumento `x`. `besselj(alpha,x,1)` computa `besselj(alpha,x).*exp(-abs(imag(x)))`.
- `besselk(alpha,x)` computa as funções de Bessel modificadas do segundo tipo (K sub alfa), para ordem real `alpha` e argumento `x`. `besselk(alpha,x,1)` computa `besselk(alpha,x).*exp(x)`.
- `bessely(alpha,x)` computa as funções de Bessel do segundo tipo (Y sub alfa), para ordem real `alpha` e argumento `x`. `bessely(alpha,x,1)` computa `bessely(alpha,x).*exp(-abs(imag(x)))`.
- `besselh(alpha [,K] ,x)` computa as funções de Bessel do terceiro tipo (função de Hankel H1 ou H2, dependendo do `K`), para ordem real `alpha` e argumentot `x`. Se omitido, `K` é suposto como sendo 1. `besselh(alpha,1,x,1)` computa `besselh(alpha,1,x).*exp(-%i*x)` e `besselh(alpha,2,x,1)` computa `besselh(alpha,2,x).*exp(%i*x)`

# Observações

Se `alpha` e `x` são arrays de mesmo tamanho, o resultado `y` também terá este tamanho. Se uma entrada é um escalar, ela é expandida para o tamanho da outra entrada. Se uma entrada é um vetor linha e a outra é um vetor coluna, o resultado `y` é um table 2-dimensional ("tabela") de valores de funções.

As funções de Bessel Y_alfa e J_alfa são duas soluções independentes da equação diferencial de Bessel:

```
2 2 2 x y" + x y' + (x - alfa ) y = 0 , alfa >= 0
```

As funções modificadas de Bessel K_alfa e I_alfa são duas soluções independentes para a equação diferencial de Bessel :

```
2 2 2 x y" + x y' - (x + alfa ) y = 0 , alfa >= 0
```

As funções de Hankel de primeiro e segundo tipos H^1_alfa e H^2_alfa, são combinações lineares das funções de Bessel de primeiro e segundo tipos:

```
H^1_alfa(z) = J_alfa(z) + i Y_alfa(z) H^2_alfa(z) = J_alfa(z) - i Y_alfa(z)
```

# Exemplos

```
//  Funções I de Bessel
// ==================
x = linspace(0.01,10,5000)';
xbasc()
subplot(2,1,1)
plot2d(x,besseli(0:4,x), style=2:6)
legend('I'+string(0:4),2);
xtitle("Algumas funções modificadas de Bessel do primeiro tipo")
subplot(2,1,2)
plot2d(x,besseli(0:4,x,1), style=2:6)
legend('I'+string(0:4),1);
xtitle("Algumas funções modificadas de Bessel do primeiro tipo escaladas")

// Funções J de Bessel
// =================
x = linspace(0,40,5000)';
xbasc()
plot2d(x,besselj(0:4,x), style=2:6, leg="J0@J1@J2@J3@J4")
legend('I'+string(0:4),1);
xtitle("Algumas funções de Bessel do primeiro tipo")

// Usando o fato de que J_(1/2)(x) = sqrt(2/(x pi)) sin(x)
// Para comparar o algoritmo de besselj(0.5,x) com uma fórmula mais direta
x = linspace(0.1,40,5000)';
y1 = besselj(0.5, x);
y2 = sqrt(2 ./(%pi*x)).*sin(x);
er = abs((y1-y2)./y2);
ind = find(er > 0 & y2 ~= 0);
xbasc()
subplot(2,1,1)
plot2d(x,y1,style=2)
xtitle("besselj(0.5,x)")
subplot(2,1,2)
plot2d(x(ind), er(ind), style=2, logflag="nl")
xtitle("Erro relativo entre as duas fórmulas para besselj(0.5,x)")
```

```
// Funções K de Bessel
// =================
x = linspace(0.01,10,5000)';
xbasc()
subplot(2,1,1)
plot2d(x,besselk(0:4,x), style=0:4, rect=[0,0,6,10])
legend('K'+string(0:4),1);
xtitle("Algumas funções modificadas de Bessel do segundo tipo")
subplot(2,1,2)
plot2d(x,besselk(0:4,x,1), style=0:4, rect=[0,0,6,10])
legend('K'+string(0:4),1);
xtitle("Algumas funções modificadas de Bessel do segundo tipo escaladas")

// Funções Y de Bessel
// =================
x = linspace(0.1,40,5000)'; // funções Y de Bessel não possuem limite para x ->
xbasc()
plot2d(x,bessely(0:4,x), style=0:4, rect=[0,-1.5,40,0.6])
legend('Y'+string(0:4),4);
xtitle("Algumas funções de Bessel do segundo tipo")

// Funções H de Bessel
// =================
x=-4:0.025:2; y=-1.5:0.025:1.5;
[X,Y] = ndgrid(x,y);
H = besselh(0,1,X+%i*Y);
clf();f=gcf();
xset("fpf"," ")
f.color_map=jetcolormap(16);
contour2d(x,y,abs(H),0.2:0.2:3.2,strf="034",rect=[-4,-1.5,3,1.5])
legends(string(0.2:0.2:3.2),1:16,"ur")
xtitle("Curvas de nível de |H1(0,z)|")
```

# Autores

Amos, D. E., (SNLA)
Daniel, S. L., (SNLA)
Weston, M. K., (SNLA)

# Função Usada

Os códigos-fontes podem ser achados em routines/calelm

Slatec : dbesi.f, zbesi.f, dbesj.f, zbesj.f, dbesk.f, zbesk.f, dbesy.f, zbesy.f, zbesh.f

Drivers para estender a área de definição (Serge Steer INRIA): dbesig.f, zbesig.f, dbesjg.f, zbesjg.f, dbeskg.f, zbeskg.f, dbesyg.f, zbesyg.f, zbeshg.f

# Name

beta — função beta

```
z = beta(x,y)
```

# Parâmetros

x, y
: dois reais positivos ou duas matrizes (ou vetores) de reais positivos de mesmo tamanho

z
: um real ou uma matriz de reais com mesmo tamanho que `x` com `z(i,j) = beta(x(i,j),y(i,j))`.

# Descrição

Computa a função beta completa :

```
                /1
                |   x-1     y-1        gamma(x) gamma(y)
  beta(x,y) =   |  t   (1-t)    dt  =  ----------------
                /0                          gamma(x+y)
```

Para `x` e `y` pequenos, o algoritmo usa a expressão em função da função gama, de outro modo, ele aplica a função exponencial no resutado da função `betaln` function fornecido no DCDFLIB: Biblioteca de Rotinas FORTRAN para Funções, Inversas e Outros Parâmetros de Distribuição Cumulativa (ver cdfbet para maiores informações sobre DCDFLIB).

# Exemplos

```
// exemplo 1 :
beta(5,2) - beta(2,5)   // simetria (deve ser exatamente 0)
beta(0.5,0.5)           // o valor exato é pi

// exemplo 2 : um estudo de erros baseado na relação B(1,x) = 1/x
// (a computação de 1/x deve levar apenas a um erro relativo de eps_m, então
//  pode ser usada como referência para avaliar o erro em B(1,x))
x = logspace(-8,8,20000)';
e = beta(ones(x),x) - (1)./x;
er = abs(e) .* x;
ind = find(er ~= 0);
eps = ones(x(ind))*number_properties("eps");
xbasc()
plot2d(x(ind),[er(ind) eps 2*eps],style=[1 2 3],logflag="ll",leg="er@eps_m@2 eps
xtitle("erro relativo aproximado na computação de beta(1,x)")
xselect()

// exemplo 3 : plotando a função beta
t = linspace(0.2,10,60);
X = t'*ones(t); Y = ones(t')*t;
Z = beta(X,Y);
xbasc()
plot3d(t, t, Z, flag=[2 4 4], leg="x@y@z", alpha=75, theta=30)
```

```
xtitle("A função beta em [0.2,10]x[0.2,10]")
xselect()
```

# Ver Também

gamma, cdfbet

# Name

calerf — computa funções de erro

# Parâmetros

x
: matriz ou vetor de reais

flag
: indicador inteiro

y
: matriz ou vetor de reais (de mesmo tamanho que x)

# Descrição

`calerf(x,0)` computa a função de erro `erf(x)`

`calerf(x,1)` computa a função de erro complementar `erfc(x)`

`calerf(x,2)` computa a função de erro complementar escalada `erfcx(x)`

# Exemplos

```
deff('y=f(t)','y=exp(-t^2)');
calerf(1,0)
2/sqrt(%pi)*intg(0,1,f)
```

# Ver Também

erf, erfc, erfcx

# Autor

W. J. Cody (código de Netlib (specfun))

# Name

dlgamma — derivada da função gammaln, função psi

```
y = dlgamma(x)
```

# Parâmetros

x
: vetor de reais

y
: vetor de reais com o mesmo tamanho

# Descrição

`dlgamma(x)` avalia, em todos os elementos de `x` a derivada logarítmica da função gama (gamma), que corresponde também à derivada da função ln(gama) (gammaln):

```
d/dx (gamma(x)) / gamma(x) = d/dx (ln gamma(x))
```

`x` deve ser real. Também é conhecida como a função psi.

# Exemplos

```
dlgamma(0.5)
```

# Ver Também

gamma, gammaln

# Autor

W. J. Cody (code from Netlib (specfun))

# Name

erf — função de erro

```
y = erf(x)
```

# Parâmetros

x
: vetor ou matriz de reais

y
: vetor ou matriz de reais (de mesmo tamanho que x)

# Descrição

`erf` computa a função de erro:

```
                    /x     2
          2         |    -t
    y = -------     |   e    dt
        sqrt(pi)  /0
```

# Exemplos

```
deff('y=f(t)','y=exp(-t^2)');
erf(0.5)-2/sqrt(%pi)*intg(0,0.5,f)
```

# Ver Também

erfc, erfcx, calerf, cdfnor

# Autor

W. J. Cody (código de Netlib (specfun))

# Name

erfc — função de erro complementar

```
y = erfc(x)
```

# Parâmetros

x
: real vector or matrix

y
: real vector or matrix (of same size than x)

# Descrição

`erfc` computa a função de erro complementar:

```
                  /+oo    2
          2       |     -t
    y = -------   |    e    dt
        sqrt(pi)  |
                  /x

    y = 1 - erf(x)
```

# Exemplos

```
erf([0.5,0.2])+erfc([0.5,0.2])
```

# Ver Também

erf, erfcx, calerf

# Autor

W. J. Cody (código de Netlib (specfun))

# Name

erfcx — função de erro complementar escalada

```
y = erfcx(x)
```

# Parameters

x
: vetor ou matriz de reais

y
: vetor ou matriz de reais (de mesmo tamanho que x)

# Descrição

`erfcx` computa a função de erro complementar escalada:

```
             2
    y = exp(x ) * erfc(x)

               1
    y  -->  ---------    quando x --> +oo
            x sqrt(pi)
```

# Ver Também

erf, erfc, calerf

# Autor

W. J. Cody (código de Netlib (specfun))

# Name

erfinv — função inversa à função de erro

```
y = erfinv(x)
```

# Parâmetros

x
: vetor ou matriz de reais

y
: vetor ou matriz de reais (de mesmo tamanho que x)

# Descrição

`erfinv` computa a função inversa à função de erro `erf`. `x = erfinv(y)` satisfaz `y = erf(x)`, -1 <= `y` < 1, -inf <= `x` <= inf.

# Exemplos

```
x=linspace(-0.99,0.99,100);
y=erfinv(x);
plot2d(x,y)
norm(x-erf(y),'inf')
```

# Ver Também

erfc, cdfnor

# Referências

Milton Abramowitz e Irene A. Stegun, eds. Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. New York: Dover, 1972.

# Name

gamma — função gama

```
y = gamma(x)
```

# Parâmetros

x
: vetor ou matriz de reais ou complexos

y
: vetor ou matriz de reais ou complexos de mesmo tamanho que x

# Descrição

`gamma(x)` avalia a função gama em todos os elementos de `x`. A função gama é defininda por :

```
            /+oo
            |   x-1  -t
gamma(x) =  |  t    e    dt
            /0
```

e generaliza a função fatorial para os números reais (`gamma(n+1) = n!`).

# Exemplos

```
// exemplos simples
gamma(0.5)
gamma(6)-prod(1:5)

// o gráfico da função gama em [a,b]
a = -3; b = 5;
x = linspace(a,b,40000)';
y = gamma(x);
xbasc()
c=xget("color")
xset("color",2)
plot2d(x, y, style=0, axesflag=5, rect=[a, -10, b, 10])
xset("color",c)
xtitle("A função gama em ["+string(a)+","+string(b)+"]")
xselect()
```

# Ver Também

gammaln, dlgamma

# Autores

W. J. Cody e L. Stoltz (código de Netlib (specfun))

# Name

gammaln — o logaritmo (natural) da função gama

```
y = gammaln(x)
```

# Parameters

x
: vetor de reais

y
: vetor de reais com o mesmo tamanho

# Description

`gammaln(x)` avalia o logaritmo (natural) da função gama em todos os elementos de `x`, evitando underflow e overflow. `x` deve ser de reais.

# Exemplos

```
gammaln(0.5)
```

# Ver Também

gamma, dlgamma

# Autores

W. J. Cody e L. Stoltz (código de Netlib (specfun))

# Name

legendre — funções associadas de Legendre

```
y = legendre(n,m,x [,normflag])
```

# Parâmetros

n
: inteiro não-negativo ou vetor de inteiros não-negativos igualmente espaçados com incremento igual a 1

m
: inteiro não-negativo ou vetor de inteiros não-negativos igualmente espaçados com incremento igual a 1

x
: vetor (linha) de reais (os elementos de `x` devem estar no intervalo `(-1,1)` )

normflag
: (opcional) escalar string

# Descrição

Quando `n` e `m` são escalares, `legendre(n,m,x)` avalia a função de Legendre associada Pnm(x) em todos os elementos de `x`. A definição usada é :

```
                                m
             m         m/2   d
Pnm(x) = (-1)  (1-x^2)     --- Pn(x)
                                m
                               dx
```

onde `Pn` é o polinômio de Legendre de grau `n`. Então, `legendre(n,0,x)` avalia o polinômio de Legendre Pn(x) em todos os elementos de `x`.

Quando normflag é igual a "norm" o resultado é uma versão normalizada (sem o fator `(-1)^m` ), precisamente :

```
                   _____________                   m
                  /(2n+1)(n-m)!          m/2    d
Pnm(x,"norm") =  /--------------  (1-x^2)     --- Pn(x)
               \/     2  (n+m)!                  m
                                               dx
```

que é útil para computar funções harmônicas esféricas (ver exemplo 3):

Por eficiência, um dos primeiros dois argumentos pode ser um vetor, por exemplo `legendre(n1:n2,0,x)` avalia todos os polinômios de Legendre de graus *n1*, *n1+1*, ..., *n2* nos elementos de `x` e `legendre(n,m1:m2,x)` avalia todas as funções de Legendre associadas Pnm para *m=m1*, *m1+1*, ..., *m2* em `x`.

# Formato de Saída

Em qualquer caso, o formato é `y` :

```
max(length(n),length(m)) x length(x)
```

e :

```
y(i,j) = P(n(i),m;x(j))   se n é um vetor
y(i,j) = P(n,m(i);x(j))   se m é um vetor
y(1,j) = P(n,m;x(j))      se ambos n e m são escalares
```

de tal modo que `x` é preferivelmente um vetor linha, mas qualquer matriz `mx x nx` é excetuada e considerada como uma matriz `1 x (mx * nx)` matrix, reformada segundo a ordem das colunas.

# Exemplos

```
// exemplo 1 : plot dos primeiros 6 polinômios de Legendre em (-1,1)
l = nearfloat("pred",1);
x = linspace(-l,l,200)';
y = legendre(0:5, 0,  x);
xbasc()
plot2d(x,y', leg="p0@p1@p2@p3@p4@p5@p6")
xtitle("Os 6 primeiros polinômios de Legendre")

// exemplo 2 : plot das funções associadas de Legendre de grau 5
l = nearfloat("pred",1);
x = linspace(-l,l,200)';
y = legendre(5, 0:5, x, "norm");
xbasc()
plot2d(x,y', leg="p5,0@p5,1@p5,2@p5,3@p5,4@p5,5")
xtitle("As funções associadas de Legendre (normalizadas) de grau 5")

// exemplo 3 : definindo e plotando uma função harmônica esférica
// 3-1 : definindo a função Ylm
function [y] = Y(l,m,theta,phi)
  // teta pode ser um escalar ou um vetor linha
  // phi pode ser um escalar ou um vetor coluna
  if m >= 0 then
     y = (-1)^m/(sqrt(2*%pi))*exp(%i*m*phi)*legendre(l, m, cos(theta), "norm")
  else
     y = 1/(sqrt(2*%pi))*exp(%i*m*phi)*legendre(l, -m, cos(theta), "norm")
  end
endfunction

// 3.2 : definindo outra função útil
function [x,y,z] = sph2cart(theta,phi,r)
  // vetor linha teta      1 x nt
  // vetor coluna phi  np x 1
  // r     escalar ou matriz np x nt (r(i,j) o comprimento em phi(i) theta(j))
  x = r.*(cos(phi)*sin(theta));
  y = r.*(sin(phi)*sin(theta));
  z = r.*(ones(phi)*cos(theta));
endfunction

// 3-3 plot de Y31(theta,phi)
l = 3; m = 1;
```

```
theta = linspace(0.1,%pi-0.1,60);
phi = linspace(0,2*%pi,120)';
f = Y(l,m,theta,phi);
[x1,y1,z1] = sph2cart(theta,phi,abs(f));       [xf1,yf1,zf1] = nf3d(x1,y1,z1);
[x2,y2,z2] = sph2cart(theta,phi,abs(real(f))); [xf2,yf2,zf2] = nf3d(x2,y2,z2);
[x3,y3,z3] = sph2cart(theta,phi,abs(imag(f))); [xf3,yf3,zf3] = nf3d(x3,y3,z3);

xbasc()
subplot(1,3,1)
plot3d(xf1,yf1,zf1,flag=[2 4 4]); xtitle("|Y31(theta,phi)|")
subplot(1,3,2)
plot3d(xf2,yf2,zf2,flag=[2 4 4]); xtitle("|Real(Y31(theta,phi))|")
subplot(1,3,3)
plot3d(xf3,yf3,zf3,flag=[2 4 4]); xtitle("|Imag(Y31(theta,phi))|")
```

# Autores

Smith, John M. (código dxlegf.f de Slatec)
B. Pincon (interface Scilab)

# Name

oldbesseli — funções de Bessel modificadas do primeiro tipo (I sub alfa).
oldbesselj — funções de Bessel do primeiro tipo (J sub alfa).
oldbesselk — funções de Bessel modificadas do segundo tipo (K sub alfa).
oldbessely — funções de Bessel do segundo tipo (Y sub alfa).

```
y = oldbesseli(alpha,x)
y = oldbesseli(alpha,x,ice)
y = oldbesselj(alpha,x)
y = oldbesselk(alpha,x)
y = oldbesselk(alpha,x,ice)
y = oldbessely(alpha,x)
```

# Parâmetros

x
: vetor de reais com entradas não-negativas

alpha
: vetor de reais com entradas não-negativas igualmente espaçadas com incremento igual a 1`alpha=alpha0+(n1:n2)`

ice
: flag inteiro com valor padrão 1

# Descrição

Funções obsoletas, use besseli, besselj, besselk, bessely no lugar. Note, contudo, que a semântica destes dois grupos de funções é diferente.

`oldbesseli(alpha,x)` computa funções modificadas de Bessel do primeiro tipo (I sub alfa), para ordem real não-negativa `alpha` e argumento real não-negativo `x`. `besseli(alpha,x,2)` computa `besseli(alpha,x).*exp(-x)`.

`oldbesselj(alpha,x)` computa funções de Bessel do primeiro tipo (J sub alfa), para ordem real não-negativa `alpha` e argumento real não-negativo `x`.

`oldbesselk(alpha,x)` computa as funções modificadas de Bessel do segundo tipo (K sub alfa), para ordem real não-negativa `alpha` e argumento real não-negativo `x`. `besselk(alpha,x,2)` computa `besselk(alpha,x).*exp(x)`.

`oldbessely(alpha,x)` computa funções de Bessel do segundo tipo (Y sub alfa), para ordem real não-negativa `alpha` e argumento real não-negativo `x`.

`alpha` e `x` podem ser vetores. A saída é `m`-por-`n` com `m = size(x,'*')`, `n = size(alpha,'*')` whose `(i,j)` cuja entrada `oldbessel?(alpha(j),x(i))`.

# Observações

As funções Y_alfa e J_alfa de Bessel são duas soluções independentes da equação diferencial de Bessel :

```
2 2 2 x y" + x y' + (x - alfa ) y = 0 , alfa = 0
```

As funções modificadas K_alfa e I_alfa de Bessel são duas soluções independentes da equação diferencial modificada de Bessel:

```
2 2 2 x y" + x y' - (x + alfa ) y = 0 , alfa >= 0
```

# Exemplos

```
// exemplo #1: exibindo algumas funções I de Bessel
x = linspace(0.01,10,5000)';
y = oldbesseli(0:4,x);
ys = oldbesseli(0:4,x,2);
xbasc()
subplot(2,1,1)
plot2d(x,y, style=2:6, leg="I0@I1@I2@I3@I4", rect=[0,0,6,10])
xtitle("Algumas funções modificadas de Bessel do primeiro tipo")
subplot(2,1,2)
plot2d(x,ys, style=2:6, leg="I0s@I1s@I2s@I3s@I4s", rect=[0,0,6,1])
xtitle("Algumas funções modificadas de Bessel do primeiro tipo escaladas")

// exemplo #2 : exibindo algumas funções J de Bessel
x = linspace(0,40,5000)';
y = besselj(0:4,x);
xbasc()
plot2d(x,y, style=2:6, leg="J0@J1@J2@J3@J4")
xtitle("Algumas funções de Bessel do primeiro tipo")

// example #3 : usando o fato de que J_(1/2)(x) = sqrt(2/(x pi)) sin(x)
//              para comparar o algoritmo de besselj(0.5,x) com
//              uma fórmula mais direta
x = linspace(0.1,40,5000)';
y1 = besselj(0.5, x);
y2 = sqrt(2 ./(%pi*x)).*sin(x);
er = abs((y1-y2)./y2);
ind = find(er > 0 & y2 ~= 0);
xbasc()
subplot(2,1,1)
plot2d(x,y1,style=2)
xtitle("besselj(0.5,x)")
subplot(2,1,2)
plot2d(x(ind), er(ind), style=2, logflag="nl")
xtitle("erro relativo entre as duas fórmulas para besselj(0.5,x)")

// exemplo #4: exibindo algumas funções K de Bessel
x = linspace(0.01,10,5000)';
y = besselk(0:4,x);
ys = besselk(0:4,x,1);
xbasc()
subplot(2,1,1)
plot2d(x,y, style=0:4, leg="K0@K1@K2@K3@K4", rect=[0,0,6,10])
xtitle("Algumas funções modificadas de Bessel do segundo tipo")
subplot(2,1,2)
plot2d(x,ys, style=0:4, leg="K0s@K1s@K2s@K3s@K4s", rect=[0,0,6,10])
xtitle("Algumas funções modificadas de Bessel do segundo tipo escaladas")

// exemplo #5: plot de várias funções Y de Bessel
x = linspace(0.1,40,5000)'; // funções  Y de Bessel não possuem limite para x -
y = bessely(0:4,x);
xbasc()
```

```
plot2d(x,y, style=0:4, leg="Y0@Y1@Y2@Y3@Y4", rect=[0,-1.5,40,0.6])
xtitle("Algumas funções de Bessel do segundo tipo")
```

# Autores

W. J. Cody, L. Stoltz (código de Netlib (specfun))

# Parte XXXI. Cadeias de Caracteres (Strings)

# Name

ascii — conversão ASCII de strings

```
a=ascii(txt)
txt=ascii(a)
```

# Parâmetros

txt
: um string ou matriz de strings

a
: um vetor de códigos ASCII inteiros

# Descrição

Esta função converte um string Scilab para um vetor de códigos ASCII, ou um vetor de códigos ASCII em strings Scilab

Se `txt` é uma matriz de strings, `ascii(txt)` é equivalente a `ascii(strcat(txt))`

# Exemplos

```
ascii(["olá";"mundo"])
ascii("scilab")
ascii([115 99 105 108 97 98])
```

# Ver Também

code2str, str2code

# Name

blanks — cria strings de caracteres em branco

```
txt=blanks(n)
```

# Parâmetros

txt
: um único string

n
: número de caracteres em branco

# Descrição

blanks(n) é um string de n caracteres em branco.

# Exemplos

```
disp(['xxx' blanks(20) 'yyy'])
```

# Name

code2str — retorna o string associado ao código Scilab inteiro

```
str=code2str(c)
```

# Parâmetros

str
: string

c
: vetor de códigos de caracteres inteiros

# Descrição

Retorna o string associado ao código Scilab inteiro. str é tal que c(i) é o código Scilab inteiro de part(str,i))

# Exemplos

```
code2str([-28 12 18 21 10 11])
str2code('Scilab')'
```

# Ver também

str2code, ascii

# Name

convstr — conversão maiúsculas-minúsculas, minúsculas-maiúsculas

```
[y]=convstr(str, [flag])
```

# Parâmetros

str, y
: uma matriz de strings

flag
: um caractere de opção com dois possíveis valores

  'u'
  : para maiúsculas

  'l'
  : para minúsculas

# Descrição

converts a matriz de strings `str-matrix` para caracteres minúsculos (`"l"` ; valor padrão) ou para maiúsculos (`"u"`).

# Exemplos

```
A=['esta','é';'a minha','matriz'];
convstr(A,'u')
```

# Name

emptystr — string de comprimento zero

```
s=emptystr()
s=emptystr(a)
s=emptystr(m,n)
```

# Parâmetros

a
: qualquer tipo de matriz

s
: matriz de strings

m,n
: inteiros

# Descrição

Retorna uma matriz de strings de comprimento zero.

Sem argumentos de entrada, retorna um string de comprimento zero.

Com uma matriz como argumento de entrada, retorna uma matriz de strings de comprimento zero de mesmo tamanho.

Com dois argumentos inteiros, retorna uma matriz mxn de strings de comprimento zero.

# Exemplos

```
x=emptystr();for k=1:10, x=x+','+string(k);end
```

# Ver Também

part, length, string

# Name

grep — acha correspondências de um string em um vetor de strings

```
row=grep(haystack,needle )
[row,which]=grep(haystack,needle )
row=grep(haystack,needle ,[flag])
[row,which]=grep(haystack,needle ,[flag])
```

# Parâmetros

haystack
: vetor linha de strings

needle
: string ou vetor linha de strings . O(s) string(s) a serem procurados em `haystack.`

row
: vetor de índices: linha onde uma correspondência foi encontrada, ou matriz vazia se nenhuma ocorrência tiver sido encontrada

which
: vetor de índices: índice do string needle encontrado, ou uma matriz vazia, se nenhuma correspondência tiver sido encontrada

flag
: caractere ("r" para expressão regular)

# Descrição

Para cada entradada de haystack , grep procura se pelo menos um string em needle corresponde a um substring. Os índices das entradas de haystack onde pelo menos uma entrada foi encontrada são retornados no argumento row. O argumento opcional which fornece o índice do primeiro string de needle encontrado. Quando se usa o terceiro parâmetro "r", needle deve ser substituído por uma expressão regular. Então, grep vai corresponder a haystack de acordo com as regras regulares expressas.

# Exemplos

```
txt=['acha correspondência em um string ou em um vetor de strings'
     'procura posição de um string em outro string'
     'Compara Strings'];

grep(txt,'strings')
grep(txt,['strings' 'Strings'])

[r,w]=grep(txt,['strings' 'Strings'])

str = ["hat";"cat";"hhat";"chat";"hcat";"ccchat";"at";"dog"]

grep(str,'/[hc]+at/','r')
grep(str,'/[hc]?at/','r')
grep(str,'/cat|dog/','r')
```

# Ver Também

strindex

# Name

isalphanum — verifica se os caracteres de um string são alfanuméricos

```
res = isalphanum(str)
```

# Parâmetros

str
: string

res
: matriz de valores booleanos

# Descrição

`res = isalphanum(str)` retorna um array de mesmo tamanho que str contendo valores lógicos %t (true) onde os elementos de str são alfanuméricos e %f (false) onde eles não são.

# Exemplos

```
s = 'A1,B2,C3';
isalphanum(s)
```

# Ver Também

isletter, isdigit

# Name

isascii — verifica se um caractere é do tipo 7-bit US-ASCII

```
res = isascii(str)
```

# Parâmetros

str
: string

res
: uma matriz de booleanos

# Descrição

`res = isascii(str)` retorna verdadeiro (%T) se c é um código 7-bit US-ASCII entre 0 e 0177 octal, inclusive. Caso contrário, retorna falso (%F).

# Exemplos

```
isascii(code2str(300))
isascii(code2str(-11))
letters = [115.    99.    105.    108.    97.    98.]
isascii(letters)
ascii(letters)
isascii('scilab')
```

# Ver Também

isalphanum

# Name

isdigit — checa se os caracteres de um string são números entre 0 e 9

```
res = isdigit(str)
```

# Parâmetros

str
: string

res
: matriz de valores booleanos

# Descrição

`res = isdigit(str)` retorna um array com mesmo tamanho que str com valores %T (verdadeiro) onde os elementos de str são dígitos de 0 a 9 e %F (falso) onde eles não são.

# Exemplos

```
s = 'A1,B2,C3';
isdigit(s)
```

# Ver Também

isalphanum, isletter

# Name

isletter — verifica se os caracteres de um string são letras do alfabeto

```
res = isletter(str)
```

# Parâmetros

str
: string

res
: matriz de booleanos

# Descrição

`res = isletter(str)` retorna um array de mesmo tamanho que str com %t (verdadeiro) onde os elementos de str são letras do alfabeto e %f (false) onde eles não são.

# Exemplos

```
s = 'A1,B2,C3';
isletter(s)
```

# Ver Também

isalphanum, isdigit

# Name

isnum — testa se um string representa um número

```
res = isnum(str)
```

# Parâmetros

str
: um string ou matriz de strings

res
: matriz de booleanos

# Descrição

`res = isnum(str)`retorna %T se str representa um número

# Exemplos

```
isnum(['1'     , ..
       '-1.23' , ..
       '+1e+23', ..
       '1d+23' , ..
       '%pi'])
```

# Ver Também

isletter, isdigit, isalphanum

# Autores

P.M

# Name

justify — justifica um array de caracteres

```
Tj=justify(T,opt)
```

# Parâmetros

T
: matriz de strings

Tj
: matriz de strings. O resultado justificado

opt
: opção com possíveis valores

  'r'
  : ou 'right' para justificação direita

  'l'
  : ou 'left' para justificação esquerda

  'c'
  : ou 'center' justificação centrada

# Descrição

`justify` justifica a coluna de uma matriz de strings de acordo com a opção dada.

# Exemplos

```
t=['1234','x','adfdfgdfghfgj'
   '1','354556','dgf'
   'sdfgd','','sdfsf'];

justify(t,'l')
justify(t,'c')
justify(t,'r')
```

# Ver Também

length, part

# Name

length — comprimento de um objeto

```
n=length(M)
```

# Parâmetros

M
: matriz (usual ou de polinômios ou de strings) ou lista

n
: inteiro ou matriz de inteiros

# Descrição

Para uma matriz usual ou de polinômios, `n` é o inteiro igual ao número de linhas vezes o número de colunas de `M`. (Também válido para `M` uma matriz de booleanos)

Para matrizes de strings (e, em particular, para um string) `length` retorna em `n` os comprimentos das entradas da matriz de strings `M`.

O comprimento de uma lista é o número de elementos da lista (também dado por `size`).

`length('123')` é `3`. `length([1,2;3,4])` is `4`.

AVISO : length para matrizes esparsas retorna o máximo das dimensões, não o produto das dimensões. (exemplo : length(sparse(eye(12,2))) retorna max(12,2), não 24)

Utilize size(...,'*') para matrizes esparsas.

# Exemplos

```
length([123 ; 456 ])
length(['olá mundo',SCI])
```

# Ver Também

size

# Name

part — extração de strings

```
[strings_out] = part(strings_in, v)
```

# Parâmetros

strings_in, strings_out
: matriz de strings

v
: vetor linha de inteiros

# Descrição

Seja `s[k]` o caractere `k` do string `s` (ou o espaço em branco se `k >length(s)`).

`part` retorna `strings_out`, uma matriz de strings, tal que `strings_out(i,j)` é o string `"s[v(1)]...s[v(n)]"` ( `s=strings_in(i,j)` ).

# Exemplos

```
// retorna caracteres da posição 8 a 11
part("Como usar ""part"" ?",8:11)

// retorna caracteres da posição 2 a 4 para cada elemento
// caractere inexistente substituído por ''
c = part(['a','abc','abcd'],2:4)

// retorna o caractere da posição 1 para cada elemento e adiciona caracteres da
c = part(['abcdefg','hijklmn','opqrstu'],[1,4:7]);

// retorna o caractere 4 para cada elemento, adiciona caracteres da posição 1 a
c = part(['abcdefg','hijklmn','opqrstu'],[4,1:7,4]);

// retorna o caractere da posição 1, adiciona de novo o caractere da posição 1 
c=part(['a','abc','abcd'],[1,1,2])

// a a a
part(['a','abc','abcd'],[1])

// aa aa aa
part(['a','abc','abcd'],[1,1])

// aa aab aab
part(['a','abc','abcd'],[1,1,2])
```

# Ver Também

string, length

# Name

regexp — acha um string que corresponde ao string de expressão regular

```
[start]=regexp(input,pattern,[flag])
[start,end,match]=regexp(input,pattern,[flag])
[start,end]=regexp(input,pattern,[flag])
[start,end,match]=regexp(input,pattern,[flag])
```

# Parâmetros

input
: string

pattern
: string (sob regras de expressão regular)

start
: o índice de início de cada substring de str que corresponde ao padrão do string de expressão regular

end
: o índice de fim de cada substring de str que corresponde ao padrão do string de expressão regular

match
: o texto de cada substring de que corresponde a `pattern`.

[flag]
: 'o' para correspondência com padrão uma vez

# Descrição

As regras de expressão regular são similares às da linguagem Perl. Para uma introdução rápido , veja http://perldoc.perl.org/perlrequick.html. Para um tutorial mais profundo, veja http://perldoc.perl.org/perlretut.html e para página de referência, veja http://perldoc.perl.org/perlre.html

Uma diferença para Perl é que correspondência entre posições, mas não entre caracteres (por exemplo, com /^/ ou /(?=o)/) é uma correspondência válida em Perl, mas não em Scilab.

# Exemplos

```
regexp('xabyabbbz','/ab*/','o')
regexp('a!','/((((((((((a))))))))))\041/')
regexp('ABCC','/^abc$/i')
regexp('ABC','/ab|cd/i')
[a b c]=regexp('XABYABBBZ','/ab*/i')
```

# Ver Também

strindex

# Name

sci2exp — converte uma expressão para um string

```
t=sci2exp(a [,nam] [,lmax])
```

# Parâmetros

a
: expressão Scilab. Pode ser:

- constante,
- polinomial
- matriz de strings
- lista
- matriz de booleanos

nam
: string

t
: vetor de strings, contém a expressão ou a instrução de simulação.

lmax
: inteiro, contém o comprimento de linha máximo. O valor padrão é 90, `lmax=0` indica ausência de controle de comprimento de linha, um único string é retornado

# Descrição

`sci2exp` converte expressão para um string de instrução se `nam` for fornecido, ou para um string de expressão.

# Exemplos

```
a=[1 2;3 4]
sci2exp(a,'aa')
sci2exp(a,'aa',0)
sci2exp(ssrand(2,2,2))
sci2exp(poly([1 0 3 4],'s'),'fi')
```

# Name

str2code — retorna códigos scilab inteiros associados aos caracteres do string

```
c=str2code(str)
```

# Parâmetros

str
: um string

c
: um vetor de códigos de caracteres inteiros

# Descrição

Retorna c tal que c(i) é o código scilab inteiro de part(str,i))

# Exemplos

```
str2code('Scilab')'
code2str([-28 12 18 21 10 11])
```

# Ver Também

code2str, ascii

# Name

strcat — concatena strings

```
txt=strcat(vector_of_strings [,string_added])
txt=strcat(vector_of_strings [,string_added],["flag"])
```

# Parâmetros

vector_of_strings
: vetor de strings

string_added
: string adicionado, o valor padrão é emptystr `""`

txt
: string

"flag"
: string ( "r" para retornar uma matriz coluna, "c" para retornar uma matriz linha)

# Descrição

`txt=strcat(vector_of_strings)` concatena strings : `txt=vector_of_strings(1)+...+vector_of_strings(n)`

`txt=strcat(vector_of_strings,string_added)` retorna `txt=vector_of_strings(1)+string_added+...+string_added +vector_of_strings(n)`.

O símbolo de mais faz o mesmo: `"a"+"b"` é o mesmo que `strcat(["a","b"])`.

`Se o tamanho de vector_of_strings for um, retorna txt=vector_of_strings(1);`

`strcat('A','B') retorna 'A' , não 'AB' como strcat(['A','B'])`

# Exemplos

```
strcat(string(1:10),',')
strcat(["a","b"])
strcat(["a","b"],'|')
strcat('A')
strcat('A','B')
strcat(['A','B'])
strcat(['A','B'],'')
```

# Ver Também

string, strings

# Name

strchr — acha a primeira ocorrência de um caractere em um string

```
res = strchr(haystack,char)
```

# Parâmetros

haystack
: string ou matriz de strings

char
: caractere

res
: string ou matriz de strings

# Descrição

`res = strchr(haystack,char)` retorna a primeira ocorrência do caractere no string str.

num deve ter as mesmas dimensões que haystack, ou apenas um caractere.

# Exemplos

```
strchr('Este é um string de amostra','s')
strchr(['Este é um string de amostra','no scilab'],'s')
strchr(['Este é um string de amostra','no scilab'],['s','a'])
```

# Ver Também

strrchr, strstr

# Name

strcmp — compara strings

```
res = strcmp(string_one,string_two,[,'i'])
```

# Parâmetros

string_one
: string ou matriz de strings

string_two
: string ou matriz de strings

'i'
: parâmetro para realização de stricmp (caso independente), o valor padrão é 's'

res
: matrix

# Descrição

`res = strcmp(string_one,string_two)` retorna um valor inteiro indicando a relação entre os strings.

Um valor maior que zero indica que o primeiro caractere não correspondente possui valor maior em `string_one` que em `string_two`

Um valor abaixo de zero indica o contrário.

# Exemplos

```
TXT1 = ['scilab','SciLab';'Strcmp','STRcmp'];
TXT2 = ['ScIlAb','sciLab';'sTrCmP','StrCMP'];
strcmp(TXT1,TXT2)
strcmp(TXT1,'scilab')
strcmp(TXT1,'SciLab')
strcmp(TXT1,TXT2,'i')
```

# Ver Também

strcat, strcmpi

# Name

strcmpi — compara strings (caso independente)

```
res = strcmpi(string_one,string_two)
```

# Parâmetros

string_one
: string ou matriz de strings

string_two
: string ou matriz de strings

res
: matriz

# Descrição

`res = strcmpi(string_one,string_two)` retorna um valor inteiro indicando a relação entre os strings.

Um valor inteiro maior que zero indica que o primeiro caractere que não corresponde possui valor maior em `string_one` que em `string_two`

Um valor negativo indica o contrário.

# Exemplos

```
TXT1 = ['scilab','SciLab';'Strcmp','STRcmp'];
TXT2 = ['ScIlAb','sciLab';'sTrCmP','StrCMP'];
strcmpi(TXT1,TXT2)
strcmpi(TXT1,'scilab')
```

# Ver Também

strcat, strcmp

# Name

strcspn — retorna extensão até um caractere em um string

```
res = strcspn(string_one,string_two)
```

# Parâmetros

string_one
: string ou matriz de strings

string_two
: string ou matriz de strings

res
: matriz

# Descrição

`res = strcspn(string_one,string_two)` lê string_one para a primeira ocorrência de qualquer caractere que esteja em string_two, retornando o número de caracteres de string_one lidos antes da primeira ocorrência.

string_one deve ter as mesmas dimensões que string_two, ou string_one deve ser um string.

# Exemplos

```
strcspn("fcba73","1234567890")
strcspn(["fcba73","f7cba73"],"1234567890")
strcspn(["fcba73","f7cba73"],["312","34567890"])
```

# Ver Também

strspn

# Name

strindex — procura posição de um string em outro

```
ind=strindex(haystack,needle,[flag])
[ind,which]=strindex(haystack,needle,[flag])
```

# Parâmetros

haystack
: string. O string onde se procurará por ocorrências de `needle`

needle
: string ou vetor de strings . O(s) string(s) a serem procurados em `haystack`

ind
: vetor de índices

which
: vetor de índices

flag
: string ("r" para expressão regular)

# Descrição

`strindex` procura índices onde `needle (i)` é encontrado em `haystack`

Para cada `k` existe um `i` tal que `part(haystack,ind(k)+(0:length(needle(i))-1))` é o mesmo string que `needle(i)`. Se o argumento `which` for requerido, ele contém esses `i`. Quando se usa o terceiro parâmetro "r", needle deve ser um string de expressão regular. Então, strindex irá corresponder a haystack de acordo com as regras regulares expressas.

`strindex` sem expressão regular é baseado no algoritmo de Knuth-Morris-Pratt.

Este algoritmo é mais poderoso que aquele usado no Scilab 4.x. Em alguns casos especiais, o resultado pode ser diferente.

Exemplo:

// Scilab 5.x

-->[k,w]=strindex('aab',['a','ab'])

w = 1. 1. 2. k = 1. 2. 2.

// scilab 4.x

-->[k,w]=strindex('aab',['a','ab'])

w = 1. 1. k = 1. 2.

As regras de expressão regular são similares às da linguagem Perl. Para uma introdução rápida , ver http://perldoc.perl.org/perlrequick.html. Para um tutorial mais profundo , ver http://perldoc.perl.org/perlretut.html e para a página de referência, ver http://perldoc.perl.org/perlre.html

# Exemplos

```
k=strindex('SCI/demos/scicos','/')
k=strindex('SCI/demos/scicos','SCI/')
k=strindex('SCI/demos/scicos','!')
k=strindex('aaaaa','aa')
k=strindex('SCI/demos/scicos',['SCI','sci'])
[k,w]=strindex('1+3*abc/2.33',['+','-','*','/'])
k=strindex('2' ,'/2(]*)?$\1/' ,'r')
```

# Ver Também

string, strings, regexp, strsubst

# Name

string — conversão para string (cadeia de caracteres)

```
string(x)
[out,in,text]=string(x)
```

# Parâmetros

x
: matriz de reais ou função

# Descrição

Converte uma matriz em uma matriz de strings.

Se `x` é uma função `[out,in,text]=string(x)` retorna três vetores de strings : `out` é o vetor de variáveis de saída, `in` é o vetor de variáveis de entrada, e `text` é o vetor (coluna) do código fonte da função.

Se `x` é uma variável `lib` (bibiloteca), text é um vetor coluna de strings. O primeiro elemento contém o endereço do arquivo biblioteca e o outro o nome da função que ela define.

Strings são definidos como `'string'` (entre aspas simples) ou `"string"` (entre aspas duplas); matrizes de strings são definidas como matrizes de constantes.

Concatenação de strings é feita pela operação `+`.

# Exemplos

```
string(rand(2,2))
deff('y=mymacro(x)','y=x+1')
[out,in,text]=string(mymacro)
x=123.356; 'Result is '+string(x)

disp('/'+string(~%t)+'/')
disp('/'+string(%i+1)+'/')
disp('/'+string(int16(-123))+'/')
disp('/'+string(1+%s+%s^3)+'/')
```

# Ver Também

part, length, quote, evstr, execstr, strsubst, strcat, strindex, sci2exp

# Name

strings — objeto Scilab, strings (cadeias de caracteres)

# Descrição

Strings são definidos como `'string'` (entre aspas simples) ou `"string"` (entre aspas duplas); matrizes de strings são definidascomo matrizes constantes, como de uso.

A concatenação de dois strings é feita por `-+` : `string1+string2`.

# Exemplos

```
['esta','é'; 'uma matriz','2x2']
"matrix"=="mat"+"rix"
```

# Ver Também

part, length, strcat

# Name

stripblanks — retira espaços em branco (e tabulações) no início ou no fim de strings

```
txt=stripblanks(txt[,tabs])
```

# Parâmetros

txt
: um string ou uma matriz de strings

tabs
: se TRUE, retira também tabulações (o valor padrão é FALSE)

# Descrição

stripblanks retira espaços em branco (e tabulações) no início ou no fim de strings.

# Exemplos

```
a='  123   ';
'!'+a+'!'
'!'+stripblanks(a)+'!'
a=['  123   ',' xyz']
strcat(stripblanks(a))
```

# Name

strncpy — copia caracteres de strings

```
res = strncpy(str1,num)
```

# Parâmetros

str1
: string ou matriz de strings

num
: matriz. Números máximos de caracteres a serem copiados da fonte

res
: string ou matriz de strings

# Descrição

`res = strncpy(str1,num)` copia os primeiros num caracteres da fonte para o destino.

num deve ter as mesmas dimensões que str1, ou deve ser um número.

# Exemplos

```
strncpy('scilab',3)
strncpy(['scilab','SciLab';'strncpy','strstr'],3)
strncpy(['scilab','SciLab';'strncpy','strstr'],[1,2;3,4])
```

# Ver Também

strcat, strcmp

# Name

strrchr — acha a última ocorrência de um caractere em um string

```
res = strrchr(str1,char)
```

# Parâmetros

str1
: string ou matriz de strings

char
: caractere

res
: string ou matriz de strings

# Descrição

`res = strrchr(str1,char)` retorna a última ocorrência de caractere no string str

num deve ter as mesmas dimensões que str1, ou apenas um caracteres char.

# Exemplos

```
strrchr('Este e um string de amostra','s')
strrchr(['Este e um string de amostra','no scilab'],'s')
strrchr(['Este e um string de amostra','no scilab'],['s','a'])
```

# Ver Também

strchr, strstr

# Name

strrev — retorna um string invertido

```
res = strrev(str1)
```

# Parâmetros

str1
: um string ou uma matriz de strings

res
: um string ou uma matriz de strings

# Descrição

`res = strrev(str1)` retorna o string em ordem inversa

# Exemplos

```
rev = strrev('This is a simple string')
strrev(rev)
strrev(['This is a simple string','scilab'])
```

# Name

strsplit — divide um string em um vetor de strings

```
v = strsplit(str,ind)
```

# Parâmetros

str
: string

ind
: um vetor de índices estritamente crescentes no intervalo `[1 length(str)-1]` .

v
: o vetor coluna resultante de strings (dimensão `size(ind,'*')+1` ).

# Descrição

`v = strsplit(str,ind)` divide o string `str` em um vetor de strings nos pontos dados pelos índices em `ind` (após cada caractere apontado pelo índice em ind).

# Exemplos

```
S='strsplit divide um string em um vetor de strings';
strsplit(S,[15 25 30])
ind=strindex(S,' ')
```

# Ver Também

strcat, tokens

# Autor

S. Steer
: INRIA

# Name

strspn — retorna a extensão do conjunto de caracteres no string

```
res = strspn(str1,str2)
```

# Parâmetros

str1
: string ou matriz de strings

str2
: string ou matriz de strings

res
: matriz

# Descrição

`res = strspn(str1,str2)` retorna o comprimento da porção inicial de str1 que consiste apenas de caracteres que fazem parte de str2.

str2 deve ter as mesmas dimensões que str2, ou str1 pode ser um string.

# Exemplos

```
i = strspn("129o","1234567890");
printf ("O comprimento do numero inicial e %d.\n",i);
i = strspn(["129o","130o"],["1234567890","130o"])
```

# Ver Também

strcspn

# Name

strstr — localiza sub-strings

```
res = strstr(haystack,needle)
```

# Parâmetros

haystack
: string ou matriz de strings

needle
: string ou matriz de strings

res
: string ou matriz de strings

# Descrição

`res = strstr(haystack,needle)` retorna uma matriz de strings começando onde da primeira ocorrência de needle em haystack até o fim de haystack, ou '' se needle não é parte de haystack.

# Exemplos

```
strstr('Este é um string simples','simples')
strstr('Este é um string simples','samples')
strstr(['Este é um string simples','no scilab'],'é')
strstr(['Este é um string simples','no scilab'],['um','scilab'])
```

# Ver Também

strrchr, strchr

# Name

strsubst — substitui um string por outro dentro de um string

```
string_out=strsubst(string_in,searchStr,replaceStr)
string_out=strsubst(string_in,searchStr,replaceStr,[flag])
```

# Parâmetros

string_in
: matriz de strings. Os strings nos quais serão procuradas ocorrências de `searchStr`

searchStr
: string a se procurar em `string`.

replaceStr
: string. O string de reposição

str_out
: matriz de strings. O resultado da substituição em `searchStr` por `replaceStr` em `string`

flag
: string ("r" para expressão regular)

# Descrição

`strsubst` substitui todas as ocorrências de `searchStr` em `string` por `replaceStr`.

Quando se utiliza o quarto argumento "r", searchStr deve ser um string de uma expressão regular. Então, strsubst o fará corresponder com string e substituirá de acordo com a regra expressas regulares.

# Exemplos

```
strsubst('SCI/demos/scicos','SCI','.')
strsubst('SCI/demos/scicos','/',' ')
strsubst('2' ,'/2(]*)?$\1/' ,'0','r')
```

# Ver Também

string, strings

# Name

strtod — converte um string para um double

```
d =  strtod(str)
[d,endstr] = strtod(str)
```

# Parâmetros

str
: string ou matriz de strings

d
: escalar real ou matriz de reais

endstr
: string ou matriz de strings (próximo caractere em str após o valor numérico).

# Descrição

`[d,endstr] = strtod(str)` analisa o string str interpretando o seu conteúdo como um número de ponto flutuante e retorna o seu valor como um real.

# Exemplos

```
strtod('123.556Este &#233; um real de amostra')
[d,endstr] = strtod('123.556Este &#233; um real de amostra)
strtod(['123.556Este &#233; um real de amostra','888.666 aqui'])
[d,endstr] =strtod(['123.556Este &#233; um real de amostra','888.666 aqui'])
```

# Name

strtok — divide um string em fichas (tokens)

```
res = strtok(str,delimiters)
```

# Parâmetros

str
: string

delimiters
: string

res
: string

# Descrição

`res = strtok(str,delimiters)` divide str em fichas, que são seqüências de caracteres contíguos separados por qualquer um dos caracteres que fazem parte do string delimiters (delimiters = delimitadores).

# Exemplos

```
TOKENS = [];
token = strtok("Um string de ,,fichas e algumas fichas mais"," ,");
TOKENS = [TOKENS,token];
while( token <> '' )
  token = strtok(" ,");
  TOKENS = [TOKENS,token];
end
disp(TOKENS);
```

# Ver Também

strrchr, strchr

# Name

tokenpos — retorna as posições das fichas (token) em um string

```
kdf=tokenpos(str [,delimiter])
```

# Parâmetros

str
: string. O string onde se deve procurar fichas

delimiter
: (opcional) um caractere ou vetor de caracteres. Os delimitadores de fichas.

kdf
: matriz de duas colunas, a primeira coluna fornece os índices dos primeiros caracteres das fichas, a segunda fornece os índices dos últimos caracteres das fichas.

# Descrição

`kdf=tokenpos(str [,delimiter])` busca as fichas inclusas no string `str`. O valor padrão de `delimiter` é [" ",<Tab>] onde <Tab> é `ascii(9)`. Retorna os índices dos primeiros e últimos caracteres de cada ficha encontrada.

# Exemplos

```
str='Isto é um string';
kdf=tokenpos(str)
first=part(str,kdf(1,1):kdf(1,2))
```

# Ver Também

strindex, tokens

# Name

tokens — retorna as fichas (tokens) de um string

```
T=tokens(str [,delimiter])
```

# Parâmetros

str
: string, o local de procura dos tokens

delimiter
: (opcional) um caractere ou um vetor de caracteres. Os delimitadores de fichas.

T
: vetor coluna de fichas encontradas

# Descrição

`T=tokens(str [,delimiter])` procura pelos tokens inclusos no string `str`. O padrão de `delimiter` é [" ",<Tab>] onde <Tab> é `ascii(9)`.

# Exemplos

```
tokens('Isto é um string')

tokens('SCI/demos/scicos','/')

tokens('y=a+b*2',['=','+','*'])
```

# Ver Também

strindex, tokenpos

# Name

tree2code — gera a definição ASCII de uma função Scilab

```
txt=tree2code(tree,prettyprint)
```

# Parâmetros

tree
: uma árvore macro (vinda de macr2tree)

prettyprint
: valor opcional booleano

  %T
  : o código gerado é identado e embelezado

  %F
  : o código gerado não é embelezado (padrão)

txt
: um vetor coluna de strings, o texto fornecendo as instruções Scilab

# Descrição

Dada uma função Scilab carregada "tree" (retornada por macr2tree), tree2code permite gerar novamente o código.

# Exemplos

```
tree=macr2tree(cosh);
txt=tree2code(tree,%T);
write(%io(2),txt,'(a)');
```

# Ver Também

macr2tree

# Autor

V.C.

# Parte XXXII. Cálculos Formais

# Name

addf — adição simbólica

```
addf("a","b")
```

# Parâmetros

"a","b"
: strings

# Descrição

`addf("a","b")` retorna a cadeia de caracteres `"a+b"`. Simplificações triviais tais como `addf("0","a")` ou `addf("1","2")` são realizadas.

# Exemplos

```
addf('0','1')
addf('1','a')
addf('1','2')
'a'+'b'
```

# Ver Também

mulf, subf, ldivf, rdivf, eval, evstr

# Name

ldivf — divisão simbólica esquerda (invertida)

```
ldivf('d','c')
```

# Descrição

Retorna o string `'c\d'` Simplificações triviais como `'1\c' = 'c'` são realizadas.

# Exemplos

```
ldivf('1','1')
ldivf('a','0')
ldivf('a','x')
ldivf('2','4')
```

# Ver Também

rdivf, addf, mulf, evstr

# Name

mulf — multiplicação simbólica

```
mulf('d','c')
```

# Descrição

Retorna o string `'c*d'` Simplificações triviais tais como `'1*c' = 'c'` são realizadas.

# Exemplos

```
mulf('1','a')
mulf('0','a')
'a'+'b'   //Cuidado...
```

# Ver Também

rdivf, addf, subf

# Name

rdivf — divisão simbólica direita-esquerda

```
["r"]=ldivf("d","c")
```

# Parâmetros

"d","c","r"
: strings

# Descrição

Retorna o string `"c/d"` Simplificações triviais tais como `"c/1" = "c"` são realizadas.

# Exemplos

```
ldivf('c','d')
ldivf('1','2')
ldivf('a','0')
```

# Ver Também

ldivf

# Name

subf — subtração simbólica

```
["c"]=subf("a","b")
```

# Parâmetros

"a","b","c"
: strings

# Descrição

Retorna o string `c="a-b"`. Simplificações triviais como `subf("0","a")` ou `subf("1","2")` são realizadas.

# Exemplos

```
subf('0','a')
subf('2','1')
subf('a','0')
```

# Ver Também

mulf, ldivf, rdivf, eval, evstr

# Parte XXXIII. Data e Hora

# Name

date — retorna string contendo a data corrente

```
dt=date()
```

# Parâmetros

dt
: a string

# Descrição

dt=date() retorna um string contendo a data no formato dd-mmm-yyyy.

# Exemplos

```
date()
```

# Ver Também

getdate, toc, tic, timer, etime

# Name

etime — tempo decorrido

```
e = etime(t2,t1)
```

# Parâmetros

t2
: um vetor com 6 ou 10 valores

t1
: um vetor com 6 ou 10 valores

e
: número de segundos entre `t2` e `t1`.

# Descrição

t1 e t2 com 10 valores
: `t2` e `t1` devem ter o mesmo formato retornado por `getdate`. Neste caso, seus terceiro, quarto e quinto valores serão ignorados.

t1 e t2 com 6 valores
: `t2` e `t1` devem ter o mesmo formato: T = [Ano Mês Dia Hora Minuto Segundo] com Segundo o número de segundos com milisegundos (ex.: 12.345).

`t2` e `t1` devem ter o mesmo tamanho.

`t2` e `t1` podem ser matrizes com cada linha contendo o formato descrito acima (todas as linhas tendo o mesmo formato).

# Exemplos

```
t1=[2004 06 10 17 00 12.345]
t2=[2004 06 10 17 01 13.965]
E1=etime(t2,t1)
t1=[2004 06 24 162 5 10 17 00 12 345]
t2=[2004 06 24 162 5 10 17 01 13 965]
E2=etime(t2,t1)
```

# Ver Também

tic, toc, getdate, datenum, datevec

# Autor

V.C.

# Name

getdate — retorna informação sobre data e hora

```
dt=getdate()
x=getdate("s")
dt=getdate(x)
```

# Parâmetros

dt
: um vetor de inteiros com 10 entradas (ver abaixo)

x
: um inteiro contendo uma data codificada em segundos a partir de 01/01/1970

# Descrição

dt=getdate()
: retorna a dada corrente no formato dado abaixo:

  dt(1)
  : o ano como um número (com o século) entre 0000 e 9999.

  dt(2)
  : o mês do ano como um número entre 01 e 12.

  dt(3)
  : o número da semana ISO 8601 como um número entre 01 e 53.

  dt(4)
  : o dia Juliano do ano como um número entre 001 e 366.

  dt(5)
  : especifica o dia da semana como um número decimal entre 1 e 7 , com 1 representando domingo.

  dt(6)
  : o dia do mês como um número entre 01 e 31.

  dt(7)
  : a hora do dia como um número entre 00 e 23.

  dt(8)
  : o minuto da hora como um número entre 00 e 59.

  dt(9)
  : o segundo do minuto como um número entre 00 e 59.

  dt(10)
  : o milissegundo do segundo como um número entre 000 e 999.

x=getdate("s")
: retorna um escalar com o número de segundos desde 01/01/1970, 00:00 UTC (Convenção de Tempo Unix)

dt=getdate(x)
: põe a data dada por x (número de segundos desde 01/01/1970, 00:00 UTC) no formato acima. Neste caso, dt(10) é sempre igual a 0.

# Exemplos

```
w=getdate()
mprintf("Ano:%d,Mês:%d,Dia:%d",w(1),w(2),w(6));

x=getdate("s")
getdate(x)
```

# Ver Também

date, timer

# Autor

V.C.

# Name

tic — inicia um cronômetro

```
tic()
```

# Descrição

A seqüência de comando `tic(); operation; toc();` imprime o número de segundos requeridos para a operação.

# Exemplos

```
tic();
realtimeinit(1);
realtime(0);
realtime(10);
toc();
```

# Ver Também

toc, timer, etime

# Autores

V.C.
A.C.

# Name

toc — lê o cronômetro

```
toc()
t = toc()
```

# Parâmetros

t
: número de segundos desde a última chamada a `tic()` (precisão em ordem de milissegundos)

# Descrição

A seqüência de comandos `tic(); operation; toc();` imprime o número de segundos requeridos para a operação.

# Exemplos

```
tic();
realtimeinit(1);
realtime(0);
realtime(10);
toc();
```

# Ver Também

tic, timer, etime

# Autores

V.C.
A.C.

# Name

calendar — Calendar

```
c=calendar()
c = calendar(y,m)
```

# Description

c = calendar returns a list containing a calendar for the current month. The calendar runs Sunday to Saturday.

c = calendar(y,m), where y and m are integers, returns a calendar for the specified month of the specified year.

# Examples

```
calendar()
calendar(1973,8)
```

# See Also

datevec , datenum

# Authors

Allan CORNET

# Name

clock — Return current time as date vector

```
c = clock
```

# Description

c = clock returns a 6-element date vector containing the current date and time in decimal form:

c = [year month day hour minute seconds]

the first five elements are integers. The seconds element is accurate to several digits beyond the decimal point.

# Examples

```
clock
```

# See Also

datenum , datevec , timer , etime , tic , toc

# Authors

P.M

# Name

datenum — Convert to serial date number

```
N = datenum()
N = datenum(DT)
N = datenum(Y, M, D)
N = datenum(Y, M, D, H, MI, S)
```

# Description

The datenum function converts date vectors (defined by datevec) into serial date numbers. Date numbers are serial days elapsed from some reference date. By default, the serial day 1 corresponds to 1-Jan-0000.

N = datenum() returns the serial date numbers corresponding to current date.

N = datenum(DT) converts one or more date vectors to serial date number N. DT can be an m-by-6 or m-by-3 matrix containing m full or partial date vector respectively.

N = datenum(Y, M, D) returns the serial date numbers for corresponding elements of the Y, M, and D (year, month, day) arrays. Y, M and D must be arrays of the same size (or any can be a scalar).

N = datenum(Y, M, D, H, MI, S) returns the serial date numbers for corresponding elements of the Y, M, D, H, MI, and S (year, month, day, hour, minute, and second) array values.Y, M, D, H, MI, and S must be arrays of the same size (or any can be a scalar).

# Examples

```
// N = datenum()
datenum()

// N = datenum(DT)
A = [ 0 1 1 0 0 0 ; 2005 2 8 21 37 30 ]
datenum(A)

// N = datenum(Y, M, D)

Years = [0; 1973; 2006]
Months = [1; 8; 2]
Days = [1; 4; 8]
datenum(Years,Months,Days)

Years = [0 0 0 ; 0 0 0]
Months = [1 1 1 ; 1 1 1]
Days = [1 2 3 ; 4 5 6]
datenum(Years,Months,Days)

// N = datenum(Y, M, D, H, MI, S)

Years = grand(5,10,'uin',0,2006)
Months = grand(5,10,'uin',1,12)
Days = grand(5,10,'uin',1,28)
Hours = grand(5,10,'uin',0,23)
Minutes = grand(5,10,'uin',0,59)
Seconds = grand(5,10,'uin',0,59)
datenum(Years,Months,Days,Hours,Minutes,Seconds)
```

## See Also

datevec , calendar

## Authors

A.C

# Name

datevec — Date components

```
V=datevec(DT)
[Y,M,D,H,MI,S]=datevec(DT)
```

# Description

V = datevec(DT) converts a serial date number (defined by datenum) to a date vector V having elements [year, month, day, hour, minute, second]. The first five vector elements are integers. DT can be an array.

[Y, M, D, H, MI, S] = datevec(DT) returns the components of the date vector as individual variables. DT can be an array.

# Examples

```
// First example
datevec(720840)

// Second example
datevec(datenum())

// Third example (With integers values)
A = grand(10,12,'uin',1,1000000)
datevec(A)

// Fourth example (With real values)
A = grand(10,12,'unf',1,1000000)
datevec(A)
```

# See Also

datenum , calendar

# Authors

A.C

# Name

eomday — Return last day of month

```
E = eomday(Y, M)
```

# Description

E = eomday(Y, M) returns the last day of the year and month given by corresponding elements of arrays Y and M.

# Examples

```
eomday(2006,3);
```

# See Also

datenum , datevec , weekday

# Authors

P.M

# Name

now — Return current date and time

```
t = now()
```

# Description

t = now() returns date and time as a serial date number. (See datenum)

# Examples

```
realtimeinit(1);
realtime(0);
t1 = now()
datevec(t1)
realtime(10);
t1 = now()
datevec(t1)
```

# See Also

clock , datenum , datevec

# Authors

P.M

# Name

realtimeinit — set time unit
realtime — set dates origin or waits until date

```
realtimeinit(time_unit)
realtime(t)
```

# Parameters

time_unit
: a real number. The number of seconds associated to the `realtime` argument

t
: a real number. A date

# Description

These two functions can be used to handle real time into Scilab.

`realtimeinit(time_unit)` defines the time unit associated to the `realtime` argument `t`

first call to `realtime(t0)` sets current date to (`t0`). subsequent calls to `realtime(t)` wait till date `t` is reached.

# Examples

```
realtimeinit(1/2);//sets time unit to half a second
realtime(0);//sets current date to 0
for k=1:10,realtime(k);mprintf('current time is '+string(k/2)+'sec .\r\n');end

//next instruction outputs a dot each 2 seconds
 realtimeinit(2); realtime(0);for k=1:10,realtime(k);mprintf('.\r\n');end

realtimeinit(1);realtime(0);
dt=getdate('s'); realtime(10);   getdate('s')-dt
```

# See Also

getdate

# Name

sleep — suspend Scilab

```
sleep(milliseconds)
```

# Description

`sleep` : Sleep process for specified number of miliseconds specified by the argument. The actual suspension time may be longer because of other activities in the system, or because of the time spent in processing the call.

# Examples

```
tic;sleep(6000);toc
```

# See Also

xpause , pause

# Authors

Allan CORNET

# Name

timer — cpu time

```
timer()
```

# Description

Returns the CPU time since the preceding call to `timer()`.

timer has a time precision of 100 nanoseconds.

NOTE: CPU time is the number of processor cycles used for a computation. This is not at all equivalent to real-world time.

CPU time can be used to compare CPU usage between different programs or functions , irrespective of background processes that might slow down the computer.

# Examples

```
timer();A=rand(100,100);timer()
```

# See Also

getdate, toc, tic, etime

# Name

weekday — Return day of week

```
[N,S] = weekday(D)
[N,S] = weekday(D, form)
```

# Description

[N,S] = weekday(D) returns the day of the week in numeric(N) and string(S) form for a given serial date number or date string D. Input argument D can represent more than one date in an array of serial date number.

[N,S] = weekday(D, form) returns the week in numeric(N) and string(S) form, where the content of S depends on the form argument. If form is 'long', then S countains the full name of the weekday (e.g, Thuesday). If form is 'short', then S contains an abbreviated name (e.g., Tue) from this table.

# Examples

```
today = datenum();
[N,S] = weekday(today)
[N,S] = weekday(today,'short')
[N,S] = weekday(today,'long')
```

# See Also

datenum , datevec , weekday

# Authors

P.M

# Parte XXXIV. Estatística

# Name

cdfbet — cumulative distribution function Beta distribution

```
[P,Q]=cdfbet("PQ",X,Y,A,B)
[X,Y]=cdfbet("XY",A,B,P,Q)
[A]=cdfbet("A",B,P,Q,X,Y)
[B]=cdfbet("B",P,Q,X,Y,A)
```

# Parameters

P,Q,X,Y,A,B
: five real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to X of the beta distribution (Input range: [0, 1].)

Q
: 1-P

X,Y (Y=1-X)
: Upper limit of integration of beta density (Input range: [0,1], Search range: [0,1]) A,B : The two parameters of the beta density (input range: (0, +infinity), Search range: [1D-300,1D300] )

# Description

Calculates any one parameter of the beta distribution given values for the others (The beta density is proportional to `t^(A-1) * (1-t)^(B-1)`.

Cumulative distribution function (P) is calculated directly by code associated with the following reference.

DiDinato, A. R. and Morris, A. H. Algorithm 708: Significant Digit Computation of the Incomplete Beta Function Ratios. ACM Trans. Math. Softw. 18 (1993), 360-373.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

# Name

cdfbin — cumulative distribution function Binomial distribution

```
[P,Q]=cdfbin("PQ",S,Xn,Pr,Ompr)
[S]=cdfbin("S",Xn,Pr,Ompr,P,Q)
[Xn]=cdfbin("Xn",Pr,Ompr,P,Q,S)
[Pr,Ompr]=cdfbin("PrOmpr",P,Q,S,Xn)
```

# Parameters

P,Q,S,Xn,Pr,Ompr
: six real vectors of the same size.

P,Q (Q=1-P)
: The cumulation from 0 to S of the binomial distribution. (Probablility of S or fewer successes in XN trials each with probability of success PR.) Input range: [0,1].

S
: The number of successes observed. Input range: [0, XN] Search range: [0, XN]

Xn
: The number of binomial trials. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Pr,Ompr (Ompr=1-Pr)
: The probability of success in each binomial trial. Input range: [0,1]. Search range: [0,1]

# Description

Calculates any one parameter of the binomial distribution given values for the others.

Formula 26.5.24 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the binomial distribution to the cumulative incomplete beta distribution.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

# Name

cdfchi — cumulative distribution function chi-square distribution

```
[P,Q]=cdfchi("PQ",X,Df)
[X]=cdfchi("X",Df,P,Q);
[Df]=cdfchi("Df",P,Q,X)
```

# Parameters

P,Q,Xn,Df
: four real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to X of the chi-square distribution. Input range: [0, 1].

X
: Upper limit of integration of the non-central chi-square distribution. Input range: [0, +infinity). Search range: [0,1E300]

Df
: Degrees of freedom of the chi-square distribution. Input range: (0, +infinity). Search range: [ 1E-300, 1E300]

# Description

Calculates any one parameter of the chi-square distribution given values for the others.

Formula 26.4.19 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the chi-square distribution to the incomplete distribution.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

# Name

cdfchn — cumulative distribution function non-central chi-square distribution

```
[P,Q]=cdfchn("PQ",X,Df,Pnonc)
[X]=cdfchn("X",Df,Pnonc,P,Q);
[Df]=cdfchn("Df",Pnonc,P,Q,X)
[Pnonc]=cdfchn("Pnonc",P,Q,X,Df)
```

# Parameters

P,Q,X,Df,Pnonc
: five real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to X of the non-central chi-square distribution. Input range: [0, 1-1E-16).

X
: Upper limit of integration of the non-central chi-square distribution. Input range: [0, +infinity). Search range: [0,1E300]

Df
: Degrees of freedom of the non-central chi-square distribution. Input range: (0, +infinity). Search range: [ 1E-300, 1E300]

Pnonc
: Non-centrality parameter of the non-central chi-square distribution. Input range: [0, +infinity). Search range: [0,1E4]

# Description

Calculates any one parameter of the non-central chi-square distribution given values for the others.

Formula 26.4.25 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to compute the cumulative distribution function.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

The computation time required for this routine is proportional to the noncentrality parameter (PNONC). Very large values of this parameter can consume immense computer resources. This is why the search range is bounded by 10,000.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

# Name

cdff — cumulative distribution function F distribution

```
[P,Q]=cdff("PQ",F,Dfn,Dfd)
[F]=cdff("F",Dfn,Dfd,P,Q);
[Dfn]=cdff("Dfn",Dfd,P,Q,F);
[Dfd]=cdff("Dfd",P,Q,F,Dfn)
```

# Parameters

P,Q,F,Dfn,Dfd
: five real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to F of the f-density. Input range: [0,1].

F
: Upper limit of integration of the f-density. Input range: [0, +infinity). Search range: [0,1E300]

Dfn
: Degrees of freedom of the numerator sum of squares. Input range: (0, +infinity). Search range: [ 1E-300, 1E300]

Dfd
: Degrees of freedom of the denominator sum of squares. Input range: (0, +infinity). Search range: [ 1E-300, 1E300]

# Description

Calculates any one parameter of the F distribution given values for the others.

Formula 26.6.2 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function for the F variate to that of an incomplete beta.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

The value of the cumulative F distribution is not necessarily monotone in either degrees of freedom. There thus may be two values that provide a given CDF value. This routine assumes monotonicity and will find an arbitrary one of the two values.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

# Name

cdffnc — cumulative distribution function non-central f-distribution

```
[P,Q]=cdffnc("PQ",F,Dfn,Dfd,Pnonc)
[F]=cdffnc("F",Dfn,Dfd,Pnonc,P,Q);
[Dfn]=cdffnc("Dfn",Dfd,Pnonc,P,Q,F);
[Dfd]=cdffnc("Dfd",Pnonc,P,Q,F,Dfn)
[Pnonc]=cdffnc("Pnonc",P,Q,F,Dfn,Dfd);
```

# Parameters

P,Q,F,Dfn,Dfd,Pnonc
: six real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to F of the non-central f-density. Input range: [0,1-1E-16).

F
: Upper limit of integration of the non-central f-density. Input range: [0, +infinity). Search range: [0,1E300]

Dfn
: Degrees of freedom of the numerator sum of squares. Input range: (0, +infinity). Search range: [ 1E-300, 1E300]

Dfd
: Degrees of freedom of the denominator sum of squares. Must be in range: (0, +infinity). Input range: (0, +infinity). Search range: [ 1E-300, 1E300]

Pnonc
: The non-centrality parameter Input range: [0,infinity) Search range: [0,1E4]

# Description

Calculates any one parameter of the Non-central F distribution given values for the others.

Formula 26.6.20 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to compute the cumulative distribution function.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

The computation time required for this routine is proportional to the noncentrality parameter (PNONC). Very large values of this parameter can consume immense computer resources. This is why the search range is bounded by 10,000.

The value of the cumulative noncentral F distribution is not necessarily monotone in either degrees of freedom. There thus may be two values that provide a given CDF value. This routine assumes monotonicity and will find an arbitrary one of the two values.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

# Name

cdfgam — cumulative distribution function gamma distribution

```
[P,Q]=cdfgam("PQ",X,Shape,Scale)
[X]=cdfgam("X",Shape,Scale,P,Q)
[Shape]=cdfgam("Shape",Scale,P,Q,X)
[Scale]=cdfgam("Scale",P,Q,X,Shape)
```

# Parameters

P,Q,X,Shape,Scale
: five real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to X of the gamma density. Input range: [0,1].

X
: The upper limit of integration of the gamma density. Input range: [0, +infinity). Search range: [0,1E300]

Shape
: The shape parameter of the gamma density. Input range: (0, +infinity). Search range: [1E-300,1E300]

Scale
: The scale parameter of the gamma density. Input range: (0, +infinity). Search range: (1E-300,1E300]

# Description

Calculates any one parameter of the gamma distribution given values for the others.

Cumulative distribution function (P) is calculated directly by the code associated with:

DiDinato, A. R. and Morris, A. H. Computation of the incomplete gamma function ratios and their inverse. ACM Trans. Math. Softw. 12 (1986), 377-393.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

The gamma density is proportional to `T**(SHAPE - 1) * EXP(- SCALE * T)`

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

# Name

cdfnbn — cumulative distribution function negative binomial distribution

```
[P,Q]=cdfnbn("PQ",S,Xn,Pr,Ompr)
[S]=cdfnbn("S",Xn,Pr,Ompr,P,Q)
[Xn]=cdfnbn("Xn",Pr,Ompr,P,Q,S)
[Pr,Ompr]=cdfnbn("PrOmpr",P,Q,S,Xn)
```

# Parameters

P,Q,S,Xn,Pr,Ompr
: six real vectors of the same size.

P,Q (Q=1-P)
: The cumulation from 0 to S of the negative binomial distribution. Input range: [0,1].

S
: The upper limit of cumulation of the binomial distribution. There are F or fewer failures before the XNth success. Input range: [0, +infinity). Search range: [0, 1E300]

Xn
: The number of successes. Input range: [0, +infinity). Search range: [0, 1E300]

Pr
: The probability of success in each binomial trial. Input range: [0,1]. Search range: [0,1].

Ompr
: 1-PR Input range: [0,1]. Search range: [0,1] PR + OMPR = 1.0

# Description

Calculates any one parameter of the negative binomial distribution given values for the others.

The cumulative negative binomial distribution returns the probability that there will be F or fewer failures before the XNth success in binomial trials each of which has probability of success PR.

The individual term of the negative binomial is the probability of S failures before XN successes and is Choose `( S, XN+S-1 ) * PR^(XN) * (1-PR)^S`

Formula 26.5.26 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce calculation of the cumulative distribution function to that of an incomplete beta.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

# Name

cdfnor — cumulative distribution function normal distribution

```
[P,Q]=cdfnor("PQ",X,Mean,Std)
[X]=cdfnor("X",Mean,Std,P,Q)
[Mean]=cdfnor("Mean",Std,P,Q,X)
[Std]=cdfnor("Std",P,Q,X,Mean)
```

# Parameters

P,Q,X,Mean,Std
: six real vectors of the same size.

P,Q (Q=1-P)
: The integral from -infinity to X of the normal density. Input range: (0,1].

X
: Upper limit of integration of the normal-density. Input range: ( -infinity, +infinity)

Mean
: The mean of the normal density. Input range: (-infinity, +infinity)

Sd
: Standard Deviation of the normal density. Input range: (0, +infinity).

# Description

Calculates any one parameter of the normal distribution given values for the others.

A slightly modified version of ANORM from Cody, W.D. (1993). "ALGORITHM 715: SPECFUN - A Portabel FORTRAN Package of Special Function Routines and Test Drivers" acm Transactions on Mathematical Software. 19, 22-32. is used to calulate the cumulative standard normal distribution.

The rational functions from pages 90-95 of Kennedy and Gentle, Statistical Computing, Marcel Dekker, NY, 1980 are used as starting values to Newton's Iterations which compute the inverse standard normal. Therefore no searches are necessary for any parameter.

For X < -15, the asymptotic expansion for the normal is used as the starting value in finding the inverse standard normal. This is formula 26.2.12 of Abramowitz and Stegun.

The normal density is proportional to exp( - 0.5 * (( X - MEAN)/SD)**2)

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

# Name

cdfpoi — cumulative distribution function poisson distribution

```
[P,Q]=cdfpoi("PQ",S,Xlam)
[S]=cdfpoi("S",Xlam,P,Q)
[Xlam]=cdfpoi("Xlam",P,Q,S);
```

# Parameters

P,Q,S,Xlam
: four real vectors of the same size.

P,Q (Q=1-P)
: The cumulation from 0 to S of the poisson density. Input range: [0,1].

S
: Upper limit of cumulation of the Poisson. Input range: [0, +infinity). Search range: [0,1E300]

Xlam
: Mean of the Poisson distribution. Input range: [0, +infinity). Search range: [0,1E300]

# Description

Calculates any one parameter of the Poisson distribution given values for the others.

Formula 26.4.21 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function to that of computing a chi-square, hence an incomplete gamma function.

Cumulative distribution function (P) is calculated directly. Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

# Name

cdft — cumulative distribution function Student's T distribution

```
[P,Q]=cdft("PQ",T,Df)
[T]=cdft("T",Df,P,Q)
[Df]=cdft("Df",P,Q,T)
```

# Parameters

P,Q,T,Df
: six real vectors of the same size.

P,Q (Q=1-P)
: The integral from -infinity to t of the t-density. Input range: (0,1].

T
: Upper limit of integration of the t-density. Input range: ( -infinity, +infinity). Search range: [ -1E150, 1E150 ]

DF:
: Degrees of freedom of the t-distribution. Input range: (0 , +infinity). Search range: [1e-300, 1E10]

# Description

Calculates any one parameter of the T distribution given values for the others.

Formula 26.5.27 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function to that of an incomplete beta.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

# Name

center — center

```
s=center(x)
s=center(x,'r') or s=center(x,1)
s=center(x,'c') or s=center(x,2)
```

# Parameters

x: real or complex vector or matrix

# Description

This function computes `s`, the centred version of the numerical matrix `x`. For a vector or a matrix `x`, `s=center(x)` returns in the `(i,j)` coefficient of the matrix `s` the value `(x(i,j)-xbar)`, where `xbar` is the mean of the values of the coefficients of `x`. `s=center(x,'r')` (or, equivalently, `s=center(x,1)`) is the rowwise center reduction of the values of `x`. It returns in the entry `s(i,j)` the value `(x(i,j)-xbarv(j))(j)` with `xbarv(j)` the mean of the values of the `j` column. `s=center(x,'c')` (or, equivalently, `s=center(x,2)`) is the columnwise center reduction of the values of `x`. It returns in the entry `s(i,j)` the value `(x(i,j)-xbarh(i)))` with `xbarh(i)` the mean of the values of the `i` row.

# Examples

```
x=[0.2113249  0.0002211 0.6653811;
   0.7560439  0.3303271 0.6283918]
s=center(x)
s=center(x,'r')
s=center(x,'c')
```

# See Also

wcenter

# Authors

Carlos Klimann

# Name

wcenter — center and weight

```
s=wcenter(x)
s=wcenter(x,'r') or s=wcenter(x,1)
s=wcenter(x,'c') or s=wcenter(x,2)
```

# Parameters

x: real or complex vector or matrix

# Description

This function computes `s`, the weigthed and centred version of the numerical matrix `x`.

For a vector or a matrix `x`, `s=wcenter(x)` returns in the `(i,j)` coefficient of the matrix `s` the value `(x(i,j)-xbar)/sigma`, where `xbar` is the mean of the values of the coefficients of `x` and `sigma` his standard deviation.

`s=wcenter(x,'r')` (or, equivalently, `s=wcenter(x,1)`) is the rowwise centre reduction of the values of `x`. It returns in the entry `s(i,j)` the value `(x(i,j)-xbarv(j))/sigmav(j)` with `xbarv(j)` the mean of the values of the `j` column and `sigmav(j)` the standard deviation of the `j` column of `x`.

`s=wcenter(x,'c')` (or, equivalently, `s=wcenter(x,2)`) is the columnwise centre reduction of the values of `x`. It returns in the entry `s(i,j)` the value `(x(i,j)-xbarh(i))/sigmah(i)` with `xbarh(i)` the mean of the values of the `i` row and `sigmah(i)` the standard deviation of the `i` row of `x`.

# Examples

```
x=[0.2113249 0.0002211 0.6653811;
   0.7560439 0.3303271 0.6283918]
s=wcenter(x)
s=wcenter(x,'r')
s=wcenter(x,'c')
```

# See Also

center

# Authors

Carlos Klimann

# Name

cmoment — central moments of all orders

```
mom=cmoment(x,ord)
mom=cmoment(x,ord,'r') or mom=cmoment(x,ord,1)
mom=cmoment(x,ord,'c') or mom=cmoment(x,ord,2)
```

# Parameters

x
: real or complex vector or matrix

ord
: positive integer

# Description

cmoment`(x,ord)` is the central moment or order ord of the elements of `x`. If a third argument of type string `'r'` (or `1`) or `'c'` (or `2`) is used, we get in the first case, a row vector `mom` such that `mom(j)` contains the central moment of order `ord` of the `j` column of `x`. `cmoment(x,ord,'c')` is used in the same way for the central moments in the rows.

# References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

# Examples

```
x=[0.2113249 0.0002211 0.6653811;
   0.7560439 0.3303271 0.6283918]
mom=cmoment(x,3)
mom=cmoment(x,2,'r')
mom=cmoment(x,3,'c')
```

# See Also

sum , median , st_deviation , mean , meanf , moment , nanmean , nanmeanf , stdev , stdevf , variance , variancef , nanstdev

# Authors

Carlos Klimann

# Name

correl — correlation of two variables

```
rho=correl(x,y,fre)
```

# Parameters

x
: real or complex vector

y
: real or complex vector

fre
: matrix of type length(x) x length(y)

# Description

`correl(x,y,fre)` computes the correlation of two variables x and y. fre is a matrix of dimensions length(x) x length(y). In fre the element of indices (i,j) corresponds to the value or number or frequences of x_i&y_j.

# References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

# Examples

```
x=[2.5 7.5 12.5 17.5]
h=[0 1 2]
fre=[.03 .12 .07;.02 .13 .11;.01 .13 .14;.01 .09 .14]
rho=correl(x,h,fre)
```

# See Also

covar

# Authors

Carlos Klimann

# Name

covar — covariance of two variables

```
s=covar(x,y,fre)
```

# Parameters

x
: real or complex vector

y
: real or complex vector

fre
: matrix of type length(x) x length(y)

# Description

`covar(x,y,fre)` computes the covariance of two variables x and y. fre is a matrix of dimensions length(x) x length(y). In fre the element of indices (i,j) corresponds to the value or number or frequences of x_i&y_j.

# References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

# Examples

```
x=[10 20 30 40]
y=[10 20 30 40]
fre=[.20 .04 .01  0;
     .10 .36 .09  0;
       0 .05 .10  0;
       0   0   0 .05]
s=covar(x,y,fre)
```

# Authors

Carlos Klimann

# Name

ftest — Fischer ratio

```
f=ftest(samples)
[f,p]=ftest(samples)
```

# Parameters

samples
real or complex matrix of type nr X nc

# Description

`f=ftest(samples)` computes the Fischer ratio of the nc samples whose values are in the columns of the matrix `samples`. Each one of these samples is composed of nr values. (The Fischer ratio is the ratio between nr times the variance of the means of samples and the mean of variances of each sample)

`[f,p]=ftest(samples)` gives in p the p-value of the computed Fischer ratio f.

# References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

# Examples

```
samples=[46 55 54;
         53 54 50;
         49 58 51;
         50 61 51;
         46 52 49]
[f,p]=ftest(samples)
```

# See Also

ftuneq

# Authors

Carlos Klimann

# Name

ftuneq — Fischer ratio for samples of unequal size.

```
f=ftuneq(sample1[,sample2[,sample3]...]])
[f,p]=ftuneq(sample1[,sample2[,sample3]...]])
```

# Parameters

sample1, sample2, sample3,...
: real or complex matrix of any type

# Description

This function computes the F ratio for samples of unequal size.

"The most efficient design is to make all samples the same size n. However when this is nor feasible, it still is possible to modify the ANOVA calculations." Note that the definition of xbarbar is no longer mean(xbar), but rather a weighted average with weights ni. Additionnally it gives (in p) the p-value of the computed Fischer ratio.

Given a number a of samples each of them composed of n_i (i from 1 to a) observations this fonction computes in f the Fischer ratio (it is the ratio between nr times the variance of the means of samples and the mean of the variances of each sample).

`f=ftest(samples)` computes the Fischer ratio of the nc samples whose values are in the columns of the matrix `samples`. Each one of these samples is composed of nr values. (The Fischer ratio is the ratio between nr times the variance of the means of samples and the mean of variances of each sample)

`[f,p]=ftest(samples)` gives in p the p-value of the computed Fischer ratio f.

# References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

# Examples

```
samples=[46 55 54;53 54 50; 49 58 51;50 61 51;46 52 49]
[f,p]=ftest(samples)
```

# See Also

ftuneq

# Authors

Carlos Klimann

# Name

geomean — geometric mean

```
gm=geomean(x)
gm=geomean(x,'r')(or, equivalently, gm=geomean(x,1))
gm=geomean(x,'c')(or, equivalently, gm=geomean(x,2))
```

# Parameters

x

real or complex vector or matrix

# Description

This function computes the geometric mean of a vector or matrix `x`. For a vector or matrix `x`, `gm=geomean(x)` returns in scalar `gm` the geometric mean of all the entries of `x`.

`gm=geomean(x,'r')` (or, equivalently, `gm=gmean(x,1)` ) returns in each entry of the row vector `gm` the geometric mean of each column of `x`.

`gm=geomean(x,'c')` (or, equivalently, `gm=gmean(x,2)` ) returns in each entry of the column vector `gm` the geometric mean of each row of `x` .

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

harmean — harmonic mean

```
hm=harmean(x)
hm=harmean(x,'r')(or, equivalently, hm=harmean(x,1))
hm=harmean(x,'c')(or, equivalently, hm=harmean(x,2))
```

# Parameters

x
: real or complex vector or matrix

# Description

This function computes the harmonic mean of a vector or matrix `x`. For a vector or matrix `x`, `hm=harmean(x)` returns in scalar `hm` the harmonic mean of all the entries of `x`.

`hm=harmean(x,'r')` (or, equivalently, `hm=harmean(x,1)` ) returns in each entry of the row vector `hm` the harmonic mean of each column of `x`.

`hm=harmean(x,'c')` (or, equivalently, `hm=harmean(x,2)` ) returns in each entry of the column vector `hm` the harmonic mean of each row of `x` .

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

iqr — interquartile range

```
q=iqr(x)
q=iqr(x,'r') (or, equivalently, q=iqr(x,1))
q=iqr(x,'c') (or, equivalently, q=iqr(x,2))
```

# Parameters

x
: real or complex vector or matrix

# Description

This function computes the interquartile range IQR= upper quartile - lower quartile of a vector or a matrix `x` .

For a vector or a matrix `x` , `q=iqr(x)` returns in the scalar `q` the interquartile range of all the entries of `x`.

`q=iqr(x,'r')` (or, equivalently, `q=iqr(x,1)`) is the rowwise interquartile range. It returns in each entry of the row vector `q` the interquartile range of each column of `x`.

`q=iqr(x,'c')` (or, equivalently, `q=iqr(x,2)`) is the columnwise interquartile range. It returns in each entry of the column vector `q` the interquartile range of each row of `x`.

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H.Wonacott, R.J.; Introductory Statistics, J.Wiley-Sons, 1990.

# Name

labostat — Statistical toolbox for Scilab

# Contents

```
centre: centering variables
centrered: centering and reducing variables
cmoment: central moments of all orders
correl: correlation
covar: covariance
ftest: fischer test and his p-value
geomean: geometric mean
harmean: harmonic mean
iqr: interquartile range
mad: mean absolute deviation
meanf:  arithmetic mean of a vector or matrix with a table of frequences
median: 50th percentile of a sample
mn: arithmetic mean of a vector or matrix
moment: moments of all orders
msd:  mean squared  deviation
mvvacov : multivariable matrix of variance-covariance
nand2mean: estimate of the difference of means of two independent samples
nanmax:  maximum ignoring NaNs
nanmean: mean ignoring NaNs
nanmeanf:  mean with frequency table ignoring NaNs
nanmedian: 50th percentile of a sample ignoring NaNs
nanmin: minimum ignoring NANs
nanstdev:  standard deviation ignoring NaNs
nanstdevf: standard deviation with frequency table ignoring NaNs
nansum: sum ignoring NaNs
nfreq: frequency of the values of a sample
pca: principal component analysys
pctl: vector of  percentiles of a sample in decreasing order
perctl: vector of  percentiles of a sample in decreasing order
quart: quartils
stdev: standard deviation
stdevf: standard deviation with frequences
strange: distance between largest and smallest value
tabul: frequences of values
var: variance
varf: variance with frequence table
```

# References


Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, 5th edition, J.Wiley & Sons, 1990.

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

# Name

mad — mean absolute deviation

```
s2=mad(x)
s2=mad(x,'r') or s2=mad(x,1)
s2=mad(x,'c') or s2=mad(x,2)
```

# Parameters

x

real or complex vector or matrix

# Description

This function computes the mean absolute deviation of a real or complex vector or matrix `x`.

For a vector or matrix `x`, `s2=mad(x)` returns in scalar `s2` the mean absolute deviation of all the entries of `x`.

`s2=mad(x,'r')` (or, equivalently, `s2=mad(x,1)`) returns in each entry of the column vector `s2` the mean absolute deviation of each column of `x`.

`s2=mad(x,'c')` (or, equivalently, `s2=mad(x,2)`) returns in each entry of the column vector `s2` the mean absolute deviation of each row of `x`.

# Bibliography

Reference: Wonacott T.H.& Wonacott R.J. .- Introductory Statistics, 5th edition, John Wiley, 1990.

# Name

mean — mean (row mean, column mean) of vector/matrix entries

```
y=mean(x)
y=mean(x,'r')
y=mean(x,'c')
y=mean(x,'m')
```

# Parameters

x
: real vector or matrix

y
: scalar or vector

# Description

For a vector or a matrix `x`, `y=mean(x)` returns in the scalar `y` the mean of all the entries of `x`.

`y=mean(x,'r')` (or, equivalently, `y=mean(x,1)`) is the rowwise mean. It returns a row vector: `y(j)= mean(x(:,j))`

`y=mean(x,'c')` (or, equivalently, `y=mean(x,2)`) is the columnwise mean. It returns a column vector: `y(i)= mean(x(i,:))`

`y=mean(x,'m')` is the mean along the first non singleton dimension of x (for compatibility with Matlab).

# Examples

```
A=[1,2,10;7,7.1,7.01];
mean(A)
mean(A,'r')
mean(A,'c')
A=matrix(1:12,[1,1,2,3,2]);
// in this case mean(A,'m') is equivalent to mean(A,3), the first non singleton
y=mean(A,'m')
```

# See Also

sum , median , st_deviation

# Name

meanf — weighted mean of a vector or a matrix

```
m=meanf(val,fre)
m=meanf(val,fre,'r') or m=meanf(val,fre,1)
m=meanf(val,fre,'c') or m=meanf(val,fre,2)
```

# Parameters

?

# Description

This function computes the mean of a vector or matrix `x`. For a vector or matrix `x`, `m=mn(x)` returns in scalar `m` the mean of all the entries of `x` . `m=mn(x,'r')` (or, equivalently, `m=mn(x,1)` ) returns in each entry of the row vector `m` the mean of each column of `x`. `m=mn(x,'c')` (or, equivalently, `m=mn(x,2)` ) returns in each entry of the column vector `m` the mean of each row of `x`.

# Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.3303271 0.6283918]
m=meanf(x,rand(x))
m=meanf(x,[10 10 10;1 1 1],'r')
m=meanf(x,[10 10 10;1 1 1],'c')
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

median — median (row median, column median,...) of vector/matrix/array entries

```
y=median(x)
y=median(x,'r')
y=median(x,'c')
y=median(x,'m')
y=median(x,dim)
```

# Parameters

x
: real vector, matrix or an array

y
: scalar,vector, matrix or an array

dim
: positive integer

# Description

For a vector or a matrix `x`, `y=median(x)` returns in the scalar `y` the median of all the entries of `x`.

`y=median(x,'r')` (or, equivalently, `y=median(x,1)`) is the median along the row index. It returns in each entry of the column vector `y` the median of each column of `x`.

`y=median(x,'c')` (or, equivalently, `y=median(x,2)`) is the median along the column index. It returns in each entry of the row vector `y` the median of each row of `x`.

`y=median(x,'m')` is the median along the first non singleton dimension of `x` (for compatibility with matlab).

`y=median(x,dim)` is the median along the dimension `dim` of `x` (for compatibility with matlab).

# Examples

```
A=[1,2,10;7,7.1,7.01];
median(A)
median(A,'r')
median(A,'c')
A=matrix([-9 3 -8 6 74 39 12 -6 -89 23 65 34],[2,3,2]);
median(A,3)
median(A,'m')
```

# See Also

sum , mean

# Name

moment — non central moments of all orders

```
mom=moment(x,ord)
mom=moment(x,ord,'r') or mom=moment(x,ord,1)
mom=moment(x,ord,'c') or mom=moment(x,ord,2)
```

# Parameters

x
: real or complex vector or matrix

ord
: positive integer

# Description

moment`(x,ord)` is the non central moment or order ord of the elements of `x`.

If a third argument of type string `'r'` (or `1`) or `'c'` (or `2`) is used, we get in the first case, a row vector `mom` such that `mom(j)` contains the non central moment of order `ord` of the `j` column of `x`. `moment(x,ord,'c')` is used in the same way for the non central moments in the rows.

# Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.3303271 0.6283918]
mom=moment(x,3)
mom=moment(x,2,'r')
mom=moment(x,3,'c')
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

msd — mean squared deviation

```
y=msd(x)
y=msd(x,'r') or m=msd(x,1)
y=msd(x,'c') or m=msd(x,2)
```

# Parameters

x
: real or complex vector or matrix

# Description

This function computes the mean squared deviation of the values of a vector or matrix `x`.

For a vector or a matrix `x`, `y=msd(x)` returns in the scalar `y` the mean squared deviation of all the entries of `x`.

`y=msd(x,'r')` (or, equivalently, `y=msd(x,1)` ) is the rowwise mean squared deviation. It returns in each entry of the row vector `y` the mean squared deviation of each column of `x` .

`y=msd(x,'c')` (or, equivalently, `m=msd(x,2)` ) is the columnwise mean squared deviation. It returns in each entry of the column vector `y` the mean squared deviation of each row of `x`.

# Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.3303271 0.6283918]
m=msd(x)
m=msd(x,'r')
m=msd(x,'c')
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

mvvacov — computes variance-covariance matrix

```
v=mvvacov(x)
```

# Parameters

x
: real or complex vector or matrix

# Description

This function computes v, the matrix of variance-covariance of the "tableau" x (x is a numerical matrix nxp) who gives the values of p variables for n individuals: the (i,j) coefficient of v is v(i,j)=E(xi-xibar)(xj-xjbar), where E is the first moment of a variable, xi is the i-th variable and xibar the mean of the xi variable.

# Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.4453586 0.6283918]
v=mvvacov(x)
```

# Authors

Carlos Klimann

# Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990. Mardia, K.V., Kent, J.T. & Bibby, J.M., Multivariate Analysis, Academic Press, 1979.

# Name

nancumsum — Thos function returns the cumulative sum of the values of a matrix

```
s = nancumsum(x,orient)
```

# Parameters

x
: x is a numerical vector or matrix.

orient
: is an optional parameter. The possible values are '*', 1, 2, 'r' or 'c'.

s
: numerical scalar or vector. It contains the cumulative sum of the values of x, ignoring the NAN's.

# Description

This function returns in scalar or vector s the cumulative sum of the values (ignoring the NANs) of a vector or matrix (real or complex) x.

This function for a vector or a matrix x, s=nancumsum(x) (or, equivalently s=nancumsum(x,'*') returns in scalar s the cumulative sum (ignoring the NANs) of all the entries of x taken columnwise.

s=nancumsum(x,'r') (or, equivalently, s=nancumsum(x,1)) returns in the cols(x) sized vector s the cumulative sum (ignoring the NANs) of the rows of x: s(:,i)=nancumsum(x(:,i))

s=nancumsum(x,'c') (or, equivalently, s=nancumsum(x,2)) returns in the rows(x) sized vector s the cumulative sum (ignoring NANs) of the columns of x: s(i,:)=nancumsum(x(i,:))

For the last two cases, if a row or column is in whole composed of NAN, the corresponding place of s will contain a NAN.

# Examples

```
a=[1 2 3;4 5 6]
s=nancumsum(a)
s=nancumsum(a,'r')
s=nancumsum(a,'c')
```

# See Also

nansum , cumsum

# Authors

Carlos Klimann

# Name

nand2mean — difference of the means of two independent samples

```
[dif]=nand2mean(sample1,sample2)
[dif]=nand2mean(sample1,sample2,conf)
```

# Parameters

sample1
: real or complex vector or matrix

sample2
: real or complex vector or matrix

conf
: real scalar between 0 and 1

# Description

This function computes an estimate (dif(1)) for the difference of the means of two independent samples (arrays sample1 and sample2) and gives the half amplitude of the range of variability of dif with an indicated confidence level (dif(2)). The choice of the normal or t fonctions as the probability fonction depends on the sizes of sample1 and sample2. We suppose that the underlying variances of both populations are equal. NAN values are not counted.

In Labostat, NAN values stand for missing values in tables.

In absence of the confidence parameter a confidence level of 95% is assumed.

# References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, 5th edition, J.Wiley & Sons, 1990.

# Name

nanmax — max (ignoring Nan's)

```
[m,index]=nanmax(x)
[m,index]=nanmax(x,'r')
[m,index]=nanmax(x,'c')
```

# Parameters

x
: real or complex vector or matrix

# Description

This function gives for a real or a numerical matrix `x` his largest element `m` (but ignoring the NANs).

For `x`, a numerical vector or matrix, `m=nanmax(x)` returns in scalar `m` the largest element of `x` (ignoring the NANs). The form `[m,index] =nanmax(x,orient)` gives in addition of the value of the largest element of `x` (ignoring the NANs) in scalar `m`, the index of this element in `x`, as a 2-vector.

`m=nanmax(x,'r')` gives in the 1xsize(x,2) matrix `m` the largest elements (ignoring the NANs) of each column of `x`. If the form `[m,index]=nanmax(x,'r')` is used, the elements of the 1xsize(x,2) matrix index are the indexes of the largest elements (ignoring the NANs) of each column of x in the corresponding column.

m=nanmax(x,'c') gives in the size(x,2)x1 matrix m the largest elements (ignoring the NANs) of each row of x. If the form [m,index]=nanmax(x,'c') is used, the elements of the size(x,2)x1 matrix index are the indexes of the largest elements (ignoring the NANs) of each row of x in the corresponding row.

In Labostat, NAN values stand for missing values in tables.

# Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmax(x)
m=nanmax(x,'r')
m=nanmax(x,'c')
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

nanmean — mean (ignoring Nan's)

```
m=nanmean(val)
m=nanmean(val,'r') (or m=nanmean(val,1))
m=nanmean(val,'c') (or m=nanmean(val,2))
```

# Parameters

val
: real or complex vector or matrix

# Description

This function returns in scalar `m` the mean of the values (ignoring the NANs) of a vector or matrix `val`.

For a vector or matrix `val`, `m=nanmean(val)` or `m=nanmean(val,'*')` returns in scalar `m` the mean of all the entries (ignoring the NANs) of `val`.

`m=nanmean(val,'r')` (or, equivalently, `m=nanmean(val,1)` ) returns in each entry of the row vector `m` of type 1xsize(val,'c') the mean of each column of `val` (ignoring the NANs).

`m=nanmeanf(val,'c')` (or, equivalently, `m=nanmean(val,2)` ) returns in each entry of the column vector `m` of type size(val,'c')x1 the mean of each row of `val` (ignoring the NANs).

In Labostat, NAN values stand for missing values in tables.

# Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmean(x)
m=nanmean(x,1)
m=nanmean(x,2)
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

nanmeanf — mean (ignoring Nan's) with a given frequency.

```
m=nanmean(val,fre)
m=nanmean(val,fre,'r') (or m=nanmean(val,fre,1))
m=nanmean(val,fre,'c') (or m=nanmean(val,fre,2))
```

# Parameters

val
: real or complex vector or matrix

fre
: integer vector or matrix with same dimensions than val

# Description

This function returns in scalar `m` the mean of the values (ignoring the NANs) of a vector or matrix `val`, each counted with a frequency signaled by the corresponding values of the integer vector or matrix `fre` with the same type of `val`.

For a vector or matrix `val`, `m=nanmeanf(val,fre)` or `m=nanmeanf(val,fre,'*')` returns in scalar `m` the mean of all the entries (ignoring the NANs) of `val`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

`m=nanmeanf(val,fre,'r')` (or, equivalently, `m=nanmeanf(val,fre,1)` ) returns in each entry of the row vector `m` of type 1xsize(val,'c') the mean of each column of val (ignoring the NANs), each value counted with the multiplicity indicated by the corresponding value of `fre`.

`m=nanmeanf(val,fre,'c')` (or, equivalently, `m=nanmeanf(val,fre,2)`) returns in each entry of the column vector m of type size(val,'c')x1 the mean of each row of val (ignoring the NANs), each value counted with the multiplicity indicated by the corresponding value of fre.

In Labostat, NAN values stand for missing values in tables.

# Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
fre=[34 12 25;12 23 5]
m=nanmeanf(x,fre)
m=nanmeanf(x,fre,1)
m=nanmeanf(x,fre,2)
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

nanmedian — median of the values of a numerical vector or matrix

```
m=nanmedian(x)
m=nanmedian(x,'r') (or m=nanmedian(x,1))
m=nanmedian(x,'c') (or m=nanmedian(x,2))
```

# Parameters

x
: real or complex vector or matrix

# Description

For a vector or a matrix `x`, `[m]=nanmedian(x)` returns in the vector `m` the median of the values (ignoring the NANs) of vector `x`.

`[m]=nanmedian(x,'r')` (or, equivalently, `[m]=nanmedian(x,1)`) are the rowwise medians. It returns in each position of the row vector `m` the medians of data (ignoring the NANs) in the corresponding column of `x`.

`[m]=nanmedian(x,'c')` (or, equivalently, `[m]=nanmedian(x,2)`) are the columnwise madians. It returns in each position of the column vector `m` the medians of data (ignoring the NANs) in the corresponding row of `x`.

In Labostat, NAN values stand for missing values in tables.

# Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmedian(x)
m=nanmedian(x,1)
m=nanmedian(x,2)
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

nanmin — min (ignoring Nan's)

```
[m,index]=nanmin(x)
[m,index]=nanmin(x,'r')
[m,index]=nanmin(x,'c')
```

# Parameters

x

real or complex vector or matrix

# Description

This function gives for a real or a numerical matrix `x` his largest element `m` (but ignoring the NANs).

For `x`, a numerical vector or matrix, `m=nanmin(x)` returns in scalar `m` the largest element of `x` (ignoring the NANs). The form `[m,index] =nanmin(x,orient)` gives in addition of the value of the largest element of `x` (ignoring the NANs) in scalar `m`, the index of this element in `x`, as a 2-vector.

`m=nanmin(x,'r')` gives in the 1xsize(x,2) matrix `m` the largest elements (ignoring the NANs) of each column of `x`. If the form `[m,index]=nanmin(x,'r')` is used, the elements of the 1xsize(x,2) matrix index are the indexes of the largest elements (ignoring the NANs) of each column of x in the corresponding column.

m=nanmin(x,'c') gives in the size(x,2)x1 matrix m the largest elements (ignoring the NANs) of each row of x. If the form [m,index]=nanmin(x,'c') is used, the elements of the size(x,2)x1 matrix index are the indexes of the largest elements (ignoring the NANs) of each row of x in the corresponding row.

In Labostat, NAN values stand for missing values in tables.

# Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmin(x)
m=nanmin(x,'r')
m=nanmin(x,'c')
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

nanstdev — standard deviation (ignoring the NANs).

```
s=nanstdev(x)
s=nanstdev(x,'r') or m=nanstdev(x,1)
s=nanstdev(x,'c') or m=nanstdev(x,2)
```

# Parameters

x
: real or complex vector or matrix

# Description

This function computes the standard deviation of the values of a vector or matrix `x` (ignoring the NANs).

For a vector or a matrix `x`, `s=nanstdev(x)` returns in the scalar `s` the standard deviation of all the entries of `x` (ignoring the NANs).

`s=nanstdev(x,'r')` (or, equivalently, `s=nanstdev(x,1)` ) is the rowwise standard deviation. It returns in each entry of the row vector `s` the standard deviation of each column of `x` (ignoring the NANs).

`s=nanstdev(x,'c')` (or, equivalently, `s=nanstdev(x,2)`) is the columnwise standard deviation. It returns in each entry of the column vector `s` the standard deviation of each row of `x` (ignoring the NANs).

In Labostat, NAN values stand for missing values in tables.

# Examples

```
x=[0.2113249 0.0002211 0.6653811;
   0.7560439 %nan      0.6283918;
   0.3       0.2       0.5      ];
s=nanstdev(x)
s=nanstdev(x,'r')
s=nanstdev(x,'c')
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

nansum — Sum of values ignoring NAN's

```
s = nansum(x,orient)
```

# Parameters

x
: numerical vector or matrix.

orient
: nothing or '*'. 'r' or 1. 'c' or 2.

s
: Numerical scalar or vector containig the value of the adding operation.

# Description

This function returns in s the sum of the values (ignoring the NAN's) of a numerical vector or matrix x.

For a vector or matrix x, s=nansum(x) (or s=nansum(x,'*')) returns in scalar s the sum of values of all entries (ignoring the NAN's) of a vector or matrix x.

s=nansum(x,'r')(or, equivalently, s=nansum(x,1)) returns in each entry of the row vector s of type 1xsize(x,'c') the sum of each column of x (ignoring the NANs).

s=nansum(x,'c')(or, equivalently, s=nansum(x,2)) returns in each entry of the column vector s of type size(x,'c')x1 the sum of each row of x (ignoring the NANs).

For the last two cases, if a row or column is in whole composed of NAN, the corresponding place of s will contain a NAN.

# Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nansum(x)
m=nansum(x,1)
m=nansum(x,2)
```

# See Also

nancumsum , sum

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. and Wonacott, R.J.; Introductory Statistics, 5th edition, J.Wiley and Sons, 1990.

# Name

nfreq — frequence of the values in a vector or matrix

```
m=nfreq(x)
```

# Parameters

x
: real or complex vector or matrix

# Description

Frequence of the values in a real or complex vector or a real or complex matrix `x`.

For a real or complex vector or a real or complex matrix `x`, `m=freq(x)` returns in the first column of the `size(x,'*')x2` matrix `m` the values of `x` and in the second column of this matrix the frequences of the corresponding values.

Note that the tabul function is more efficient, applies also to vector of strings and returns a sorted `m`.

# Examples

```
x=[2 8 0 3 7 6 8 7 9 1]
m=nfreq(x)
```

# See Also

tabul , dsearch , histplot

# Authors

Carlos Klimann

# Name

pca — Computes principal components analysis with standardized variables

```
[lambda,facpr,comprinc] = pca(x)
```

# Parameters

x
: is a nxp (n individuals, p variables) real matrix. Note that `pca` center and normalize the columns of `x` to produce principal components analysis with standardized variables.

lambda
: is a p x 2 numerical matrix. In the first column we find the eigenvalues of V, where V is the correlation p x p matrix and in the second column are the ratios of the corresponding eigenvalue over the sum of eigenvalues.

facpr
: are the principal factors: eigenvectors of V. Each column is an eigenvector element of the dual of `R^p`.

comprinc
: are the principal components. Each column (c_i=Xu_i) of this n x n matrix is the M-orthogonal projection of individuals onto principal axis. Each one of this columns is a linear combination of the variables x1, ...,xp with maximum variance under condition `u'_i M^(-1) u_i=1`

# Description

This function performs several computations known as "principal component analysis".

The idea behind this method is to represent in an approximative manner a cluster of n individuals in a smaller dimensional subspace. In order to do that, it projects the cluster onto a subspace. The choice of the k-dimensional projection subspace is made in such a way that the distances in the projection have a minimal deformation: we are looking for a k-dimensional subspace such that the squares of the distances in the projection is as big as possible (in fact in a projection, distances can only stretch). In other words, inertia of the projection onto the k dimensional subspace must be maximal.

Warning, the graphical part of the old version of `pca` as been removed. It can now be performed using the show_pca function.

# Examples

```
a=rand(100,10,'n');
[lambda,facpr,comprinc] = pca(a);
show_pca(lambda,facpr)
```

# See Also

show_pca , princomp

# Authors

Carlos Klimann

# Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

# Name

perctl — computation of percentils

```
p=perctl(x,y)
```

# Parameters

x
: real or complex vector or matrix

y
: vector of positif values between 0 and 100.

# Description

Compute the matrix p of percentils (in increasing order, column first) of the real vector or matrix x indicated by the entries of y, the values of entries of y must be positive integers between 0 and 100. p is a matrix whose type is length(y) x 2 and the content of its first column are the percentils values. The contents of its second column are the places of the computed percentiles in the input matrix x.

The minimum or maximum values in x are assigned to percentiles for percent values outside that range.

# Examples

```
x=[6  7 0 7 10 4 2 2 7  1;
   6  0 5 5 5  2 0 6 8  10;
   8  6 4 3 5  9 8 3 4  7;
   1  3 2 7 6  1 1 4 8  2;
   6  3 5 1 6  5 9 9 5  5;
   1  6 4 4 5  4 0 8 1  8;
   7  1 3 7 8  0 2 8 10 8;
   3  6 1 9 8  5 5 3 2  1;
   5  7 6 2 10 8 7 4 0  8;
   10 3 3 4 8  6 9 4 8  3]
y=[10 20 30]
p=perctl(x,y)
```

# Authors

Carlos Klimann

# Bibliography

HYNDMAN,Rob J. and FAN Yanan, Sample Quantiles in Statistical Packages, The American Statistician, Nov.1996, Vol 50, No.4

# Name

princomp — Principal components analysis

```
[facpr,comprinc,lambda,tsquare] = princomp(x,eco)
```

# Parameters

x
: is a n-by-p (n individuals, p variables) real matrix.

eco
: a boolean, use to allow economy size singular value decomposition.

facpr
: A p-by-p matrix. It contains the principal factors: eigenvectors of the correlation matrix V.

comprinc
: a n-by-p matrix. It contains the principal components. Each column of this matrix is the M-orthogonal projection of individuals onto principal axis. Each one of this columns is a linear combination of the variables x1, ...,xp with maximum variance under condition `u'_i M^(-1) u_i=1`

lambda
: is a p column vector. It contains the eigenvalues of V, where V is the correlation matrix.

tsquare
: a n column vector. It contains the Hotelling's T^2 statistic for each data point.

# Description

This function performs "principal component analysis" on the n-by-p data matrix x.

The idea behind this method is to represent in an approximative manner a cluster of n individuals in a smaller dimensional subspace. In order to do that, it projects the cluster onto a subspace. The choice of the k-dimensional projection subspace is made in such a way that the distances in the projection have a minimal deformation: we are looking for a k-dimensional subspace such that the squares of the distances in the projection is as big as possible (in fact in a projection, distances can only stretch). In other words, inertia of the projection onto the k dimensional subspace must be maximal.

To compute principal component analysis with standardized variables may use `princomp(wcenter(x,1))` or use the pca function.

# Examples

```
a=rand(100,10,'n');
[facpr,comprinc,lambda,tsquare] = princomp(a);
```

# See Also

wcenter , pca

# Authors

Carlos Klimann

# Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

# Name

quart — computation of quartiles

```
s=quart(x)
s=quart(x,'r') or m=quart(x,1)
s=quart(x,'c') or m=quart(x,2)
```

# Parameters

x
: real or complex vector or matrix

# Description

For a vector or a matrix x, [q]=quart(x,y) returns in the vector q the quartiles of x. [q]=quart(x,'r') (or, equivalently, [q]=quart(x,1)) are the rowwise percentiles. It returns in each column of the matrix q the quartiles of data in the corresponding column of x.

[q]=quart(x,'c') (or, equivalently, [q]=quart(x,2)) are the columnwise quartiles. It returns in each row of the matrix q the quartiles of data in the corresponding row of x.

# Examples

```
x=[6 7 0 7 10 4 2 2 7 1;
   6 0 5 5 5 2 0 6 8 10;
   8 6 4 3 5 9 8 3 4 7;
   1 3 2 7 6 1 1 4 8 2;
   6 3 5 1 6 5 9 9 5 5;
   1 6 4 4 5 4 0 8 1 8;
   7 1 3 7 8 0 2 8 10 8;
   3 6 1 9 8 5 5 3 2 1;
   5 7 6 2 10 8 7 4 0 8;
   10 3 3 4 8 6 9 4 8 3]
q=quart(x)
q=quart(x,'r')
q=quart(x,'c')
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

regress — regression coefficients of two variables

```
coefs=regress(x,y)
```

# Parameters

x,y
: real or complex vector

# Description

This function computes the regresion coefficients of two variables `x` and `y`, both numerical vectors of same number of elements `n`. `coefs=[a b]` be a 1x2 matrix such that `Y=a+bX` will be the equation of the ordinary least square approximation to our data.

# References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

# Examples

```
x=[0.5608486 0.6623569 0.7263507 0.1985144 0.5442573 0.2320748 0.2312237]
y=[0.3616361 0.2922267 0.5664249 0.4826472 0.3321719 0.5935095 0.5015342]
coefs=regress(x,y)
```

# See Also

covar

# Authors

Carlos Klimann

# Name

sample — Sampling with replacement

```
s = sample(n,X,orient)
```

# Parameters

n
: positive integer (size of sample)

X
: matrix. Samples will be extracted from this matrix.

orient
: Optional parameter. Admissible values are 1, 2, 'r' or 'c'

s
: vector or matrix containing sample

# Description

This function gives a vector (or matrix) nx1. It contains a random sample of n extractions, with replacement, from the matrix X.

s=sample(n,X) (or s=sample(n,X,'*')) returns a vector s whose values are a random sample of n values from X, extracted with replacement, from X .

s=sample(n,X,'r') (or, equivalently, s=sample(n,X,1)) returns a matrix of type size(X,'r')xn. It contains a random sample of n rows, extracted with replacement, from the rows of X.

s=sample(n,X,'c') (or, equivalently, s=sample(n,X,2)) returns a matrix of type nxsize(X,'c'). It contains a random sample of n columns, extracted with replacement from the columns of X.

# Examples

```
X=['a' 'dd' 'arreu'; 'ber' 'car' 'zon']
s=sample(25,X)
s=sample(25,X,'r')
s=sample(25,X,'c')
```

# See Also

samplef , samwr

# Authors

Carlos Klimann

# Name

samplef — sample with replacement from a population and frequences of his values.

```
s = samplef(n,X,f,orient)
```

# Parameters

n
: positive integer (size of sample)

X
: matrix. Samples will be extracted from this matrix

f
: positive integer matrix with same type than X. It indicates frequences of corresponding values of X.

orient
: Optional parameter. Admissible values are 1, 2, 'r' or 'c'

s
: vector or matrix containing sample

# Description

This function gives s, a vector of lenght n. It contains a sample of n extractions, with replacement, from the vector (or matrix) X, each element counted with the frequence given by the corresponding value in vector f.

s=samplef(n,X,f) (or s=samplef(n,X,f,'*')) returns a vector s whose values are a random sample of n values from X, each value with a probability to be sampled proportional to the corresponding value of f, extracted with replacement, from X. f must have same lenght than X.

s=samplef(n,X,f,'r') (or, equivalently, s=samplef(n,X,f,1)) returns a matrix of type size(X,'r')xn. It contains a random sample of n rows from X, each row with a probability to be sampled proportional to the corresponding value of f, extracted with replacement, from the rows of X. The lenght of f must be equal to the number of rows of X.

s=samplef(n,X,f,'c') (or, equivalently, s=samplef(n,X,f,2)) returns a matrix of type nxsize(X,'c'). It contains a random sample of n columns from X, each column with a probability to be sampled proportional to the corresponding value of f, extracted with replacement, from the columns of X. The lenght of f must be equal to the number of columns of X.

# Examples

```
a=[3 7 9;22 4 2]
f1=[10 1 1 1 1 1]
f2=[1 ; 15]
f3=[10 1 1]
s=samplef(15,a,f1)
s=samplef(15,a,f2,'r')
s=samplef(15,a,f3,'c')
```

# See Also

sample , samwr

# Authors

Carlos Klimann

# Name

samwr — Sampling without replacement

```
s = samwr(sizam,numsamp,X)
```

# Parameters

sizam
: integer. Size of a sample. It must be less or equal than size of X.

numsamp
: integer. Number of samples to be extracted.

X
: column vector. It contains the population.

s
: matrix of type sizsam x numsamp. It contains numsamp random samples (the columns) each of sizam (size(X,'*')) extractions, without replacement, from the column vector X.

# Description

Gives samples without replacement from a column vector.

# Examples

```
a=[0.33 1.24 2.1 1.03]
s=samwr(4,12,a)
```

# See Also

sample , samplef

# Authors

Carlos Klimann

# Name

show_pca — Visualization of principal components analysis results

```
show_pca(lambda,facpr,N)
```

# Parameters

lambda
: is a p x 2 numerical matrix. In the first column we find the eigenvalues of V, where V is the correlation p x p matrix and in the second column are the ratios of the corresponding eigenvalue over the sum of eigenvalues.

facpr
: are the principal factors: eigenvectors of V. Each column is an eigenvector element of the dual of `R^p`.

N
: Is a 2x1 integer vector. Its coefficients point to the eigenvectors corresponding to the eigenvalues of the correlation matrix `p` by `p` ordered by decreasing values of eigenvalues. If `N`. is missing, we suppose `N=[1 2]`..

# Description

This function visualize the pca results.

# Examples

```
a=rand(100,10,'n');
[lambda,facpr,comprinc] = pca(a);
show_pca(lambda,facpr)
```

# See Also

pca , princomp

# Authors

Carlos Klimann

# Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

# Name

st_deviation — standard deviation (row or column-wise) of vector/matrix entries
stdev — standard deviation (row or column-wise) of vector/matrix entries

```
y=st_deviation(x)
y=st_deviation(x,'r')
y=st_deviation(x,'c')
y=stdev(x)
y=stdev(x,'r')
y=stdev(x,'c')
```

# Parameters

x
: real vector or matrix

y
: scalar or vector

# Description

st_deviation computes the "sample" standard deviation, that is, it is normalized by N-1, where N is the sequence length.

For a vector or a matrix `x`, `y=st_deviation(x)` returns in the scalar `y` the standard deviation of all the entries of `x`.

`y=st_deviation(x,'r')` (or, equivalently, `y=st_deviation(x,1)`) is the rowwise standard deviation. It returns in each entry of the column vector `y` the standard deviation of each row of `x`.

`y=st_deviation(x,'c')` (or, equivalently, `y=st_deviation(x,2)`) is the columnwise st_deviation. It returns in each entry of the row vector `y` the standard deviation of each column of `x`.

# Examples

```
A=[1,2,10;7,7.1,7.01];
st_deviation(A)
st_deviation(A,'r')
st_deviation(A,'c')
```

# See Also

sum , median , mean , nanstdev , stdevf

# Name

stdevf — standard deviation

```
s=stdevf(x,fre)
s=stdevf(x,fre,'r') or s=stdevf(x,fre,1)
s=stdevf(x,fre,'c') or s=stdevf(x,fre,2)
```

# Parameters

x
: real or complex vector or matrix

# Description

This function computes the standard deviation of the values of a vector or matrix `x`, each of them counted with a frequency given by the corresponding values of the integer vector or matrix `fre` who has the same type of `x`.

For a vector or matrix `x`, s=stdevf(x,fre) (or `s=stdevf(x,fre,'*')` returns in scalar `s` the standard deviation of all the entries of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

`s=stdevf(x,fre,'r')` (or, equivalently, `s=stdevf(x,fre,1)`) returns in each entry of the row vector `s` of type 1xsize(x,'c') the standard deviation of each column of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

`s=stdevf(x,fre,'c')` (or, equivalently, `s=stdevf(x,fre,2)`) returns in each entry of the column vector `s` of type size(x,'c')x1 the standard deviation of each row of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

# Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.9546254 0.6283918]
fre=[1 2 3;3 4 3]
m=stdevf(x,fre)
m=stdevf(x,fre,'r')
m=stdevf(x,fre,'c')
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

strange — range

```
[r]=strange(x)
[r]=strange(x,'r') (or, equivalently, [r]=strange(x,1))
[r]=strange(x,'c') (or, equivalently, [r]=strange(x,2))
```

# Parameters

x
: real or complex vector or matrix

# Description

The range is the distance between the largest and smaller value, [r]=strange(x) computes the range of vector or matrix x.

[r]=strange(x,'r') (or equivalently [r]=strange(x,1)) give a row vector with the range of each column.

[r]=strange(x,'c') (or equivalently [r]=strange(x,2)) give a column vector with the range of each row.

# References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

# Authors

Carlos klimann

# Name

tabul — frequency of values of a matrix or vector

```
[m]=tabul(X [,order])
```

# Parameters

X
: vector or matrix (of real or complex numbers or strings)

order
: (optionnal) a character equal to "d" or "i" (default value "d")

m
: a 2 columns matrix (if `X` is a numerical vector or matrix) or a list with 2 members (if `X` is a string vector or matrix).

# Description

This function computes the frequency of values of the components of a vector or matrix `X` of numbers or string characters :

if X is a numerical vector or matrix
: then `m` is a two column matrix who contains in the first column the distinct values of `X` and in the other column the number of occurrences of those values (m(i,2) is the number of occurrences of m(i,1)).

if X is a string vector or matrix
: then `m` is a list whose first member is a string (column) vector composed with the distinct values of `X` and the second member is a (column) vector whose components are the number of occurrences of those values ( m(i)(2) is the number of occurrences of the string m(i)(1) ).

The optional parameter `order` must be "d" or "i" (by default order="d") and gives the order (decreasing or increasing) the distinct values of X will be sorted.

# Examples

```
// first example
X = [2 8 0 3 7 6 8 7 9 1 6 7 7 2 5 2 2 2 9 7]
m1 = tabul(X)
m2 = tabul(X, "i")

// second example
X = ["ba" "baba" "a" "A" "AA" "a" "aa" "aa" "aa" "A" "ba"]
m = tabul(X,"i")

// third example
n = 50000;
X = grand(n,1,"bin",70,0.5);
m = tabul(X,"i");
clf()
plot2d3(m(:,1), m(:,2)/n)
xtitle("empirical probabilities of B(70,0.5)")

// last example : computes the occurrences of words of the scilab license
```

```
text = read(SCI+"/license.txt",-1,1,"(A)"); // read the scilab license
bigstr = strcat(text," ");                  // put all the lines in a big strin
sep =  [" " "," "." ";" "*" ":" "-" """"];  // words separators
words = tokens(bigstr, sep);                // cut the big string into words
m = tabul(words);                           // computes occurrences of each wor
[occ , p] = gsort(m(2));                     // sort by decreasing frequencies
results = [m(1)(p) string(occ)]             // display result
```

# See Also

dsearch , histplot

# Authors

Carlos Klimann (original author)
J.S. Giet and B. Pincon (new version)

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

thrownan — eliminates nan values

```
[nonan,numb]=thrownan(x)
```

# Parameters

x
: real or complex vector or matrix

# Description

This function returns in vector `nonan` the values (ignoring the NANs) of a vector or matrix `x` and in the corresponding places of vector `numb` the indexes of the value.

For a vector or matrix `x`, `[nonan,numb]=thrownan(x)` considers `x`, whatever his dimensions are, like a vector (columns first).

In Labostat, NAN values stand for missing values in tables.

# Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
[nonan numb]=thrownan(x)
```

# Authors

Carlos Klimann

# Name

trimmean — trimmed mean of a vector or a matrix

```
m=trimmean(x[,discard [,flag [,verbose]]])
```

# Parameters

x
: real or complex vector or matrix

discard
: Optional real value between 0 and 100 representing the part of the data to discard. It discard is not in the [0,100] range, an error is generated. Default value for discard=50.

flag
: Optional string or real parameter which controls the behaviour when x is a matrix. Available values for flag are : "all", 1, 2, r or c (default is flag="all"). The two values flag=r and flag=1 are equivalent. The two values flag=c and flag=2 are equivalent.

verbose
: Optional integer. If set to 1, then enables verbose logging. Default is 0.

# Description

A trimmed mean is calculated by discarding a certain percentage of the lowest and the highest scores and then computing the mean of the remaining scores. For example, a mean trimmed 50% is computed by discarding the lower and higher 25% of the scores and taking the mean of the remaining scores.

The median is the mean trimmed 100% and the arithmetic mean is the mean trimmed 0%.

A trimmed mean is obviously less susceptible to the effects of extreme scores than is the arithmetic mean. It is therefore less susceptible to sampling fluctuation than the mean for extremely skewed distributions. The efficiency of a statistic is the degree to which the statistic is stable from sample to sample. That is, the less subject to sampling fluctuation a statistic is, the more efficient it is. The efficiency of statistics is measured relative to the efficiency of other statistics and is therefore often called the relative efficiency. If statistic A has a smaller standard error than statistic B, then statistic A is more efficient than statistic B. The relative efficiency of two statistics may depend on the distribution involved. For instance, the mean is more efficient than the median for normal distributions but not for some extremely skewed distributions. The efficiency of a statistic can also be thought of as the precision of the estimate: The more efficient the statistic, the more precise the statistic is as an estimator of the parameter. The trimmed mean is less efficient than the mean for normal distributions.

For a vector or matrix `x`, `t=trimmean(x,discard)` returns in scalar `t` the mean of all the entries of `x`, after discarding `discard/2` highest values and `discard/2` lowest values.

`t=trimmean(x,discard,'r')` (or, equivalently, `t=trimmean(x,discard,1)`) returns in each entry of the row vector `t` the trimmed mean of each column of `x`.

`t=trimmean(x,discard,'c')` (or, equivalently, `t=trimmean(x,discard,2)`) returns in each entry of the column vector `t` the trimmed mean of each row of `x`.

This function computes the trimmed mean of a vector or matrix `x`.

For a vector or matrix `x`, `m=trimmean(x,discard)` returns in scalar `m` the trimmed mean of all the entries of `x`.

`m=trimmean(x,'r')` (or, equivalently, `m=trimmean(x,1)` ) returns in each entry of the row vector `m` the trimmed mean of each column of `x`.

`m=trimmean(x,'c')` (or, equivalently, `m=trimmean(x,2)` ) returns in each entry of the column vector `m` the trimmed mean of each row of `x`.

# Example with x as vector

In the following example, one computes the trimmed mean of one data vector, with the default discard value equals to 50 and verbose logging. The data is made of 9 entries. The algorithms sorts the vector and keeps only indices from 3 to 7, skipping indices 1, 2, 8 and 9. The value 4000, which is much larger than the others is not taken into account. The computed trimmed mean is therefore 50.

```
data = [10, 20, 30, 40, 50, 60, 70, 80, 4000];
computed = trimmean(data,verbose=1);
```

# Example with x as matrix

In the following example, one computes the trimmed mean of one data matrix. The chosen discard value is 50. The orientation is "r", which means that the data is sorted row by row. For each column of the matrix, one computes a trimmed mean. The trimmed mean is the line vector [25 25 25 25].

```
data = [10 10 10 10
        20 20 20 20
        30 30 30 30
        4000 4000 4000 4000];
computed = trimmean(data,50,orien="r");
```

# References

Luis Angel Garcia-Escudero and Alfonso Gordaliza, Robustness Properties of Means and Trimmed Means, JASA, Volume 94, Number 447, Sept 1999, pp956-969

Trimmed Mean, http://davidmlane.com/hyperstat/A11971.html

# Authors

Carlos Klimann

# Name

variance — variance of the values of a vector or matrix

```
s=variance(x[,orien[,w]])
s=variance(x,'r') or m=variance(x,1)
s=variance(x,'c') or m=variance(x,2)
```

# Parameters

x
: real or complex vector or matrix

orien
: the orientation of the computation. Valid values or the orien parameter are 1, "r", 2 and "c".

w
: w : type of normalization to use. Valid values are 0 and 1. This depends on the number of columns of x (if orien = 1 is chosen), the number of rows (if orien = 2 is chosen). If w = 0, normalizes with m-1, provides the best unbiased estimator of the variance (this is the default). If w = 1, normalizes with m, this provides the second moment around the mean. If no orien option is given, the normalization is done with n * m - 1, where n * m is the total number of elements in the matrix.

# Description

This function computes the variance of the values of a vector or matrix `x`.

For a vector or a matrix `x`, `s=variance(x)` returns in the scalar `s` the variance of all the entries of `x`.

`s=variance(x,'r')` (or, equivalently, `s=variance(x,1)`) is the rowwise variance. It returns in each entry of the row vector s the variance of each column of `x`. The generalized formulae is used, which manages complex values.

`s=variance(x,'c')` (or, equivalently, `s=variance(x,2)`) is the columnwise standard deviation. It returns in each entry of the column vector `s` the variance of each row of `x`. The generalized formulae is used, which manages complex values.

# Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.4453586 0.6283918]
s=variance(x)
s=variance(x,'r')
s=variance(x,'c')
```

# See Also

mtlb_var

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Name

variancef — standard deviation of the values of a vector or matrix

```
s=variancef(x,fre)
s=variancef(x,fre,'r') or s=variancef(x,fre,1)
s=variancef(x,fre,'c') or s=variancef(x,fre,2)
```

# Parameters

x
: real or complex vector or matrix

# Description

This function computes the variance of the values of a vector or matrix `x`, each of them counted with a frequency signaled by the corresponding values of the integer vector or matrix `fre` with the same type of `x`.

For a vector or matrix `x`, `s=variancef(x,fre)` (or `s=variancef(x,fre,'*')`) returns in scalar `s` the variance of all the entries of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

`s=variancef(x,fre,'r')`(or, equivalently, `s=variancef(x,fre,1)`) returns in each entry of the row vector `s` of type 1xsize(x,'c') the variance of each column of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

`s=variancef(x,fre,'c')`(or, equivalently, `s=variancef(x,fre,2)`) returns in each entry of the column vector `s` of type size(x,'c') x1 the variance of each row of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

# Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.9546254 0.6283918]
fre=[1 2 3;3 4 3]
m=variancef(x,fre)
m=variancef(x,fre,'r')
m=variancef(x,fre,'c')
```

# Authors

Carlos Klimann

# Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

# Parte XXXV. ARnoldi PACKage

# Name

dnaupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute approximations to a few eigenpairs of a real linear operator

```
[IDO,RESID,V,IPARAM,IPNTR,WORKD,WORKL,INFO] = dnaupd(ID0,BMAT,N,WHICH,NEV,TOL,R
```

# Parameters

ID0

Integer. (INPUT/OUTPUT) Reverse communication flag. IDO must be zero on the first call to dnaupd. IDO will be set internally to indicate the type of operation to be performed. Control is then given back to the calling routine which has the responsibility to carry out the requested operation and call dnaupd with the result. The operand is given in WORKD(IPNTR(1)), the result must be put in WORKD(IPNTR(2)).

- IDO = 0: first call to the reverse communication interface
- IDO = -1: compute Y = OP * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. This is for the initialization phase to force the starting vector into the range of OP.
- IDO = 1: compute Y = OP * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. In mode 3 and 4, the vector B * X is already available in WORKD(ipntr(3)). It does not need to be recomputed in forming OP * X.
- IDO = 2: compute Y = B * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y.
- IDO = 3: compute the IPARAM(8) real and imaginary parts of the shifts where INPTR(14) is the pointer into WORKL for placing the shifts. See Remark 5 below.
- IDO = 99: done

BMAT

Character, specifies the type of the matrix B that defines the semi-inner product for the operator OP.

- 'I' - standard eigenvalue problem A*x = lambda*x
- 'G' - generalized eigenvalue problem A*x = lambda*B*x

N

Integer, dimension of the eigenproblem.

WHICH

string of length 2, Specify which of the Ritz values of OP to compute.

- 'LM' - want the NEV eigenvalues of largest magnitude.
- 'SM' - want the NEV eigenvalues of smallest magnitude.
- 'LR' - want the NEV eigenvalues of largest real part.
- 'SR' - want the NEV eigenvalues of smallest real part.
- 'LI' - want the NEV eigenvalues of largest imaginary part.
- 'SI' - want the NEV eigenvalues of smallest imaginary part.

NEV

Integer, number of eigenvalues of OP to be computed. 0 < NEV < N-1.

TOL
: scalar. Stopping criterion: the relative accuracy of the Ritz value is considered acceptable if BOUNDS(I) <= TOL*ABS(RITZ(I)). If TOL <= 0. is passed the machine precision is set.

RESID
: array of length N (INPUT/OUTPUT)

On INPUT: If INFO==0, a random initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

On OUTPUT: RESID contains the final residual vector.

NCV
: Integer, number of columns of the matrix V. NCV must satisfy the two inequalities 2 <= NCV-NEV and NCV <= N.

This will indicate how many Arnoldi vectors are generated at each iteration.

After the startup phase in which NEV Arnoldi vectors are generated, the algorithm generates approximately NCV-NEV Arnoldi vectors at each subsequent update iteration. Most of the cost in generating each Arnoldi vector is in the matrix-vector operation OP*x.

NOTE: 2 <= NCV-NEV in order that complex conjugate pairs of Ritz values are kept together. (See remark 4 below)

V
: N by NCV array. Contains the final set of Arnoldi basis vectors.

IPARAM
: array of length 11. (INPUT/OUTPUT)

- IPARAM(1) = ISHIFT: method for selecting the implicit shifts. The shifts selected at each iteration are used to restart the Arnoldi iteration in an implicit fashion.
  - ISHIFT = 0: the shifts are provided by the user via reverse communication. The real and imaginary parts of the NCV eigenvalues of the Hessenberg matrix H are returned in the part of the WORKL array corresponding to RITZR and RITZI. See remark 5 below.
  - ISHIFT = 1: exact shifts with respect to the current Hessenberg matrix H. This is equivalent to restarting the iteration with a starting vector that is a linear combination of approximate Schur vectors associated with the "wanted" Ritz values.
- IPARAM(2) = LEVEC No longer referenced.
- IPARAM(3) = MXITER

  On INPUT: maximum number of Arnoldi update iterations allowed.

  On OUTPUT: actual number of Arnoldi update iterations taken.
- IPARAM(4) = NB: blocksize to be used in the recurrence. The code currently works only for NB = 1.
- IPARAM(5) = NCONV: number of "converged" Ritz values. This represents the number of Ritz values that satisfy the convergence criterion.
- IPARAM(6) = IUPD No longer referenced. Implicit restarting is ALWAYS used.
- IPARAM(7) = MODE On INPUT determines what type of eigenproblem is being solved. Must be 1,2,3,4; See under Description of dnaupd for the five modes available.
- IPARAM(8) = NP When ido = 3 and the user provides shifts through reverse communication (IPARAM(1)=0), dnaupd returns NP, the number of shifts the user is to provide.

0 < NP <= NCV-NEV. See Remark 5 below.

- IPARAM(9) = NUMOP,

  IPARAM(10) = NUMOPB,

  IPARAM(11) = NUMREO,

  On OUTPUT: NUMOP = total number of OP*x operations, NUMOPB = total number of B*x operations if BMAT='G', NUMREO = total number of steps of re-orthogonalization.

IPNTR
array of length 14. Pointer to mark the starting locations in the WORKD and WORKL arrays for matrices/vectors used by the Arnoldi iteration.

- IPNTR(1): pointer to the current operand vector X in WORKD.
- IPNTR(2): pointer to the current result vector Y in WORKD.
- IPNTR(3): pointer to the vector B * X in WORKD when used in the shift-and-invert mode.
- IPNTR(4): pointer to the next available location in WORKL that is untouched by the program.
- IPNTR(5): pointer to the NCV by NCV upper Hessenberg matrix H in WORKL.
- IPNTR(6): pointer to the real part of the ritz value array RITZR in WORKL.
- IPNTR(7): pointer to the imaginary part of the ritz value array RITZI in WORKL.
- IPNTR(8): pointer to the Ritz estimates in array WORKL associated with the Ritz values located in RITZR and RITZI in WORKL.
- IPNTR(14): pointer to the NP shifts in WORKL. See Remark 5 below.

Note: IPNTR(9:13) is only referenced by dneupd . See Remark 2.

- IPNTR(9): pointer to the real part of the NCV RITZ values of the original system.
- IPNTR(10): pointer to the imaginary part of the NCV RITZ values of the original system.
- IPNTR(11): pointer to the NCV corresponding error bounds.
- IPNTR(12):pointer to the NCV by NCV upper quasi-triangular Schur matrix for H.
- IPNTR(13): pointer to the NCV by NCV matrix of eigenvectors of the upper Hessenberg matrix H. Only referenced by dneupd if RVEC = 1 See Remark 2 below.

WORKD
Double precision work array of length 3*N. (REVERSE COMMUNICATION)

Distributed array to be used in the basic Arnoldi iteration for reverse communication. The user should not use WORKD as temporary workspace during the iteration. Upon termination WORKD(1:N) contains B*RESID(1:N). If an invariant subspace associated with the converged Ritz values is desired, see remark 2 below, subroutine dneupd uses this output. See Data Distribution Note below.

WORKL
work array of length at least 3*NCV**2 + 6*NCV. (OUTPUT/WORKSPACE) Private (replicated) array on each PE or array allocated on the front end. See Data Distribution Note below.

INFO
Integer. (INPUT/OUTPUT)

If INFO == 0, a randomly initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

Error flag on output.

- 0: Normal exit.
- 1: Maximum number of iterations taken. All possible eigenvalues of OP has been found. IPARAM(5) returns the number of wanted converged Ritz values.
- 2: No longer an informational error. Deprecated starting with release 2 of ARPACK.
- 3: No shifts could be applied during a cycle of the Implicitly restarted Arnoldi iteration. One possibility is to increase the size of NCV relative to NEV. See remark 4 below.
- -1: N must be positive.
- -2: NEV must be positive.
- -3: NCV-NEV >= 2 and less than or equal to N.
- -4: The maximum number of Arnoldi update iterations allowed must be greater than zero.
- -5: WHICH must be one of 'LM', 'SM', 'LR', 'SR', 'LI', 'SI'
- -6: BMAT must be one of 'I' or 'G'.
- -7: Length of private work array WORKL is not sufficient.
- -8: Error return from LAPACK eigenvalue calculation;
- -9: Starting vector is zero.
- -10: IPARAM(7) must be 1,2,3,4.
- -11: IPARAM(7) = 1 and BMAT = 'G' are incompatable.
- -12: IPARAM(1) must be equal to 0 or 1.
- -9999: Could not build an Arnoldi factorization. IPARAM(5) returns the size of the current Arnoldi factorization. The user is advised to check that enough workspace and array storage has been allocated.

# Description

Reverse communication interface for the Implicitly Restarted Arnoldi iteration. This subroutine computes approximations to a few eigenpairs of a linear operator "OP" with respect to a semi-inner product defined by a symmetric positive semi-definite real matrix B. B may be the identity matrix. NOTE: If the linear operator "OP" is real and symmetric with respect to the real positive semi-definite symmetric matrix B, i.e. B*OP = (OP`)*B, then subroutine dsaupd should be used instead.

The computed approximate eigenvalues are called Ritz values and the corresponding approximate eigenvectors are called Ritz vectors.

dnaupd is usually called iteratively to solve one of the following problems:

- Mode 1: A*x = lambda*x. `OP = A , B = I`.
- Mode 2: A*x = lambda*M*x, M symmetric positive definite `OP = inv[M]*A, B = M`. (If M can be factored see remark 3 below)
- Mode 3: A*x = lambda*M*x, M symmetric positive semi-definite. `OP = Real_Part{ inv[A - sigma*M]*M }, B = M`. shift-and-invert mode (in real arithmetic)

  If `OP*x = amu*x`, then

  `amu = 1/2 * [ 1/(lambda-sigma) + 1/(lambda-conjg(sigma))]`.

Note: If sigma is real, i.e. imaginary part of sigma is zero; `Real_Part{ inv[A - sigma*M]*M } == inv[A - sigma*M]*M` `amu == 1/(lambda-sigma)`.

- Mode 4: A*x = lambda*M*x, M symmetric semi-definite `OP = Imaginary_Part{ inv[A - sigma*M]*M } , B = M`. shift-and-invert mode (in real arithmetic)

  If `OP*x = amu*x`, then `amu = 1/2i * [ 1/(lambda-sigma) - 1/(lambda-conjg(sigma)) ]`.

Both mode 3 and 4 give the same enhancement to eigenvalues close to the (complex) shift sigma. However, as lambda goes to infinity, the operator OP in mode 4 dampens the eigenvalues more strongly than does OP defined in mode 3.

NOTE: The action of w <- inv[A - sigma*M]*v or w <- inv[M]*v should be accomplished either by a direct method using a sparse matrix factorization and solving `[A - sigma*M]*w = v` or `M*w = v`, or through an iterative method for solving these systems. If an iterative method is used, the convergence test must be more stringent than the accuracy requirements for the eigenvalue approximations.

# Example

```
// The following sets dimensions for this problem.

nx    = 10;

nev   = 3;
ncv   = 6;
bmat  = 'I';
which = 'LM';

maxiter = 10;

// Local Arrays

iparam  = zeros(11,1);
ipntr   = zeros(14,1);
_select = zeros(ncv,1);
d       = zeros(nev+1,1);
resid   = zeros(nx,1);
v       = zeros(nx,ncv);
workd   = zeros(3*nx+1,1);
workev  = zeros(3*ncv,1);
workl   = zeros(3*ncv*ncv+6*ncv,1);

// Build the test matrix

A            = diag(10*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(-6*ones(nx-1,1));

tol    = 0;
ido    = 0;

ishfts = 1;
maxitr = 300;
mode1  = 1;

iparam(1) = ishfts;
```

```
iparam(3) = maxitr;
iparam(7) = mode1;

sigmar = 0; // the real part of the shift
sigmai = 0; // the imaginary part of the shift

// M A I N   L O O P (Reverse communication)

iter = 0;
while(iter<maxiter)
  info_dnaupd = 0;
  iter = iter + 1;
  // Repeatedly call the routine DNAUPD and take actions indicated by parameter
  // either convergence is indicated or maxitr has been exceeded.

  [ido,resid,v,iparam,ipntr,workd,workl,info_dnaupd] = dnaupd(ido,bmat,nx,which

  if (ido==99) then
    // BE CAREFUL: DON'T CALL dneupd IF ido == 99 !!
    break;
  end

  if (ido==-1 | ido==1) then
    // Perform matrix vector multiplication
    workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
    // L O O P   B A C K to call DNAUPD again.
    continue
  end

  if (info_dnaupd < 0) then
    printf('\nError with dnaupd, info = %d\n',info_dnaupd);
    printf('Check the documentation of dnaupd\n\n');
  else
    // Post-Process using DNEUPD.

    rvec    = 1;
    howmany = 'A';

    info_dneupd = 0;

    [d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_dneupd] = dneupd(rvec,how
                                                                      bmat,nx,
                                                                      iparam,i


    if (info_dneupd~=0) then
      printf('\nError with dneupd, info = %d\n', info_dneupd);
      printf('Check the documentation of dneupd.\n\n');
    end

    // Print additional convergence information.
    if (info_dneupd==1) then
      printf('\nMaximum number of iterations reached.\n\n');
    elseif (info_dneupd==3) then
      printf('\nNo shifts could be applied during implicit Arnoldi update, try
    end
  end
end
```

```
printf('\nDNSIMP\n');
printf('======\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam
printf('The number of OP*x is %d\n', iparam(9));
printf('The convergence criterion is %d\n', tol);
```

# Remarks

1. The computed Ritz values are approximate eigenvalues of OP. The selection of WHICH should be made with this in mind when Mode = 3 and 4. After convergence, approximate eigenvalues of the original problem may be obtained with the ARPACK subroutine dneupd.

2. If a basis for the invariant subspace corresponding to the converged Ritz values is needed, the user must call dneupd immediately following completion of dnaupd. This is new starting with release 2 of ARPACK.

3. If M can be factored into a Cholesky factorization M = LL` then Mode = 2 should not be selected. Instead one should use Mode = 1 with OP = inv(L)*A*inv(L`). Appropriate triangular linear systems should be solved with L and L` rather than computing inverses. After convergence, an approximate eigenvector z of the original problem is recovered by solving L`z = x where x is a Ritz vector of OP.

4. At present there is no a-priori analysis to guide the selection of NCV relative to NEV. The only formal requrement is that NCV > NEV + 2. However, it is recommended that NCV >= 2*NEV+1. If many problems of the same type are to be solved, one should experiment with increasing NCV while keeping NEV fixed for a given test problem. This will usually decrease the required number of OP*x operations but it also increases the work and storage required to maintain the orthogonal basis vectors. The optimal "cross-over" with respect to CPU time is problem dependent and must be determined empirically. See Chapter 8 of Reference 2 for further information.

5. When IPARAM(1) = 0, and IDO = 3, the user needs to provide the NP = IPARAM(8) real and imaginary parts of the shifts in locations

```
real part                   imaginary part
------------------------    --------------
1   WORKL(IPNTR(14))            WORKL(IPNTR(14)+NP)
2   WORKL(IPNTR(14)+1)          WORKL(IPNTR(14)+NP+1)
.                           .
.                           .
.                           .
NP  WORKL(IPNTR(14)+NP-1)       WORKL(IPNTR(14)+2*NP-1).
```

Only complex conjugate pairs of shifts may be applied and the pairs must be placed in consecutive locations. The real part of the eigenvalues of the current upper Hessenberg matrix are located in WORKL(IPNTR(6)) through WORKL(IPNTR(6)+NCV-1) and the imaginary part in WORKL(IPNTR(7)) through WORKL(IPNTR(7)+NCV-1). They are ordered according to the order defined by WHICH. The complex conjugate pairs are kept together and the associated Ritz estimates are located in WORKL(IPNTR(8)), WORKL(IPNTR(8)+1), ... , WORKL(IPNTR(8)+NCV-1).

# See Also

dsaupd

# Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

# Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.

2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.

3. B.N. Parlett, "The Symmetric Eigenvalue Problem". Prentice-Hall, 1980.

4. B.N. Parlett, B. Nour-Omid, "Towards a Black Box Lanczos Program", Computer Physics Communications, 53 (1989), pp 169-179.

5. B. Nour-Omid, B.N. Parlett, T. Ericson, P.S. Jensen, "How to Implement the Spectral Transformation", Math. Comp., 48 (1987), pp 663-673.

6. R.G. Grimes, J.G. Lewis and H.D. Simon, "A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems", SIAM J. Matr. Anal. Apps., January (1993).

7. L. Reichel, W.B. Gragg, "Algorithm 686: FORTRAN Subroutines for Updating the QR decomposition", ACM TOMS, December 1990, Volume 16 Number 4, pp 369-377.

8. R.B. Lehoucq, D.C. Sorensen, "Implementation of Some Spectral Transformations in a k-Step Arnoldi Method". In Preparation.

# Used Functions

Based on ARPACK routine dnaupd

# Name

dneupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute the converged approximations to eigenvalues of A*z = lambda*B*z approximations to a few eigenpairs of a real linear operator

```
[Dr,Di,Z,RESID,V,IPARAM,IPNTR,WORKD,WORKL,INFO] = dneupd(RVEC,HOWMANY,SELECT,Dr
                                                         BMAT,N,WHICH,NEV,TOL,R
```

# Parameters

RVEC

Integer (INPUT)

Specifies whether a basis for the invariant subspace corresponding to the converged Ritz value approximations for the eigenproblem A*z = lambda*B*z is computed.

- RVEC = 0 Compute Ritz values only.
- RVEC = 1 Compute the Ritz vectors or Schur vectors.

See Remarks below.

HOWMANY

Character (INPUT)

Specifies the form of the basis for the invariant subspace corresponding to the converged Ritz values that is to be computed.

- 'A': Compute NEV Ritz vectors;
- 'P': Compute NEV Schur vectors;
- 'S': compute some of the Ritz vectors, specified by the integer array SELECT

SELECT

Integer array of dimension NCV. (INPUT)

If HOWMANY = 'S', SELECT specifies the Ritz vectors to be computed. To select the Ritz vector corresponding to a Ritz value (DR(j), DI(j)), SELECT(j) must be set to 1.

If HOWMANY = 'A' or 'P', SELECT is used as internal workspace.

DR

Double precision array of dimension NEV+1. (OUTPUT)

If IPARAM(7) = 1,2 or 3 and SIGMAI=0.0 then on exit: DR contains the real part of the Ritz approximations to the eigenvalues of A*z = lambda*B*z.

If IPARAM(7) = 3, 4 and SIGMAI is not equal to zero, then on exit: DR contains the real part of the Ritz values of OP computed by DNAUPD .

A further computation must be performed by the user to transform the Ritz values computed for OP by DNAUPD to those of the original system A*z = lambda*B*z. See remark 3 below

DI

Double precision array of dimension NEV+1. (OUTPUT)

On exit, DI contains the imaginary part of the Ritz value approximations to the eigenvalues of A*z = lambda*B*z associated with DR.

NOTE: When Ritz values are complex, they will come in complex conjugate pairs. If eigenvectors are requested, the corresponding Ritz vectors will also come in conjugate pairs and the real and imaginary parts of these are represented in two consecutive columns of the array Z (see below).

Z

Double precision N by NEV+1 array

if RVEC = 1 and HOWMANY = 'A'. (OUTPUT)

On exit, if RVEC = 1 and HOWMANY = 'A', then the columns of Z represent approximate eigenvectors (Ritz vectors) corresponding to the NCONV=IPARAM(5) Ritz values for eigensystem A*z = lambda*B*z. The complex Ritz vector associated with the Ritz value with positive imaginary part is stored in two consecutive columns. The first column holds the real part of the Ritz vector and the second column holds the imaginary part. The Ritz vector associated with the Ritz value with negative imaginary part is simply the complex conjugate of the Ritz vector associated with the positive imaginary part.

If RVEC = 0 or HOWMANY = 'P', then Z is not referenced.

NOTE: If if RVEC = 1 and a Schur basis is not required, the array Z may be set equal to first NEV+1 columns of the Arnoldi basis array V computed by DNAUPD . In this case the Arnoldi basis will be destroyed and overwritten with the eigenvector basis.

SIGMAR

Double precision (INPUT)

If IPARAM(7) = 3 or 4, represents the real part of the shift.

Not referenced if IPARAM(7) = 1 or 2.

SIGMAI

Double precision (INPUT)

If IPARAM(7) = 3 or 4, represents the imaginary part of the shift.

Not referenced if IPARAM(7) = 1 or 2. See remark 3 below.

WORKEV

Double precision work array of dimension 3*NCV. (WORKSPACE)

NOTE: The remaining arguments BMAT, N, WHICH, NEV, TOL, RESID, NCV, V, IPARAM, IPNTR, WORKD, WORKL, LWORKL, INFO must be passed directly to DNEUPD following the last call to DNAUPD .

These arguments MUST NOT BE MODIFIED between the the last call to DNAUPD and the call to DNEUPD .

Three of these parameters (V, WORKL, INFO) are also output parameters.

V

Double precision N by NCV array. (INPUT/OUTPUT)

Upon INPUT: the NCV columns of V contain the Arnoldi basis vectors for OP as constructed by DNAUPD.

Upon OUTPUT: If RVEC = 1 the first NCONV=IPARAM(5) columns contain approximate Schur vectors that span the desired invariant subspace. See Remark 2 below.

NOTE: If the array Z has been set equal to first NEV+1 columns of the array V and RVEC=1 and HOWMANY= 'A', then the Arnoldi basis held by V has been overwritten by the desired Ritz vectors. If a separate array Z has been passed then the first NCONV=IPARAM(5) columns of V will contain approximate Schur vectors that span the desired invariant subspace.

WORKL

Double precision work array of length LWORKL. (OUTPUT/WORKSPACE)

WORKL(1:ncv*ncv+3*ncv) contains information obtained in dnaupd . They are not changed by dneupd .

WORKL(ncv*ncv+3*ncv+1:3*ncv*ncv+6*ncv) holds the real and imaginary part of the untransformed Ritz values, the upper quasi-triangular matrix for H, and the associated matrix representation of the invariant subspace for H.

Note: IPNTR(9:13) contains the pointer into WORKL for addresses of the above information computed by dneupd .

- IPNTR(9): pointer to the real part of the NCV RITZ values of the original system.
- IPNTR(10): pointer to the imaginary part of the NCV RITZ values of the original system.
- IPNTR(11): pointer to the NCV corresponding error bounds.
- IPNTR(12): pointer to the NCV by NCV upper quasi-triangular Schur matrix for H.
- IPNTR(13): pointer to the NCV by NCV matrix of eigenvectors of the upper Hessenberg matrix H. Only referenced by dneupd if RVEC = 1 See Remark 2 below.

INFO

Error flag on output.

- 0: Normal exit.
- 1: The Schur form computed by LAPACK routine dlahqr could not be reordered by LAPACK routine dtrsen . Re-enter subroutine dneupd with IPARAM(5)=NCV and increase the size of the arrays DR and DI to have dimension at least dimension NCV and allocate at least NCV columns for Z.

  NOTE: Not necessary if Z and V share the same space. Please notify the authors if this error occurs.
- -1: N must be positive.
- -2: NEV must be positive.
- -3: NCV-NEV >= 2 and less than or equal to N.
- -5: WHICH must be one of 'LM', 'SM', 'LR', 'SR', 'LI', 'SI'
- -6: BMAT must be one of 'I' or 'G'.
- -7: Length of private work WORKL array is not sufficient.
- -8: Error return from calculation of a real Schur form. Informational error from LAPACK routine dlahqr .
- -9: Error return from calculation of eigenvectors. Informational error from LAPACK routine dtrevc .
- -10: IPARAM(7) must be 1,2,3,4.
- -11: IPARAM(7) = 1 and BMAT = 'G' are incompatible.
- -12: HOWMANY = 'S' not yet implemented
- -13: HOWMANY must be one of 'A' or 'P' if RVEC = 1
- -14: DNAUPD did not find any eigenvalues to sufficient accuracy.
- -15: DNEUPD got a different count of the number of converged Ritz values than DNAUPD got. This indicates the user probably made an error in passing data from DNAUPD to DNEUPD or that the data was modified before entering DNEUPD

# Description

This subroutine returns the converged approximations to eigenvalues of A*z = lambda*B*z and (optionally):

1. The corresponding approximate eigenvectors;

2. An orthonormal basis for the associated approximate invariant subspace;

3. Both.

There is negligible additional cost to obtain eigenvectors. An orthonormal basis is always computed.

There is an additional storage cost of n*nev if both are requested (in this case a separate array Z must be supplied).

The approximate eigenvalues and eigenvectors of A*z = lambda*B*z are derived from approximate eigenvalues and eigenvectors of of the linear operator OP prescribed by the MODE selection in the call to DNAUPD. DNAUPD must be called before this routine is called.

These approximate eigenvalues and vectors are commonly called Ritz values and Ritz vectors respectively. They are referred to as such in the comments that follow.

The computed orthonormal basis for the invariant subspace corresponding to these Ritz values is referred to as a Schur basis.

See documentation in the header of the subroutine DNAUPD for definition of OP as well as other terms and the relation of computed Ritz values and Ritz vectors of OP with respect to the given problem A*z = lambda*B*z.

For a brief description, see definitions of IPARAM(7), MODE and WHICH in the documentation of DNAUPD .

# Remarks

1. Currently only HOWMNY = 'A' and 'P' are implemented.

   Let trans(X) denote the transpose of X.

2. Schur vectors are an orthogonal representation for the basis of Ritz vectors. Thus, their numerical properties are often superior. If RVEC = 1 then the relationship

   A * V(:,1:IPARAM(5)) = V(:,1:IPARAM(5)) * T, and trans(V(:,1:IPARAM(5))) * V(:,1:IPARAM(5)) = I

   are approximately satisfied.

   Here T is the leading submatrix of order IPARAM(5) of the real upper quasi-triangular matrix stored workl(ipntr(12)). That is, T is block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each 2-by-2 diagonal block has its diagonal elements equal and its off-diagonal elements of opposite sign. Corresponding to each 2-by-2 diagonal block is a complex conjugate pair of Ritz values. The real Ritz values are stored on the diagonal of T.

3. If IPARAM(7) = 3 or 4 and SIGMAI is not equal zero, then the user must form the IPARAM(5) Rayleigh quotients in order to transform the Ritz values computed by DNAUPD for OP to those of A*z = lambda*B*z. Set RVEC = 1 and HOWMNY = 'A', and compute

   trans(Z(:,I)) * A * Z(:,I) if DI(I) = 0.

   If DI(I) is not equal to zero and DI(I+1) = - D(I), then the desired real and imaginary parts of the Ritz value are

trans(Z(:,I)) * A * Z(:,I) + trans(Z(:,I+1)) * A * Z(:,I+1),

trans(Z(:,I)) * A * Z(:,I+1) - trans(Z(:,I+1)) * A * Z(:,I),

respectively.

Another possibility is to set RVEC = 1 and HOWMANY = 'P' and compute

trans(V(:,1:IPARAM(5))) * A * V(:,1:IPARAM(5))

and then an upper quasi-triangular matrix of order IPARAM(5) is computed. See remark 2 above.

# Example

```
// The following sets dimensions for this problem.

nx    = 10;

nev   = 3;
ncv   = 6;
bmat  = 'I';
which = 'LM';

maxiter = 10;

// Local Arrays

iparam  = zeros(11,1);
ipntr   = zeros(14,1);
_select = zeros(ncv,1);
d       = zeros(nev+1,1);
resid   = zeros(nx,1);
v       = zeros(nx,ncv);
workd   = zeros(3*nx+1,1);
workev  = zeros(3*ncv,1);
workl   = zeros(3*ncv*ncv+6*ncv,1);

// Build the test matrix

A            = diag(10*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(-6*ones(nx-1,1));

tol    = 0;
ido    = 0;

ishfts = 1;
maxitr = 300;
mode1  = 1;

iparam(1) = ishfts;
iparam(3) = maxitr;
iparam(7) = mode1;

sigmar = 0; // the real part of the shift
sigmai = 0; // the imaginary part of the shift
```

```
// M A I N   L O O P (Reverse communication)

iter = 0;
while(iter<maxiter)
  info_dnaupd = 0;
  iter = iter + 1;
  // Repeatedly call the routine DNAUPD and take actions indicated by parameter
  // either convergence is indicated or maxitr has been exceeded.

  [ido,resid,v,iparam,ipntr,workd,workl,info_dnaupd] = dnaupd(ido,bmat,nx,which

  if (ido==99) then
    // BE CAREFUL: DON'T CALL dneupd IF ido == 99 !!
    break;
  end

  if (ido==-1 | ido==1) then
    // Perform matrix vector multiplication
    workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
    // L O O P   B A C K to call DNAUPD again.
    continue
  end

  if (info_dnaupd < 0) then
    printf('\nError with dnaupd, info = %d\n',info_dnaupd);
    printf('Check the documentation of dnaupd\n\n');
  else
    // Post-Process using DNEUPD.

    rvec    = 1;
    howmany = 'A';

    info_dneupd = 0;

    [d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_dneupd] = dneupd(rvec,how
                                                                       bmat,nx,
                                                                       iparam,i


    if (info_dneupd~=0) then
      printf('\nError with dneupd, info = %d\n', info_dneupd);
      printf('Check the documentation of dneupd.\n\n');
    end

    // Print additional convergence information.
    if (info_dneupd==1) then
      printf('\nMaximum number of iterations reached.\n\n');
    elseif (info_dneupd==3) then
      printf('\nNo shifts could be applied during implicit Arnoldi update, try
    end
  end
end

printf('\nDNSIMP\n');
printf('======\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);
```

```
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam
printf('The number of OP*x is %d\n', iparam(9));
printf('The convergence criterion is %d\n', tol);
```

# See Also

dsaupd, dnaupd

# Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

# Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.

2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.

3. B.N. Parlett, "The Symmetric Eigenvalue Problem". Prentice-Hall, 1980.

4. B.N. Parlett, B. Nour-Omid, "Towards a Black Box Lanczos Program", Computer Physics Communications, 53 (1989), pp 169-179.

5. B. Nour-Omid, B.N. Parlett, T. Ericson, P.S. Jensen, "How to Implement the Spectral Transformation", Math. Comp., 48 (1987), pp 663-673.

6. R.G. Grimes, J.G. Lewis and H.D. Simon, "A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems", SIAM J. Matr. Anal. Apps., January (1993).

7. L. Reichel, W.B. Gragg, "Algorithm 686: FORTRAN Subroutines for Updating the QR decomposition", ACM TOMS, December 1990, Volume 16 Number 4, pp 369-377.

8. R.B. Lehoucq, D.C. Sorensen, "Implementation of Some Spectral Transformations in a k-Step Arnoldi Method". In Preparation.

# Used Functions

Based on ARPACK routine dneupd

# Name

dsaupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute approximations to a few eigenpairs of a real and symmetric linear operator

```
[IDO,RESID,V,IPARAM,IPNTR,WORKD,WORKL,INFO] = dsaupd(ID0,BMAT,N,WHICH,NEV,TOL,R
```

# Parameters

ID0

Integer. (INPUT/OUTPUT)

Reverse communication flag. IDO must be zero on the first call to dsaupd . IDO will be set internally to indicate the type of operation to be performed. Control is then given back to the calling routine which has the responsibility to carry out the requested operation and call dsaupd with the result. T

he operand is given in WORKD(IPNTR(1)), the result must be put in WORKD(IPNTR(2)). (If Mode = 2 see remark 5 below)

- IDO = 0: first call to the reverse communication interface
- IDO = -1: compute Y = OP * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. This is for the initialization phase to force the starting vector into the range of OP.
- IDO = 1: compute Y = OP * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. In mode 3,4 and 5, the vector B * X is already available in WORKD(ipntr(3)). It does not need to be recomputed in forming OP * X.
- IDO = 2: compute Y = B * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y.
- IDO = 3: compute the IPARAM(8) shifts where IPNTR(11) is the pointer into WORKL for placing the shifts. See remark 6 below.
- IDO = 99: done

BMAT

Character. (INPUT)

Specifies the type of the matrix B that defines the semi-inner product for the operator OP.

- 'I': standard eigenvalue problem A*x = lambda*x
- 'G': generalized eigenvalue problem A*x = lambda*B*x

N

Integer.

Dimension of the eigenproblem.

WHICH

String of length 2.

Specify which of the Ritz values of OP to compute.

- 'LA' - compute the NEV largest (algebraic) eigenvalues.
- 'SA' - compute the NEV smallest (algebraic) eigenvalues.

- 'LM' - compute the NEV largest (in magnitude) eigenvalues.
- 'SM' - compute the NEV smallest (in magnitude) eigenvalues.
- 'BE' - compute NEV eigenvalues, half from each end of the spectrum. When NEV is odd, compute one more from the high end than from the low end. (see remark 1 below)

NEV
: Integer.

Number of eigenvalues of OP to be computed. 0 < NEV < N.

TOL
: scalar.

Stopping criterion: the relative accuracy of the Ritz value is considered acceptable if BOUNDS(I) <= TOL*ABS(RITZ(I)). If TOL <= 0. is passed the machine precision is set.

RESID
: Array of length N (INPUT/OUTPUT)

On INPUT: If INFO==0, a random initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

On OUTPUT: RESID contains the final residual vector.

NCV
: Integer.

Number of columns of the matrix V (less than or equal to N).

This will indicate how many Lanczos vectors are generated at each iteration. After the startup phase in which NEV Lanczos vectors are generated, the algorithm generates NCV-NEV Lanczos vectors at each subsequent update iteration. Most of the cost in generating each Lanczos vector is in the matrix-vector product OP*x. (See remark 4 below).

V
: N by NCV array.

The NCV columns of V contain the Lanczos basis vectors.

IPARAM
: array of length 11. (INPUT/OUTPUT)

- IPARAM(1) = ISHIFT: method for selecting the implicit shifts. The shifts selected at each iteration are used to restart the Arnoldi iteration in an implicit fashion.
  - ISHIFT = 0: the shifts are provided by the user via reverse communication. The NCV eigenvalues of the current tridiagonal matrix T are returned in the part of WORKL array corresponding to RITZ. See remark 6 below.
  - ISHIFT = 1: exact shifts with respect to the reduced tridiagonal matrix T. This is equivalent to restarting the iteration with a starting vector that is a linear combination of Ritz vectors associated with the "wanted" Ritz values.
- IPARAM(2) = LEVEC No longer referenced. See remark 2 below.
- IPARAM(3) = MXITER

  On INPUT: maximum number of Arnoldi update iterations allowed.

  On OUTPUT: actual number of Arnoldi update iterations taken.

- IPARAM(4) = NB: blocksize to be used in the recurrence. The code currently works only for NB = 1.
- IPARAM(5) = NCONV: number of "converged" Ritz values. This represents the number of Ritz values that satisfy the convergence criterion.
- IPARAM(6) = IUPD No longer referenced. Implicit restarting is ALWAYS used.
- IPARAM(7) = MODE On INPUT determines what type of eigenproblem is being solved. Must be 1,2,3,4,5; See under Description of dsaupd for the five modes available.
- IPARAM(8) = NP When ido = 3 and the user provides shifts through reverse communication (IPARAM(1)=0), dsaupd returns NP, the number of shifts the user is to provide. 0 < NP <= NCV-NEV. See Remark 6 below.
- IPARAM(9) = NUMOP,

  IPARAM(10) = NUMOPB,

  IPARAM(11) = NUMREO, OUTPUT: NUMOP = total number of OP*x operations, NUMOPB = total number of B*x operations if BMAT='G', NUMREO = total number of steps of re-orthogonalization.

IPNTR
array of length 11. Pointer to mark the starting locations in the WORKD and WORKL arrays for matrices/vectors used by the Lanczos iteration.

- IPNTR(1): pointer to the current operand vector X in WORKD.
- IPNTR(2): pointer to the current result vector Y in WORKD.
- IPNTR(3): pointer to the vector B * X in WORKD when used in the shift-and-invert mode.
- IPNTR(4): pointer to the next available location in WORKL that is untouched by the program.
- IPNTR(5): pointer to the NCV by 2 tridiagonal matrix T in WORKL.
- IPNTR(6): pointer to the NCV RITZ values array in WORKL.
- IPNTR(7): pointer to the Ritz estimates in array WORKL associated with the Ritz values located in RITZ in WORKL.
- IPNTR(11): pointer to the NP shifts in WORKL. See Remark 6 below.

Note: IPNTR(8:10) is only referenced by dseupd . See Remark 2.

- IPNTR(8): pointer to the NCV RITZ values of the original system.
- IPNTR(9): pointer to the NCV corresponding error bounds.
- IPNTR(10): pointer to the NCV by NCV matrix of eigenvectors of the tridiagonal matrix T. Only referenced by dseupd if RVEC = 1. See Remarks

WORKD
work array of length 3*N. (REVERSE COMMUNICATION)

Distributed array to be used in the basic Arnoldi iteration for reverse communication. The user should not use WORKD as temporary workspace during the iteration. Upon termination WORKD(1:N) contains B*RESID(1:N). If the Ritz vectors are desired subroutine dseupd uses this output. See Data Distribution Note below.

WORKL
work array of length at least NCV**2 + 8*NCV. (OUTPUT/WORKSPACE)

Private (replicated) array on each PE or array allocated on the front end. See Data Distribution Note below. add here the parameter description

INFO
Integer. (INPUT/OUTPUT)

If INFO == 0, a randomly initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

Error flag on output.

- 0: Normal exit.
- 1: Maximum number of iterations taken. All possible eigenvalues of OP has been found. IPARAM(5) returns the number of wanted converged Ritz values.
- 2: No longer an informational error. Deprecated starting with release 2 of ARPACK.
- 3: No shifts could be applied during a cycle of the Implicitly restarted Arnoldi iteration. One possibility is to increase the size of NCV relative to NEV. See remark 4 below.
- -1: N must be positive.
- -2: NEV must be positive.
- -3: NCV must be greater than NEV and less than or equal to N.
- -4: The maximum number of Arnoldi update iterations allowed must be greater than zero.
- -5: WHICH must be one of 'LM', 'SM', 'LA', 'SA' or 'BE'.
- -6: BMAT must be one of 'I' or 'G'.
- -7: Length of private work array WORKL is not sufficient.
- -8: Error return from trid. eigenvalue calculation; Informatinal error from LAPACK routine dsteqr .
- -9: Starting vector is zero.
- -10: IPARAM(7) must be 1,2,3,4,5.
- -11: IPARAM(7) = 1 and BMAT = 'G' are incompatable.
- -12: IPARAM(1) must be equal to 0 or 1.
- -13: NEV and WHICH = 'BE' are incompatable.
- -9999: Could not build an Arnoldi factorization. IPARAM(5) returns the size of the current Arnoldi factorization. The user is advised to check that enough workspace and array storage has been allocated.

# Description

Reverse communication interface for the Implicitly Restarted Arnoldi Iteration. For symmetric problems this reduces to a variant of the Lanczos method. This method has been designed to compute approximations to a few eigenpairs of a linear operator OP that is real and symmetric with respect to a real positive semi-definite symmetric matrix B, i.e.`B*OP = (OP`)*B`.

Another way to express this condition is `< x,OPy > = < OPx,y > where <z,w > = z`Bw` .

In the standard eigenproblem B is the identity matrix. ( A` denotes transpose of A)

The computed approximate eigenvalues are called Ritz values and the corresponding approximate eigenvectors are called Ritz vectors.

dsaupd is usually called iteratively to solve one of the following problems:

- Mode 1: A*x = lambda*x, A symmetric ===> OP = A and B = I.
- Mode 2: A*x = lambda*M*x, A symmetric, M symmetric positive definite ===> OP = inv[M]*A and B = M. ===> (If M can be factored see remark 3 below)
- Mode 3: K*x = lambda*M*x, K symmetric, M symmetric positive semi-definite ===> OP = (inv[K - sigma*M])*M and B = M. ===> Shift-and-Invert mode
- Mode 4: K*x = lambda*KG*x, K symmetric positive semi-definite, KG symmetric indefinite ===> OP = (inv[K - sigma*KG])*K and B = K. ===> Buckling mode
- Mode 5: A*x = lambda*M*x, A symmetric, M symmetric positive semi-definite ===> OP = inv[A - sigma*M]*[A + sigma*M] and B = M. ===> Cayley transformed mode

NOTE: The action of w <- inv[A - sigma*M]*v or w <- inv[M]*v should be accomplished either by a direct method using a sparse matrix factorization and solving `[A - sigma*M]*w = v or M*w = v`,

or through an iterative method for solving these systems. If an iterative method is used, the convergence test must be more stringent than the accuracy requirements for the eigenvalue approximations.

# Remarks

1. The converged Ritz values are always returned in ascending algebraic order. The computed Ritz values are approximate eigenvalues of OP. The selection of WHICH should be made with this in mind when Mode = 3,4,5. After convergence, approximate eigenvalues of the original problem may be obtained with the ARPACK subroutine dseupd .

2. If the Ritz vectors corresponding to the converged Ritz values are needed, the user must call dseupd immediately following completion of dsaupd . This is new starting with version 2.1 of ARPACK.

3. If M can be factored into a Cholesky factorization M = LL` then Mode = 2 should not be selected. Instead one should use Mode = 1 with OP = inv(L)*A*inv(L`). Appropriate triangular linear systems should be solved with L and L` rather than computing inverses. After convergence, an approximate eigenvector z of the original problem is recovered by solving L`z = x where x is a Ritz vector of OP.

4. At present there is no a-priori analysis to guide the selection of NCV relative to NEV. The only formal requrement is that NCV > NEV. However, it is recommended that NCV >= 2*NEV. If many problems of the same type are to be solved, one should experiment with increasing NCV while keeping NEV fixed for a given test problem. This will usually decrease the required number of OP*x operations but it also increases the work and storage required to maintain the orthogonal basis vectors. The optimal "cross-over" with respect to CPU time is problem dependent and must be determined empirically.

5. If IPARAM(7) = 2 then in the Reverse commuication interface the user must do the following. When IDO = 1, Y = OP * X is to be computed. When IPARAM(7) = 2 OP = inv(B)*A. After computing A*X the user must overwrite X with A*X. Y is then the solution to the linear set of equations B*Y = A*X.

6. When IPARAM(1) = 0, and IDO = 3, the user needs to provide the NP = IPARAM(8) shifts in locations: 1 WORKL(IPNTR(11)) 2 WORKL(IPNTR(11)+1) . . . NP WORKL(IPNTR(11)+NP-1). The eigenvalues of the current tridiagonal matrix are located in WORKL(IPNTR(6)) through WORKL(IPNTR(6)+NCV-1). They are in the order defined by WHICH. The associated Ritz estimates are located in WORKL(IPNTR(8)), WORKL(IPNTR(8)+1), ... , WORKL(IPNTR(8)+NCV-1).

# Example

```
// The following sets dimensions for this problem.

nx    = 10;

nev   = 3;
ncv   = 6;
bmat  = 'I';
which = 'LM';

maxiter = 10;

// Local Arrays

iparam  = zeros(11,1);
ipntr   = zeros(14,1);
_select = zeros(ncv,1);
d       = zeros(nev+1,1);
resid   = zeros(nx,1);
v       = zeros(nx,ncv);
workd   = zeros(3*nx+1,1);
workl   = zeros(ncv*ncv+8*ncv,1);

// Build the symmetric test matrix

A            = diag(10*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(6*ones(nx-1,1));

tol    = 0;
ido    = 0;

ishfts = 1;
maxitr = 300;
mode1  = 1;

iparam(1) = ishfts;
iparam(3) = maxitr;
iparam(7) = mode1;

sigma = 0; // the real part of the shift

// M A I N   L O O P (Reverse communication)

iter = 0;
while(iter<maxiter)
  info_dsaupd = 0;
  iter = iter + 1;
  // Repeatedly call the routine DSAUPD and take actions indicated by parameter
  // either convergence is indicated or maxitr has been exceeded.

  [ido,resid,v,iparam,ipntr,workd,workl,info_dsaupd] = dsaupd(ido,bmat,nx,which

  if (ido==99) then
```

```
    // BE CAREFUL: DON'T CALL dseupd IF ido == 99 !!
    break;
  end

  if (ido==-1 | ido==1) then
    // Perform matrix vector multiplication
    workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
    // L O O P   B A C K to call DSAUPD again.
    continue
  end

  if (info_dsaupd < 0) then
    printf('\nError with dsaupd, info = %d\n',info_dsaupd);
    printf('Check the documentation of dsaupd\n\n');
  else
    // Post-Process using DSEUPD.

    rvec    = 1;
    howmany = 'A';

    info_dseupd = 0;

    [d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_dseupd] = dseupd(rvec,how
                                                                      bmat,nx,
                                                                      iparam,i


    if (info_dseupd~=0) then
      printf('\nError with dseupd, info = %d\n', info_dseupd);
      printf('Check the documentation of dseupd.\n\n');
    end

    if (info_dseupd==1) then
      printf('\nMaximum number of iterations reached.\n\n');
    elseif (info_dseupd==3) then
      printf('\nNo shifts could be applied during implicit Arnoldi update, try
    end
  end
end

// Done with program dssimp.
printf('\nDSSIMP\n');
printf('======\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam
printf('The number of OP*x is %d\n', iparam(9));
printf('The convergence criterion is %d\n', tol);
```

## See Also

dnaupd

# Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

# Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.

2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.

3. B.N. Parlett and Y. Saad, "Complex Shift and Invert Strategies for Real Matrices", Linear Algebra and its Applications, vol 88/89, pp 575-595, (1987).

# Used Functions

Based on ARPACK routine dsaupd

# Name

dsaupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute approximations to the converged approximations to eigenvalues of A*z = lambda*B*z

```
[D,Z,RESID,V,IPARAM,IPNTR,WORKD,WORKL,INFO] = dseupd(RVEC,HOWMANY,SELECT,D,Z,SI
      NEV,TOL,RESID,NCV,V,IPARAM,IPNTR,WORKD,WORKL,INFO)
```

# Parameters

RVEC

Integer (INPUT)

Specifies whether Ritz vectors corresponding to the Ritz value approximations to the eigenproblem A*z = lambda*B*z are computed.

- RVEC = 0 Compute Ritz values only.
- RVEC = 1 Compute Ritz vectors.

HOWMNY

Character*1 (INPUT)

Specifies how many Ritz vectors are wanted and the form of Z the matrix of Ritz vectors. See remark 1 below.

- 'A': compute NEV Ritz vectors;
- 'S': compute some of the Ritz vectors, specified by the integer array SELECT.

SELECT

Integer array of dimension NCV. (INPUT/WORKSPACE)

If HOWMANY = 'S', SELECT specifies the Ritz vectors to be computed. To select the Ritz vector corresponding to a Ritz value D(j), SELECT(j) must be set to 1.

If HOWMANY = 'A' , SELECT is used as a workspace for reordering the Ritz values.

D

Double precision array of dimension NEV. (OUTPUT)

On exit, D contains the Ritz value approximations to the eigenvalues of A*z = lambda*B*z. The values are returned in ascending order.

If IPARAM(7) = 3,4,5 then D represents the Ritz values of OP computed by dsaupd transformed to those of the original eigensystem A*z = lambda*B*z.

If IPARAM(7) = 1,2 then the Ritz values of OP are the same as the those of A*z = lambda*B*z.

Z

Double precision N by NEV array.

If HOWMNY = 'A'. (OUTPUT) On exit, Z contains the B-orthonormal Ritz vectors of the eigensystem A*z = lambda*B*z corresponding to the Ritz value approximations.

If RVEC = 0 then Z is not referenced.

NOTE: The array Z may be set equal to first NEV columns of the Arnoldi/Lanczos basis array V computed by DSAUPD .

SIGMA

Double precision (INPUT)

If IPARAM(7) = 3,4,5 represents the shift. Not referenced if IPARAM(7) = 1 or 2.

NOTE: The remaining arguments BMAT, N, WHICH, NEV, TOL, RESID, NCV, V, IPARAM, IPNTR, WORKD, WORKL, LWORKL, INFO must be passed directly to DSEUPD following the last call to DSAUPD .

These arguments MUST NOT BE MODIFIED between the the last call to DSAUPD and the call to DSEUPD.

Two of these parameters (WORKL, INFO) are also output parameters.

WORKL

Double precision work array of length LWORKL. (OUTPUT/WORKSPACE)

WORKL(1:4*ncv) contains information obtained in dsaupd. They are not changed by dseupd.

WORKL(4*ncv+1:ncv*ncv+8*ncv) holds the untransformed Ritz values, the computed error estimates, and the associated eigenvector matrix of H.

Note: IPNTR(8:10) contains the pointer into WORKL for addresses of the above information computed by dseupd .

- IPNTR(8): pointer to the NCV RITZ values of the original system.
- IPNTR(9): pointer to the NCV corresponding error bounds.
- IPNTR(10): pointer to the NCV by NCV matrix of eigenvectors of the tridiagonal matrix T. Only referenced by dseupd if RVEC = 1 See Remarks.

INFO

Integer. (OUTPUT)

Error flag on output.

- 0: Normal exit.
- -1: N must be positive.
- -2: NEV must be positive.
- -3: NCV must be greater than NEV and less than or equal to N.
- -5: WHICH must be one of 'LM', 'SM', 'LA', 'SA' or 'BE'.
- -6: BMAT must be one of 'I' or 'G'.
- -7: Length of private work WORKL array is not sufficient.
- -8: Error return from trid. eigenvalue calculation; Information error from LAPACK routine dsteqr .
- -9: Starting vector is zero.
- -10: IPARAM(7) must be 1,2,3,4,5.
- -11: IPARAM(7) = 1 and BMAT = 'G' are incompatible.
- -12: NEV and WHICH = 'BE' are incompatible.
- -14: DSAUPD did not find any eigenvalues to sufficient accuracy.
- -15: HOWMNY must be one of 'A' or 'S' if RVEC = 1
- -16: HOWMNY = 'S' not yet implemented
- -17: DSEUPD got a different count of the number of converged Ritz values than DSAUPD got. This indicates the user probably made an error in passing data from DSAUPD to DSEUPD or that the data was modified before entering DSEUPD.

# Description

This subroutine returns the converged approximations to eigenvalues of A*z = lambda*B*z and (optionally):

1. the corresponding approximate eigenvectors,

2. an orthonormal (Lanczos) basis for the associated approximate invariant subspace,

3. Both.

There is negligible additional cost to obtain eigenvectors. An orthonormal (Lanczos) basis is always computed. There is an additional storage cost of n*nev if both are requested (in this case a separate array Z must be supplied).

These quantities are obtained from the Lanczos factorization computed by DSAUPD for the linear operator OP prescribed by the MODE selection (see IPARAM(7) in DSAUPD documentation.) DSAUPD must be called before this routine is called.

These approximate eigenvalues and vectors are commonly called Ritz values and Ritz vectors respectively. They are referred to as such in the comments that follow.

The computed orthonormal basis for the invariant subspace corresponding to these Ritz values is referred to as a Lanczos basis.

See documentation in the header of the subroutine DSAUPD for a definition of OP as well as other terms and the relation of computed Ritz values and vectors of OP with respect to the given problem A*z = lambda*B*z.

The approximate eigenvalues of the original problem are returned in ascending algebraic order.

The user may elect to call this routine once for each desired Ritz vector and store it peripherally if desired. There is also the option of computing a selected set of these vectors with a single call.

# Remarks

1. The converged Ritz values are always returned in increasing (algebraic) order. c 2. Currently only HOWMNY = 'A' is implemented. It is included at this stage for the user who wants to incorporate it.

# Example

```
// The following sets dimensions for this problem.

nx    = 10;

nev   = 3;
ncv   = 6;
bmat  = 'I';
which = 'LM';

maxiter = 10;

// Local Arrays

iparam  = zeros(11,1);
ipntr   = zeros(14,1);
_select = zeros(ncv,1);
d       = zeros(nev+1,1);
```

```
resid   = zeros(nx,1);
v       = zeros(nx,ncv);
workd   = zeros(3*nx+1,1);
workl   = zeros(ncv*ncv+8*ncv,1);

// Build the symmetric test matrix

A            = diag(10*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(6*ones(nx-1,1));

tol    = 0;
ido    = 0;

ishfts = 1;
maxitr = 300;
mode1  = 1;

iparam(1) = ishfts;
iparam(3) = maxitr;
iparam(7) = mode1;

sigma = 0; // the real part of the shift

// M A I N   L O O P (Reverse communication)

iter = 0;
while(iter<maxiter)
  info_dsaupd = 0;
  iter = iter + 1;
  // Repeatedly call the routine DSAUPD and take actions indicated by parameter
  // either convergence is indicated or maxitr has been exceeded.

  [ido,resid,v,iparam,ipntr,workd,workl,info_dsaupd] = dsaupd(ido,bmat,nx,which

  if (ido==99) then
    // BE CAREFUL: DON'T CALL dseupd IF ido == 99 !!
    break;
  end

  if (ido==-1 | ido==1) then
    // Perform matrix vector multiplication
    workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
    // L O O P   B A C K to call DSAUPD again.
    continue
  end

  if (info_dsaupd < 0) then
    printf('\nError with dsaupd, info = %d\n',info_dsaupd);
    printf('Check the documentation of dsaupd\n\n');
  else
    // Post-Process using DSEUPD.

    rvec    = 1;
    howmany = 'A';

    info_dseupd = 0;
```

```
    [d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_dseupd] = dseupd(rvec,how
                                                                      bmat,nx,
                                                                      iparam,i


    if (info_dseupd~=0) then
      printf('\nError with dseupd, info = %d\n', info_dseupd);
      printf('Check the documentation of dseupd.\n\n');
    end

    if (info_dseupd==1) then
      printf('\nMaximum number of iterations reached.\n\n');
    elseif (info_dseupd==3) then
      printf('\nNo shifts could be applied during implicit Arnoldi update, try
    end
  end
end

// Done with program dssimp.
printf('\nDSSIMP\n');
printf('======\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam
printf('The number of OP*x is %d\n', iparam(9));
printf('The convergence criterion is %d\n', tol);
```

# See Also

dsaupd, dneupd

# Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

# Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.

2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.

3. B.N. Parlett and Y. Saad, "Complex Shift and Invert Strategies for Real Matrices", Linear Algebra and its Applications, vol 88/89, pp 575-595, (1987).

# Used Functions

Based on ARPACK routine dsaupd

# Name

znaupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute a few eigenpairs of a complex linear operator OP with respect to a semi-inner product defined by a hermitian positive semi-definite real matrix B.

```
[IDO,RESID,V,IPARAM,IPNTR,WORKD,WORKL,RWORKINFO] = znaupd(ID0,BMAT,N,WHICH,NEV,
```

# Parameters

IDO

Integer. (INPUT/OUTPUT)

Reverse communication flag. IDO must be zero on the first call to znaupd. IDO will be set internally to indicate the type of operation to be performed. Control is then given back to the calling routine which has the responsibility to carry out the requested operation and call znaupd with the result.

The operand is given in WORKD(IPNTR(1)), the result must be put in WORKD(IPNTR(2)).

- IDO = 0: first call to the reverse communication interface
- IDO = -1: compute Y = OP * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y.

  This is for the initialization phase to force the starting vector into the range of OP.
- IDO = 1: compute Y = OP * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y.

  In mode 3, the vector B * X is already available in WORKD(ipntr(3)). It does not need to be recomputed in forming OP * X.
- IDO = 2: compute Y = M * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y.
- IDO = 3: compute and return the shifts in the first NP locations of WORKL.
- IDO = 99: done

After the initialization phase, when the routine is used in the "shift-and-invert" mode, the vector M * X is already available and does not need to be recomputed in forming OP*X.

BMAT

Character. (INPUT)

BMAT specifies the type of the matrix B that defines the semi-inner product for the operator OP.

- 'I': standard eigenvalue problem A*x = lambda*x
- 'G': generalized eigenvalue problem A*x = lambda*M*x

N

Integer. (INPUT)

Dimension of the eigenproblem.

WHICH

Characters. (INPUT)

- 'LM': want the NEV eigenvalues of largest magnitude.
- 'SM': want the NEV eigenvalues of smallest magnitude.

- 'LR': want the NEV eigenvalues of largest real part.
- 'SR': want the NEV eigenvalues of smallest real part.
- 'LI': want the NEV eigenvalues of largest imaginary part.
- 'SI': want the NEV eigenvalues of smallest imaginary part.

NEV
: Integer. (INPUT)

Number of eigenvalues of OP to be computed. 0 < NEV < N-1.

TOL
: Double precision scalar. (INPUT)

Stopping criteria: the relative accuracy of the Ritz value is considered acceptable if BOUNDS(I) .LE. TOL*ABS(RITZ(I)) where ABS(RITZ(I)) is the magnitude when RITZ(I) is complex. DEFAULT = dlamch('EPS') (machine precision as computed by the LAPACK auxiliary subroutine dlamch).

RESID
: Complex*16 array of length N. (INPUT/OUTPUT)

On INPUT: If INFO .EQ. 0, a random initial residual vector is used. If INFO .NE. 0, RESID contains the initial residual vector, possibly from a previous run.

On OUTPUT: RESID contains the final residual vector.

NCV
: Integer. (INPUT)

Number of columns of the matrix V. NCV must satisfy the two inequalities 1 <= NCV-NEV and NCV <= N.

This will indicate how many Arnoldi vectors are generated at each iteration. After the startup phase in which NEV Arnoldi vectors are generated, the algorithm generates approximately NCV-NEV Arnoldi vectors at each subsequent update iteration. Most of the cost in generating each Arnoldi vector is in the matrix-vector operation OP*x. (See remark 4 below.)

V
: Complex*16 array N by NCV. (OUTPUT)

Contains the final set of Arnoldi basis vectors.

IPARAM
: Integer array of length 11. (INPUT/OUTPUT)

- IPARAM(1) = ISHIFT: method for selecting the implicit shifts. The shifts selected at each iteration are used to filter out the components of the unwanted eigenvector.
  - ISHIFT = 0: the shifts are to be provided by the user via reverse communication. The NCV eigenvalues of the Hessenberg matrix H are returned in the part of WORKL array corresponding to RITZ.
  - ISHIFT = 1: exact shifts with respect to the current Hessenberg matrix H. This is equivalent to restarting the iteration from the beginning after updating the starting vector with a linear combination of Ritz vectors associated with the "wanted" eigenvalues.
  - ISHIFT = 2: other choice of internal shift to be defined.
- IPARAM(2) = No longer referenced

- IPARAM(3) = MXITER

  On INPUT: maximum number of Arnoldi update iterations allowed.

  On OUTPUT: actual number of Arnoldi update iterations taken.

- IPARAM(4) = NB: blocksize to be used in the recurrence. The code currently works only for NB = 1.

- IPARAM(5) = NCONV: number of "converged" Ritz values. This represents the number of Ritz values that satisfy the convergence criterion.

- IPARAM(6) = IUPD No longer referenced. Implicit restarting is ALWAYS used.

- IPARAM(7) = MODE On INPUT determines what type of eigenproblem is being solved. Must be 1,2,3; See under Description of znaupd for the four modes available.

- IPARAM(8) = NP When ido = 3 and the user provides shifts through reverse communication (IPARAM(1)=0), _naupd returns NP, the number of shifts the user is to provide. 0 < NP < NCV-NEV.

- IPARAM(9) = NUMOP,

  IPARAM(10) = NUMOPB,

  IPARAM(11) = NUMREO, OUTPUT: NUMOP = total number of OP*x operations, NUMOPB = total number of B*x operations if BMAT='G', NUMREO = total number of steps of re-orthogonalization.

IPNTR

Integer array of length 14. (OUTPUT)

Pointer to mark the starting locations in the WORKD and WORKL arrays for matrices/vectors used by the Arnoldi iteration.

- IPNTR(1): pointer to the current operand vector X in WORKD.

- IPNTR(2): pointer to the current result vector Y in WORKD.

- IPNTR(3): pointer to the vector B * X in WORKD when used in the shift-and-invert mode.

- IPNTR(4): pointer to the next available location in WORKL that is untouched by the program.

- IPNTR(5): pointer to the NCV by NCV upper Hessenberg matrix H in WORKL.

- IPNTR(6): pointer to the ritz value array RITZ

- IPNTR(7): pointer to the (projected) ritz vector array Q

- IPNTR(8): pointer to the error BOUNDS array in WORKL.

- IPNTR(14): pointer to the NP shifts in WORKL. See Remark 5 below.

Note: IPNTR(9:13) is only referenced by zneupd. See Remark 2 below.

- IPNTR(9): pointer to the NCV RITZ values of the original system.

- IPNTR(10): Not Used

- IPNTR(11): pointer to the NCV corresponding error bounds.

- IPNTR(12): pointer to the NCV by NCV upper triangular Schur matrix for H.

- IPNTR(13): pointer to the NCV by NCV matrix of eigenvectors of the upper Hessenberg matrix H. Only referenced by zneupd if RVEC = 1 See Remark 2 below.

WORKD
: Complex*16 work array of length 3*N. (REVERSE COMMUNICATION)

  Distributed array to be used in the basic Arnoldi iteration for reverse communication.

  The user should not use WORKD as temporary workspace during the iteration !!!!!!!!!!

  See Data Distribution Note below.

WORKL
: Complex*16 work array of length 3*NCV**2 + 5*NCV. (OUTPUT/WORKSPACE)

  Private (replicated) array on each PE or array allocated on the front end. See Data Distribution Note below.

RWORK
: Double precision work array of length NCV (WORKSPACE) Private (replicated) array on each PE or array allocated on the front end.

INFO
: Integer. (INPUT/OUTPUT)

  If INFO == 0, a randomly initial residual vector is used.

  If INFO ~= 0, RESID contains the initial residual vector, possibly from a previous run.

  Error flag on output.

  - 0: Normal exit.
  - 1: Maximum number of iterations taken. All possible eigenvalues of OP has been found. IPARAM(5) returns the number of wanted converged Ritz values.
  - 2: No longer an informational error. Deprecated starting with release 2 of ARPACK.
  - 3: No shifts could be applied during a cycle of the Implicitly restarted Arnoldi iteration. One possibility is to increase the size of NCV relative to NEV. See remark 4 below.
  - -1: N must be positive.
  - -2: NEV must be positive.
  - -3: NCV-NEV >= 1 and less than or equal to N.
  - -4: The maximum number of Arnoldi update iteration must be greater than zero.
  - -5: WHICH must be one of 'LM', 'SM', 'LR', 'SR', 'LI', 'SI'
  - -6: BMAT must be one of 'I' or 'G'.
  - -7: Length of private work array is not sufficient.
  - -8: Error return from LAPACK eigenvalue calculation;
  - -9: Starting vector is zero.
  - -10: IPARAM(7) must be 1,2,3.
  - -11: IPARAM(7) = 1 and BMAT = 'G' are incompatible.
  - -12: IPARAM(1) must be equal to 0 or 1.
  - -9999: Could not build an Arnoldi factorization. User input error highly likely. Please check actual array dimensions and layout. IPARAM(5) returns the size of the current Arnoldi factorization.

# Description

Reverse communication interface for the Implicitly Restarted Arnoldi iteration. This is intended to be used to find a few eigenpairs of a complex linear operator OP with respect to a semi-inner product defined by a hermitian positive semi-definite real matrix B. B may be the identity matrix.

NOTE: if both OP and B are real, then dsaupd or dnaupd should be used.

The computed approximate eigenvalues are called Ritz values and the corresponding approximate eigenvectors are called Ritz vectors. znaupd is usually called iteratively to solve one of the following problems:

- Mode 1: A*x = lambda*x.

  ===> OP = A and B = I.

- Mode 2: A*x = lambda*M*x, M hermitian positive definite

  ===> OP = inv[M]*A and B = M.

  ===> (If M can be factored see remark 3 below)

- Mode 3: A*x = lambda*M*x, M hermitian semi-definite

  ===> OP = inv[A - sigma*M]*M and B = M.

  ===> shift-and-invert mode If OP*x = amu*x, then lambda = sigma + 1/amu.

NOTE: The action of w <- inv[A - sigma*M]*v or w <- inv[M]*v should be accomplished either by a direct method using a sparse matrix factorization and solving

[A - sigma*M]*w = v or M*w = v,

or through an iterative method for solving these systems. If an iterative method is used, the convergence test must be more stringent than the accuracy requirements for the eigenvalue approximations.

# Remarks

1. The computed Ritz values are approximate eigenvalues of OP. The selection of WHICH should be made with this in mind when using Mode = 3. When operating in Mode = 3 setting WHICH = 'LM' will compute the NEV eigenvalues of the original problem that are closest to the shift SIGMA . After convergence, approximate eigenvalues of the original problem may be obtained with the ARPACK subroutine zneupd.

2. If a basis for the invariant subspace corresponding to the converged Ritz values is needed, the user must call zneupd immediately following completion of znaupd. This is new starting with release 2 of ARPACK.

3. If M can be factored into a Cholesky factorization M = LL` then Mode = 2 should not be selected. Instead one should use Mode = 1 with OP = inv(L)*A*inv(L`). Appropriate triangular linear systems should be solved with L and L` rather than computing inverses. After convergence, an approximate eigenvector z of the original problem is recovered by solving L`z = x where x is a Ritz vector of OP.

4. At present there is no a-priori analysis to guide the selection of NCV relative to NEV. The only formal requirement is that NCV > NEV + 1. However, it is recommended that NCV .ge. 2*NEV. If many problems of the same type are to be solved, one should experiment with increasing NCV while keeping NEV fixed for a given test problem. This will usually decrease the required number of OP*x operations but it also increases the work and storage required to maintain the orthogonal basis vectors. The optimal "cross-over" with respect to CPU time is problem dependent and must be determined empirically. See Chapter 8 of Reference 2 for further information.

5. When IPARAM(1) = 0, and IDO = 3, the user needs to provide the NP = IPARAM(8) complex shifts in locations

   WORKL(IPNTR(14)), WORKL(IPNTR(14)+1), ... , WORKL(IPNTR(14)+NP).

   Eigenvalues of the current upper Hessenberg matrix are located in WORKL(IPNTR(6)) through WORKL(IPNTR(6)+NCV-1). They are ordered according to the order defined by WHICH. The associated Ritz estimates are located in

   WORKL(IPNTR(8)), WORKL(IPNTR(8)+1), ... , WORKL(IPNTR(8)+NCV-1).

# Example

```
// The following sets dimensions for this problem.

nx    = 10;

nev   = 3;
ncv   = 6;
bmat  = 'I';
which = 'LM';

maxiter = 10;

// Local Arrays

iparam  = zeros(11,1);
ipntr   = zeros(14,1);
_select = zeros(ncv,1);
d       = zeros(nev+1,1);
resid   = zeros(nx,1) + 0*%i;
v       = zeros(nx,ncv) + 0*%i;
workd   = zeros(3*nx+1,1) + 0*%i;
workev  = zeros(3*ncv,1);
rwork   = zeros(ncv,1);
workl   = zeros(3*ncv*ncv+5*ncv,1) + 0*%i;

// Build the complex test matrix
A            = diag(10*ones(nx,1)+%i*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(-6*ones(nx-1,1));

tol    = 0;
ido    = 0;

ishfts = 1;
maxitr = 300;
mode1  = 1;

iparam(1) = ishfts;
iparam(3) = maxitr;
iparam(7) = mode1;

sigma = 0; // the real part of the shift

// M A I N   L O O P (Reverse communication)
```

```
iter = 0;
while(iter<maxiter)
  info_znaupd = 0;
  iter = iter + 1;
  // Repeatedly call the routine ZNAUPD and take actions indicated by parameter
  // either convergence is indicated or maxitr has been exceeded.

  [ido,resid,v,iparam,ipntr,workd,workl,info_znaupd] = znaupd(ido,bmat,nx,which

  if (ido==99) then
    // BE CAREFUL: DON'T CALL zneupd IF ido == 99 !!
    break;
  end

  if (ido==-1 | ido==1) then
    // Perform matrix vector multiplication
    workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
    // L O O P   B A C K to call ZNAUPD again.
    continue
  end

  if (info_znaupd < 0) then
    printf('\nError with znaupd, info = %d\n',info_znaupd);
    printf('Check the documentation of znaupd\n\n');
  else
    // Post-Process using ZNEUPD.

    rvec    = 1;
    howmany = 'A';

    info_zneupd = 0;

    [d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_zneupd] = zneupd(rvec,how
                                                                    bmat,nx,
                                                                    iparam,i


    if (info_zneupd~=0) then
      printf('\nError with zneupd, info = %d\n', info_zneupd);
      printf('Check the documentation of zneupd.\n\n');
    end

    if (info_zneupd==1) then
      printf('\nMaximum number of iterations reached.\n\n');
    elseif (info_zneupd==3) then
      printf('\nNo shifts could be applied during implicit Arnoldi update, try
    end
  end
end

// Done with program znsimp.
printf('\nZNSIMP\n');
printf('======\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);
```

```
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam
printf('The number of OP*x is %d\n', iparam(9));
printf('The convergence criterion is %d\n', tol);
```

# See Also

dnaupd, dneupd, zneupd

# Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

# Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.

2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.

3. B.N. Parlett and Y. Saad, "Complex Shift and Invert Strategies for Real Matrices", Linear Algebra and its Applications, vol 88/89, pp 575-595, (1987).

# Used Functions

Based on ARPACK routine znaupd

# Name

zneupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute approximations to the converged approximations to eigenvalues of A*z = lambda*B*z

```
[D,Z,RESID,IPARAM,IPNTR,WORKD,WORKL,RWORK,INFO] = zneupd(RVEC,HOWMANY,SELECT,D,Z
                                        BMAT,N,WHICH,NEV,TOL,RESID,NCV,V,IPARAM,I
```

# Parameters

RVEC

Integer (INPUT)

Specifies whether a basis for the invariant subspace corresponding to the converged Ritz value approximations for the eigenproblem A*z = lambda*B*z is computed.

- RVEC = 0 Compute Ritz values only.
- RVEC = 1 Compute Ritz vectors or Schur vectors. See Remarks below.

HOWMANY

Character (INPUT)

Specifies the form of the basis for the invariant subspace corresponding to the converged Ritz values that is to be computed.

- 'A': Compute NEV Ritz vectors;
- 'P': Compute NEV Schur vectors;
- 'S': compute some of the Ritz vectors, specified by the integer array SELECT.

SELECT

Integer array of dimension NCV. (INPUT)

If HOWMANY = 'S', SELECT specifies the Ritz vectors to be computed. To select the Ritz vector corresponding to a Ritz value D(j), SELECT(j) must be set to 1.

If HOWMANY = 'A' or 'P', SELECT need not be initialized but it is used as internal workspace.

D

Complex*16 array of dimension NEV+1. (OUTPUT)

On exit, D contains the Ritz approximations to the eigenvalues lambda for A*z = lambda*B*z.

Z

Complex*16 N by NEV array (OUTPUT)

On exit,

If RVEC = 1 and HOWMANY = 'A', then the columns of Z represents approximate eigenvectors (Ritz vectors) corresponding to the NCONV=IPARAM(5) Ritz values for eigensystem A*z = lambda*B*z.

If RVEC = 0 or HOWMANY = 'P', then Z is NOT REFERENCED.

NOTE: If if RVEC = 1 and a Schur basis is not required, the array Z may be set equal to first NEV+1 columns of the Arnoldi basis array V computed by ZNAUPD. In this case the Arnoldi basis will be destroyed and overwritten with the eigenvector basis.

SIGMA

Complex*16 (INPUT)

If IPARAM(7) = 3 then SIGMA represents the shift.

Not referenced if IPARAM(7) = 1 or 2.

WORKev
Complex*16 work array of dimension 2*NCV. (WORKSPACE)

NOTE: The remaining arguments BMAT, N, WHICH, NEV, TOL, RESID, NCV, V, IPARAM, IPNTR, WORKD, WORKL, LWORKL, RWORK, INFO must be passed directly to ZNEUPD following the last call to ZNAUPD.

These arguments MUST NOT BE MODIFIED between the the last call to ZNAUPD and the call to ZNEUPD.

Three of these parameters (V, WORKL and INFO) are also output parameters.

V
Complex*16 N by NCV array. (INPUT/OUTPUT)

Upon INPUT: the NCV columns of V contain the Arnoldi basis vectors for OP as constructed by ZNAUPD.

Upon OUTPUT: If RVEC = 1 the first NCONV=IPARAM(5) columns contain approximate Schur vectors that span the desired invariant subspace.

NOTE: If the array Z has been set equal to first NEV+1 columns of the array V and RVEC=1 and HOWMANY= 'A', then the Arnoldi basis held by V has been overwritten by the desired Ritz vectors. If a separate array Z has been passed then the first NCONV=IPARAM(5) columns of V will contain approximate Schur vectors that span the desired invariant subspace.

WORKL
Double precision work array of length LWORKL. (OUTPUT/WORKSPACE)

WORKL(1:ncv*ncv+2*ncv) contains information obtained in znaupd. They are not changed by zneupd.

WORKL(ncv*ncv+2*ncv+1:3*ncv*ncv+4*ncv) holds the untransformed Ritz values, the untransformed error estimates of the Ritz values, the upper triangular matrix for H, and the associated matrix representation of the invariant subspace for H.

Note: IPNTR(9:13) contains the pointer into WORKL for addresses of the above information computed by zneupd.

- IPNTR(9): pointer to the NCV RITZ values of the original system.
- IPNTR(10): Not used
- IPNTR(11): pointer to the NCV corresponding error estimates.
- IPNTR(12): pointer to the NCV by NCV upper triangular Schur matrix for H.
- IPNTR(13): pointer to the NCV by NCV matrix of eigenvectors of the upper Hessenberg matrix H. Only referenced by zneupd if RVEC = 1 See Remark 2 below.

INFO
Integer. (OUTPUT)

Error flag on output.

- 0: Normal exit.
- 1: The Schur form computed by LAPACK routine csheqr could not be reordered by LAPACK routine ztrsen. Re-enter subroutine zneupd with IPARAM(5)=NCV and increase the size of the array D to have dimension at least dimension NCV and allocate at least NCV columns for Z.

  NOTE: Not necessary if Z and V share the same space. Please notify the authors if this error occurs.

- -1: N must be positive.
- -2: NEV must be positive.
- -3: NCV-NEV >= 1 and less than or equal to N.
- -5: WHICH must be one of 'LM', 'SM', 'LR', 'SR', 'LI', 'SI'
- -6: BMAT must be one of 'I' or 'G'.
- -7: Length of private work WORKL array is not sufficient.
- -8: Error return from LAPACK eigenvalue calculation. This should never happened.
- -9: Error return from calculation of eigenvectors. Informational error from LAPACK routine ztrevc
- -10: IPARAM(7) must be 1,2,3
- -11: IPARAM(7) = 1 and BMAT = 'G' are incompatible.
- -12: HOWMANY = 'S' not yet implemented
- -13: HOWMANY must be one of 'A' or 'P' if RVEC = .true.
- -14: ZNAUPD did not find any eigenvalues to sufficient accuracy.
- -15: ZNEUPD got a different count of the number of converged Ritz values than ZNAUPD got. This indicates the user probably made an error in passing data from ZNAUPD to ZNEUPD or that the data was modified before entering ZNEUPD

# Description

This subroutine returns the converged approximations to eigenvalues of A*z = lambda*B*z and (optionally):

1. The corresponding approximate eigenvectors;
2. An orthonormal basis for the associated approximate invariant subspace;
3. Both.

There is negligible additional cost to obtain eigenvectors. An orthonormal basis is always computed.

There is an additional storage cost of n*nev if both are requested (in this case a separate array Z must be supplied).

The approximate eigenvalues and eigenvectors of A*z = lambda*B*z are derived from approximate eigenvalues and eigenvectors of of the linear operator OP prescribed by the MODE selection in the call to ZNAUPD.

ZNAUPD must be called before this routine is called.

These approximate eigenvalues and vectors are commonly called Ritz values and Ritz vectors respectively. They are referred to as such in the comments that follow.

The computed orthonormal basis for the invariant subspace corresponding to these Ritz values is referred to as a Schur basis.

The definition of OP as well as other terms and the relation of computed Ritz values and vectors of OP with respect to the given problem A*z = lambda*B*z may be found in the header of ZNAUPD. For a brief description, see definitions of IPARAM(7), MODE and WHICH in the documentation of ZNAUPD.

# Remarks

1. Currently only HOWMNY = 'A' and 'P' are implemented.

2. Schur vectors are an orthogonal representation for the basis of Ritz vectors. Thus, their numerical properties are often superior.

   If RVEC = 1 then the relationship

   A * V(:,1:IPARAM(5)) = V(:,1:IPARAM(5)) * T,

   and

   transpose( V(:,1:IPARAM(5)) ) * V(:,1:IPARAM(5)) = I

   are approximately satisfied. Here T is the leading submatrix of order IPARAM(5) of the upper triangular matrix stored workl(ipntr(12)).

# Example

```
// The following sets dimensions for this problem.

nx    = 10;

nev   = 3;
ncv   = 6;
bmat  = 'I';
which = 'LM';

maxiter = 10;

// Local Arrays

iparam  = zeros(11,1);
ipntr   = zeros(14,1);
_select = zeros(ncv,1);
d       = zeros(nev+1,1);
resid   = zeros(nx,1) + 0*%i;
v       = zeros(nx,ncv) + 0*%i;
workd   = zeros(3*nx+1,1) + 0*%i;
workev  = zeros(3*ncv,1);
rwork   = zeros(ncv,1);
workl   = zeros(3*ncv*ncv+5*ncv,1) + 0*%i;

// Build the complex test matrix
A            = diag(10*ones(nx,1)+%i*ones(nx,1));
A(1:$-1,2:$) = A(1:$-1,2:$) + diag(6*ones(nx-1,1));
A(2:$,1:$-1) = A(2:$,1:$-1) + diag(-6*ones(nx-1,1));

tol    = 0;
ido    = 0;

ishfts = 1;
maxitr = 300;
mode1  = 1;

iparam(1) = ishfts;
```

```
iparam(3) = maxitr;
iparam(7) = mode1;

sigma = 0; // the real part of the shift

// M A I N   L O O P (Reverse communication)

iter = 0;
while(iter<maxiter)
  info_znaupd = 0;
  iter = iter + 1;
  // Repeatedly call the routine ZNAUPD and take actions indicated by parameter
  // either convergence is indicated or maxitr has been exceeded.

  [ido,resid,v,iparam,ipntr,workd,workl,info_znaupd] = znaupd(ido,bmat,nx,which

  if (ido==99) then
    // BE CAREFUL: DON'T CALL zneupd IF ido == 99 !!
    break;
  end

  if (ido==-1 | ido==1) then
    // Perform matrix vector multiplication
    workd(ipntr(2)+1:ipntr(2)+nx) = A*workd(ipntr(1)+1:ipntr(1)+nx);
    // L O O P   B A C K to call ZNAUPD again.
    continue
  end

  if (info_znaupd < 0) then
    printf('\nError with znaupd, info = %d\n',info_znaupd);
    printf('Check the documentation of znaupd\n\n');
  else
    // Post-Process using ZNEUPD.

    rvec    = 1;
    howmany = 'A';

    info_zneupd = 0;

    [d,d(1,2),v,resid,v,iparam,ipntr,workd,workl,info_zneupd] = zneupd(rvec,how
                                                                    bmat,nx,
                                                                    iparam,i


    if (info_zneupd~=0) then
      printf('\nError with zneupd, info = %d\n', info_zneupd);
      printf('Check the documentation of zneupd.\n\n');
    end

    if (info_zneupd==1) then
      printf('\nMaximum number of iterations reached.\n\n');
    elseif (info_zneupd==3) then
      printf('\nNo shifts could be applied during implicit Arnoldi update, try
    end
  end
end

// Done with program znsimp.
printf('\nZNSIMP\n');
```

```
printf('======\n');
printf('\n');
printf('Iterations is %d\n', iter);
printf('Size of the matrix is %d\n', nx);
printf('The number of Ritz values requested is %d\n', nev);
printf('The number of Arnoldi vectors generated (NCV) is %d\n', ncv);
printf('What portion of the spectrum: %s\n', which);
printf('The number of Implicit Arnoldi update iterations taken is %d\n', iparam
printf('The number of OP*x is %d\n', iparam(9));
printf('The convergence criterion is %d\n', tol);
```

# See Also

znaupd, dnaupd, dneupd

# Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

# Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.

2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.

3. B.N. Parlett and Y. Saad, "Complex Shift and Invert Strategies for Real Matrices", Linear Algebra and its Applications, vol 88/89, pp 575-595, (1987).

# Used Functions

Based on ARPACK routine zneupd

# Parte XXXVI. Funções de Compatibilidade

# Name

asciimat — string matrix ascii conversions

```
a=asciimat(txt)
txt=asciimat(a)
```

# Parameters

txt
: character string or matrix of strings.

a
: vector or matrix of integer ASCII codes

# Description

This function convert Scilab string to ASCII code or a matrix of ASCII code to Scilab string. Output is a matrix having same number of lines than input, what is not the case with `ascii`.

# See Also

ascii

# Authors

V.C.

# Name

firstnonsingleton — Finds first dimension which is not 1

```
[dim]=firstnonsingleton(A[,opt])
```

# Parameters

dim
: first dimension of A which is not 1

A
: a variable of any Scilab data type

opt
: a character string giving the type of output we want

  "num"
  : returned value is a numerical value

  "str"
  : returned value is a character string if possible ("r" instead of 1 and "c" instead of 2)

# Description

This function is used by `mfile2sci` to emulate Matlab behavior under Scilab, particularly for functions which treat the values along the first non-singleton dimension of A in Matlab while in Scilab they treat all values of A.

# Examples

```
A = [1 2 3;4 5 6];
// Scilab max
M = max(A)
// Matlab max emulation
M = max(A,firstnonsingleton(A))
```

# Authors

V.C.

# Nom

makecell — Creates a cell array.

```
s = makecell(dims,a1,a2,...an)
```

# Parameters

dims
: a vector with positive integer elements, the cell array dimension

a1,a2,...,an
: Sequence of Scilab variables, `n` must be equal to `prod(dims)`

s
: resulting cell array

# Description

`s = makecell(dims,a1,a2,...an)` creates a cell array of dimensions given by `dims` with the given input arguments. The `ai` are stored along the last dimension first.

# Examples

```
makecell([2,3],1,2,3,'x','y','z')
```

# See Also

cell

# Authors

Farid Belhacen, INRIA

# Name

mstr2sci — character string matrix to character matrix conversion

```
a=mstr2sci(txt)
```

# Parameters

txt
: character string or string matrix

a
: character vector or matrix

# Description

This function converts a Scilab character string to a vector of characters. Result is the Scilab equivalent for a Matlab string.

Caution: `mstr2sci` has not to be used for hand coded functions.

# See Also

Matlab-Scilab_character_strings

# Authors

V.C.

# Name

mtlb_0 — Matlab non-conjugate transposition emulation function

# Description

Matlab and Scilab non-conjugate transposition behave differently in some particular cases:

- With character strings operands: The `.'` operator is used to transpose whole character strings in Scilab while Matlab realizes the transposition of each character.

The function `mtlb_0(A)` is used by `mfile2sci` to replace `A.'` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_0` calls:

- If `A` is not a character string matrix `mtlb_0(A)` may be replaced by `A.'`

Caution: `mtlb_0` has not to be used for hand coded functions.

# See Also

Matlab-Scilab_character_strings

# Authors

V.C.

# Name

mtlb_a — Matlab addition emulation function

# Description

Matlab and Scilab addition behave differently in some particular cases:

- With character string operands: The `+` operator is used into Scilab to catenate character strings, while Matlab realizes the sum of the operands ASCII codes.
- With empty matrix: In Scilab, if one of the operands is an empty matrix the result of the addition is the other operand. In Matlab if one of the operand is an empty matrix the result of the addition should be an error or an empty matrix.

The function `mtlb_a(A,B)` is used by `mfile2sci` to replace `A+B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_a` calls:

- If `A` and `B` are character strings, `mtlb_a(A,B)` may be replaced by `asciimat(A)+asciimat(B)`
- If both `A` and `B` are not empty matrices `mtlb_a(A,B)` may be replaced by `A+B`, else `mtlb_a(A,B)` may be replaced by `[]`.
- If mtlb_mode==%T, then `mtlb_a(A,B)` may be replaced by `A+B` in any case where `A` and `B` are not character string matrices.

Caution: `mtlb_a` has not to be used for hand coded functions.

# See Also

mtlb_mode

# Authors

V.C.

# Name

mtlb_all — Matlab all emulation function

# Description

Matlab `all` and Scilab `and` behave differently in some particular cases:

- When used with one input (`all(A)`), Matlab `all` treats the values along the first non-singleton dimension of A as vectors while Scilab `and` treats all A values.
- When used with two inputs (`all(A,dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of A while Scilab returns an error message in this case.

The function `R = mtlb_all(A[,dim])` is used by `mfile2sci` to replace `all(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_all` calls:

- If `A` is a scalar or a vector `R = mtlb_all(A)` may be replaced by `R = and(A)`
- If `A` is a matrix `R = mtlb_all(A)` may be replaced by `R = and(A,1)`
- If `A` is an hypermatrix `R = mtlb_all(A)` may be replaced by `R = and(A,firstnonsingleton(A))` or by `R = and(A,user_defined_value)` if the first non-singleton dimensions of A is known.
- If `dim` is less equal to the number of dimensions of A `R = mtlb_all(A,dim)` may be replaced by `R = and(A,dim)`
- If `dim` is greater than then number of dimensions of A `R = mtlb_all(A,dim)` may be replaced by `R = A<>0` if A is not an empty matrix or by `R = A` if A is an empty matrix.

Caution: `mtlb_all` has not to be used for hand coded functions.

# See Also

firstnonsingleton

# Authors

V.C.

# Name

mtlb_any — Matlab any emulation function

# Description

Matlab `any` and Scilab `or` behave differently in some particular cases:

- When used with one input (`any(A)`), Matlab `any` treats the values along the first non-singleton dimension of A as vectors while Scilab `or` treats all A values.
- When used with two inputs (`any(A,dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of A while Scilab returns an error message in this case.

The function `R = mtlb_any(A[,dim])` is used by `mfile2sci` to replace `any(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_any` calls:

- If `A` is a scalar or a vector `R = mtlb_any(A)` may be replaced by `R = or(A)`
- If `A` is a matrix `R = mtlb_any(A)` may be replaced by `R = or(A,1)`
- If `A` is an hypermatrix `R = mtlb_any(A)` may be replaced by `R = or(A,firstnonsingleton(A))` or by `R = or(A,user_defined_value)` if the first non-singleton dimensions of A is known.
- If `dim` is less equal to the number of dimensions of A `R = mtlb_any(A,dim)` may be replaced by `R = or(A,dim)`
- If `dim` is greater than then number of dimensions of A `R = mtlb_any(A,dim)` may be replaced by `R = A<>0` if A is not an empty matrix or by `R = A` if A is an empty matrix.

Caution: `mtlb_any` has not to be used for hand coded functions.

# See Also

firstnonsingleton

# Authors

V.C.

# Name

mtlb_axis — Matlab axis emulation function

```
mtlb_axis(X)
mtlb_axis(st)
mtlb_axis(axeshandle, ...)
[mode,visibility,direction]=mtlb_axis('state')
```

# Parameters

X
: a vector of reals ([xmin xmax ymin ymax] or [xmin xmax ymin ymax zmin zmax])

st
: a string ('auto', 'manual', 'tight', 'ij', 'xy', 'equal', 'square', 'vis3d', 'off', 'on')

axeshandle
: handle of the current axes entity

# Description

Matlab `axis` have not a Scilab equivalent function.

The function `mtlb_axis(...)` is used by `mfile2sci` to replace `axis(...)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time (axis `fill`, axis `image` and axis `normal` are not implemented). If you want to have a more efficient code it is possible to replace `mtlb_axis` call by `get(axeshandle, prop)` call (prop is a axes property, see axis_properties)

Caution: `mtlb_axis` has not to be used for hand coded functions.

# Authors

F.B.

# Name

mtlb_beta — Matlab beta emulation function

# Description

Matlab and Scilab `beta` behave differently in some particular cases:

- With inputs having different sizes: Matlab `beta` input parameters must have the same size unless one of them is a scalar. In Scilab `beta` input parameters must have the same size even if one of them is a scalar.

The function `mtlb_beta(A,B)` is used by `mfile2sci` to replace `beta(A,B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_beta` calls:

- If `A` is a scalar but not `B` `Y = mtlb_beta(A,B)` may be replaced by `C=B;C(:)=A;Y = mtlb_beta(C,B);`
- If `B` is a scalar but not `A` `Y = mtlb_beta(A,B)` may be replaced by `C=A;C(:)=B;Y = mtlb_beta(A,C);`

Caution: `mtlb_beta` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_box — Matlab box emulation function

# Description

There is no Scilab equivalent function for Matlab `box` but it can be easyly replaced.

The function `mtlb_box([axes_handle[,val]])` is used by `mfile2sci` to replace `box([axes_handle[,va]]l)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_box` calls:

- When called without input parameters, `mtlb_box()` may be replaced by `a=gca();if a.box=="on" then a.box="off";else a.box="on";end;`
- If `axes_handle` is a character string, `mtlb_box(axes_handle)` may be replaced by `a=gca();a.box=convstr(axes_handle,"l");`
- If `axes_handle` is a graphic handle `mtlb_box(axes_handle)` may be replaced by `if axes_handle.box=="on" then axes_handle.box="off";else axes_handle.box="on";end;`
- If `axes_handle` is a graphic handle and `val` is a character string `mtlb_box(axes_handle,val)` may be replaced by `axes_handle.box=convstr(val,"l");`

Caution: `mtlb_box` has not to be used for hand coded functions.

# See Also

axes_properties

# Authors

V.C.

# Name

mtlb_close — Matlab close emulation function

# Description

Scilab equivalent for Matlab `close` is different according to the current figure type (uicontrol or graphic one).

- When current figure is a uicontrol one: Scilab equivalent for Matlab `close` is `close`
- When current figure is a graphic one: Scilab equivalent for Matlab `close` is `xdel` or `delete`
- In Scilab we do not get an error status.

The function `mtlb_close([h])` is used by `mfile2sci` to replace `close([h])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_close` calls:

- If `h` is a uicontrol figure `mtlb_close(h)` may be replaced by `close(h)`
- If `h` is a graphic figure `mtlb_close(h)` may be replaced by `delete(h)`

Caution: `mtlb_close` has not to be used for hand coded functions.

# See Also

xdel , delete , winsid , close

# Authors

V.C.

# Name

mtlb_colordef — Matlab colordef emulation function

# Description

There is no Scilab equivalent function for Matlab `colordef` but there are equivalent instructions.

The function `h = mtlb_colordef(color_option)` or `h = mtlb_colordef(fig,color_option)` is used by `mfile2sci` to replace `colordef(color_option)` or `colordef(fig,color_option)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_colordef` calls:

- When called with one input parameter, if `color_option` is equal to "black" or "none" `mtlb_colordef(color_option)` may be replaced by `fig = gcf();fig.background = -1;`
- When called with one input parameter, if `color_option` is equal to "white" `mtlb_colordef(color_option)` may be replaced by `fig = gcf();fig.background = -2;`
- When called with two input parameters, if `fig` is a graphic handle and `color_option` is equal to "black" or "none" `mtlb_colordef(color_option)` may be replaced by `fig.background = -1;`
- When called with two input parameters, if `fig` is a graphic handle and `color_option` is equal to "white" `mtlb_colordef(color_option)` may be replaced by `fig.background = -2;`
- When called with two input parameters, if `fig` is equal to "new" and `color_option` is equal to "black" or "none" `mtlb_colordef(color_option)` may be replaced by `fig = scf(max(winsid())+1);fig.background = -1;`
- When called with two input parameters, if `fig` is equal to "new" and `color_option` is equal to "white" `mtlb_colordef(color_option)` may be replaced by `fig = scf(max(winsid())+1);fig.background = -2;`
- When called with one output parameter `h`, just add `h = fig;` after equivalent instructions.

Caution: `mtlb_colordef` has not to be used for hand coded functions.

# See Also

figure_properties

# Authors

V.C.

# Name

mtlb_conv — Matlab conv emulation function

# Description

Matlab `conv` and Scilab `convol` behave differently in some particular cases:

- With column vector inputs: if at least input is a column vector Matlab `conv` returns a column vector but Scilab `convol` always returns a row vector.

The function `mtlb_conv(u,v)` is used by `mfile2sci` to replace `conv(u,v)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_conv` calls:

- If `u` and `v` are row vectors, `mtlb_conv(u,v)` may be replaced by `convol(u,v)`
- If `u` or `v` is a column vector, `mtlb_conv(u,v)` may be replaced by `convol(u,v).'`
- If `u` and `v` are column vectors, `mtlb_conv(u,v)` may be replaced by `convol(u,v).'`

Scilab `convol` sometimes returns values that may be rounded using `clean` to have a closer result from Matlab one.

Caution: `mtlb_conv` has not to be used for hand coded functions.

# See Also

clean

# Authors

V.C.

# Name

mtlb_cumprod — Matlab cumprod emulation function

# Description

Matlab and Scilab `cumprod` behave differently in some particular cases:

- When used with one input (`cumprod(A)`), Matlab `cumprod` treats the values along the first non-singleton dimension of A as vectors while Scilab `cumprod` treats all A values.
- When used with two inputs (`cumprod(A,dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of A while Scilab returns an error message in this case.

The function `R = mtlb_cumprod(A[,dim])` is used by `mfile2sci` to replace `cumprod(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_cumprod` calls:

- If `dim` is less equal to the number of dimensions of A `R = mtlb_cumprod(A,dim)` may be replaced by `R = cumprod(A,dim)`
- If `dim` is greater than then number of dimensions of A `R = mtlb_cumprod(A,dim)` may be replaced by `R = A`.

Caution: `mtlb_cumprod` has not to be used for hand coded functions.

# See Also

firstnonsingleton

# Authors

V.C.

# Name

mtlb_cumsum — Matlab cumsum emulation function

# Description

Matlab and Scilab `cumsum` behave differently in some particular cases:

- When used with one input (`cumsum(A)`), Matlab `cumsum` treats the values along the first non-singleton dimension of A as vectors while Scilab `cumsum` treats all A values.
- When used with two inputs (`cumsum(A,dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of A while Scilab returns an error message in this case.

The function `R = mtlb_cumsum(A[,dim])` is used by `mfile2sci` to replace `cumsum(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_cumsum` calls:

- If `dim` is less equal to the number of dimensions of A `R = mtlb_cumsum(A,dim)` may be replaced by `R = cumsum(A,dim)`
- If `dim` is greater than then number of dimensions of A `R = mtlb_cumsum(A,dim)` may be replaced by `R = A`.

Caution: `mtlb_cumsum` has not to be used for hand coded functions.

# See Also

firstnonsingleton

# Authors

V.C.

# Name

mtlb_dec2hex — Matlab dec2hex emulation function

# Description

Matlab and Scilab `dec2hex` behave differently in some particular cases:

- With empty matrix: Matlab `dec2hex` returns "" but in Scilab you get [].
- With complex inputs: Matlab `dec2hex` automatically removes complex part of inputs but not Scilab one.
- Matlab `dec2hex` always returns a row vector but in Scilab `dec2hex` returns a value which have the same size as the input.
- Matlab `dec2hex` can have two inputs but not Scilab one.

The function `mtlb_dec2hex(D[,N])` is used by `mfile2sci` to replace `dec2hex(D[,N])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_dec2hex` calls:

- If `D` is not an empty matrix, `mtlb_dec2hex(D)` may be replaced by `matrix(dec2hex(real(D)),-1,1)` if `D` is complex and by `matrix(dec2hex(D),-1,1)` else.

Caution: `mtlb_dec2hex` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_delete — Matlab delete emulation function

# Description

The function `mtlb_delete(A)` is used by `mfile2sci` to replace `delete(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_delete` calls:

- If `A` is a character string `mtlb_delete(A)` may be replaced by `mdelete(A)`
- If `A` is a graphic handle `mtlb_delete(A)` may be replaced by `delete(A)`

Caution: `mtlb_delete` has not to be used for hand coded functions.

# See Also

mdelete

# Authors

V.C.

# Name

mtlb_diag — Matlab diag emulation function

# Description

Matlab and Scilab `diag` behave differently in some particular cases:

- With character string matrices: Scilab `diag` function considers each character string as an object while Matlab considers each character individually.

The function `y = mtlb_diag(x[,dim])` is used by `mfile2sci` to replace `y = diag(x[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_diag` calls:

- If `x` is not a character string matrix `y = mtlb_diag(x[,dim])` may be replaced by `y = diag(x[,dim])`

Caution: `mtlb_diag` has not to be used for hand coded functions.

# See Also

Matlab-Scilab_character_strings

# Authors

V.C.

# Name

mtlb_diff — Matlab diff emulation function

# Description

Matlab and Scilab `diff` behave differently in some particular cases:

- With two input parameters: Scilab `diff` considers all values of first input as a vector what Matlab does not.
- With three input parameters: If dimension of first input along dimension given by third parameter reaches 1 before n interations have been made,Matlab switches to next non-singleton dimension what Scilab does not.

The function `mtlb_diff(A[,n[,dim]])` is used by `mfile2sci` to replace `diff(A[,n[,dim]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_diff` calls:

- With two inputs, if `A` is a scalar or a vector `mtlb_diff(A,n)` may be replaced by `diff(A,n)`
- With three inputs, if size of A along dimension given by `dim` can not reach 1 `mtlb_diff(A,n,dim)` may be replaced by `diff(A,n,dim)`

Caution: `mtlb_diff` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_dir — Matlab dir emulation function

# Description

Matlab and Scilab `dir` behave differently:

- When result is stored in a variable: Matlab `dir` returns a structure but Scilab `dir` returns a 'dir' tlist, so data can not be extracted in the same way.

The function `mtlb_dir([path])` is used by `mfile2sci` to replace `dir([path])` when output is stored in a variable. There is no replacement possibility for it, else (when `mtlb_dir` is replaced by `dir`) data can not be extracted like in Matlab. For example, Scilab equivalent for Matlab `L=dir;file_name=L(1).name;` is `L=dir();file_name=L.name(1);`.

Caution: `mtlb_dir` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_double — Matlab double emulation function

# Description

Matlab and Scilab `double` behave differently in some particular cases:

- With character string input: Scilab `double` does not work with this type of input while Matlab `double` returns a matrix of ASCII codes.
- With boolean input: Scilab `double` does not work with this type of input while Matlab `double` returns a matrix of double values.

The function `mtlb_double(A)` is used by `mfile2sci` to replace `double(A)` when it was not possible to know what were theinput while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_double` calls:

- If `A` is a character string, `mtlb_double(A)` may be replaced by `asciimat(A)`
- If `A` is a boolean matrix, `mtlb_double(A)` may be replaced by `bool2s(A)`
- If `A` is a double matrix, `mtlb_double(A)` may be replaced by `A`

Caution: `mtlb_double` has not to be used for hand coded functions.

# See Also

asciimat , bool2s

# Authors

V.C.

# Name

mtlb_e — Matlab extraction emulation function

# Description

Matlab and Scilab extraction behave differently in some particular cases:

- When extracting data from a matrix with a vector as index: Matlab returns a row vector and Scilab returns a column vector.
- When extracting data from a character string matrix: due to the fact that character string matrices in Matlab can be adressed as any other matrix (each character can be adressed), extraction in such a type of matrix does not differ from other. But in Scilab it can't be done so and `part` function has to be used.

The function `mtlb_e(B,k)` is used by `mfile2sci` to replace `A=B(k)` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_e` calls:

- If `B` is a vector `A=mtlb_e(B,k)` may be replaced by `A=B(k)`
- If `B` is a matrix `A=mtlb_e(B,k)` may be replaced by `A=B(k).'`
- If `B` is a character string matrix and k is a scalar or a vector `A=mtlb_e(B,k)` may be replaced by `A=part(B,k)`

Caution: `mtlb_e` has not to be used for hand coded functions.

# See Also

Matlab-Scilab_character_strings , part

# Authors

V.C.

# Name

mtlb_echo — Matlab echo emulation function

# Description

There is no equivalent for Matlab `echo` in Scilab but some cases can be replaced by calls to Scilab `mode`:

- `echo`: is equivalent to Scilab `mode(abs(mode()-1))` for scripts and non-compiled functions
- `echo on`: is equivalent to Scilab `mode(1)` for scripts and non-compiled functions
- `echo off`: is equivalent to Scilab `mode(0)`

The function `mtlb_echo(arg1[,arg2])` is used by `mfile2sci` to replace `echo arg1 [arg2]` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_echo` calls:

- If `arg1` is equal to "on" `mtlb_echo(arg1)` may be replaced by `mode(1)`
- If `arg1` is equal to "off" `mtlb_echo(arg1)` may be replaced by `mode(0)`

Caution: `mtlb_echo` has not to be used for hand coded functions.

# See Also

mode

# Authors

V.C.

# Name

mtlb_eval — Matlab eval emulation function

# Description

Scilab equivalent for Matlab `eval` is different according to its inputs and outputs

The function `mtlb_eval(str1,str2)` is used by `mfile2sci` to replace `eval` because it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_eval` calls:

- When called with one input and no output, `mtlb_eval(str1)` may be replaced by `evstr(str1)` if `str1` is a valid Scilab expression or by `execstr(str1)` if `str1` is a valid Scilab instruction.
- When called with one input and one output, `val=mtlb_eval(str1)` may be replaced by `val=evstr(str1)` if `str1` is a valid Scilab instruction.
- When called with two inputs and no output, `mtlb_eval(str1,str2)` may be replaced by: `if execstr(str1,"errcatch")<>0 then execstr(str2);end` if `str1` and `str2` are valid Scilab instructions.
- When called with more than one output and one input, `[val1,val2,...]=mtlb_eval(str1)` may be replaced by `execstr("[val1,val2,...]"+str1)` if `str1` is a valid Scilab instruction.
- When called with two inputs and more than one output, `[val1,val2,...]=mtlb_eval(str1,str2)` may be replaced by:

  ```
  if execstr("[val1,val2,...]"+str1,"errcatch")<>0 then
    execstr("[val1,val2,...]"+str2);
  end
  ```

  if `str1` and `str2` are valid Scilab instructions.

Caution: `mtlb_eval` has not to be used for hand coded functions.

# See Also

evstr , execstr

# Authors

V.C.

# Name

mtlb_exist — Matlab exist emulation function

# Description

There is no Scilab equivalent for Matlab `exist` except when input is a variable. Scilab `mtlb_exist` is a partial emulation of of this function.

The function `r = mtlb_exist(nam[,tp])` is used by `mfile2sci` to replace `exist(nam[,tp])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_exist` calls:

- When called with one input and if `nam` is a variable name, `mtlb_exist(nam)` may be replaced by `exists(nam)`

Caution: `mtlb_exist` has not to be used for hand coded functions.

# See Also

exists

# Authors

V.C.

# Name

mtlb_eye — Matlab eye emulation function

# Description

Matlab and Scilab `eye` behave differently in some particular cases:

- With a scalar input: Matlab `eye` returns a n x n matrix while Scilab returns a 1.

The function `mtlb_eye(A)` is used by `mfile2sci` to replace `eye(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_eye` calls:

- If `A` is a scalar `mtlb_eye(A)` may be replaced by `eye(A,A)`
- If `A` is not a scalar `mtlb_eye(A)` may be replaced by `eye(A)`

Caution: `mtlb_eye` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_false — Matlab false emulation function

# Description

There is no Scilab equivalent for Matlab `false`. However, Scilab `zeros` returns a result interpreted in an equivalent way for Scilab.

Matlab `false` and Scilab `zeros` behave differently in some particular cases:

- With a scalar input: Matlab `false` returns a n x n matrix of zeros while Scilab `zeros` returns a 0.

The function `mtlb_false(A)` is used by `mfile2sci` to replace `false(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_false` calls:

- If `A` is a scalar `mtlb_false(A)` may be replaced by `zeros(A,A)`
- If `A` is not a scalar `mtlb_false(A)` may be replaced by `zeros(A)`

Caution: `mtlb_false` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_fft — Matlab fft emulation function

# Description

Matlab and Scilab `fft` behave differently in some particular cases:

- With one input parameter: If input is a scalar or vector Scilab equivalent for Matlab `fft` is `fft(...,-1)` else if input is a matrix Scilab equivalent for Matlab `fft` is `fft(...,-1,2,1)`

The function `mtlb_fft(X[,n,[job]])` is used by `mfile2sci` to replace `fft(X[,n,[job]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fft` calls:

- If `X` is a scalar or a vector `mtlb_fft(X,-1)` may be replaced by `fft(X,-1)`
- If `X` is a matrix `mtlb_fft(X,-1)` may be replaced by `fft(X,-1,2,1)`

Caution: `mtlb_fft` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_fftshift — Matlab fftshift emulation function

# Description

Matlab and Scilab `fftshift` behave differently in some particular cases:

- With character string input: due to the fact that character strings are not considered in the same way in Matlab and Scilab, results can be different for this kind of input.
- With two inputs: Matlab `fftshift` can work even if `dim` parameter is greater than number of dimensions of first input.

The function `mtlb_fftshift(A[,dim])` is used by `mfile2sci` to replace `fftshift(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fftshift` calls:

- If `A` is not a character string matrix, `mtlb_fftshift(A)` may be replaced by `fftshift(A)`
- If `A` is not a character string matrix and `dim` is not greater than `size(size(a),"*")`, `mtlb_fftshift(A,dim)` may be replaced by `fftshift(A,dim)`

Caution: `mtlb_fftshift` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_find — Matlab find emulation function

# Description

Matlab and Scilab `find` behave differently in some particular cases:

- With column vectors and matrices as input: Matlab `find` returns column vectors while Scilab returns row vectors.
- When called with three outputs: Matlab `find` can have three outputs but Scilab not.

The function `[i[,j[,v]]] = mtlb_find(B)` is used by `mfile2sci` to replace `[i[,j[,v]]] = find(B)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_find` calls:

- When called with one output, if `B` is a row vector `i = mtlb_find(B)` may be replaced by `i = find(B)`
- When called with one output, if `B` is not a row vector `i = mtlb_find(B)` may be replaced by `i = matrix(find(B),-1,1)`
- When called with two outputs, if `B` is a row vector `[i,j] = mtlb_find(B)` may be replaced by `[i,j] = find(B)`
- When called with two outputs, if `B` is not a row vector `[i,j] = mtlb_find(B)` may be replaced by `[i,j] = find(B);i = matrix(i,-1,1);j = matrix(j,-1,1);`

Caution: `mtlb_find` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_findstr — Matlab findstr emulation function

# Description

There is no Scilab equivalent for Matlab `findstr`.

The function `mtlb_findstr(A,B)` is used by `mfile2sci` to replace `findstr(A,B)` when it was not possible to know what were the operands/inputs[CUSTOM] while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_findstr` calls:

- If `A` is always longer than `B` (i.e. `findstr` can be replaced by `strfind` in Matlab, `mtlb_findstr(A,B)` may be replaced by `strindex(A,B)`

Caution: `mtlb_findstr` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_fliplr — Matlab fliplr emulation function

# Description

Matlab and Scilab `fliplr` behave differently in some particular cases:

- With character string matrices: due to te fact that Scilab and Matlab do not consider character string matrices in the same way, result can be different for input of this type.

The function `mtlb_fliplr(A)` is used by `mfile2sci` to replace `fliplr(A)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fliplr` calls:

- If `A` is not a character string matrix, `mtlb_fliplr(A)` may be replaced by `A(:,$:-1:1)`

Caution: `mtlb_fliplr` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_fopen — Matlab fopen emulation function

# Description

Matlab `fopen` and Scilab `mopen` behave differently in some particular cases:

- Scilab function returns no usable error message
- Scilab file identified does not exist in case of error but Matlab one is set to `-1`.
- Matlab function can work with inputs which do not exist in Scilab such as permission options...

The function `mtlb_fopen(filename,permission)` is used by `mfile2sci` to replace `mopen(filename,permission)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fopen` calls:

- If error message is not used and no error can occurs, `mtlb_fopen(filename,permission)` may be replaced by `mopen(filename,permission,0)`

Caution: `mtlb_fopen` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_format — Matlab format emulation function

# Description

Matlab and Scilab `format` behave differently in some particular cases:

- Some Matlab formats do not exist in Scilab
- Calling sequence for `format` is different in Scilab and Matlab

The function `mtlb_format(type,prec)` is used by `mfile2sci` to replace `format([type prec])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_format` calls:

- If `type=""` and `prec=""` `mtlb_format("","")` may be replaced by `format("v",6)`
- If `type="+"` and `prec=""` `mtlb_format("+","")` may be replaced by `format(6)`
- If `type="long"` and `prec=""` `mtlb_format("long","")` may be replaced by `format(16)`
- If `type="long"` and `prec="e"` `mtlb_format("long","e")` may be replaced by `format("e"16)`
- If `type="long"` and `prec="g"` `mtlb_format("long","g")` may be replaced by `format("e"16)`
- If `type="short"` and `prec=""` `mtlb_format("short","")` may be replaced by `format(6)`
- If `type="short"` and `prec="e"` `mtlb_format("short","e")` may be replaced by `format("e"6)`
- If `type="short"` and `prec="g"` `mtlb_format("short","g")` may be replaced by `format("e"6)`

Caution: `mtlb_format` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_fprintf — Matlab fprintf emulation function

# Description

There is no Scilab exact equivalent for Matlab `fprintf`. Scilab `mfprintf` and Matlab `fprintf` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fprintf(varargin)` is used by `mfile2sci` to replace `fprintf`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fprintf` by calls to mfprintf.

Caution: `mtlb_fprintf` has not to be used for hand coded functions.

# See Also

mfprintf

# Authors

V.C.

# Name

mtlb_fread — Matlab fread emulation function

# Description

There is no Scilab equivalent for Matlab `fread`. Scilab `mget` and Matlab `fread` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fread(varargin)` is used by `mfile2sci` to replace `fread`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fread` by calls to mget.

Caution: `mtlb_fread` has not to be used for hand coded functions.

# See Also

mget

# Authors

V.C.

# Name

mtlb_fscanf — Matlab fscanf emulation function

# Description

There is no Scilab exact equivalent for Matlab `fscanf`. Scilab `mfscanf` and Matlab `fscanf` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fscanf(varargin)` is used by `mfile2sci` to replace `fscanf`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fscanf` by calls to mfscanf.

Caution: `mtlb_fscanf` has not to be used for hand coded functions.

# See Also

mfscanf

# Authors

V.C.

# Name

mtlb_full — Matlab full emulation function

# Description

Matlab and Scilab `full` behave differently in some particular cases:

- With character strings input: Matlab `full` can be used with character string input while Scilab function cannot.
- With boolean input: Matlab `full` can be used with boolean input while Scilab function cannot.

The function `mtlb_full(A)` is used by `mfile2sci` to replace `full(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_full` calls:

- If `A` is a double matrix `mtlb_full(A)` may be replaced by `full(A)`
- If `A` is a boolean matrix `mtlb_full(A)` may be replaced by `full(bool2s(A))`

Caution: `mtlb_full` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_fwrite — Matlab fwrite emulation function

# Description

There is no Scilab equivalent for Matlab `fwrite`. Scilab `mput` and Matlab `fwrite` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fwrite(varargin)` is used by `mfile2sci` to replace `fwrite`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fwrite` by calls to mput.

Caution: `mtlb_fwrite` has not to be used for hand coded functions.

# See Also

mput

# Authors

V.C.

# Name

mtlb_grid — Matlab grid emulation function

# Description

There is no Scilab equivalent function for Matlab `grid` but there are equivalent instructions.

The function `mtlb_grid(flag_or_handle[,flag])` is used by `mfile2sci` to replace `grid(flag_or_handle[,flag])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_grid` calls:

- With one input, if `flag` is equal to "on" `mtlb_grid(flag)` may be replaced by `set(gca(),"grid",[1 1])`
- With one input, if `flag` is equal to "off" `mtlb_grid(flag)` may be replaced by `set(gca(),"grid",[-1 -1])`
- With two inputs, if `flag` is equal to "on" `mtlb_grid(axes_handle,flag)` may be replaced by `axes_handle.grid=[1 1]`
- With two inputs, if `arg2` is equal to "off" `mtlb_grid(axes_handle,flag)` may be replaced by `axes_handle.grid=[-1 -1]`

Caution: `mtlb_grid` has not to be used for hand coded functions.

# See Also

axes_properties

# Authors

V.C.

# Name

mtlb_hold — Matlab hold emulation function

# Description

There is no Scilab equivalent function for Matlab `hold` but there are equivalent instructions.

The function `mtlb_hold(flag)` is used by `mfile2sci` to replace `hold flag` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_hold` calls:

- If `flag` is equal to "on" `mtlb_hold(flag)` may be replaced by `set(gca(),"auto_clear","off")`
- If `flag` is equal to "off" `mtlb_hold(flag)` may be replaced by `set(gca(),"auto_clear","on")`

Caution: `mtlb_hold` has not to be used for hand coded functions.

# See Also

axes_properties

# Authors

V.C.

# Name

mtlb_i — Matlab insertion emulation function

# Description

Matlab and Scilab insertion behave differently in some particular cases:

- When inserting a matrix in a variable: Matlab automatically adjusts output variable to fit with matrix to insert but not Scilab. For example, with A=1, A([1,2,3,4])=[1,2;3,4]) returns an error in Scilab while in Matlab we get A=[1,2,3,4]. If values miss comparing to indexes, Matlab fills output value with 0.

- When inserting data into a character string matrix: due to the fact that character string matrices in Matlab can be adressed as any other matrix (each character can be adressed), insertion in such a type of matrix does not differ from other. But in Scilab it can't be done so...`mtlb_is` is an alternative.

The function `A=mtlb_i(A,k,B)` is used by `mfile2sci` to replace `A(k)=B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_i` calls:

- If `A` is not a vector `A=mtlb_i(A,k,B)` may be replaced by `A(k)=B`

- If `A` and `B` are both row or column vectors `A=mtlb_i(A,k,B)` may be replaced by `A(k)=B`

Caution: `mtlb_i` has not to be used for hand coded functions.

# See Also

Matlab-Scilab_character_strings , mtlb_is

# Authors

V.C.

# Name

mtlb_ifft — Matlab ifft emulation function

# Description

Matlab `ifft` and Scilab `fft` behave differently in some particular cases:

- With one input parameter: If input is a scalar or vector Scilab equivalent for Matlab `ifft(A)` is `fft(A,1)` else if input is a matrix Scilab equivalent for Matlab `fft` is `fft(A,1,2,1)`

The function `mtlb_ifft(X[,n,[job]])` is used by `mfile2sci` to replace `ifft(X[,n,[job]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_ifft` calls:

- If `X` is a scalar or a vector `mtlb_ifft(X,1)` may be replaced by `fft(X,1)`
- If `X` is a matrix `mtlb_ifft(X,1)` may be replaced by `fft(X,1,2,1)`

Caution: `mtlb_ifft` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_imp — Matlab colon emulation function

# Description

Matlab and Scilab colon behave differently in some particular cases:

- With empty matrices: The `:` operator must be used with scalars in Scilab and gives an error message when used with empty matrices while Matlab returns [] in these cases.

The function `mtlb_imp(A,B[,C])` is used by `mfile2sci` to replace `A:B[:C]` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_imp` calls:

- If `A`, `B` and `C` are not empty matrices `mtlb_imp(A,B[,C])` may be replaced by `A:B[:C]`

Caution: `mtlb_imp` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_int16 — Matlab int16 emulation function

# Description

Matlab and Scilab `int16` behave differently in some particular cases:

- With complex input: Matlab `int16` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `int16` returns 32767 and Scilab returns -32768.
- With `%nan`: Matlab `int16` returns 0 and Scilab returns -32768.
- With `-%nan`: Matlab `int16` returns 0 and Scilab returns -32768.

The function `mtlb_int16(A)` is used by `mfile2sci` to replace `int16(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_int16` calls:

- If `A` does not contain `%inf`, `%nan` or `-%nan` values `mtlb_int16(A)` may be replaced by `int16(A)`

Caution: `mtlb_int16` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_int32 — Matlab int32 emulation function

# Description

Matlab and Scilab `int32` behave differently in some particular cases:

- With complex input: Matlab `int32` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `int32` returns 2147483647 and Scilab returns -2147483648.
- With `%nan`: Matlab `int32` returns 0 and Scilab returns -2147483648.
- With `-%nan`: Matlab `int32` returns 0 and Scilab returns -2147483648.

The function `mtlb_int32(A)` is used by `mfile2sci` to replace `int32(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_int32` calls:

- If `A` does not contain `%inf`, `%nan` or `-%nan` values `mtlb_int32(A)` may be replaced by `int32(A)`

Caution: `mtlb_int32` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_int8 — Matlab int8 emulation function

# Description

Matlab and Scilab `int8` behave differently in some particular cases:

- With complex input: Matlab `int8` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `int8` returns 127 and Scilab returns 0.
- With `-%inf`: Matlab `int8` returns -128 and Scilab returns 0.

The function `mtlb_int8(A)` is used by `mfile2sci` to replace `int8(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_int8` calls:

- If `A` does not contain `%inf` or `-%inf` values `mtlb_int8(A)` may be replaced by `int8(A)`

Caution: `mtlb_int8` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_is — Matlab string insertion emulation function

# Description

Matlab and Scilab insertion behave differently for strings due to the fact that they do not consider character strings in the same way.

The function `str = mtlb_is(sto,sfrom,i,j)` is used by `mfile2sci` to replace insertion operations for character strings. This function will determine the correct semantic at run time. There is no replacement possibility for it.

Caution: `mtlb_is` has not to be used for hand coded functions.

# See Also

Matlab-Scilab_character_strings , mtlb_i

# Authors

V.C.

# Name

mtlb_isa — Matlab isa emulation function

# Description

There is no Scilab equivalent function for Matlab `isa` but some equivalent expressions can be used when the object "class" exists in Scilab.

The function `mtlb_isa(OBJ,class)` is used by `mfile2sci` to replace `isa(OBJ,class)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_isa` calls:

- If `class` is equal to "logical", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==4`
- If `class` is equal to "char", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==10`
- If `class` is equal to "numeric", `mtlb_isa(OBJ,class)` may be replaced by `or(type(OBJ)==[1,5,8])`
- If `class` is equal to "intX" (X being equal to 8, 16, or 32), `mtlb_isa(OBJ,class)` may be replaced by `typeof(OBJ)=="intX"`
- If `class` is equal to "uintX" (X being equal to 8, 16, or 32), `mtlb_isa(OBJ,class)` may be replaced by `typeof(OBJ)=="uintX"`
- If `class` is equal to "single", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==1`
- If `class` is equal to "double", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==1`
- If `class` is equal to "cell", `mtlb_isa(OBJ,class)` may be replaced by `typeof(OBJ)=="ce"`
- If `class` is equal to "struct", `mtlb_isa(OBJ,class)` may be replaced by `typeof(OBJ)=="st"`
- If `class` is equal to "function_handle", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==13`
- If `class` is equal to "sparse", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==5`
- If `class` is equal to "lti", `mtlb_isa(OBJ,class)` may be replaced by `typeof(OBJ)=="state-space"`

Caution: `mtlb_isa` has not to be used for hand coded functions.

# See Also

type , typeof

# Authors

V.C.

# Name

mtlb_isfield — Matlab isfield emulation function

# Description

There is no Scilab equivalent function for Matlab `isfield(st,f)` and equivalent expressions behave differently in some particular cases:

- If `st` is not a structure: Scilab equivalent returns an error message but Matlab returns `0`.

The function `mtlb_isfield(st,f)` is used by `mfile2sci` to replace `isfield(st,f)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_isfield` calls:

- If `st` is a structure `tf = mtlb_isfield(st,f)` may be replaced by `allf=getfield(1,st);tf=or(allf(3:$)==f);`
- If `st` is not a structure `tf = mtlb_isfield(st,f)` may be replaced by `tf=%F;`

Caution: `mtlb_isfield` has not to be used for hand coded functions.

# See Also

getfield

# Authors

V.C.

# Name

mtlb_isletter — Matlab isletter emulation function

# Description

There is no Scilab equivalent function for Matlab `isletter` and equivalent instructions are quite ugly, so `mfile2sci` uses `mtlb_isletter(A)` to replace `isletter(A)`. If you want to have a more efficient code it is possible to replace `mtlb_isletter` calls:

- If `A` is not a character string `tf = mtlb_isletter(A)` may be replaced by `tf = zeros(A)`
- If `A` is a character string `tf = mtlb_isletter(A)` may be replaced by `tf = (asciimat(A)>=65&asciimat(A)<=90)|(asciimat(A)>=97&asciimat(A)<=122)`

Caution: `mtlb_isletter` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_isspace — Matlab isspace emulation function

# Description

There is no Scilab function equivalent for Matlab `isspace` but its behavior can be reproduced.

The function `mtlb_isspace(A)` is used by `mfile2sci` to replace `isspace(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_isspace` calls:

- If `A` is a character string matrix `mtlb_isspace(A)` may be replaced by `asciimat(A)==32`
- If `A` is not a character string matrix `mtlb_isspace(A)` may be replaced by `zeros(A)`

Caution: `mtlb_isspace` has not to be used for hand coded functions.

# See Also

asciimat

# Authors

V.C.

# Name

mtlb_l — Matlab left division emulation function

# Description

Matlab and Scilab left division behave differently in some particular cases:

- With character string operands: The `\` operator can not be used into Scilab with character strings, while in Matlab it can. And in this case, result is transposed in a very strange way...

The function `mtlb_l(A,B)` is used by `mfile2sci` to replace `A\B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_l` calls:

- If both `A` and `B` are not character strings `mtlb_l(A,B)` may be replaced by `A\B`.

Caution: `mtlb_l` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_legendre — Matlab legendre emulation function

```
P = mtlb_legendre(n,X)
P = mtlb_legendre(n,X[,normflag])
```

# Parameters

X
: a scalar, vector, matrix or hypermatrix with elements in [-1,1]

n
: a positive scalar integer

normflag
: a string ('sch' or 'norm')

# Description

Matlab and Scilab `legendre` behave differently in some particular cases:

- Scilab `legendre(m,n,X)` evaluates the `legendre` function of degree n and order n for the `X` elements. Matlab `legendre(n,X)` evaluates the Legendre functions of degree `n` and order `m`=0,1,...,n. (emulated by mtlb_legendre) for the `X` elements.
- The option normflag= `'sch'` doesn't exist for Scilab legendre (emulated)
- If `X` is a hypermatrix then Scilab `legendre(n,X)` doesn't work (emulated)

The function `mtlb_legendre(n,X[,normflag])` is used by `mfile2sci` to replace `legendre(n,X[,normflag])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_legendre` call:

- If `X` is a scalar, a vector or a matrix `mtlb_legendre(n,X[,'norm'])` may be replaced by `legendre(n,0:n,X[,'norm'])`

Caution: `mtlb_legendre` has not to be used for hand coded functions.

# Authors

F.B.

# Name

mtlb_linspace — Matlab linspace emulation function

# Description

Matlab and Scilab `linspace` behave differently in some particular cases:

- With character string inputs: Matlab `linspace(A,B[,n])` returns a character string vector if `A` and/or `B` are character strings, but Scilab function does not work with such inputs.

The function `mtlb_linspace(A,B[,n])` is used by `mfile2sci` to replace `linspace(A,B[,n])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_linspace` calls:

- If `A` and `B` are not character strings `mtlb_linspace(A,B[,n])` may be replaced by `linspace(A,B[,n])`
- If `A` or `B` are character strings `mtlb_linspace(A,B[,n])` may be replaced by `ascii(linspace(ascii(A),ascii(B)[,n]))`

Caution: `mtlb_linspace` has not to be used for hand coded functions.

# See Also

ascii

# Authors

V.C.

# Name

mtlb_logic — Matlab logical operators emulation function

# Description

Matlab and Scilab logical operator behave differently in some particular cases:

- With complex operands: The <, <=, > and >= operators can not be used into Scilab with complex operands, while in Matlab they can. And in this case, only real part of complex operands is compared.
- With empty matrices: If both operands of <, <=, > and >= operators are empty matrices, Scilab returns an error message, while Matlab returns an empty matrix. For operators == and ~=, if at least one operand is an empty matrix, Matlab returns [] while Scilab returns a boolean value %T or %F.

The function `mtlb_logic(A,symbol,B)` (with "symbol" a character string containing the operator symbol) is used by `mfile2sci` to replace `A symbol B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_logic` calls:

- If both `A` and `B` are not complex values nor empty matrices `mtlb_logic(A,symbol,B)` may be replaced by `A symbol B`.

Caution: `mtlb_logic` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_logical — Matlab logical emulation function

# Description

There is no Scilab equivalent function for Matlab `logical` but its behavior can be easyly reproduced.

The function `mtlb_logical(A)` is used by `mfile2sci` to replace `logical(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_logical` calls:

- If `A` is a boolean matrix `mtlb_logical(A)` may be replaced by `A`
- If `A` is not an empty matrix `mtlb_logical(A)` may be replaced by `A<>[]`
- If `A` is an empty matrix `mtlb_logical(A)` may be replaced by `[]`

Caution: `mtlb_logical` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_lower — Matlab lower emulation function

# Description

Matlab `lower(A)` and Scilab `convstr(A,"l")` behave differently in some particular cases:

- If `A` is not a chacarter string matrix: Matlab `lower` can be used with a not-character-string `A` but not Scilab `convstr`.

The function `mtlb_lower(A)` is used by `mfile2sci` to replace `lower(A)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_lower` calls:

- If `A` is a character string matrix `mtlb_lower(A)` may be replaced by `convstr(A,"l")`
- If `A` is not a character string matrix `mtlb_lower(A)` may be replaced by `A`

Caution: `mtlb_lower` has not to be used for hand coded functions.

# See Also

convstr

# Authors

V.C.

# Name

mtlb_max — Matlab max emulation function

# Description

Matlab and Scilab `max` behave differently in some particular cases:

- With complex values: Matlab `max` can be used with complex values but not Scilab function.
- When called with one input: Matlab `max` threats values along the first non-singleton dimension but Scilab threats all input values.
- When called with two inputs: if one is an empty matrix, Scilab returns an error message but Matlab returns [].
- When called with three inputs: if `dim` parameter is greater than number of dimensions of first input, Scilab returns an error message and Matlab returns the first input.

The function `[r[,k]] = mtlb_max(A[,B[,dim]])` is used by `mfile2sci` to replace `[r[,k]] = max(A[,B[,dim]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_max` calls:

- When called with one input, if `A` is a vector or a scalar `[r[,k]] = mtlb_max(A)` may be replaced by `max(A)`
- When called with one input, if `A` is a matrix `[r[,k]] = mtlb_max(A)` may be replaced by `max(A,"r")`
- When called with two inputs, if `A` and `B` are real matrices and not empty matrices `[r[,k]] = mtlb_max(A,B)` may be replaced by `max(A,B)`
- When called with three inputs, if `dim` is lesser than the number of dimensions of `A` `[r[,k]] = mtlb_max(A,[],dim)` may be replaced by `max(A,dim)`

Caution: `mtlb_max` has not to be used for hand coded functions.

# See Also

firstnonsingleton

# Authors

V.C.

# Name

mtlb_min — Matlab min emulation function

# Description

Matlab and Scilab `min` behave differently in some particular cases:

- With complex values: Matlab `min` can be used with complex values but not Scilab function.
- When called with one input: Matlab `min` threats values along the first non-singleton dimension but Scilab threats all input values.
- When called with two inputs: if one is an empty matrix, Scilab returns an error message but Matlab returns [].
- When called with three inputs: if `dim` parameter is greater than number of dimensions of first input, Scilab returns an error message and Matlab returns the first input.

The function `[r[,k]] = mtlb_min(A[,B[,dim]])` is used by `mfile2sci` to replace `[r[,k]] = min(A[,B[,dim]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_min` calls:

- When called with one input, if `A` is a vector or a scalar `[r[,k]] = mtlb_min(A)` may be replaced by `min(A)`
- When called with one input, if `A` is a matrix `[r[,k]] = mtlb_min(A)` may be replaced by `min(A,"r")`
- When called with two inputs, if `A` and `B` are real matrices and not empty matrices `[r[,k]] = mtlb_min(A,B)` may be replaced by `min(A,B)`
- When called with three inputs, if `dim` is lesser than the number of dimensions of `A` `[r[,k]] = mtlb_min(A,[],dim)` may be replaced by `min(A,dim)`

Caution: `mtlb_min` has not to be used for hand coded functions.

# See Also

firstnonsingleton

# Authors

V.C.

# Name

mtlb_more — Matlab more emulation function

# Description

Matlab `more` and Scilab `lines` behave differently in some particular cases:

- With character strings as input: Matlab `more` can take "on" and "off" as input but not Scilab `lines` but there are equivalents (0 and 60).

The function `mtlb_more(in)` is used by `mfile2sci` to replace `more(in)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_more` calls:

- If `in` is equal to "on" `mtlb_more(in)` may be replaced by `lines(60)`
- If `in` is equal to "off" `mtlb_more(in)` may be replaced by `lines(0)`
- If `in` is a double value `mtlb_more(in)` may be replaced by `lines(in)`

Caution: `mtlb_more` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_num2str — Matlab num2str emulation function

# Description

Matlab `num2str` and Scilab equivalents (`string`, `msprintf`) behave differently in some particular cases:

- With two input parameters, the second giving precision: There is no Scilab equivalent function, but `msprintf` can be used to emulate.
- With two input parameters, the second giving format: Scilab equivalent for Matlab `num2string(a,format)` is `msprintf(format,a)`.

The function `mtlb_num2str(x,f)` is used by `mfile2sci` to replace `num2str(x,f)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_num2str` calls:

- If `f` is a character string `mtlb_num2str(x,f)` may be replaced by `msprintf(f,x)`

Caution: `mtlb_num2str` has not to be used for hand coded functions.

# See Also

string , msprintf

# Authors

V.C.

# Name

mtlb_ones — Matlab ones emulation function

# Description

Matlab and Scilab `ones` behave differently in some particular cases:

- With a scalar input: Matlab `ones` returns a n x n matrix while Scilab returns a 1.

The function `mtlb_ones(A)` is used by `mfile2sci` to replace `ones(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_ones` calls:

- If `A` is a scalar `mtlb_ones(A)` may be replaced by `ones(A,A)`
- If `A` is not a scalar `mtlb_ones(A)` may be replaced by `ones(A)`

Caution: `mtlb_ones` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_plot — Matlab plot emulation function

# Description

Matlab `plot` and Scilab `plot2d` behave differently.

The function `mtlb_plot(varargin)` is used by `mfile2sci` to replace `plot(varargin)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_plot` calls when there is no output value, however in this case, you have to set colors manualy:

- With one input, if `Y` is real, `mtlb_plot(Y)` may be replaced by `plot2d(Y)`
- With one input, if `Y` is complex, `mtlb_plot(Y)` may be replaced by `plot2d(real(Y),imag(Y))`
- With two inputs `X` and `Y`, if `Y` is not a character string, `mtlb_plot(X,Y)` may be replaced by `plot2d(X,Y)`

Caution: `mtlb_plot` has not to be used for hand coded functions.

# See Also

plot2d

# Authors

V.C.

# Name

mtlb_prod — Matlab prod emulation function

# Description

Matlab and Scilab `prod` behave differently in some particular cases:

- When called with one input: Matlab `prod` threts the values along the first non-singleton dimension of input while Scilab `prod` threats all values of input.
- When called with two inputs: Matlab `prod` can be used with second input giving a non-existing dimension of first input while Scilab `prod` returns an error message.

The function `mtlb_prod(A[,dim])` is used by `mfile2sci` to replace `prod(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_prod` calls:

- When called with one input, if `A` is an empty matrix, a scalar or a vector, `mtlb_prod(A)` may be replaced by `prod(A)`
- When called with one input, if `A` is a not-empty matrix, `mtlb_prod(A)` may be replaced by `prod(A,1)`
- When called with one input, if `A` is a multidimensional array, `mtlb_prod(A)` may be replaced by `prod(A,firstnonsingleton(A))`
- When called with two inputs, if `dim` is lesser than the number of dimensions of `A` `mtlb_prod(A,dim)` may be replaced by `prod(A,dim)`

Caution: `mtlb_prod` has not to be used for hand coded functions.

# See Also

firstnonsingleton

# Authors

V.C.

# Name

mtlb_rand — Matlab rand emulation function

# Description

Matlab and Scilab `rand` behave differently in some particular cases:

- With a scalar input: Matlab `rand` returns a n x n matrix while Scilab returns a scalar.

The function `mtlb_rand(A)` is used by `mfile2sci` to replace `rand(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_rand` calls:

- If `A` is a scalar `mtlb_rand(A)` may be replaced by `rand(A,A)`
- If `A` is not a scalar `mtlb_rand(A)` may be replaced by `rand(A)`

Caution: `mtlb_rand` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_randn — Matlab randn emulation function

# Description

Matlab `rand` and Scilab `rand(...,"normal")` behave differently in some particular cases:

- With a scalar input: Matlab `randn` returns a n x n matrix while Scilab `rand(...,"normal")` returns a scalar.

The function `mtlb_randn(A)` is used by `mfile2sci` to replace `randn(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_randn` calls:

- If `A` is a scalar `mtlb_randn(A)` may be replaced by `rand(A,A,"normal")`
- If `A` is not a scalar `mtlb_randn(A)` may be replaced by `rand(A,"normal")`

Caution: `mtlb_randn` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_rcond — Matlab rcond emulation function

# Description

Matlab and Scilab `rcond` behave differently in some particular cases:

- With empty matrix: Matlab `rcond` returns `Inf` and Scilab `rcond` returns `[]`

The function `mtlb_rcond(A)` is used by `mfile2sci` to replace `rcond(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_rcond` calls:

- If `A` is not an empty matrix, `mtlb_rcond(A)` may be replaced by `rcond(A)`

Caution: `mtlb_rcond` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_realmax — Matlab realmax emulation function

# Description

Scilab equivalent for Matlab `realmax` is `number_properties` but not all cases are implemented:

- Scilab equivalent for Matlab `realmax` or `realmax('double')` is `number_properties("huge")`.
- There is no Scilab equivalent for Matlab `realmax('single')`

The function `mtlb_realmax(prec)` is used by `mfile2sci` to replace `realmax(prec)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_realmax` calls:

- If `prec` is equal to `"double"` `mtlb_realmax(prec)` may be replaced by `number_properties("huge")`

Caution: `mtlb_realmax` has not to be used for hand coded functions.

# See Also

number_properties

# Authors

V.C.

# Name

mtlb_realmin — Matlab realmin emulation function

# Description

Scilab equivalent for Matlab `realmin` is `number_properties` but not all cases are implemented:

- Scilab equivalent for Matlab `realmin` or `realmin('double')` is `number_properties("tiny")`.
- There is no Scilab equivalent for Matlab `realmin('single')`

The function `mtlb_realmin(prec)` is used by `mfile2sci` to replace `realmin(prec)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_realmin` calls:

- If `prec` is equal to `"double"` `mtlb_realmin(prec)` may be replaced by `number_properties("tiny")`

Caution: `mtlb_realmin` has not to be used for hand coded functions.

# See Also

number_properties

# Authors

V.C.

# Name

mtlb_repmat — Matlab repmat emulation function

# Description

There is no Scilab equivalent function for Matlab `repmat` but there are equivalent instructions.

The function `mtlb_repmat(M,m[,n])` is used by `mfile2sci` to replace `repmat(M,m[,n])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_repmat` calls:

- If `A` is of Double type and `m` is a scalar, `mtlb_repmat(M,m)` may be replaced by `ones(m,m).*.M` and `mtlb_repmat(M,m,n)` may be replaced by `ones(m,n).*.M`
- If `A` is of Boolean type and `m` is a scalar, `mtlb_repmat(M,m)` may be replaced by `ones(m,m).*.bool2s(M)` and `mtlb_repmat(M,m,n)` may be replaced by `ones(m,n).*.bool2s(M)`
- If `A` is of String type and `m` is a scalar, `mtlb_repmat(M,m)` may be replaced by `asciimat(ones(m,m).*.asciimat(M))` and `mtlb_repmat(M,m,n)` may be replaced by `asciimat(ones(m,n).*.asciimat(M))`
- If `A` is of Double type and `m` is a vector, `mtlb_repmat(M,m)` may be replaced by `ones(m(1),m(2),...).*.M`
- If `A` is of Boolean type and `m` is a vector, `mtlb_repmat(M,m)` may be replaced by `ones(m(1),m(2),...).*.bool2s(M)`
- If `A` is of String type and `m` is a vector, `mtlb_repmat(M,m)` may be replaced by `asciimat(ones(m(1),m(2),...).*.asciimat(M))`

Caution: `mtlb_repmat` has not to be used for hand coded functions.

# See Also

ones , kron

# Authors

V.C.

# Name

mtlb_s — Matlab substraction emulation function

# Description

Matlab and Scilab substraction behave differently in some particular cases:

- With character string operands: The `-` operator can not be used into Scilab with character strings, while Matlab realizes the substraction of the operands ASCII codes.
- With empty matrix: In Scilab, if one of the operands is an empty matrix the result of the substraction is the other operand. In Matlab if one of the operand is an empty matrix the result of the substraction should be an error or an empty matrix.

The function `mtlb_s(A,B)` is used by `mfile2sci` to replace `A-B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_s` calls:

- If `A` and `B` are character strings, `mtlb_s(A,B)` may be replaced by `asciimat(A)-asciimat(B)`
- If both `A` and `B` are not empty matrices `mtlb_s(A,B)` may be replaced by `A-B`, else `mtlb_s(A,B)` may be replaced by `[]`.
- If mtlb_mode()==%T, then `mtlb_s(A,B)` may be replaced by `A-B` in any case where `A` and `B` are not character string matrices.

Caution: `mtlb_s` has not to be used for hand coded functions.

# See Also

mtlb_mode

# Authors

V.C.

# Name

mtlb_setstr — Matlab setstr emulation function

# Description

Matlab `setstr` and Scilab `ascii` behave differently in some particular cases:

- With character string input: Matlab `setstr` returns a character string while Scilab `ascii` returns ASCII codes.
- With double matrix input: Matlab `setstr` returns a character matrix having the same size as input while Scilab `ascii` returns a single character string

The function `mtlb_setstr(A)` is used by `mfile2sci` to replace `setstr(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_setstr` calls:

- If `A` is a character string or a character string matrix `mtlb_setstr(A)` may be replaced by `A`
- If `A` is a double row vector `mtlb_setstr(A)` may be replaced by `ascii(A)`

Caution: `mtlb_setstr` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_size — Matlab size emulation function

# Description

Matlab and Scilab `size` behave differently in some particular cases:

- With two inputs: Matlab `size` can be used with second parameter giving a not-existing dimension of first parameter (returns 1 in this case) but not Scilab one.
- With more than one output: if number of output is lesser than number of dimensions, last output is the product of all remaining dimensions in Matlab but not in Scilab. If number of output is greater than number of dimensions, outputs corresponding to a not-existing dimension are set to 1 in Matlab but Scilab gives an error in this case.

The function `[d1,[d2,...]] = mtlb_size(X[,dim])` is used by `mfile2sci` to replace `[d1,[d2,...]] = size(X[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_size` calls:

- With two inputs: if `dim` gives an existing dimension of `X` `mtlb_size(X,dim)` may be replaced by `size(X,dim)`
- With more than one outputs: if the number of outputs is equal to the number of dimensions of `X` `[d1,[d2,...]] = mtlb_size(X)` may be replaced by `[d1,[d2,...]] = size(X)`

Caution: `mtlb_size` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_sort — Matlab sort emulation function

```
P = mtlb_sort(X)
P = mtlb_sort(X,dim[,mode])
```

# Parameters

X
: a scalar, vector, matrix of reals, booleans or a string

dim
: a positive scalar integer

mode
: a string ("ascend" or "descend")

# Description

Matlab `sort` and Scilab `gsort` behave differently in some particular cases:

- For a vector `X` the Matlab `sort(X,'g','i')` function call is equivalent to the Scilab `gsort(X)` function call.
- The value `1` (resp. `2`) of the Matlab dim is equivalent to the Scilab `"r"` flag (resp. `"c"`).
- The Matlab "`ascend`" (resp. `"descend"`) mode is equivalent to the Scilab `"i"` (resp. `"d"`) flag.

The function `mtlb_sort(X[,dim[,mode]])` is used by `mfile2sci` to replace `sort(X[,dim[,mode]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_sort` call by gsort call.

Caution: `mtlb_sort` has not to be used for hand coded functions.

# Authors

F.B.

# Name

mtlb_strcmp — Matlab strcmp emulation function

# Description

There is no Scilab function equivalent for Matlab `strcmp`, there is equivalent instructions.

The function `mtlb_strcmp(A,B)` is used by `mfile2sci` to replace `strcmp(A,B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strcmp` calls:

- If `A` and `B` are character strings `mtlb_strcmp(A,B)` may be replaced by `A==B`
- If `A` and/or `B` is not a character string `mtlb_strcmp(A,B)` may be replaced by `0`

Caution: `mtlb_strcmp` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_strcmpi — Matlab strcmpi emulation function

# Description

There is no Scilab function equivalent for Matlab `strcmpi`, there is equivalent instructions.

The function `mtlb_strcmpi(A,B)` is used by `mfile2sci` to replace `strcmpi(A,B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strcmpi` calls:

- If `A` and `B` are character strings `mtlb_strcmpi(A,B)` may be replaced by `convstr(A)==convstr(B)`
- If `A` and/or `B` is not a character string `mtlb_strcmpi(A,B)` may be replaced by `0`

Caution: `mtlb_strcmpi` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_strfind — Matlab strfind emulation function

# Description

Matlab `strfind` and Scilab `strindex` behave differently in some particular cases:

- With inputs which not character strings: Matlab `strfind` can be used with not character strings inputs but not Scilab `strindex`.

The function `mtlb_strfind(A,B)` is used by `mfile2sci` to replace `strfind(A,B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strfind` calls:

- If `A` and `B` are character strings `mtlb_strfind(A,B)` may be replaced by `strindex(A,B)`

Caution: `mtlb_strfind` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_strrep — Matlab strrep emulation function

# Description

Matlab `strrep` and Scilab `strsubst` behave differently in some particular cases:

- With inputs which not character strings: Matlab `strrep` can be used with not character strings inputs but not Scilab `strsubst`.

The function `mtlb_strrep(A,B,C)` is used by `mfile2sci` to replace `strrep(A,B,C)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strrep` calls:

- If `A`, `B` and `C` are character strings `mtlb_strrep(A,B,C)` may be replaced by `strsubst(A,B,C)`

Caution: `mtlb_strrep` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_sum — Matlab sum emulation function

# Description

Matlab and Scilab `sum` behave differently in some particular cases:

- When called with one input: Matlab `sum` threts the values along the first non-singleton dimension of input while Scilab `sum` threats all values of input.
- When called with two inputs: Matlab `sum` can be used with second input giving a non-existing dimension of first input while Scilab `sum` returns an error message.

The function `mtlb_sum(A[,dim])` is used by `mfile2sci` to replace `sum(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_sum` calls:

- When called with one input, if `A` is an empty matrix, a scalar or a vector, `mtlb_sum(A)` may be replaced by `sum(A)`
- When called with one input, if `A` is a not-empty matrix, `mtlb_sum(A)` may be replaced by `sum(A,1)`
- When called with one input, if `A` is a multidimensional array, `mtlb_sum(A)` may be replaced by `sum(A,firstnonsingleton(A))`
- When called with two inputs, if `dim` is lesser than the number of dimensions of `A` `mtlb_sum(A,dim)` may be replaced by `sum(A,dim)`

Caution: `mtlb_sum` has not to be used for hand coded functions.

# See Also

firstnonsingleton

# Authors

V.C.

# Name

mtlb_t — Matlab transposition emulation function

# Description

Matlab and Scilab transposition behave differently in some particular cases:

- With character strings operands: The `'` operator is used to transpose whole character strings in Scilab while Matlab realizes the transposition of each character.

The function `mtlb_t(A)` is used by `mfile2sci` to replace `A'` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_t` calls:

- If `A` is not a character string matrix `mtlb_t(A)` may be replaced by `A'`

Caution: `mtlb_t` has not to be used for hand coded functions.

# See Also

Matlab-Scilab_character_strings

# Authors

V.C.

# Name

mtlb_toeplitz — Matlab toeplitz emulation function

# Description

Matlab and Scilab `toeplitz` behave differently in some particular cases:

- With one input parameter: if this parameter is complex or is a matrix, output value of Matlab and Scilab `toeplitz` can be different.
- With two input parameters: if they are vectors and their first elements are not equal, Scilab returns an error but Matlab gives priority to the column element. If they are matrices, output value of Matlab and Scilab `toeplitz` are different.

The function `mtlb_toeplitz(c[,r])` is used by `mfile2sci` to replace `toeplitz(c[,r])` when it was not possible to know what were the operands/inputs[CUSTOM] while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_toeplitz` calls:

- When called with one input, if `c` is a real scalar or vector `mtlb_toeplitz(c)` may be replaced by `toeplitz(c)`
- When called with two inputs, if `c` and `r` are scalars or vectors and their first elements are equal `mtlb_toeplitz(c,r)` may be replaced by `toeplitz(c,r)`

Caution: `mtlb_toeplitz` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_tril — Matlab tril emulation function

# Description

Matlab and Scilab `tril` behave differently in some particular cases:

- With complex input: Matlab `tril` can be used with complex data but not Scilab one.
- With character strings inputs: due to the fact the Matlab and Scilab do not consider character strings in the same way, Scilab and Matlab `tril` do not give the same results for this type of input.
- With boolean inputs: Matlab `tril` can be used with boobean data but not Scilab one.

The function `mtlb_tril(x,k)` is used by `mfile2sci` to replace `tril(x,k)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_tril` calls:

- If `X` contains real double values `mtlb_tril(x,k)` may be replaced by `tril(x,k)`
- If `X` contains boolean values `mtlb_tril(x,k)` may be replaced by `tril(bool2s(x),k)`

Caution: `mtlb_tril` has not to be used for hand coded functions.

# See Also

Matlab-Scilab_character_strings

# Authors

V.C.

# Name

mtlb_triu — Matlab triu emulation function

# Description

Matlab and Scilab `triu` behave differently in some particular cases:

- With complex input: Matlab `triu` can be used with complex data but not Scilab one.
- With character strings inputs: due to the fact the Matlab and Scilab do not consider character strings in the same way, Scilab and Matlab `triu` do not give the same results for this type of input.
- With boolean inputs: Matlab `triu` can be used with boobean data but not Scilab one.

The function `mtlb_triu(x,k)` is used by `mfile2sci` to replace `triu(x,k)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_triu` calls:

- If `X` contains real double values `mtlb_triu(x,k)` may be replaced by `triu(x,k)`
- If `X` contains boolean values `mtlb_triu(x,k)` may be replaced by `triu(bool2s(x),k)`

Caution: `mtlb_triu` has not to be used for hand coded functions.

# See Also

Matlab-Scilab_character_strings

# Authors

V.C.

# Name

mtlb_true — Matlab true emulation function

# Description

There is no Scilab equivalent for Matlab `true`. However, Scilab `ones` returns a result interpreted in an equivalent way for Scilab.

Matlab `true` and Scilab `ones` behave differently in some particular cases:

- With a scalar input: Matlab `true` returns a n x n matrix of ones while Scilab `ones` returns a 1.

The function `mtlb_true(A)` is used by `mfile2sci` to replace `true(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_true` calls:

- If `A` is a scalar `mtlb_true(A)` may be replaced by `ones(A,A)`
- If `A` is not a scalar `mtlb_true(A)` may be replaced by `ones(A)`

Caution: `mtlb_true` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_uint16 — Matlab uint16 emulation function

# Description

Matlab and Scilab `uint16` behave differently in some particular cases:

- With complex input: Matlab `uint16` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `uint16` returns 65535 and Scilab returns 0.

The function `mtlb_uint16(A)` is used by `mfile2sci` to replace `uint16(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_uint16` calls:

- If `A` does not contain `%inf` values `mtlb_uint16(A)` may be replaced by `uint16(A)`

Caution: `mtlb_uint16` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_uint32 — Matlab uint32 emulation function

# Description

Matlab and Scilab `uint32` behave differently in some particular cases:

- With complex input: Matlab `uint32` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `uint32` returns 4294967295 and Scilab returns 0.

The function `mtlb_uint32(A)` is used by `mfile2sci` to replace `uint32(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_uint32` calls:

- If `A` does not contain `%inf` values `mtlb_uint32(A)` may be replaced by `uint32(A)`

Caution: `mtlb_uint32` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_uint8 — Matlab uint8 emulation function

# Description

Matlab and Scilab `uint8` behave differently in some particular cases:

- With complex input: Matlab `uint8` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `uint8` returns 255 and Scilab returns 0.

The function `mtlb_uint8(A)` is used by `mfile2sci` to replace `uint8(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_uint8` calls:

- If `A` does not contain `%inf` values `mtlb_uint8(A)` may be replaced by `uint8(A)`

Caution: `mtlb_uint8` has not to be used for hand coded functions.

# Authors

V.C.

# Name

mtlb_upper — Matlab upper emulation function

# Description

Matlab `upper(A)` and Scilab `convstr(A,"u")` behave differently in some particular cases:

- If `A` is not a chacarter string matrix: Matlab `upper` can be used with a not-character-string `A` but not Scilab `convstr`.

The function `mtlb_upper(A)` is used by `mfile2sci` to replace `upper(A)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_upper` calls:

- If `A` is a character string matrix `mtlb_upper(A)` may be replaced by `convstr(A,"u")`
- If `A` is not a character string matrix `mtlb_upper(A)` may be replaced by `A`

Caution: `mtlb_upper` has not to be used for hand coded functions.

# See Also

convstr

# Authors

V.C.

# Name

mtlb_var — Matlab var emulation function

# Parameters

x
: a real or a complex vector or matrix.

s
: a real scalar or real vector.

- If x is a vector, s is the variance of x.
- If x is a matrix, s is a row vector containing the variance of each column of x.

w
: type of normalization to use. Valid values are, depending on the number of columns m of x :

- w = 0 : normalizes with m-1, provides the best unbiased estimator of the variance (this is the default).
- w = 1: normalizes with m, this provides the second moment around the mean.

dim
: the dimension along which the variance is computed (default is 1, i.e. column by column). If dim is 2, the variance is computed row by row.

# Description

This function computes the variance of the values of a vector or matrix x. It provides the same service as Octave and Matlab. It differs from Scilab's variance primitive:

- mtlb_var returns a real (i.e. with a zero imaginary part) variance, even if x is a complex vector or matrix. The Scilab variance primitive returns a complex value if the input vector x is complex and if no option additionnal is used.
- Whatever the type of the input data x (i.e. vector or matrix), mtlb_var computes the variance either on dimension 1 or on dimension 2 while, if no option is passed to the Scilab's variance primitive, the variance is computed on all dimension at once.

# Examples

The following 3 examples illustrates the use of the mtlb_var function. In the first case, a column vector is passed to the function, which returns the value 750. In the second case, a matrix is passed to the function, which returns the row vector [0.16 0.09]. In the third case, a complex column vector is passed to the function, which returns a value close to 2.

```
x = [10; 20; 30; 40; 50; 60; 70; 80; 90];
computed = mtlb_var(x);

x = [0.9    0.7
    0.1    0.1
    0.5    0.4];
computed = mtlb_var(x);

N=1000;
x = grand(N,1,'nor',0,1) + %i*grand(N,1,'nor',0,1);
```

```
computed = mtlb_var(x);
```

## See Also

variance

## Authors

Michael Baudin

# Name

mtlb_zeros — Matlab zeros emulation function

# Description

Matlab and Scilab `zeros` behave differently in some particular cases:

- With a scalar input: Matlab `zeros` returns a n x n matrix while Scilab returns a 0.

The function `mtlb_zeros(A)` is used by `mfile2sci` to replace `zeros(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_zeros` calls:

- If `A` is a scalar `mtlb_zeros(A)` may be replaced by `zeros(A,A)`
- If `A` is not a scalar `mtlb_zeros(A)` may be replaced by `zeros(A)`

Caution: `mtlb_zeros` has not to be used for hand coded functions.

# Authors

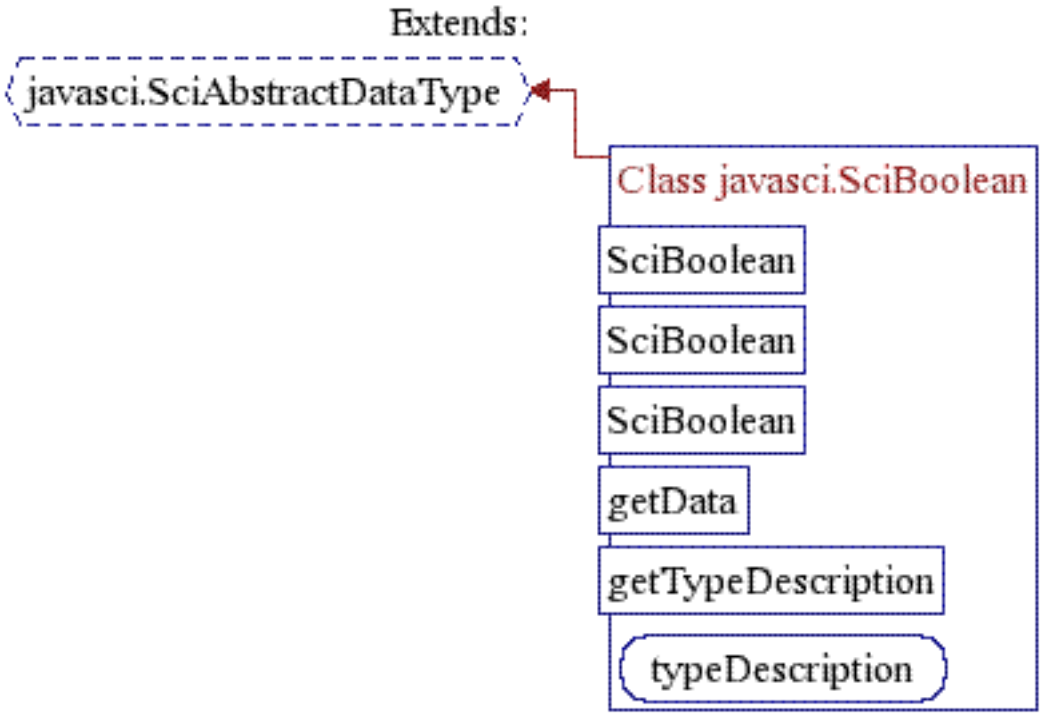
V.C.

# Parte XXXVII. Interface Java

# Name

javasci.SciBoolean — Class to use boolean object with scilab

# Description




Method Summary :

`public SciBoolean(String name,SciBoolean Obj)`

`public SciBoolean(String name)` Constructor (if name exists in Scilab and has the same type, variable is imported from Scilab)

`public SciBoolean(String name,boolean Value )`

`public String getName()`Get Name of scilab object

`public boolean getData()`Get Value of scilab object

`public void Get()`Get in java object , value of scilab object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Send()`Send to scilab object , value of java object

`public void disp()`disp object

# Examples

```
// See SCI/modules/javasci/examples directory
```

# See Also

javasci.Scilab , Compile and run with Javasci, SciBooleanArray , SciDoubleArray , SciString , SciStringArray

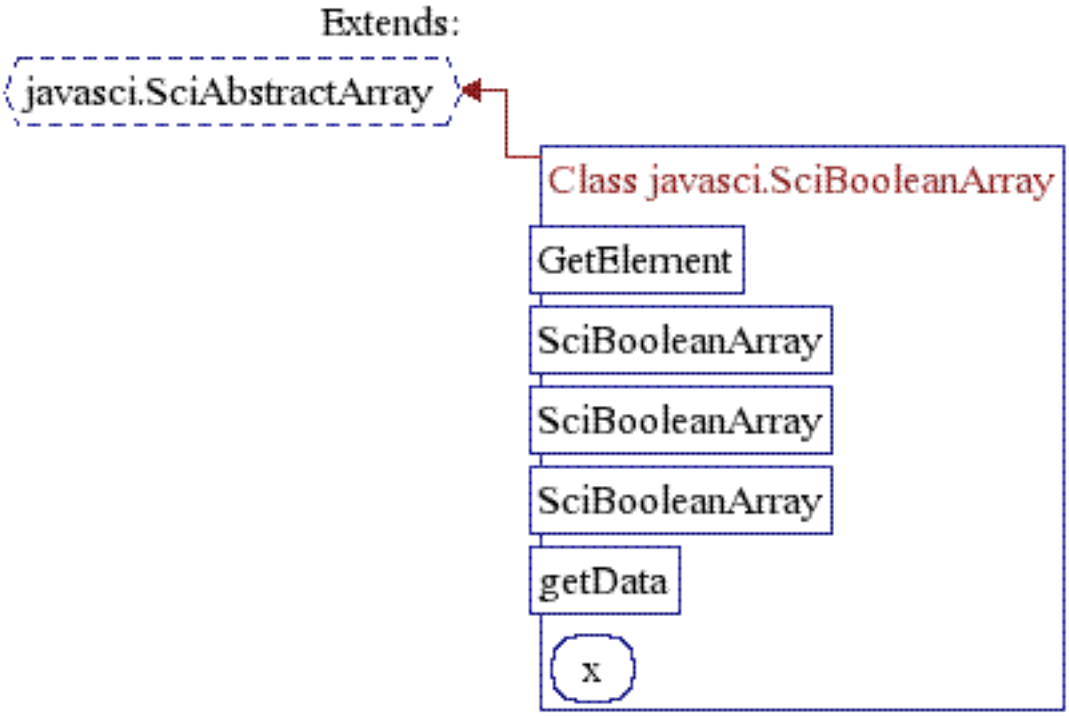
# Authors

A.C

# Name

javasci.SciBooleanArray — Class to use boolean matrix in Scilab.

# Description




Method Summary :

`public SciBooleanArray(String name,SciBooleanArray Obj)`

`public SciBooleanArray(String name,int r,int c)`

`public SciBooleanArray(String name,int r,int c,boolean [] x )`Constructor

`public int getNumbersOfRows()`Get number of rows

`public int getNumbersOfCols()` Get number of colons

`public String getName()`Get Name of scilab object

`public boolean[] getData()` Get Value of scilab object

`public void disp()` disp object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Get()`Get in java object , value of scilab object

`public void Send()`Send to scilab object , value of java object

`public boolean GetElement(int indr, int indc)`Get a specific element of scilab object

# Examples

```
// See SCI/modules/javasci/examples directory
```

## See Also

javasci.Scilab , Compile and run with Javasci, SciBoolean , SciDouble , SciString , SciStringArray

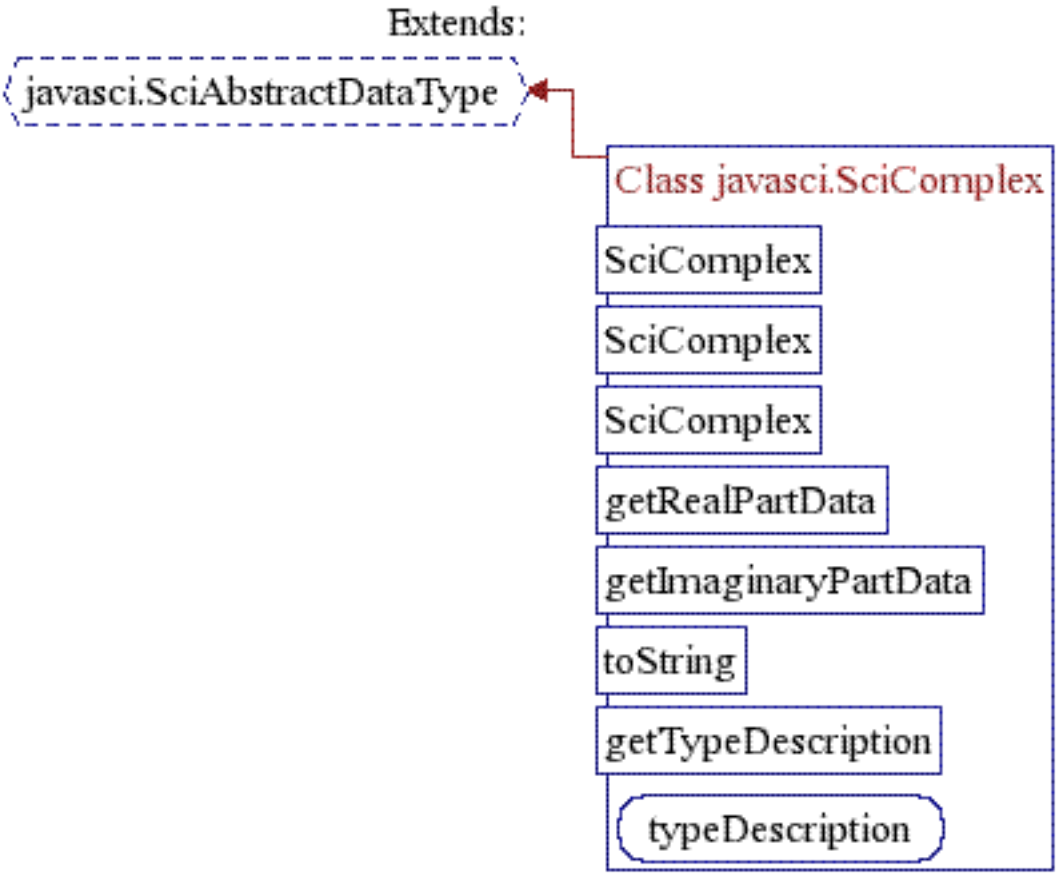
## Authors

A.C

# Name

javasci.SciComplex — Class to use complex object with scilab

# Description




Method Summary :

`public SciComplex(String name,SciComplex Obj)`

`public SciComplex(String name)` Constructor (if name exists in Scilab and has the same type, variable is imported from Scilab)

`public SciComplex(String name,double realpart,double imaginary-part )`Constructor

`public String getName()`Get Name of scilab object

`public double getRealPartData()`Get Real Part Value of scilab object

`public double getImaginaryPartData()`Get Imaginary Part Value of scilab object

`public void Get()`Get in java object , value of scilab object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Send()`Send to scilab object , value of java object

`public void disp()`disp object

`public void toString()`convert complex to a string

# Examples

```
// See SCI/modules/javasci/examples directory
```

# See Also

javasci.Scilab , Compile and run with Javasci, SciComplexArray , SciString , SciStringArray , SciDoubleArray , SciDouble

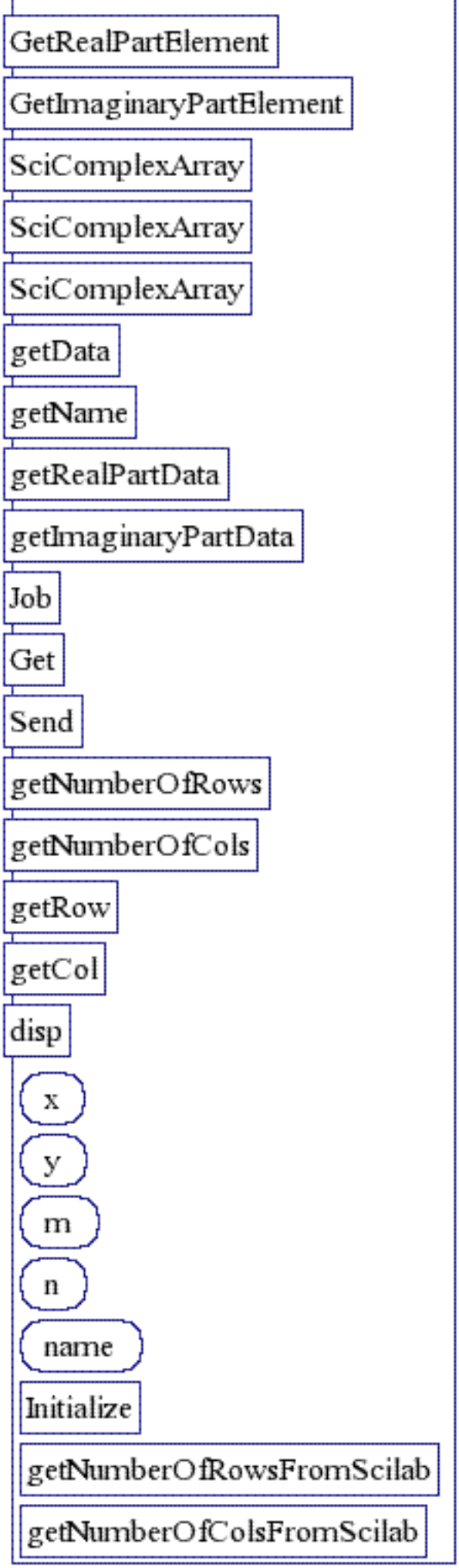
# Authors

A.C

# Name

javasci.SciComplexArray — Class to use complex matrix in Scilab.

## Descr


GetRealPartElement
GetImaginaryPartElement
SciComplexArray
SciComplexArray
SciComplexArray
getData
getName
getRealPartData
getImaginaryPartData
Job
Get
Send
getNumberOfRows
getNumberOfCols
getRow
getCol
disp
x
y
m
n
name
Initialize
getNumberOfRowsFromScilab
getNumberOfColsFromScilab

Method Summary :

`public SciComplexArray(String name,SciComplexArray Obj)`

`public SciComplexArray(String name,int r,int c)`

`public SciComplexArray(String name,int r,int c,double [] realpart,double [] imaginarypart)`Constructor

`public int getNumbersOfRows()`Get number of rows

`public int getNumbersOfCols()` Get number of colons

`public String getName()`Get Name of scilab object

`public double[] getRealPartData()`Get Real Part Value of scilab object

`public double[] getImaginaryPartData()`Get Imaginary Part Value of scilab object

`public void disp()` disp object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Get()`Get in java object , value of scilab object

`public void Send()`Send to scilab object , value of java object

`public double GetRealPartElement(int indr, int indc)`Get a specific element of scilab object

`public double GetImaginaryPartElement(int indr, int indc)`Get a specific element of scilab object

# Examples

```
// See SCI/modules/Javasci/examples directory
```

# See Also

javasci.Scilab , Compile and run with Javasci, SciComplex , SciDouble , SciDoubleArray , SciString , SciStringArray

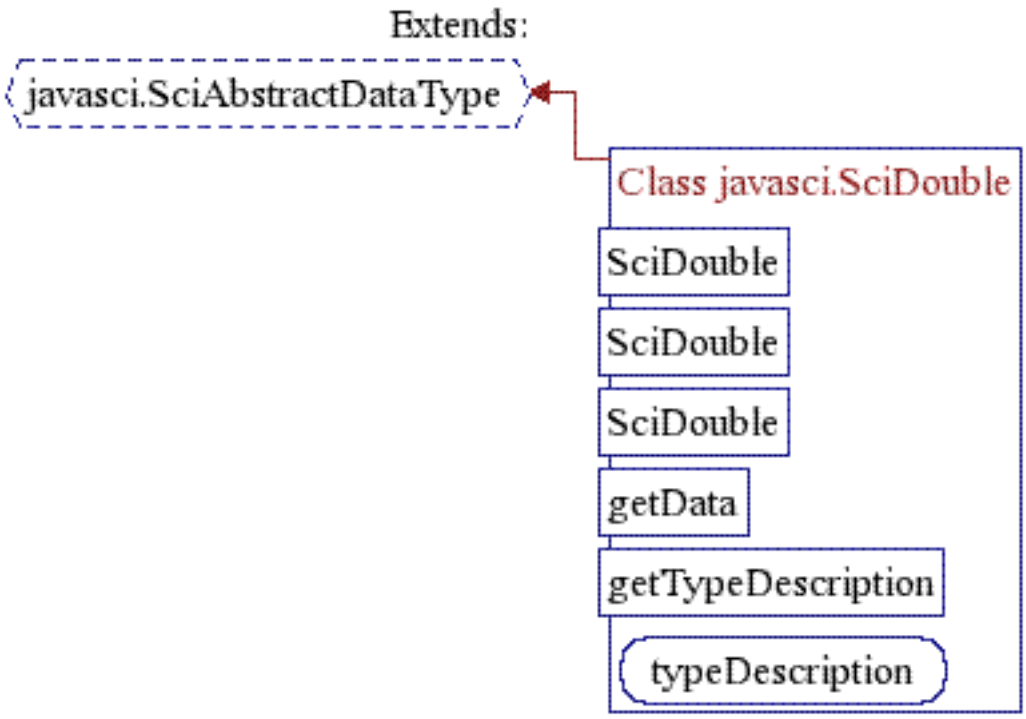
# Authors

A.C

# Name

javasci.SciDouble — Class to use double object with scilab

# Description




Method Summary :

`public SciDouble(String name,SciDouble Obj)`

`public SciDouble(String name)` Constructor (if name exists in Scilab and has the same type, variable is imported from Scilab)

`public SciDouble(String name,double Value )`

`public String getName()`Get Name of scilab object

`public double getData()`Get Value of scilab object

`public void Get()`Get in java object , value of scilab object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Send()`Send to scilab object , value of java object

`public void disp()`disp object

# Examples

```
// See SCI/modules/Javasci/examples directory
```

# See Also

javasci.Scilab , Compile and run with Javasci, SciDoubleArray , SciString , SciStringArray

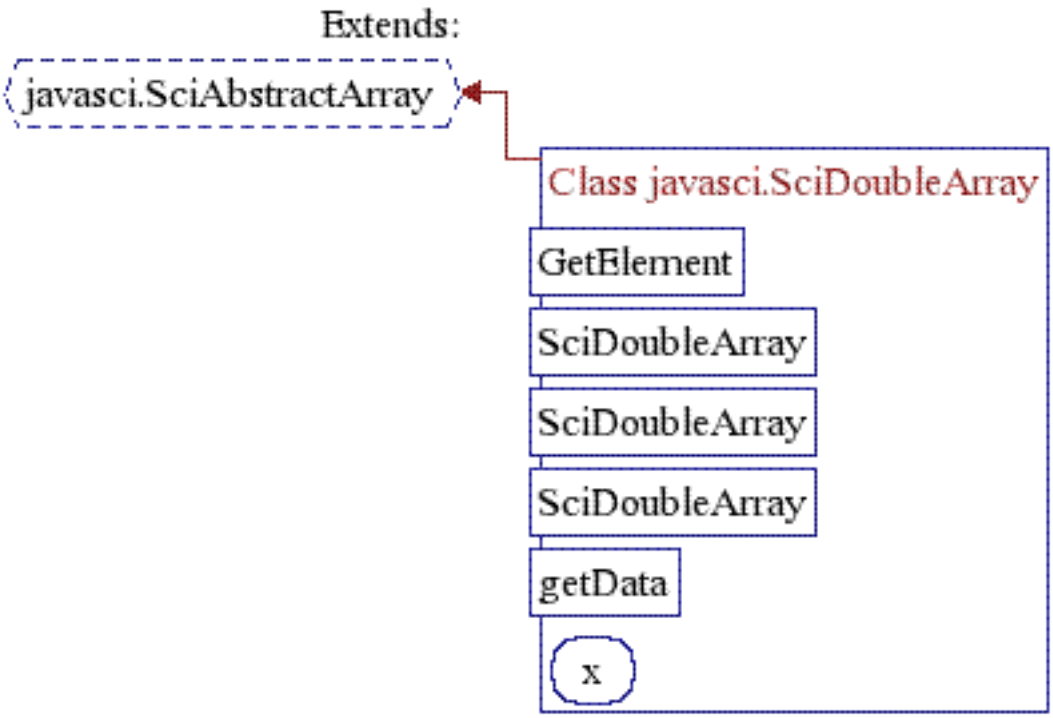
# Authors

A.C

# Name

javasci.SciDoubleArray — Class to use real matrix in Scilab.

# Description




Method Summary :

`public SciDoubleArray(String name,SciDoubleArray Obj)`

`public SciDoubleArray(String name,int r,int c)`

`public SciDoubleArray(String name,int r,int c,double [] x )`Constructor

`public int getNumbersOfRows()`Get number of rows

`public int getNumbersOfCols()` Get number of colons

`public String getName()`Get Name of scilab object

`public double[] getData()` Get Value of scilab object

`public void disp()` disp object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Get()`Get in java object , value of scilab object

`public void Send()`Send to scilab object , value of java object

`public double GetElement(int indr, int indc)`Get a specific element of scilab object

# Examples

```
// See SCI/modules/Javasci/examples directory
```

## See Also

javasci.Scilab , Compile and run with Javasci, SciDouble , SciString , SciStringArray

## Authors

A.C

# Name

javasci.SciInteger — Class to use integer object with scilab

# Description

Method Summary :

`public SciInteger(String name,SciInteger Obj)`

`public SciInteger(String name)` Constructor (if name exists in Scilab and has the same type, variable is imported from Scilab)

`public SciInteger(String name,int Value )`

`public String getName()`Get Name of scilab object

`public int getData()`Get Value of scilab object

`public void Get()`Get in java object , value of scilab object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Send()`Send to scilab object , value of java object

`public void disp()`disp object

# Examples

```
// See SCI/modules/Javasci/examples directory
```

# See Also

javasci.Scilab , Compile and run with Javasci, SciIntegerArray , SciString , SciStringArray

# Authors

A.C

# Name

javasci.SciIntegerArray — Class to use int matrix in Scilab.

# Description

Method Summary :

`public SciIntegerArray(String name,SciIntegerArray Obj)`

`public SciIntegerArray(String name,int r,int c)`

`public SciIntegerArray(String name,int r,int c,int [] x )`Constructor

`public int getNumbersOfRows()`Get number of rows

`public int getNumbersOfCols()` Get number of colons

`public String getName()`Get Name of scilab object

`public int[] getData()` Get Value of scilab object

`public void disp()` disp object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Get()`Get in java object , value of scilab object

`public void Send()`Send to scilab object , value of java object

`public int GetElement(int indr, int indc)`Get a specific element of scilab object

# Examples

```
// See SCI/modules/Javasci/examples directory
```

# See Also

javasci.Scilab , Compile and run with Javasci, SciInteger , SciString , SciStringArray

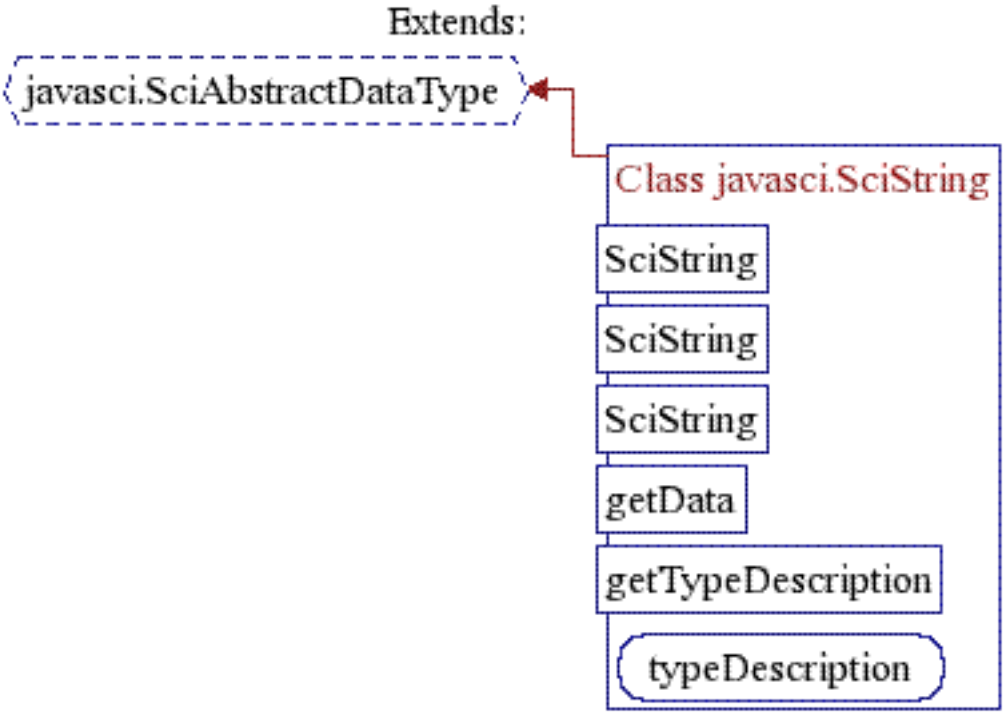
# Authors

A.C

# Name

javasci.SciString — Map a Java String into a Scilab string.

# Description




Method Summary :

`public SciString(String name,SciString Obj)`

`public SciString(String name)` Constructor (if name exists in Scilab and has the same type, variable is imported from Scilab)

`public String getName()`Get Name of scilab object

`public void Get()`Get in java object , value of scilab object

`public String getData()` :Get Value of scilab object

`public void Send()`Send to scilab object , value of java object

`public void disp()` disp object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

# Examples

```
// See SCI/modules/Javasci/examples directory
```

# See Also

javasci.Scilab , Compile and run with Javasci, SciDouble , SciDoubleArray , SciStringArray

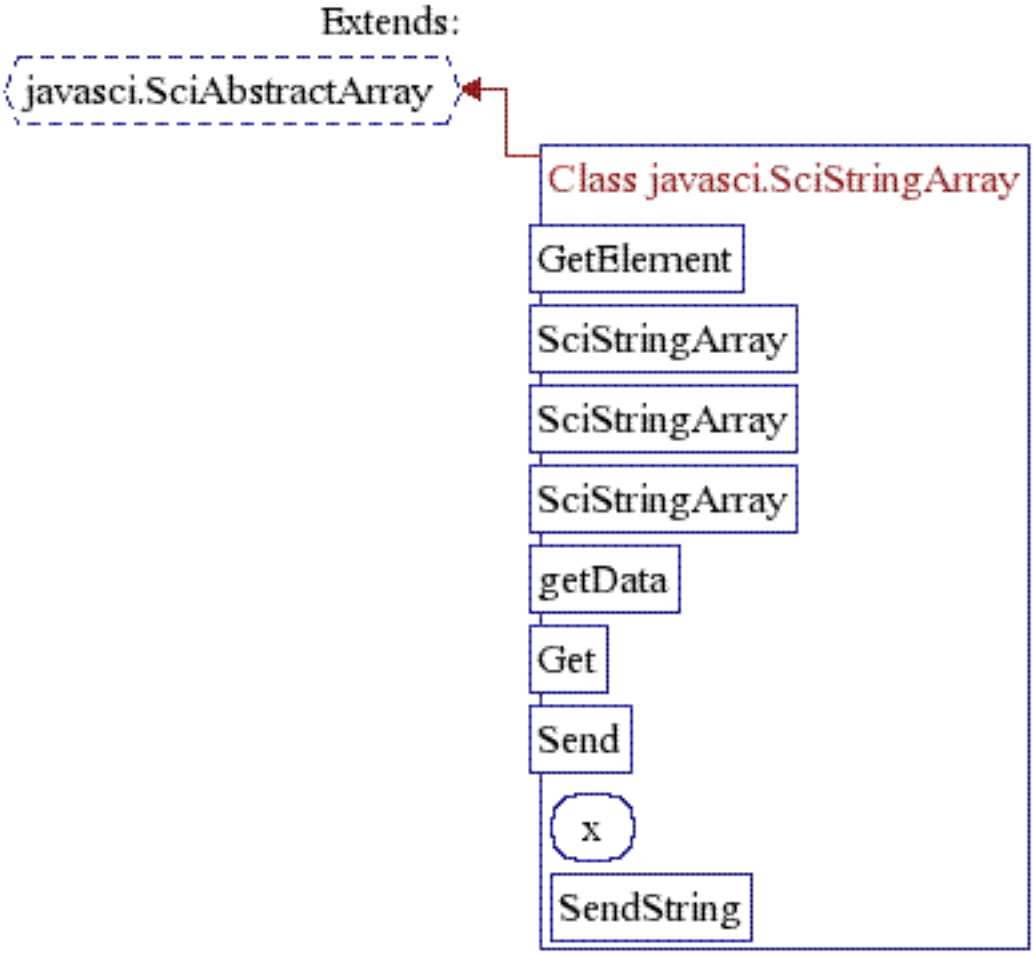
# Authors

A.C

# Name

javasci.SciStringArray — Classe to use String matrix in Scilab.

# Description




Method Summary :

`public SciDoubleArray(String name,SciDoubleArray Obj)`

`public SciStringArray(String name,int r,int c)`

`public SciStringArray(String name,int r,int c,String [] x )`Constructor

`public int getNumbersOfRows()`Get number of rows

`public int getNumbersOfCols()` Get number of colons

`public String getName()`Get Name of scilab object

`public String[] getData()`Get Value of scilab object

`public void Get()` Get in java object , value of scilab object

`public void Send()` Send to scilab object , value of java object

`public void disp()` disp object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public String GetElement(int indr, int indc)`Get a specific element of scilab object

# Examples

```
// See SCI/modules/Javasci/examples directory
```

# See Also

javasci.Scilab , Compile and run with Javasci, SciDouble , SciDoubleArray , SciString

# Authors

A.C

# Name

javasci.Scilab — This class provides the basic methods to execute Scilab code and scripts.

# Description

This class is static. Since the Scilab environnement is persistant, all variables will remain accessible with the Java application.

**Method Summary :**

- *`public static void Events()`* - Execute a Scilab event
- *`public static boolean HaveAGraph()`* -Check if there is any scilab graphic window open (return True if it is the case).
- *`public static boolean Exec(String job)`* - Execute a job in scilab. return true if there is no error.

  Note that the given expression must be consistent by itself. Then, a serie of `Exec` defining a function will not work. Please consider calling it with a single string instruction or using a temporary file with the method `ExecuteScilabScript`.

  For example: `Scilab.Exec("function foo(); disp('bar'); endfunction");` will work when `Scilab.Exec("function foo();"); Scilab.Exec("disp('bar');"); Scilab.Exec("endfunction;");` will not work since each statement being processed independently
- *`public static native boolean ExistVar(String VarName)`* - Detect if VarName exists in Scilab. return true if Varname exist.
- *`public static native int TypeVar(String VarName)`* - Return Scilab type of VarName. See type
- *`public static native int GetLastErrorCode()`* - Return last Error code. See lasterror
- *`public static boolean ExecuteScilabScript(String scilabscriptfilename)`* - Execute a scilab script (.sce) return true if there is no error.
- *`public static native boolean Finish()`* - Terminate scilab (call scilab.quit , close a scilab object)

# Examples

```
// A Scilab / Java example
// Filename: ScilabExample.java

import javasci.Scilab;

public class ScilabExample {
 public static void main(String []args){
  String myVar="myVariable";
  Scilab.Exec(myVar+"=(%pi^4)/90;disp(myVariable);"); // Simple display
  if (Scilab.ExistVar(myVar)) {
   System.out.println("Variable "+myVar+" exists. Type: "+Scilab.TypeVar(myVar)
  }
  if (!Scilab.Exec("disp(plop);")) { // Error
```

```
   System.err.println("Last error: "+Scilab.GetLastErrorCode()); // Error
  }
  Scilab.Finish();
 }
}
```

## See Also

Compile and run with Javasci, SciDouble, SciDoubleArray, SciString, SciStringArray, type, lasterror

## Authors

A.C

# Name

Compile and run with javasci — How to compile a Java application using Javasci

# Description

To compile a Java code based on Javasci, it is only necessary to have javasci.jar defined in the classpath.

For example, with the code defined in the example of this page, the command would be:

on Linux/Unix/MacOSX:

```
$ javac -cp $SCI/modules/javasci/jar/javasci.jar BasicExample.java
```

on Windows:

```
D:\> javac -cp %SCI%\modules\javasci\jar\javasci.jar BasicExample.java
```

To run Scilab, there are a few other things to set up.

Some global variables must me set:

- SCI - Path to Scilab files
  - Linux/Unix/MacOSX:
    - In the binary version of Scilab, SCI will point to /path/to/scilab/share/scilab/
    - In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/source/tree/
  - Windows
    - path of scilab root directory:

      set SCI=C:\program files\scilab-5.1
- LD_LIBRARY_PATH - Paths to libscilab.so and libjavasci.so (or .jnilib...)
  - Linux/Unix/MacOSX:
    - In the binary version of Scilab, SCI will point to /path/to/scilab/lib/scilab/
    - In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/modules/javasci/.libs/ and /path/to/scilab/.libs/
  - Windows: Path to libscilab.dll and javasci.dll
    - equivalent to LD_LIBRARY_PATH on Windows is PATH

      set PATH="%SCI%\bin";%PATH%

To launch the Java Application, you can either provide them with environnement variable

- Linux/Unix/MacOSX:
  - `LD_LIBRARY_PATH=/path/to/libjavasci/ SCI=/path/to/scilab/ java -cp modules/javasci/jar/javasci.jar:. BasicExample`
  - `SCI=/path/to/scilab/ java -Djava.library.path=/path/to/libjavasci/ -cp modules/javasci/jar/javasci.jar:. BasicExample`

or with the arguments

- Windows:

```
set SCI=c:\program files\scilab-5.1

set PATH="%SCI%\bin";%PATH%

D:\java -cp "%SCI%\modules\javasci\jar\javasci.jar";. BasicExample
```

optional options to launch java : `-Djava.compiler=JIT -Xmx256m`

(With these arguments , you launch javasci with same initial options that standard scilab).

Note that two environnement variables are taken in account for specific needs:

- `SCI_DISABLE_TK=1` Disables Tk (Tcl's GUI)
- `SCI_JAVA_ENABLE_HEADLESS=1` Launch Java in headless mode (no AWT/Swing)

# Examples

```
// A simple Java example
// Filename: BasicExample.java

import javasci.Scilab;

public class BasicExample {
 public static void main(String []args){
 Scilab.Exec("disp((%pi^2)/6);");
 }
}
```

# See Also

Javasci, Javasci FAQ, SciDouble, SciDoubleArray, SciString, SciStringArray

# Authors

Allan Cornet

Sylvestre Ledru

# Name

javasci — Call Scilab engine from a Java application

# Description

Scilab offers the possibility to be called from a Java application.

This help describes the features of the javasci API.

# Examples

```
// A simple Java example
// Filename: DisplayPI.java

import javasci.Scilab;

public class DisplayPI {
public static void main(String []args){
Scilab.Exec("disp(%pi);");
}
}
```

# See Also

Compile and run with Javasci, Javasci FAQ, SciDouble, SciDoubleArray, SciString, SciStringArray

# Authors

Allan Cornet

Sylvestre Ledru

# Name

javasci FAQ — The frequently asked questions

# Questions / Answers

1. Running an application based on javasci, I get the error Exception in thread "main" java.lang.NoClassDefFoundError: javasci/Scilab

   javasci.jar is probably not defined in the CLASSPATH. See

# See Also

Javasci, Compile and run with javasci

# Authors

Sylvestre Ledru

# Parte XXXVIII. Interface Maple

# Name

sci2map — Scilab to Maple variable conversion

```
txt=sci2map(a,Map-name)
```

# Parameters

a
: Scilab object (matrix, polynomial, list, string)

Map-name
: string (name of the Maple variable)

txt
: vector of strings containing the corresponding Maple code

# Description

Makes Maple code necessary to send the Scilab variable `a` to Maple : the name of the variable in Maple is `Map-name`. A Maple procedure `maple2scilab` can be found in `SCIDIR/maple` directory.

# Examples

```
txt=[sci2map([1 2;3 4],'a');
     sci2map(%s^2+3*%s+4,'p')]
```

# Parte XXXIX. Dicas de Conversão de Matlab para Scilab

# Name

About_M2SCI_tools — Generally speaking about tools to convert Matlab files to Scilab...

# Description

Scilab 3.0 includes a new version of useful tools to convert Matlab M-files to Scilab.

Taking a Matlab M-file, `mfile2sci` modifies this files so that it can be compiled by Scilab. After that this compiled code is converted to a "tree" of instructions by `macr2tree`. This "tree" is an imbrication of Scilab lists and tlists and is the basis for conversion. Each instruction of this "tree" is converted to Scilab and inference is done to known what are the variables. Once this "tree" is converted to Scilab, code is generated using `tree2code`.

All tlists used for coding this tree (and we call "MSCI tlists") are listed below:

- funcall
  tlist representing a function call created by `Funcall`

  operation
  tlist representing an operation created by `Operation`

  variable
  tlist representing a variable created by `Variable`

  cste
  tlist representing a constant created by `Cste`

  equal
  tlist representing an instruction created by `Equal`

  ifthenelse
  tlist representing an IF/THEN/ELSE control instruction created inside M2SCI kernel functions

  while
  tlist representing a WHILE control instruction created inside M2SCI kernel functions

  selectcase
  tlist representing a SELECT/CASE control instruction created inside M2SCI kernel functions

  for
  tlist representing a FOR control instruction created inside M2SCI kernel functions

The contents of these tlists is described in corresponding help pages.

Operations are converted using a fonction named `%<opcode>2sci` with opcode the Scilab code for this operator. See help page for overloading to have these codes. All these functions are already written and are in directory SCI/modules/m2sci/macros/percent/.

Function calls are converted using a function called `sci_<Matlab_function_name>`. Some of these functions have been written and are in directory SCI/modules/m2sci/macros/sci_files/. We are working on increasing the set of Matlab functions converted. However, everybody can written such functions using help page sci_files.

Inference is done using tlists of type "infer" containing fields:

- dims
  list of dimensions

  type
  "type" tlist

contents
"contents" tlist if a Cell or a Struct

Type is a tlist of type "type" containing fields:

- vtype
  data type

property
property

To have more details about inference see help page for m2scideclare.

# See Also

mfile2sci , translatepaths , overloading , sci_files , Funcall , Operation , Variable , Cste , Infer , Type , Equal , m2scideclare

# Authors

V.C.

# Name

Contents — Create a tree containing contents inference data

```
contents=Contents(list_of_index,list_of_infer)
```

# Parameters

list_of_index
: list of indexes similar to indexes returned by macr2tree.

list_of_infer
: list of "infer" tlists containing inference data for matching index.

contents
: a "contents" tlist

# Description

This function create a `tlist` representing inference data for the contents of a Cell or a Struct when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists". (Unknown=-1 in M2SCI) Please ensure that for each entry you insert in `list_of_index`, you also insert an entry in `list_of_infer`.

# See Also

get_contents_infer , Funcall , Operation , Variable , Cste , Infer , Type , Equal

# Authors

V.C.

# Name

Cste — Create a tree representing a constant

```
const=Cste(value)
```

# Parameters

value
: constante value

const
: a "cste" tlist

# Description

This function create a `tlist` representing a constant when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

# See Also

Funcall , Operation , Variable , Infer , Contents , Type , Equal

# Authors

V.C.

# Name

Equal — Create a tree representing an instruction

```
eq=Equal(lhslist,expression)
```

# Parameters

lhslist
: list of lhs parameters (list of "M2SCI tlists")

expression
: right member of equal (an "M2SCI tlist")

eq
: an "equal" tlist

# Description

This function create a `tlist` representing an instruction when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

# See Also

Funcall , Operation , Variable , Cste , Infer , Contents , Type

# Authors

V.C.

# Name

Funcall — Create a tree representing a function call

```
fc=Funcall(name,lhsnb,rhslist,lhslist)
```

# Parameters

name
: function name (character string)

lhsnb
: number of outputs (constant)

rhslist
: list of inputs (list of "M2SCI tlists")

lhslist
: list of outputs (list of "M2SCI tlists")

fc
: a "funcall" tlist

# Description

This function create a `tlist` representing a function call when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

# See Also

Operation , Variable , Cste , Infer , Contents , Type , Equal

# Authors

V.C.

# Name

Infer — Create a tree containing inference data

```
infer=Infer(varargin)
```

# Parameters

varargin
: data for inference

  varargin(1)
  : list of dimensions default value is list(Unknown,Unknown)

  varargin(2)
  : type ("type" tlist, see Type help page) default value is Type(Unknown,Unknown)

  varargin(3)
  : contents ("contents" tlist, see Contents help page) default value is Contents(list(),list()). This field is only used if represented data is a Cell or a Struct.

infer
: an "infer" tlist

# Description

This function create a `tlist` representing inference data when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists". (Unknown=-1 in M2SCI)

# See Also

Funcall , Operation , Variable , Cste , Contents , Type , Equal

# Authors

V.C.

# Name

Matlab-Scilab_character_strings — Generally speaking about...

# Description

Matlab and Scilab character strings are not considered in the same way. Here is a little talk about differences between them.

Matlab considers a character string as Scilab considers a matrix of characters. For example, a Scilab equivalent for Matlab 'mystring' could be ["m","y","s","t","r","i","n","g"]. So in Scilab, a character string is a object of type string (10) and always have size 1 x 1 but in Matlab, a character string have size equal to 1 x number_of_characters.

Considering this, we can see that a Matlab character string matrix column can only be made of same-size character strings what is not true in Scilab. We can say that a Scilab character string matrix is equivalent to a Matlab cell of character strings.

All these differences can lead to different results while executing same commands in Scilab or in Matlab, particularly for "dimension" functions such as length() or size().

# See Also

mstr2sci

# Authors

V.C.

# Name

Operation — Create a tree representing an operation

```
op=Operation(operator,operands,out)
```

# Parameters

operator
: operator symbol (character string)

operands
: list of operands (list of "M2SCI tlists")

out
: list of outputs (list of "M2SCI tlists")

op
: an "operation" tlist

# Description

This function create a `tlist` representing an operation when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

# See Also

Funcall , Variable , Cste , Infer , Contents , Type , Equal

# Authors

V.C.

# Name

Type — Create a tree containing type inference data

```
tp=Type(vtype,property)
```

# Parameters

vtype
: data type (see m2scideclare)

property
: property of data (see m2scideclare)

tp
: a "type" tlist

# Description

This function create a `tlist` representing type inference data when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists". (Unknown=-1 in M2SCI)

# See Also

Funcall , Operation , Variable , Cste , Infer , Contents , Equal , m2scideclare

# Authors

V.C.

# Name

Variable — Create a tree representing a variable

```
var=Variable(name,infer)
```

# Parameters

var
: variable name (character string)

infer
: inference data (a tlist of type "infer", see Infer help page)

var
: a "variable" tlist

# Description

This function create a `tlist` representing a variable when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

# See Also

Funcall , Operation , Cste , Infer , Contents , Type , Equal

# Authors

V.C.

# Name

get_contents_infer — Search for information in a "M2SCi tlist" contents

```
[infer,pos]=get_contents_infer(m2scitlist,index)
```

# Parameters

m2scitlist
: a "M2SCI tlist"

index
: an index similar to indexes returned by `macr2tree`.

infer
: an "infer" tlist

pos
: position of information in list

# Description

This functions searches for inference informations of a given index in the contents of a Cell or a Struct taken in account the `*`. If no information has been found, returned values are infer=infer() and pos=0.

# See Also

Infer , Contents

# Authors

V.C.

# Name

m2scideclare — Giving tips to help M2SCI...

# Description

The main difficulty for M2SCI (`mfile2sci`) is to find what variables are: dimensions, type...

To help this tool, just add comments beginning with `%m2scideclare` in the M-file to convert, (`%m2sciassume` was used in previous Scilab versions and is now obsolete).

The syntax of this command is:

```
%m2scideclare variable_name|dimensions|data_type|property
```

with :

- • variable_name: name of the variable declared. It can be a Struct field (e.g. `x(1,2).name`) or describe the contents of a Cell using syntax `x(1,2).entries`. NOTE that for Cells and Structs, `*` can be used as an index (see examples below).
  - dimensions: dimensions of the variable declared separated by blanks, if a dimension is unknown, replace it by `?`. NOTE that String dimensions must be similar to Matlab ones e.g. `1 6` for character string 'string'.
  - data_type: data type of the variable which can be:

| m2scideclare data type | Scilab "equivalent" type |
|---|---|
| Double | 1 |
| Boolean | 4 |
| Sparse | 5 |
| Int | 8 |
| Handle | 9 |
| String | 10 |
| Struct | Matlab struct (16) |
| Cell | Matlab cell (17) |
| Void | No type (0) |
| ? | Unknown type |

  - property: property of the variable which can be:

| m2scideclare property | Scilab "equivalent" |
|---|---|
| Real | Real data |
| Complex | Complex data |
| ? | Unknown property |

  This field is ignored for following datatypes: `Cell`, `Struct`, `String` and `Boolean`.

All data given by `m2scideclare` are compared with infered data, in case of conflict, infered data are kept and a warning message is displayed. If you are sure about your data, report a bug.

Some examples are given below:

- • `%m2scideclare var1|2 3|Double|Real`var1 is declared as a 2x3 Double matrix containing real data

- `%m2scideclare var2|2 3 10|Double|Complex`var2 is declared as a 2x3x10 Double hypermatrix containing complex data
- `%m2scideclare var3(1,2).name|1 10|String|?`var3 is declared as a Struct array containing a 1x10 character string in field 'name' of struct at index (1,2)
- `%m2scideclare var4(1,5).entries|1 ?|Boolean|?`var4 is declared as a Cell containing a row boolean vector at index (1,5)
- `%m2scideclare var4(1,6).entries|? ?|Int|?`var4 is declared as a Cell containing a row boolean vector at index (1,5) and integer data at index (1,6)
- `%m2scideclare var5(*,*).name|1 ?|String|?`var5 is declared as a Struct array containing a 1xn character string in all fields 'name'
- `%m2scideclare var6(2,*).entries|1 3|Double|Real`var6 is declared as a Cell array containing a 1x3 double vector in each element of its second row

# Authors

V.C.

# Name

matfile2sci — converts a Matlab 5 MAT-file into a Scilab binary file

```
matfile2sci(mat_file_path,sci_file_path)
```

# Parameters

mat_file_path
: character string containing the path of the Matlab input file

sci_file_path
: character string containing the path of the Scilab output file

# Description

Converts a Matlab 5 MAT-file into a Scilab binary file compatible with the function `load` The Matlab data types are converted into the Scilab equivalents.

# See Also

loadmatfile , load , mfile2sci

# Authors

Serge Steer (INRIA)

# Bibliography

This function has been developped according to the document "MAT-File Format": >Mat-File Format [http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf]

# Name

mfile2sci — Matlab M-file to Scilab conversion function

```
mfile2sci([M-file-path [,result-path [,Recmode [,only-double [,verbose-mode [,p
```

# Parameters

M-file-path
: a character string which gives the path of Matlab M-file to convert

result-path
: a character string which gives the directory where the result has to be written. Default value is current directory.

Recmode
: Boolean flag, used by translatepaths function for recursive conversion. Must be %F to convert a single mfile. Default value : %f

only-double
: Boolean flag, if %T mfile2sci considers that numerical function have been used only with numerical data (no Scilab overloading function is needed). Default value: %F

verbose-mode
: display information mode

  0
  : no information displayed

  1
  : information written as comment is resulting SCI-file

  2
  : information written as comment is resulting SCI-file and in logfile

  3
  : information written as comment is resulting SCI-file, in logfile and displayed in Scilab window

prettyprintoutput
: Boolean flag, if %T generated code is beautified. Default value: %F

# Description

M2SCI (and particularly mfile2sci) is Matlab M-file to Scilab function conversion tools. It tries whenever possible to replace call to Matlab functions by the equivalent Scilab primitives and functions.

To convert a Matlab M-file just enter the Scilab instruction: mfile2sci(file)

where file is a character string giving the path name of the M-file mfile2sci will generate three files in the same directory

<function-name>.sci
: the Scilab equivalent of the M-file

<function-name>.cat
: the Scilab help file associated to the function

sci_<function-name>.sci
: the Scilab function required to convert the calls to this Matlab M-file in other Matlab M-files. This function may be improved "by hand". This function is only useful for conversion not for use of translated functions.

Some functions like eye, ones, size, sum,... behave differently according to the dimension of their arguments. When mfile2sci cannot infer dimensions it replaces these function call by a call to an emulation function named mtlb_<function_name>. For efficiency these functions may be replaced by the proper scilab equivalent instructions. To get information about replacement, enter: help mtlb_<function_name> in Scilab command window

Some other functions like plot, has no straightforward ï¿½quivalent in scilab. They are also replaced by an emulation function named mtlb_<function_name>.

When translation may be incorrect or may be improved mfile2sci adds a comment which begins by "//!" (according to verbose-mode)

When called without rhs, `mfile2sci()` launches a GUI to help to select a file/directory and options.

# Examples

```
// Create a simple M-file
write(TMPDIR+'/rot90.m',['function B = rot90(A,k)'
 '[m,n] = size(A);'
 'if nargin == 1'
 '    k = 1;'
 'else'
 '    k = rem(k,4);'
 '    if k < 0'
 '        k = k + 4;'
 '    end'
 'end'
 'if k == 1'
 '    A = A.'';'
 '    B = A(n:-1:1,:);'
 'elseif k == 2'
 '    B = A(m:-1:1,n:-1:1);'
 'elseif k == 3'
 '    B = A(m:-1:1,:);'
 '    B = B.'';'
 'else'
 '    B = A;'
 'end']);

// Convert it to scilab
mfile2sci(TMPDIR+'/rot90.m',TMPDIR)

// Show the new code
write(%io(2),read(TMPDIR+'/rot90.sci',-1,1,'(a)'))

// get it into scilab
exec(TMPDIR+'/rot90.sci')

// Execute it
m=rand(4,2);rot90(m,1)
```

# See Also

translatepaths

# Authors

V. Couvert
S. Steer

# Name

sci_files — How to write conversion functions

# Description

To convert calls to Matlab functions, `mfile2sci` uses a function called `sci_<Matlab_function_name>`. All these functions are defined in `sci_files` in directory SCI/modules/m2sci/macros/sci_files/. The set of `sci_files` given in Scilab distribution does not allow to convert calls to all Matlab functions yet. However, a Scilab user can add `sci_files` (for Matlab functions or for user defined functions) to Scilab using the following tips.

In M2SCI, a function call is considered as a "tree" (it is also the case for the instructions of the file to convert), represented in Scilab by a `tlist` with following fields:

- name
  Matlab function name

  lhsnb
  number of Matlab function output parameters

  lhs
  list of Matlab function output parameters

  rhs
  list of Matlab function input parameters

A `sci_function` has one input called `tree` which is also the output of the function. A `sci_function` has to convert this incoming "tree" so that it is compatible with Scilab by changing name, lhsnb, lhs and/or rhs. The other task that has to be done by this function is inference. Incoming tree contains inference data in its lhs that have to be updated with what can be infered for the outputs of this function.

Some useful functions have been written to help to create M2SCI tlists while writing this conversion function:

- Funcall
  create a tree representing a function call

  Operation
  create a tree representing an operation

  Variable
  create a tree representing a variable

  Cste
  create a tree representing a constante value

  Infer
  create a tree representing inference data

  Type
  create a tree representing type for inference

  Equal
  create a tree representing an instruction

Some other functions have been designed to get properties of operands/inputs. Considering A is tlist used in macro tree, you can use the following functions:

| Function | returns `%T` if... |
|---|---|

| | |
|---|---|
| `is_empty(A)` | all dimensions of A are 0 |
| `not_empty(A)` | all dimensions of A are known and at least one dimension of A is not 0 |
| `is_a_scalar(A)` | all dimensions of A are 1 |
| `not_a_scalar(A)` | all dimensions of A are known and at least one dimension of A is not 1 |
| `is_a_vector(A)` | all dimensions of A are known and all dimensions of A but one are equal to 1 |
| `not_a_vector(A)` | all dimensions of A are known and at least two dimensions of A are greater than one |
| `is_real(A)` | A is real |
| `is_complex(A)` | A is complex |
| `isdefinedvar(A)` | A is a variable already created in M-file currently converted |
| `allunknown(A)` | all dimensions of A are unknown |

Some other functions have been written for specific needs while writing conversion files:

- first_non_singleton
  is an equivalent to firstnonsingleton for an M2SCI tlist. Calling sequence: `dim = first_non_singleton(A)`

  gettempvar
  generates a temporary variable having a name which does not already exist. Calling sequence: `v = gettempvar()`

  insert
  allows to insert instructions. Calling sequence: `insert(Equal(...),opt)` with `opt~=1` to insert before current instruction and `opt=1` to insert after it.

  getoperands
  can be used to get each operand as a variable. Calling sequence: `[A,B] = getoperands(operation_tlist)`

  getrhs
  can be used to get each parameter as a variable. Calling sequence: `[A,...] = getrhs(funcall_tlist)`

  convert2double
  change type of input when this type is not implemented for a particular function is Scilab. Calling sequence: `A = convert2double(A)`

To have more information about how to write such files, refer to directory SCI/modules/m2sci/macros/sci_files/ which gives many examples from very simple ones (e.g. sci_abs.sci) to very complex ones (e.g. sci_zeros.sci).

# See Also

m2scideclare , Funcall , Operation , Variable , Cste , Infer , Type , Equal

# Authors

V.C.

# Name

translatepaths — convert a set of Matlab M-files directories to Scilab

```
translatepaths(dirs_path [,res_path])
```

# Parameters

dirs_path
: a character string vector which gives the paths of Matlab M-file directories to convert

res_path
: a character string which gives the path of the directory where the Scilab functions are written to. Default value is current directory.

# Description

`translatepaths`, converts all Matlab M-file contained in a set of directories to Scilab functions. Each function is converted by `mfile2sci`.

Trace of conversion information is stored in a file named `"log"` in the `res_path` directory

When called without rhs, `translatepaths()` launches a GUI to help to select a file/directory and options.

# See Also

mfile2sci

# Authors

V. Couvert
S. Steer

# Parte XL. Interfaces com Tcl/Tk

# Name

ScilabEval — tcl instruction : Evaluate a string with scilab interpreter

```
ScilabEval instruction
ScilabEval instruction "seq"
ScilabEval instruction "sync"
ScilabEval instruction "sync" "seq"
ScilabEval "flush"
```

# Parameters

instruction
: tcl string character contains a Scilab instruction to evaluate with the current Scilab interpreter.

# Description

This function must be called in a tcl/tk script executed from Scilab. It allows to associate Scilab actions to tcl/tk widgets (graphic objects) or to use Scilab to perform some computations within a tcl script.

ScilabEval instruction
: If the `ScilabEval instruction` syntax is used, the `instruction` is first stored in a FIFO queue, `ScilabEval` returns immediately. Scilab executes the queued `instructions` when possible (it should be at the prompt but also at the end of each instructions of the currently running function) in the order they were submitted. This syntax can be used to associate Scilab actions to tcl/tk widgets but not into a tcl script executed by TCL_EvalFile or TCL_EvalStr because in this situation the Scilab interpreter is blocked up to the end of the script. Note that with the `ScilabEval instruction` syntax, if there are many `ScilabEval` commands stored in the queue the execution of the second one can be started in the middle of the execution of the first one (in particular if the first one contains more than a simple expression).

  If the `"seq"` option is added, the associated instruction evaluation should be finished (or paused) before the next queued instruction evaluation can be started. The next callback stored in the command queue will only be taken into account when the current one will be finished or paused.

ScilabEval instruction "sync"
: If the `ScilabEval instruction "sync"` syntax is used, the instruction is executed immediately (not queued) and the `ScilabEval`returns when the `instruction` evaluation is finished. The scilab `instruction` evaluation may be interrupted by new or queued commands.

  If the `"seq"` option is added, the associated instruction evaluation should be finished (or paused) before any queued instruction evaluation can be started. The scilab `instruction` evaluation may not be interrupted by new or queued commands (except if it is paused).

ScilabEval "flush"
: If the `ScilabEval "flush"` syntax is used, all the previously queued `instructions` are executed immediately and the ScilabEval returns when the execution is finished. Each `instruction` is executed with the option used at the time of queuing up (i.e. `seq` or no option).

The evaluation context of all these cases is the current Scilab context when the`instruction` evaluation starts.

# Examples

```
//Callbacks and "seq" option usage
```

```
//create tcl instructions
tcl_script=['toplevel .w1'
            'button .w1.b -text ""Click here to execute without seq option""  -c
            'button .w1.b1 -text ""Click here to execute with seq option""  -con
            'pack .w1.b .w1.b1'
            'proc  WithoutSeq {} { ';
            '  ScilabEval ""cont=%f;;cont=%t;"" '
            '  ScilabEval ""if cont then disp(''ok''),else disp(''wrong'');end;e
            '}'
            'proc WithSeq {} { ';
            '  ScilabEval ""cont=%f;;cont=%t;""  ""seq""'
            '  ScilabEval ""if cont then disp(''ok''),else disp(''wrong'');end;e
            '}'];
mputl(tcl_script,TMPDIR+'/test.tcl') //write them to a file

// Execute the tcl script
cont=%f;
TCL_EvalFile(TMPDIR+'/test.tcl');;

//scripts and "sync" option usage

//----------------without "sync"----------------
tcl_script=['  set t ""0""'
            '  while {$t != ""10""} {'
            '     ScilabEval ""a=$t;mprintf(''%d '',a);""'
            '     incr t'
            '  }'];

mputl(tcl_script,TMPDIR+'/test.tcl') //write them to a file

// Execute the tcl script
TCL_EvalFile(TMPDIR+'/test.tcl');mprintf('TCL_EvalFile finished\n');

// The ScilabEval are executed  after the and of TCL_EvalFile

//----------------with "sync"----------------
tcl_script=['  set t ""0""'
            '  while {$t != ""10""} {'
            '     ScilabEval ""a=$t;mprintf(''%d '',a);"" ""sync""'
            '     incr t'
            '  }'];

mputl(tcl_script,TMPDIR+'/test.tcl') //write them to a file

// Execute the tcl script
TCL_EvalFile(TMPDIR+'/test.tcl');mprintf('TCL_EvalFile finished\n');

// The ScilabEval are executed  synchronously with TCL_EvalFile
```

# See Also

TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar

# Authors

Bertrand Guiheneuf

# Name

TCL_CreateSlave — Create a TCL slave interpreter

```
TCL_CreateSlave(slaveName[, isSafe])
```

# Parameters

slaveName
: String: Name of the TCL slave interpreter to create.

isSafe
: Boolean: `%T` to create a safe slave interpreter, `%F` otherwise. The default value is `%F`. A safe slave is not allowed to perform some operations, see the TCL documentation for more information.

# Description

This routine allows to create a TCL slave interpreter.

# Examples

```
TCL_CreateSlave("TCLinterp")
TCL_SetVar("a","r","TCLinterp")
TCL_ExistVar("a","TCLinterp")
TCL_ExistVar("a")
TCL_DeleteInterp("TCLinterp")

TCL_CreateSlave("TCLinterp", %T)
TCL_SetVar("a","r","TCLinterp")
TCL_ExistVar("a","TCLinterp")
TCL_ExistVar("a")
TCL_DeleteInterp("TCLinterp")
```

# See Also

TCL_SetVar , TCL_ExistVar , TCL_DeleteInterp

# Authors

Allan CORNET

V.C.

# Name

TCL_DeleteInterp — delete TCL interpreter

```
TCL_DeleteInterp(interp)
TCL_DeleteInterp()
```

# Parameters

interp
: character string parameter. Name of the slave tcl interpreter to delete. If not provided, it defaults to the main tcl interpreter created by Scilab.

# Description

This routine allows to delete a TCL slave interpreter or the main scilab TCL interpreter.

# Examples

```
TCL_SetVar("Scilab","OK")
TCL_ExistVar("Scilab")
TCL_DeleteInterp()
TCL_ExistVar("Scilab")
TCL_CreateSlave('BisInterp')
TCL_ExistInterp('BisInterp')
TCL_SetVar("Scilab","OK",'BisInterp')
TCL_ExistVar("Scilab",'BisInterp')
TCL_DeleteInterp('BisInterp')
TCL_ExistInterp('BisInterp')
```

# See Also

TCL_SetVar , TCL_ExistVar , TCL_CreateSlave , TCL_ExistInterp

# Authors

Allan CORNET

# Name

TCL_EvalFile — Reads and evaluate a tcl/tk file

```
TCL_EvalFile(filename [,interp])
```

# Parameters

filename
: character string. Contains the name of the file to read and evaluate.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

# Description

With this routine, one can read and evaluate the content of a file containing tcl/tk scripts. This allows to create powerful tk interfaces.

The filename might be relative or absolute.

# Advantages and drawbacks of this functionality

This routines allows to use directly tcl/tk scripts. This thus allows, for instance to use Interface Builders such as SpecTcl to design the interface. The interfaces built directly with tcl/tk scripts are much faster than the ones built with the Scilab Graphic Object library provided with tksci (see uicontrol for example). Indeed, those Objects are warpings around tk graphic widgets. Nevertheless, this way of creating graphic user interface should only be used when one aims at addressing directly specific tk/tcl features. There are two main reasons for this. First of all, there is no simple way to manipulate Scilab objects from within a tcl/tk script. Thus, the interface designer has to write two sets of callbacks routines. One to describe the changes occuring in the interface when the user acts on the widgets. The second set of call routines will perform the (pure) Scilab reactions to the user actions.

Here is an example: Suppose you design a scrollbar corresponding to a spline tension value. You want the spline to be displayed in a graphic windows and updated each time the user moves the scrollbar. At the same time, you want the value of this tension parameter to be displayed within the Interface. You will have to write a first tcl/tk (callback) function which will be automatically called by the tk scrollbar ("-command" option). This callback function will update the displayed value of the parameter in the interface and will then call the scilab routine ("ScilabEval" command) to update the graph.

# Remarks on the tcl/tk script style

Because Scilab manages the tcl/tk events, it creates the root window ".", this window should not be destroyed nor directly used by your tcl/tk scripts. You should thus always create your own toplevel windows. Moreover, since this module was written at a time when namespaces didn't exist, some variables defined by scilab tcl/tk scripts could collide your code. Running your scripts in a slave interpreter may help in such a case.

# Examples

```
TCL_EvalFile(SCI+"/modules/tclsci/demos/tk/puzzle");
```

# See Also

ScilabEval , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_ExistVar , TCL_UnsetVar , TCL_UpVar

# Authors

Allan CORNET

# Name

TCL_EvalStr — Evaluate a string whithin the Tcl/Tk interpreter

```
TCL_EvalStr(str [,interp])
res = TCL_EvalStr(str [,interp])
```

# Parameters

str
: string or matrix of strings, contains a Tcl/Tk script in each element.

interp
: optional character string parameter. Name of the slave Tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main Tcl interpreter created by Scilab.

res
: result of the evaluation, if it is successful. This is a character string matrix giving the evaluation result for each element of the input argument str

# Description

This routine allows to evaluate Tcl/Tk instructions with the Tcl/Tk interpreter launched with Scilab (when the `interp` parameter is not given), or in a slave interpreter.

When Tcl/Tk support is enabled in Scilab, you can evaluate Tcl/Tk expression from Scilab interpreter. In fact, Scilab launches a main Tcl/Tk interpreter. The Scilab instruction `TCL_EvalStr` can be used to evaluate expressions without having to write Tcl/Tk instructions in a separated file (this capability is provided by `TCL_EvalFile`).

# Examples

```
//with one call
TCL_EvalStr(["toplevel .foo1"
             "label .foo1.l -text ""TK married Scilab !!!"""
             "pack .foo1.l"
             "button .foo1.b -text close -command {destroy .foo1}"
             "pack .foo1.b"])

//step by step (debugging)
TCL_EvalStr("toplevel .foo2");

// creates a toplevel TK window.
TCL_EvalStr("label .foo2.l -text ""TK married Scilab !!!""");

// create a static label
TCL_EvalStr("pack .foo2.l");

// pack the label widget. It appears on the screen.
text="button .foo2.b -text close -command {destroy .foo2}";
TCL_EvalStr(text);
TCL_EvalStr("pack .foo2.b");

//kill the windows by program
TCL_EvalStr("destroy .foo1");
TCL_EvalStr("destroy .foo2");
```

```
//with one call, and in a slave interpreter
TCL_CreateSlave('TCLSlave');
TCL_EvalStr('set test ""in Slave TCL Interp""','TCLSlave');
TCL_GetVar('test','TCLSlave')

TCL_DeleteInterp('TCLSlave')

// return a result
res = TCL_EvalStr("expr 1+1")
res = TCL_EvalStr("tk_messageBox -message Hello -type okcancel")
res = TCL_EvalStr(["expr 4+5" "lsearch -all {a b c a b c} c" ; "list [list a b c
```

# See Also

ScilabEval , TCL_EvalFile , TCL_GetVar , TCL_SetVar , TCL_ExistVar , TCL_UnsetVar , TCL_UpVar

# Authors

Allan CORNET

# Name

TCL_ExistArray — Return %T if a tcl array exists

```
OK=TCL_ExistArray(arrayname [,interp])
```

# Parameters

arrayname
: character string. Contains the name of the tcl/tk array.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok
: boolean. %T if arrayname exists.

# Description

This routine allows to test if a tcl array exists.

# Examples

```
TCL_ExistVar("A")
a=["A","B","C";"D","E","F"];
TCL_SetVar("A",a)
TCL_ExistVar("A")
TCL_ExistArray("A")
```

# See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_UnsetVar , TCL_UpVar , TCL_CreateSlave

# Authors

Allan CORNET

# Name

TCL_ExistInterp — Return %T if a tcl slave interperter exists

```
OK=TCL_ExistInterp(interp)
```

# Parameters

interp
: character string parameter. Name of the slave tcl interpreter.

ok
: boolean. %T if TCL interpreter exists.

# Description

This routine allows to test if TCL interpreter exists.

# Examples

```
TCL_ExistInterp('SlaveInterp')
TCL_CreateSlave('SlaveInterp')
TCL_ExistInterp('SlaveInterp')
TCL_DeleteInterp('SlaveInterp')
```

# See Also

TCL_CreateSlave , TCL_DeleteInterp

# Authors

Allan CORNET

# Name

TCL_ExistVar — Return %T if a tcl variable exists

```
OK=TCL_ExistVar(varname [,interp])
```

# Parameters

varname
: character string. Contains the name of the tcl/tk variable.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok
: boolean. %T if varname exists.

# Description

This routine allows to test if a tcl variable exists.

# Examples

```
TCL_SetVar("Scilab","OK")
TCL_GetVar("Scilab")
TCL_UnsetVar("Scilab")
TCL_ExistVar("Scilab")

TCL_SetVar("aa",1)
TCL_CreateSlave('SlaveInterp');
TCL_SetVar("aa",2,'SlaveInterp')
TCL_ExistVar("aa")
TCL_GetVar("aa")
TCL_UnsetVar("aa")
TCL_GetVar("aa",'SlaveInterp')
TCL_UnsetVar("aa",'SlaveInterp')
TCL_ExistVar("aa",'SlaveInterp')
TCL_DeleteInterp('SlaveInterp')
```

# See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_UnsetVar , TCL_UpVar , TCL_CreateSlave

# Authors

Allan CORNET

# Name

TCL_GetVar — Get a tcl/tk variable value

```
value=TCL_GetVar(Varname [,interp])
```

# Parameters

varname
: character string. Contains the name of the tcl/tk variable.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

value
: may be a character string or a strings matrix. Contains the value of the tcl/tk variable `varname` in the interpreter `interp`.

# Description

When tcl/tk support is enabled in Scilab, this routine can be used to retreive the value of a tcl/tk variable.

# Examples

```
TCL_EvalStr("toplevel .tst1");

// creates a toplevel TK window.
TCL_EvalStr("entry .tst1.e -textvariable tvar");

// create an editable entry
TCL_EvalStr("set tvar foobar");

// set the entry value
TCL_EvalStr("pack .tst1.e");

// pack the entry widget. It appears on the screen.
text=TCL_GetVar("tvar")

// retrieve the variable value
// change the entry text and repeat the last command ...
//delete the toplevel TK window.
TCL_EvalStr("destroy .tst1")

//-----------------------------------------------------
a=["A","B","C";"D","E","F"];
TCL_SetVar("A",a)
AfromTCL=TCL_GetVar("A")

//-----------------------------------------------------
b=[6,4,1;2,3,5];
TCL_SetVar("B",b)
BfromTCL=TCL_GetVar("B")

//-----------------------------------------------------
```

```
TCL_SetVar("StringTCL","string")
StringFromTCL=TCL_GetVar("StringTCL")

//-----------------------------------------------------
TCL_SetVar("ScalarTCL",1.22)
ScalarFromTCL=TCL_GetVar("ScalarTCL")

//-----------------------------------------------------
// Examples with a slave interpreter
//-----------------------------------------------------
a=['AA','BB','CC';'DD','EE','FF'];
TCL_CreateSlave('SlaveInterp')
TCL_SetVar("A_slave",a,'SlaveInterp')
AfromTCL_slave=TCL_GetVar('A_slave','SlaveInterp')
TCL_DeleteInterp('SlaveInterp')

//-----------------------------------------------------
b=[66,44,11;22,33,55];
TCL_CreateSlave('SlaveInterp1')
TCL_SetVar("B_slave",b,'SlaveInterp1')
BfromTCL_slave=TCL_GetVar('B_slave','SlaveInterp1')
TCL_DeleteInterp('SlaveInterp1')

//-----------------------------------------------------
TCL_CreateSlave('SlaveInterp2')
TCL_SetVar("StringTCL_slave","string in slave interpreter",'SlaveInterp2')
StringFromTCL_slave=TCL_GetVar("StringTCL_slave",'SlaveInterp2')
TCL_DeleteInterp('SlaveInterp2')

//-----------------------------------------------------
TCL_CreateSlave('SlaveInterp3')
TCL_SetVar("ScalarTCL_slave",1.22,'SlaveInterp3')
ScalarFromTCL_slave=TCL_GetVar("ScalarTCL_slave",'SlaveInterp3')
TCL_DeleteInterp('SlaveInterp3')
```

# See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_SetVar , TCL_ExistVar , TCL_UnsetVar , TCL_UpVar , TCL_CreateSlave , TCL_DeleteInterp

# Authors

Allan CORNET

# Name

TCL_GetVersion — get the version of the TCL/TK library at runtime.

```
TCL_GetVersion()
ret=TCL_GetVersion('numbers')
```

# Description

get the version of the TCL/TK library at runtime.

ret=TCL_GetVersion('numbers') returns a matrix with the version of the TCL/TK library at runtime.

# Examples

```
TCL_GetVersion()
TCL_GetVersion("numbers")
```

# Authors

Allan CORNET

# Name

TCL_SetVar — Set a tcl/tk variable value

```
TCL_SetVar(varname, value [,interp])
```

# Parameters

varname
: character string. Contains the name of the tcl/tk variable to set.

value
: may be a character string, a scalar, a real or string matrix (m x n). Contains the value to give to the tcl/tk variable.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

# Description

This routine allows to set a variable within a tcl/tk interpreter. When tcl/tk support is enabled in scilab, this routine can be used to set up the value of a tcl/tk variable. This can be useful to change some value in the tcl/tk interpreter without having to build a tcl/tk instruction (and use `TCL_EvalStr`).

# Examples

```
TCL_EvalStr("toplevel .tst1");

// creates a toplevel TK window.
TCL_EvalStr("entry .tst1.e -textvariable tvar");

// create an editable entry
TCL_EvalStr("set tvar foobar");

// set the entry value
TCL_EvalStr("pack .tst1.e");

// pack the entry widget. It appears on the screen.
text=TCL_GetVar("tvar")

// retrieve the variable value
// change the entry text and repeat the last command ...
//delete the toplevel TK window.
TCL_EvalStr("destroy .tst1")

//-----------------------------------------------------
a=["A","B","C";"D","E","F"];
TCL_SetVar("A",a)
AfromTCL=TCL_GetVar("A")

//-----------------------------------------------------
b=[6,4,1;2,3,5];
TCL_SetVar("B",b)
BfromTCL=TCL_GetVar("B")
```

```
//-----------------------------------------------------
TCL_SetVar("StringTCL","string")
StringFromTCL=TCL_GetVar("StringTCL")

//-----------------------------------------------------
TCL_SetVar("ScalarTCL",1.22)
ScalarFromTCL=TCL_GetVar("ScalarTCL")

//-----------------------------------------------------
// Examples with a slave interpreter
//-----------------------------------------------------
TCL_CreateSlave('TCLSlave')
a=['AA','BB','CC';'DD','EE','FF'];
TCL_SetVar("A_slave",a,'TCLSlave')
AfromTCL_slave=TCL_GetVar('A_slave','TCLSlave')
TCL_DeleteInterp('TCLSlave')

//-----------------------------------------------------
TCL_CreateSlave('TCLSlave')
b=[66,44,11;22,33,55];
TCL_SetVar("B_slave",b,'TCLSlave')
BfromTCL_slave=TCL_GetVar('B_slave','TCLSlave')
TCL_DeleteInterp('TCLSlave')

//-----------------------------------------------------
TCL_CreateSlave('TCLSlave')
TCL_SetVar("StringTCL_slave","string in slave interpreter",'TCLSlave')
StringFromTCL_slave=TCL_GetVar("StringTCL_slave",'TCLSlave')
TCL_DeleteInterp('TCLSlave')

//-----------------------------------------------------
TCL_CreateSlave('TCLSlave')
TCL_SetVar("ScalarTCL_slave",1.22,'TCLSlave')
ScalarFromTCL_slave=TCL_GetVar("ScalarTCL_slave",'TCLSlave')
TCL_DeleteInterp('TCLSlave')
```

# See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_ExistVar , TCL_UnsetVar , TCL_UpVar , TCL_CreateSlave , TCL_DeleteInterp

# Authors

Allan CORNET

# Name

TCL_UnsetVar — Remove a tcl variable

```
OK=TCL_UnsetVar(varname [,interp])
```

# Parameters

varname
: character string. Contains the name of the tcl/tk variable to unset.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok
: boolean. %T if varname was deleted.

# Description

This routine allows to unset a tcl variable.

# Examples

```
TCL_SetVar("Scilab","OK")
TCL_GetVar("Scilab")
TCL_UnsetVar("Scilab")
TCL_ExistVar("Scilab")

TCL_CreateSlave('InterpSlave');
TCL_SetVar("Scilab","Good",'InterpSlave')
TCL_GetVar("Scilab",'InterpSlave')
TCL_UnsetVar("Scilab",'InterpSlave')
TCL_ExistVar("Scilab",'InterpSlave')
TCL_DeleteInterp('InterpSlave')
```

# See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_ExistVar , TCL_UpVar , TCL_CreateSlave , TCL_DeleteInterp

# Authors

Allan CORNET

# Name

TCL_UpVar — Make a link from a tcl source variable to a tcl destination variable

```
OK=TCL_UpVar(varname1,varname2,[interp])
```

# Parameters

varname1
: character string. Contains the name of the tcl source variable.

varname2
: character string. Contains the name of the tcl destination variable.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok
: boolean. %T if it is ok.

# Description

Make a link from a tcl source variable to a tcl destination variable.

# Examples

```
TCL_SetVar("Scilab","OK")
TCL_UpVar("Scilab","ScilabBis")
TCL_GetVar("ScilabBis")
TCL_SetVar("Scilab","NOK")
TCL_GetVar("ScilabBis")
TCL_SetVar("ScilabBis","modified")
TCL_GetVar("ScilabBis")
TCL_GetVar("Scilab")

TCL_CreateSlave('InterpBis')
TCL_SetVar("Scilab","Good",'InterpBis')
TCL_UpVar("Scilab","ScilabBis",'InterpBis')
TCL_GetVar("ScilabBis",'InterpBis')
TCL_SetVar("Scilab","Not good",'InterpBis')
TCL_GetVar("ScilabBis",'InterpBis')
TCL_SetVar("ScilabBis","modified again",'InterpBis')
TCL_GetVar("ScilabBis",'InterpBis')
TCL_GetVar("Scilab",'InterpBis')
TCL_DeleteInterp('InterpBis')
```

# See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_ExistVar , TCL_UnsetVar , TCL_CreateSlave , TCL_DeleteInterp

# Authors

Allan CORNET

# Name

browsevar — Scilab variable browser

```
browsevar()
```

# Description

`browsevar` is an embedded Scilab variable browser written in TCL/TK.

`browsevar` can show all variables and function (like who). `browsevar` can be costumized to show all or some type of variable. It's also posibble exclude variable names.

# Examples

```
browsevar();
```

# Authors

Jaime Urzua

# Name

config — Scilab general configuration.

```
config()
```

# Description

`config()` allows configure scilab parameters like lines to display, stacksize, %ODEOPTIONS.

# Authors

Jaime Urzua

# Name

editvar — Scilab variable editor

```
editvar varname
```

# Parameters

varname
: variable name. The variable must exist in scilab.

# Description

`editvar` is an embedded Scilab variable editor written in TCL/TK.

`editvar` can edit the following variable type: real or complex constant matrix (type 1), boolean matrix (type 4) an matrix of character strings (type 10).

# Examples

```
a=rand(10,10);
editvar a;
b=['hello';'good bye'];
editvar b;
```

# Authors

Jaime Urzua

# Name

tk_getdir — dialog to get a directory path

```
path=tk_getdir([Title="string"])
path=tk_getdir(startdir,[Title="string"])
path=tk_getdir(startdir,windowtitle)
```

# Parameters

startdir
: a character string which gives the initial directory used for directory search. By default tk_getdir uses the previously selected directory.

path
: is the user selected file path if user answers "Ok" or the " " string if user answers "Cancel"

Title="string"
: Optional argument which gives the title for the tk_getdir window. `Warning`: Use the new variable `Title` instead of the old variable `title`.

# Description

WARNING: this function is deprecated (see uigetdir as a replacement of tk_getdir). It will be removed in Scilab 5.3.

Creates a dialog window for file selection.

# Examples

```
tk_getdir()
tk_getdir("SCI/modules/")
tk_getdir(Title="Choose a directory name")
```

# See Also

uigetdir , uigetfile , file , fileinfo

# Name

tk_savefile — dialog to get a file path for writing

```
path=tk_savefile([Title='string'])
path=tk_savefile(file_mask,[Title='string'])
path=tk_savefile(file_mask,dir,[Title='string'])
path=tk_savefile(file_mask,dir,'string')
```

# Parameters

file_mask
: a character string which gives the file mask to use for file selection. `file_mask` is written with Unix convention. the default value is '*'.

dir
: a character string which gives the initial directory used for file search. by default tk_savefile uses the previously selected directory.

path
: is the user selected file path if user answers "Ok" or the " " string if user answers "Cancel"

Title='string'
: Optional argument which gives the title for the tk_savefile window. `Warning`: Use the new variable `Title` instead of the old variable `title`.

# Description

WARNING: this function is obsolete (see uigetfile as a replacement of tk_savefile). It will be removed in Scilab 5.3.

Creates a dialog window for output file selection

# Examples

```
tk_savefile()
tk_savefile('*.sci','SCI/modules/graphics/macros')
tk_savefile(Title='Choose a file name ')
```

# See Also

uigetfile , uigetdir , file , fileinfo

# Name

winclose — close windows created by sciGUI

```
winclose(winIds)
```

# Parameters

winIds
: matrix of integer greater than 0, window identificator.

# Description

`winclose(winIds)` close windows created by sciGUI.

# Examples

```
//CREATE SOME WINDOWS
win1=waitbar('This is an example');
win2=waitbar('HELLO!');
winclose([win1,win2]);
```

# Authors

Jaime Urzua

# Name

winlist — Return the winId of current window created by sciGUI

```
winIds=winlist()
```

# Parameters

winIds
: matrix of integer greater than 0, window identificator.

# Description

`winlist()` Return the winId of current window created by sciGUI.

# Authors

Jaime Urzua

# Parte XLI. Texmacs

# Name

pol2tex — convert polynomial to TeX format. This function is obsolete and will be removed in Scilab 5.3. Please use prettyprint

```
[y]=pol2tex(x)
```

# Parameters

x
: polynomial

y
: list

# Description

Latex source code for the polynomial `x`. (For use with `texprint`)

This function is obsolete and will be removed in Scilab 5.3. Please use prettyprint

# Examples

```
s=poly(0,'s');
p=s^3+2*s-5;
pol2tex(p)
```

# See Also

prettyprint

# Name

texprint — TeX output of Scilab object. This function is obsolete and will be removed in Scilab 5.3. Please use prettyprint

```
[text]= texprint(a)
```

# Parameters

a
: Scilab object

text
: list

# Description

returns the Tex source code of the Scilab variable `a`. `a` is a matrix (constant, polynomial, rational) or a linear system (`syslin` list).

This function is obsolete and will be removed in Scilab 5.3. Please use prettyprint

# Examples

```
s=poly(0,'s');
texprint([1/s,s^2])
```

# See Also

prettyprint , pol2str

# Parte XLII. Manipulação de Arquivos de Som

# Name

analyze — frequency plot of a sound signal

# Parameters

fmin,fmax,rate,points

scalars. default values fmin=100,fmax=1500,rate=22050,points=8192;

# Description

Make a frequency plot of the signal `w` with sampling rate `rate`. The data must be at least `points` long. The maximal frequency plotted will be `fmax`, the minimal `fmin`.

# Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);

// Then we generate the sound.
s=sin(440*t)+sin(220*t)/2+sin(880*t)/2;
[nr,nc]=size(t);
s(nc/2:nc)=sin(330*t(nc/2:nc));
analyze(s);
```

# Name

auread — load .au sound file

```
y=auread(aufile)
y=auread(aufile,ext)
[y,Fs,bits]=auread(aufile)
[y,Fs,bits]=auread(aufile,ext)
```

# Parameters

aufile
: string (The .au extension is appended if no extension is given)

Fs
: ...

[]
: integer, frequency sampling in Hz.

ext
: string (`'size'` or `'snd'`) or integer (to read n samples) or 1 x 2 integer vector [n1,n2] (to read from n1 to n2).

# Description

Utility function to read `.au` sound file. `auread(aufile)` loads a sound file specified by the string aufile, returning the sampled data in y. Amplitude values are in the range [-1,+1].

Supports multi-channel data in the following formats: 8-bit mu-law, 8-, 16-, and 32-bit linear, and floating point.

`[y,Fs,bits]=auread(aufile)` returns the sample rate (Fs) in Hertz and the number of bits per sample used to encode the data in the file.

`auread(aufile,n)` returns the first n samples from each channel.

`auread(aufile,[n1,n2])` returns samples n1 to n2.

`auread(aufile,'size')` returns the size of the audio data contained in the file in place of the actual audio data, returning the vector as [samples channels].

`auread(aufile,'snd')` returns information about the sample and data as a tlist.

# Examples

```
y=wavread('SCI/modules/sound/demos/chimes.wav');

// default is 8-bits mu-law
auwrite(y,TMPDIR+'/tmp.au');
y1=auread(TMPDIR+'/tmp.au');
maxi(abs(y-y1))
```

# See Also

savewave , analyze , mapsound

# Name

auwrite — writes .au sound file

```
auwrite(y,aufile)
auwrite(y,Fs,aufile)
auwrite(y,Fs,bits,aufile)
auwrite(y,Fs,bits,method,aufile)
```

# Parameters

y
: real vector or matrix with entries in [-1,1].

aufile
: string (The .au extension is appended if no extension is given)

Fs
: integer, frequency sampling in Hz.

bits
: integer, number of bits in the encoding.

method
: string , `'mu'` (default) or `'linear'`, encoding method.

# Description

Utility function to save `.au` sound file. `auwrite(y,aufile)` writes a sound file specified by the string aufile. The data should be arranged with one channel per column. Amplitude values outside the range [-1,+1] are ignored. Supports multi-channel data for 8-bit mu-law, and 8, 16, 32, 64 bits linear formats.

`auwrite(y,Fs,aufile)` specifies in `Fs` the sample rate of the data in Hertz.

`auwrite(y,Fs,bits,aufile)` selects the number of bits in the encoder. Allowable settings are bits in [8,16,32,64]. `auwrite(y,Fs,bits,method,aufile)` allows selection of the encoding method, which can be either 'mu' or 'linear'. Note that bits must be 8 for 'mu' choice. The default method is 8-bits mu-law enconding.

# Examples

```
A=matrix(1:6,2,3);
auwrite(A/6,22050,64,'linear',TMPDIR+'/foo.au');
B=auread(TMPDIR+'/foo.au');
maxi(abs(A- round(B*6)))
```

# See Also

auread , wavread , savewave , analyze , mapsound

# Name

beep — Produce a beep sound

```
beep();
beep('on')
beep('off')
s=beep()
```

# Description

`beep()` produces your computer's default beep sound.

`beep('on')` turns the beep on

`beep('off')` turns the beep off

`s=beep()` returns the current beep mode (on or off).

# Authors

A.C

# Name

lin2mu — linear signal to mu-law encoding

```
mu=lin2mu(y)
```

# Parameters

y
: real vector

mu
: real vector

# Description

Utility fct: converts linear signal to mu-law encoding. `mu = lin2mu(y)` converts linear audio signal amplitudes in the range -1 <= y <= 1 to mu-law in the range 0 <= mu <= 255.

# See Also

mu2lin

# Name

loadwave — load a sound wav file into scilab

```
x=loadwave(filename);
[x,y]=loadwave(filename);
```

# Parameters

filename
: a string. The path of the wav file to be loaded

x
: a matrix one line for each channel

y
: vector as [data format, number of channels, samples per second per channel, estimate of bytes per second needed, byte alignment of a basic sample block, bits per sample, length of sound data in bytes, bytes per sample (per channel)].

# Description

Reads a `.wav` sound file into Scilab as a matrix. If y is given, it is filled with information about the samples (See the message sent by `loadwave`).

# Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);

// Then we generate the sound: a two channels sound.
s=[sin(2*%pi*440*t);sin(2*%pi*350*t)];
savewave(TMPDIR+'/foo.wav',s);
s1=loadwave(TMPDIR+'/foo.wav');
max(abs(s1-s))
```

# See Also

savewave, analyze, mapsound

# Name

mapsound — Plots a sound map

```
mapsound (w,dt,fmin,fmax,simpl,rate)
```

# Parameters

dt,fmin,fmax,simpl,rate
: scalars. default values dt=0.1,fmin=100,fmax=1500,simpl=1,rate=22050;

# Description

Plots a sound map for a sound. It does FFT at time increments dt. `rate` is the sampling rate. `simpl` points are collected for speed reasons. `fmin` and `fmax` are used for graphic boundaries.

# Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);

// Then we generate the sound.
s=sin(440*t)+sin(220*t)/2+sin(880*t)/2;
[nr,nc]=size(t);
s(nc/2:nc)=sin(330*t(nc/2:nc));
mapsound(s);
```

# Name

mu2lin — mu-law encoding to linear signal

```
mu=lin2mu(y)
```

# Parameters

y
: real vector

mu
: real vector

# Description

Utility fct: `y=mu2lin(mu)` converts mu-law encoded 8-bit audio signals, stored in the range 0 <= mu <= 255, to linear signal amplitude in the range -s < y < s where s = 32124/32768 ~= .9803. The input mu is often obtained using mget(...,'uc') to read byte-encoded audio files. Translation of C program by Craig Reese: IDA/Supercomputing Research Center Joe Campbell: Department of Defense

# See Also

mu2lin

# Name

playsnd — sound player facility

```
[]=playsnd(y)
[]=playsnd(y,rate,bits [,command])
```

# Parameters

y
: A matrix. Each line descibe a channel

fs
: real number, sampling frequency (default value is 22050).

bits
: real number, number of bits (usually 8 or 16). Unused yet.

command
: Only used on Unix systems it gives the name of the command to use for playing sound (wav) files. The defaut value is `play`. If set `/dev/audio` then a 8 bits mu-law raw sound file is created and send to `/dev/audio`

# Description

Plays a multi channel signal given by a Scilab matrix were sound is sampled at rate given by `rate`.

# Examples

```
// a two channel signal
y=loadwave("SCI/modules/sound/demos/chimes.wav");
playsnd(y)
```

# See Also

lin2mu , wavread

# Name

savewave — save data into a sound wav file.

```
savewave(filename,x [, rate , nbits]);
```

# Parameters

filename
: a string. The path of the output wav file

x
: a mxn matrix where m is the number of channels and n the number of samples for each channel

rate
: a scalar giving the sampling rate (number of sample per second) 22050 is the default value.

# Description

save x into a wav sound file. you can transform other sound files into wav file with the `sox` program.

# Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);

// Then we generate the sound.
s=sin(2*%pi*440*t)+sin(2*%pi*220*t)/2+sin(2*%pi*880*t)/2;
[nr,nc]=size(t);
s(nc/2:nc)=sin(2*%pi*330*t(nc/2:nc));
savewave(TMPDIR+'/foo.wav',s);
```

# See Also

loadwave, analyze, mapsound

# Name

sound — sound player facility

```
sound(y [,fs,bits,command)
```

# Parameters

y
: real vector

fs
: real number, sampling frequency in sample per second (default value is 22050)

bits
: real number, number of bits (unused)

command
: Only used on Unix systems it gives the name of the command to use for playing sound (wav) files. The defaut value is `aplay`. If set `/dev/audio` then a 8 bits mu-law raw sound file is created and send to `/dev/audio`

# Description

`sound(y,fs)` plays the sound signal given by matrix `y` (with sample frequency fs). In fact this function is just a wrapper for `playsnd`. Values in y are assumed to be in the range -1.0 <= y <= 1.0. Values outside that range are truncated. The number of rows of `y` gives the number of channels. `sound(y)` plays the sound at the default sample rate of 22050 sample per second. `sound(y,fs,nbits)` plays the sound using `nbits` bits/sample if possible (it is in fact unused). Most platforms support bits=8 or 16.

# Examples

```
// a two channel signal
y=loadwave("SCI/modules/sound/demos/chimes.wav");
sound(y)
```

# See Also

playsnd

# Name

soundsec — generates n sampled seconds of time parameter

```
t=soundsec(n [,rate])
```

# Parameters

n
: an integer, the number of seconds to generate.

rate
: an integer, the number of samples per second.

# Description

generates a vector coding time from 0 to `n`seconds at sampled rate `rate`.

# Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);

// Then we generate the sound.
s=sin(2*%pi*440*t);
analyze(s,200,600,22050);
```

# See Also

playsnd , analyze

# Name

wavread — load .wav sound file

```
y=wavread(wavfile)
y=wavread(wavfile,ext)
[y,Fs,bits]=wavread(wavfile)
[y,Fs,bits]=wavread(wavfile,ext)
```

# Parameters

wavfile
: string (The .wav extension is appended if no extension is given)

Fs
: integer, frequency sampling in Hz (number of samples per second).

ext
: string (`'size'`) or string('info') or integer (to read n samples) or 1 x 2 integer vector [n1,n2] (to read from n1 to n2).

# Description

Utility function to read `.wav` sound file. `wavread(wavfile)` loads a sound file specified by the string wavfile, returning the sampled data in y. Amplitude values are in the range [-1,+1]. Supports multi-channel data in the following formats: 8-bit mu-law, 8-, 16-, and 32-bit linear, and floating point.

`[y,Fs,bits]=wavread(wavfile)` returns the sample rate (Fs) in Hertz and the number of bits per sample used to encode the data in the file.

`wavread(wavfile,n)` returns the first n samples from each channel.

`wavread(wavfile,[n1,n2])` returns samples n1 to n2.

`wavread(wavfile,'size')` returns the size of the audio data contained in the file in place of the actual audio data, returning the vector as [channels samples].

`wavread(wavfile,'info')` returns information about the audio data contained in the file in place of the actual audio data, returning the vector as [data format, number of channels, samples per second per channel, estimate of bytes per second needed, byte alignment of a basic sample block, bits per sample, length of sound data in bytes, bytes per sample (per channel)].

# Examples

```
wavread("SCI/modules/sound/demos/chimes.wav","size")
[y,Fs,bits]=wavread("SCI/modules/sound/demos/chimes.wav");Fs,bits
subplot(2,1,1)
plot2d(y(1,:)) // first channel
subplot(2,1,2)
plot2d(y(2,:)) // second channel
y=wavread("SCI/modules/sound/demos/chimes.wav",[1 5]) //the first five samples
```

# See Also

auread, savewave, analyze, mapsound

# Name

wavwrite — writes .wav sound file

```
wavwrite(y, wavfile)
wavwrite(y, Fs, wavfile)
wavwrite(y, Fs, nbits, wavfile)
```

# Parameters

y
: real vector or matrix with entries in [-1,1].

wavfile
: string (The .wav extension is appended if no extension is given)

Fs
: integer, frequency sampling in Hz. 22500 is the default value.

nbits
: bit-depth 8, 16, 24, 32 bits. it describes the number of bits of information recorded for each sample. 16 is the default value.

# Description

Utility function to save `.wav` sound file. `wavwrite(y,wavfile)` writes a sound file specified by the string wavfile. The data should be arranged with one channel per column. Amplitude values outside the range [-1,+1] are ignored.

`wavwrite(y,Fs,wavfile)` specifies in `Fs` the sample rate of the data in Hertz.

# Examples

```
A=matrix(1:6,2,3);
wavwrite(A/6,TMPDIR+'/foo.wav');
B=wavread(TMPDIR+'/foo.wav');
```

# See Also

auread, wavread, savewave, analyze, mapsound

# Parte XLIII. Randlib

# Name

grand — Random number generator(s)

```
Y=grand(m, n, dist_type [,p1,...,pk])
Y=grand(X, dist_type [,p1,...,pk])
Y=grand(n, dist_type [,p1,...,pk])
S=grand(action [,q1,....,ql])
```

# Parameters

m, n
: integers, size of the wanted matrix `Y`

X
: a matrix whom only the dimensions (say `m x n`) are used

dist_type
: a string given the distribution which (independants) variates are to be generated ('bin', 'nor', 'poi', etc ...)

p1, ..., pk
: the parameters (reals or integers) required to define completly the distribution `dist_type`

Y
: the resulting `m x n` random matrix

action
: a string given the action onto the base generator(s) ('setgen' to change the current base generator, 'getgen' to retrieve the current base generator name, 'getsd' to retrieve the state (seeds) of the current base generator, etc ...)

q1, ..., ql
: the parameters (generally one string) needed to define the action

S
: output of the action (generaly a string or a real column vector)

# Description

This function may be used to generate random numbers from various distributions. In this case you must apply one of the `three first forms` of the possible calling sequences to get an `m x n` matrix. The two firsts are equivalent if `X` is a `m x n` matrix, and the third form corresponds to 'multivalued' distributions (e.g. multinomial, multivariate gaussian, etc...) where a sample is a column vector (says of dim `m`) and you get then `n` such random vectors (as an `m x n` matrix).

The last form is used to undertake various manipulations onto the base generators like changing the base generator (since v 2.7 you may choose between several base generators), changing or retrieving its internal state (seeds), etc ... These base generators give random integers following a uniform distribution on a large integer interval (lgi), all the others distributions being gotten from it (in general via a scheme lgi -> U([0,1)) -> wanted distribution).

# Getting random numbers from a given distribution

beta
: `Y=grand(m,n,'bet',A,B)` generates random variates from the beta distribution with parameters `A` and `B`. The density of the beta is (`0 < x < 1`) :

```
 A-1    B-1
x   (1-x)   / beta(A,B)
```

A and B must be reals >10^(-37). Related function(s) : cdfbet.

binomial
: Y=grand(m,n,'bin',N,p) generates random variates from the binomial distribution with parameters N (positive integer) and p (real in [0,1]) : number of successes in N independant Bernouilli trials with probability p of success. Related function(s) : binomial, cdfbin.

negative binomial
: Y=grand(m,n,'nbn',N,p) generates random variates from the negative binomial distribution with parameters N (positive integer) and p (real in (0,1)) : number of failures occurring before N successes in independant Bernouilli trials with probability p of success. Related function(s) : cdfnbn.

chisquare
: Y=grand(m,n,'chi', Df) generates random variates from the chisquare distribution with Df (real > 0.0) degrees of freedom. Related function(s) : cdfchi.

non central chisquare
: Y=grand(m,n,'nch',Df,Xnon) generates random variates from the non central chisquare distribution with Df degrees of freedom (real >= 1.0) and noncentrality parameter Xnonc (real >= 0.0). Related function(s) : cdfchn.

exponential
: Y=grand(m,n,'exp',Av) generates random variates from the exponential distribution with mean Av (real >= 0.0).

F variance ratio
: Y=grand(m,n,'f',Dfn,Dfd) generates random variates from the F (variance ratio) distribution with Dfn (real > 0.0) degrees of freedom in the numerator and Dfd (real > 0.0) degrees of freedom in the denominator. Related function(s) : cdff.

non central F variance ratio
: Y=grand(m,n,'nf',Dfn,Dfd,Xnon) generates random variates from the noncentral F (variance ratio) distribution with Dfn (real >= 1) degrees of freedom in the numerator, and Dfd (real > 0) degrees of freedom in the denominator, and noncentrality parameter Xnonc (real >= 0). Related function(s) : cdffnc.

gamma
: Y=grand(m,n,'gam',shape,scale) generates random variates from the gamma distribution with parameters shape (real > 0) and scale (real > 0). The density of the gamma is :

```
     shape  (shape-1)   -scale x
scale       x          e          /  gamma(shape)
```

Related function(s) : gamma, cdfgam.

Gauss Laplace (normal)
: Y=grand(m,n,'nor',Av,Sd) generates random variates from the normal distribution with mean Av (real) and standard deviation Sd (real >= 0). Related function(s) : cdfnor.

multivariate gaussian (multivariate normal)
: Y=grand(n,'mn',Mean,Cov) generates n multivariate normal random variates ; Mean must be a m x 1 matrix and Cov a m x m symetric positive definite matrix (Y is then a m x n matrix).

geometric
: `Y=grand(m,n,'geom', p)` generates random variates from the geometric distribution with parameter `p` : number of Bernouilli trials (with probability succes of `p`) until a succes is met. `p` must be in `[pmin,1]` (with `pmin = 1.3 10^(-307)`).

  `Y` contains positive real numbers with integer values, with are the "number of trials to get a success".

markov
: `Y=grand(n,'markov',P,x0)` generate `n` successive states of a Markov chain described by the transition matrix `P`. Initial state is given by `x0`. If `x0` is a matrix of size `m=size(x0,'*')` then `Y` is a matrix of size `m x n`. `Y(i,:)` is the sample path obtained from initial state `x0(i)`.

multinomial
: `Y=grand(n,'mul',nb,P)` generates `n` observations from the Multinomial distribution : class `nb` events in `m` categories (put `nb` "balls" in `m` "boxes"). `P(i)` is the probability that an event will be classified into category i. `P` the vector of probabilities is of size `m-1` (the probability of category `m` being `1-sum(P)`). `Y` is of size `m x n`, each column `Y(:,j)` being an observation from multinomial distribution and `Y(i,j)` the number of events falling in category `i` (for the `j` th observation) (`sum(Y(:,j)) = nb`).

Poisson
: `Y=grand(m,n,'poi',mu)` generates random variates from the Poisson distribution with mean `mu (real >= 0.0)`. Related function(s) : cdfpoi.

random permutations
: `Y=grand(n,'prm',vect)` generate `n` random permutations of the column vector (`m x 1`) `vect`.

uniform (def)
: `Y=grand(m,n,'def')` generates random variates from the uniform distribution over `[0,1)` (1 is never return).

uniform (unf)
: `Y=grand(m,n,'unf',Low,High)` generates random reals uniformly distributed in `[Low, High)`.

uniform (uin)
: `Y=grand(m,n,'uin',Low,High)` generates random integers uniformly distributed between `Low` and `High` (included). `High` and `Low` must be integers such that `(High-Low+1) < 2,147,483,561`.

uniform (lgi)
: `Y=grand(m,n,'lgi')` returns the basic output of the current generator : random integers following a uniform distribution over :

  - `[0, 2^32 - 1]` for mt, kiss and fsultra
  - `[0, 2147483561]` for clcg2
  - `[0, 2^31 - 2]` for clcg4
  - `[0, 2^31 - 1]` for urand.

# Set/get the current generator and its state

Since Scilab-2.7 you have the possibility to choose between different base generators (which give random integers following the 'lgi' distribution, the others being gotten from it) :

mt
: the Mersenne-Twister of M. Matsumoto and T. Nishimura, period about `2^19937`, state given by an array of `624` integers (plus an index onto this array); this is the default generator.

kiss
: The Keep It Simple Stupid of G. Marsaglia, period about `2^123`, state given by `4` integers.

clcg2
: a Combined 2 Linear Congruential Generator of P. L'Ecuyer, period about `2^61`, state given by `2` integers ; this was the only generator previously used by grand (but slightly modified)

clcg4
: a Combined 4 Linear Congruential Generator of P. L'Ecuyer, period about `2^121`, state given by 4 integers ; this one is splitted in `101` different virtual (non over-lapping) generators which may be useful for different tasks (see 'Actions specific to clcg4' and 'Test example for clcg4').

urand
: the generator used by the scilab function rand, state given by `1` integer, period of `2^31` (based on theory and suggestions given in d.e. knuth (1969), vol 2. State). This is the faster of this list but a little outdated (don't use it for serious simulations).

fsultra
: a Subtract-with-Borrow generator mixing with a congruential generator of Arif Zaman and George Marsaglia, period more than `10^356`, state given by an array of 37 integers (plus an index onto this array, a flag (0 or 1) and another integer).

The differents actions common to all the generators, are:

action= 'getgen'
: `S=grand('getgen')` returns the current base generator ( `S` is a string among 'mt', 'kiss', 'clcg2', 'clcg4', 'urand', 'fsultra'.

action= 'setgen'
: `grand('setgen',gen)` sets the current base generator to be `gen` a string among 'mt', 'kiss', 'clcg2', 'clcg4', 'urand', 'fsultra' (notes that this call returns the new current generator, ie gen).

action= 'getsd'
: `S=grand('getsd')` gets the current state (the current seeds) of the current base generator ; `S` is given as a column vector (of integers) of dimension `625` for mt (the first being an index in `[1,624]`), `4` for kiss, `2` for clcg2, `40` for fsultra, `4` for clcg4 (for this last one you get the current state of the current virtual generator) and `1` for urand.

action= 'setsd'
: `grand('setsd',S), grand('setsd',s1[,s2,s3,s4])` sets the state of the current base generator (the new seeds) :

for mt
: `S` is a vector of integers of dim `625` (the first component is an index and must be in `[1,624]`, the `624` last ones must be in `[0,2^32[`) (but must not be all zeros) ; a simpler initialisation may be done with only one integer `s1` (`s1` must be in `[0,2^32[`) ;

for kiss
: `4` integers `s1,s2, s3,s4` in `[0,2^32[` must be provided ;

for clcg2
: `2` integers `s1` in `[1,2147483562]` and `s2` in `[1,2147483398]` must be given ;

for clcg4
: `4` integers `s1` in `[1,2147483646]`, `s2` in `[1,2147483542]`, `s3` in `[1,2147483422]`, `s4` in `[1,2147483322]` are required ; `CAUTION` : with clcg4 you set the seeds of the current virtual generator but you may lost the synchronisation between

this one and the others virtuals generators (ie the sequence generated is not warranty to be non over-lapping with a sequence generated by another virtual generator)=> use instead the 'setall' option.

for urand
`1` integer `s1` in `[0,2^31[` must be given.

for fsultra
`S` is a vector of integers of dim `40` (the first component is an index and must be in `[0,37]`, the 2d component is a flag (0 or 1), the 3d an integer in [1,2^32[ and the 37 others integers in [0,2^32[) ; a simpler (and recommanded) initialisation may be done with two integers `s1` and `s2` in `[0,2^32[`.

action= 'phr2sd'
`Sd=grand('phr2sd', phrase)` given a `phrase` (character string) generates a `1 x 2` vector `Sd` which may be used as seeds to change the state of a base generator (initialy suited for clcg2).

# Options specific to clcg4

The clcg4 generator may be used as the others generators but it offers the advantage to be splitted in several (`101`) virtual generators with non over-lapping sequences (when you use a classic generator you may change the initial state (seeds) in order to get another sequence but you are not warranty to get a complete different one). Each virtual generator corresponds to a sequence of `2^72` values which is further split into `V=2^31` segments (or blocks) of length `W=2^41`. For a given virtual generator you have the possibility to return at the beginning of the sequence or at the beginning of the current segment or to go directly at the next segment. You may also change the initial state (seed) of the generator `0` with the 'setall' option which then change also the initial state of the other virtual generators so as to get synchronisation (ie in function of the new initial state of gen `0` the initial state of gen `1..100` are recomputed so as to get `101` non over-lapping sequences.

action= 'setcgn'
`grand('setcgn',G)` sets the current virtual generator for clcg4 (when clcg4 is set, this is the virtual (clcg4) generator number `G` which is used); the virtual clcg4 generators are numbered from `0,1,..,100` (and so `G` must be an integer in `[0,100]`) ; by default the current virtual generator is `0`.

action= 'getcgn'
`S=grand('getcgn')` returns the number of the current virtual clcg4 generator.

action= 'initgn'
`grand('initgn',I)` reinitializes the state of the current virtual generator

I = -1
sets the state to its initial seed

I = 0
sets the state to its last (previous) seed (i.e. to the beginning of the current segment)

I = 1
sets the state to a new seed `W` values from its last seed (i.e. to the beginning of the next segment) and resets the current segment parameters.

action= 'setall'
`grand('setall',s1,s2,s3,s4)` sets the initial state of generator `0` to `s1,s2,s3,s4`. The initial seeds of the other generators are set accordingly to have synchronisation. For constraints on `s1, s2, s3, s4` see the 'setsd' action.

action= 'advnst'
`grand('advnst',K)` advances the state of the current generator by `2^K` values and resets the initial seed to that value.

# Test example for clcg4

An example of the need of the splitting capabilities of clcg4 is as follows. Two statistical techniques are being compared on data of different sizes. The first technique uses bootstrapping and is thought to be as accurate using less data than the second method which employs only brute force. For the first method, a data set of size uniformly distributed between 25 and 50 will be generated. Then the data set of the specified size will be generated and analyzed. The second method will choose a data set size between 100 and 200, generate the data and analyze it. This process will be repeated 1000 times. For variance reduction, we want the random numbers used in the two methods to be the same for each of the 1000 comparisons. But method two will use more random numbers than method one and without this package, synchronization might be difficult. With clcg4, it is a snap. Use generator 0 to obtain the sample size for method one and generator 1 to obtain the data. Then reset the state to the beginning of the current block and do the same for the second method. This assures that the initial data for method two is that used by method one. When both have concluded, advance the block for both generators.

# See Also

rand

# Authors

randlib
: The codes to generate sequences following other distributions than def, unf, lgi, uin and geom are from "Library of Fortran Routines for Random Number Generation", by Barry W. Brown and James Lovato, Department of Biomathematics, The University of Texas, Houston.

mt
: The code is the mt19937int.c by M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January, pp.3-30 1998.

kiss
: The code was given by G. Marsaglia at the end of a thread concerning RNG in C in several newsgroups (whom sci.math.num-analysis) "My offer of RNG's for C was an invitation to dance..." only kiss have been included in Scilab (kiss is made of a combinaison of severals others which are not visible at the scilab level).

clcg2
: The method is from P. L'Ecuyer but the C code is provided at the Luc Devroye home page (http://cgm.cs.mcgill.ca/~luc/rng.html).

clcg4
: The code is from P. L'Ecuyer and Terry H.Andres and provided at the P. L'Ecuyer home page ( http://www.iro.umontreal.ca/~lecuyer/papers.html) A paper is also provided and this new package is the logical successor of an old 's one from : P. L'Ecuyer and S. Cote. Implementing a Random Number Package with Splitting Facilities. ACM Transactions on Mathematical Software 17:1,pp 98-111.

fsultra
: code from Arif Zaman (arif@stat.fsu.edu) and George Marsaglia (geo@stat.fsu.edu)

scilab packaging
: By Jean-Philippe Chancelier and Bruno Pincon

# Parte XLIV. Ferramentas de Desenvolvimento

# Name

bench_run — Launch benchmark tests

```
bench_run()
bench_run(module[,test_name[,options]])
```

# Parameters

module
: a vector of string. It can be the name of a module or the absolute path of a toolbox.

test_name
: a vector of string

options
: a vector of string

- list : list of the benchmark tests available in a module
- help : displays some examples of use in the Scilab console
- nb_run=value : repeat the benchmark test value times

# Description

Search for .tst files in benchmark test library execute them, and display a report about execution time. The .tst files are searched in directories SCI+"/modules/*/tests/benchmark".

Special tags may be inserted in the .tst file, which help to control the processing of the corresponding test. These tags are expected to be found in Scilab comments.

These are the available tags :

- <-- BENCH NB RUN : 10 --> This test will be repeated 10 times.
- <-- BENCH START --> <-- BENCH END --> The interesting part of the benchmark must be enclosed by these tags.

# Examples

Some simple examples of invocation of bench_run

```
// Launch all tests
bench_run();
bench_run([]);
bench_run([],[]);

// Test one or several module
bench_run('core');
bench_run('core',[]);
bench_run(['core','string']);

// Launch one or several test in a specified module
bench_run('core',['trycatch','opcode']);
```

```
// With options
bench_run([],[],'list');
bench_run([],[],'help');
bench_run([],[],'nb_run=2000');
```

An example of a benchmark file. This file corresponds to the file SCI/modules/linear_algebra/tests/benchmarks/bench_chol.tst.

```
// ============================================================================
// Scilab ( http://www.scilab.org/ ) - This file is part of Scilab
// Copyright (C) 2007-2008 - INRIA
//
//  This file is distributed under the same license as the Scilab package.
// ============================================================================

//============================================================================
// Benchmark for chol function
//============================================================================

// <-- BENCH NB RUN : 10 -->

stacksize(30000000);

a = 0;
b = 0;
a = rand(900, 900, 'n');
a = a'*a;

// <-- BENCH START -->
b = chol(a);
// <-- BENCH END -->
```

The result of the test

```
-->bench_run('linear_algebra','bench_chol')

             Boucle For ....................................       0.19 µs [

   001/001 - [linear_algebra] bench_chol ......................  151576.71 µs [
```

# See Also

test_run

# Authors

Yann Collette

# Name

tbx_build_cleaner — Generate a cleaner.sce script (toolbox compilation process)

```
tbx_build_loader(toolbox_name)
tbx_build_loader(toolbox_name, toolbox_path)
```

# Parameters

toolbox_name
: Toolbox short name ; that is, the prefix of the .start file of the toolbox (which shall be in the etc subdirectory).

toolbox_path
: Root directory of toolbox sources ; the script will be generated here (default: current directory).

# Examples

```
// Recommended usage
tbx_build_cleaner("mytoolbox", get_absolute_file_path('builder.sce'))
```

# Authors

Allan CORNET

# Name

tbx_build_gateway — Build a gateway (toolbox compilation process)

```
tbx_build_gateway(libname, names, files, [gateway_path [, libs [, ldflags
```

# Parameters

libname
: a character string, the generic name of the library without path and extension.

names
: 2 column string matrix giving the table of pairs 'scilab-name', 'interface name'

files
: string matrix giving objects files needed for shared library creation

gateway_path
: Path to the sources of the gateway ; in a normal toolbox it should be the directory containing the builder_gateway_(lang).sce script (which should be the script calling this function). Default is current directory.

libs
: string matrix giving extra libraries needed for shared library creation

ldflags,cflags,fflags
: character strings to provide options for the loader, the C compiler and the Fortran compiler.

cc
: character string. The name of or path to the compiler.

makename
: character string. The path of the Makefile file without extension.

ismex
: Internal variable to specify if we are working with mex or not.

# Examples

```
// Recommended usage
tbx_build_gateway('mytoolbox', ['c_sum','sci_csum';'c_sub','sci_csub'], ['sci_c
                  get_absolute_file_path('builder_gateway_c.sce'), ..
                  ['../../src/c/libcsum']);
```

# See Also

ilib_build

# Authors

SL

# Name

tbx_build_gateway_clean — Generate a cleaner_gateway.sce script (toolbox compilation process)

```
tbx_build_gateway_loader(langs)
tbx_build_gateway_loader(langs, gateway_path)
```

# Parameters

langs
: Languages of source files.

gateway_path
: Path to the sources of the gateway ; in a normal toolbox it should be the directory containing the builder_gateway.sce script (which should be the script calling this function). Default is current directory.

# Examples

```
// Recommended usage
tbx_build_gateway_clean(['c', 'fortran'], get_absolute_file_path('builder_gatew
```

# Authors

Allan CORNET

# Name

tbx_build_gateway_loader — Generate a loader_gateway.sce script (toolbox compilation process)

```
tbx_build_gateway_loader(langs)
tbx_build_gateway_loader(langs, gateway_path)
```

# Parameters

langs
: Languages of source files.

gateway_path
: Path to the sources of the gateway ; in a normal toolbox it should be the directory containing the builder_gateway.sce script (which should be the script calling this function). Default is current directory.

# Examples

```
// Recommended usage
tbx_build_gateway_loader(['c', 'fortran'], get_absolute_file_path('builder_gate
```

# Authors

SL

# Name

tbx_build_help — Generate help files (toolbox compilation process)

```
tbx_build_help(title)
tbx_build_help(title, help_path)
```

# Parameters

title
: Title of the chapter.

help_path
: Directory where the files will be generated ; in a normal toolbox it should be the directory containing the build_help.sce script (which should be the script calling this function). Default is current directory.

# Examples

```
// Recommended usage
tbx_build_help("Toolbox Example", get_absolute_file_path('build_help.sce'))
```

# Authors

SL

# Name

tbx_build_help_loader — Generate a addchapter.sce script (toolbox compilation process)

```
tbx_build_help_loader(title)
tbx_build_help_loader(title, help_path)
```

# Parameters

title
: Title of the chapter.

help_path
: Directory where the script will be generated ; in a normal toolbox it should be the directory containing the build_help.sce script (which should be the script calling this function). Default is current directory.

# Examples

```
// Recommended usage
tbx_build_help_loader("Toolbox Example", get_absolute_file_path('build_help.sce
```

# Authors

SL

# Name

tbx_build_loader — Generate a loader.sce script (toolbox compilation process)

```
tbx_build_loader(toolbox_name)
tbx_build_loader(toolbox_name, toolbox_path)
```

# Parameters

toolbox_name
: Toolbox short name ; that is, the prefix of the .start file of the toolbox (which shall be in the etc subdirectory).

toolbox_path
: Root directory of toolbox sources ; the script will be generated here (default: current directory).

# Examples

```
// Recommended usage
tbx_build_loader("mytoolbox", get_absolute_file_path('builder.sce'))
```

# Authors

SL

# Name

tbx_build_macros — Compile macros (toolbox compilation process)

```
tbx_build_macros(toolbox_name)
tbx_build_macros(toolbox_name, macros_path)
```

# Parameters

toolbox_name
: Toolbox short name ; that is, the prefix of the .start file of the toolbox (which shall be in the etc subdirectory).

macros_path
: Directory where the macros files can be found and where the compiled macros will be placed into ; in a normal toolbox it should be the directory containing the buildmacros.sce script (which should be the script calling this function). Default is current directory.

# Examples

```
// Recommended usage
tbx_build_macros("toolbox_example", get_absolute_file_path('buildmacros.sce'))
```

# Authors

SL

# Name

tbx_build_src — Build sources (toolbox compilation process)

```
tbx_build_src(names, files, flag, [src_path [, libs [, ldflags [, cflags
```

# Parameters

names
: a string matrix giving the entry names which are to be linked.

files
: string matrix giving objects files needed for shared library creation

flag
: a string flag ("c" or "f") for C or Fortran entry points.

src_path
: Path to the source files ; in a normal toolbox it should be the directory containing the builder_src_(lang).sce script (which should be the script calling this function). Default is current directory.

libs
: string matrix giving extra libraries needed for shared library creation

ldflags
: optional character string. It can be used to add specific linker options in the generated Makefile. Default value is "

cflags
: optional character string. It can be used to add specific C compiler options in the generated Makefile. Default value is "

fflags
: optional character string. It can be used to add specific Fortran compiler options in the generated Makefile. Default value is "

cc
: optional character string. It can be used to specify a C compiler. Default value is "

libname
: optional character string. The name of the generated shared library (default value is ", and in this case the name is derived from `names(1)`).

loadername
: character string. The pathname of the loader file (default value is `loader.sce`).

makename
: character string. The pathname of the Makefile file without extension (default value `Makelib`).

# Examples

```
// Recommended usage
tbx_build_src(['csum','csub'], ['csum.c','csub.c'], 'c', ..
              get_absolute_file_path('builder_c.sce'));
```

# See Also

ilib_for_link

# Authors

SL

# Name

tbx_builder_gateway — Run builder_gateway.sce script if it exists (toolbox compilation process)

```
tbx_builder_gateway(toolbox_path)
```

# Parameters

toolbox_path
: Root directory of toolbox sources ; builder_gateway.sce script will be searched in the sci_gateway subdirectory of this directory.

# Examples

```
// Recommended usage
tbx_builder_gateway(get_absolute_file_path('builder.sce'))
```

# Authors

SL

# Name

tbx_builder_gateway_lang — Run builder_gateway_(language).sce script if it exists (toolbox compilation process)

```
tbx_builder_gateway_lang(lang)
tbx_builder_gateway_lang(lang, gw_path)
```

# Parameters

lang
: Language of sources files ; the builder_gateway_(lang).sce script will be searched in the subdirectory lang (e.g. fortran) of the gw_path directory.

gw_path
: Path to the sources of the gateway ; in a normal toolbox it should be the directory containing the builder_gateway.sce script (which should be the script calling this function). Default is current directory.

# Examples

```
// Recommended usage
tbx_builder_gateway_lang("fortran", get_absolute_file_path('builder_gateway.sce
```

# Authors

SL

# Name

tbx_builder_help — Run builder_help.sce script if it exists (toolbox compilation process)

```
tbx_builder_help(toolbox_path)
```

# Parameters

toolbox_path
: Root directory of toolbox sources ; builder_help.sce script will be searched in the help subdirectory of this directory.

# Examples

```
// Recommended usage
tbx_builder_help(get_absolute_file_path('builder.sce'))
```

# Authors

SL

# Name

tbx_builder_help_lang — Run build_help.sce script if it exists (toolbox compilation process)

```
tbx_builder_help_lang(lang)
tbx_builder_help_lang(lang, help_path)
```

# Parameters

lang
: Language of help files to use ; the build_help.sce script will be searched in the subdirectory lang (e.g. en_US) of the help_path directory

help_path
: Path to help directory ; in a normal toolbox it should be the directory containing the builder_help.sce script (which should be the script calling this function). Default is current directory.

# Examples

```
// Recommended usage
tbx_builder_help_lang("en_US", get_absolute_file_path('builder_help.sce'))
```

# Authors

SL

# Name

tbx_builder_macros — Run buildmacros.sce script if it exists (toolbox compilation process)

```
tbx_builder_macros(toolbox_path)
```

# Parameters

toolbox_path
: Root directory of toolbox sources ; buildmacros.sce script will be searched in the macros subdirectory of this directory.

# Examples

```
// Recommended usage
tbx_builder_macros(get_absolute_file_path('builder.sce'))
```

# Authors

SL

# Name

tbx_builder_src — Run builder_src.sce script if it exists (toolbox compilation process)

```
tbx_builder_src(toolbox_path)
```

# Parameters

toolbox_path
: Root directory of toolbox sources ; builder_src.sce script will be searched in the src subdirectory of this directory.

# Examples

```
// Recommended usage
tbx_builder_src(get_absolute_file_path('builder.sce'))
```

# Authors

SL

# Name

tbx_builder_src_lang — Run builder_(language).sce script if it exists (toolbox compilation process)

```
tbx_builder_src_lang(lang)
tbx_builder_src_lang(lang, src_path)
```

# Parameters

lang
: Language of sources files ; the builder_(lang).sce script will be searched in the subdirectory lang (e.g. fortran) of the src_path directory.

src_path
: Path to the sources ; in a normal toolbox it should be the directory containing the builder_src.sce script (which should be the script calling this function). Default is current directory.

# Examples

```
// Recommended usage
tbx_builder_src_lang("fortran", get_absolute_file_path('builder_src.sce'))
```

# Authors

SL

# Name

test_run — Launch tests

```
N = test_run()
N = test_run(module[,test_name[,options]])
```

# Parameters

module
: a vector of string. It can be the name of a module or the absolute path of a toolbox.

test_name
: a vector of string

options
: a vector of string

- no_check_ref : does not check if the .dia and .dia.ref are equal
- no_check_error_output
- create_ref : create the .dia.ref file and does not check if the .dia and .dia.ref are equal
- list : does not perform the tests but displays a list of available tests
- help : display some examples about how to use this command
- mode_nw : add the "-nw" option to the launch
- mode_nwni : add the "-nwni" option to the launch
- nonreg_test : runs only the non-regression tests, skipping unit tests
- unit_test : runs only the unit tests, skipping non-regression tests
- skip_tests

# Description

Search for .tst files in the unit test and non-regression test library execute them, and display a report about success of failures. The .tst files are searched in directories SCI+"/modules/*/tests/unit_tests" and SCI+"/modules/*/tests/nonreg_tests". Whenever a test is executed, a .dia file is generated which contains the full list of commands executed along with message which appears in the console. When the script is done, the .dia file is compared with the .dia.ref file which is expected to be in the same directory as the .tst file. If the two file are different, the test fails.

Special tags may be inserted in the .tst file, which help to control the processing of the corresponding test. These tags are expected to be found in Scilab comments.

These are the available tags :

- <-- INTERACTIVE TEST --> This test will be skipped because it is interactive.
- <-- NOT FIXED --> This test will be skipped because it is a known, but unfixed bug.
- <-- TEST WITH GRAPHIC --> This test will not be executed if the option "mode_nwni" is used.
- <-- NO TRY CATCH -->

- <-- NO CHECK ERROR OUTPUT --> The error output file is not checked
- <-- NO CHECK REF --> The .dia and the .dia.ref files are not compared.
- <-- ENGLISH IMPOSED --> This test will be executed with the -l en_US option.
- <-- FRENCH IMPOSED --> This test will be executed with the -l fr_FR option.
- <-- JVM NOT MANDATORY --> This test will be executed with the nwni mode by default.
- <-- WINDOWS ONLY --> If the operating system isn't Windows, the test is skipped.
- <-- UNIX ONLY --> If the operating system isn't an unix OS, the test is skipped.
- <-- LINUX ONLY --> If the operating system isn't Linux, the test is skipped.
- <-- MACOSX ONLY --> If the operating system isn't MacOSX, the test is skipped.

Each test is executed in a separated process, created with the "host" command. That enables the current command to continue, even if the test as created an unstable environment. It also enables the tests to be independent from one another.

# Platform-specific tests

It may happen that the output of a test depends on the platform on which it is executed. In this case, the `.ref` file cannot be correct for all platforms and unit tests may fail for some platform. In this case, we can create a default `.ref` and create additionnal `.ref` file for each platform.

The various platform-specific `.ref` files must have one of the following extensions.

- `.unix.dia.ref` for Unix platform,
- `.linux..diaref` for Linux platform,
- `.win.dia.ref` for Windows platform,
- `.macosx.dia.ref` for Mac OS X platform.

The algorithm is the following. First, the `.ref` is considered. If this file does not exist, the platform-specific `.ref` file is examined depending on the current platform.

- on windows platforms: `.win.dia.ref`,
- on Max OS X platforms: `.unix.dia.ref`, `.macosx.dia.ref`,
- on Linux platforms: `.unix.dia.ref`, `.linux.dia.ref`.

# Examples

```
// Launch all tests
// ==============================================

test_run();
test_run([]);
test_run([],[]);

// Test one or several module
// ==============================================

// Test one module
```

```
test_run('time');

// Test several modules
test_run(['time','string']);

// Test a submodule
test_run(['optimization|neldermead']);

// Refer to a module by its path
test_run(SCI+'/modules/core');

// Launch a specific test
// ==============================================

// One specific test
test_run('time','datenum');

// Several tests
test_run('time',['datenum';'calendar']);

// Skip some tests
// ==============================================

test_run('time',['datenum';'calendar'],'skip_tests');

// Options
// ==============================================

// does not check if the .dia and .dia.ref are equal
test_run('time','datenum','no_check_ref');

// Create the .dia.ref file and does not check if the .dia and .dia.ref are equa
test_run([],[],'create_ref');

// Does not perform the tests but displays a list of available tests
test_run([],[],'list');

// Display some examples about how to use this command
test_run([],[],'help');

// Runs only the non-regression tests, skipping unit tests
test_run([],[],'nonreg_test');

// Runs only the unit tests, skipping non-regression tests
test_run([],[],'unit_test');

// Do not check the error output (std err)
test_run('boolean','bug_2799','no_check_error_output');

// Combine several options
test_run([],[],['no_check_ref','mode_nw']);
```

# Internal Design

The tests are performed in the temporary directory, not in the directory which originaly contain the tests files. The .tst file is copied into the temporary directory, the test is performed and the .dia.ref is copied back into the original location.

The .tst script is not run as is. Instead, a header and a footer are inserted at the beginning and at the end of the .tst at the time the script is copied into the temporary directory. The role of this modification is to redirect the output messages into the .dia file, so that the user can have a log file once the test is performed.

# Authors

Pierre Maréchal
Michael Baudin

# Parte XLV. Ferramentas de Demonstração

# Nom

demo_begin — begin a demonstration

```
demo_begin()
```

# Description

The function demo_begin is used to begin a demonstration. It sets the script and the values in mode of non display on the console, save the environment variables in a temporary file and save the width of the console. This function shall be used with the function demo_end.

# Voir Aussi

demo_end , demo_run , demo_message

# Auteurs

G.H

# Name

demo_choose — create a dialog box for the choice of options

```
num = demo_choose(fil)
```

# Description

The function demo_choose create a dialog box for the choice of options. It takes as argument the binary file 'fil'. This file is built by a .sce file written like below. It shall contain the variables 'choice', an array of text within bracket (the different options), and 'titl', the title of the dialog box. After that, the .sce file shall save both variables in the binary file in argument. Before the use of demo_choose, the .sce file shall be executed. The function demo_choose returns the number of line chosen in the options array.

# Examples

```
exec('SCI/demos/control/pid/pid_ch_2.sce');
[n]=demo_choose('SCI/demos/control/pid/pid_ch_2.bin');
select n
  case 0
    break
  case 1
    mode(1)
  case 2
    mode(-1)
end
```

# See Also

x_choose , demo_mdialog

# Authors

G.H

# Name

demo_compiler — test the presence of a compileur

```
status = demo_compiler()
```

# Description

The function demo_compiler asks the computer if it owns a compileur C or not. It returns a boolean indicating wether the compiler exists or not.

# Examples

```
select num,
 case 0
  return;
 case 2 then
  st = demo_compiler(); //The compiler will be used
  if (st==%t) then
    mode(0);
    wheel_build_and_load()
  end
 case 1 then // A precomputed value
  x=read(path+'/x.wrt',8,301);
  wheelg=wheelgs;
  show(x);
end
```

# See Also

findmsvccompiler

# Authors

G.H

# Nom

demo_end — completes a demonstration

```
demo_end()
```

# Description

The function demo_end() is used to complete a demonstration. It shall be used complementarily with the function demo_begin. It resets the state of the environment as it was before to use the function demo_begin : width of the console and the variables value.

# Voir Aussi

demo_begin , demo_run , demo_message

# Auteurs

G.H

# Name

demo_file_choice — choose and executes an item within a list

```
demo_file_choice(path,ch)
```

# Description

The function demo_file_choice choose and executes an item chosen in the 'demolist' variable, that shall be written above. The variable 'demolist' is a text matrix whose first column contains names of items displayed in an options window and whose second column contains the names of the files that will be executed. The title of the options window is 'Choose a demo'. The 'path' variable is the access to the files called in the second column. The 'ch' variable allows to avoid the special cases 'root' and 'anim' that are used in peculiar demonstrations of Scilab. Then you have to enter another word than 'root' or 'ch', 'no' for example. If you choose to cancel the options window, the console is put back to its previous width.

# Examples

```
demolist=['n-pendulum','npend/npend_gateway.sce';
   'Wheel simulation','wheel2/wheel_gateway.sce';
   'Bike Simulation','bike/bike_gateway.sce';
   'ODE''S','ode/ode.dem';
   'DAE''S','dae/dae.dem']

demo_file_choice('SCI/demos/simulation/','no');
```

# See Also

demo_function_choice

# Authors

G.H

# Name

demo_function_choice — choose and execute an item within a list

```
demo_function_choice()
```

# Description

The function demo_function_choice choose and execute an item chosen in the variable 'demolist' that shall appear above. The variable 'demolist' is a text matrix whose first column contains item names displayed in an options window and whose second column contains the function that will be called. The title of the options window is 'Choose a demo'. If the options window is cancelled, the console is put back to its previous width.

# Examples

```
demolist=[
 'Simulation of a binomial random variable','set figure_style new;clf();Binomia
 'Simulation of a discrete random variable','set figure_style new;clf();RndDisc
 'Simulation of a geometric random variable','set figure_style new;clf();GeomT(
 'Simulation of a Poisson random variable','set figure_style new;clf();PoissonT
 'Simulation of an exponential random variable','set figure_style new;clf();Exp
 'Simulation of a Weibull random variable','set figure_style new;clf();WeibullT
 'Simulation of an hyper geometric random variable','set figure_style new;clf()
 'Simulation of an Erlang random variable','set figure_style new;clf();ErlangT(

demo_function_choice();
```

# See Also

demo_file_choice

# Authors

G.H

# Name

demo_mdialog — create a dialog box

```
resp = demo_mdialog(fil)
```

# Description

The function demo_mdialog create a dialog box. It takes as argument a binary file. This file is built by a .sce file written like below. It shall contain the variables 'titl', the title a the dialog box, 'namevar', the name of the fields to fill, and 'value', the values written by default. After this, these three variables shall be saved in the binary file. The use of demo_mdialog shall be preceded by the execution of the .sce associated. The function demo_mdialog returns 'resp', the values chosen by the user.

# Examples

```
exec('SCI/demos/control/pid/pid_dial_4.sce');
[defv]=demo_mdialog('SCI/demos/control/pid/pid_dial_4.bin');

if defv==[] then warning('Demo stops!');return;end
```

# See Also

demo_choose , x_mdialog , x_dialog

# Authors

G.H

# Name

demo_message — display a message

```
demo_message(fil)
```

# Description

The function demo_message displays the text message within the file 'fil' given in argument.

# Examples

```
demo_message('SCI/demos/control/pid/pid_3.sce');
```

# See Also

demo_run , messagebox , demo_begin , demo_end

# Authors

G.H

# Name

demo_run — script file execution

```
demo_run(fil)
```

# Description

The function demo_run executes a script in the file 'fil' given in argument.

# See Also

exec , demo_message , demo_begin , demo_end

# Authors

G.H

# Parte XLVI. Planilhas

# Name

readxls — lê um arquivo Excel

```
sheets = readxls(file_path)
```

# Parâmetros

file_path
: string: o endereço do arquivo Excel

sheets
: um `mlist` de tipo `xls`, com um campo chamado `sheets`

# Descrição

Dado um endereço de um arquivo Excel, esta função retorna uma estrutura de dados `mlist` do tipo `xls`, com um campo chamado `sheets`. O campo `sheets` contém uma lista da estutura de dados folha.

`sheet=mlist(['xlssheet','name','text','value'],sheetname,Text,Value)` onde `sheetname` é um string contendo o nome da folha, `Text` é uma matriz de strings que contém os strings e todas as células `Value` é uma matriz de números que contém todos os valores das células.

AVISO: apenas arquivos Excel BIFF8 (última versão do Excel) são manipulados

# Exemplos

```
Sheets = readxls('SCI/modules/spreadsheet/demos/xls/t1.xls')
// algumas operações básicas sobre Sheets
typeof(Sheets)
s1=Sheets(1) //obtendo a primeira folha
typeof(s1)
s1.value //obtendo o campo do valor da primeira folha
s1.text  //obtendo o campo do texto da primeira folha
s1(2,:)  //obtendo a segunda linha da folha
typeof(s1(2,:))

editvar s1
```

# Ver Também

xls_open, xls_read

# Autores

Pierrick Mode
: INRIA

Serge Steer
: INRIA

# Funções Utilizadas

Esta função é baseada nas funções Scilab `xls_open` e `xls_read`

# Name

xls_open — abre um arquivo Excel para leitura

```
[fd,SST,Sheetnames,Sheetpos] = xls_open(file_path)
```

# Parâmetros

file_path
: string: o endereço do arquivo Excel.

fd
: número, a unidade lógica do fluxo de dados do Excel

SST
: vetor de todos os strings que aparecem na folha do Excel

Sheetnames
: vetor de strings, os nomes das folhas

Sheetpos
: vetor de números,as posições dos inícios das folhas no fluxo de dados do Excel

# Descrição

Esta função primeiro analisa a estrutura de dados ole2 associada ao dado arquivo para extrair o fluxo de dado do Excel no qual está contido. Após isso, o fluxo de dados do Excel é salvo no diretório `TMDIR` e aberto. Uma unidade lógica `fd` aponta para este arquivo temporário. Então, a primeira folha neste fluxo é lida para se obter informações globais como o número de folhas, o nome das folhas `Sheetnames`, endereços das folhas dentro do fluxo `Sheetpos` e `SST` que contém todos os strings utilizados nas folhas.

Os dados `fd` e `Sheetpos` devem ser passados para xls_read para se ler as folhas de dados

A função readxls pode ser usada para se ler todo um arquivo Excel em uma função através de uma única chamada de função.

AVISO: apenas arquivos Excel BIFF8 (última versão do Excel (2003)) são manipulados.

# Exemplos

```
//Decodificando o arquivo ole, extraindo e abrindo o fluxo de dados do Excel
[fd,SST,Sheetnames,Sheetpos] = xls_open('SCI/modules/spreadsheet/demos/xls/Test
//Lendo a primeira folha de dados
[Value,TextInd] = xls_read(fd,Sheetpos(1))
//fechando o fluxo da planilha
mclose(fd)
```

# Ver Também

xls_read, readxls

# Autores

Pierrick Mode
: INRIA

Serge Steer
INRIA

# Bibliografia

Esta função é baseada na documentação do arquivo ole2 da Microsoft (http://chicago.sourceforge.net/devel/docs/ole/) e na descrição do OpenOffice sobre fluxo de dados do Excel (http://sc.openoffice.org/spreadsheetfileformat.pdf).

# Funções Utilizadas

O procedure ripole-0.1.4 (http://www.pldaniels.com/ripole) é utilizado para extrair do arquivo ole o fluxo de dados da planilha.

# Name

xls_read — lê uma folha em um arquivo Excel

```
[Value,TextInd] = xls_read(fd,Sheetpos)
```

# Parâmetros

fd
: um número, a unidade lógica do fluxo de dados do Excel retornado por xls_open.

Sheetpos
: um número: a posição do início da folha no fluxo de dados Excel. Esta posição é uma daquelas retornadas por xls_open.

Value
: matriz de números, os dados numéricos encontrados na folha. A célula sem valores numéricos é representada por valores `NaN`.

TextInd
: uma matriz de índices com o mesmo tamanho que `Value`. Os índices zeros indicam que não existem strings na celula Excel correspondente. Um índice positivo `i` aponta para o string `SST(i)` onde `SST` é dado por xls_open.

# Descrição

Esta função lê uma folha do Excel, dada uma unidade lógica, em um fluxo de dados do Excel e a posição do início da folha neste fluxo de dados. Ela retorna os dados numéricos e os strings contidos nas células do Excel.

A função readxls pode ser utilizada para ler todo um arquivo Excel em uma função utilizando apenas uma chamada a função.

AVISO: apenas arquivos Excel BIFF8 (última versão do Excel) são manipulados.

# Exemplos

```
//decodificando um arquivo ole, extraindo e abrindo um fluxo de dados do Excel
[fd,SST,Sheetnames,Sheetpos] = xls_open('SCI/modules/spreadsheet/demos/xls/Test
//lendo a primeira folha de dados
[Value,TextInd] = xls_read(fd,Sheetpos(1))
//fechando o fluxo de dados da planilha
mclose(fd)
```

# Ver Também

xls_open, readxls

# Autores

Pierrick Mode
: INRIA

Serge Steer
: INRIA

# Bibliografia

Esta função é baseada na descrição do OpenOffice sobre fluxo de dados (http://sc.openoffice.org/spreadsheetfileformat.pdf).

# Funções Utilizadas

Esta função utiliza o arquivo xls.c que pode ser encontrado em uma versão-fonte do Scilab no diretório directory SCIDIR/modules/spreadsheet/src/c

# Name

excel2sci — obsolete see read_csv

# See Also

read_csv , write_csv

# Name

read_csv — Read comma-separated value file

```
M = read_csv(fname [,sep])
```

# Parameters

fname
: a character string. The file path.

sep
: a character string. Field separator used, default value is ","

M
: a matrix of strings.

# Description

Given an ascii file with delimited fields, for instance created by a spreadsheet using "Text and comma" format, read_csv(fname) returns the corresponding Scilab matrix of strings. Use read_csv(fname,sep) for another choice of separator.

Note: You may eval all or part of M using function evstr in order to convert string variables into numeric variables.

# Examples

```
// create a file with some data separated with tab
A = 1:50;
mputl(strcat(string(A),ascii(9)), TMPDIR + '/foo.csv');

// read csv file
B = read_csv(TMPDIR + '/foo.csv');

// eval B
C = evstr(B);

// compares original data and result
and(A == C)
```

# See Also

write_csv, evstr

# Authors

Allan CORNET

# Name

write_csv — Read comma-separated value file

```
write_csv(M, filename)
```

# Parameters

filename
: a character string. The file path.

M
: a matrix of strings.

# Description

write_csv(M, filename) writes matrix M into filename as comma-separated values. The filename input is a string.

# Examples

```
// save a matrix as csv file format
A = [1:10] * 0.1;
write_csv(A, TMPDIR + '/datas.csv');

// read as text
mgetl(TMPDIR + '/datas.csv')

r = read_csv(TMPDIR + '/datas.csv',ascii(9));
r = strsubst(r,',','.');
evstr(r)
```

# See Also

read_csv, evstr, mgetl

# Authors

Allan CORNET

# Parte XLVII. call_scilab API

# Name

Boolean management — How to manage Scilab's boolean read and write process using call_scilab and api_scilab

# Description

This help describes how boolean and matrix of booleans can be handle through the Call Scilab API and API Scilab.

There are several functions which can be used to read / write from the memory of Scilab. These functions are described in dedicated pages.

# Examples

```
/*
 * Write a matrix of boolean into Scilab
 * B=[F F T F;
 *    F F F T ]
 * Note that it is done column by column
 */
int B[]={0,0,0,0,1,0,0,1};   /* Declare the matrix */
int rowB=2, colB=4; /* Size of the matrix */
char variableNameB[] = "B";

/* Write it into Scilab's memory */
createNamedMatrixOfBoolean(pvApiCtx, variableNameB, rowB, colB, B);
/*
 * Prior to Scilab 5.2:
 * C2F(cwritebmat)(variableNameB, &rowB, &colB, B, strlen(variableNameB));
 */

printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display B */
```

```
/* Read the previously declared matrix of boolean B */
int rowB_ = 0, colB_ = 0, lp_ = 0;
int i = 0, j = 0;
int *matrixOfBooleanB = NULL; /* Int instead of double */

char variableToBeRetrievedB[] = "B";

/* First, retrieve the size of the matrix */
readNamedMatrixOfBoolean(pvApiCtx, variableToBeRetrievedB, &rowB_, &colB_, NULL
/*
 * Prior to Scilab 5.2:
 * C2F(cmatbptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableT
 */

/* Alloc the memory */
matrixOfBooleanB=(int*)malloc((rowB_*colB_)*sizeof(int));

/* Load the matrix */
readNamedMatrixOfBoolean(pvApiCtx, variableToBeRetrievedB, &rowB_, &colB_, matr
/*
```

```
 * Prior to Scilab 5.2:
 * C2F(creadbmat)(variableToBeRetrievedB,&rowB_,&colB_,matrixOfBooleanB,strlen(
 */

printf("\n");
printf("Display from B formated (size: %d, %d):\n",rowB_, colB_);
for(j = 0 ; j < rowB_ ; j++)
{
 for(i = 0 ; i < colB_ ; i++)
 {
  /* Display the formated matrix ... the way the user expects */
  printf("%d ",matrixOfBooleanB[i * rowB_ + j]);
 }
 printf("\n"); /* New row of the matrix */
}
```

# See Also

API_Scilab: Boolean reading, API_Scilab: Boolean writing, SendScilabJob, StartScilab, Call_Scilab: Double Management, Call_Scilab: Complex Management, Call_Scilab: String Management, API_Scilab: Double Reading, API_Scilab: Double Writing, API_Scilab: String Reading, API_Scilab: String Writing

# Authors

Sylvestre Ledru

# Name

Complex management — How to manage Scilab's complex variable read and write process using call_scilab

# Description

This help describes how doubles and matrix of complex can be handle through the Call Scilab API.

There are several functions which can be used to read / write from the memory of Scilab. These functions are described in dedicated pages.

# Examples

```
/*
 * Write a matrix into Scilab
 * B=[1+%i 4-%i 2+1/2*%i 3;
 *    3 9 8+42*%i 2 ]
 * Note that it is done column by column
 */

double B[]={1,3,4,9,2,8,3,2,1,0,-1,0,1/2,42,0,0};   /* Declare the matrix */
int rowB=2, colB=4; /* Size of the matrix */
char variableNameB[] = "B";

C2F(cwritecmat)(variableNameB, &rowB, &colB, B, strlen(variableNameB)); /* Write

printf("\n");
printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display B */
```

```
int rowB_ = 0, colB_ = 0, lp_ = 0;
int i = 0,j = 0;

double *matrixOfComplexB = NULL;
char variableToBeRetrievedB[] = "B";

/* First, retrieve the size of the matrix */
C2F(cmatcptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableToBe

/* Alloc the memory */
matrixOfComplexB = (double*)malloc((rowB_*colB_*2)*sizeof(double));

/* Load the matrix */
C2F(creadcmat)(variableToBeRetrievedB,&rowB_,&colB_,matrixOfComplexB,strlen(var

printf("\n");
printf("Display from B formated (size: %d, %d):\n",rowB_, colB_);
for(j = 0 ; j < rowB_ ; j++)
 {
  for(i = 0 ; i < colB_ ; i++)
   {
    /* Display the formated matrix ... the way the user
     * expect */
    printf("%5.2f + %5.2f.i  ",matrixOfComplexB[i * rowB_ + j],matrixOfComplexB
```

```
  }
 printf("\n"); /* New row of the matrix */
}
```

# See Also

cwritecmat, creadcmat, SendScilabJob, StartScilab, Call_Scilab: Complex Management, Call_Scilab: Boolean Management, Call_Scilab: String Management, API_Scilab: Boolean Reading, API_Scilab: Boolean Writing, API_Scilab: String Reading, API_Scilab: String Writing

# Authors

Sylvestre Ledru

# Name

DisableInteractiveMode — Disables some features (plotting, gui creation, Tcl/Tk...) and leave only the computing engine

# Synopsis

```
void DisableInteractiveMode(void);
```

# Description

Calling this fonction will disable some features of Scilab.

# Examples

```
// A simple DisableInteractiveMode example

DisableInteractiveMode();
int code=SendScilabJob("plot3d()"); /* This will failed since plot3d is
                                       among the disable features*/
if (code!=0){
 char lastjob[4096];
 if (GetLastJob(lastjob,4096))  {
  printf("Failed command: %s\n",lastjob);
 }
}
```

# See Also

Compile and run with call_scilab SendScilabJob TerminateScilab, Double Management, Boolean Management, Complex Management, String Management

# Authors

Sylvestre Ledru

# Name

Double management — How to manage Scilab's variable read and write process using call_scilab and api_scilab

# Description

This help describes how doubles and matrix of doubles can be handle through the Call Scilab API and API Scilab.

There are several functions which can be used to read / write from the memory of Scilab. These functions are described in dedicated pages.

Note that the default type of a numeric value in Scilab is double. For example, in *a=1* or *a=[1,2]*, *a* will be consider as a double.

# Examples

```
/*
 * Write a matrix into Scilab
 * B=[1 4 2 3;
 *    3 9 8 2 ]
 * Note that it is done column by column
 */
double B[] = {1,3,4,9,2,8,3,2};   /* Declare the matrix */
int rowB = 2, colB = 4; /* Size of the matrix */
char variableNameB[] = "B";

/*
 * Write it into Scilab's memory
 */
createNamedMatrixOfDouble(pvApiCtx,variableNameB,rowB,colB, B); /* pvApiCtx is a
/*
 * Prior to Scilab 5.2:
 * C2F(cwritemat)(variableNameB, &rowB, &colB, B, strlen(variableNameB));
*/

printf("\n");
printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display B */
```

```
/* Read the previously declared matrix of double B */
int rowB_ = 0, colB_ = 0, lp_ = 0;
double *matrixOfDoubleB = NULL;
int i = 0, j = 0;
char variableToBeRetrievedB[] = "B";


/* First, retrieve the size of the matrix */
readNamedMatrixOfDouble(pvApiCtx, variableToBeRetrievedB, &rowB_, &colB_, NULL)
/*
 * Prior to Scilab 5.2:
 * C2F(cmatptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableToB
 */
```

```
/* Alloc the memory */
matrixOfDoubleB = (double*)malloc((rowB_*colB_)*sizeof(double));

/* Load the matrix */
readNamedMatrixOfDouble(pvApiCtx, variableToBeRetrievedB, &rowB_, &colB_, matri
/*
 * Prior to Scilab 5.2:
 * C2F(creadmat)(variableToBeRetrievedB,&rowB_,&colB_,matrixOfDoubleB,strlen(va
 */

printf("\n");
printf("Display from B formated (size: %d, %d):\n",rowB_, colB_);
for(j = 0 ; j < rowB_ ; j++)
 {
  for(i = 0 ; i < colB_ ; i++)
   {
    /* Display the formated matrix ... the way the user
     * expect */
    printf("%5.2f ",matrixOfDoubleB[i * rowB_ + j]);
   }
  printf("\n"); /* New row of the matrix */
 }
```

# See Also

API_Scilab: Double reading, API_Scilab: Double writing, SendScilabJob, StartScilab, Call_Scilab: BooleanManagement, Call_Scilab: ComplexManagement, Call_Scilab: StringManagement, API_Scilab: Boolean Reading, API_Scilab: Boolean Writing, API_Scilab: String Reading, API_Scilab: String Writing

# Authors

Sylvestre Ledru

# Name

GetLastJob — Returns the latest job sent to Scilab engine

# Synopsis

```
BOOL GetLastJob(char *JOB,int nbcharsJOB);
```

# Description

This fonction returns the latest job sent to Scilab engine with the command SendScilabJob or SendScilabJobs. This can be used to display a command which failed.

BOOL is just a simple typedef on int (`typedef int BOOL`). TRUE is defined on 1 (`#define TRUE 1`) and FALSE is defined on 0 (`#define FALSE 0`).

# Parameters

JOB
: a standard C char* which will be filled with the latest job

nbcharsJOB
: The number of char of JOB

returns
: 1 (TRUE) if the operation is successfull.

  0 (FALSE) if an error during initialization occured.

# Examples

```
// A simple GetLastJob example
// See SCI/modules/call_scilab/examples/basicExamples/GetLastJob.c for
// the full code
int code=SendScilabJob("failedMyCurrentJob=%pi*3/0");
if (code!=0){
 char lastjob[4096];
 if (GetLastJob(lastjob,4096))  {
  printf("Failed command: %s\n",lastjob);
 }
}
```

# See Also

Compile and run with call_scilab, SendScilabJob, TerminateScilab, Double Management, Boolean Management, Complex Management, String Management

# Authors

Sylvestre Ledru

# Name

ScilabHaveAGraph — Check if any Scilab graphics have been opened.

# Synopsis

```
int ScilabHaveAGraph(void);
```

# Description

Returns if there is any Scilab graphics opened.

# Examples

```
// A simple ScilabHaveAGraph example
int code=SendScilabJob("plot3d()"); /* This will failed since plot3d is
                                       among the disable features*/
if (ScilabHaveAGraph()){
 printf("Graphics\n");
}else{
 printf("NO Graphics\n");
}
```

# See Also

Compile and run with call_scilab SendScilabJob, TerminateScilab, Double Management, Boolean Management, Complex Management, String Management

# Authors

Sylvestre Ledru

# Name

SendScilabJob — Send a Scilab task from a C/C++ code (call_scilab)

# Synopsis

```
int SendScilabJob(char *job);
```

# Description

This fonction is provided in call_scilab. This function send a task which will be proccessed by the Scilab engine.

# Parameters

job
: a standard C char* containing the Scilab instructions

returns
: 0 is the operation is successfull.

: -1 if Call Scilab has not been able to write the job into Scilab.

: -2 or -3 if Call Scilab has not been able to read the error code from Scilab.

: -4 if Call Scilab has not been able to allocate the job.

# Examples

```
// A simple SendScilabJob example

if (SendScilabJob("disp('unfinished quote)")!=0){ // This will fail
 printf("An error occured\n");
}
SendScilabJob("myMatrix=['sample','for the help']");
```

# See Also

Compile and run with call_scilab, SendScilabJob, Double Management, Boolean Management, Complex Management, String Management

# Authors

Sylvestre Ledru

# Name

SendScilabJobs — Send Scilab tasks from a C/C++ code (call_scilab)

# Synopsis

```
int SendScilabJobs(char **jobs, int numberjobs);
```

# Description

This fonction is provided in call_scilab. This function send many tasks which will be proccessed by the Scilab engine.

Note that the ending ";" is not mandatory at the end of a command.

# Parameters

jobs
: an array of standard C char* containing the Scilab instructions

numberjobs
: The number of the Scilab instructions

returns
: 0 is all the operations are successful.

  -10 if .

  <0 and > -10 when an error occurred in the execution.

# Examples

```
// A simple SendScilabJobs example

char* jobs[3];
jobs[0]="a = 1";
jobs[1]="b = 3";
jobs[2]="c = a + b;";
SendScilabJobs(jobs,3);
SendScilabJob("disp(c);"); // Will display   4.
```

# See Also

SendScilabJob, Compile and run with call_scilab, Double Management, Boolean Management, Complex Management, String Management

# Authors

Sylvestre Ledru

# Name

StartScilab — Initializes and starts Scilab engine in Call Scilab

# Synopsis

```
BOOL StartScilab(char *SCIpath, char *ScilabStartup, int
    *Stacksize);
```

# Description

This fonction starts the Scilab engine. This is mandatory to use SendScilabJob functions and to manage Scilab's data.

BOOL is just a simple typedef on int (`typedef int BOOL`). TRUE is defined on 1 (`#define TRUE 1`) and FALSE is defined on 0 (`#define FALSE 0`).

# Parameters

SCIpath
: a standard C char* containing the path to Scilab data

  This argument is mandatory under Linux, Unix or Mac OS X.

  Under Windows, if SCIpath is NULL, Scilab will find the path.

ScilabStartup
: a standard C char* containing the path to Scilab startup script (scilab.start)

  If ScilabStartup is NULL, Scilab will use the default path (detected from SCIpath).

Stacksize
: a standard int* defining the size of the Scilab stack

  If Stacksize is NULL, Scilab will use the default stacksize of Scilab.

returns
: 1 (TRUE) if the operation is successfull.

  0 (FALSE) if an error during initialization occured.

# Examples

```
// A simple StartScilab example
if ( StartScilab(getenv("SCI"),NULL,NULL) == FALSE )
 {
  fprintf(stderr,"Error while calling StartScilab\n");
  return -1;
 }
```

# See Also

Compile and run with call_scilab, SendScilabJob, TerminateScilab, Double Management, Boolean Management, Complex Management, String Management

# Authors

Sylvestre Ledru

# Name

String management — How to manage Scilab's String read and write process using call_scilab and api_scilab

# Description

This help describes how strings and matrix of strings can be handle through the Call Scilab APIand API Scilab.

There are several functions which can be used to read / write from the memory of Scilab. These functions are described in dedicated pages.

# Examples

```
// This example shows how to write a Scilab string in Scilab engine
// It is the equivalent to A="my Message"; in Scilab interpretor
// See: modules/call_scilab/examples/basicExamples/readwritestring.c

// StartScilab
int row = 1, col = 1; /* Size of the matrix */
/* Declare the string */
char **myMatrixOfString = (char**)malloc(sizeof(char*) * row * col);
myMatrixOfString[0]="my Message";
char variableName[] = "A";

/* Write it into Scilab's memory */
createNamedMatrixOfString(pvApiCtx, variableName, row, col, myMatrixOfString);
/*
 * Prior to Scilab 5.2
 * C2F(cwritechain)(variableName, &sizeOfMyString  , myString, strlen(variableN
*/

printf("Display from Scilab of A:\n");
SendScilabJob("disp(A);"); /* Display A */
```

```
/* Load the previously set variable A */
// See: modules/call_scilab/examples/basicExamples/readwritestring.c


char variableToBeRetrieved[]="A";
int iRows       = 0;
int iCols       = 0;
int i,j;
int* piAddr     = NULL;
int* piLen      = NULL;
char** pstData  = NULL;

//fisrt call to retrieve dimensions
readNamedMatrixOfString(pvApiCtx,variableToBeRetrieved,&iRows, &iCols, NULL, NU
piLen = (int*)malloc(sizeof(int) * iRows * iCols);
//second call to retrieve length of each string
readNamedMatrixOfString(pvApiCtx,variableToBeRetrieved, &iRows, &iCols, piLen, 
pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
```

```
 {
  pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null term
 }
//third call to retrieve data
readNamedMatrixOfString(pvApiCtx, variableToBeRetrieved, &iRows, &iCols, piLen,

printf("\n");
printf("Load and display of A:\n");
for(j = 0 ; j < iCols ; j++)
{
 for(i = 0 ; i < iRows ; i++)
 {
  /* Display the formated matrix with same scilab indice */
  printf("[%d,%d] = %s\n",j+1,i+1,pstData[j* iRows + i]);
 }
}

printf("\n");
free(piLen);
for(i = 0 ; i < iRows * iCols ; i++)
 {
  free(pstData[i]);
 }
free(pstData);
```

# See Also

cwritechain, creadchain, SendScilabJob, StartScilab, Call_Scilab: Boolean Management, Call_Scilab: Double Management, API_Scilab: Boolean Reading, API_Scilab: Boolean Writing, API_Scilab: Double Reading, API_Scilab: Double Writing, Complex Management

# Authors

Sylvestre Ledru

# Name

TerminateScilab — Stops and terminates Scilab engine in Call Scilab

# Synopsis

```
BOOL TerminateScilab(char *ScilabQuit);
```

# Description

This fonction stops the Scilab engine. It is strongly recommanded to call this function at the end when using Call Scilab .

BOOL is just a simple typedef on int (`typedef int BOOL`). TRUE is defined on 1 (`#define TRUE 1`) and FALSE is defined on 0 (`#define FALSE 0`).

# Parameters

ScilabQuit
: a standard C char* containing the path to Scilab quit script (scilab.quit)

  If ScilabStartup is NULL, Scilab will use the default path (detected from SCIpath).

returns
: 1 (TRUE) if the operation is successfull.

  0 (FALSE) if an error during initialization occured.

# Examples

```
// A simple TerminateScilab example
if ( TerminateScilab(NULL) == FALSE ) {
 fprintf(stderr,"Error while calling TerminateScilab\n");
 return -2;
}
```

# See Also

Compile and run with call_scilab, StartScilab, SendScilabJob, Double Management, Boolean Management, Complex Management, String Management

# Authors

Sylvestre Ledru

# Name

call_scilab — call_scilab is an interface which provides the ability to call Scilab engine from C/C++ code

# Description

Scilab offers the possibility to be called from a native (C/C++) application.

Thanks to this module, it is possible to call Scilab from C/C++ in order to interface Scilab's features from an other code/application or to be able to interface Scilab's features from an other language.

It is also possible to read and write the most important Scilab's datastructures thanks to this module

This help describes the features of the call_scilab API.

Note that the javasci module is based on call_scilab.

# Examples

```
// A simple call_scilab example

#include <stdio.h> /* stderr */

#include "stack-c.h" /* Provide functions to access to the memory of Scilab */
#include "call_scilab.h" /* Provide functions to call Scilab engine */

// Filename: simple_call_scilab.c

int main(void)
{
/****** INITIALIZATION **********/
#ifdef _MSC_VER
 if ( StartScilab(NULL,NULL,NULL) == FALSE )
#else
 if ( StartScilab(getenv("SCI"),NULL,NULL) == FALSE )
#endif
  {
   fprintf(stderr,"Error while calling StartScilab\n");
   return -1;
  }

/****** ACTUAL Scilab TASKS *******/

 SendScilabJob("myMatrix=['sample','for the help']");
 SendScilabJob("disp(myMatrix);"); // Will display !sample  for the help  !
 SendScilabJob("disp([2,3]+[-44,39]);"); // Will display   - 42.    42.

/****** TERMINATION **********/
 if ( TerminateScilab(NULL) == FALSE ) {
  fprintf(stderr,"Error while calling TerminateScilab\n");
  return -2;
 }
 return 0;
}
```

# See Also

Compile and run with call_scilab, Matrix Management, Boolean Management, Complex Management, String Management

# Authors

Sylvestre Ledru

# Name

Compile and run with Call Scilab — How to compile a native application based on or using Scilab

# Compilation

To compile a native code based on Call Scilab, it is necessary to define some arguments, variables and paths.

- CFLAGS. Call Scilab needs to have access to the headers of Scilab core and call Scilab module.
  - Linux/Unix/MacOSX:
    - In the binary version of Scilab, CFLAGS must be set to /path/to/scilab/include/scilab/core/ and /path/to/scilab/include/scilab/call_scilab/
    - In the source tree of Scilab, CFLAGS must be set to /path/to/scilab/modules/core/includes/ and /path/to/scilab/modules/call_scilab/includes/
  - Windows
- LD_LIBRARY_PATH - Paths to libscilab.so and libjavasci.so (or .dll, .jnilib...)
  - Linux/Unix/MacOSX:
    - In the binary version of Scilab, SCI will point to /path/to/scilab/lib/scilab/
    - In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/modules/call_scilab/.libs/ and /path/to/scilab/.libs/
- LDFLAGS - The name of the library to link against
  - Linux/Unix/MacOSX: It is only mandatory to link against scilab. This should include the other libraries.

```
# A sample Makefile building a C code using Call Scilab using Scilab built in i
PATH_SCILAB = /path/to/scilab/sources.
SCILAB_CFLAGS = -I$(PATH_SCILAB)/modules/core/includes/ -I$(PATH_SCILAB)/module
SCILAB_LDFLAGS = -lscilab
PATH_TO_LIB_SCILAB = $(PATH_SCILAB)/modules/.libs/
PATH_TO_LIB_CALL_SCILAB = $(PATH_SCILAB)/modules/call_scilab/.libs/

all: simple_call_scilab.c
 export LD_LIBRARY_PATH=$(PATH_TO_LIB_SCILAB):$(PATH_TO_LIB_CALL_SCILAB)
 gcc -o myExample $(SCILAB_LDFLAGS) -L$(PATH_TO_LIB_SCILAB) -L$(PATH_TO_LIB_CAL
```

# Running

To run an application based on Call Scilab, there are a few other things to set up.

Some global variables must me set:

- SCI - Path to Scilab files
  - Linux/Unix/MacOSX:
    - In the binary version of Scilab, SCI will point to /path/to/scilab/share/scilab/

    - In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/source/tree/
  - Windows:
- LD_LIBRARY_PATH - Paths to libscilab.so and libjavasci.so (or .dll, .jnilib...)
  - Linux/Unix/MacOSX:
    - In the binary version of Scilab, SCI will point to /path/to/scilab/lib/scilab/
    - In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/modules/javasci/.libs/ and /path/to/scilab/.libs/
- LD_LIBRARY_PATH (Java) - Paths to Java native libraries (libjava.so, libjvm.so, libhpi.so)... It is usually provided by the operating system or by Scilab distribution. Please note that won't be necessary in Scilab 5.2
  - Linux/Unix:
    - JAVA_HOME/jre/lib/<arch>/, JAVA_HOME/jre/lib/<arch>/server, JAVA_HOME/jre/lib/<arch>/native_threads/ (<arch> can be i386, etc...)
  - Mac OS X:
  - Windows

Note that two environnement variables are taken in account for specific needs:

- `SCI_DISABLE_TK=1` Disables Tk (Tcl's GUI)
- `SCI_JAVA_ENABLE_HEADLESS=1` Launch Java in headless mode (no AWT/Swing)

# Examples

```
# Serie of declarations to execute my binary.

# With a Scilab source tree:
$ SCI=<path to Scilab source tree>
$ export LD_LIBRARY_PATH=$SCI/modules/.libs/:$SCI/modules/call_scilab/.libs/:$S

# With a Scilab binary:
$ SCI=<path to Scilab binary>
$ export LD_LIBRARY_PATH=$SCI/lib/scilab/

# Don't forget to update i386 to whatever is necessary
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/jvm/java-6-openjdk//jre/lib/
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/jvm/java-6-openjdk//jre/lib/
$ export SCI=/path/to/scilab/
$ ./myExample
!sample  for the help  !

  - 42.    42.
```

# See Also

call_scilab, StartScilab, SendScilabJob, SendScilabJobs, Double Management, Boolean Management, Complex Management, String Management

# Authors

Allan Cornet

Sylvestre Ledru

# Name

creadbmat (obsolete) — Read a single boolean or a matrix of boolean from Scilab memory using call_scilab. Starting with Scilab 5.2, this function is obsolete. See API_Scilab: Boolean reading for remplacement.

# Synopsis

```
int C2F(creadbmat)(char *name, int *m, int *n, int *scimat, unsigned long name_
```

# Parameters

name
: The name of the future Scilab variable

m
: Number of rows

n
: Number of columns

scimat
: The actual matrix of boolean (array of int). Note that it is going to be stored in Scilab columnwise.

name_len
: The length of the variable *name* (fortran compatibility)

C2F
: C2F is just a macro which provides to this function the ability to be called from fortran

# Description

This help describes how to use the function creadbmat.

Using this function will retrieve a variable called *name* from Scilab memory into a standard C *int* *.

# Examples

```
/* Load the previously set variable B */
// See: modules/call_scilab/examples/basicExamples/readwriteboolean.c
int rowB_ = 0, colB_ = 0, lp_ = 0;
int i = 0, j = 0;
int *matrixOfBooleanB = NULL; /* Int instead of double */

char variableToBeRetrievedB[] = "B";

/* First, retrieve the size of the matrix */
C2F(cmatbptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableToBe

/* Alloc the memory */
matrixOfBooleanB=(int*)malloc((rowB_*colB_)*sizeof(int));

/* Load the matrix */
C2F(creadbmat)(variableToBeRetrievedB,&rowB_,&colB_,matrixOfBooleanB,strlen(var

printf("\n");
printf("Display from B formated (size: %d, %d):\n",rowB_, colB_);
```

```
for(j = 0 ; j < rowB_ ; j++)
{
 for(i = 0 ; i < colB_ ; i++)
 {
  /* Display the formated matrix ... the way the user
   * expect */
  printf("%d ",matrixOfBooleanB[i * rowB_ + j]);
 }
 printf("\n"); /* New row of the matrix */
}
```

# See Also

API_Scilab: Boolean reading, API_Scilab: Boolean writing, SendScilabJob, StartScilab, cwritebmat, Boolean Management

# Authors

Sylvestre Ledru

# Name

creadchain (obsolete) — Read a single string from Scilab memory using call_scilab. Starting with Scilab 5.2, this function is obsolete. See API_Scilab: String reading for remplacement.

# Synopsis

```
int C2F(creadchain)(char *name, int *itslen, char *myString, unsigned long name_
```

# Parameters

name
: The name of the future Scilab variable

itslen
: The length of the future buffer (usually, use *bsiz*, it skips the need to detect the size of a char). This variable will be altered to contain the actual size of *myString*

myString
: The actual String (char *) which is going to content the content of the Scilab variable *name*

name_len
: The length of the variable *name* (fortran compatibility)

myString_len
: The length of the string (fortran compatibility)

C2F
: C2F is just a macro which provides to this function the ability to be called from fortran

# Description

This help describes how to use the function creadchain.

Using this function will retrieve a variable called *name* from Scilab memory into a standard C *char* *.

# Examples

```
/* Load the previously set variable A */
// See: modules/call_scilab/examples/basicExamples/readwritestring.c
int sizeA = 0;
char myStringFromScilab[bsiz]; /* Static char */
int length_myStringFromScilab = bsiz; /* Max size (it is going to be changed by
char variableToBeRetrieved[]="A";

/* We are loading a single string from Scilab */
C2F(creadchain)(variableToBeRetrieved,&length_myStringFromScilab,myStringFromSc

printf("\n");
printf("Display of A (size %d): %s\n", length_myStringFromScilab, myStringFromS
```

# See Also

API_Scilab: String reading, API_Scilab: String writing, SendScilabJob, StartScilab, cwritechain, String Management

# Authors

Sylvestre Ledru

# Name

creadcmat (obsolete) — Read a single complex or a matrix of complex from Scilab memory using call_scilab. Starting with Scilab 5.2, this function is obsolete. See API_Scilab: Complex double reading for remplacement.

# Synopsis

```
int C2F(creadcmat)(char *name, int *m, int *n, double *scimat, unsigned long na
```

# Parameters

name
: The name of the future Scilab variable

m
: Number of rows

n
: Number of columns

scimat
: The actual matrix of complex (array of double). Note that it is going to be stored in Scilab columnwise and the second half of the array is used for complex values.

name_len
: The length of the variable *name* (fortran compatibility)

C2F
: C2F is just a macro which provides to this function the ability to be called from fortran

# Description

This help describes how to use the function creadcmat.

Using this function will retrieve a variable called *name* from Scilab memory into a standard C *double **.

# Examples

```
/* Load the previously set variable B */
// See: modules/call_scilab/examples/basicExamples/readwritecomplexmatrix.c
int rowB_ = 0, colB_ = 0, lp_ = 0;
int i = 0,j = 0;

double *matrixOfComplexB = NULL;
char variableToBeRetrievedB[] = "B";

/* First, retrieve the size of the matrix */
C2F(cmatcptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableToBe

/* Alloc the memory */
matrixOfComplexB = (double*)malloc((rowB_*colB_*2)*sizeof(double));

/* Load the matrix */
C2F(creadcmat)(variableToBeRetrievedB,&rowB_,&colB_,matrixOfComplexB,strlen(var

printf("\n");
```

```
printf("Display from B formated (size: %d, %d):\n",rowB_, colB_);
for(j = 0 ; j < rowB_ ; j++)
 {
  for(i = 0 ; i < colB_ ; i++)
   {
    /* Display the formated matrix ... the way the user
     * expect */
    printf("%5.2f + %5.2f.i  ",matrixOfComplexB[i * rowB_ + j],matrixOfComplexB
   }
  printf("\n"); /* New row of the matrix */
 }
```

# See Also

API_Scilab: Complex double reading, API_Scilab: Double writing, SendScilabJob, StartScilab, cwritecmat, Complex Management

# Authors

Sylvestre Ledru

# Name

creadmat (obsolete) — Read a single double or a matrix of double from Scilab memory using call_scilab. Note that it is the default datatype in Scilab. Starting with Scilab 5.2, this function is obsolete. See API_Scilab: Double reading for remplacement.

# Synopsis

```
int C2F(creadmat)(char *name, int *m, int *n, double *scimat, unsigned long name
```

# Parameters

name
: The name of the future Scilab variable

m
: Number of rows

n
: Number of columns

scimat
: The actual matrix of double (array of double). Note that it is going to be stored in Scilab columnwise.

name_len
: The length of the variable *name* (fortran compatibility)

C2F
: C2F is just a macro which provides to this function the ability to be called from fortran

# Description

This help describes how to use the function creadmat.

Using this function will retrieve a variable called *name* from Scilab memory into a standard C *double **.

# Examples

```
/* Load the previously set variable B */
// See: modules/call_scilab/examples/basicExamples/readwritematrix.c
int rowB_ = 0, colB_ = 0, lp_ = 0;
double *matrixOfDoubleB = NULL;
int i = 0, j = 0;

char variableToBeRetrievedB[] = "B";

/* First, retrieve the size of the matrix */
C2F(cmatptr)(variableToBeRetrievedB, &rowB_, &colB_, &lp_, strlen(variableToBeR

/* Alloc the memory */
matrixOfDoubleB = (double*)malloc((rowB_*colB_)*sizeof(double));

/* Load the matrix */
C2F(creadmat)(variableToBeRetrievedB,&rowB_,&colB_,matrixOfDoubleB,strlen(varia

printf("\n");
```

```
printf("Display from B formated (size: %d, %d):\n",rowB_, colB_);
for(j = 0 ; j < rowB_ ; j++)
 {
  for(i = 0 ; i < colB_ ; i++)
   {
    /* Display the formated matrix ... the way the user
     * expect */
    printf("%5.2f ",matrixOfDoubleB[i * rowB_ + j]);
   }
  printf("\n"); /* New row of the matrix */
 }
```

# See Also

API_Scilab: Double reading,API_Scilab: Double writing, SendScilabJob, StartScilab, cwritemat, Double Management

# Authors

Sylvestre Ledru

# Name

cwritebmat (obsolete) — Write a single boolean or a matrix of boolean into Scilab memory using call_scilab. Starting with Scilab 5.2, this function is obsolete. See API_Scilab: Boolean writing for remplacement.

# Synopsis

```
int C2F(cwritebmat)(char *name, int *m, int *n, int *mat, unsigned long name_le
```

# Parameters

name
: The name of the future Scilab variable

m
: Number of rows

n
: Number of columns

mat
: The actual matrix of boolean (array of int). Note that it is going to be stored in Scilab columnwise.

name_len
: The length of the variable *name* (fortran compatibility)

C2F
: C2F is just a macro which provides to this function the ability to be called from fortran

# Description

This help describes how to use the function cwritebmat.

Using this function will basically do the same as *A=[ T F F T ];* but straight into Scilab memory with call_scilab.

# Examples

```
// This example shows how to write a Scilab matrix of boolean in Scilab engine
// It is the equivalent to
// B=[F F T F;
//    F F F T ] in Scilab interpretor
// See: modules/call_scilab/examples/basicExamples/readwriteboolean.c

// StartScilab
int B[]={0,0,0,0,1,0,0,1};   /* Declare the matrix */
int rowB=2, colB=4; /* Size of the matrix */
char variableNameB[] = "B";
C2F(cwritebmat)(variableNameB, &rowB, &colB, B, strlen(variableNameB)); /* Writ
printf("\n");
printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display B */
```

# See Also

API_Scilab: Boolean reading, API_Scilab: Boolean writing, SendScilabJob, StartScilab, creadbmat, Boolean Management

# Authors

Sylvestre Ledru

# Name

cwritechain (obsolete) — Write a single string into Scilab memory using call_scilab. Starting with Scilab 5.2, this function is obsolete. See API_Scilab: String writing for remplacement.

# Synopsis

```
int C2F(cwritechain)(char *name, int *myStringSize, char *myString, unsigned lo
```

# Parameters

name
: The name of the future Scilab variable

myStringSize
: The length of the string which is going to be write into Scilab memory

myString
: The actual String (char *)

name_len
: The length of the variable *name* (fortran compatibility)

myString_len
: The length of the string (fortran compatibility)

C2F
: C2F is just a macro which provides to this function the ability to be called from fortran

# Description

This help describes how to use the function cwritechain.

Using this function will basically do the same as *A = "my own String";* but straight into Scilab memory with call_scilab.

# Examples

```
// This example shows how to write a Scilab string in Scilab engine
// It is the equivalent to A="my Message"; in Scilab interpretor
// See: modules/call_scilab/examples/basicExamples/readwritestring.c

// StartScilab
char *myString = "my Message";   /* Declare the string */
char variableName[] = "A"; / * The name of the future variable in Scilab */
int sizeOfMyString=strlen(myString);

C2F(cwritechain)(variableName, &sizeOfMyString  , myString, strlen(variableName

printf("Display from Scilab of A:\n");
SendScilabJob("disp(A);"); /* Display A */
```

# See Also

API_Scilab: String reading, API_Scilab: String writing, SendScilabJob StartScilab creadchains String Management

# Authors

Sylvestre Ledru

# Name

cwritecmat (obsolete) — Write a single complex or a matrix of complex into Scilab memory using call_scilab. Starting with Scilab 5.2, this function is obsolete. See API_Scilab: Complex double writing for remplacement.

# Synopsis

```
int C2F(cwritecmat)  (char *name, int *m, int *n, double *mat, unsigned long na
```

# Parameters

name
: The name of the future Scilab variable

m
: Number of rows

n
: Number of columns

mat
: The actual matrix of complex (array of double). Note that it is going to be stored in Scilab columnwise and the second half of the array is used for complex values.

name_len
: The length of the variable *name* (fortran compatibility)

C2F
: C2F is just a macro which provides to this function the ability to be called from fortran

# Description

This help describes how to use the function cwritecmat.

Using this function will basically do the same as *A=[ 1+i 3i 4 2+2i ];* but straight into Scilab memory with call_scilab.

# Examples

```
// This example shows how to write a Scilab matrix of complex in Scilab engine
/*
 * Write a matrix into Scilab
 * B=[1+%i 4-%i 2+1/2*%i 3;
 *    3 9 8+42*%i 2 ]
 * Note that it is done column by column
 */

double B[]={1,3,4,9,2,8,3,2,1,0,-1,0,1/2,42,0,0};   /* Declare the matrix */
int rowB=2, colB=4; /* Size of the matrix */
char variableNameB[] = "B";

C2F(cwritecmat)(variableNameB, &rowB, &colB, B, strlen(variableNameB)); /* Writ

printf("\n");
printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display B */
```

## See Also

API_Scilab: Complex double reading, API_Scilab: Complex double writing, SendScilabJob, StartScilab, Complex Management, creadcmat

## Authors

Sylvestre Ledru

# Name

cwritemat (obsolete) — Write a single double or a matrix of double into Scilab memory using call_scilab. Note that it is the default datatype in Scilab.Starting with Scilab 5.2, this function is obsolete. See API_Scilab: Double writing for remplacement.

# Synopsis

```
int C2F(cwritemat)(char *name, int *m, int *n, int *mat, unsigned long name_len
```

# Parameters

name
: The name of the future Scilab variable

m
: Number of rows

n
: Number of columns

mat
: The actual matrix of double (array of double). Note that it is going to be stored in Scilab columnwise.

name_len
: The length of the variable *name* (fortran compatibility)

C2F
: C2F is just a macro which provides to this function the ability to be called from fortran

# Description

This help describes how to use the function cwritemat.

Using this function will basically do the same as *A=[ 1 3 4 2 ];* but straight into Scilab memory with call_scilab.

# Examples

```

// This example shows how to write a Scilab matrix of double in Scilab engine
// * Write a matrix into Scilab
// * B=[1 4 2 3;
// *    3 9 8 2 ]
// * Note that it is done column by column
// See: modules/call_scilab/examples/basicExamples/readwritematrix.c

// StartScilab
double B[] = {1,3,4,9,2,8,3,2};   /* Declare the matrix */
int rowB = 2, colB = 4; /* Size of the matrix */
char variableNameB[] = "B";

C2F(cwritemat)(variableNameB, &rowB, &colB, B, strlen(variableNameB)); /* Write

printf("\n");
printf("Display from Scilab of B:\n");
SendScilabJob("disp(B);"); /* Display A */
```

## See Also

API_Scilab: Double reading,API_Scilab: Double writing, SendScilabJob, StartScilab, Double Management, creadmat

## Authors

Sylvestre Ledru

# Name

fromc — Checks if current Scilab is called from an external C program (by StartScilab).

```
r=fromc()
```

# Parameters

r
: a boolean

# Description

Returns `%t`Checks if current Scilab is called from an external C program or `%f` if not.

# See Also

fromjava

# Name

fromjava — Checks if current Scilab is called from javasci

```
r = fromjava()
```

# Parameters

r
: a boolean

# Description

Returns `%t` if Scilab is called from javasci or `%f` if not.

# See Also

fromc

# Parte XLVIII. FFTW

# Name

fftw — Fast fourier transform based on the fftw library

```
[y]=fftw(x)
[y]=fftw(x,sign)
[y]=fftw(x,sign,dim,incr)
[y]=fftw(x,sign,[dim1 dim2 ...dimN],[incr1 incr2 ...incrN])
```

# Parameters

y,x
: matrix/vector of real/complex data. Input/output data to be transformed.

sign
: Integer. 1 or -1. Set direct or inverse transform.

dim
: integer. Set the dimension (the length) of the transform.

incr
: integer. Set the stride (the span) of the transform.

# Description

This function realizes direct/inverse Discrete Fourier Transform (DFT) with the help of the FFTW library.

One can compute vector, 2D, M-D transform with this function.

For more details of fftw syntax see fft scilab function.

For more details about FFTW library see FFTW Web site : http://www.fftw.org

Remark: fftw function automatically stores his last parameters in memory to re-use it in a second time.

This improves greatly the time computation when consecutives calls (with same parameters) are performed.

# Examples

```
//simple vector direct transform
a = rand(50,1)+%i*rand(50,1);
y = fftw(a);
y = fftw(a,-1);
//inverse transform
b = fftw(y,1);

//2D transform
a = rand(512,512)+%i*rand(512,512);
y = fftw(a);

//M-D transform -old calling sequence-
a = rand(120,1);
y = a;
dim=[5 6 4];incr=[1 5 30];
for i=1:3
```

```
  y = fftw(y,-1,dim(i),incr(i));
end

//M-D transform -new calling sequence-
//More efficient than old
y = fftw(a,-1,[5 6 4],[1 5 30]);
b = fftw(y,1,[5 6 4],[1 5 30]);
```

# See Also

fftw_flags, get_fftw_wisdom, set_fftw_wisdom, fftw_forget_wisdom

# Bibliography

Matteo Frigo and Steven G. Johnson, "FFTW Manual fo version 3.1.2" June 2006. Available : http://www.fftw.org

# Name

fftw_flags — set computation method of fast fourier transform of the fftw function

```
[a,[S]]=fftw_flags([x1;x2;...])
```

# Parameters

[x1;x2;...]
: Matrix of string or integers. Entry to switch the method of fft computation for fftw.

a
: Integer. Give the current value of the flag of the fftw function.

S
: String matrix. Give the string value of the fftw flag.

# Description

This function enables the change of the `unsigned flags` parameter of the `fftw_plan_guru_split_dft` function that is used in fftw function.

Default value is FFTW_ESTIMATE

Accepted entries are :

- FFTW_MEASURE or 0
- FFTW_DESTROY_INPUT or 1
- FFTW_UNALIGNED or 2
- FFTW_CONSERVE_MEMORY or 4
- FFTW_EXHAUSTIVE or 8
- FFTW_PRESERVE_INPUT or 16
- FFTW_PATIENT or 32
- FFTW_ESTIMATE or 64
- FFTW_ESTIMATE_PATIENT or 128
- FFTW_BELIEVE_PCOST or 256
- FFTW_NO_DFT_R2HC or 512
- FFTW_NO_NONTHREADED or 1024
- FFTW_NO_BUFFERING or 2048
- FFTW_NO_INDIRECT_OP or 4096
- FFTW_ALLOW_LARGE_GENERIC or 8192
- FFTW_NO_RANK_SPLITS or 16384
- FFTW_NO_VRANK_SPLITS or 32768
- FFTW_NO_VRECURSE or 65536

- FFTW_NO_SIMD or 131072
- FFTW_NO_SLOW or 262144
- FFTW_NO_FIXED_RADIX_LARGE_N or 524288
- FFTW_ALLOW_PRUNING or 1048576

Rmk : when using FFTW_MEASURE/FFTW_PATIENT/FFTW_EXHAUSTIVE you must call two times fftw. (first call for initialisation, second and others calls for computation)

# Examples

```
//return the integer value of the flag
fftw_flags()

//change flags
fftw_flags(["FFTW_MEASURE";"FFTW_CONSERVE_MEMORY"]);

//change flags and display current value of fftw flags (both integer and string
[a,S]=fftw_flags("FFTW_PATIENT")
```

# See Also

fftw

# Name

fftw_forget_wisdom — Reset fftw wisdom

```
fftw_forget_wisdom()
```

# Description

This function reset the current fftw wisdom.

# Examples

```
//return fftw wisdom
txt=get_fftw_wisdom();
//set fftw wisdom
set_fftw_wisdom(txt);
//reset fftw wisdom
fftw_forget_wisdom();
```

# See Also

fftw , get_fftw_wisdom , set_fftw_wisdom

# Name

get_fftw_wisdom — return fftw wisdom

```
[txt]=get_fftw_wisdom()
```

# Parameters

txt
: String matrix that contains fftw wisdom.

# Description

This function return the fftw wisdom in a string matrix.

# Examples

```
//return fftw wisdom
txt=get_fftw_wisdom();
//set fftw wisdom
set_fftw_wisdom(txt);
//reset fftw wisdom
fftw_forget_wisdom();
```

# See Also

fftw , set_fftw_wisdom , fftw_forget_wisdom

# Name

set_fftw_wisdom — set fftw wisdom

```
set_fftw_wisdom(txt)
```

# Parameters

txt
: String matrix that contains fftw wisdom.

# Description

This function set the fftw wisdom with a string matrix.

# Examples

```
//return fftw wisdom
txt=get_fftw_wisdom();
//set fftw wisdom
set_fftw_wisdom(txt);
//reset fftw wisdom
fftw_forget_wisdom();
```

# See Also

fftw , get_fftw_wisdom , fftw_forget_wisdom

# Parte XLIX. Interfaces com UMFPACK

# Name

PlotSparse — plot the pattern of non nul elements of a sparse matrix

```
PlotSparse(A [,style])
```

# Parameters

A

a sparse matrix

style

(optional) a string given the color and/or the marker type of the form "[color][mark]" where color may be a number referring the color you want to use (in the current colormap). If you use the std colormap then color may be one of the following letters :

```
k  for black       b  for blue
r  for red         g  for green
c  for cyan        m  for magenta
y  for yellow      t  for turquoise
G  a dark green
```

mark must be one of the following :

```
.  point           +  plus
x  cross           *  circled plus
D  filled diamond  d  diamond
^  upper triangle  v  down triangle
o  circle
```

by default you have "b." (in fact the 2d color) and this is also forced in case of error.

# Description

plot the pattern of non nul elements of a sparse matrix : each non nul element is drawn with a marker. For "big" matrix use essentially the point . as marker

# Examples

```
[A,description,ref,mtype] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.
set figure_style old
PlotSparse(A,"y+")
xtitle(ref + "." + mtype + " : " + description)
```

# See Also

ReadHBSparse

# Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

# Name

ReadHBSparse — read a Harwell-Boeing sparse format file

```
[A, description, ref, mtype] = ReadHBSparse([filename])
```

# Parameters

filename
: (optional) a string given the filename (eventually preceeding by the path), if filename is not given then the function use uigetfile to get filename interactively

A
: the sparse matrix

description
: a string given some information about the matrix

ref
: a string given the reference of the matrix

mtype
: a string given the type of the matrix

# Description

An utility to read the Harwell-Boeing sparse matrix format. Currently don't work for unassembled matrix. Also the eventual rhs vectors of the file are not returned. Generally the file name is of the form ref.mtype where mtype is a 3 letters word abc given some information (already inside the file) on the matrix :

```
a = R|C|P   for real|complex|pattern (no values given)
b = S|H|Z|U for symetric|hermitian|skew symetric|unsymetric
c = A|E     for assembled|unassembled matrix
            (case E is not treated by this func)
```

# References

Users' Guide for the Harwell-Boeing Sparse Matrix Collection Iain S. Duff, Roger G. Grimes, John G. Lewis. You may found this guide and numerous sparse matrices (in the Harwell-Boeing format) at the University of Florida Sparse Matrix Collection

web site : http://www.cise.ufl.edu/research/sparse/matrices/

maintained by Tim Davis (http://www.cise.ufl.edu/~davis/)

# Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.rua");
```

# See Also

PlotSparse

# Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

# Name

cond2sp — computes an approximation of the 2-norm condition number of a s.p.d. sparse matrix

```
[K2, lm, vm, lM, vM] = cond2sp(A, C_ptr [, rtol, itermax, verb])
```

# Parameters

A
: a real symetric positive definite sparse matrix

C_ptr
: a pointer to a Cholesky factorization (got with taucs_chfact)

rtol
: (optional) relative tolerance (default 1.e-3) (see details in DESCRIPTION)

itermax
: (optional) maximum number of iterations in the underlying algorithms (default 30)

verb
: (optional) boolean, must be %t for displaying the intermediary results, and %f (default) if you don't want.

K2
: estimated 2-norm condition number `K2 = ||A||_2 ||A^(-1)||_2 = lM/lm`

lm
: (real positive scalar) minimum eigenvalue

vm
: associated eigenvector

lM
: (real positive scalar) maximum eigenvalue

vM
: associated eigenvector

# Description

This quick and dirty function computes `(lM,vM)` using the iterative power method and `(lm,vm)` with the inverse iterative power method, then `K2 = lM/lm`. For each method the iterations are stopped until the following condition is met :

```
| (l_new - l_old)/l_new | < rtol
```

but 4 iterations are nevertheless required and also the iterations are stopped if itermax is reached (and a warning message is issued). As the matrix is symetric this is the rayleigh quotient which gives the estimated eigenvalue at each step (`lambda = v'*A*v`). You may called this function with named parameter, for instance if you want to see the intermediary result without setting yourself the rtol and itermax parameters you may called this function with the syntax :

```
[K2, lm, vm, lM, vM] = cond2sp(A , C_ptr, verb=%t )
```

# Caution

Currently there is no verification for the input parameters !

# Remark

This function is intended to get an approximation of the 2-norm condition number (K2) and with the methods used, the precision on the obtained eigenvectors (vM and vm) are generally not very good. If you look for a smaller residual `||Av - l*v||`, you may apply some inverse power iterations from v0 with the matrix :

```
B = A - l0*speye(A)
```

For instance, applied 5 such iterations for `(lm,vm)` is done with :

```
l0 = lm ; v0 = vm;  // or l0 = lM ; v0 = vM;  // to polish (lM,vM)
B = A - l0*speye(A);
LUp = umf_lufact(B);
vr = v0; nstep = 5;
for i=1:nstep, vr = umf_lusolve(LUp, vr, "Ax=b", B); vr = vr/norm(vr) ; end
umf_ludel(LUp); // if you don't use anymore this factorization
lr = vr'*A*vr;
norm_r0 = norm(A*v0 - l0*v0);
norm_rr = norm(A*vr - lr*vr);
// Bauer-Fike error bound...
mprintf(" first estimated eigenvalue : l0 = %e \n\t", l0)
mprintf(" |l-l0| <= ||Av0-l0v0|| = %e , |l-l0|/l0 <= %e \n\r", norm_r0, norm_r0
mprintf(" raffined estimated eigenvalue : lr = %e \n\t", lr)
mprintf(" |l-lr| <= ||Avr-lrvr|| = %e , |l-lr|/lr <= %e \n\r", norm_rr, norm_rr
```

# Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
C_ptr = taucs_chfact(A);
[K2, lm, vm, lM, vM] = cond2sp(A , C_ptr, 1.e-5, 50, %t );
taucs_chdel(C_ptr)
```

# See Also

condestsp , taucs_chfact , rcond

# Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

# Name

condestsp — estimate the condition number of a sparse matrix

```
[K1] = condestsp(A, LUp, t)
[K1] = condestsp(A, LUp)
[K1] = condestsp(A, t)
[K1] = condestsp(A)
```

# Parameters

A
: a real or complex square sparse matrix

LUp
: (optional) a pointer to (umf) LU factors of A obtained by a call to umf_lufact ; if you have already computed the LU (= PAQ) factors it is recommanded to give this optional parameter (as the factorization may be time consuming)

t
: (optional) a positive integer (default value 2) by increasing this one you may hope to get a better (even exact) estimate

K1
: estimated 1-norm condition number of A

# Description

Give an estimate of the 1-norm condition number of the sparse matrix A by Algorithm 2.4 appearing in :

```
"A block algorithm for matrix 1-norm estimation
 with an application to 1-norm pseudospectra"
 Nicholas J. Higham and Francoise Tisseur
 Siam J. Matrix Anal. Appl., vol 21, No 4, pp 1185-1201
```

Noting the exact condition number `K1e = ||A||_1 ||A^(-1)||_1`, we have allways `K1 <= K1e` and this estimate gives in most case something superior to `1/2 K1e`

# Examples

```
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
K1 = condestsp(A)
// verif by direct computation
K1e = norm(A,1)*norm(inv(full(A)),1)

// another example
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.rua");
K1 = condestsp(A)
// this example is not so big so that we can do the verif
```

```
K1e = norm(A,1)*norm(inv(full(A)),1)

// if you have already the lu factors condestsp(A,Lup) is faster
// because lu factors are then not computed inside condestsp
Lup = umf_lufact(A);
K1 = condestsp(A,Lup)
umf_ludel(Lup)         // clear memory
```

# See Also

umf_lufact , rcond

# Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

# Name

rafiter — (obsolete) iterative refinement for a s.p.d. linear system

```
[xn, rn] = rafiter(A, C_ptr, b, x0, [, nb_iter, verb])
```

# Parameters

A
: a real symetric positive definite sparse matrix

C_ptr
: a pointer to a Cholesky factorization (got with taucs_chfact)

b
: column vector (r.h.s of the linear system) but "matrix" (multiple r.h.s.) are allowed.

x0
: first solution obtained with taucs_chsolve(C_ptr, b)

nb_iter
: (optional) number of raffinement iterations (default 2)

verb
: (optional) boolean, must be %t for displaying the intermediary results, and %f (default) if you don't want.

xn
: new refined solution

rn
: residual (`A*xn - b`)

# Description

This function is somewhat obsolete, use `x = taucs_chsolve(C_ptr,b,A)` (see taucs_chsolve) which do one iterative refinement step.

To use if you want to improve a little the solution got with taucs_chsolve. Note that with verb=%t the displayed internal steps are essentially meaningful in the case where b is a column vector.

# Caution

Currently there is no verification for the input parameters !

# Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
C_ptr = taucs_chfact(A);
b = rand(size(A,1),1);
x0 = taucs_chsolve(C_ptr, b);
norm(A*x0 - b)
[xn, rn] = rafiter(A, C_ptr, b, x0, verb=%t);
norm(A*xn - b)
taucs_chdel(C_ptr)
```

# See Also

taucs_chsolve , taucs_chfact

# Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

# Name

res_with_prec — computes the residual r = Ax-b with precision

```
[r,norm2_r] = res_with_prec(A, x, b)
```

# Parameters

A
: real or complex sparse matrix (m x n)

x
: column vector (n x 1) or matrix (n x p)

b
: column vector (m x 1) or matrix (m x p)

r
: column vector (m x 1) or matrix (m x p)

norm2_r
: scalar or vector (1 x p) when b is a m x p matrix

# Description

This function computes the residual of a linear system `r = Ax - b` (together with its 2-norm) with the additionnal precision provided on "Intel like" FPU (80 bits in place of 64) if the compiler translate "long double" to use it. Else one must get the same than using `A*x - b` at the scilab level. In both cases using `[r, nr] = res_with_prec(A,x,b)` is faster than `r = A*x - b` (and faster than `r = A*x - b; nr = norm(r)`).

When `p > 1`, `norm2_r(i)` is the 2-norm of the vector `r(:,i)`.

# Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
C_ptr = taucs_chfact(A);
b = rand(size(A,1),1);
x0 = taucs_chsolve(C_ptr, b);
norm(A*x0 - b)
norm(res_with_prec(A, x0, b))
```

# See Also

rafiter

# Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

# Name

taucs_chdel — utility function used with taucs_chfact

```
taucs_chdel(C_ptr) or taucs_chdel()
```

# Parameters

C_ptr
: a pointer to a Cholesky factorization

# Description

This function is used in conjunction with taucs_chfact and taucs_chsolve. It clears the internal memory space used to store the Cholesky factorization (got with taucs_chfact). Use without argument it frees the memory for all the current scilab (taucs) Cholesky factorizations.

# Examples

see the example section of taucs_chfact

# See Also

taucs_chfact , taucs_chsolve , taucs_chinfo , taucs_chget

# Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

# Name

taucs_chfact — cholesky factorisation of a sparse s.p.d. matrix

```
C_ptr = taucs_chfact(A)
```

# Parameters

A
: a sparse real symetric positive definite (s.p.d.) matrix

C_ptr
: a pointer to the Cholesky factors (C,p : A(p,p)=CC')

# Description

This function computes a Cholesky factorization of the sparse symetric positive definite (s.p.d.) matrix A and retrieves at the scilab level, a pointer (C_ptr) to an handle of the Cholesky factors (C,p) (the memory used for them is "outside" scilab space).

If your matrix is s.p.d. this function must be used in place of umf_lufact or in place of the scilab function chfact for a gain in speed (also as chfact uses the scilab memory for the factors the user must set the stacksize with a large value because of the fill-in occuring in computing the factor C which then may take more memory than the initial matrix A).

When such a factorisation have been computed, a linear system must be solved with taucs_chsolve. **To free the memory used by the Cholesky factors, use taucs_chdel(C_ptr)**; to retrieve the Cholesky factors at the scilab level (for example to display their sparse patterns), use taucs_chget; to get some information (number of non zeros in C), use taucs_chinfo. To compute an approximation of the condition number in norm 2 use cond2sp.

# Remarks

- taucs_chfact works only with the upper triangle of A, and the matrix A must be provided either in its complete form (that is with the lower triangle also) or only with its upper triangle;
- currently taucs_chfact uses the genmmd (generalized minimum degree) algorithm of Liu to find in a first step the permutation p (so as to minimize the fill-in in the factorization); future versions will let the user choose his/her own reordering by providing a supplementary argument p.

# Examples

```
// Example #1 : a small linear test system
// whom solution must be [1;2;3;4;5]
A = sparse( [ 2 -1  0  0  0;
             -1  2 -1  0  0;
              0 -1  2 -1  0;
              0  0 -1  2 -1;
              0  0  0 -1  2] );
b = [0 ; 0; 0; 0; 6];
Cp = taucs_chfact(A);
x = taucs_chsolve(Cp,b)
// don't forget to clear memory with
taucs_chdel(Cp)

// Example #2 a real example
```

```
// first load a sparse matrix
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
// compute the factorisation
Cp = taucs_chfact(A);
b = rand(size(A,1),1); // a random rhs
// use taucs_chsolve for solving Ax=b
x = taucs_chsolve(Cp,b);
norm(A*x - b)
// the same with one iterative refinement step
x = taucs_chsolve(Cp,b,A);
norm(A*x - b)
// don't forget to clear memory
taucs_chdel(Cp)
```

# See Also

taucs_chsolve , taucs_chdel , taucs_chinfo , taucs_chget , cond2sp

# Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

# Name

taucs_chget — retrieve the Cholesky factorization at the scilab level

```
[Ct,p] = taucs_chget(C_ptr)
```

# Parameters

C_ptr
: a pointer to the Cholesky factorization (C,p : A(p,p)=CC')

Ct
: a scilab sparse matrix (you get the upper triangle i.e. Ct is equal to C')

p
: column vector storing the permutation

# Description

This function may be used if you want to plot the sparse pattern of the Cholesky factorization (or if you code something which use the factors). Traditionnaly, the factorization is written :

```
P A P' = C C'
```

with P' the permutation matrix associated to the permutation p. As we get the upper triangle Ct (= C'), in scilab syntax we can write :

```
A(p,p) = Ct' * Ct
```

# Examples

```
// Example #1 : a small linear test system
A = sparse( [ 2 -1  0  0  0;
             -1  2 -1  0  0;
              0 -1  2 -1  0;
              0  0 -1  2 -1;
              0  0  0 -1  2] );
Cp = taucs_chfact(A);
[Ct, p] = taucs_chget(Cp);
full(A(p,p) - Ct'*Ct)  // this must be near the null matrix
taucs_chdel(Cp)

// Example #2 a real example
stacksize(3000000)  // the last PlotSparse need memory
// first load a sparse matrix
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
// compute the factorisation
Cptr = taucs_chfact(A);
// retrieve the factor at scilab level
[Ct, p] = taucs_chget(Cptr);
// plot the initial matrix
xset("window",0) ; clf()
```

```
PlotSparse(A) ; xtitle("Initial matrix A (bcsstk24.rsa)")
// plot the permuted matrix
B = A(p,p);
xset("window",1) ; clf()
PlotSparse(B) ; xtitle("Permuted matrix B = A(p,p)")
// plot the upper triangle Ct
xset("window",2) ; clf()
PlotSparse(Ct) ; xtitle("The pattern of Ct (A(p,p) = C*Ct)")
// retrieve cnz
[OK, n, cnz] = taucs_chinfo(Cptr)
// cnz is superior to the realnumber of non zeros elements of C :
cnz_exact = nnz(Ct)
// don't forget to clear memory
taucs_chdel(Cptr)
```

# See Also

taucs_chfact , taucs_chsolve , taucs_chdel , taucs_chinfo , taucs_chget , cond2sp

# Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

# Name

taucs_chinfo — get information on Cholesky factors

```
[OK, n, cnz] = taucs_chinfo(C_ptr)
```

# Parameters

C_ptr
: a pointer to a Cholesky factorization

OK
: a scalar boolean

n
: a scalar integer

cnz
: a scalar integer

# Description

This function may be used to know basic information about the Cholesky factor created with taucs_chfact :

- first `OK` is `%t` if `C_ptr` is a valid pointer to an Cholesky factorization (and `%f` else)
- if `OK` is `%t` then `n` and `cnz` are respectively the matrix order and the number of non zeros elements in the supernodal structure storing `C` ; if `OK` is `%f`, `n` and `cnz` are set to the void matrix [].

# Details

Due to the supernodal structure used for `C`, `cnz` is larger than the exact number of non-zeros elements in `C` (and so this `cnz` is a mesure of the memory used internally). To get the exact `cnz` you may retrieve the Cholesky factor with taucs_chget then apply the nnz scilab function (see the 2d example in taucs_chget).

# See Also

taucs_chfact , taucs_chsolve , taucs_chdel , taucs_chget

# Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

# Name

taucs_chsolve — solve a linear sparse (s.p.d.) system given the Cholesky factors

```
[x] = taucs_chsolve(C_ptr, b [, A])
```

# Parameters

C_ptr
: a pointer to a handle of the Cholesky factors (C,p with A(p,p)=CC')

b
: a real column vector or a matrix (multiple rhs)

x
: a real column vector or a matrix in case of multiple rhs ( x(:,i) is solution of A x(:,i) = b(:,i))

A
: (optional) the real s.p.d. matrix A (to use for iterative refinement step)

# Description

This function must be used in conjonction with taucs_chfact which computes the Cholesky factorization of a sparse real s.p.d. matrix. When the matrix `A` is provided, one iterative refinement step is done (the refined solution is accepted if it improves the 2-norm of the residual `Ax-b`).

Like in taucs_chfact the matrix A may be provided either in its complete form (that is with the lower triangle also) or only with its upper triangle.

# Examples

see the example section of taucs_chfact

# See Also

taucs_chfact , taucs_chdel , taucs_chinfo , taucs_chget , cond2sp

# Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

# Name

taucs_license — display the taucs license

# Copyright

TAUCS Version 1.0, November 29, 2001. Copyright (c) 2001 by Sivan Toledo, Tel-Aviv Univesity, stoledo@tau.ac.il. All Rights Reserved.

# TAUCS License

Your use or distribution of TAUCS or any derivative code implies that you agree to this License. THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY EXPRESSED OR IMPLIED. ANY USE IS AT YOUR OWN RISK. Permission is hereby granted to use or copy this program, provided that the Copyright, this License, and the Availability of the original version is retained on all copies. User documentation of any code that uses this code or any derivative code must cite the Copyright, this License, the Availability note, and "Used by permission." If this code or any derivative code is accessible from within MATLAB, then typing "help taucs" must cite the Copyright, and "type taucs" must also cite this License and the Availability note. Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. This software is provided to you free of charge.

# Availability

http://www.tau.ac.il/~stoledo/taucs/

# Name

umf_license — display the umfpack license

# Copyright

UMFPACK, Copyright (c) 1995-2006 by Timothy A. Davis. All Rights Reserved. UMFPACK is available under alternate licences; contact T. Davis for details.

# UMFPACK License

Your use or distribution of UMFPACK or any modified version of UMFPACK implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies. User documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included.

# Availability

http://www.cise.ufl.edu/research/sparse

# Name

umf_ludel — utility function used with umf_lufact

```
umf_ludel(LU_ptr) or umf_ludel()
```

# Parameters

LU_ptr
: a pointer to an handle of umf lu factors (L,U,p,q,R)

# Description

This function must be used in conjunction with umf_lufact and umf_lusolve. It clears the internal memory space used to store the LU factors (got with umf_lufact). Use without argument it frees the memory for all the current scilab umfpack LU factors.

# Examples

see the example section of umf_lufact

# See Also

umfpack , umf_lufact , umf_lusolve , umf_luget , umf_luinfo

# Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon

# Name

umf_lufact — lu factorisation of a sparse matrix

```
LU_ptr = umf_lufact(A)
```

# Parameters

A
: a sparse, real or complex, square or rectangular, matrix

LU_ptr
: a pointer to umf lu factors (L,U,p,q,R)

# Description

This function computes a LU factorisation of the sparse matrix A () and return at the scilab level, a pointer (LU_ptr) to an handle of the LU factors (L,U,p,q,R) (the memory used for them is "outside" scilab stack).

This function must be used in place of umfpack if you have multiple linear systems with the same matrix to solve when the rhs are not known at the same time (for instance A x1 = b1 and A x2 = b2 but b2 depends on x1, etc...).

When such a factorisation have been computed, a linear system must be solved with umf_lusolve (in general x = umf_lusolve(LU_ptr, b) but others options are possible, see umf_lusolve. **To free the memory used by the LU factors, use umf_ludel(LU_ptr) (umf_ludel);** to retrieve the LU factors at the scilab level (for example to display their sparse patterns), use umf_luget; to get some information (number of non zeros in L and U), use umf_luinfo. To compute an approximation of the condition number use condestsp

# Examples

```
// this is the small linear test system from UMFPACK
// whom solution must be [1;2;3;4;5]
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
b = [8 ; 45; -3; 3; 19];
Lup = umf_lufact(A);
x = umf_lusolve(Lup,b)

// solve now A'x=b
x = umf_lusolve(Lup,b,"A''x=b")
norm(A'*x - b)

// don't forget to clear memory with
umf_ludel(Lup)

// a real (but small)  example
// first load a sparse matrix
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.rua");
// compute the factorisation
Lup = umf_lufact(A);
```

```
b = rand(size(A,1),1); // a random rhs
// use umf_lusolve for solving Ax=b
x = umf_lusolve(Lup,b);
norm(A*x - b)

// now the same thing with iterative refiment
x = umf_lusolve(Lup,b,"Ax=b",A);
norm(A*x - b)

// solve now the system A'x=b
x = umf_lusolve(Lup,b,"A''x=b");  // without refinement
norm(A'*x - b)
x = umf_lusolve(Lup,b,"A''x=b",A);  // with refinement
norm(A'*x - b)

// don't forget to clear memory
umf_ludel(Lup)
```

# See Also

umfpack , umf_luget , umf_lusolve , umf_ludel , umf_luinfo , condestsp

# Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon with contributions from Antonio Frasson

# Name

umf_luget — retrieve lu factors at the scilab level

```
[L,U,p,q,Rd] = umf_luget(LU_ptr)
```

# Parameters

LU_ptr
: a pointer to umf lu factors (L,U,p,q,R)

L,U
: scilab sparse matrix

p,q
: column vectors storing the permutations

Rd
: vector storing the (row) scaling factors

# Description

This function may be used if you want to plot the sparse pattern of the lu factors (or if you code something which use the lu factors). The factorization provided by umfpack is of the form:

P R^(-1) A Q = LU

where P and Q are permutation matrices, R is a diagonal matrix (row scaling), L a lower triangular matrix with a diagonal of 1, and U an upper triangular matrix. The function provides the matrices L and U as Sparse scilab matrices but P and Q are given as permutation vectors p and q (in fact p is the permutation associated to P') and Rd is the vector corresponding to the diagonal of R.

# Examples

```
// this is the test matrix from UMFPACK
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
Lup = umf_lufact(A);
[L,U,p,q,R] = umf_luget(Lup);
B = A;
for i=1:5, B(i,:) = B(i,:)/R(i); end // apply the row scaling
B(p,q) - L*U  // must be a (quasi) nul matrix

umf_ludel(Lup) // clear memory

// the same with a complex matrix
A = sparse( [ 2+%i  3+2*%i  0      0    0;
              3-%i  0       4+%i   0    6-3*%i;
              0    -1+%i   -3+6*%i 2-%i 0;
              0     0       1-5*%i 0    0;
              0     4       2-%i   0    1] );
Lup = umf_lufact(A);
[L,U,p,q,R] = umf_luget(Lup);
```

```
B = A;
for i=1:5, B(i,:) = B(i,:)/R(i); end // apply the row scaling
B(p,q) - L*U  // must be a (quasi) nul matrix

umf_ludel(Lup) // clear memory
```

# See Also

umfpack , umf_lufact , umf_lusolve , umf_ludel , umf_luinfo

# Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon

# Name

umf_luinfo — get information on LU factors

```
[OK, nrow, ncol, lnz, unz, udiag_nz, it] = umf_luinfo(LU_ptr)
```

# Parameters

LU_ptr
: a pointer to umf lu factors (L,U,p,q, R)

OK
: a scalar boolean

nrow, ncol, lnz, unz, udiag_nz, it
: scalars (integers)

# Description

This function may be used to know basic information about LU factors created with umf_lufact :

first `OK` is %t if `LU_ptr` is a valid pointer to an umfpack LU numeric handle (and %f else)

if `OK` is %t then:

nrow, ncol
: are the matrix size (L is `nrow x n` and U is `n x ncol` where `n = min(nrow,ncol)`

lnz, unz
: are the number of non zeros elements in L and in U;

udiag_nz
: are the number of non zeros elements on the diagonal of U; if the matrix is square (`nrow = ncol = n`) then it is not inversible if udiag_nz < n (more precisely it appears to be numericaly not inversible through the LU factorization).

it
: 0 if the factors are real and 1 if they are complex.

if OK is %f then all the others outputs are set to the empty matrix [].

# Examples

```
// this is the test matrix from UMFPACK
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
Lup = umf_lufact(A);
[OK, nrow, ncol, lnz, unz, udiag_nz, it] = umf_luinfo(Lup)  // OK must be %t, n
[L,U,p,q,R] = umf_luget(Lup);
nnz(L)  // must be equal to lnz
nnz(U)  // must be equal to unz
umf_ludel(Lup) // clear memory
```

# See Also

umfpack , umf_lufact , umf_lusolve , umf_ludel , umf_luget

# Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon

# Name

umf_lusolve — solve a linear sparse system given the LU factors

```
[x] = umf_lusolve(LU_ptr, b [, st, A])
```

# Parameters

LU_ptr
: a pointer to umf lu factors (L,U,p,q,R)

b
: a real or complex column vector or a matrix (multiple rhs)

st
: (optional) a string "Ax=b" (default) or "Ax'=b" (to be written "Ax"=b" in scilab langage: a quote in a string must be doubled !)

A
: (optional) the sparse square matrix corresponding to the LU factors (LU_ptr must be got with LU_ptr = umf_lufact(A))

x
: a column vector or a matrix in case of multiple rhs ( x(:,i) is solution of A x(:,i) = b(:,i) or A'x(:,i) = b(:,i) )

# Description

This function must be used in conjonction with umf_lufact which computes the LU factors of a sparse matrix. The optional `st` argument lets us choose between the solving of Ax=b (general case) or of A'x=b (sometimes useful). If you give the 4th argument then iterative refinement will be also proceceed (as in umfpack) to give a better numerical solution.

# Examples

see the example section of umf_lufact

# See Also

umfpack , umf_lufact , umf_luget , umf_ludel , umf_luinfo

# Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon with contributions from Antonio Frasson

# Name

umfpack — solve sparse linear system

```
x = umfpack(A,"\",b)
x = umfpack(b,"/",A)
```

# Parameters

A
: a sparse (real or complex) square matrix n x n

b
: in the first case, a column vector (n x 1) or a n x m matrix ; in the second case, a row vector (1 x n) or a m x n matrix

x
: in the first case , a column vector (n x 1) or a n x m matrix ; in the second case, a row vector (1 x n) or a m x n matrix

2d arg
: string specifier "\" or "/"

# Description

This function is intended to work like the classic operators \ and / x = A\b and x = b/A) i.e. it solves a linear system Ax = b or xA = b with a sparse square (says n x n) real or complex matrix and with a compatible rhs b : n x m in the first case and m x n in the second.

# Details

First an LU factorisation of the matrix is computed ( `P R^(-1) A Q = LU` where P and Q are permutation matrices, R is a diagonal matrix (row scaling), L a lower triangular matrix with a diagonal of 1, and U an upper triangular matrix) then a first solution is computed with forward/backward subtitutions ; finaly the solution is improved by iterative refinement.

# Examples

```
// this is the small linear test system from UMFPACK
// whom solution must be [1;2;3;4;5]
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
b = [8 ; 45; -3; 3; 19];
x = umfpack(A,"\",b)

// test the other form x A = b
b = [8  20  13  6  17];
x = umfpack(b,"/",A)   // solution must be [1 2 3 4 5]

// test multiple rhs
b = rand(5,3);
x = umfpack(A,"\",b)
norm(A*x - b)
```

```
// test multiple rhs for x A = b
b = rand(3,5);
x = umfpack(b,"/",A)
norm(x*A - b)

// solve a complex system
A = sparse( [ 2+%i  3+2*%i  0      0    0;
              3-%i  0       4+%i   0    6-3*%i;
              0    -1+%i   -3+6*%i 2-%i 0;
              0     0       1-5*%i 0    0;
              0     4       2-%i   0    1] );
b = [ 3+13*%i ; 58+32*%i ; -19+13*%i ; 18-12*%i ; 22+16*%i ];
x = umfpack(A,"\",b)  // x must be [1+i; 2+2i; 3+3i; 4 + 4i; 5+5i]

// A benchmark of several linear solvers

[A,descr,ref,mtype] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa"

b = 0*ones(size(A,1),1);

tic();
res = umfpack(A,'\',b);
printf('\ntime needed to solve the system with umfpack: %.3f\n',toc());

tic();
res = linsolve(A,b);
printf('\ntime needed to solve the system with linsolve: %.3f\n',toc());

tic();
res = A\b;
printf('\ntime needed to solve the system with the backslash operator: %.3f\n',
```

# See Also

umf_lufact , umf_lusolve , umf_ludel , umf_luinfo , umf_luget , linsolve , backslash

# Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon with contributions from Antonio Frasson

# Parte L. Algoritmos Genéticos

# Name

coding_ga_binary — A function which performs conversion between binary and continuous representation

```
pop_out = coding_ga_binary(pop_in,direction,param)
```

# Parameters

pop_in
: a list which contains all the individuals in the current population.

direction
: 'code' or 'decode'. If direction == 'code' then we perform a continuous to binary encoding. Else, we convert from binary to continuous.

param
: a parameter list.

- 'binary_length': the number of bits by variables. If binary_length = 8 and the variable X is of dimension 2 then the binary code will be 16 bits length.
- 'minboun': a vector of minimum bounds for the variable X.
- 'maxbound': a vector of maximum bounds for the variable X.

pop_out
: the population coded to binary or decoded to continuous values.

# Description

- This function allows to code or decode a population of individuals from (resp. to) continuous variables to (resp. from) binary.

# See Also

optim_ga , mutation_ga_binary , crossover_ga_binary

# Authors

Yann COLLETTE
: ycollet@freesurf.fr

# Name

coding_ga_identity — A "no-operation" conversion function

```
pop_out = coding_ga_identity(pop_in,direction,param)
```

# Parameters

pop_in
: the population to be converted.

direction
: 'code' or 'decode'. This value has no influence of the state of pop_in.

param
: a parameter list. For this function, there are no useful parameters set.

pop_out
: a population identical to pop_in.

# Description

- This function is a do-nothing function. It is essentially useful to implement an evolutionnary algorithm. In an evolutionnary algorithm, we work directly on the variable and not on a binary code.

# See Also

mutation_func_default , crossover_func_default , init_func_default , optim_ga

# Authors

Yann COLLETTE
: ycollet@freesurf.fr

# Name

crossover_ga_binary — A crossover function for binary code

```
[Crossed_Indiv1,Crossed_Indiv2] = crossover_ga_binary(Indiv1,Indiv2,param)
```

# Parameters

Indiv1
: the first individual (here a binary code) to be crossed-over.

Indiv2
: the second individual to be crossed-over.

param
: a list of parameters.

- 'binary_length': the length of the binary code.
- 'multi_cross': a boolean. If %T then we allow several cuts in the binary code.
- 'multi_cross_nb': the number of cuts in the binary code. Only used when multi_cross is set to %T.

Crossed_Indiv1
: The first individual obtained by the cross-over function.

Crossed_Indiv2
: The second individual obtained by the cross-over function.

# Description

- This function implements a classical binary cross-over.

# See Also

crossover_ga_binary , crossover_ga_default , mutation_ga_binary , optim_ga

# Authors

Yann COLLETTE
: ycollet@freesurf.fr

# Name

crossover_ga_default — A crossover function for continuous variable functions

```
[Crossed_Indiv1,Crossed_Indiv2] = crossover_ga_default(Indiv1,Indiv2,param)
```

# Parameters

Indiv1
: The first individual to be crossed-over.

Indiv2
: The second individual to be crossed-over.

param
: a list of parameters.

- 'beta': the range of the random generator. A random value will be sampled between -beta and 1+beta. This sampled value will be used to perform a convex combination between Indiv1 and Indiv2.
- 'minbound': a vector of minimum bounds for the variable X.
- 'maxbound': a vector of maximum bounds for the variable X.

Crossed_Indiv1
: The first individual resulting from the crossover.

Crossed_Indiv2
: The second individual resulting from the crossover.

# Description

crossover_ga_default is a crossover function for functions with continuous variables. This crossover function is an extension of a convexe combination. The crossed individuals are computed with the following equations :

```
mix = (1 + 2*Beta)*rand(1,1) - Beta;
Crossed_Indiv1 =     mix*Indiv1 + (1-mix)*Indiv2;
Crossed_Indiv2 = (1-mix)*Indiv1 +     mix*Indiv2;
```

The Beta parameter should be set to a positive value. If Beta is set to 0, the resulting crossover is a simple convexe combination between the two parents. That may lead to a too fast convergence of the genetic algorithm and may decrease the diversity of the individuals of the population. If Beta is chosen strictly positive, that may allow children to explore the domain beyond the domain explored by their parents.

# See Also

crossover_ga_binary , mutation_ga_default , init_ga_default , optim_ga

# References

Michalewicz, Zbigniew Genetic Algorithms + Data Structures = Evolution Programs

# Authors

Yann COLLETTE
ycollet@freesurf.fr

# Name

init_ga_default — A function a initialize a population

```
Pop_init = init_ga_default(popsize,param)
```

# Parameters

popsize
: the number of individuals to generate.

param
: a list of parameters.

- 'dimension': the size of the vector X.
- 'minbound': a vector of minimum bounds for the variable X.
- 'maxbound': a vector of maximum bounds for the variable X.

Pop_init
: a list which contains the initial population of individuals.

# Description

- This function generate an initial population containing pop_size individuals.

# See Also

crossover_ga_default , mutation_ga_default , mutation_ga_binary , optim_ga

# Authors

Yann COLLETTE
: ycollet@freesurf.fr

# Name

mutation_ga_binary — A function which performs binary mutation

```
Mut_Indiv = mutation_ga_binary(Indiv,param)
```

# Parameters

Indiv
: the individual on which we will perform the mutation.

param
: a list of parameters.

- 'binary_length': the size of the binary code.
- 'multi_mut': a boolean. If %T, several random bits will be flipped.
- 'multi_mut_nd': the number of bits to be flipped. Works only when multi_mut is set to %T.

Mut_Indiv
: The mutated individual.

# Description

- This function performs a classical multi-bits binary mutation.

# See Also

mutation_ga_default , crossover_ga_binary , init_func_default , optim_ga

# Authors

Yann COLLETTE
: ycollet@freesurf.fr

# Name

mutation_ga_default — A continuous variable mutation function

```
Mut_Indiv = mutation_ga_default(Indiv,param)
```

# Parameters

Indiv
: The individual to be mutated.

param
: a list of parameters.

- 'delta': a random perturbation will be sampled via an uniform distribution between -delta and + delta.
- 'minbound': a vector of minimum bound for the variable X.
- 'maxbound': a vector of maximum bound for the variable X.

Mut_Indiv
: The resulting mutated individual.

# Description

- This function performs the classical continuous variable mutation function.

# See Also

mutation_ga_binary , crossover_ga_default , init_ga_default , optim_ga

# Authors

Yann COLLETTE
: ycollet@freesurf.fr

# Name

optim_ga — A flexible genetic algorithm

```
[pop_opt,fobj_pop_opt,pop_init,fobj_pop_init] = optim_ga(ga_f,pop_size,nb_genera
```

# Parameters

ga_f
: the function to be optimized. The prototype if y = f(x) or y = list(f,p1,p2,...).

pop_size
: the size of the population of individuals (default value: 100).

nb_generation
: the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut
: the mutation probability (default value: 0.1).

p_cross
: the crossover probability (default value: 0.7).

Log
: if %T, we will display to information message during the run of the genetic algorithm.

param
: a list of parameters.

- 'codage_func': the function which will perform the coding and decoding of individuals (default function: codage_identity).
- 'init_func': the function which will perform the initialization of the population (default function: init_ga_default).
- 'crossover_func': the function which will perform the crossover between two individuals (default function: crossover_ga_default).
- 'mutation_func': the function which will perform the mutation of one individual (default function: mutation_ga_default).
- 'selection_func': the function whcih will perform the selection of individuals at the end of a generation (default function: selection_ga_elitist).
- 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
- 'pressure': the value the efficiency of the worst individual (default value: 0.05).

pop_opt
: the population of optimal individuals.

fobj_pop_opt
: the set of objective function values associated to pop_opt (optional).

pop_init
: the initial population of individuals (optional).

fobj_pop_init
: the set of objective function values associated to pop_init (optional).

# Description

- This function implements the classical genetic algorithm.

# Examples

```
deff('y=f(x)','y = sum(x.^2)');

PopSize     = 100;
Proba_cross = 0.7;
Proba_mut   = 0.1;
NbGen       = 10;
NbCouples   = 110;
Log         = %T;
nb_disp     = 10; // Nb point to display from the optimal population
pressure    = 0.05;

ga_params = init_param();
// Parameters to adapt to the shape of the optimization problem
ga_params = add_param(ga_params,'minbound',[-2; -2]);
ga_params = add_param(ga_params,'maxbound',[2; 2]);
ga_params = add_param(ga_params,'dimension',2);
ga_params = add_param(ga_params,'beta',0);
ga_params = add_param(ga_params,'delta',0.1);
// Parameters to fine tune the Genetic algorithm. All these parameters are opti
// If you need to adapt the GA to a special problem, you
ga_params = add_param(ga_params,'init_func',init_ga_default);
ga_params = add_param(ga_params,'crossover_func',crossover_ga_default);
ga_params = add_param(ga_params,'mutation_func',mutation_ga_default);
ga_params = add_param(ga_params,'codage_func',codage_ga_identity);
ga_params = add_param(ga_params,'selection_func',selection_ga_elitist);
//ga_params = add_param(ga_params,'selection_func',selection_ga_random);
ga_params = add_param(ga_params,'nb_couples',NbCouples);
ga_params = add_param(ga_params,'pressure',pressure);

Min = get_param(ga_params,'minbound');
Max = get_param(ga_params,'maxbound');
x0  = (Max - Min) .* rand(size(Min,1),size(Min,2)) + Min;

[pop_opt, fobj_pop_opt, pop_init, fobj_pop_init] = optim_ga(f, PopSize, NbGen, 
```

# See Also

optim_moga , optim_nsga , optim_nsga2

# References

Michalewicz, Zbigniew Genetic Algorithms + Data Structures = Evolution Programs

# Authors

Yann COLLETTE
ycollet@freesurf.fr

# Name

optim_moga — multi-objective genetic algorithm

```
[pop_opt,fobj_pop_opt,pop_init,fobj_pop_init] = optim_moga(ga_f,pop_size,nb_gen
```

# Parameters

ga_f
: the function to be optimized. The header of the function is the following :

```
y = f(x)
```

or

```
y = list(f,p1,p2,...)
```

pop_size
: the size of the population of individuals (default value: 100).

nb_generation
: the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut
: the mutation probability (default value: 0.1).

p_cross
: the crossover probability (default value: 0.7).

Log
: if %T, we will display to information message during the run of the genetic algorithm.

param
: a list of parameters.

- 'codage_func': the function which will perform the coding and decoding of individuals (default function: codage_identity).
- 'init_func': the function which will perform the initialization of the population (default function: init_ga_default).
- 'crossover_func': the function which will perform the crossover between two individuals (default function: crossover_ga_default).
- 'mutation_func': the function which will perform the mutation of one individual (default function: mutation_ga_default).
- 'selection_func': the function whcih will perform the selection of individuals at the end of a generation (default function: selection_ga_elitist).
- 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
- 'pressure': the value the efficiency of the worst individual (default value: 0.05).

pop_opt
: the population of optimal individuals.

fobj_pop_opt
: the set of multi-objective function values associated to pop_opt (optional).

pop_init
: the initial population of individuals (optional).

fobj_pop_init
: the set of multi-objective function values associated to pop_init (optional).

# Description

- This function implements the classical "Multi-Objective Genetic Algorithm". For a demonstration: see SCI/modules/genetic_algorithms/examples/MOGAdemo.sce.

# See Also

optim_ga , optim_nsga , optim_nsga2

# Authors

Yann COLLETTE
: ycollet@freesurf.fr

# Name

optim_nsga — A multi-objective Niched Sharing Genetic Algorithm

```
[pop_opt,fobj_pop_opt,pop_init,fobj_pop_init] = optim_nsga(ga_f,pop_size,nb_gen
```

# Parameters

ga_f
: the function to be optimized. The prototype if y = f(x) or y = list(f,p1,p2,...).

pop_size
: the size of the population of individuals (default value: 100).

nb_generation
: the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut
: the mutation probability (default value: 0.1).

p_cross
: the crossover probability (default value: 0.7).

Log
: if %T, we will display to information message during the run of the genetic algorithm.

param
: a list of parameters.

- 'codage_func': the function which will perform the coding and decoding of individuals (default function: codage_identity).
- 'init_func': the function which will perform the initialization of the population (default function: init_ga_default).
- 'crossover_func': the function which will perform the crossover between two individuals (default function: crossover_ga_default).
- 'mutation_func': the function which will perform the mutation of one individual (default function: mutation_ga_default).
- 'selection_func': the function whcih will perform the selection of individuals at the end of a generation (default function: selection_ga_elitist).
- 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
- 'pressure': the value the efficiency of the worst individual (default value: 0.05).

sigma
: the radius of the sharing area.

pow
: the power coefficient of the penalty formula.

pop_opt
: the population of optimal individuals.

fobj_pop_opt
: the set of objective function values associated to pop_opt (optional).

pop_init
: the initial population of individuals (optional).

fobj_pop_init
: the set of objective function values associated to pop_init (optional).

# Description

- This function implements the classical "Niched Sharing Genetic Algorithm". For a demonstration, see SCI/modules/genetic_algorithms/examples/NSGAdemo.sce.

# See Also

optim_moga , optim_ga , optim_nsga2

# Authors

Yann COLLETTE
: ycollet@freesurf.fr

# Name

optim_nsga2 — A multi-objective Niched Sharing Genetic Algorithm version 2

```
[pop_opt,fobj_pop_opt,pop_init,fobj_pop_init] = optim_nsga2(ga_f,pop_size,nb_ge
```

# Parameters

ga_f
: the function to be optimized. The prototype if y = f(x) or y = list(f,p1,p2,...).

pop_size
: the size of the population of individuals (default value: 100).

nb_generation
: the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut
: the mutation probability (default value: 0.1).

p_cross
: the crossover probability (default value: 0.7).

Log
: if %T, we will display to information message during the run of the genetic algorithm.

param
: a list of parameters.

- 'codage_func': the function which will perform the coding and decoding of individuals (default function: codage_identity).
- 'init_func': the function which will perform the initialization of the population (default function: init_ga_default).
- 'crossover_func': the function which will perform the crossover between two individuals (default function: crossover_ga_default).
- 'mutation_func': the function which will perform the mutation of one individual (default function: mutation_ga_default).
- 'selection_func': the function whcih will perform the selection of individuals at the end of a generation (default function: selection_ga_elitist).
- 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
- 'pressure': the value the efficiency of the worst individual (default value: 0.05).

pop_opt
: the population of optimal individuals.

fobj_pop_opt
: the set of objective function values associated to pop_opt (optional).

pop_init
: the initial population of individuals (optional).

fobj_pop_init
: the set of objective function values associated to pop_init (optional).

# Description

- This function implements the classical "Niched Sharing Genetic Algorithm". For a demonstration, see SCI/modules/genetic_algorithms/examples/NSGA2demo.sce.

# See Also

optim_moga , optim_ga , optim_nsga

# Authors

Yann COLLETTE
ycollet@freesurf.fr

# Name

pareto_filter — A function which extracts non dominated solution from a set

```
[F_out,X_out,Ind_out] = pareto_filter(F_in,X_in)
```

# Parameters

F_in
: the set of multi-objective function values from which we want to extract the non dominated solutions.

X_in
: the associated values in the parameters space.

F_out
: the set of non dominated multi-objective function values.

X_out
: the associated values in the parameters space.

Ind_out
: the set of indexes of the non dominated individuals selected from the set X_in.

# Description

- This function applies a Pareto filter to extract non dominated solutions from a set of values.

# See Also

optim_moga , optim_nsga , optim_nsga2

# Authors

Yann COLLETTE
: ycollet@freesurf.Fr

# Name

selection_ga_elitist — An 'elitist' selection function

```
[Pop_out,FObj_Pop_out,Efficiency,MO_Total_FObj_out] = selection_ga_elitist(Pop_
```

# Parameters

Pop_in
: The initial population of individuals.

Indiv1
: a first set of childs generated via crossover + mutation.

Indiv2
: a second set of childs generated via crossover + mutation.

FObj_Pop_in
: a vector of objective function values associated to each individuals of Pop_in.

FObj_Indiv1
: a vector of objective function values associated to each individuals of Indiv1.

FObj_Indiv2
: a vector of objective function values associated to each individuals of Indiv2.

MO_Total_FObj_in
: a matrix of multi-objective function values associated to each individuals of Pop_in.

MO_FObj_Indiv1
: a matrix of multi-objective function values associated to each individuals of Indiv1.

MO_FObj_Indiv2
: a matrix of multi-objective function values associated to each individuals of Indiv2.

param
: a list of parameters. - 'pressure': the selection pressure coefficient. Each individuals with 0 efficiency will have an efficiency value equal to 'pressure'.

Pop_out
: all the selected individuals in a population of size pop_size.

FObj_Pop_out
: all the objective function values associated to each individuals of Pop_out.

Efficiency
: all the efficiency values associated to each individuals of Pop_out.

MO_Total_FObj_out
: all the multi-objective function values associated to each individuals of Pop_out.

# Description

- This function performs the elitist selection function. We select the best individuals in the set of parents and childs individuals.

# See Also

selection_ga_random , mutation_ga_default , crossover_ga_default , init_ga_default , optim_ga

# Authors

Yann COLLETTE
ycollet@freesurf.fr

# Name

selection_ga_random — A function which performs a random selection of individuals

```
[Pop_out,FObj_Pop_out,Efficiency,MO_Total_FObj_out] = selection_ga_random(Pop_i
```

# Parameters

Pop_in
: The initial population of individuals.

Indiv1
: a first set of childs generated via crossover + mutation.

Indiv2
: a second set of childs generated via crossover + mutation.

FObj_Pop_in
: a vector of objective function values associated to each individuals of Pop_in.

FObj_Indiv1
: a vector of objective function values associated to each individuals of Indiv1.

FObj_Indiv2
: a vector of objective function values associated to each individuals of Indiv2.

MO_Total_FObj_in
: a matrix of multi-objective function values associated to each individuals of Pop_in.

MO_FObj_Indiv1
: a matrix of multi-objective function values associated to each individuals of Indiv1.

MO_FObj_Indiv2
: a matrix of multi-objective function values associated to each individuals of Indiv2.

param
: a list of parameters.

- 'pressure': the selection pressure coefficient. Each individuals with 0 efficiency will have an efficiency value equal to 'pressure'.

Pop_out
: all the selected individuals in a population of size pop_size.

FObj_Pop_out
: all the objective function values associated to each individuals of Pop_out.

Efficiency
: all the efficiency values associated to each individuals of Pop_out.

MO_Total_FObj_out
: all the multi-objective function values associated to each individuals of Pop_out.

# Description

- This function performs the random selection function. We select pop_size individuals in the set of parents and childs individuals at random.

# See Also

selection_ga_elitist , mutation_ga_default , crossover_ga_default , init_ga_default , optim_ga

# Authors

Yann COLLETTE
ycollet@freesurf.fr

# Parte LI. Arrefecimento Simulado

# Name

compute_initial_temp — A SA function which allows to compute the initial temperature of the simulated annealing

```
T_init = compute_initial_temp(x0,f,proba_init,ItMX,neigh_func,param_neigh_func)
```

# Parameters

x0
: the starting point

f
: the objective function which will be send to the simulated annealing for optimization

proba_init
: the initial probability of accepting a bad solution (usually around 0.7)

ItMX
: the number of iterations of random walk (usually around 100)

neigh_func
: a function which returns a neighbor of a given point (see the help page of neigh_func to see the prototype of this function)

param_neigh_func
: some parameters (can be a list) which will be sent as parameters to neigh_func

T_init
: The initial temperature corresponding to the given probability of accepting a bad solution

# Description

- This function computes an initial temperature given an initial probability of accepting a bad solution. This computation is based on some iterations of random walk.

# Examples

```
deff('y=f(x)','y=sum(x.^2)');

x0 = [2 2];
Proba_start = 0.7;
It_Pre = 100;
x_test = neigh_func_default(x0);

comp_t_params = init_param();
comp_t_params = add_param(comp_t_params,'neigh_func', neigh_func_default);

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, comp_t_sa);
```

# See Also

optim_sa , neigh_func_default , temp_law_default

# Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

# Name

neigh_func_csa — The classical neighborhood relationship for the simulated annealing

```
x_neigh = neigh_func_csa(x_current,T,param)
```

# Parameters

x_current
: the point for which we want to compute a neighbor

T
: the current temperature

param
: a vector with the same size than x_current. A normalisation vector which allows to distort the shape of the neighborhood. This parameter allows to take into account the differences of interval of variation between variables. By default, this parameter is set to a vector of ones.

x_neigh
: the computed neighbor

# Description

- This function implements the classical neighborhood relationship for the simulated annealing. The neighbors distribution is a gaussian distribution which is more and more peaked as the temperature decrease.

# See Also

neigh_func_default , temp_law_huang , optim_sa

# Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

# Name

neigh_func_default — A SA function which computes a neighbor of a given point

```
x_neigh = neigh_func_default(x_current,T,param)
```

# Parameters

x_current
: the point for which we want to compute a neighbor

T
: the current temperature

param
: a two columns vector. The first column correspond to the negative amplitude of variation and the second column corresponds to the positive amplitude of variation of the neighborhood. By default, the first column is a column of -0.1 and the second column is a column of 0.1.

x_neigh
: the computed neighbor

# Description

- This function computes a neighbor of a given point. For example, for a continuous vector, a neighbor will be produced by adding some noise to each component of the vector. For a binary string, a neighbor will be produced by changing one bit from 0 to 1 or from 1 to 0.

# Examples

```
// We produce a neighbor by adding some noise to each component of a given vecto
function x_neigh = neigh_func_default(x_current, T)
  sa_min_delta = -0.1*ones(size(x_current,1),size(x_current,2));
  sa_max_delta = 0.1*ones(size(x_current,1),size(x_current,2));
  x_neigh = x_current + (sa_max_delta - sa_min_delta).*rand(size(x_current,1),s
endfunction
```

# See Also

optim_sa , compute_initial_temp , temp_law_default

# Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

# Name

neigh_func_fsa — The Fast Simulated Annealing neghborhood relationship

```
x_neigh = neigh_func_fsa(x_current,T,param)
```

# Parameters

x_current
: the point for which we want to compute a neighbor

T
: the current temperature

param
: a vector with the same size than x_current. A normalisation vector which allows to distort the shape of the neighborhood. This parameter allows to take into account the differences of interval of variation between variables. By default, this parameter is set to a vector of ones.

x_neigh
: the computed neighbor

# Description

- This function computes the FSA neighborhood of a given point. The corresponding distribution is a Cauchy distribution which is more and more peaked as the temperature decrease.

# See Also

optim_sa , temp_law_fsa , neigh_func_default

# Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

# Name

neigh_func_vfsa — The Very Fast Simulated Annealing neighborhood relationship

```
x_neigh = neigh_func_vfsa(x_current,T,param)
```

# Parameters

x_current
: the point for which we want to compute a neighbor

T
: the current temperature

param
: a ones column vector. The column correspond to the amplitude of variation of the neighborhood. By default, the column is a column of 0.1.

x_neigh
: the computed neighbor

# Description

- This function implements the Very Fast Simulated Annealing relationship. This distribution is more and more peaked as the temperature decrease.

# See Also

optim_sa , neigh_func_vfsa , temp_law_huang

# Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

# Name

optim_sa — A Simulated Annealing optimization method

```
[x_best,f_best,mean_list,var_list,f_history,temp_list,x_history] = optim_sa(x0,
```

# Parameters

x0
: the initial solution

f
: the objective function to be optimized (the prototype if f(x))

ItExt
: the number of temperature decrease

ItInt
: the number of iterations during one temperature stage

T0
: the initial temperature (see compute_initial_temp to compute easily this temperature)

Log
: if %T, some information will be displayed during the run of the simulated annealing

temp_law
: the temperature decrease law (see temp_law_default for an example of such a function)

param_temp_law
: a structure (of any kind - it depends on the temperature law used) which is transmitted as a parameter to temp_law

neigh_func
: a function which computes a neighbor of a given point (see neigh_func_default for an example of such a function)

param_neigh_func
: a structure (of any kind like vector, list, it depends on the neighborhood function used) which is transmitted as a parameter to neigh_func

x_best
: the best solution found so far

f_best
: the objective function value corresponding to x_best

mean_list
: the mean of the objective function value for each temperature stage. A vector of float (optional)

var_list
: the variance of the objective function values for each temperature stage. A vector of float (optional)

f_history
: the computed objective function values for each iteration. Each input of the list corresponds to a temperature stage. Each input of the list is a vector of float which gathers all the objective function values computed during the corresponding temperature stage - (optional)

temp_list
: the list of temperature computed for each temperature stage. A vector of float (optional)

x_history
: the parameter values computed for each iteration. Each input of the list corresponds to a temperature stage. Each input of the list is a vector of input variables which corresponds to all the variables computed during the corresponding temperature stage - (optional - can slow down a lot the execution of optim_sa)

# Description

- A Simulated Annealing optimization method.

# Examples

```
function y = rastrigin(x)
  y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0          = [2 2];
Proba_start = 0.7;
It_Pre      = 100;
It_extern   = 100;
It_intern   = 1000;
x_test = neigh_func_default(x0);

comp_t_params = init_param();
comp_t_params = add_param(comp_t_params,'neigh_func', neigh_func_default);

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, comp_t_sa);

[x_opt, f_opt, sa_mean_list, sa_var_list] = optim_sa(x0, rastrigin, It_extern,

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
```

# See Also

compute_initial_temp , neigh_func_default , temp_law_default

# Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

# Name

temp_law_csa — The classical temperature decrease law

```
T_out = temp_law_csa(T_in,step_mean,step_var,temp_stage,n,param)
```

# Parameters

T_in
: the temperature at the current stage

step_mean
: the mean value of the objective function computed during the current stage

step_var
: the variance value of the objective function computed during the current stage

temp_stage
: the index of the current temperature stage

n
: the dimension of the decision variable (the x in f(x))

param
: not used for this temperature law

T_out
: the temperature for the temperature stage to come

# Description

- This function implements the classical annealing temperature schedule (the one for which the convergence of the simulated annealing has been proven).

# Examples

```
function y = rastrigin(x)
  y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre    = 100;

printf('SA: the CSA algorithm\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_defaul
printf('Initial temperatore T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin,

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
```

```
subplot(2,1,1);
xtitle('Classical simulated annealing','Iteration','Mean / Variance');
t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
legend(['Mean','Variance']);
subplot(2,1,2);
xtitle('Temperature evolution','Iteration','Temperature');
plot(t,temp_list,'k-');
```

# See Also

optim_sa , temp_law_huang , neigh_func_default

# Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

# Name

temp_law_default — A SA function which computed the temperature of the next temperature stage

```
T_next = temp_law_default(T,step_mean,step_var,temp_stage,n,param)
```

# Parameters

T
: the temperature applied during the last temperature stage

step_mean
: the mean of the objective function values computed during the last temperature stage

step_var
: the variance of the obejective function values computed during the last temperature stage

temp_stage
: the index of the current temperature stage

n
: the dimension of the decision variable (the x in f(x))

param
: a float between 0 and 1. Corresponds to the decrease in temperature of the geometric law (0.9 by default)

T_next
: the new temperature to be applied for the next temperature stage

# Description

- A SA function which computed the temperature of the next temperature stage

# Examples

```
// This function implements the simple geometric temperature law
function T = temp_law_default(T, step_mean, step_var)
  _alpha = 0.9;
  T = _alpha*T;
endfunction
```

# See Also

optim_sa , compute_initial_temp , neigh_func_default

# Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

# Name

temp_law_fsa — The Szu and Hartley Fast simulated annealing

```
T_out = temp_law_fsa(T_in,step_mean,step_var,temp_stage,n,param)
```

# Parameters

T_in
: the temperature at the current stage

step_mean
: the mean value of the objective function computed during the current stage

step_var
: the variance value of the objective function computed during the current stage

temp_stage
: the index of the current temperature stage

n
: the dimension of the decision variable (the x in f(x))

param
: not used for this temperature law

T_out
: the temperature for the temperature stage to come

# Description

- This function implements the Fast simulated annealing of Szu and Hartley.

# Examples

```
function y = rastrigin(x)
  y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre    = 100;

printf('SA: the FSA algorithm\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_defaul
printf('Initial temperatore T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin, 

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
subplot(2,1,1);
```

```
xtitle('Fast simulated annealing','Iteration','Mean / Variance');
t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
legend(['Mean','Variance']);
subplot(2,1,2);
xtitle('Temperature evolution','Iteration','Temperature');
plot(t,temp_list,'k-');
```

# See Also

optim_sa , temp_law_huang , neigh_func_default

# Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

# Name

temp_law_huang — The Huang temperature decrease law for the simulated annealing

```
T_out = temp_law_huang(T_in,step_mean,step_var,temp_stage,n,param)
```

# Parameters

T_in
: the temperature at the current stage

step_mean
: the mean value of the objective function computed during the current stage

step_var
: the variance value of the objective function computed during the current stage

temp_stage
: the index of the current temperature stage

n
: the dimension of the decision variable (the x in f(x))

param
: a float corresponding to the lambda parameter of the Huang temperature decrease law (0.01 by default)

T_out
: the temperature for the temperature stage to come

# Description

- This function implements the Huang temperature decrease law for the simulated annealing.

# Examples

```
function y = rastrigin(x)
  y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre    = 100;

printf('SA: the Huang temperature decrease law\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_defaul
printf('Initial temperatore T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin,

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
```

```
subplot(2,1,1);
xtitle('Huang simulated annealing','Iteration','Mean / Variance');
t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
legend(['Mean','Variance']);
subplot(2,1,2);
xtitle('Temperature evolution','Iteration','Temperature');
plot(t,temp_list,'k-');
```

# See Also

optim_sa , temp_law_csa , neigh_func_csa

# Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

# Name

temp_law_vfsa — This function implements the Very Fast Simulated Annealing from L. Ingber

```
T_out = temp_law_vfsa(T_in,step_mean,step_var,temp_stage,n, param)
```

# Parameters

T_in
: the temperature at the current stage

step_mean
: the mean value of the objective function computed during the current stage

step_var
: the variance value of the objective function computed during the current stage

temp_stage
: the index of the current temperature stage

n
: the dimension of the decision variable (the x in f(x))

param
: a float: the 'c' parameter of the VFSA method (0.01 by default)

T_out
: the temperature for the temperature stage to come

# Description

- This function implements the Very Fast Simulated Annealing from L. Ingber.

# Examples

```
function y = rastrigin(x)
  y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre    = 100;

printf('SA: the VFSA algorithm\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_defaul
printf('Initial temperatore T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin, 

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
subplot(2,1,1);
```

```
xtitle('VFSA simulated annealing','Iteration','Mean / Variance');
t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
legend(['Mean','Variance']);
subplot(2,1,2);
xtitle('Temperature evolution','Iteration','Temperature');
plot(t,temp_list,'k-');
```

# See Also

optim_sa , neigh_func_vfsa , temp_law_huang

# Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

# Parte LII. Parâmetros

# Name

add_param — Add a parameter to a list of parameters

```
[param_list,err] = add_param(list_name,param_name,param_value)
```

# Parameters

list_name
: the list of parameters. This list must have been initialize by a call to init_param.

param_name
: a string. The name of the parameter to be added in the list of parameters.

param_value
: the value associated to the parameter param_name. This parameter is optional. You can set the value of this parameter via a call to set_param.

param_list
: the updated list of parameters.

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

# Description

This function creates a new parameter in a list of parameters. You can set the value of the parameter using this function or you can set it via a call to set_param.

# Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
```

# See Also

init_param , set_param , get_param , remove_param , is_param

# Authors

collette
: ycollet@freesurf.fr

# Name

get_param — Get the value of a parameter in a parameter list

```
[res,err] = get_param(list_name,param_name,default_value)
```

# Parameters

list_name
: the list of parameters. This list must have been initialized by a call to init_param.

param_name
: a string. The name of the parameter to be add in the list of parameters.

default_value
: the default value to be stored in the parameter if param_name has not been found.

res
: the value of the parameter. If the parameter doesn't exist, res = [].

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

# Description

This function returns the value of the parameter param_name in a parameter list. If the err output parameter is not present, when an error occurs, a message is printed in the console.

# Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
disp(get_param(mylist,'minbound',-[1 1 1]));
disp(get_param(mylist,'maxbound', [1 1 1]));
```

# See Also

init_param , set_param , add_param , remove_param , is_param

# Authors

collette
: ycollet@freesurf.fr

# Name

init_param — Initialize the structure which will handles the parameters list

```
param_list = init_param()
```

# Parameters

param_list
: an initialized list of parameters (this list is empty and is of type plist).

# Description

This function initialize an empty list of parameters. You must initialize the list of parameters before using it.

# Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
```

# See Also

add_param , set_param , get_param , remove_param , is_param

# Authors

collette
: ycollet@freesurf.fr

# Name

is_param — Check if a parameter is present in a parameter list

```
[res,err] = is_param(list_name,param_name)
```

# Parameters

list_name
: the list of parameters. This list must have been initialize by a call to init_param.

param_name
: a string. The name of the parameter to be add in the list of parameters.

res
: the result: %T is the parameter is present, %F otherwise.

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

# Description

This function checks if a parameter is present in a parameter list. If the err output parameter is not present, when an error occurs, a message is printed in the console.

# Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
disp(is_param(mylist,'minbound'));
disp(is_param(mylist,'maxbound'));
```

# See Also

init_param , set_param , get_param , remove_param , add_param

# Authors

collette
: ycollet@freesurf.fr

# Name

list_param — List all the parameters name in a list of parameters

```
[string_list,err] = list_param(list_name)
```

# Parameters

list_name
: the list of parameters. This list must have been initialize by a call to init_param.

string_list
: the list of parameters name.

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

# Description

List all the parameters name in a list of parameters. If the err output parameter is not present, when an error occurs, a message is printed in the console.

# Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
mylist = add_param(mylist,'maxbound',[1 1 1]);
disp(list_param(mylist));
```

# See Also

init_param , set_param , get_param , remove_param , is_param

# Authors

collette
: ycollet@freesurf.fr

# Name

remove_param — Remove a parameter and its associated value from a list of parameters

```
[param_list,err] = remove_param(list_name,param_name)
```

# Parameters

list_name
: the list of parameters. This list must have been initialize by a call to init_param.

param_name
: a string. The name of the parameter to be removed from the list of parameters. If the parameter doesn't exist, nothing happens.

param_list
: the updated list of parameters.

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

# Description

This function allows to remove a parameter and its associated value from a list of parameters. If the err output parameter is not present, when an error occurs, a message is printed in the console.

# Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
mylist = add_param(mylist,'maxbound',[0 0 0]);
mylist = remove_param(mylist,'minbound');
```

# See Also

init_param , set_param , get_param , add_param , is_param

# Authors

collette
: ycollet@freesurf.fr

# Name

set_param — Set the value of a parameter in a parameter list

```
[param_list,err] = set_param(list_name,param_name,param_value)
```

# Parameters

list_name
: the list of parameters. This list must have been initialize by a call to init_param.

param_name
: a string. The name of the parameter to be added in the list of parameters.

param_value
: the value to be associated to the parameter param_name.

param_list
: the updated list of parameters.

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

# Description

This function sets the value of an already existing parameter. If the parameter doesn't exist, err is set to %T. If the err output parameter is not present, when an error occurs, a message is printed in the console.

# Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
[mylist,err] = set_param(mylist,'minbound',[1 1 1]); disp(err);
[mylist,err] = set_param(mylist,'maxbound',[1 1 1]); disp(err);
```

# See Also

init_param , add_param , get_param , remove_param , is_param

# Authors

collette
: ycollet@freesurf.fr

# Parte LIII. Atoms

# Name

Getting started

# Introduction

This page teaches how to get started with ATOMS module manager on the scilab platform towards a session example. It describes how one can install a module and load it in Scilab environment.

# Atoms

- **Configure ATOMS to display extra-informations**

```
-->atomsSetConfig('Verbose','True');
```

- **List available modules**

```
-->atomsList();

      ampl_toolbox - An interface to load .nl files created by AMPL
       ANN_Toolbox - ANN Toolbox
            conmin - A Scilab interface to the conmin optimization method
             CUTEr - Testing environment for optimization and linear algebra s
       dace_scilab - This is a conversion of the well known DACE kriging toolb
       dde_toolbox - Dynamic Data Exchange client for Scilab
        HYDROGRv50 - Models and function for operational hydrology
           lolimot - A fast neural network - LOcal LInear MOdel Tree
      module_lycee - Scilab pour les lycées
              NISP - Non Intrusive Spectral Projection
           plotlib - "Matlab-like" Plotting library for Scilab
          scilab2c - Translate Scilab code into C code
            scipad - Scipad 7.20
           simplex - This package contains the simplex optimization method
   sndfile_toolbox - Read & write sound files
           stixbox - Statistics toolbox for Scilab 5.2
```

- **Install a module** Installing a module download and extract it.

```
-->atomsInstall('NISP');

    NISP (2.1) will be installed in the 'allusers' section
    Installing NISP (2.1) ... success
```

- **Load a module** The module is installed but it's not loaded in the scilab environment and its functionnalities are not available yet.

By default, a module is added to the list of modules to load at Scilab start when it's installed. *(>> More information on the autoload system : atomsAutoloadList , atomsAutoloadAdd , atomsAutoloadDel ).*

```
-->atomsLoad('NISP');

    Start NISP Toolbox
      Load gateways
      Load help
      Load demos
```

- **Remove a module:**

```
-->atomsRemove NISP

    NISP (2.1) will be removed from the 'allusers' section
    the package NISP (2.1) is currently loaded, It will removed at next Scilab
```

# Authors

Pierre MARECHAL - DIGITEO

# Name

Functions Summary

# Install/Remove/Update modules ...

| | |
|---|---|
| atomsInstall | Install one or several modules |
| atomsUpdate | Update one or several modules |
| atomsRemove | Remove one or several modules |
| atomsGetInstalled | Get the list of installed external modules |
| atomsIsInstalled | Determines whether the module is installed or not |

# Load modules ...

| | |
|---|---|
| atomsLoad | Load one or several modules |
| atomsGetLoaded | Get the list of loaded modules |
| atomsIsLoaded | Determines whether the module is loaded or not |
| atomsAutoloadList | Get the list of modules registered to autoload at Scilab start. |
| atomsAutoloadAdd | Add one or several modules to autoload system. |
| atomsAutoloadDel | Remove one or several modules from autoload system. |

# Get information ...

| | |
|---|---|
| atomsList | List available modules |
| atomsSearch | Searches for modules |
| atomsShow | Show the caracteristics of a module |
| atomsDepTreeShow | Show the dependency tree of a module |

# Manage repositories ...

| | |
|---|---|
| atomsRepositoryList | Get the list of managed repositories |
| atomsRepositoryAdd | Add one or several URLs to the list of managed repositories |
| atomsRepositoryDel | Remove one or several URLs from the list of managed repositories |

# Manage ATOMS system ...

| | |
|---|---|
| atomsSystemUpdate | Update the list of available modules |
| atomsSetConfig | Manage ATOMS parameters |

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsAutoloadAdd — Add one or several modules to autoload

```
nbAdd = atomsAutoloadAdd(modules[,section])
```

# Parameters

modules

mx1, mx2 or mx3 Matrix of strings:

| | | | |
|---|---|---|---|
| *1st Col.* | **Technical name** | Mandatory | |
| *2nd Col.* | **Version** | Optionnal | If this field is empty or is not present, the most recent version is used |
| *3rd Col.* | **Installed section** | Optionnal | If this field is empty or is not present, and `module` is installed in both "user" and "allusers" sections, the section of autoload list is used. |

section

This argument controls which autoload list is changed.

`section` is a single-string and its value should be :

- `"allusers"`: modules are added to the "allusers" autoload list and all users of scilab are affected.
- `"user"`: modules are added to the "user" autoload list and only the current user is affected.

If SCI/contrib is write accessible, "allusers" is the default value. Otherwise, the default value is "user".

nbAdd

An integer : the number of modules successfully added.

# Description

# Examples

```
atomsRepositoryAdd('http://scene1.test.atoms.scilab.org');
atomsSetConfig("autoloadAddAfterInstall","False");

atomsInstall("toolbox_5","user");
atomsAutoloadList()

atomsAutoloadAdd("toolbox_5","user");
atomsAutoloadList()
```

```
atomsAutoloadDel(["toolbox_5"]);
atomsAutoloadAdd(["toolbox_5" "1.0"],"user");
atomsAutoloadList()

atomsAutoloadDel("toolbox_5");
atomsAutoloadAdd(["toolbox_5" "1.0" "user"],"user");
atomsAutoloadList()

atomsRemove("toolbox_5","user");
atomsSetConfig("autoloadAddAfterInstall","True");
atomsRepositoryDel('http://scene1.test.atoms.scilab.org');

atomsAutoloadList()
```

# See Also

atomsAutoloadDel , atomsAutoloadList

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsAutoloadDel — Remove one or several modules from the autoload system

```
nbDel = atomsAutoloadDel(modules[,section])
```

# Parameters

modules

mx1, mx2 or mx3 character string matrix:

| | | | |
|---|---|---|---|
| *1st Col.* | **Technical name** | Mandatory | |
| *2nd Col.* | **Version** | Optionnal | If this field is empty or is not present, all versions of the module are removed from the autoload system |
| *3rd Col.* | **Installed section** | Optionnal | If this field is empty or is not present, modules installed in both "user" and "allusers" sections are searched. |

section

This argument controls the list of sections where search modules(s) to remove from autoload system.

`section` is a single-string and its value should be :

- `"all"`: module(s) to remove from autotoload list are searched in both "user" and "allusers" sections.
- `"allusers"`: module(s) to remove from autotoload system are only searched in the "allusers" autoload list.
- `"user"`: module(s) to remove from autotoload system are only searched in the "user" autoload list.

If SCI is write accessible, "all" is the default value. Otherwise, the default value is "user".

nbDel

An integer : the number of modules successfully removed from the autoload system.

# Description

`atomsAutoloadDel` remove one or several modules from the autoload system.

# Examples

```
atomsRepositoryAdd('http://scene1.test.atoms.scilab.org');

atomsInstall("toolbox_5","user");
atomsAutoloadList()
```

```
atomsAutoloadDel("toolbox_5");
atomsAutoloadList()


atomsAutoloadAdd(["toolbox_5" "1.0" "user"],"user");
atomsAutoloadList()
atomsAutoloadDel(["toolbox_5" "1.0"]);
atomsAutoloadList()

atomsAutoloadAdd(["toolbox_5" "1.0" "user"],"user");
atomsAutoloadList()
atomsAutoloadDel(["toolbox_5" "1.0" "user"]);
atomsAutoloadList()

atomsAutoloadAdd(["toolbox_5" "1.0" "user"],"user");
atomsAutoloadList()
atomsAutoloadDel(["toolbox_5" "1.0" "user"],"user");
atomsAutoloadList()

atomsRemove("toolbox_5","user");
atomsRepositoryDel('http://scene1.test.atoms.scilab.org');
```

# See Also

atomsAutoloadAdd , atomsAutoloadList

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsAutoloadList — Get the list of modules registered to autoload

```
modules = atomsAutoloadList([section])
```

# Parameters

section

This argument controls the list of section where search URL(s).

`section` is a single-string and its value should be :

- `"all"`: module(s) present on both "user" and "allusers" autoload lists are returned.
- `"allusers"`: only module(s) present on the "allusers" autoload lists are returned.
- `"user"`: only module(s) present on the "user" autoload lists are returned.

The default value is "all".

modules

4xn character string matrix:

| | |
|---|---|
| *1st Col.* | **Module's technical name** |
| *2nd Col.* | **Module's version** |
| *3rd Col.* | **Module's installed section** |
| *4th Col.* | **Autoload list section** |

# Description

`atomsAutoloadList` returns the list of modules registered to autoload

# Examples

```
atomsRepositoryAdd('http://scene1.test.atoms.scilab.org');
atomsInstall("toolbox_1");

atomsAutoloadList('user')
atomsAutoloadList('allusers')
atomsAutoloadList('all')

atomsRemove("toolbox_1");
atomsRepositoryDel('http://scene1.test.atoms.scilab.org');
```

# See Also

atomsAutoloadAdd , atomsAutoloadDel

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsDepTreeShow — Show the dependency tree of a module

```
atomsDepTreeShow(module)
```

# Parameters

module

1x1 or 1x2 Matrix of strings:

| | | | |
|---|---|---|---|
| *1st Col.* | **Technical name** | Mandatory | |
| *2nd Col.* | **Version** | Optionnal | If this field is empty or is not present, the most recent version is used |

# Description

`atomsDepTreeShow` shows the dependency tree of an external module.

# Examples

```
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");
atomsDepTreeShow("toolbox_6")
```

# See Also

atomsShow , atomsList

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsGetInstalled — Get the list of installed external modules

```
installed = atomsGetInstalled(section)
```

# Parameters

installed

5xn String matrix :

- `1st column` : External module's technical name
- `2nd column` : External module's version
- `3rd column` : allusers/user, this parameter tell if the external module has been installed for all users or only for the current user.
- `4th column` : External module's installation path
- `5th column` : I/A, this parameter tell if the external module has been automatically or intentionnaly installed.

section

This argument filter the output list.

`section` is a single-string and its value should be :

- `"all"`: atomsGetInstalled() lists external modules installed in both "user" and "allusers" sections.
- `"allusers"`: atomsGetInstalled() only lists external modules installed in the "allusers" section.
- `"user"`: atomsGetInstalled() only lists external modules installed in the "user" section.

The default value is "all".

# Description

`atomsGetInstalled` returns the list of installed external modules

# Examples

```
atomsSetConfig("Verbose","True");
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");
atomsInstall("toolbox_5");
atomsGetInstalled();
atomsRemove("toolbox_5");
```

# See Also

atomsInstall , atomsIsInstalled

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsGetLoaded — Get the list of loaded external modules

```
loaded = atomsGetLoaded()
```

# Parameters

loaded

5xn String matrix :

- `1st column` : External module's technical name
- `2nd column` : External module's version
- `3rd column` : allusers/user, this parameter tell if the external module has been installed for all users or only for the current user.
- `4th column` : External module's installation path
- `5th column` : I/A, this parameter tell if the external module has been automatically or intentionnaly installed.

# Description

`atomsGetLoaded` returns the list of loaded external modules

# Examples

```
atomsSetConfig("Verbose","True");
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");
atomsInstall("toolbox_5");
atomsLoad("toolbox_5");

atomsGetLoaded("toolbox_5");

atomsRemove("toolbox_5");
```

# See Also

atomsLoad , atomsIsLoaded

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsInstall — Install one or several external modules

```
result = atomsInstall(modules[,section])
```

# Parameters

modules

mx1, mx2 Matrix of strings:

| | | | |
|---|---|---|---|
| *1st Col.* | **Technical name** | Mandatory | |
| *2nd Col.* | **Version** | Optionnal | If this field is empty or is not present, the most recent version is used |

section

This argument controls where the external module is installed.

`section` is a single-string and its value should be :

- `"allusers"`: the external module is installed for all users of the computer and is located in SCI/contrib ("allusers zone").
- `"user"`: the external module is only installed for the current user and is located in SCIHOME/atoms ("user zone").

If SCI/contrib is write accessible, "allusers" is the default value. Otherwise, the default value is "user".

result

5xn character string matrix:

| | | |
|---|---|---|
| *1st Col.* | **Technical name** | |
| *2nd Col.* | **Version** | |
| *3rd Col.* | **Installation section** | this parameter determines whether the module has been installed for all users or only for the current user. |
| *4th Col.* | **Installation path** | |
| *5th Col.* | **Status** | "I" stands for "Intentionnaly", "A" stands for "Automatically" |

# Description

`atomsInstall` install one or more external modules.

# Examples

```
// Display some additionnal informations
atomsSetConfig("Verbose","True");
```

```
// Load the test repository
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");

// Install a module
atomsInstall("toolbox_1");

// Install a specific version
atomsInstall(["toolbox_2" "2.0"]);

// Install several modules
atomsInstall(["toolbox_4" "1.0" ; "toolbox_2" "1.0"]);

// Install a module in the user section
atomsInstall(["toolbox_5"],"user");

// Install a module in the allusers section
// (write access on SCI directory is needed):
atomsInstall(["toolbox_6" "1.0";"toolbox_3" "1.0"],"allusers");

// Get the list of installed modules:
disp( atomsGetInstalled() );

// Cleaning :)
atomsRemove(["toolbox_1";"toolbox_2";"toolbox_3";"toolbox_4";"toolbox_5";"toolb
```

# See Also

atomsIsInstalled , atomsGetInstalled , atomsRemove

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsIsInstalled — Determines whether the module is installed. Returns true if the module is installed, false otherwise.

```
res = atomsIsInstalled(modules[,section])
```

# Parameters

modules

mx1, mx2 Matrix of strings:

| | | | |
|---|---|---|---|
| *1st Col.* | **Technical name** | Mandatory | |
| *2nd Col.* | **Version** | Optionnal | If this field is empty or is not present, module's version is ignored. |
| *3rd Col.* | **Section** | Optionnal | If this field is empty or is not present, module's section is ignored. |

section

This argument controls the list of searched modules.

`section` is a single-string and its value should be :

- `"all"`: atomsIsInstalled() searchs external modules installed in both "user" and "allusers" sections.
- `"allusers"`: atomsIsInstalled() searchs external modules installed in the "allusers" section.
- `"user"`: atomsIsInstalled() searchs external modules installed in the "user" section.

The default value is "all".

res

# Description

# Examples

```
atomsSetConfig("Verbose","True");
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");

// Install the needed module for the purpose of the example
atomsInstall("toolbox_5","user");

// simplest way
atomsIsInstalled("toolbox_5");

// Check several modules ...
atomsIsInstalled(["toolbox_5" "toolbox_4"])
```

```
// ... with a specific version
atomsIsInstalled(["toolbox_5" "1.0" ; "toolbox_4" "1.0" ; "toolbox_5" "1.1"])

// ... installed in a specific section
atomsIsInstalled(["toolbox_5" "1.0" ; "toolbox_4" "1.0" ; "toolbox_5" "1.1"],"u
atomsIsInstalled(["toolbox_5" "1.0" ; "toolbox_4" "1.0" ; "toolbox_5" "1.1"],"a

// Some cleaning ...
atomsRepositoryDel("http://scene2.test.atoms.scilab.org");
atomsRemove("toolbox_2");
```

# See Also

atomsInstall , atomsGetInstalled

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsIsLoaded — determines whether a module is loaded or not

```
result = atomsIsLoaded(modules)
```

# Parameters

modules

mx1, mx2 Matrix of strings:

| | | | |
|---|---|---|---|
| *1st Col.* | **Technical name** | Mandatory | |
| *2nd Col.* | **Version** | Optionnal | If this field is empty or is not present, module's version is ignored. |
| *3rd Col.* | **Section** | Optionnal | If this field is empty or is not present, module's section is ignored. |

result

mx1 boolean matrix

# Description

`atomsIsLoaded` determines whether a module is loaded or not.

# Examples

```
atomsSetConfig("Verbose","True");
atomsRepositoryAdd("http://scene2.test.atoms.scilab.org");

// Install toolbox_4, both 1.0 and 1.1 version
atomsInstall(["toolbox_4" "1.0"],"user");

// Load the version 1.0
atomsLoad(["toolbox_4" "1.0"]);


// Ignore the module's version
atomsIsLoaded("toolbox_4")

// With a specific version
atomsIsLoaded(["toolbox_4" "1.0"])

// Check several modules
atomsIsLoaded( ["toolbox_4" "1.0" ; "toolbox_4" "1.1" ; "toolbox_1" "1.0" ] )

// Some cleaning ...
atomsRepositoryDel("http://scene2.test.atoms.scilab.org");
atomsRemove("toolbox_2");
```

## See Also

atomsLoad , atomsGetLoaded

## Authors

Pierre MARECHAL - DIGITEO

# Name

atomsList — List available external modules

```
atomsList()
```

# Description

List available external modules

# Examples

```
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");
atomsList()
```

# See Also

atomsShow , atomsDepTreeShow

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsLoad — Install one or several external modules

```
result = atomsLoad(name[,version])
```

# Parameters

name
: Matrix of strings : External module name

version
: Matrix of strings : External module version. This is an optional parameter. If it's not defined, the most recent version of the module is used.

result

# Description

`atomsLoad` install one or more external modules.

# Examples

```
atomsSetConfig("Verbose","True");
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");
atomsInstall(["toolbox_2" "1.0"]);
atomsInstall(["toolbox_2" "2.0"]);

atomsLoad(["toolbox_2" "1.0"]);
t2_version()
t2_function1()

atomsRemove("toolbox_2");
```

# See Also

atomsIsLoaded , atomsGetLoaded

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsRemove — Remove one or several modules

```
result = atomsRemove(modules[,section])
```

# Parameters

modules

mx1 or mx2 character string matrix:

| | | | |
|---|---|---|---|
| *1st Col.* | **Technical name** | Mandatory | |
| *2nd Col.* | **Version** | Optionnal | If this field is empty or is not present, all versions of the module are removed. |

section

This argument controls the list of sections where search modules to remove.

`section` is a single-string and its value should be :

- `"all"`: Modules to remove are searched in both "user" and "allusers" sections.
- `"allusers"`: Modules to remove are only searched in the "allusers" section.
- `"user"`: Modules to remove are only searched in the "user" section.

If SCI is write accessible, "all" is the default value. Otherwise, the default value is "user".

result

5xn character string matrix:

| | | |
|---|---|---|
| *1st Col.* | **Technical name** | |
| *2nd Col.* | **Version** | |
| *3rd Col.* | **Installation section** | this parameter determines whether the module has been installed for all users or only for the current user. |
| *4th Col.* | **Installation path** | |
| *5th Col.* | **Status** | "I" stands for "Intentionnaly", "A" stands for "Automatically" |

# Description

`atomsRemove` remove one or more modules.

# Examples

```
// Display some additionnal informations
atomsSetConfig("Verbose","True");
```

```
// Load the test repository
atomsRepositoryAdd("http://scene2.test.atoms.scilab.org");

// install toolbox_4 : both 1.0 and 1.1 versions


// Remove all versions of a module
atomsInstall(["toolbox_4" "1.0";"toolbox_4" "1.1"],"user");
atomsRemove(["toolbox_4"]);


// Remove a specific version
atomsInstall(["toolbox_4" "1.0";"toolbox_4" "1.1"],"user");
atomsRemove(["toolbox_4" "1.0"]);
atomsRemove(["toolbox_4" "1.1"]);


// Remove several modules
atomsInstall(["toolbox_4";"toolbox_3"],"user");
atomsRemove(["toolbox_4";"toolbox_3"]);


// Remove a module from a specific section
// ! This example needs write access on SCI directory

atomsInstall("toolbox_4","user");
atomsInstall("toolbox_4","allusers");

disp(atomsGetInstalled());
atomsRemove("toolbox_4","user");
disp(atomsGetInstalled());
atomsRemove("toolbox_4","allusers");
disp(atomsGetInstalled());

// Unload the test repository
atomsRepositoryDel("http://scene2.test.atoms.scilab.org");
```

# See Also

atomsRemove , atomsIsInstalled , atomsGetInstalled

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsRepositoryAdd — Add one or several URLs to the list of managed repositories

```
nbAdd = atomsRepositoryAdd(url[,section])
```

# Parameters

url
: Matrix of strings : list of the URLs to add

section
: This argument controls where the repository is added.

  `section` is a single-string and its value should be :

  - `"allusers"`: the repository is added to the "allusers" list and all user of scilab are affected.
  - `"user"`: the repository is added to the "user" list and only the current user is affected.

  If SCI/contrib is write accessible, "allusers" is the default value. Otherwise, the default value is "user".

nbAdd
: An integer : the number of repositories successfully added.

# Description

# Examples

```
atomsRepositoryList()
atomsRepositoryAdd(["http://scene1.test.atoms.scilab.org"])
atomsRepositoryList()
```

# See Also

atomsRepositoryDel , atomsRepositoryList

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsRepositoryDel — Remove one or several URLs from the list of managed repositories

```
nbDel = atomsRepositoryDel(url[,section])
```

# Parameters

url
: character string matrix: list of the URLs to remove

section
: This argument controls the list of sections where search URL(s) to remove.

  `section` is a single-string and its value should be :

  - `"all"`: URL(s) to remove are searched in both "user" and "allusers" sections.
  - `"allusers"`: URL(s) to remove are only searched in the "allusers" section.
  - `"user"`: URL(s) to remove are only searched in the "user" section.

  If SCI is write accessible, "all" is the default value. Otherwise, the default value is "user".

nbDel
: An integer : the number of repositories successfully removed.

# Description

# Examples

```
atomsRepositoryList()
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org")
atomsRepositoryList()
atomsRepositoryDel("http://scene1.test.atoms.scilab.org")
atomsRepositoryList()
```

# See Also

atomsRepositoryAdd , atomsRepositoryList

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsRepositoryList — Get the list of managed repositories

```
repositories = atomsRepositoryList([section])
```

# Parameters

section

This argument controls the list of section where search URL(s).

`section` is a single-string and its value should be :

- `"all"`: URL(s) present in the "user", "allusers" and "official" section are listed.
- `"allusers"`: only URL(s) present in the "allusers" section are listed.
- `"user"`: only URL(s) present in the "user" section are listed.
- `"official"`: only URL(s) present in the "official" section are listed.

The default value is "all".

repositories

Matrix of strings : the first column give the list of repositories managed by ATOMS and the second column indicate if the repository is a official repository, if the repository has been added for all users or only for the current user.

# Description

`atomsRepositoryList` return a matrix that give the list of available repositories.

# Examples

```
atomsRepositoryAdd('http://scene1.atoms.scilab.org');
atomsRepositoryList()
atomsRepositoryList('all')
atomsRepositoryList('official')
atomsRepositoryList('allusers')
atomsRepositoryList('user')
atomsRepositoryDel('http://scene1.atoms.scilab.org');
```

# See Also

atomsRepositoryAdd , atomsRepositoryDel

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsSearch — Searches for external modules.

```
result = atomsSearch(pattern)
```

# Parameters

pattern
: String : The pattern to search for.

result

# Description

`atomsSearch` searches for packages matching one of the patterns supplied as input argument.

# Examples

```
atomsSetConfig("Verbose","True");
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");
atomsSearch("toolbox");
```

# See Also

atomsList

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsSetConfig — Manage ATOMS system parameters

```
result = atomsSetConfig(parameter,value)
```

# Parameters

parameter
: single-string matrix

value
: single-string matrix

result
: Number of changed parameters

# Proxy

| Parameter | Description | Values |
|---|---|---|
| useProxy | Use/Don't use proxies | **True**/False |
| proxyHost | the hostname (IP or DNS name) | |
| proxyPort | the port | |
| proxyUser | Specify the username for authentication on a proxy server | |
| proxyPassword | Specify the password for authentication on a proxy server | |

# Network

| Parameter | Description | Values |
|---|---|---|
| offLine | If set to "True", the system only works with local repositories. The offline mode permits the user to install modules from a local repository or a local package (hard disk, USB keys, ...) even if the network is unreachable. | True/**False** |

# Autoload System

| Parameter | Description | Values |
|---|---|---|
| autoload | Enable/Disable autoload system | **True**/False |
| autoloadAddAfterInstall | Automaticaly add a module to the list of module to autoload at Scilab start | **True**/False |

# Miscellenous

| Parameter | Description | Values |
|---|---|---|

| | | |
|---|---|---|
| verbose | Display or not extra-informations | True/**False** |

# Description

`atomsSetConfig` returns the list of modules registered to autoload

# Examples

```
// Display extra-informations
atomsSetConfig('Verbose','True')

// Disable autoload system
atomsSetConfig('autoload','False')
```

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsShow — Show the caracteristics of a module

```
atomsShow(module)
```

# Parameters

module

1x1 or 1x2 Matrix of strings:

| | | | |
|---|---|---|---|
| *1st Col.* | **Technical name** | Mandatory | |
| *2nd Col.* | **Version** | Optionnal | If this field is empty or is not present, the most recent version is used |

# Description

Show the caracteristics of a module

# Examples

```
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");
atomsShow("toolbox_5");
```

# See Also

atomsList , atomsDepTreeShow

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsSystemUpdate — Update the list of available modules

```
atomsSystemUpdate()
```

# Description

`atomsSystemUpdate` update the list of available modules.

# Examples

```
atomsSystemUpdate();
```

# Authors

Pierre MARECHAL - DIGITEO

# Name

atomsUpdate — Update one or several external modules

```
result = atomsUpdate()
result = atomsUpdate(name[,section]])
```

# Parameters

name
: 1xn character string matrix : module's technical name

section
: This argument controls the list of sections where search modules to update.

  `section` is a single-string and its value should be :

  - `"all"`: Modules to remove are searched in both "user" and "allusers" sections.
  - `"allusers"`: Modules to remove are only searched in the "allusers" section.
  - `"user"`: Modules to remove are only searched in the "user" section.

  If SCI is write accessible, "all" is the default value. Otherwise, the default value is "user".

result

# Description

`atomsUpdate` update one or more external modules.

# Examples

```
atomsSetConfig("Verbose","True");
atomsRepositoryAdd("http://scene1.test.atoms.scilab.org");

// Install toolbox_5
atomsInstall("toolbox_5");

disp(atomsGetInstalled());

// Load the 2nd scenario in which toolbox_4 has been updated:
// toolbox_4 version 1.1 has been added
// (toolbox_4 is a dependency of toolbox_5)

atomsRepositoryDel("http://scene1.test.atoms.scilab.org");
atomsRepositoryAdd("http://scene2.test.atoms.scilab.org");

// Update toolbox_5
atomsUpdate("toolbox_5");
disp(atomsGetInstalled());

// Some cleaning
```

```
atomsRepositoryDel("http://scene2.test.atoms.scilab.org");
atomsRemove("toolbox_5");
```

# See Also

atomsInstall , atomsRemove , atomsGetInstalled

# Authors

Pierre MARECHAL - DIGITEO

# Parte LIV. Entrada/Saída de Arquivos Matlab Binários

# Name

loadmatfile — loads a Matlab V6 MAT-file (binary or ASCII) into Scilab

```
loadmatfile(format,filename[,var1[,var2[,...]]])
loadmatfile(filename[,format[,var1[,var2[,...]]]])
loadmatfile(filename[,var1[,var2,[,...[,format]]]])
```

# Parameters

filename
: character string containing the path of the file (needed)

format
: file format (if not given and file has extension ".mat", file is considered to be binary)

  "-mat"
  : binary file

  "-ascii"
  : option to force Scilab to read file as an ASCII file

var1, var2
: character strings containing the name of the variables to load (only for binary files)

# Description

loads a Matlab MAT-file into Scilab. The Matlab data types are converted into the Scilab equivalents.

# Examples

```
A = rand(10,10);
B = sprand(100,100,0.1);
savematfile('test_matfile.mat','A','B','-v6');
clear;
loadmatfile('test_matfile.mat');
disp(A)
disp(B)
```

# See Also

load , savematfile , save , mfile2sci , matfile2sci

# Authors

Serge Steer (INRIA)
V.C

# Bibliography

This function has been developped following the "MAT-File Format" description: `Mat-File Format [http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf]`

# Name

matfile_close — Closes a Matlab V5 binary MAT-file.

```
status = matfile_close(fd)
```

# Parameters

fd
: Real: file descriptor (returned by matfile_open).

status
: Boolean: *%T* if closure succeeds, *%F* otherwise.

# Description

Closes a Matlab binary MAT-file opened by matfile_open.

# See Also

matfile_open , matfile_varreadnext , matfile_varwrite , matfile_listvar

# Authors

V.C

# Bibliography

This function uses MATIO library (`http://sourceforge.net/projects/matio/`).

# Name

matfile_listvar — Lists variables of a Matlab V5 binary MAT-file.

```
[names[, classes[, types]]] = matfile_listvar(fd)
```

# Parameters

fd
: Real: file descriptor (returned by matfile_open).

names
: String vector: names of the variables.

classes
: Real vector: classes of the variables.

types
: Real vector: data types of the variables.

# Description

Lists variables of a Matlab binary MAT-file opened by matfile_open.

# See Also

matfile_open , matfile_close , matfile_varwrite , matfile_varreadnext

# Authors

V.C

# Bibliography

This function uses MATIO library (`http://sourceforge.net/projects/matio/`).

# Name

matfile_open — Opens a Matlab V5 binary MAT-file.

```
fd = matfile_open(filename[, mode])
```

# Parameters

filename
: String: the path of the file. Must contain only ANSI character.

mode
: String: file access type (`"r"` by default).

- *"r"*: opens the file for reading.
- *"w"*: opens the file for writing.

fd
: Real: file descriptor (-1 if opening failed).

# Description

Opens a Matlab binary MAT-file for reading or writing data.

# See Also

matfile_close , matfile_varreadnext , matfile_varwrite , matfile_listvar

# Authors

V.C

# Bibliography

This function uses MATIO library (`http://sourceforge.net/projects/matio/`).

# Name

matfile_varreadnext — Reads next variable in a Matlab V5 binary MAT-file.

```
[name[, value[, vartype]]] = matfile_varreadnext(fd)
```

# Parameters

fd
: Real: file descriptor (returned by matfile_open).

name
: String: name of the variable read or `""` if reading failed.

value
: Any Scilab type: value of the variable read or an empty matrix if reading failed.

vartype
: Real: type of the variable if reading succeeds or:

- *0*: if the variable type is unknown.
- *-1*: if end of file has been reached.

# Description

Reads next variable in a Matlab binary MAT-file opened by matfile_open.

# See Also

matfile_open , matfile_close , matfile_varwrite , matfile_listvar

# Authors

V.C

# Bibliography

This function uses MATIO library (`http://sourceforge.net/projects/matio/`).

# Name

matfile_varwrite — Write a variable in a Matlab V5 binary MAT-file.

```
status = matfile_varreadnext(fd, name, value, compressionflag)
```

# Parameters

fd
: Real: file descriptor (returned by matfile_open).

name
: String: name of the variable to write in the file.

value
: Any Scilab type: value of the variable to write in the file.

compressionflag
: Boolean: indicate if data compression has to be used (flag equaled to *%T*) or not.

status
: Boolean: *%T* if writing succeeds, *%F* otherwise.

# Description

Writes a variable in a Matlab binary MAT-file opened by matfile_open.

# See Also

matfile_open , matfile_close , matfile_varreadnext , matfile_listvar

# Authors

V.C

# Bibliography

This function uses MATIO library (`http://sourceforge.net/projects/matio/`).

# Name

savematfile — write a Matlab MAT-file (binary or ASCII)

```
savematfile('filename')
savematfile('filename', 'var1', 'var2', ...)
savematfile('filename', '-struct', 's')
savematfile('filename', '-struct', 's', 'f1', 'f2', ...)
savematfile(..., '-v4')
savematfile(..., '-v6')
savematfile(..., '-v7')
savematfile(..., '-v7.3')
savematfile filename var1 var2 ...
```

# Parameters

filename
: character string containing the path of the file (needed)

format
: data format to use

"-mat"
: binary MAT-file (default)

"-ascii"
: 8-bit ASCII format

"-ascii" "-double"
: 16-bit ASCII format

"-ascii" "-tabs"
: delimits with tabs

"-ascii" "-double" "-tabs"
: 16-digit ASCII format, tab delimited

"-v4"
: A format that MATLAB Version 4 can open

"-v6"
: A format that MATLAB Version 6 and earlier can open

"-v7"
: A format that MATLAB Version 7 and earlier can open (default)

"-v7.3"
: A format that MATLAB Version 7.3 and earlier can open

var1, var2
: character strings containing the name of the variables to load (only for binary files)

"-struct" "s"
: saves all fields of the scalar structure s as individual variables within the file filename.

"-struct" "s" "f1" "f2"
: saves as individual variables only those structure fields specified (s.f1, s.f2, ...).

# Description

saves variables in a Matlab MAT-file from Scilab. The Scilab data types are converted into the Matlab equivalents.

# Examples

```
A = rand(10,10);
B = sprand(100,100,0.1);
savematfile('test_matfile.mat','A','B','-v6');
clear;
loadmatfile('test_matfile.mat');
disp(A)
disp(B)
```

# See Also

load , save , loadmatfile , mfile2sci

# Authors

Serge Steer (INRIA)
V.C

# Bibliography

This function has been developped following the "MAT-File Format" description: `Mat-File Format [http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf]`

# Parte LV. xcos

# Índice


6. Batch functions ..... 2351
    lincos ..... 2352
    scicos ..... 2353
    scicos_simulate ..... 2354
    scicosim ..... 2356
    steadycos ..... 2358
7. palettes ..... 2360
    1. Annotations palette ..... 2360
    2. Commonly used blocks palette ..... 2363
    3. Continuous time systems palette ..... 2371
    4. Demonstrations blocks palette ..... 2396
    5. Discontinuities palette ..... 2413
    6. Discrete time systems palette ..... 2426
    7. Electrical palette ..... 2443
    8. Event handling palette ..... 2501
    9. Implicit palette ..... 2547
    10. Integer palette ..... 2552
    11. Lookup tables palette ..... 2583
    12. Math operations palette ..... 2590
    13. Matrix operation palette ..... 2637
    14. Port & Subsystem palette ..... 2700
    15. Signal processing palette ..... 2707
    16. Signal routing palette ..... 2712
    17. Sinks palette ..... 2748
    18. Sources palette ..... 2803
    19. Thermohydraulics palette ..... 2856
    20. User defined functions palette ..... 2870
    21. Zero crossing detection palette ..... 2896
8. Programming xcos Blocks ..... 2906
    1. C Computational Functions ..... 2906
    2. Scilab Computational Functions ..... 2920
    3. Utilities Functions ..... 2924
9. Scilab Data Structures ..... 2935
    1. Blocks ..... 2935
    2. Compilation/Simulation ..... 2946
    3. Diagram ..... 2954
    4. Links ..... 2959
10. Scilab Utilities Functions ..... 2962
    buildouttb ..... 2963
    create_palette ..... 2964
    get_scicos_version ..... 2965
    scicos_debug ..... 2966
    var2vec ..... 2967
    vec2var ..... 2968

# Capítulo 6. Batch functions

# Nome

lincos — Constructs by linearization a linear state-space model from a general dynamical system described by a xcos diagram

```
sys= lincos(scs_m [,x0,u0 [,param] ])
```

# Module

- xcos

# Description

Construct a linear state-space system by linearizing a model given as a xcos diagram.

The output is a Scilab data structure of type continuous-time state-space linear system.

```
sys= lincos(scs_m [,x0,u0 [,param] ])
```

# Parameters

- **scs_m :** a xcos data structure
- **x0 :** column vector. Continuous state around which linearization to be done (default 0)
- **u0 :** column vector. Input around which linearization to be done (default 0)
- **param :** param: list with two elements (default list(1.d-6,0)) param(1): scalar. Perturbation level for linearization; the following variation is used del([x;u])_i = param(1)+param(1)*1d-4*abs([x;u])_i. param(2): scalar. Time t.
- **sys :** state-space system

# File content

- SCI/modules/scicos/macros/scicos_auto/lincos.sci

# See Also

- steadycos - Finds an equilibrium state of a general dynamical system described by a xcos diagram (Scilab Function)
- scicos_simulate - Function for running xcos simulation in batch mode (Scilab Function)

# Authors

**Ramine Nikoukhah** - INRIA

# Nome

scicos — OBSOLETE - see xcos

```
scs_m = scicos([toto])
```

# Module

- xcos

# See Also

- xcos

# Nome

scicos_simulate — Function for running xcos simulation in batch mode

```
Info=scicos_simulate(scs_m,Info[,%scicos_context][,flag])
```

# Module

- xcos

# Description

This function is used to simulate xcos diagrams in batch mode. It requires the scs_m structure which can be obtained by loading in Scilab the .cos file (e.g. load mydiagram.cos).

Contrary to the function , the diagram need not be compiled before being saved.

```
Info=scicos_simulate(scs_m,Info[,%scicos_context][,flag])
```

# Parameters

- **scs_m :** xcos diagram (obtained by "load file.cos"). Note that the version of file.cos must be the current version. If not, load into xcos and save.
- **Info :** a list. It must be list() at the first call, then use output Info as input Info for the next calls. Info contains compilation and simulation information and is used to avoid recompilation when not needed.
- **%scicos_context :** a Scilab struct containing values of symbolic variables used in the context and xcos blocks. This is often used to change a parameter in the diagram context. In that case, make sure that in the diagram context the variable is defined such that it can be modified. Say a variable "a" is to be defined in the context having value 1, and later in batch mode, we want to change the value of "a". In that case, in the context of the diagram place: if exists('a') then a=1,end If you want then to run the simulation in batch mode using the value a=2, set: %scicos_context.a=2
- **flag :** string. If it equals 'nw' (no window), then blocks using graphical windows are not executed. Note that the list of such blocks must be updated as new blocks are added.
- **Info :** contains compilation and simulation information and is used to avoid recompilation when not needed. Normally it should not be modified.

# File content

- SCI/modules/scicos/macros/scicos_auto/scicos_simulate.sci

# See Also

- scicosim - Scicos (batch) simulation function (Scilab Function)
- xcos - Block diagram editor and GUI for the hybrid simulator (Scilab Function)
- steadycos - Finds an equilibrium state of a general dynamical system described by a xcos diagram (Scilab Function)

- lincos - Constructs by linearization a linear state-space model from a general dynamical system described by a xcos diagram (Scilab Function)

# Authors

**Ramine Nikoukhah** - INRIA

# Nome

scicosim — xcos (batch) simulation function

```
[state,t] = scicosim(state,tcur,tf,sim,str,tol)
```

# Module

- xcos

# Description

Simulator for xcos compiled diagram.

Usually scicosim is called by xcos to perform simulation of a diagram.

But scicosim may also be called outside xcos. Typical usage in such a case may be :

For advanced user it is possible to "manually" change some parameters or state values.

```
[state,t] = scicosim(state,tcur,tf,sim,str,tol)
```

# Parameters

- **state :** Scilab tlist containing initial state. Usually generated by xcos Compile. After loading a compiled .cos file, it can be found in %cpr.state.
- **tcur :** starting time of simulation. At the beginning it must be zero.
- **tf :** final simulation time.
- **sim :** Scilab tlist containing compilation results. Usually generated by xcos Compile. After loading a compiled .cos file, it can be found in %cpr.sim.
- **str :** 'start' , 'run' or 'finish'. Function must be first called with 'start', then with 'run' one or more times, and finally with 'finish'.
- **tol :** vector [atol,rtol,ttol,deltat,realtimescale,solver,hmax] where atol, rtol are respectively the absolute and relative tolerances for ode or dae solver, ttol is the precision on event dates (must be very small), deltat is maximum integration interval for each call to ode solver (sometimes needed to force restaring the call to solver), realtimescale is the correspondance between simulation time and real time (0 means no slowing down), solver is the choice of solver (0: lsodar, 100: daskr), hmax: max step size used by solver. Default: [0.0001,1.000E-06,1.000E-10,100001,0,0]
- **state :** state after simulation
- **t :** final reached time. Usually tf but not always.

# See Also

- scicos_simulate - Function for running xcos simulation in batch mode (Scilab Function)
- xcos - Block diagram editor and GUI for the hybrid simulator (Scilab Function)

# Authors


- **Ramine Nikoukhah** INRIA
- **Alan Layec** INRIA

# Nome

steadycos — Finds an equilibrium state of a general dynamical system described by a xcos diagram

```
[X,U,Y,XP]=steadycos(scs_m,X,U,Y,Indx,Indu,Indy [,Indxp [,param ] ])
```

# Module

- xcos

# Description

This function finds the steady state for a given system described by a xcos diagram. The diagram consists in general of a Super block with input and output port blocks. The steady state concern only the continuous-time dynamics.

```
[X,U,Y,XP]=steadycos(scs_m,X,U,Y,Indx,Indu,Indy [,Indxp [,param ] ])
```

# Parameters

- **scs_m :** a xcos data structure
- **X :** column vector. Continuous state. Can be set to [] if zero.
- **U :** column vector. Input. Can be set to [] if zero.
- **Y :** column vector. Output. Can be set to [] if zero.
- **Indx :** index of entries of X that are not fixed. If all can vary, set to 1:$
- **Indu :** index of entries of U that are not fixed. If all can vary, set to 1:$
- **Indy :** index of entries of Y that are not fixed. If all can vary, set to 1:$
- **Indxp :** index of entries of XP (derivative of x) that need not be zero. If all can vary, set to 1:$. Default [].
- **param :** list with two elements (default list(1.d-6,0)). param(1): scalar. Perturbation level for linearization; the following variation is used del([x;u])_i = param(1)+param(1)*1d-4*abs([x;u])_i. param(2): scalar. Time t.
- **X :** steady state X
- **U :** stationary input U
- **Y :** output corresponding to steady state found
- **XP :** derivative of the state corresponding to steady state found

# File content

- SCI/modules/scicos/macros/scicos_auto/steadycos.sci

# See Also

- lincos - Constructs by linearization a linear state-space model from a general dynamical system described by a xcos diagram (Scilab Function)

- scicos_simulate - Function for running xcos simulation in batch mode (Scilab Function)
- xcos - Block diagram editor and GUI for the hybrid simulator (Scilab Function)

# Authors

**Ramine Nikoukhah** - INRIA

# Capítulo 7. palettes

## 1. Annotations palette

# Nome

Annotations_pal — Annotations palette

## Block Screenshot

## Module

- xcos

## Description

This palette includes blocks used for annotations.

## Blocks

- TEXT_f - Text

# Nome

TEXT_f — Text

## Block Screenshot

## Contents


- Text
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

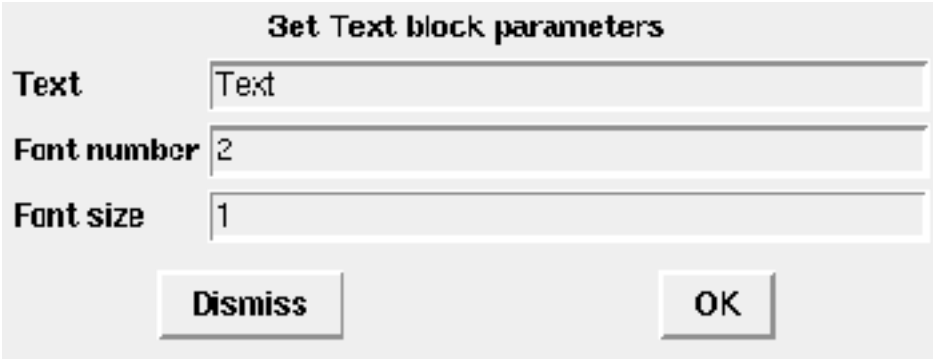

## Palette

- Annotations palette

## Description

This special block is only use to add text at any point of the diagram window. It has no effect on the simulation.

## Dialog box




- **Text**

  a character string, Text to be displayed

  Properties : Type 'str' of size -1

- **Font number**

  a positive integer less than 6, number of selected font (see**xset** )

Properties : Type 'vec' of size 1

- **Font size**

  a positive integer, selected font size (see**xset** ).

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *text*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/TEXT_f.sci

## Authors

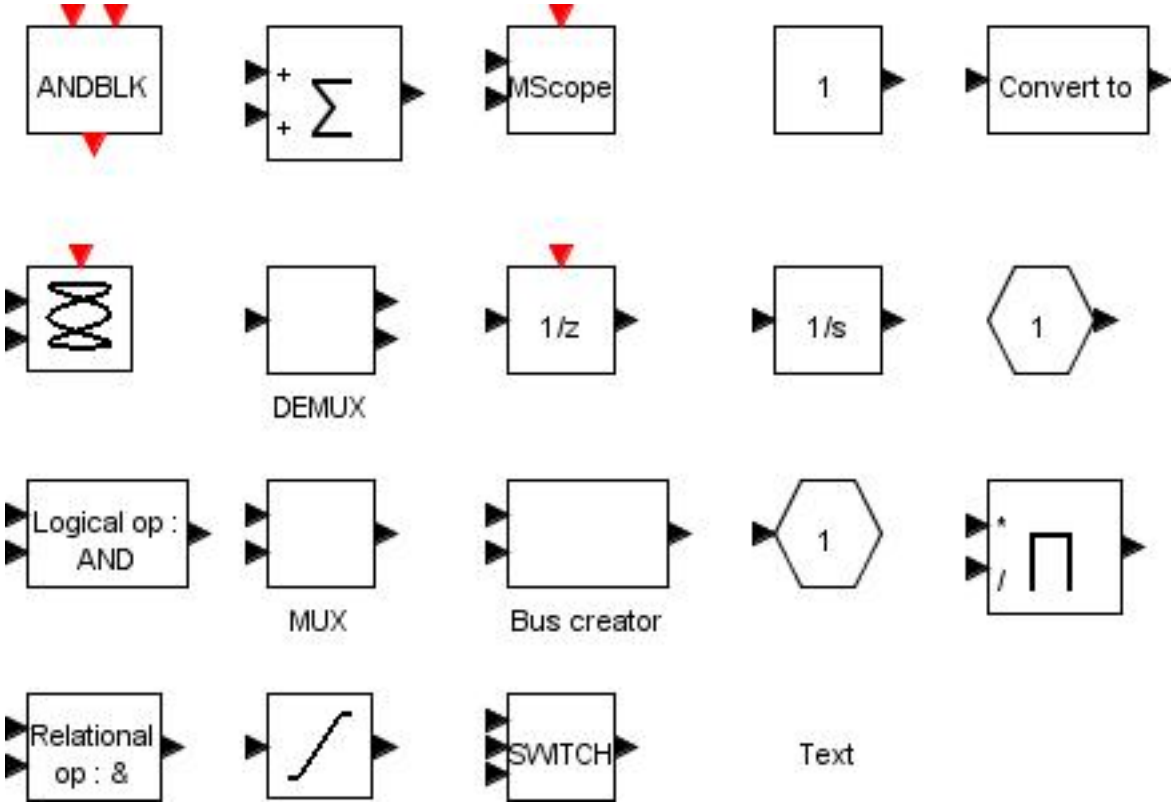
**Ramine Nikoukhah** - INRIA

# 2. Commonly used blocks palette

# Nome

Commonlyusedblocks_pal — Commonly used blocks palette

## Block Screenshot




## Module

- xcos

## Description

In the Commonly used blocks palette, you can find blocks from other palettes that most models use.

## Blocks

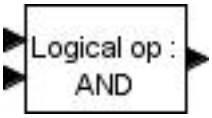
- ANDBLK - Activation and
- BIGSOM_f - Sum
- CMSCOPE - Multi display scope
- CONST_m - Constant
- CONVERT - CONVERT Data Type Conversion
- CSCOPXY - y=f(x) permanent viewer
- DEMUX - Demultiplexer
- DOLLAR_f - Delay operator
- INTEGRAL_f - Integration

- IN_f - Input Port
- LOGICAL_OP - Logical operation
- MUX - Multiplexer
- NRMSOM_f - Merge data
- OUT_f - Output Port
- PRODUCT - Product
- RELATIONALOP — Relational operation
- SATURATION - Saturation
- SWITCH2_m - Switch2
- TEXT_f - Text

# Nome

LOGICAL_OP — Logical operation

## Block Screenshot




## Contents


- Logical operation
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

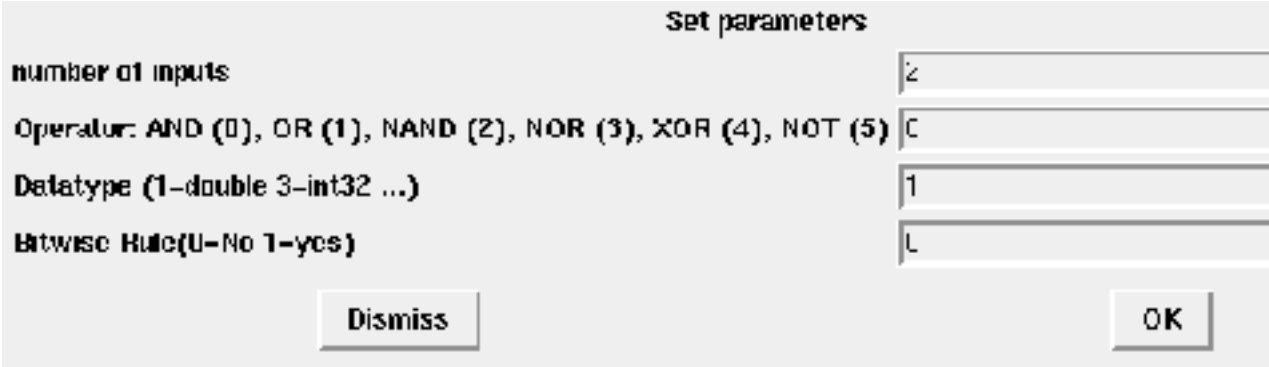

## Palette

- Commonly used blocks palette

## Description

The Logical Operator block performs the specified logical operation on its inputs. An input value is TRUE (1) if it is nonzero and FALSE (0) if it is zero.

## Dialog box




- **number of inputs**

  The number of block inputs. The value must be appropriate for the selected operator.

  Properties : Type 'vec' of size 1

- **Operator: AND**

The logical operator to be applied to the block inputs. Valid choices are the operators from the list.

Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**

  **- port 2 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *logicalop*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/LOGICAL_OP.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/logicalop.c
- SCI/modules/scicos_blocks/src/c/logicalop_i32.c
- SCI/modules/scicos_blocks/src/c/logicalop_i16.c
- SCI/modules/scicos_blocks/src/c/logicalop_i8.c
- SCI/modules/scicos_blocks/src/c/logicalop_ui32.c
- SCI/modules/scicos_blocks/src/c/logicalop_ui16.c
- SCI/modules/scicos_blocks/src/c/logicalop_ui8.c

## Authors

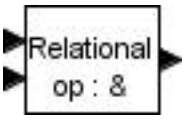
- **Fady NASSIF** INRIA

- **Ramine Nikoukhah** INRIA

# Nome

RELATIONALOP — Relational operation

## Block Screenshot




## Contents

- Relational operation
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

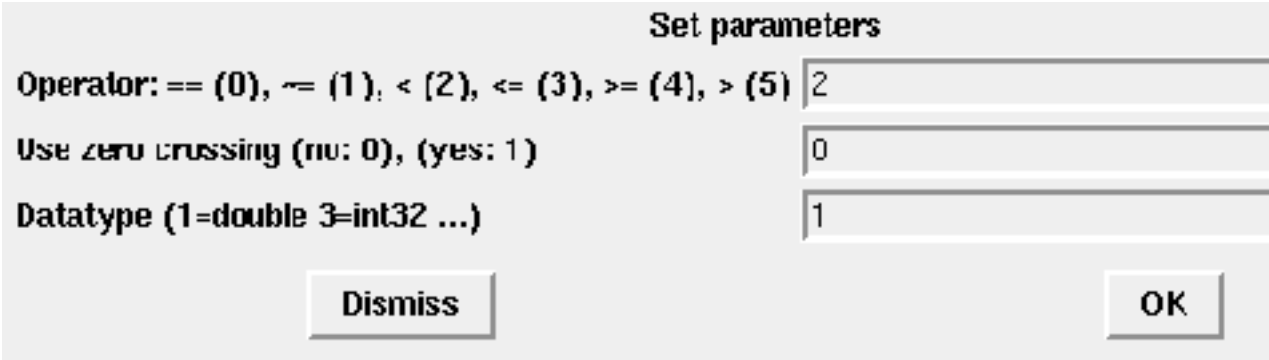
## Palette

- Commonly used blocks palette

## Description

It performs logical comparison of its two inputs.

## Dialog box




- **Operator: ==**

  Designate the relational operator used to compare the two inputs.

  Properties : Type 'vec' of size 1

- **Use zero crossing**

Select to enable zero crossing detection.

Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *relationalop*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/RELATIONALOP.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/relational_op.c
- SCI/modules/scicos_blocks/src/c/relational_op_i32.c
- SCI/modules/scicos_blocks/src/c/relational_op_i16.c
- SCI/modules/scicos_blocks/src/c/relational_op_i8.c
- SCI/modules/scicos_blocks/src/c/relational_op_ui32.c
- SCI/modules/scicos_blocks/src/c/relational_op_ui16.c
- SCI/modules/scicos_blocks/src/c/relational_op_ui8.c

## Authors

- **Fady NASSIF** INRIA

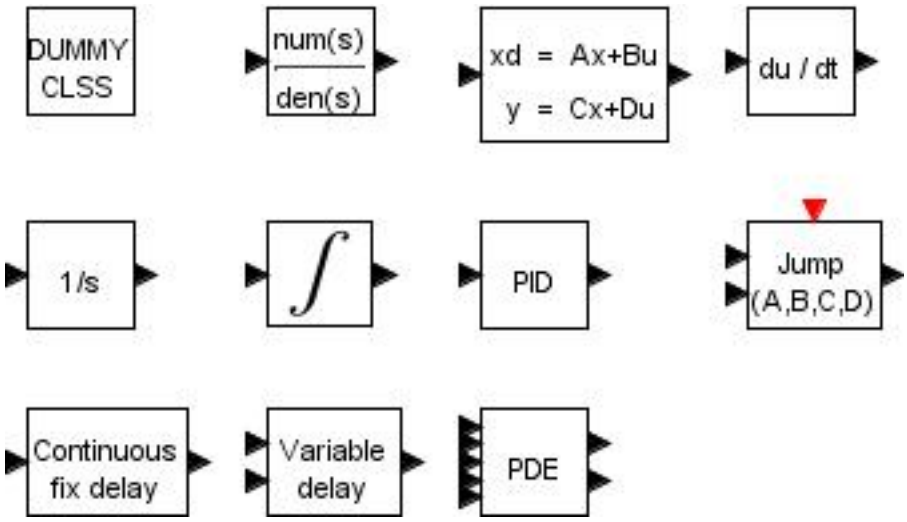
- **Ramine Nikoukhah** INRIA

# 3. Continuous time systems palette

# Nome

Continuous_pal — Continuous time systems palette

## Block Screenshot




## Module

- xcos

## Description

The Continuous time systems palette includes basic linear blocks .

## Blocks

- CLINDUMMY_f — Dummy
- CLR — Continuous transfer function
- CLSS - Continuous state-space system
- DERIV - Derivative
- INTEGRAL_f - Integration
- INTEGRAL_m — Integration
- PID - PID regulator
- TCLSS — Continuous linear system with jump
- TIME_DELAY - Time delay
- VARIABLE_DELAY — Variable delay

# Nome

CLINDUMMY_f — Dummy

## Block Screenshot

## Contents


- Dummy
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Authors”


## Palette

- Continuous time systems palette

## Description

This block should be placed in any block diagram that contains a zero-crossing block but no continuous system with state. The reason for that is that it is the ode solver that find zero crossing surfaces.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cdummy*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/CLINDUMMY_f.sci

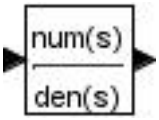
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

CLR — Continuous transfer function

## Block Screenshot




## Contents


- Continuous transfer function
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

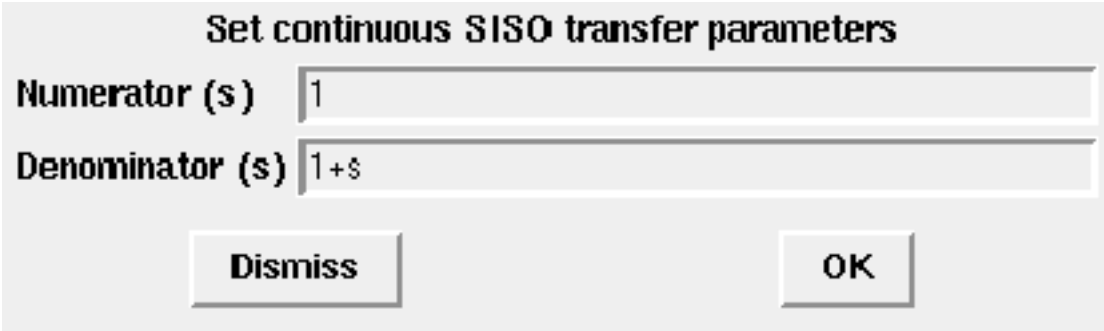

## Palette

- Continuous time systems palette

## Description

This block realizes a SISO linear system represented by its rational transfer function **Numerator/Denominator**. The rational function must be proper.

## Dialog box




- **Numerator**

  This parameter sets the numerator of the transfer function.

  This must be a polynomial in s.

  Properties : Type 'pol' of size 1.

- **Denominator**

This parameter sets the denominator of the transfer function.

This must be a polynomial in s.

Properties : Type 'pol' of size 1.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csslti4*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/CLR.sci

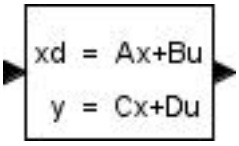
## Computational function

- SCI/modules/scicos_blocks/src/c/csslti4.c (Type 4)

# Nome

CLSS — Continuous state-space system

## Block Screenshot




## Contents


- Continuous state-space system
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”


## Palette

- Continuous time systems palette

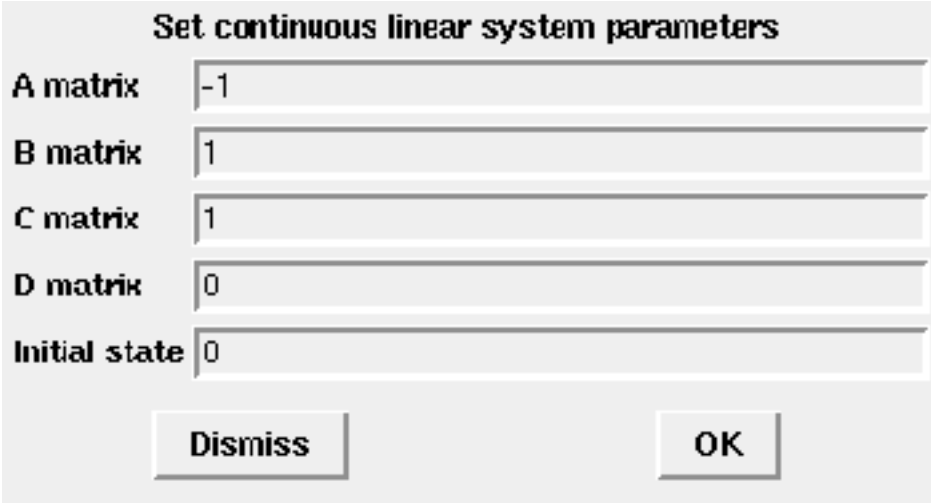
## Description

This block realizes a continuous-time linear state-space system.

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= C * x + D * u \end{aligned}$$

The system is defined by the $(A, B, C, D)$ matrices and the initial state $x_0$. The dimensions must be compatible.

## Dialog box




- **$A$ matrix**

  A square matrix.

  Properties : Type 'mat' of size [-1,-1].

- **B matrix**

  The $B$ matrix, [] if system has no input.

  Properties : Type 'mat' of size ["size(%1,2)","-1"].

- **C matrix**

  The $C$ matrix , [] if system has no output.

  Properties : Type 'mat' of size ["-1","size(%1,2)"].

- **D matrix**

  The $D$ matrix, [] if system has no D term.

  Properties : Type 'mat' of size [-1,-1].

- **Initial state**

  A vector/scalar initial state of the system.

  Properties : Type 'vec' of size "size(%1,2)".

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no

- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csslti4*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/CLSS.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/csslti4.c (Type 4)

# Nome

DERIV — Derivative

## Block Screenshot

## Contents


- Derivative
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Continuous time systems palette

## Description

The Derivative block approximates the derivative of its input by computing:

$$\frac{\Delta u}{\Delta t}$$

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0

- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *deriv*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DERIV.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/deriv.c (Type 4)

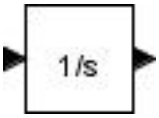
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

INTEGRAL_f — Integration

## Block Screenshot




## Contents


- Integration
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

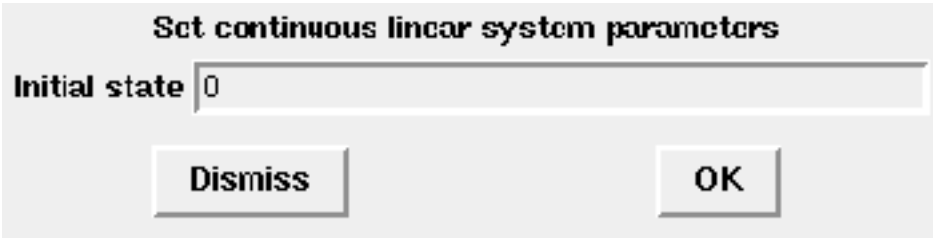

## Palette

- Continuous time systems palette

## Description

This block is an integrator. The output is the integral of the input.

## Dialog box




- **Initial Condition**

  A scalar that gives the initial condition.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no

- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *integr*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/INTEGRAL_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/integr.f (Type 0)

# Nome

INTEGRAL_m — Integration

## Block Screenshot

## Contents


- Integration
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

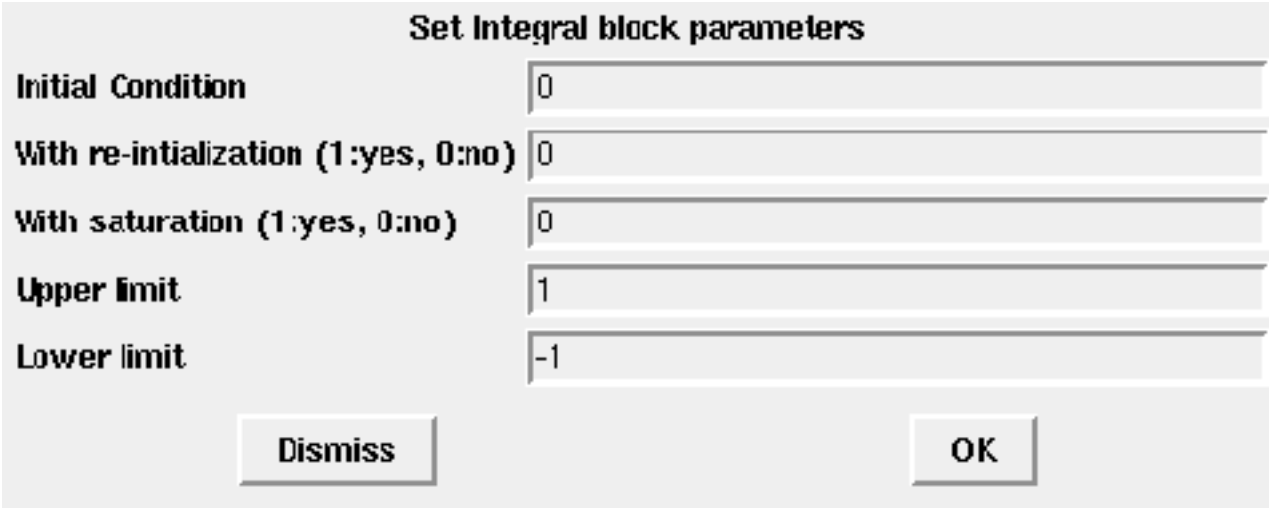

## Palette

- Continuous time systems palette

## Description

This block is an integrator. The output is the integral of the input.

## Dialog box




- **Initial Condition**

  A vector/scalar initial conditions.

With that parameter, one can define the datatype of the input/output. It can be a real or a complex type.

Properties : Type 'mat' of size [-1,-1].

- **With re-initialization**

  To reset its state to the specified initial condition based on an external signal select**1** .

  Properties : Type 'vec' of size 1.

- **With saturation**

  If selected, limits the states to a value between the Lower saturation limit and Upper saturation limit parameters.

  Properties : Type 'vec' of size 1.

- **Upper limit**

  The upper limit for the integral.

  Properties : Type 'mat' of size [-1,-1].

- **Lower limit**

  The lower limit for the integral.

  Properties : Type 'mat' of size [-1,-1].

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *integral_func*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/INTEGRAL_m.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/integral_func.c
- SCI/modules/scicos_blocks/src/c/integralz_func.c

## Authors


- **Fady NASSIF** INRIA
- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA

# Nome

PID — PID regulator

## Block Screenshot

## Contents

- PID regulator
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “Authors”

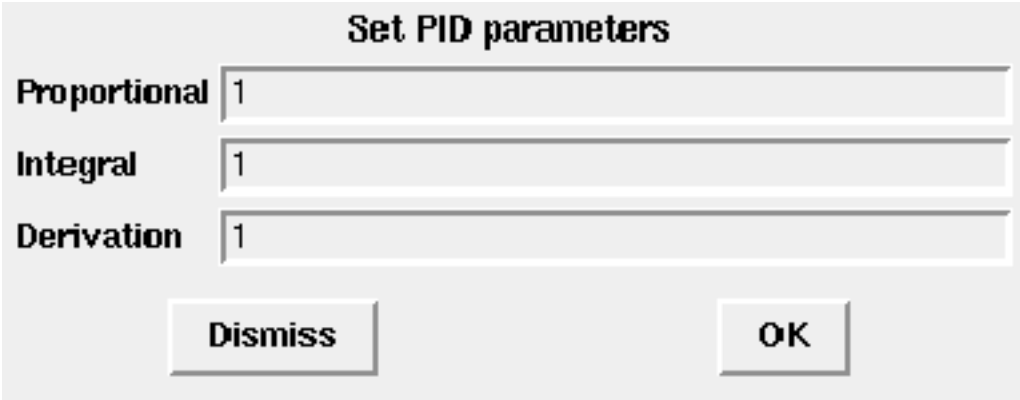
## Palette

- Continuous time systems palette

## Description

This block implements a PID controller. The PID controller calculation (algorithm) involves three separate parameters; the Proportional, the Integral and Derivative values. The Proportional value determines the reaction to the current error, the Integral determines the reaction based on the sum of recent errors and the Derivative determines the reaction to the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve or the power supply of a heating element.

## Dialog box



- **Proportional**

  The value of the gain that multiply the error.

  Properties : Type 'vec' of size -1.

- **Integral**

  The value of the integral time of the error.(1/Integral)

  Properties : Type 'vec' of size -1.

- **Derivation**

  The value of the derivative time of the error.

  Properties : Type 'vec' of size -1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

- **regular outputs:**

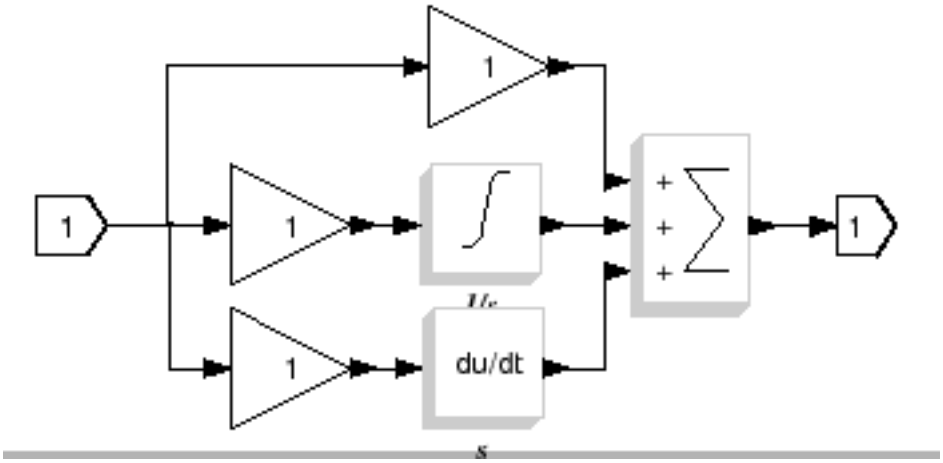
  **- port 1 : size [-1,-2] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/PID.sci

## Compiled Super Block content



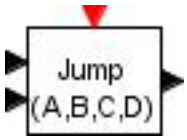

## Authors

**Fady NASSIF** - INRIA

# Nome

TCLSS — Continuous linear system with jump

## Block Screenshot




## Contents


- Continuous linear system with jump
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

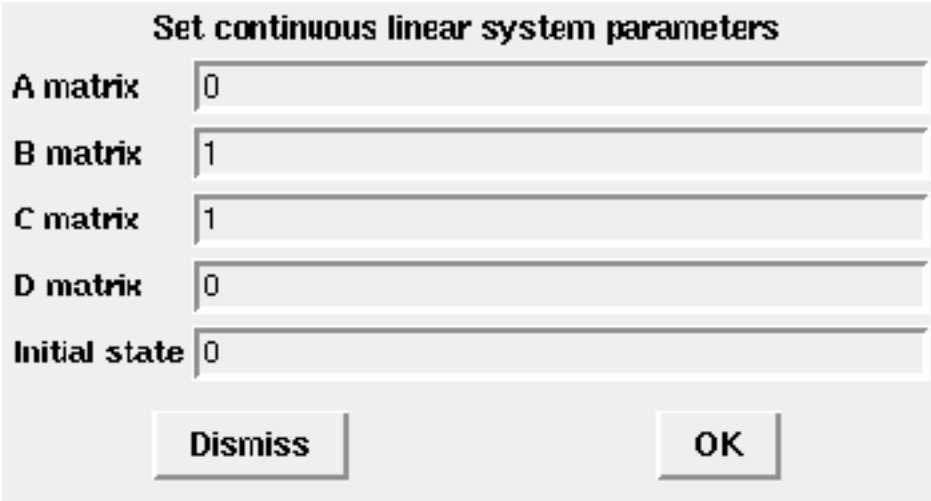

## Palette

- Continuous time systems palette

## Description

This block realizes a continuous-time linear state-space system with the possibility of jumps in the state. The number of inputs to this block is two. The first input is the regular input of the linear system, the second carries the new value of the state which is copied into the state when an event arrives at the unique event input port of this block. That means the state of the system jumps to the value present on the second input (of size equal to that of the state). The system is defined by the matrices and the initial state . The dimensions must be compatible. The sizes of inputs and outputs are adjusted automatically.

## Dialog box



- **$A$ matrix**

  A square matrix.

  Properties : Type 'mat' of size [-1,-1].

- **B matrix**

  The $B$ matrix, [] if system has no input.

  Properties : Type 'mat' of size ["size(%1,2)","-1"].

- **C matrix**

  The $C$ matrix , [] if system has no output.

  Properties : Type 'mat' of size ["-1","size(%1,2)"].

- **D matrix**

  The $D$ matrix, [] if system has no D term.

  Properties : Type 'mat' of size [-1,-1].

- **Initial state**

  A vector/scalar initial state of the system.

  Properties : Type 'vec' of size "size(%1,2)".

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no

- **name of computational function:** *tcslti4*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/TCLSS.sci

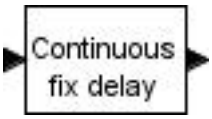
## Computational function

- SCI/modules/scicos_blocks/src/c/tcslti4.c (Type 4)

# Nome

TIME_DELAY — Time delay

## Block Screenshot




## Contents


- Time delay
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

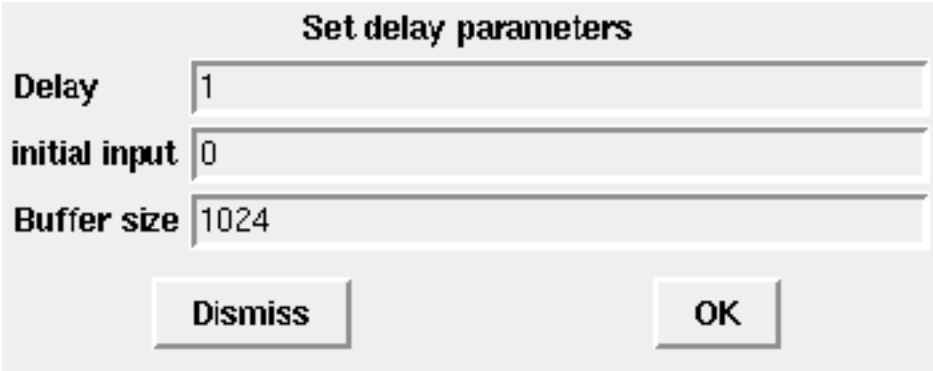
- Continuous time systems palette

## Description

The Transport Delay block delays the input by a specified amount of time. It can be used to simulate a time delay. At the start of the simulation, the block outputs the Initial input parameter until the simulation time exceeds the Time delay parameter, when the block begins generating the delayed input.

The Time delay parameter must be non-negative.

## Dialog box




- **Delay**

  The amount of simulation time that the input signal is delayed before being propagated to the output. The value must be nonnegative.

Properties : Type 'vec' of size 1

- **initial input**

  The output generated by the block between the start of the simulation and the Time delay.

  Properties : Type 'vec' of size 1

- **Buffer size**

  The initial memory allocation for the number of points to store.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *time_delay*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/TIME_DELAY.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/time_delay.c (Type 4)

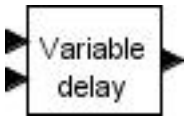
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

VARIABLE_DELAY — Variable delay

## Block Screenshot




## Contents


- Variable delay
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

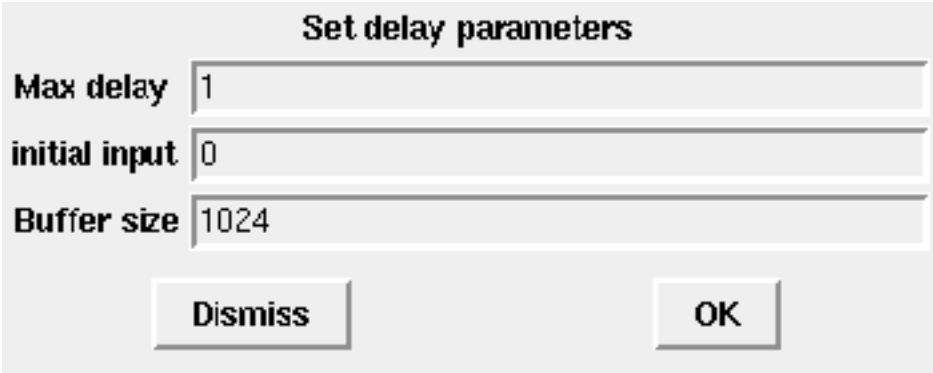

## Palette

- Continuous time systems palette

## Description

The Variable Transport Delay block can be used to simulate a variable time delay. The block might be used to model a system with a pipe where the speed of a motor pumping fluid in the pipe is variable. The block accepts two inputs: the first input is the signal that passes through the block; the second input is the time delay.

## Dialog box




- **Max delay**

  It defines the largest value the time delay input can have.The value cannot be negative.

Properties : Type 'vec' of size 1.

- **initial input**

  The output generated by the block until the simulation time first exceeds the time delay input.

  Properties : Type 'vec' of size 1.

- **Buffer size**

  The number of points the block can store.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *variable_delay*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/VARIABLE_DELAY.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/variable_delay.c (Type 4)

## Authors

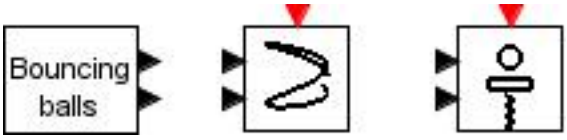
**Ramine Nikoukhah** - INRIA

# 4. Demonstrations blocks palette

# Nome

Demonstrationsblocks_pal — Demonstrations blocks palette

## Block Screenshot




## Module

- xcos

## Description

The Demonstrations blocks palette contains blocks used in some Xcos demonstration diagrams.

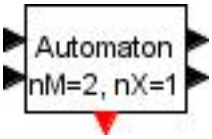
## Blocks

- AUTOMAT — hybrid automata
- BOUNCE - Balls coordinates generator
- BOUNCEXY — Balls viewer
- BPLATFORM — Balls under a platform viewer
- PDE — 1D PDE block

# Nome

AUTOMAT — automata (finite state machine)

## Block Screenshot




## Contents


- automata (finite state machine)
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Demonstrations blocks palette

## Description

This block gives the possibility to construct hybrid automata, i.e., a hybrid system whose discrete part is defined via modes and transitions between modes, and the continuous part is defined via DAE (differential algebraic equations).

The automaton block provides a switching mechanism between subsystems corresponding to control modes of an automaton. Subsystems are constructed in such a way that they have the state vector as input ( coming from the automaton block) and compute the flow and jump functions (zero-crossing) and pass them back to the automaton block. The state variables are defined in the automaton block and the subsystems are static functions.

Suppose that a hybrid automaton consists of $M$ control modes. The continuous-time dynamics in mode is defined with DAE ( ) where and the dimension of is () for any . Suppose that in control mode , there are jump conditions indicating jumps toward other modes. The jump conditions are defined by functions where .

When a jump function changes sign and becomes positive, a mode transition will happen. When transition function becomes positive, a transition to mode happens and state vector is reset to , for .

In order to develop an automaton containing a mode with multiple reset functions, the value of the current and previous active modes should be used. These values are available at the first output port of the block.

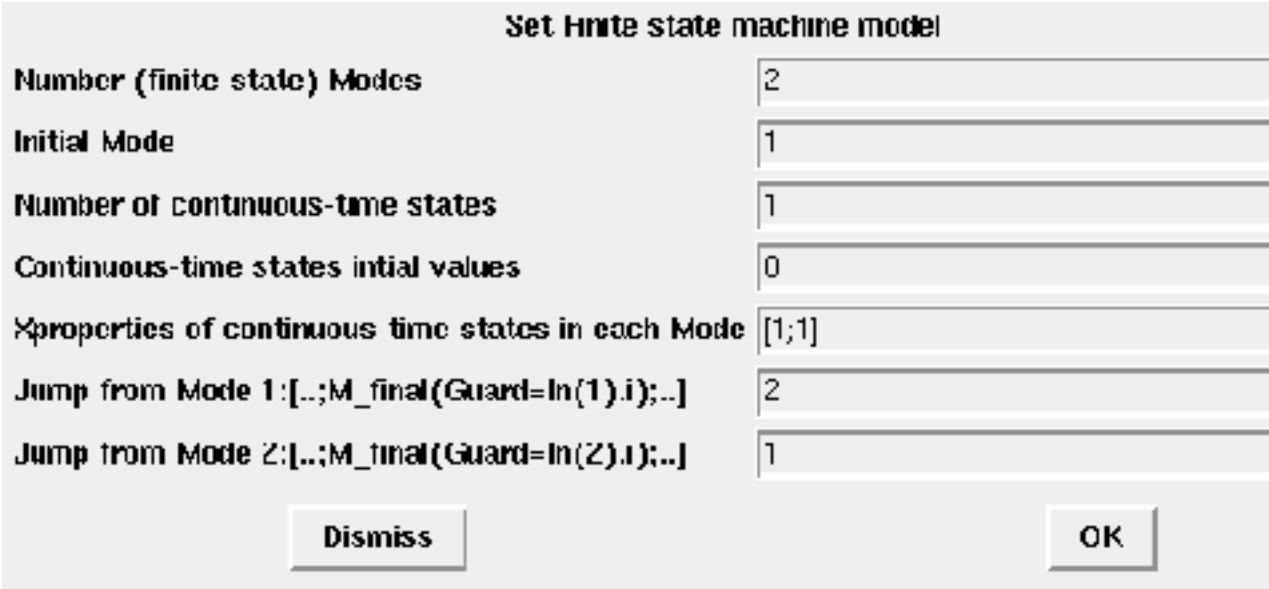
The automaton block has the following input/output ports.

- **Output 1:** The first output port is a vector of size two consisting of the current and the previous active control modes, i.e., $OUT_1 = [current - mode,\ previous - mode]^t$ .
- **Output 2:** The second output port is a vector of size $2N$ providing the state vector and its first time derivative, $[x, \dot{x}]^t$ .
- **Inputs:** The automaton block has $M$ vector input ports corresponding to $M$ modes or subsystems of the automaton. Each input defines the dynamic behavior in the control each mode as well as the reset functions and the transition functions. The input port $i$ which is the output of the $i^{th}$ subsystem is a vector of size $2N + Z_i$ . Each input is composed of the following vector.
  - The first $N$ elements of the $Input_i$ are the continuous-time dynamics. The dynamics of the system in the control mode $i$ is described by a smooth index-1 DAE ( $0 = F_i(\dot{x}, x, u, t)$ ).
  - The next $N$ elements of $Input_i$ are the values used to reset the continuous-time state when a transition to control mode $i$ is activated.
  - The next $Z_i$ elements of $Input_i$ are the jump or zero-crossing functions. If the $j^{th}$ zero-crossing function of mode $i$ crosses zero with negative to positive direction, a transition to $j^{th}$ destination mode happens.
- **Event Output:** This is an event output port, which is activated whenever a mode transition happens. This event is useful when an event is needed to activate or initialize a part of the subsystem not included in the internal dynamics of the automaton block.

In the interface window, the number of control modes, the initial control mode and the initial value of continuous-time state at the beginning of the simulation should be given.

Find more documentation and demos about the Automaton block oat www.scicos.org. Interested users are referred to the paper "Modeling Hybrid Automata in Scicos", Masoud Najafi, Ramine Nikoukhah, 2007 IEEE Multi-conference on Systems and Control, Singapore.

## Dialog box




- **Number of (finite-state) Modes**

  Number of modes in the automation.

  Properties : Type 'vec' of size [1,1].

- **Initial Mode**

  Initial active mode at the beginning of the simulation.

  Properties : Type 'vec' of size [1,1].

- **Number of continuous-time states** Number of continuous-time states at modes. Note that the number of continuous-time states is the same in all modes.

  Properties : Type 'vec' of size [-1,1].

- **Continuous-time states initial values**

  Initial value of continuous-time states at the beginning of the simulation.

  Properties : Type 'vec' of size [-1,1].

- **Xproperties of continuous-time states in each Mode**

  In this field the state types in mode are given. A state in an index 1 DAE can be either differential state or algebraic state. vector is coded in an M*N matrix, where M is the number of modes and N is the number of states. This matrices indicates whether a continuous-time state is algebraic or differential in each control mode. If in the $i^{th}$ mode, $j^{th}$ state is differential, the (i,j)-th element of the Xproperty matrix should set to "+1", otherwise it should set to "-1". Xproperty can be given as a 1*N vector if type of states remain the same in all modes.

  Properties : Type 'mat' of size [-1,-1].

- **Jump from Mode 1:[..;M_final(Guard=In(1).i);..]**

The fields express the mode transition information. Suppose that all control modes are labeled from 1 to M. Then, in the field corresponding to control mode**i** , destination modes of mode**i** are defined in a vector.**j-th** element of this vector gives the destination mode when**j-th** jump function :

becomes positive. For example, if in the field of the mode**2** , the user defines [1;3;4], it means that in mode**2** , there are three active jump functions. When, for example, the third jump function becomes positive, a mode transition to mode**4** will be activated.

Properties : Type 'vec' of size [-1,1].

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**

  **- port 1 : size [3,1] / type 1**

  **- port 2 : size [3,1] / type 1**
- **regular outputs:**

  **- port 1 : size [2,1] / type 1**

  **- port 2 : size [2,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *automat*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/AUTOMAT.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/automat.c (Type 10004)

## Authors

**Masoud Najafi** - INRIA

# Nome

BOUNCE — Balls coordinates generator

## Block Screenshot

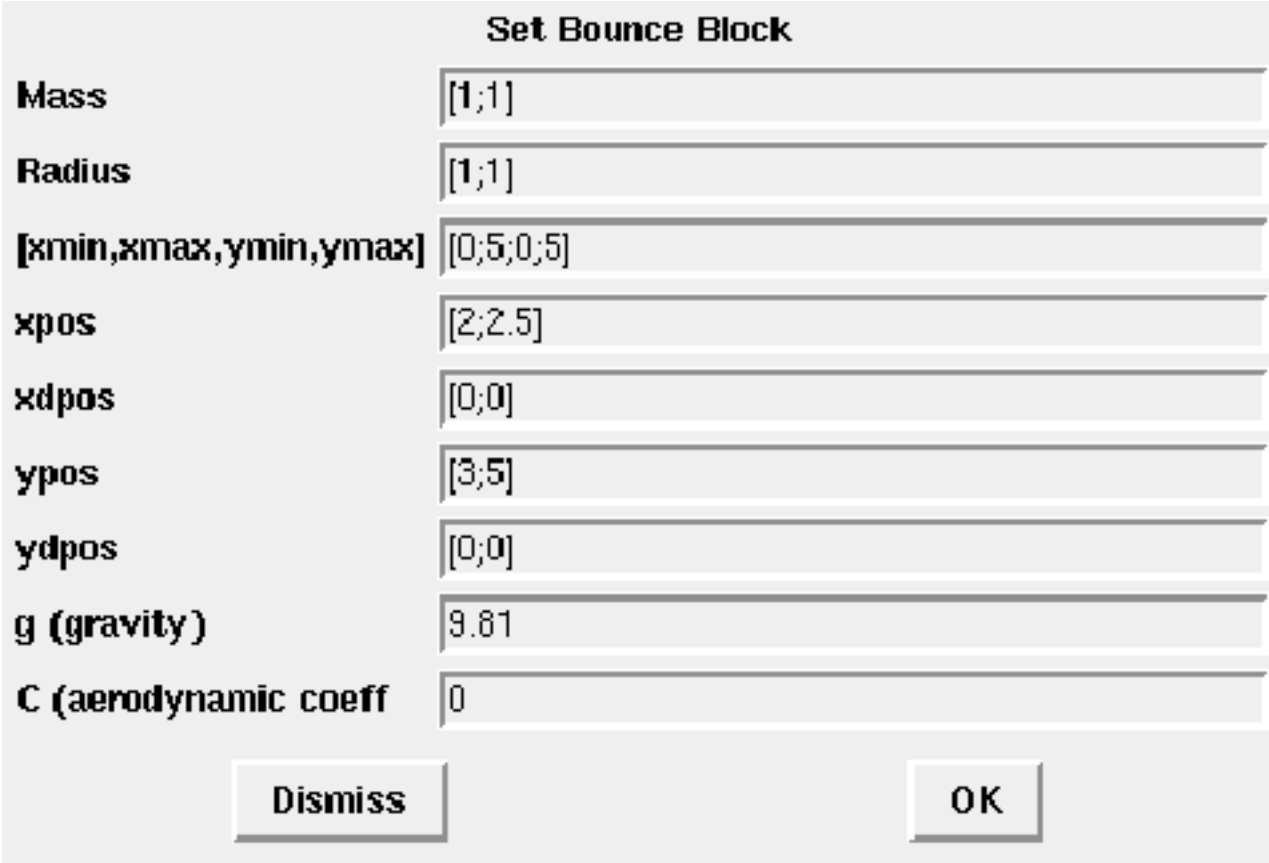
## Contents

- Balls coordinates generator
- • “Palette”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

## Palette

- Demonstrations blocks palette

## Dialog box



- **Mass**

  The parameter description 1.

  Properties : Type 'vec' of size -1.

- **Radius**

  The parameter description 2.

  Properties : Type 'vec' of size -1.

- **[xmin,xmax,ymin,ymax]**

  The parameter description 3.

  Properties : Type 'vec' of size -1.

- **xpos**

  The parameter description 4.

  Properties : Type 'vec' of size -1.

- **xdpos**

  The parameter description 5.

  Properties : Type 'vec' of size -1.

- **ypos**

  The parameter description 6.

  Properties : Type 'vec' of size -1.

- **ydpos**

  The parameter description 7.

  Properties : Type 'vec' of size -1.

- **g (gravity)**

  The parameter description 8.

  Properties : Type 'vec' of size 1.

- **C (aerodynamic coeff**

  The parameter description 9.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** yes
- **mode:** no

- **regular outputs:**

  **- port 1 : size [2,1] / type 1**

  **- port 2 : size [2,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *bounce_ball*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/BOUNCE.sci

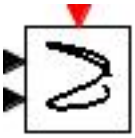
## Computational function

- SCI/modules/scicos_blocks/src/c/bounce_ball.c (Type 4)

# Nome

BOUNCEXY — Balls viewer

## Block Screenshot



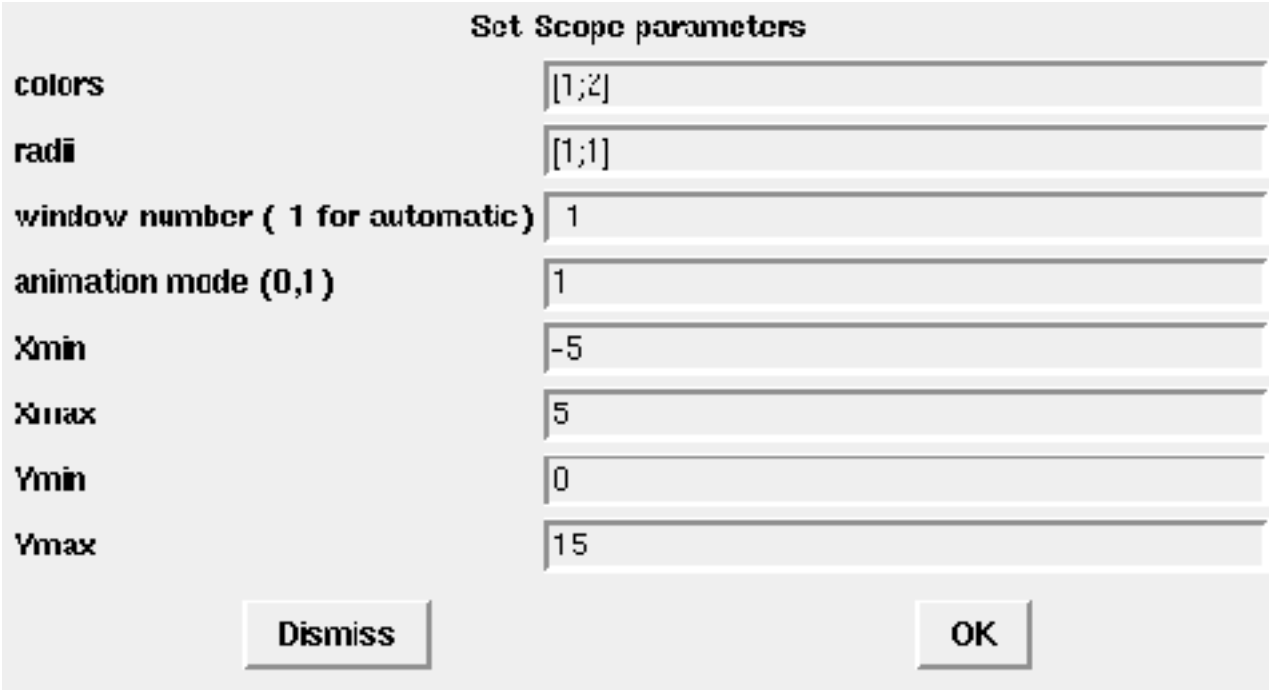
## Contents


- Balls viewer
- • “Palette”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Demonstrations blocks palette

## Dialog box




- **colors**

The parameter description 1.

Properties : Type 'vec' of size -1.

- **radii**

  The parameter description 2.

  Properties : Type 'vec' of size -1.

- **window number (-1 for automatic)**

  The parameter description 3.

  Properties : Type 'vec' of size 1.

- **animation mode (0,1)**

  The parameter description 4.

  Properties : Type 'vec' of size 1.

- **Xmin**

  The parameter description 5.

  Properties : Type 'vec' of size 1.

- **Xmax**

  The parameter description 6.

  Properties : Type 'vec' of size 1.

- **Ymin**

  The parameter description 7.

  Properties : Type 'vec' of size 1.

- **Ymax**

  The parameter description 8.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**

  **- port 2 : size [-1,1] / type 1**

- **number/sizes of activation inputs:** 1

- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *bouncexy*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/BOUNCEXY.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/bouncexy.c (Type 4)

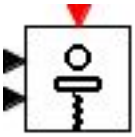
## Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

# Nome

BPLATFORM — Balls under a platform viewer

## Block Screenshot



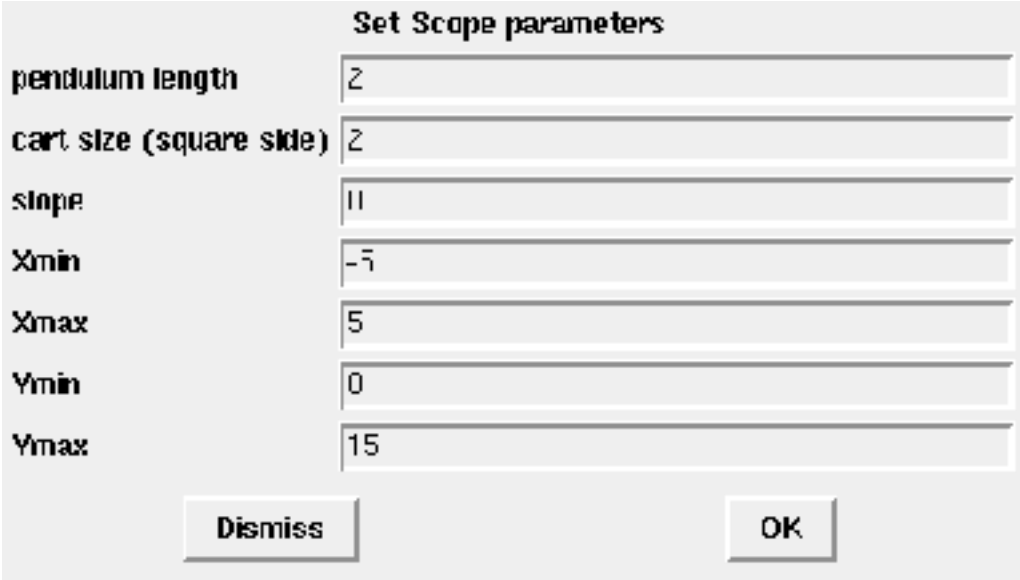
## Contents


- Balls under a platform viewer
- • “Palette”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”


## Palette

- Demonstrations blocks palette

## Dialog box




## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no

- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *bplatform2*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/BPLATFORM.sci

## Authors

**M. Najafi** - INRIA

# Nome

PDE — 1D PDE block

## Block Screenshot

## Contents


- 1D PDE block
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

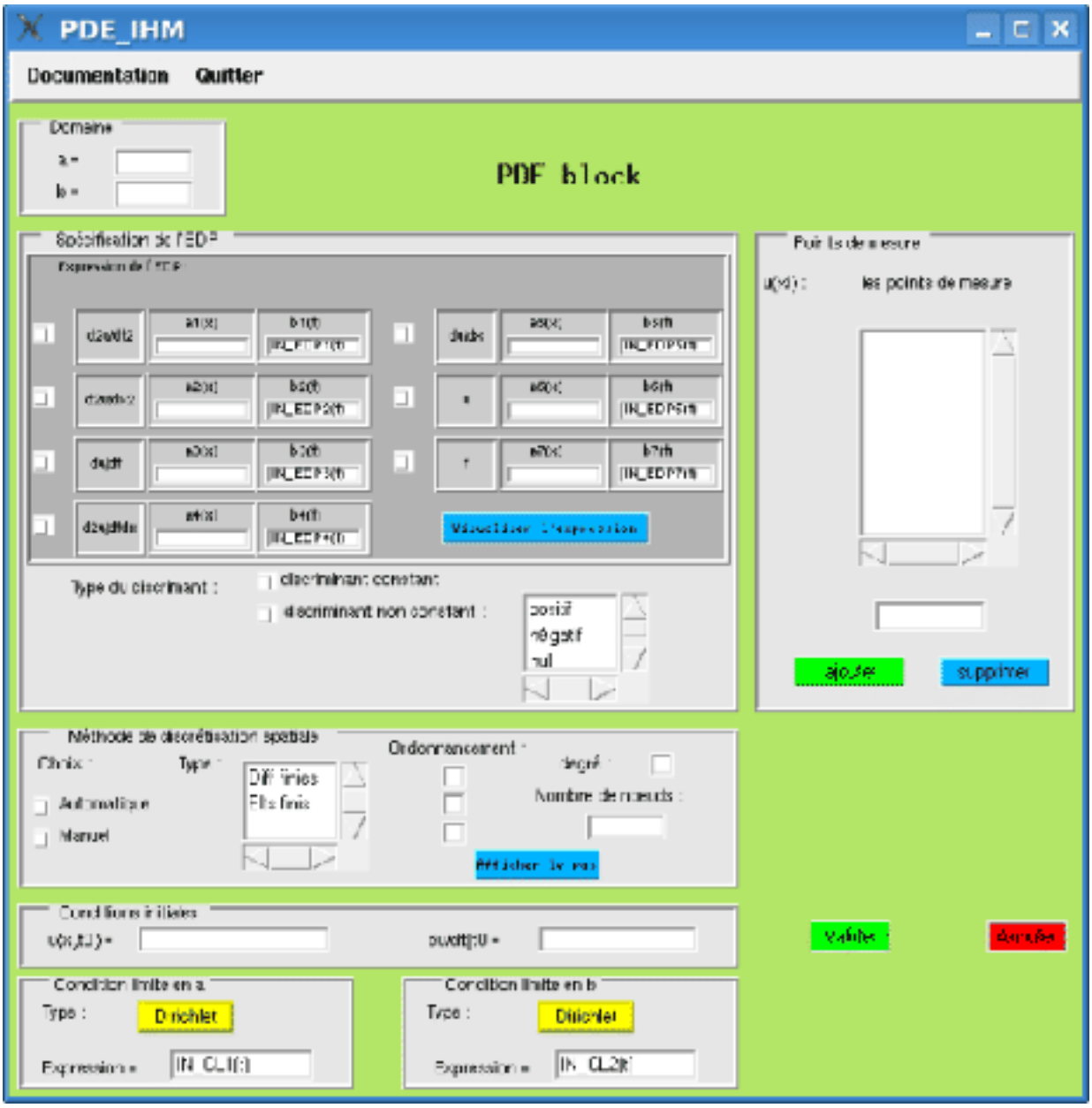

## Palette

- Demonstrations blocks palette

## Description

This block is an implementation of several numerical schemes (Finite Elements (1st and 2nd order), Finite Differences (1st and 2nd order), Finite Volumes (1st order)) to solve mono dimensional PDE (Partial Differential Equation) within SCICOS. The mathematical framwork was restricts in PDEs linear scalars with maximum order 2 in time and space. The goal is to provide engineers and physicists with an easy to use toolbox in SCICOS that will let them graphically describe the PDE to be solved. A decision system selects the most efficient numerical scheme depending on the type of the PDE and runs the solvers.

# Dialog box




- **a et b**

  (double) The two edges of the discretization field.

- **specification de l'EDP**

  check box to select the PDE operators. ai(x), bi(t) (i=1:7) are the operator coefficients. type of PDE discriminant (constant or variable, in the later case, the sign should be given)

- **Discretization methode**

  choix (check box) : is the choice for the manual or the automatic mode. type : in the manual mode we can give the method type (Finite differences, finite elements or finite volumes). degré : method degre (1 or 2 for the FD and FE methods, 1 for the FV method). Nombre de noeuds : to give the number of the nodal points.

- **Conditions initiales**

  u(x,t0)=, du/dt at t0= : to give the initial conditions.

- **Conditions aux limites**

  type : two type of the boundray conditions are possible : Dirichlet or Neumann. expressions : to give then boundray conditions expressions.

- **Points de mesures**

  To give the list of mesurment points.

- **Name**

  A getvalue box to give the block name's.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 0**

  **- port 3 : size [1,1] / type 0**

  **- port 4 : size [1,1] / type 0**

  **- port 5 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [10,1] / type 1**

  **- port 2 : size [0,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *PDE*

## Interfacing function

- SCI/modules/scicos_blocks/macros/PDE/PDE.sci

## Authors

- EADS 2005-01-16

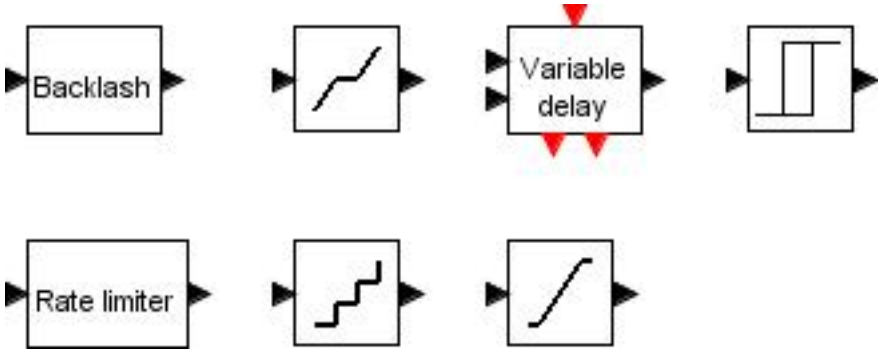
EADS 2005-01-16

# 5. Discontinuities palette

# Nome

discontinuities_pal — discontinuities palette

## Block Screenshot




## Module

- xcos

## Description

discontinuities palette includes blocks whose outputs are discontinuities functions of their inputs.

## Blocks

- BACKLASH - Backlash
- DEADBAND - Deadband
- DELAYV_f — Variable time delay
- HYSTHERESIS — Hystheresis
- RATELIMITER - Rate limiter
- QUANT_f - Quatization
- SATURATION — Saturation

# Nome

BACKLASH — Backlash

## Block Screenshot

## Contents


- Backlash
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

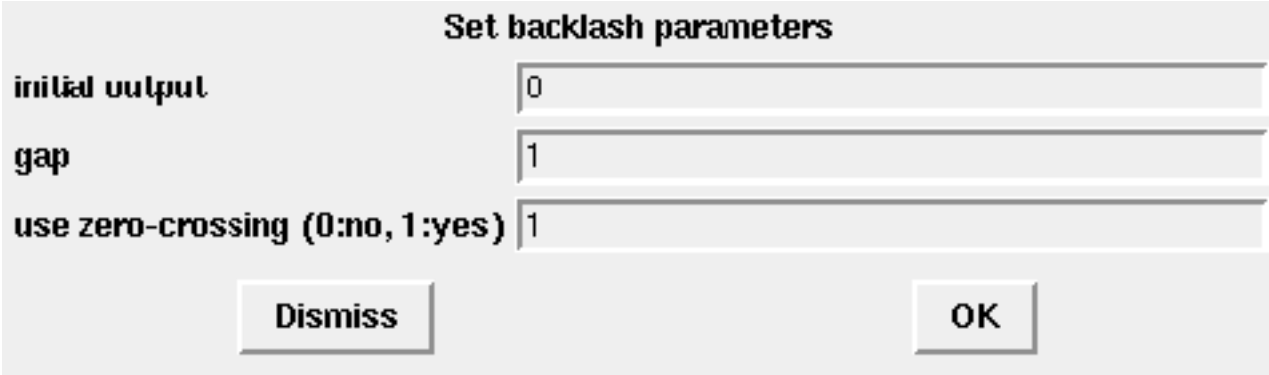

## Palette

- discontinuities palette

## Description

The Backlash block implements a system in which a change in input causes an equal change in output. However, when the input changes direction, an initial change in input has no effect on the output. The amount of side-to-side play in the system is referred to as the .

## Dialog box




- **initial output**

  The initial output value.

  Properties : Type 'vec' of size 1

- **gap**

  The width of the dead-band.

  Properties : Type 'vec' of size 1

- **use zero-crossing**

  Select to enable use of zero crossing detection to detect engagement with lower and upper thresholds.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *backlash*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/BACKLASH.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/backlash.c (Type 4)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

DEADBAND — Deadband

## Block Screenshot

## Contents


- Deadband
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

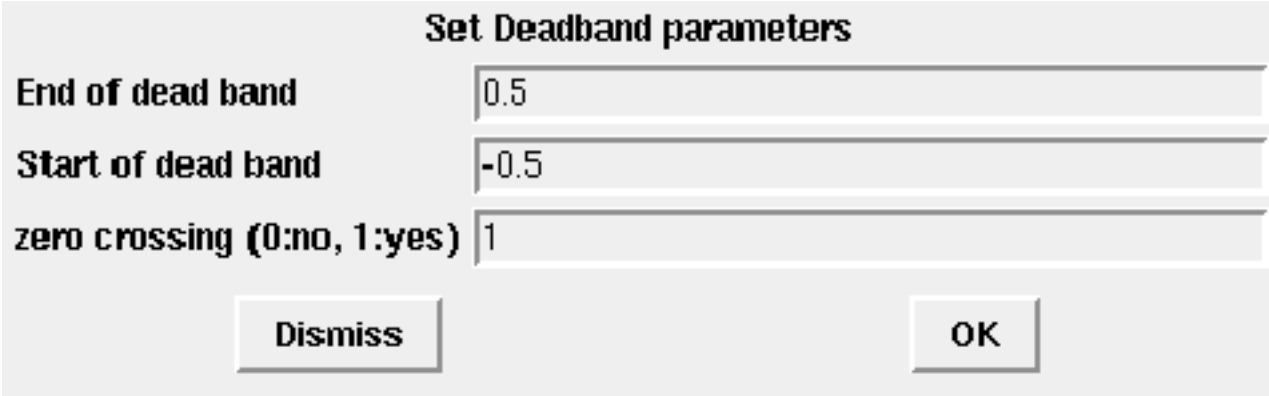

## Palette

- discontinuities palette

## Description

Provide a region of zero output

## Dialog box




- **End of dead band**

  The upper limit of the dead band.

  Properties : Type 'vec' of size 1

- **Start of dead band**

The lower limit of the dead band.

Properties : Type 'vec' of size 1

- **zero crossing**

  Select to enable zero crossing detection to detect when the limits are reached.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *deadband*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/DEADBAND.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/deadband.c (Type 4)

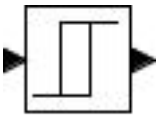
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

HYSTHERESIS — Hystheresis

## Block Screenshot



## Contents


- Hystheresis
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

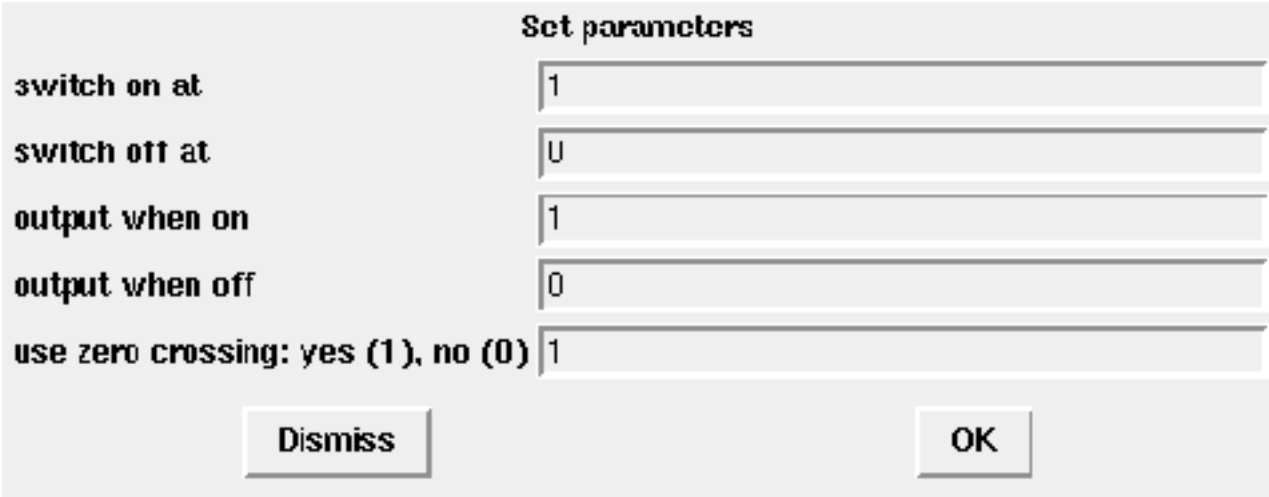

## Palette

- discontinuities palette

## Description

Switch output between two constants.The Relay block allows its output to switch between two specified values. When the relay is on, it remains on until the input drops below the value of the Switch off point parameter. When the relay is off, it remains off until the input exceeds the value of the Switch on point parameter. The block accepts one input and generates one output.

## Dialog box



- **switch on at**

  The Switch on point parameter is converted to the input data type offline using round-to-nearest and saturation.

  Properties : Type 'vec' of size 1

- **switch off at**

  The Switch off point parameter is converted to the input data type offline using round-to-nearest and saturation.

  Properties : Type 'vec' of size 1

- **output when on**

  The output when the relay is on.

  Properties : Type 'vec' of size 1

- **output when off**

  The output when the relay is off.

  Properties : Type 'vec' of size 1

- **use zero crossing: yes**

  Select to enable zero crossing detection to detect switch-on and switch-off points.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *hystheresis*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/HYSTHERESIS.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/hystheresis.c (Type 4)

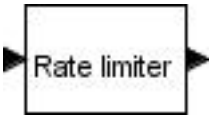
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

RATELIMITER — Rate limiter

## Block Screenshot




## Contents


- Rate limiter
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

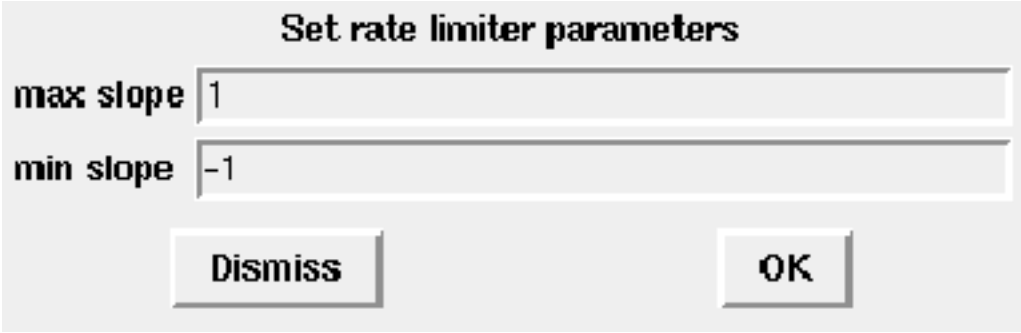

## Palette

- discontinuities palette

## Description

The Rate Limiter block limits the first derivative of the signal passing through it. The output changes no faster than the specified limit.

## Dialog box




- **max slope**

  The limit of the derivative of an increasing input signal.

  Properties : Type 'vec' of size 1

- **min slope**

The limit of the derivative of a decreasing input signal.

Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *ratelimiter*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/RATELIMITER.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/ratelimiter.c (Type 4)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

SATURATION — Saturation

## Block Screenshot

## Contents


- Saturation
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

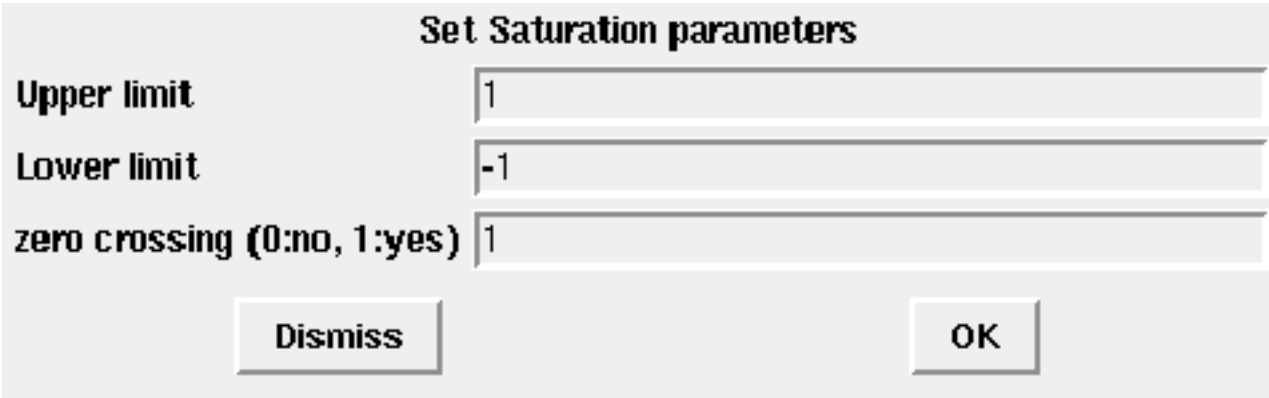

## Palette

- discontinuities palette

## Description

The Saturation block imposes upper and lower bounds on a signal. When the input signal is within the range specified by the Lower limit and Upper limit parameters, the input signal passes through unchanged. When the input signal is outside these bounds, the signal is clipped to the upper or lower bound. When the Lower limit and Upper limit parameters are set to the same value, the block outputs that value.

## Dialog box




- **Upper limit**

Specify the upper bound on the input signal. When the input signal to the Saturation block is above this value, the output of the block is clipped to this value.

Properties : Type 'vec' of size 1

- **Lower limit**

  Specify the lower bound on the input signal. When the input signal to the Saturation block is below this value, the output of the block is clipped to this value.

  Properties : Type 'vec' of size 1

- **zero crossing**

  Select to enable zero crossing detection.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *satur*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/SATURATION.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/satur.c (Type 4)

## Authors

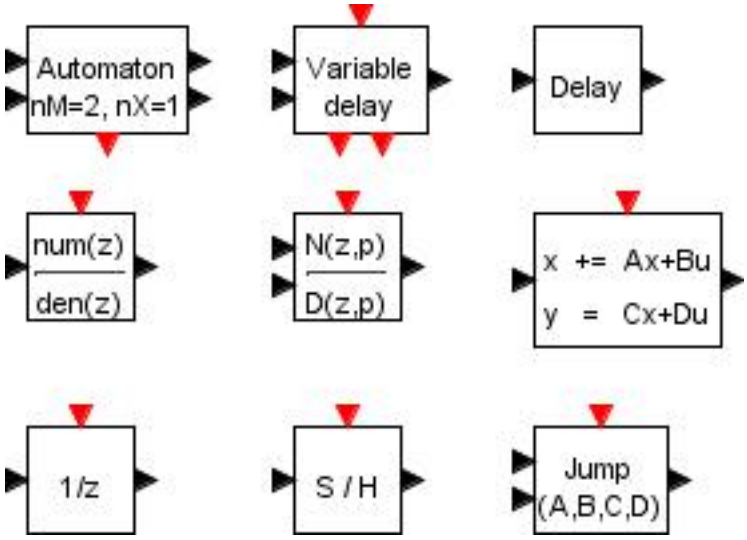
**Ramine Nikoukhah** - INRIA

# 6. Discrete time systems palette

# Nome

Discrete_pal — Discrete time systems palette

## Block Screenshot




## Module

- xcos

## Description

The Discrete time systems palette contains blocks for medeling discrete time systems.

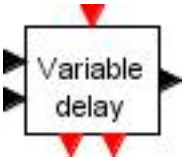
## Blocks

- DELAY_f — Discrete time delay
- DELAYV_f — Variable delay
- DLR — Discrete transfer function
- DLRADAPT_f — Discrete Zero-Pole function
- DLSS — Discrete state-space system
- DOLLAR_f — Delay operator
- REGISTER — Shift Register
- SAMPHOLD_m — Sample and hold
- TCLSS — Continuous linear system with jump

# Nome

DELAYV_f — Variable delay

## Block Screenshot




## Contents


- Variable delay
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

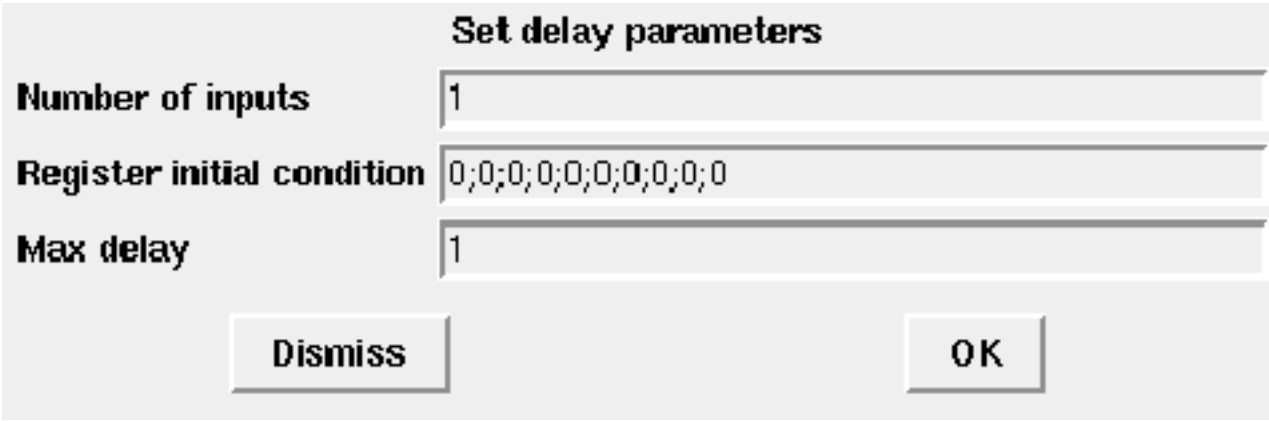

## Palette

- Discrete time systems palette

## Description

Add here a paragraph of the function description

## Dialog box




- **Number of inputs**

  Set the vector size of the first regular input and the vector size of the regular output port.

  Properties : Type 'vec' of size 1.

- **Register initial condition**

  Set the length and the initial conditions of the register.

  Properties : Type 'vec' of size -1.

- **Max delay**

  It defines the largest value the time delay input can have.The value cannot be negative.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 2
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *delayv*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DELAYV_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/delayv.f (Type 1)

# Nome

DELAY_f — Discrete time delay

## Block Screenshot

## Contents

- Discrete time delay
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content

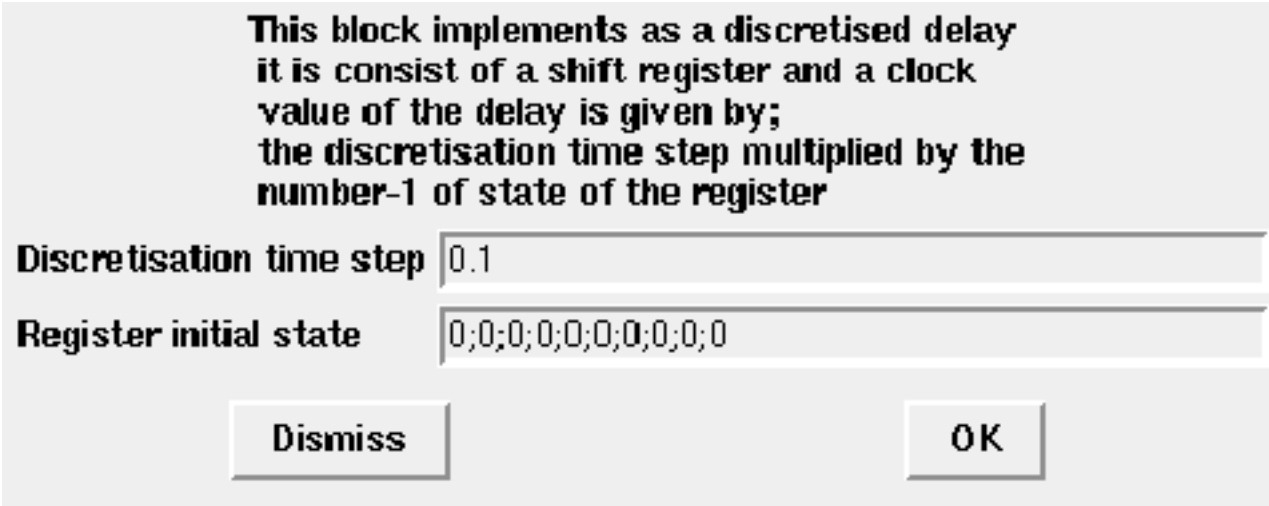
## Palette

- Discrete time systems palette

## Description

This compiled super-block implements a discretized delay It is build with a shift register and a clock. The value of the delay is given by the discretization time step multiplied by the number-1 of state of the register.

## Dialog box




- **Discretization time step**

  Set the time period of the integrated clock.

Properties : Type 'vec' of size 1.

- **Register initial state**

  Set the length and the initial conditions of the register.

  Properties : Type 'vec' of size -1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

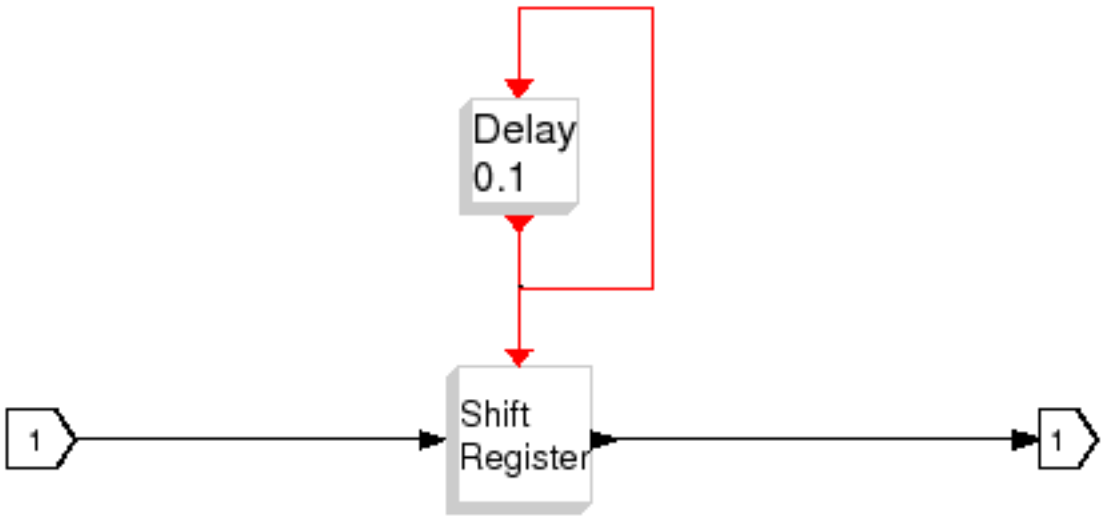
  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DELAY_f.sci

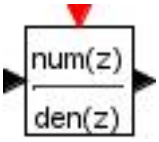
## Compiled Super Block content



# Nome

DLR — Discrete transfer function

## Block Screenshot




## Contents


- Discrete transfer function
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

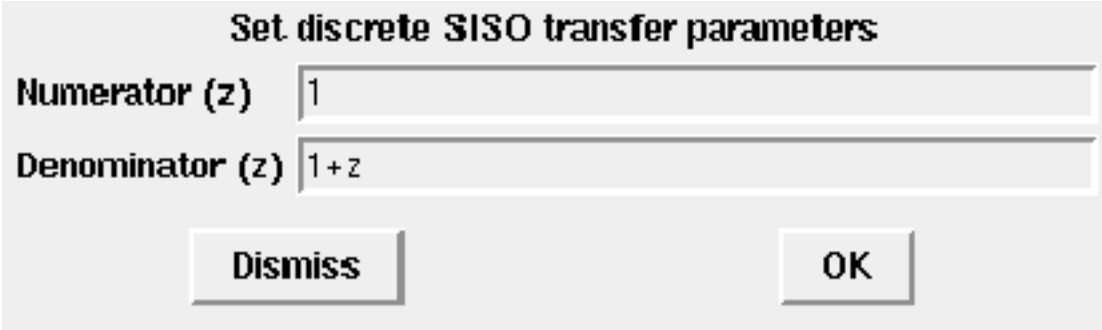

## Palette

- Discrete time systems palette

## Description

This block realizes a SISO linear system represented by its rational transfer function (in the symbolic variable z). The rational function must be proper.

## Dialog box




- **Numerator (z)**

  This parameter sets the numerator of the transfer function.

  This must be a polynomial in**z** .

  Properties : Type 'pol' of size 1.

- **Denominator (z)**

This parameter sets the denominator of the transfer function.

This must be a polynomial in**z** .

Properties : Type 'pol' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *dsslti4*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DLR.sci

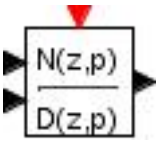
## Computational function

- SCI/modules/scicos_blocks/src/c/dsslti4.c (Type 4)

# Nome

DLRADAPT_f — Discrete Zero-Pole

## Block Screenshot




## Contents

- Discrete Zero-Pole
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

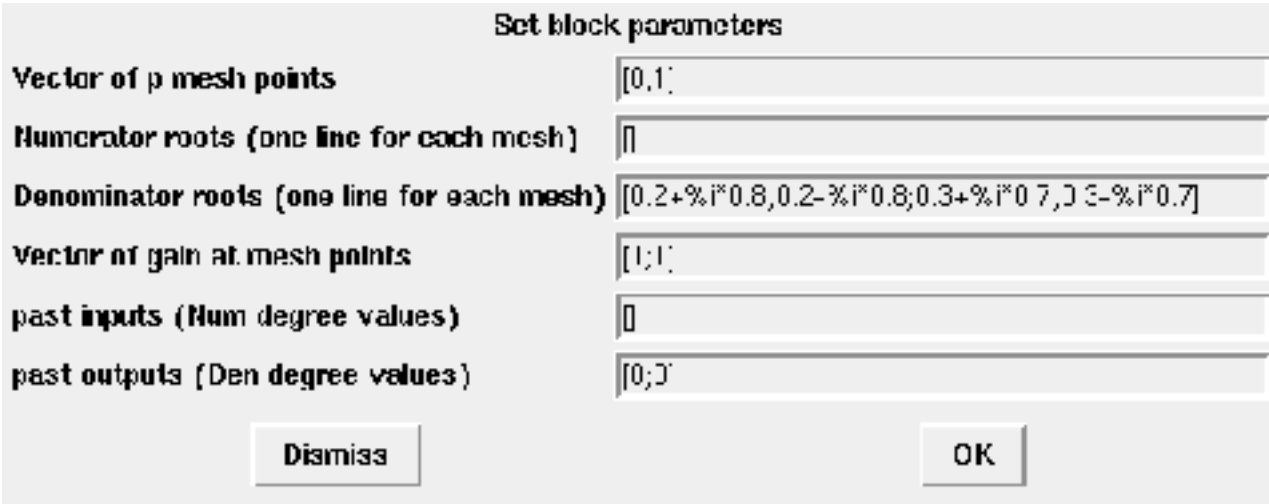
## Palette

- Discrete time systems palette

## Description

This block models system represented by zeros and poles of discrete transfer function.

## Dialog box




- **Vector of p mesh points**

  The parameter description 1.

Properties : Type 'vec' of size -1.

- **Numerator roots (one line for each mesh)**

  The parameter description 2.

  Properties : Type 'mat' of size [-1,-1].

- **Denominator roots (one line for each mesh)**

  The parameter description 3.

  Properties : Type 'mat' of size ["size(%1,"*")","-1"].

- **Vector of gain at mesh points**

  The parameter description 4.

  Properties : Type 'vec' of size "size(%1,"*")".

- **past inputs (Num degree values)**

  The parameter description 5.

  Properties : Type 'vec' of size "size(%2,2)".

- **past outputs (Den degree values)**

  The parameter description 6.

  Properties : Type 'vec' of size "size(%3,2)".

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no

- **name of computational function:** *dlradp*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/DLRADAPT_f.sci

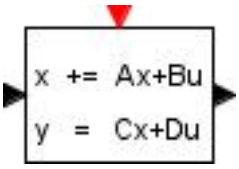
## Computational function

- SCI/modules/scicos_blocks/src/fortran/dlradp.f (Type 0)

# Nome

DLSS — Discrete state-space system

## Block Screenshot




## Contents


- Discrete state-space system
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

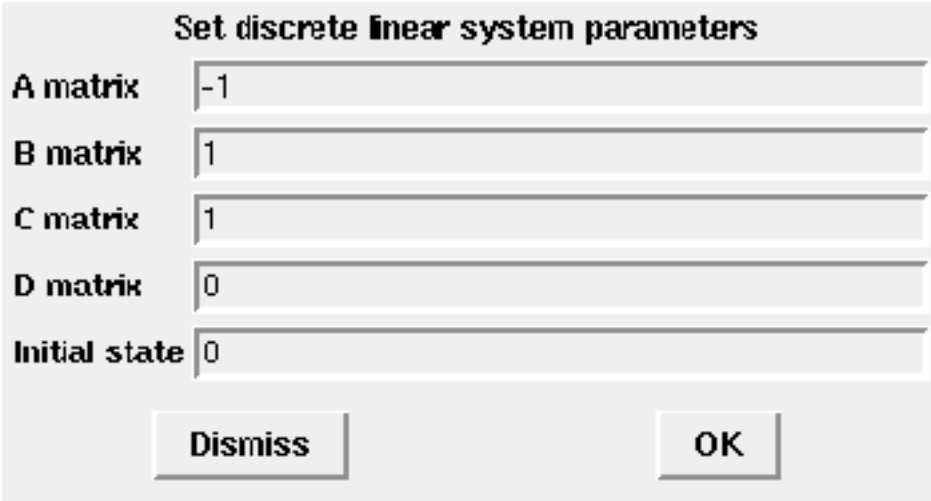

## Palette

- Discrete time systems palette

## Description

This block realizes a discrete-time linear state-space system. The system is defined by the matrices and the initial state . The dimensions must be compatible. At the arrival of an input event on the unique input event port, the state is updated.

## Dialog box




- $A$ **matrix**

A square matrix.

Properties : Type 'mat' of size [-1,-1].

- **B matrix**

  The $B$ matrix, [] if system has no input.

  Properties : Type 'mat' of size ["size(%1,2)","-1"].

- **C matrix**

  The $C$ matrix , [] if system has no output.

  Properties : Type 'mat' of size ["-1","size(%1,2)"].

- **D matrix**

  The $D$ matrix, [] if system has no D term.

  Properties : Type 'mat' of size [-1,-1].

- **Initial state**

  A vector/scalar initial state of the system.

  Properties : Type 'vec' of size "size(%1,2)".

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *dsslti4*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DLSS.sci

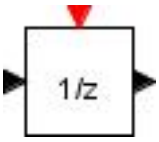
## Computational function

- SCI/modules/scicos_blocks/src/c/dsslti4.c (Type 4)

# Nome

DOLLAR_f — Delay operator

## Block Screenshot




## Contents


- Delay operator
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

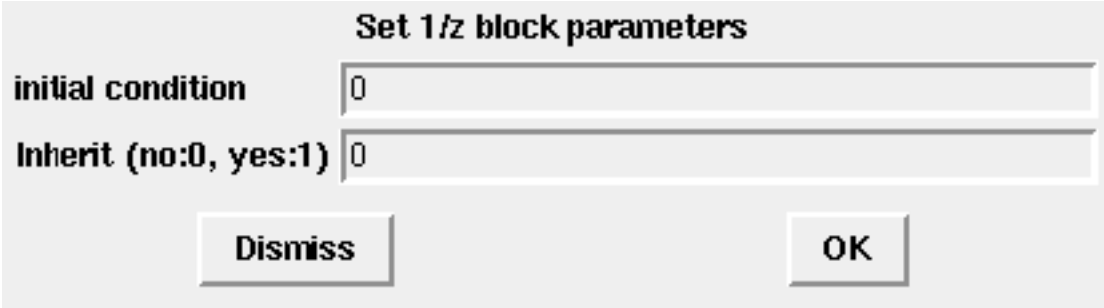

## Palette

- Discrete time systems palette

## Description

The Unit Delay block delays its input by the specified sample period. This block is equivalent to the z-1 discrete-time operator. The block accepts one input and generates one output, which can be either both scalar or both vector. If the input is a vector, all elements of the vector are delayed by the same sample period.

## Dialog box




- **initial condition**

  The output of the simulation for the first sampling period, during which the output of the Unit Delay block is otherwise undefined.

  Properties : Type 'vec' of size -1.

- **Inherit (no:0, yes:1)**

  When "Inherit" is yes, the block inherit its event input.

  Properties : Type 'vec' of size -1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *dollar*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/DOLLAR_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/dollar.f (Type 0)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

REGISTER — Shift Register

## Block Screenshot

## Contents


- Shift Register
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

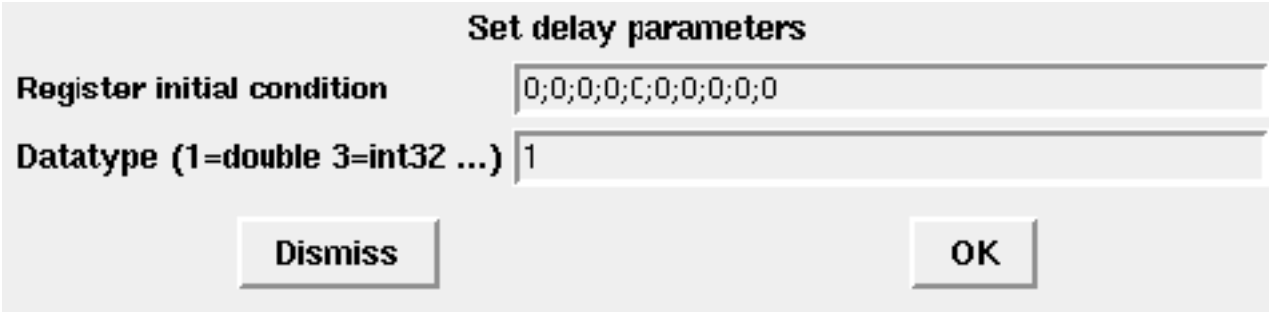

## Palette

- Discrete time systems palette

## Description

This block realizes a shift register. At every input event, the register is shifted one step.

## Dialog box




- **Register initial condition**

  A column vector. It contains the initial state of the register.

  Properties : Type 'vec' of size -1

- **Datatype**

  This block support all datatypes besides complex.

Properties : Type 'vec' of size -1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *delay4*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/REGISTER.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/delay4_i32.c
- SCI/modules/scicos_blocks/src/c/delay4_i16.c
- SCI/modules/scicos_blocks/src/c/delay4_i8.c
- SCI/modules/scicos_blocks/src/c/delay4_ui32.c
- SCI/modules/scicos_blocks/src/c/delay4_ui16.c
- SCI/modules/scicos_blocks/src/c/delay4_ui8.c

## Authors

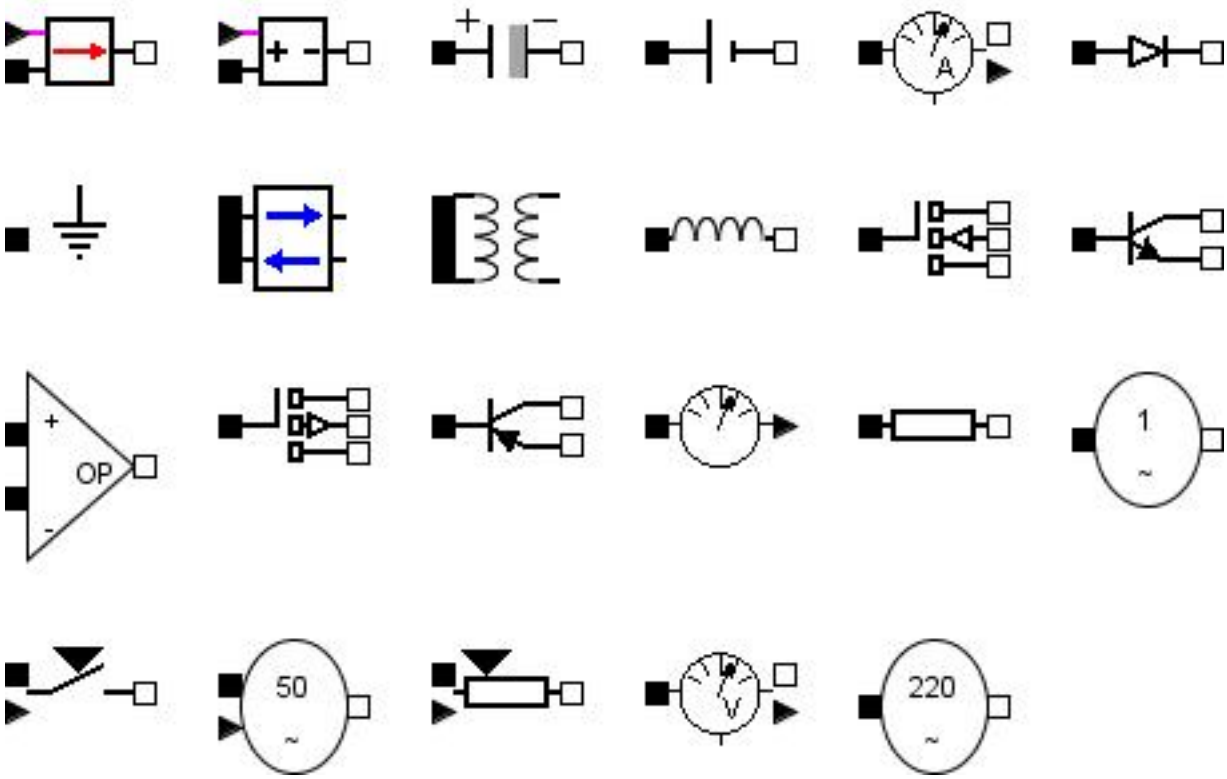
**Fady NASSIF** - INRIA

# 7. Electrical palette

# Nome

Electrical_pal — Electrical palette

## Block Screenshot




## Module

- xcos

## Description

Electrical toolbox contains very basic electrical components such as voltage source, diode, capacitor, etc.

## Blocks

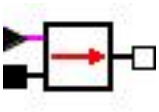
- Capacitor - Electrical capacitor
- CCS - Controllable Modelica current source
- ConstantVoltage - Electrical DC voltage source
- CurrentSensor - Electrical current sensor
- CVS - Controllable Modelica voltage source
- Diode - Electrical diode
- Ground - Ground (zero potential reference)
- Gyrator - Modelica Gyrator
- IdealTransformer - Ideal Transformer

- Inductor - Electrical inductor
- NMOS - Simple NMOS Transistor
- NPN - NPN transistor
- OpAmp - Ideal opamp (norator-nullator pair)
- PMOS - Simple PMOS Transistor
- PNP - PNP transistor
- PotentialSensor - Potential sensor
- Resistor - Electrical resistor
- SineVoltage - Sine voltage source
- Switch - Non-ideal electrical switch
- VariableResistor - Electrical variable resistor
- VoltageSensor - Electrical voltage sensor
- VsourceAC - Electrical AC voltage source
- VVsourceAC - Variable AC voltage source

# Nome

CCS — Controllable Modelica current source

## Block Screenshot



## Contents


- Controllable Modelica current source
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - Modelica model
  - “See also”
  - “Authors”


## Palette

- Electrical.cosf - Electrical toolbox

## Description

This block is an ideal current source. The current value is controlled through the explicit input of the block (connected to standard Xcos blocks). The voltage across the block is independent of the current value.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'Iin'

    Explicit **variable.**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**
- **File name of the model :** CCS

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/CCS.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/CCS.mo

## See also

- CVS - Controllable Modelica voltage source

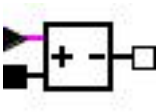
## Authors

**Masoud Najafi** - INRIA

# Nome

CVS — Controllable Modelica voltage source

## Block Screenshot



## Contents


- Controllable Modelica voltage source
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - Modelica model
  - “See also”
  - “Authors”


## Palette

- Electrical.cosf - Electrical toolbox

## Description

This block is an ideal voltage source. The voltage value is controlled through the explicit input of the block (connected to standard Xcos blocks). The current passing through the block is independent of the voltage across the block terminals.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'vin'

    Explicit **variable.**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**
- **File name of the model :** CVS

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/CVS.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/CVS.mo

## See also

- CCS - Controllable Modelica current source

## Authors

**Masoud Najafi** - INRIA

# Nome

Capacitor — Electrical capacitor

## Block Screenshot

## Contents


- Electrical capacitor
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model


## Palette

- Electrical.cosf - Electrical toolbox

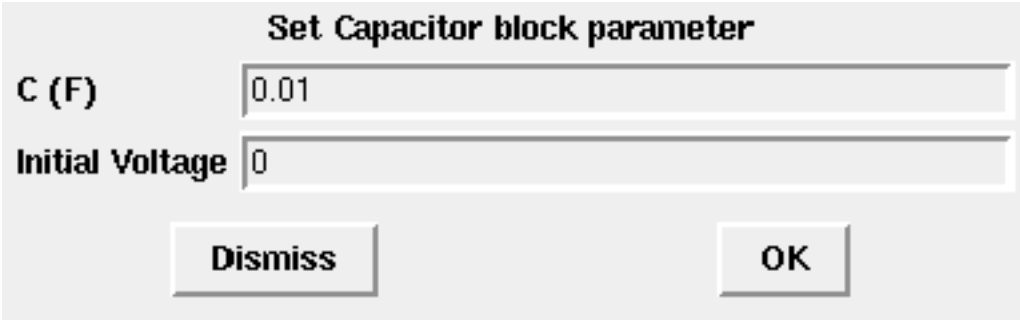
## Description

A capacitor is an electrical component that can store energy in electrical circuits. The relationship between the voltage across a capacitor with capacitance and the current passing through it is given by the:

$$I = C\frac{dV}{dt}$$

Capacitors can also be used to differentiate between high-frequency and low-frequency signals and this makes them useful in electronic filters. A capacitor has a high impedance when a signal is low frequency signals.

## Dialog box



- **C (F)**

  Capacitance

  Properties : Type 'vec' of size 1.

- **Initial Voltage**

  Initial Voltage

  Properties : Type 'vec' of size 1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'C'

    **Default value :** 0.01

    **Is a state variable :** no.
  - **Modelica parameter name :** 'v'

    **Default value :** 0

    **Is a state variable :** yes.
- **File name of the model :** Capacitor

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Capacitor.sci

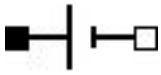
## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Capacitor.mo

# Nome

ConstantVoltage — Electrical DC voltage source

## Block Screenshot



## Contents


- Electrical DC voltage source
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model

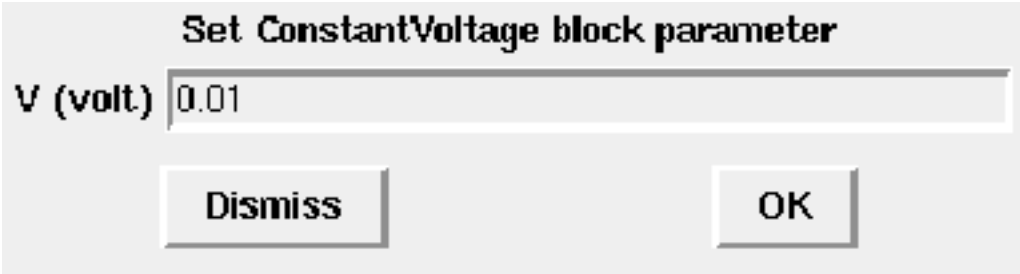

## Palette

- Electrical.cosf - Electrical toolbox

## Description

This component is a model for any device or system that produces a constant electromotive force between its port. The output voltage of this DC voltage source is defined by the user. The black port indicates the positive voltage. The ohmic resistance of this DC voltage source is zero.

## Dialog box




- **V (volt)**

  Output voltage

  Properties : Type 'vec' of size 1.

## Default properties

- **Inputs :**

- **Modelica variable name :** 'p'

  Implicit **variable.**

- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**

- **Parameters :**
  - **Modelica parameter name :** 'V'

    **Default value :** 0.01

    **Is a state variable :** no.

- **File name of the model :** ConstantVoltage

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/ConstantVoltage.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/ConstantVoltage.mo

# Nome

CurrentSensor — Electrical current sensor

## Block Screenshot

## Contents

- Electrical current sensor
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - Modelica model

## Palette

- Electrical.cosf - Electrical toolbox

## Description

This block is inserted in series in an electrical circuit to measure the current passing through the component. The measure is given to the explicit part of the model via an explicit pout. Conventionally, current flowing into the black port is considered positive. The ohmic resistance of this block is zero.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**
  - **Modelica variable name :** 'i'

    Explicit **variable.**
- **File name of the model :** CurrentSensor

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/CurrentSensor.sci

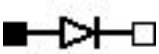
## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/CurrentSensor.mo

# Nome

Diode — Electrical diode

## Block Screenshot



## Contents


- Electrical diode
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model


## Palette

- Electrical.cosf - Electrical toolbox

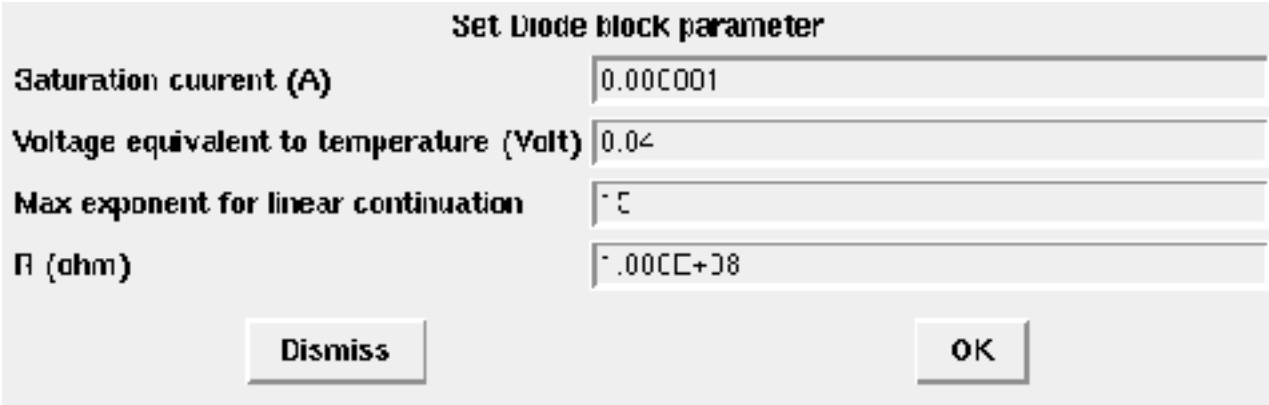
## Description

This component consists of a simple diode parallel with an ohmic resistance (). The current passing through this component is defined as a function of the voltage across the ports, ,

$$i = I_{ds}(\exp^{v/V_t} - 1) + \frac{v}{R}$$

where and are the saturation current and the voltage equivalent of temperature, respectively. If the exponent reaches a certain limit (), the diode characteristic becomes linear to avoid overflow.

## Dialog box



- **Saturation current (A)**

  Saturation current

  Properties : Type 'vec' of size 1.

- **Voltage equivalent to temperature (Volt)**

  Voltage equivalent of temperature

  Properties : Type 'vec' of size 1.

- **Max exponent for linear continuation**

  Max exponent for linear continuation

  Properties : Type 'vec' of size 1.

- **R (ohm)**

  Parallel ohmic resistance.

  Properties : Type 'vec' of size 1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**

- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**

- **Parameters :**
  - **Modelica parameter name :** 'Ids'

    **Default value :** 0.000001

    **Is a state variable :** no.

  - **Modelica parameter name :** 'Vt'

    **Default value :** 0.04

    **Is a state variable :** no.

  - **Modelica parameter name :** 'Maxexp'

    **Default value :** 15

    **Is a state variable :** no.

  - **Modelica parameter name :** 'R'

    **Default value :** 1.000E+08

**Is a state variable :** no.

- **File name of the model :** Diode

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Diode.sci

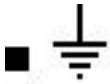
## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Diode.mo

# Nome

Ground — Ground (zero potential reference)

## Block Screenshot



## Contents


- Ground (zero potential reference)
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - Modelica model


## Palette

- Electrical.cosf - Electrical toolbox

## Description

The Ground element is a single port component providing a reference voltage in electrical circuits. The potential at the ground node is zero. Every electrical circuit has to contain at least one ground element.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
- **File name of the model :** Ground

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Ground.sci

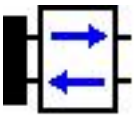
## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Ground.mo

# Nome

Gyrator — Modelica Gyrator

## Block Screenshot



## Contents


- Modelica Gyrator
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model
  - “Authors”

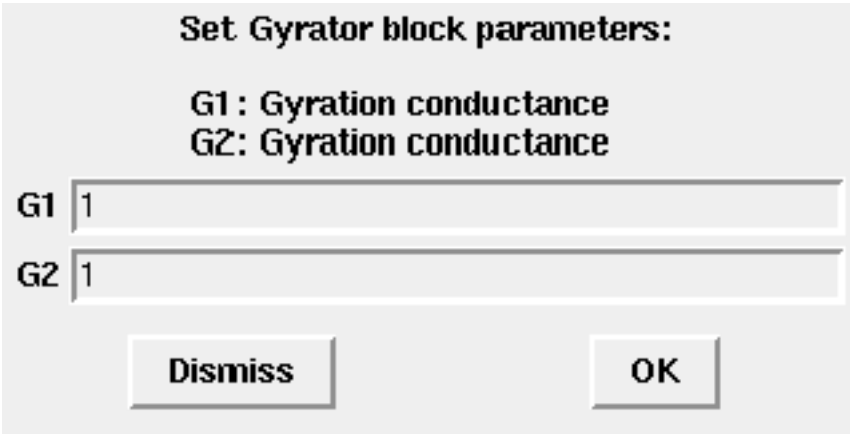

## Palette

- Electrical.cosf - Electrical toolbox

## Description

A gyrator is a two-port element defined by the following equations: i1 = G2 * v2 i2 = -G1 * v1 where the constants G1, G2 are called the gyration conductance.

## Dialog box




- **G1**

  Gyration conductance (-i2/v1) .

Properties : Type 'vec' of size 1.

- **G2**

  Gyration conductance (i1/v2).

  Properties : Type 'vec' of size 1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p1'

    Implicit **variable.**
  - **Modelica variable name :** 'n1'

    Implicit **variable.**
  - **Modelica variable name :** 'p2'

    Implicit **variable.**
  - **Modelica variable name :** 'n2'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'G1'

    **Default value :** 1

    **Is a state variable :** no.
  - **Modelica parameter name :** 'G2'

    **Default value :** 1

    **Is a state variable :** no.
- **File name of the model :** Gyrator

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Gyrator.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Gyrator.mo

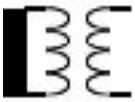
## Authors

**Masoud Najafi** - INRIA

# Nome

IdealTransformer — Ideal Transformer

## Block Screenshot



## Contents


- Ideal Transformer
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model
  - “Authors”

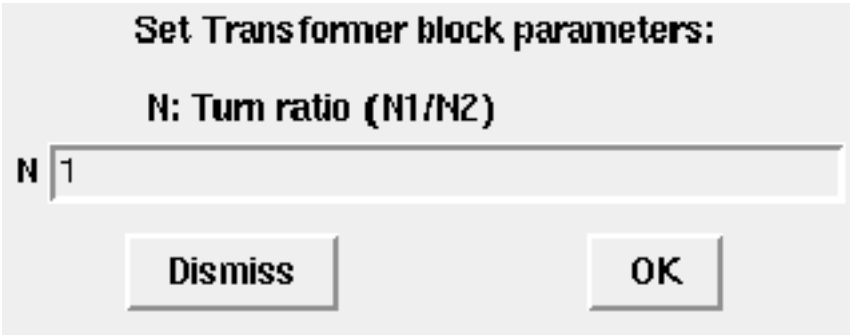

## Palette

- Electrical.cosf - Electrical toolbox

## Description

The ideal transformer is an ideal two-port resistive circuit element which is characterized by the following two equations: v1 = n * v2 i2 = -n * i1 where n is a real number called the turns ratio.

## Dialog box




- **N**

  Turns ratio (N1/N2)

  Properties : Type 'vec' of size 1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p1'

    Implicit **variable.**
  - **Modelica variable name :** 'n1'

    Implicit **variable.**
  - **Modelica variable name :** 'p2'

    Implicit **variable.**
  - **Modelica variable name :** 'n2'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'N'

    **Default value :** 1

    **Is a state variable :** no.
- **File name of the model :** IdealTransformer

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/IdealTransformer.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/IdealTransformer.mo

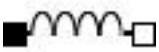
## Authors

**Masoud Najafi** - INRIA

# Nome

Inductor — Electrical inductor

## Block Screenshot



## Contents

- Electrical inductor
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model

## Palette

- Electrical.cosf - Electrical toolbox

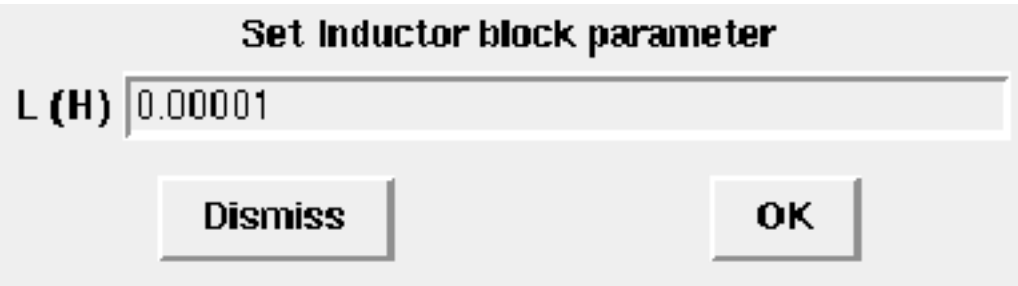
## Description

Inductor is an electrical component that can store energy in electrical circuits. The relationship between the voltage across the ports of an inductor of inductance and the current passing through it is given by:

$$v = L\frac{dI}{dt}$$

Inductors can also be used to differentiate between high-frequency and low-frequency signals and this makes them useful in electronic filters. An inductor shows a high impedance for high frequency signals.

## Dialog box



- **L (H)**

  Inductance

  Properties : Type 'vec' of size 1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'L'

    **Default value :** 0.00001

    **Is a state variable :** no.
- **File name of the model :** Inductor

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Inductor.sci

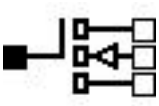
## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Inductor.mo

# Nome

NMOS — Simple NMOS Transistor

## Block Screenshot



## Contents


- Simple NMOS Transistor
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model
  - “See also”
  - “Authors”


## Palette

- Electrical.cosf - Electrical toolbox

## Description

Description

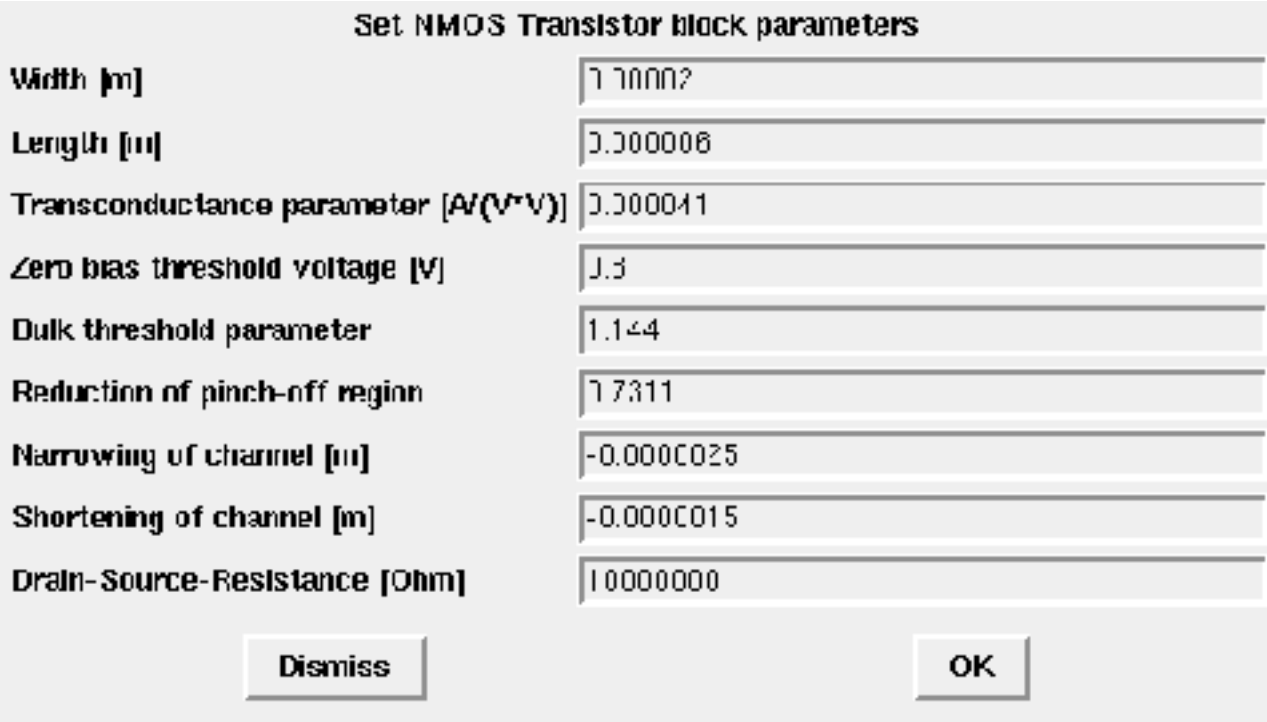
The NMos model is a simple model of a n-channel metal-oxide semiconductor FET. It differs slightly from the device used in the SPICE simulator. For more details please care for H. Spiro.

The model does not consider capacitances. A small fixed drain-source resistance is included (to avoid numerical difficulties).

| W [m] | L [m] | Beta [1/V²] | Vt [V] | K2 | K5 | DW [m] | DL[m] | |
|---|---|---|---|---|---|---|---|---|
| 12.e-6 | 4.e-6 | .062 | -4.5 | .24 | .61 | -1.2e-6 | -.9e-6 | depletion |
| 60.e-6 | 3.e-6 | .048 | .1 | .08 | .68 | -1.2e-6 | -.9e-6 | enhance-ment |
| 12.e-6 | 4.e-6 | .0625 | -.8 | .21 | .78 | -1.2e-6 | -.9e-6 | zero |
| 50.e-6 | 8.e-6 | .0299 | .24 | 1.144 | .7311 | -5.4e-6 | -4.e-6 | |
| 20.e-6 | 6.e-6 | .041 | .8 | 1.144 | .7311 | -2.5e-6 | -1.5e-6 | |
| 30.e-6 | 9.e-6 | .025 | -4. | .861 | .878 | -3.4e-6 | -1.74e-6 | |
| 30.e-6 | 5.e-6 | .031 | .6 | 1.5 | .72 | 0 | -3.9e-6 | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 50.e-6 | 6.e-6 | .0414 | -3.8 | .34 | .8 | -1.6e-6 | -2.e-6 | depletion |
| 50.e-6 | 5.e-6 | .03 | .37 | .23 | .86 | -1.6e-6 | -2.e-6 | enhance-ment |
| 50.e-6 | 6.e-6 | .038 | -.9 | .23 | .707 | -1.6e-6 | -2.e-6 | zero |
| 20.e-6 | 4.e-6 | .06776 | .5409 | .065 | .71 | -.8e-6 | -.2e-6 | |
| 20.e-6 | 4.e-6 | .06505 | .6209 | .065 | .71 | -.8e-6 | -.2e-6 | |
| 20.e-6 | 4.e-6 | .05365 | .6909 | .03 | .8 | -.3e-6 | -.2e-6 | |
| 20.e-6 | 4.e-6 | .05365 | .4909 | .03 | .8 | -.3e-6 | -.2e-6 | |
| 12.e-6 | 4.e-6 | .023 | -4.5 | .29 | .6 | 0 | 0 | depletion |
| 60.e-6 | 3.e-6 | .022 | .1 | .11 | .65 | 0 | 0 | enhance-ment |
| 12.e-6 | 4.e-6 | .038 | -.8 | .33 | .6 | 0 | 0 | zero |
| 20.e-6 | 6.e-6 | .022 | .8 | 1 | .66 | 0 | 0 | |

## Dialog box




- **Width [m]**

  W

  Properties : Type 'vec' of size 1.

- **Length [m]**

  L

  Properties : Type 'vec' of size 1.

- **Transconductance parameter [A/(V*V)]**

  Beta

Properties : Type 'vec' of size 1.

- **Zero bias threshold voltage [V]**

  Vt

  Properties : Type 'vec' of size 1.

- **Bulk threshold parameter**

  K2

  Properties : Type 'vec' of size 1.

- **Reduction of pinch-off region**

  K5

  Properties : Type 'vec' of size 1.

- **Narrowing of channel [m]**

  dW

  Properties : Type 'vec' of size 1.

- **Shortening of channel [m]**

  dL

  Properties : Type 'vec' of size 1.

- **Drain-Source-Resistance [Ohm]**

  RDS

  Properties : Type 'vec' of size 1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'G'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'D'

    Implicit **variable.**
  - **Modelica variable name :** 'B'

    Implicit **variable.**
  - **Modelica variable name :** 'S'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'W'

**Default value :** 0.00002

**Is a state variable :** no.

- **Modelica parameter name :** 'L'

  **Default value :** 0.000006

  **Is a state variable :** no.

- **Modelica parameter name :** 'Beta'

  **Default value :** 0.000041

  **Is a state variable :** no.

- **Modelica parameter name :** 'Vt'

  **Default value :** 0.8

  **Is a state variable :** no.

- **Modelica parameter name :** 'K2'

  **Default value :** 1.144

  **Is a state variable :** no.

- **Modelica parameter name :** 'K5'

  **Default value :** 0.7311

  **Is a state variable :** no.

- **Modelica parameter name :** 'dW'

  **Default value :** -0.0000025

  **Is a state variable :** no.

- **Modelica parameter name :** 'dL'

  **Default value :** -0.0000015

  **Is a state variable :** no.

- **Modelica parameter name :** 'RDS'

  **Default value :** 10000000

  **Is a state variable :** no.

- **File name of the model :** NMOS

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/NMOS.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/NMOS.mo

## See also

- PMOS - Simple PMOS Transistor

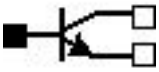
## Authors

- www.modelica.org

# Nome

NPN — NPN transistor

## Block Screenshot



## Contents


- NPN transistor
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model
  - “See also”
  - “Authors”

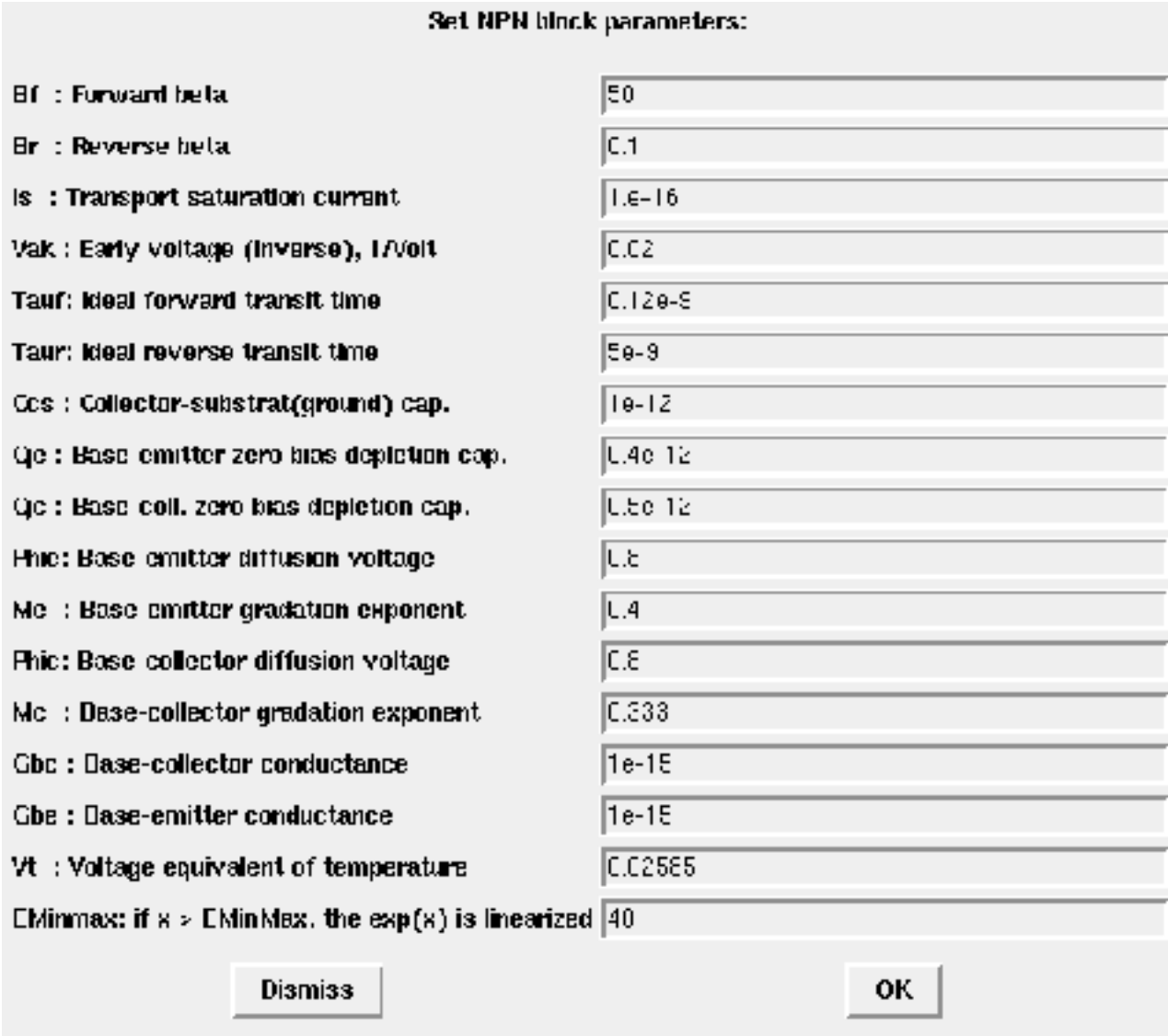

## Palette

- Electrical.cosf - Electrical toolbox

## Description

This model is a simple model of a bipolar NPN junction transistor according to Ebers-Moll.

## Dialog box




| Parameter | Default value | Description |
|---|---|---|
| Bf | 50 | Forward beta |
| Br | 0.1 | Reverse beta |
| Is | 1.e-16 | Transport saturation current [A] |
| Vak | 0.02 | Early voltage (inverse), 1/Volt [1/V] |
| Tauf | 0.12e-9 | Ideal forward transit time [s] |
| Taur | 5e-9 | Ideal reverse transit time [s] |
| Ccs | 1e-12 | Collector-substrat(ground) cap. [F] |
| Cje | 0.4e-12 | Base-emitter zero bias depletion cap. [F] |
| Cjc | 0.5e-12 | Base-coll. zero bias depletion cap. [F] |

| | | |
|---|---|---|
| Phie | 0.8 | Base-emitter diffusion voltage [V] |
| Me | 0.4 | Base-emitter gradation exponent |
| Phic | 0.8 | Base-collector diffusion voltage [V] |
| Mc | 0.333 | Base-collector gradation exponent |
| Gbc | 1e-15 | Base-collector conductance [S] |
| Gbe | 1e-15 | Base-emitter conductance [S] |
| Vt | 0.02585 | Voltage equivalent of temperature [V] |
| EMin | -100 | if x < EMin, the exp(x) function is linearized |
| EMax | 40 | if x > EMax, the exp(x) function is linearized |

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'B'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'C'

    Implicit **variable.**
  - **Modelica variable name :** 'E'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'Bf'

    **Default value :** 50

    **Is a state variable :** no.
  - **Modelica parameter name :** 'Br'

    **Default value :** 0.1

    **Is a state variable :** no.
  - **Modelica parameter name :** 'Is'

    **Default value :** 0

    **Is a state variable :** no.
  - **Modelica parameter name :** 'Vak'

    **Default value :** 0.02

**Is a state variable :** no.

- **Modelica parameter name :** 'Tauf'

  **Default value :** 1.200E-10

  **Is a state variable :** no.

- **Modelica parameter name :** 'Taur'

  **Default value :** 5.000E-09

  **Is a state variable :** no.

- **Modelica parameter name :** 'Ccs'

  **Default value :** 1.000E-12

  **Is a state variable :** no.

- **Modelica parameter name :** 'Cje'

  **Default value :** 4.000E-13

  **Is a state variable :** no.

- **Modelica parameter name :** 'Cjc'

  **Default value :** 5.000E-13

  **Is a state variable :** no.

- **Modelica parameter name :** 'Phie'

  **Default value :** 0.8

  **Is a state variable :** no.

- **Modelica parameter name :** 'Me'

  **Default value :** 0.4

  **Is a state variable :** no.

- **Modelica parameter name :** 'Phic'

  **Default value :** 0.8

  **Is a state variable :** no.

- **Modelica parameter name :** 'Mc'

  **Default value :** 0.333

  **Is a state variable :** no.

- **Modelica parameter name :** 'Gbc'

  **Default value :** 1.000E-15

  **Is a state variable :** no.

- **Modelica parameter name :** 'Gbe'

  **Default value :** 1.000E-15

  **Is a state variable :** no.

- **Modelica parameter name :** 'Vt'

  **Default value :** 0.02585

  **Is a state variable :** no.

- **Modelica parameter name :** 'EMinMax'

  **Default value :** 40

  **Is a state variable :** no.

- **File name of the model :** NPN

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/NPN.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/NPN.mo

## See also

- PNP - PNP transistor

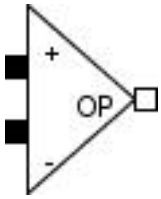
## Authors

- www.modelica.org

# Nome

OpAmp — Ideal opamp (norator-nullator pair)

## Block Screenshot




## Contents


- Ideal opamp (norator-nullator pair)
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - Modelica model
  - “Authors”


## Palette

- Electrical.cosf - Electrical toolbox

## Description

The ideal OpAmp is a two-port. The left port is fixed to v1=0 and i1=0 (nullator). At the right port both any voltage v2 and any current i2 are possible (norator).

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'in_p'

    Implicit **variable.**
  - **Modelica variable name :** 'in_n'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'out'

    Implicit **variable.**
- **File name of the model :** OpAmp

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/OpAmp.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/OpAmp.mo

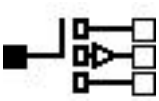
## Authors

- www.modelica.org

# Nome

PMOS — Simple PMOS Transistor

## Block Screenshot



## Contents


- Simple PMOS Transistor
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model
  - “See also”
  - “Authors”


## Palette

- Electrical.cosf - Electrical toolbox

## Description

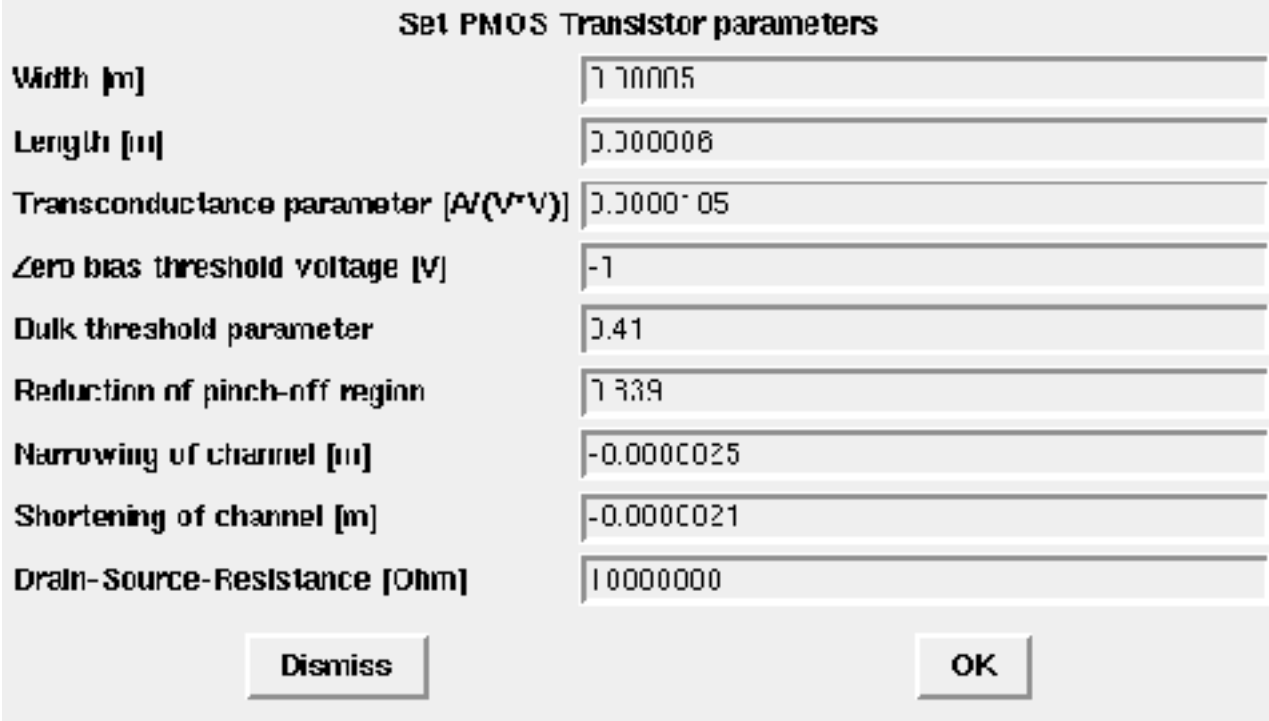
The PMOS model is a simple model of a p-channel metal-oxide semiconductor FET. It differs slightly from the device used in the SPICE simulator. For more details please care for H. Spiro.

The model does not consider capacitances. A small fixed drain-source resistance is included (to avoid numerical difficulties).

Some typical parameter sets are:

| W [m] | L[m] | Beta [1/V²] | Vt [V] | K2 | K5 | DW [m] | DL [m] |
|---|---|---|---|---|---|---|---|
| 50.e-6 | 8.e-6 | .0085 | -.15 | .41 | .839 | -3.8e-6 | -4.0e-6 |
| 20.e-6 | 6.e-6 | .0105 | -1.0 | .41 | .839 | -2.5e-6 | -2.1e-6 |
| 30.e-6 | 5.e-6 | .0059 | -.3 | .98 | 1.01 | 0 | -3.9e-6 |
| 30.e-6 | 5.e-6 | .0152 | -.69 | .104 | 1.1 | -.8e-6 | -.4e-6 |
| 30.e-6 | 5.e-6 | .0163 | -.69 | .104 | 1.1 | -.8e-6 | -.4e-6 |
| 30.e-6 | 5.e-6 | .0182 | -.69 | .086 | 1.06 | -.1e-6 | -.6e-6 |
| 20.e-6 | 6.e-6 | .0074 | -1. | .4 | .59 | 0 | 0 |

## Dialog box




- **Width [m]**

  W

  Properties : Type 'vec' of size 1.

- **Length [m]**

  L

  Properties : Type 'vec' of size 1.

- **Transconductance parameter [A/(V*V)]**

  Beta

  Properties : Type 'vec' of size 1.

- **Zero bias threshold voltage [V]**

  Vt

  Properties : Type 'vec' of size 1.

- **Bulk threshold parameter**

  K2

  Properties : Type 'vec' of size 1.

- **Reduction of pinch-off region**

  K5

  Properties : Type 'vec' of size 1.

- **Narrowing of channel [m]**

  dW

  Properties : Type 'vec' of size 1.

- **Shortening of channel [m]**

  dL

  Properties : Type 'vec' of size 1.

- **Drain-Source-Resistance [Ohm]**

  RDS

  Properties : Type 'vec' of size 1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'G'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'D'

    Implicit **variable.**
  - **Modelica variable name :** 'B'

    Implicit **variable.**
  - **Modelica variable name :** 'S'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'W'

    **Default value :** 0.00005

    **Is a state variable :** no.
  - **Modelica parameter name :** 'L'

    **Default value :** 0.000006

    **Is a state variable :** no.
  - **Modelica parameter name :** 'Beta'

    **Default value :** 0.0000105

    **Is a state variable :** no.
  - **Modelica parameter name :** 'Vt'

    **Default value :** -1

**Is a state variable :** no.

- **Modelica parameter name :** 'K2'

  **Default value :** 0.41

  **Is a state variable :** no.

- **Modelica parameter name :** 'K5'

  **Default value :** 0.839

  **Is a state variable :** no.

- **Modelica parameter name :** 'dW'

  **Default value :** -0.0000025

  **Is a state variable :** no.

- **Modelica parameter name :** 'dL'

  **Default value :** -0.0000021

  **Is a state variable :** no.

- **Modelica parameter name :** 'RDS'

  **Default value :** 10000000

  **Is a state variable :** no.

- **File name of the model :** PMOS

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/PMOS.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/PMOS.mo

## See also

- NMOS - Simple NMOS Transistor

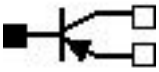
## Authors

- www.modelica.org

# Nome

PNP — PNP transistor

## Block Screenshot



## Contents


- PNP transistor
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model
  - “See also”
  - “Authors”

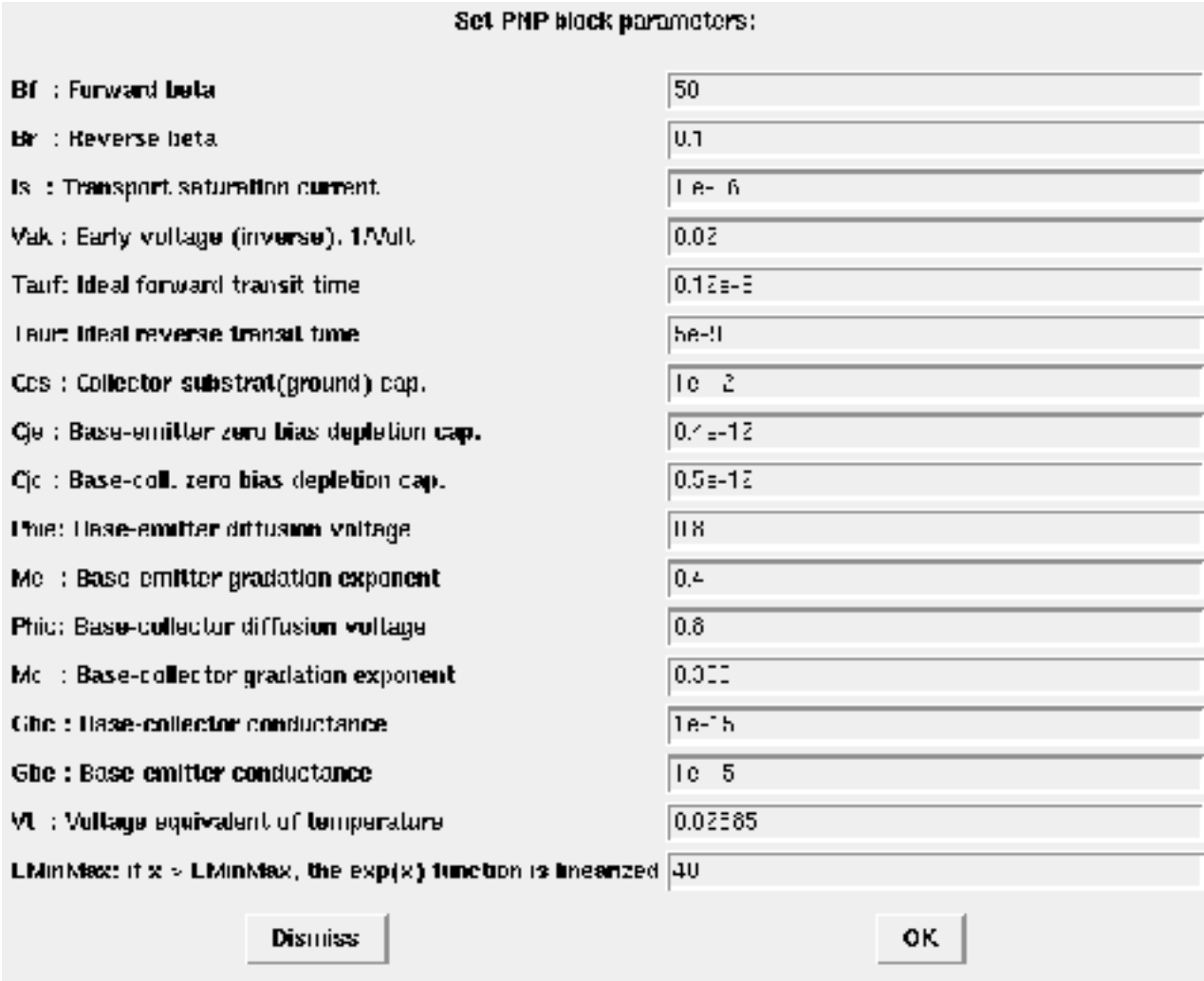

## Palette

- Electrical.cosf - Electrical toolbox

## Description

This model is a simple model of a bipolar PNP junction transistor according to Ebers-Moll.

## Dialog box




| Parameter | Default value | Description |
|---|---|---|
| Bf | 50 | Forward beta |
| Br | 0.1 | Reverse beta |
| Is | 1.e-16 | Transport saturation current [A] |
| Vak | 0.02 | Early voltage (inverse), 1/Volt [1/V] |
| Tauf | 0.12e-9 | Ideal forward transit time [s] |
| Taur | 5e-9 | Ideal reverse transit time [s] |
| Ccs | 1e-12 | Collector-substrat(ground) cap. [F] |
| Cje | 0.4e-12 | Base-emitter zero bias depletion cap. [F] |
| Cjc | 0.5e-12 | Base-coll. zero bias depletion cap. [F] |
| Phie | 0.8 | Base-emitter diffusion voltage [V] |
| Me | 0.4 | Base-emitter gradation exponent |

| | | |
|---|---|---|
| Phic | 0.8 | Base-collector diffusion voltage [V] |
| Mc | 0.333 | Base-collector gradation exponent |
| Gbc | 1e-15 | Base-collector conductance [S] |
| Gbe | 1e-15 | Base-emitter conductance [S] |
| Vt | 0.02585 | Voltage equivalent of temperature [V] |
| EMin | -100 | if x < EMin, the exp(x) function is linearized |
| EMax | 40 | if x > EMax, the exp(x) function is linearized |

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'B'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'C'

    Implicit **variable.**
  - **Modelica variable name :** 'E'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'Bf'

    **Default value :** 50

    **Is a state variable :** no.
  - **Modelica parameter name :** 'Br'

    **Default value :** 0.1

    **Is a state variable :** no.
  - **Modelica parameter name :** 'Is'

    **Default value :** 0

    **Is a state variable :** no.
  - **Modelica parameter name :** 'Vak'

    **Default value :** 0.02

    **Is a state variable :** no.
  - **Modelica parameter name :** 'Tauf'

**Default value :** 1.200E-10

**Is a state variable :** no.

- **Modelica parameter name :** 'Taur'

  **Default value :** 5.000E-09

  **Is a state variable :** no.

- **Modelica parameter name :** 'Ccs'

  **Default value :** 1.000E-12

  **Is a state variable :** no.

- **Modelica parameter name :** 'Cje'

  **Default value :** 4.000E-13

  **Is a state variable :** no.

- **Modelica parameter name :** 'Cjc'

  **Default value :** 5.000E-13

  **Is a state variable :** no.

- **Modelica parameter name :** 'Phie'

  **Default value :** 0.8

  **Is a state variable :** no.

- **Modelica parameter name :** 'Me'

  **Default value :** 0.4

  **Is a state variable :** no.

- **Modelica parameter name :** 'Phic'

  **Default value :** 0.8

  **Is a state variable :** no.

- **Modelica parameter name :** 'Mc'

  **Default value :** 0.333

  **Is a state variable :** no.

- **Modelica parameter name :** 'Gbc'

  **Default value :** 1.000E-15

  **Is a state variable :** no.

- **Modelica parameter name :** 'Gbe'

  **Default value :** 1.000E-15

**Is a state variable :** no.

- **Modelica parameter name :** 'Vt'

  **Default value :** 0.02585

  **Is a state variable :** no.

- **Modelica parameter name :** 'EMinMax'

  **Default value :** 40

  **Is a state variable :** no.

- **File name of the model :** PNP

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/PNP.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/PNP.mo

## See also

- NPN - NPN transistor

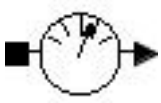
## Authors

- www.modelica.org

# Nome

PotentialSensor — Potential sensor

## Block Screenshot



## Contents


- Potential sensor
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - Modelica model


## Palette

- Electrical.cosf - Electrical toolbox

## Description

This block is used to measure the voltage with respect to the reference voltage (**Ground** block) in an electrical circuit. The voltage is given to the explicit part of the model via an explicit output port.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'v'

    Explicit **variable.**
- **File name of the model :** PotentialSensor

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/PotentialSensor.sci

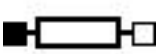
## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/PotentialSensor.mo

# Nome

Resistor — Electrical resistor

## Block Screenshot



## Contents


- Electrical resistor
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model


## Palette

- Electrical.cosf - Electrical toolbox

## Description

A resistor is a two-port electrical component that resists an electric current by producing a voltage drop () between its terminals according to the Ohm's law.

$$R = \frac{V}{I}$$

The electrical resistance () is equal to the voltage drop across the resistor divided by the current through the resistor ().

## Dialog box

- **R (ohm)**

Resistance

Properties : Type 'vec' of size 1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'R'

    **Default value :** 0.01

    **Is a state variable :** no.
- **File name of the model :** Resistor

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Resistor.sci

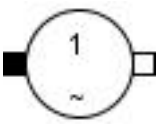
## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Resistor.mo

# Nome

SineVoltage — Sine voltage source

## Block Screenshot




## Contents


- Sine voltage source
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model
  - “Authors”

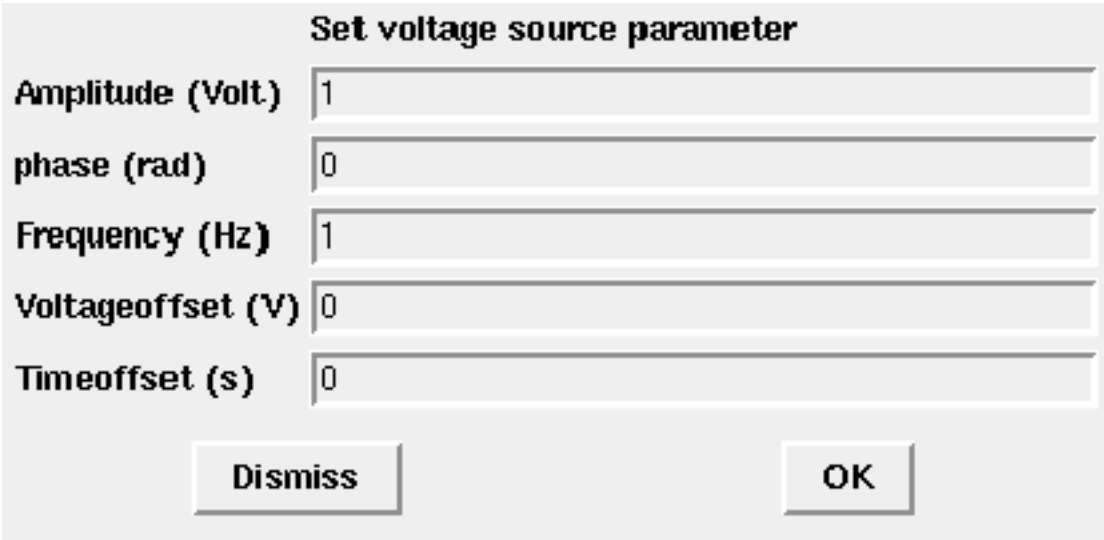

## Palette

- Electrical.cosf - Electrical toolbox

## Description

This Modelica block a general sine voltage source. The internal ohmic resistance is zero.

## Dialog box



- **Amplitude (Volt)**

  Sine voltage amplitude

  Properties : Type 'vec' of size 1.

- **phase (rad)**

  phase shift of the sine voltage

  Properties : Type 'vec' of size 1.

- **Frequency (Hz)**

  Sine voltage frequency

  Properties : Type 'vec' of size 1.

- **Voltageoffset (V)**

  Offset Voltage of the sine voltage

  Properties : Type 'vec' of size 1.

- **Timeoffset (s)**

  Start time. During the start time, the signal value is equal to the voltage offset.

  Properties : Type 'vec' of size 1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**

- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**

- **Parameters :**
  - **Modelica parameter name :** 'V'

    **Default value :** 1

    **Is a state variable :** no.

  - **Modelica parameter name :** 'phase'

    **Default value :** 0

    **Is a state variable :** no.

  - **Modelica parameter name :** 'freqHz'

    **Default value :** 1

    **Is a state variable :** no.

- **Modelica parameter name :** 'offset'

  **Default value :** 0

  **Is a state variable :** no.

- **Modelica parameter name :** 'startTime'

  **Default value :** 0

  **Is a state variable :** no.

- **File name of the model :** SineVoltage

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/SineVoltage.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/SineVoltage.mo

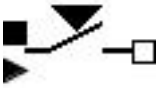
## Authors

- www.modelica.org

# Nome

Switch — Non-ideal electrical switch

## Block Screenshot



## Contents


- Non-ideal electrical switch
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model
  - “Authors”

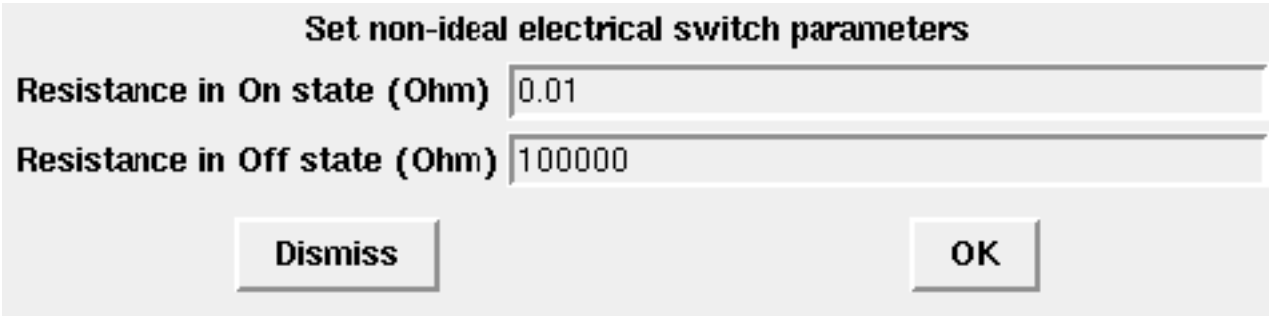

## Palette

- Electrical.cosf - Electrical toolbox

## Description

This is a non-ideal two-pole switch. If the explicit input become positive, two pins are connected via a resistor of resistance RON). Otherwise, two pins are connected via ROFF resistance. Note that using this block may result in a stiff model, so try to choose proper error tolerances.

## Dialog box




- **Resistance in On state (Ohm)**

  Switch resistance when the Switch is closed

  Properties : Type 'vec' of size 1.

- **Resistance in Off state (Ohm)**

  Switch resistance when the switch is open

  Properties : Type 'vec' of size 1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
  - **Modelica variable name :** 'inp'

    Explicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'Ron'

    **Default value :** 0.01

    **Is a state variable :** no.
  - **Modelica parameter name :** 'Roff'

    **Default value :** 100000

    **Is a state variable :** no.
- **File name of the model :** Switch

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/Switch.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/Switch.mo

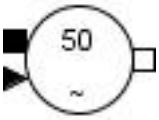
## Authors

**Masoud Najafi** - INRIA

# Nome

VVsourceAC — Variable AC voltage source

## Block Screenshot




## Contents


- Variable AC voltage source
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model


## Palette

- Electrical.cosf - Electrical toolbox

## Description

The variable voltage source block is a model for a controlled AC voltage source. This component provides a sinusoid voltage across its ports. The amplitude of the output voltage is governed by the explicit input and the frequency is defined by the user. The ohmic resistance of the block is zero.

## Dialog box

- **Frequency (Hz)**

  Frequency of the output sinosoid voltage

  Properties : Type 'vec' of size -1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

Implicit **variable.**

- **Modelica variable name :** 'VA'

  Explicit **variable.**

- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**

- **Parameters :**
  - **Modelica parameter name :** 'f'

    **Default value :** 50

    **Is a state variable :** no.

- **File name of the model :** VVsourceAC

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/VVsourceAC.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/VVsourceAC.mo

# Nome

VariableResistor — Electrical variable resistor

## Block Screenshot

## Contents


- Electrical variable resistor
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - Modelica model


## Palette

- Electrical.cosf - Electrical toolbox

## Description

This component represents a variable ohmic resistor. The resistance () is controlled via an explicit input port.

$$R_x = \frac{V}{I}$$

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
  - **Modelica variable name :** 'R'

    Explicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**
- **File name of the model :** VariableResistor

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/VariableResistor.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/VariableResistor.mo

# Nome

VoltageSensor — Electrical voltage sensor

## Block Screenshot

## Contents

- Electrical voltage sensor
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - Modelica model

## Palette

- Electrical.cosf - Electrical toolbox

## Description

This component is used to measure the voltage difference between two nodes in an electrical circuit. The output signal is the difference between the voltages of the black port and the white port, ,

$$V_{output} = V_{black} - V_{white}$$

The ohmic conductance of this block is zero.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**
  - **Modelica variable name :** 'v'

    Explicit **variable.**
- **File name of the model :** VoltageSensor

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/VoltageSensor.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Electrical/VoltageSensor.mo

# Nome

VsourceAC — Electrical AC voltage source

## Block Screenshot

## Contents


- Electrical AC voltage source
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model

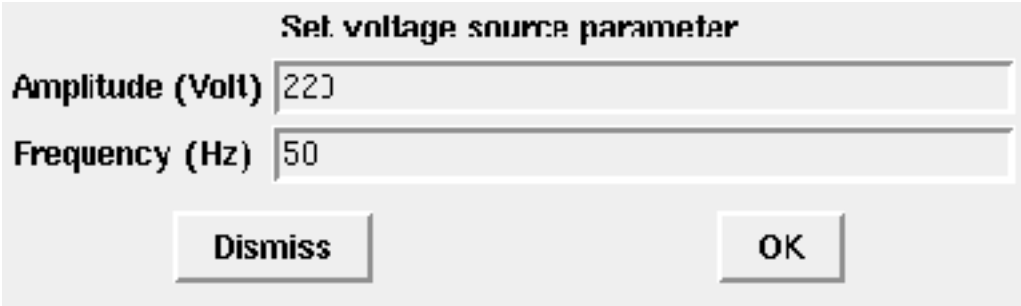

## Palette

- Electrical.cosf - Electrical toolbox

## Description

This component is an AC voltage source with sinusoid output voltage. The amplitude and the frequency of the output voltage is set by the user. The ohmic resistance of this block is zero.

## Dialog box




- **Amplitude (Volt)**

  Amplitude of the output sinusoid voltage

  Properties : Type 'vec' of size -1.

- **Frequency (Hz)**

  Frequency of the output sinusoid voltage

Properties : Type 'vec' of size -1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'p'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'n'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'VA'

    **Default value :** 220

    **Is a state variable :** no.
  - **Modelica parameter name :** 'f'

    **Default value :** 50

    **Is a state variable :** no.
- **File name of the model :** VsourceAC

## Interfacing function

- SCI/modules/scicos_blocks/macros/Electrical/VsourceAC.sci

## Modelica model

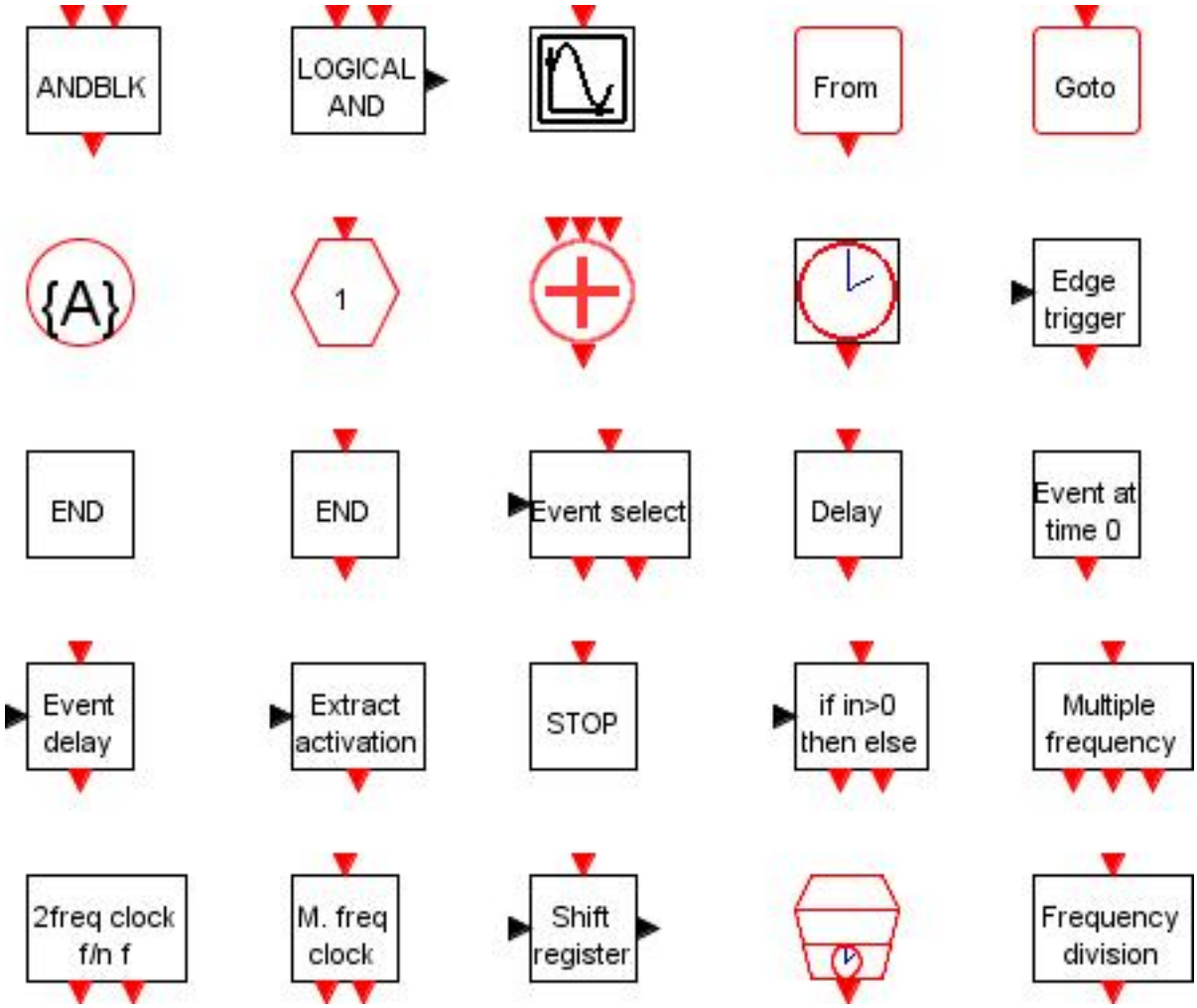
- SCI/modules/scicos_blocks/macros/Electrical/VsourceAC.mo

# 8. Event handling palette

# Nome

Events_pal — Event handling palette

## Block Screenshot




## Module

- xcos

## Description

The Event handling palette is used to handle events in the diagram. It contains several activation clocks, synchronous blocks and blocks to gather events in a single link.

## Blocks

- ANDBLK - Activation and
- ANDLOG_f - Logical and
- CEVENTSCOPE — Activation scope
- CLKFROM — Receives data from a corresponding CLKGOTO
- CLKGOTO — Pass block input to CLKFROM block

- CLKGotoTagVisibility — Define Scope of CLKGOTO tag visibility
- CLKOUTV_f — Output activation port
- CLKSOMV_f — Activation union
- CLOCK_c — Activation clock
- EDGE_TRIGGER — EDGE_TRIGGER block
- ENDBLK — END block
- END_c — END_c block
- ESELECT_f — Synchronous block Event-Select
- EVTDLY_c — Event delay
- EVTGEN_f — Event generator
- EVTVARDLY — Event variable delay
- Extract_Activation — Extract_Activation block
- HALT_f — Stop simulation
- IFTHEL_f — Synchronous block If-Then-Else
- M_freq — Multiple Frequencies
- MCLOCK_f — Clock Frequency division
- MFCLCK_f — Clock Multiple Frequencies
- REGISTER — Shift Register
- SampleCLK — Sample Time Clock
- freq_div — Frequency division

# Nome

ANDBLK — Activation and

## Block Screenshot

## Contents


- Activation and
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “Authors”


## Palette

- Event handling palette

## Description

The Bus Creator block combines a set of signals, i.e., a group of signals represented by a single line in a block diagram. It allows you to reduce the number of lines required to route signals from one part of a diagram to another. This makes your easier to understand.

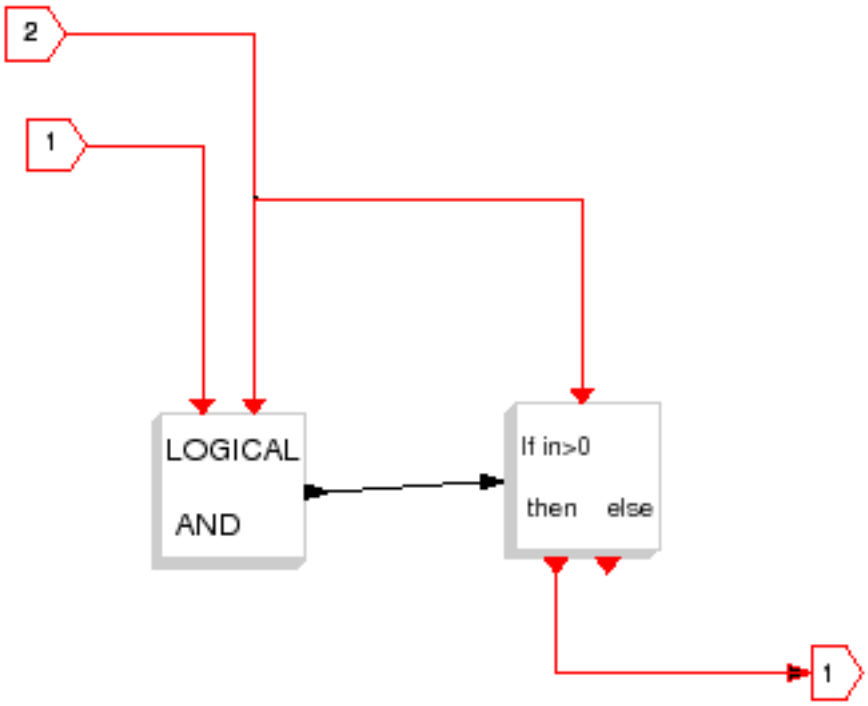
## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 2
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/ANDBLK.sci

## Compiled Super Block content




## Authors

**Ramine Nikoukhah** - INRIA

# Nome

ANDLOG_f — Logical and

## Block Screenshot

## Contents


- Logical and
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Authors”


## Palette

- Event handling palette

## Description

This block, with two event inputs and a regular output, outputs **+1** or **-1** on its regular output depending on input events.

- **+1 :** When events are synchronously (present on both event input ports),
- **-1 :** When only one event is present.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 2
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *andlog*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/ANDLOG_f.sci

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

CEVENTSCOPE — Activation scope

## Block Screenshot

## Contents


- Activation scope
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

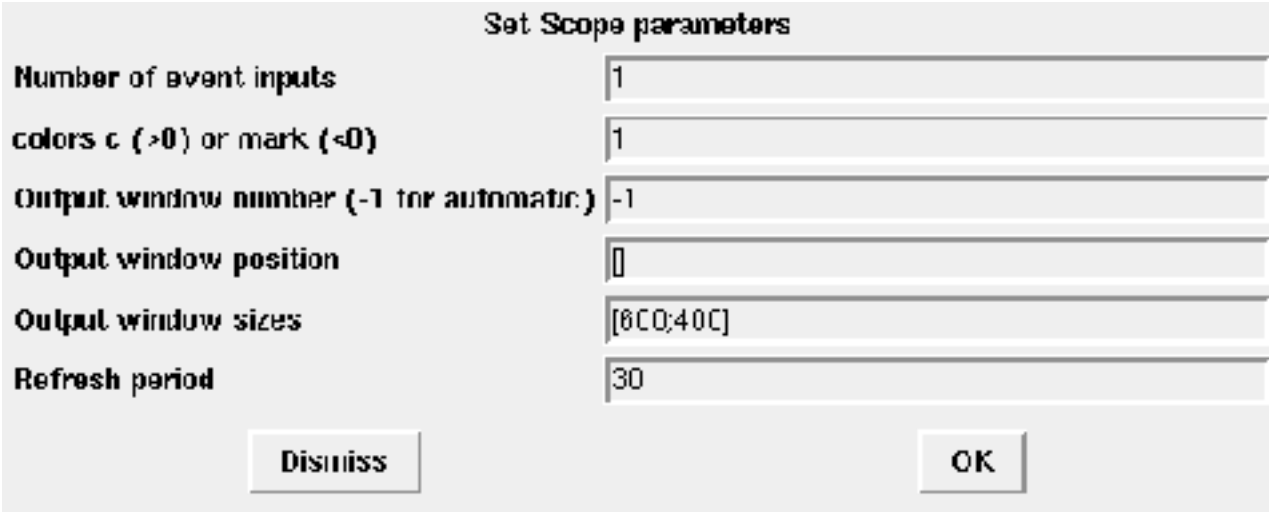

## Palette

- Event handling palette

## Description

This block realizes the visualization of the input event signals.

## Dialog box




- **Number of event inputs**

an integer giving the number of event input ports colors : a vector of integers. The i-th element is the color number ( $>= 0$ ) or dash type ( $< 0$ ) used to draw the evolution of the i-th input port signal. See**xset** for color (dash type) definitions.

Properties : Type 'vec' of size 1

- **colors c**

  an integer. It is the color number ( $> 0$ ) or dash type ( $< 0$ ) used to draw the evolution of the input port signal. See**plot2d** for color (dash type) definitions.

  Properties : Type 'vec' of size -1

- **Output window number**

  The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired). Output window position : a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

  Properties : Type 'vec' of size 1

- **Output window position**

  Properties : Type 'vec' of size -1

- **Output window sizes**

  a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

  Properties : Type 'vec' of size -1

- **Refresh period**

  Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *cevscpe*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CEVENTSCOPE.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/cevscpe.c (Type 4)

## Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

# Nome

CLKFROM — Receives data from a corresponding CLKGOTO

## Block Screenshot

## Contents

- Receives data from a corresponding CLKGOTO
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “See also”
  - “Authors”

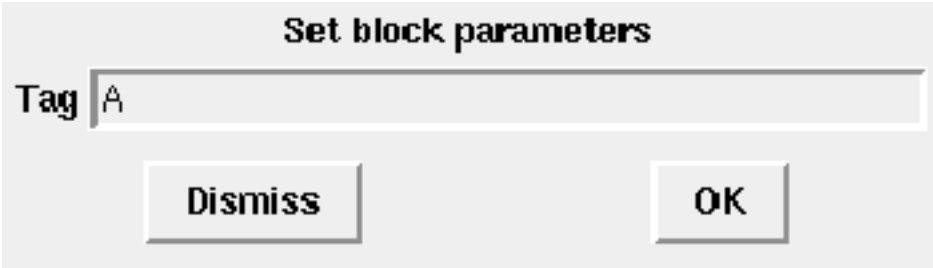
## Palette

- Event handling palette

## Description

This block is used to connect events ports. For more information on how it works please refer to the documentation of the FROM block by clicking on the link in the "See also" field.

## Dialog box




- **Tag**

  The tag of the CLKGOTO block passing the signal to this CLKFROM block.

  Properties : Type 'str' of size -1.

## Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *clkfrom*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/CLKFROM.sci

## See also

- FROM - FROM Receives data from a corresponding GOTO

## Authors

**Fady NASSIF** - INRIA

# Nome

CLKGOTO — Pass block input to CLKFROM block

## Block Screenshot

## Contents

- Pass block input to CLKFROM block
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “See also”
  - “Authors”

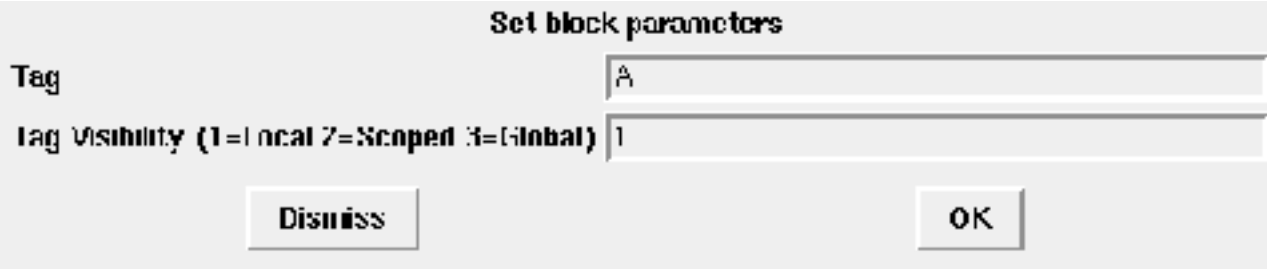
## Palette

- Event handling palette

## Description

This block is used to connect events ports. For more information on how it works please refer to the documentation of the GOTO block by clicking on the link in the "See also" field.

## Dialog box




- **Tag**

  This parameter identifies the Goto block whose scope is defined in this block.

  Properties : Type 'str' of size -1.

- **Tag Visibility (1=Local 2=Scoped 3=Global)**

This parameter idetifies the visibility of the block. It can be local(1), scoped(2) or global(3).

Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *clkgoto*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/CLKGOTO.sci

## See also

- GOTO - GOTO Pass block input to From block

## Authors

**Fady NASSIF** - INRIA

# Nome

CLKGotoTagVisibility — Define Scope of CLKGOTO tag visibility

## Block Screenshot

## Contents


- Define Scope of CLKGOTO tag visibility
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “See also”
  - “Authors”

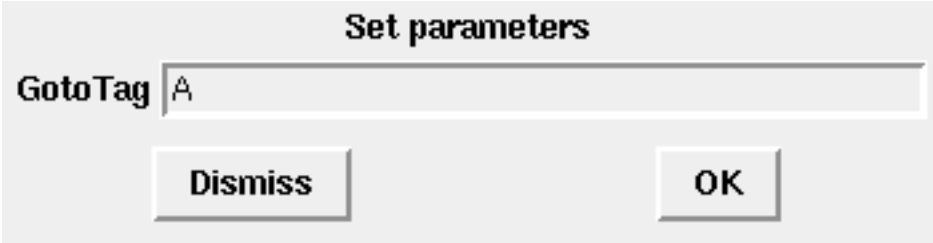

## Palette

- Event handling palette

## Description

This block is used in the event case. For more information on how it works please refer to the documentation of the GotoTagVisibility block by clicking on the link in the "See also" field.

## Dialog box




- **GotoTag**

  The Goto block tag whose visibility is defined by the location of this block.

  Properties : Type 'str' of size -1.

## Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *clkgototagvisibility*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/CLKGotoTagVisibility.sci

## See also

- GotoTagVisibility - Define Scope of GOTO tag visibility

## Authors

**Fady NASSIF** - INRIA

# Nome

CLKOUTV_f — Output activation port

## Block Screenshot

## Contents

- Output activation port
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

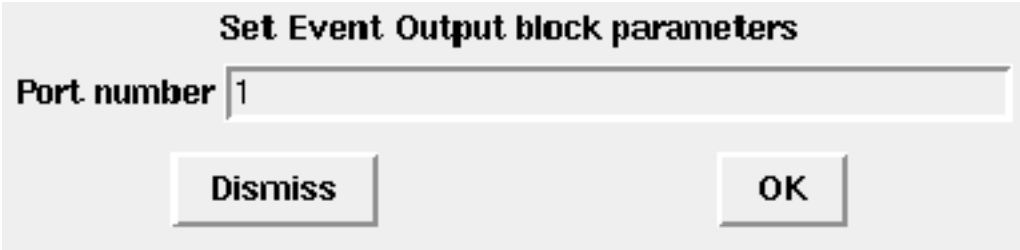
## Palette

- Event handling palette

## Description

This block must only be used inside Xcos Super Blocks to represent an event output port. In a Super Block, the event output ports must be numbered from 1 to the number of event output ports.

## Dialog box




- **Port number**

  an integer defining the port number.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no

- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *output*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CLKOUTV_f.sci

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

CLKSOMV_f — Activation union

## Block Screenshot

## Contents


- Activation union
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Event handling palette

## Description

This block is an event addition block with up to three inputs. The output reproduces the events on all the input ports. Strictly speaking, CLKSOMV is not a Xcos block because it is discarded at the compilation phase. The inputs and output of CLKSOMV are synchronized.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 3
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sum*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/CLKSOMV_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/sum.c (Type 0)

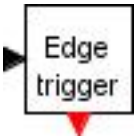
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

EDGE_TRIGGER — EDGE_TRIGGER block

## Block Screenshot




## Contents


- EDGE_TRIGGER block
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content

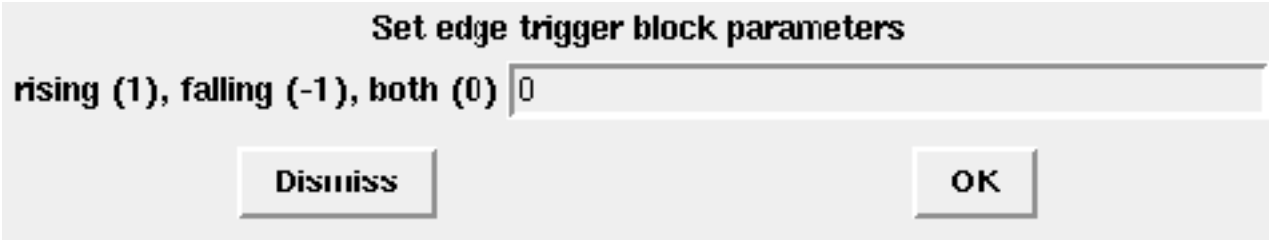

## Palette

- Event handling palette

## Description

This block generates an event on rising, falling or both edges of the input signal (depending on block parameter). A rising edge is a change in value from strictly negative to positive or zero, or a change in value from zero to strictly positive. A falling edge is the opposite. Note that this block only generates an event if the input jumps due to an event. The generated event is synchronous with the event causing the jump. This block does not detect continuous-time zero-crossings.

## Dialog box




- **rising (1), falling (-1), both (0)**

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no

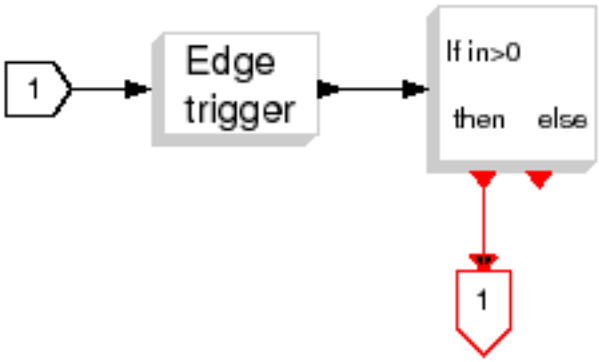
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/EDGE_TRIGGER.sci

## Compiled Super Block content



# Nome

ESELECT_f — Synchronous block Event-Select

## Block Screenshot

## Contents

- Synchronous block Event-Select
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

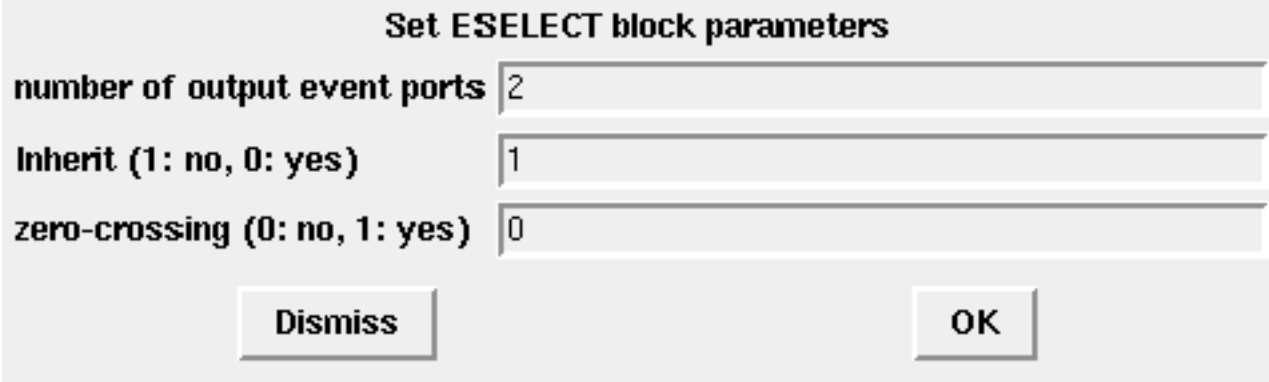
## Palette

- Event handling palette

## Description

Special block similar to If-Then-Else. Input and output are synchronized. The incoming event is directed to one of the output event ports depending on the value of the regular input.

## Dialog box




- **number of output event ports**

  A scalar. Number of output event ports.

  Properties : Type 'vec' of size 1

- **Inherit**

If no, then it inherits the event from event input port, elseif yes, then event is activated by regular input port.

Properties : Type 'vec' of size 1

- **zero-crossing**

  Select to enable zero crossing detection.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type -1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 2
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *eselect*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/ESELECT_f.sci

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

EVTDLY_c — Event delay

## Block Screenshot

## Contents


- Event delay
- • “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Event handling palette

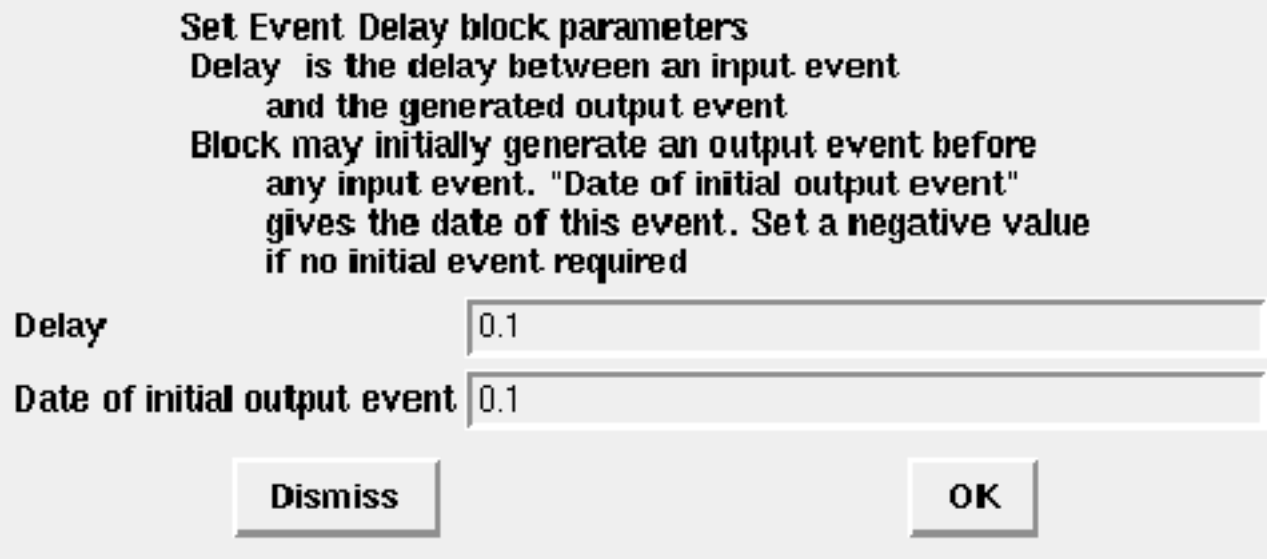
## Description

One event is generated **Delay** after an event enters the unique input event port. Block may also generate an initial output event. The event date of that block is computed by the formula :

$$t_i = t_{\mathrm{init}} + i * T_{\mathrm{delay}},$$

where $t_{\mathrm{init}}$ the date of initial output event, $T_{\mathrm{delay}}$ the delay and $i$ and internal integer discrete counter.

## Dialog box




- **Delay**

  scalar. Time delay between input and output event.

  Properties : Type 'vec' of size 1

- **Date of initial output event**

  scalar. If**Auto-exec>=0** block initially generates an output event at date**Auto-exec** .

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *evtdly4*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/EVTDLY_c.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/evtdly4.c (Type 4)

## See also

- CLOCK_c - Activation clock

## Authors

**Alan Layec** - INRIA

# Nome

EVTGEN_f — Event generator

## Block Screenshot

## Contents

- Event generator
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

## Palette

- Event handling palette

## Description

One event is generated on the unique output event port if Event time is larger than equal to zero, if not, no event is generated.

## Dialog box

- **Event Time**

  scalar. date of the initial event.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *trash*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/EVTGEN_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/trash.f (Type 0)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

EVTVARDLY — Event variable delay

## Block Screenshot

## Contents


- Event variable delay
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

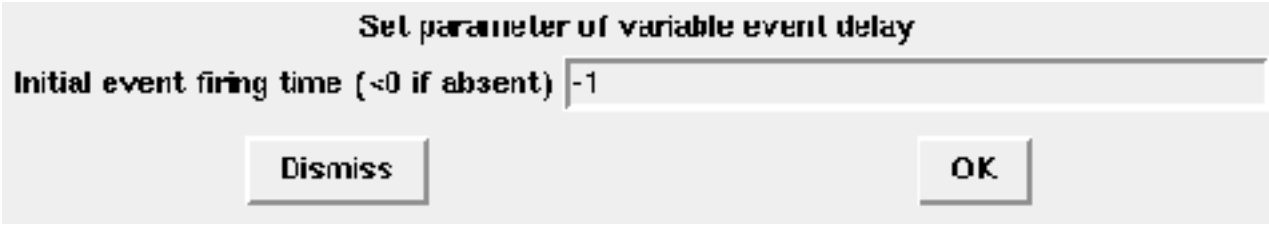

## Palette

- Event handling palette

## Description

One event is generated Delay after an event enters the unique input event port. The value of the delay is read from the regular input port. Block may also generate an initial output event.

## Dialog box




- **Initial event firing time**

  One event is generated on the unique output event port if Event time is larger than equal to zero, if not, no event is generated.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *evtvardly*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/EVTVARDLY.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/evtvardly.c (Type 4)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

Extract_Activation — Extract_Activation block

## Block Screenshot

## Contents

- Extract_Activation block
- • “Palette”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content

## Palette

- Event handling palette

## Default properties

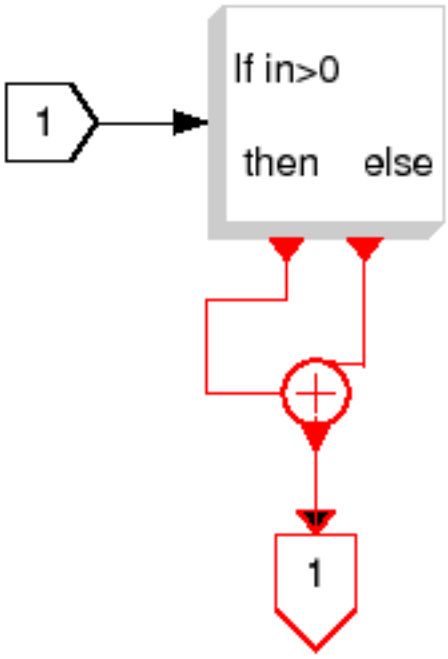
- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/Extract_Activation.sci

## Compiled Super Block content



# Nome

HALT_f — Halt

## Block Screenshot

## Contents


- Halt
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Event handling palette

## Description

This block has a unique input event port. Upon the arrival of an event, the simulation is stopped and the main Xcos window is activated. Simulation can be restarted or continued (Run button).

## Dialog box

- **State on halt**

  A scalar value to be placed in the state of the block. For debugging purposes this allows to distinguish between different halts.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *hltblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/HALT_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/hltblk.f (Type 0)

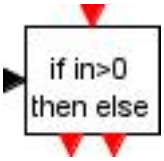
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

IFTHEL_f — Synchronous block If-Then-Else

## Block Screenshot




## Contents

- Synchronous block If-Then-Else
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

## Palette

- Event handling palette

## Description

One event is generated on one of the output event ports when an input event arrives. Depending on the sign of the regular input, the event is generated on the first or second output. This is a Synchro block, , input and output event are synchronized.

## Dialog box

- **Inherit**

  If no, then it inherits the event from event input port, elseif yes, then event is activated by regular input port.

  Properties : Type 'vec' of size 1.

- **zero-crossing**

Select to enable zero crossing detection.

Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**

  **- port 1 : size [1,1] / type -1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 2
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *ifthel*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/IFTHEL_f.sci

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

MCLOCK_f — MCLOCK_f title

## Block Screenshot

## Contents

- Multiple Frequencies f/n
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content

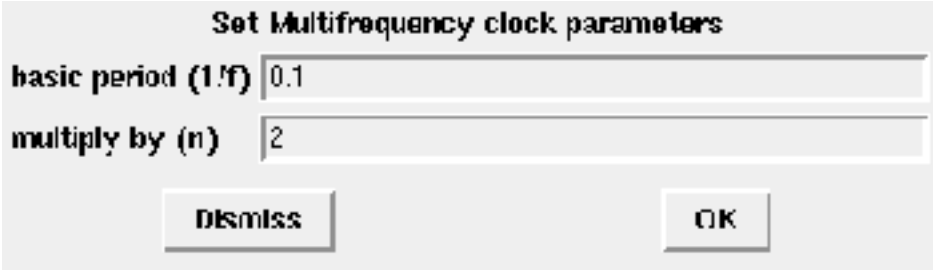
## Palette

- Event handling palette

## Description

Add here a paragraph of the function description

## Dialog box




- **basic period (1/f)**

  The parameter description 1.

  Properties : Type 'vec' of size 1.

- **multiply by (n)**

  The parameter description 2.

  Properties : Type 'vec' of size 1.

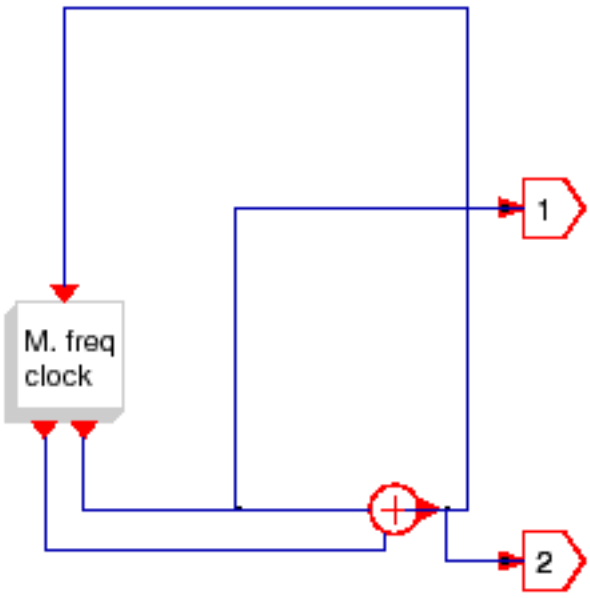
## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 2
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/MCLOCK_f.sci

## Compiled Super Block content



# Nome

MFCLCK_f — MFCLCK_f title

## Block Screenshot

## Contents

- Clock Frequency division
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

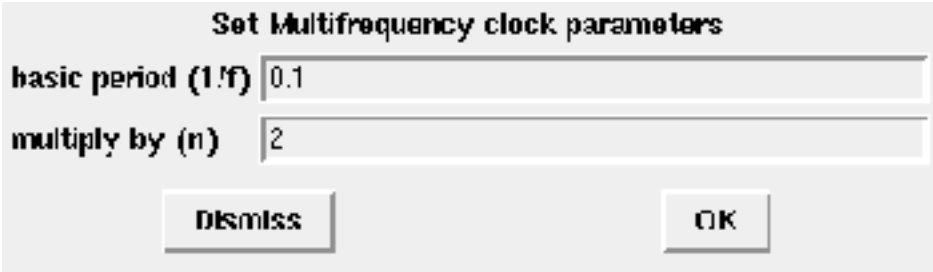
## Palette

- Event handling palette

## Description

Add here a paragraph of the function description

## Dialog box




- **basic period (1/f)**

  The parameter description 1.

  Properties : Type 'vec' of size 1.

- **multiply by (n)**

  The parameter description 2.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 2
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *mfclck*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/MFCLCK_f.sci

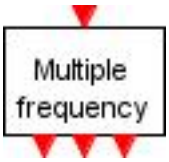
## Computational function

- SCI/modules/scicos_blocks/src/fortran/mfclck.f (Type 0)

# Nome

M_freq — Multiple Frequencies

## Block Screenshot




## Contents


- Multiple Frequencies
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Event handling palette

## Description

This blocks generates events at specific sample time of the simulation time. The sample time is given in the "Sample Time" field and the offset is given in the "Offset" field. This block has one event input, the number of event outputs depends on the number of different sample time. For example if the vector of sample time is [1 1 2] and the vector of offset is [0 .5 0] then the block has 7 outputs.

- The first output is activated when the simulation time is equal to a multiple of the first sample time plus the first offset

- The second output is activated when the simulation time is equal to a multiple of the second sample time plus the second offset.

- The third output is activated when we have both cases, first case and second case.

- The fourth output is activated when the simulation time is equal to a multiple of the third sample time plus the third offset.

- The fifth output is activated when we have both cases, first case and forth case.

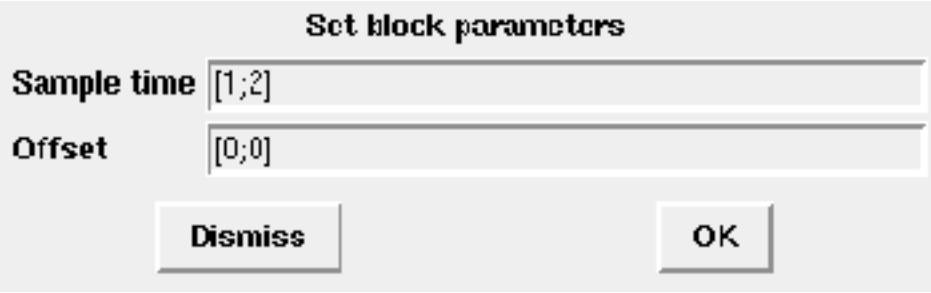
- The sixth output is activated when we have both cases, second case and fourth case.

- The seventh output is activated when we have both cases, third case and forth case.

etc...

So the number of outputs is equal to 2**number of different time values. Each of these time values is represented by a binary number associated to the output's number in decimal.

## Dialog box




- **Sample time**

  Vector of sample time values.

  Properties : Type 'vec' of size -1.

- **Offset**

  Vector of offset values. Must have the same size as the Sample time and each offset value must be less than its corresponding sample time.

  Properties : Type 'vec' of size -1.

## Example

```
Let us take the example where the sample time is equal to [1 1 2] and the offse
When t=0, the fifth output is activated (001 + 100).
When t=0.5, the second output is activated (010).
When t=1, the first output is activated (001).
When t=1.5, the second output is activated (010).
When t=2 we loop back to 0.
```

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 3
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *m_frequ*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/M_freq.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/m_frequ.c (Type 4)

## See also

- MFCLCK_f - MFCLCK_f title

## Authors

**Fady NASSIF** - INRIA

# Nome

freq_div — Frequency division

## Block Screenshot

## Contents


- Frequency division
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “Authors”

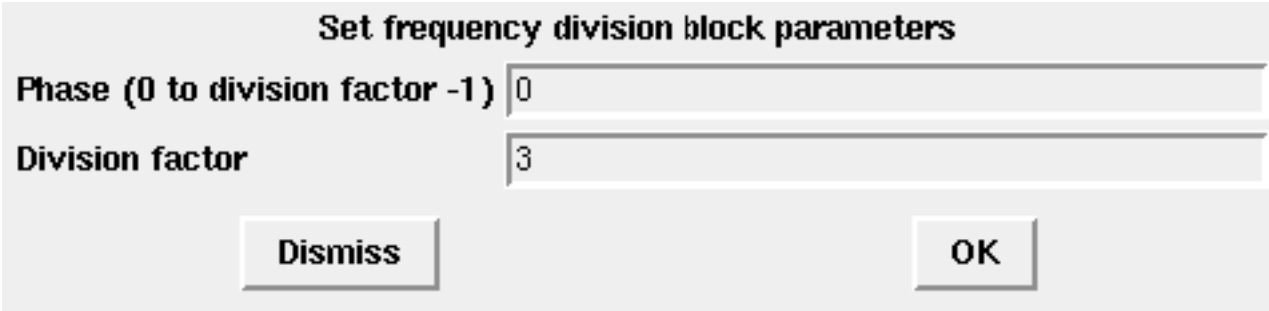

## Palette

- Event handling palette

## Description

This block is a Super Block. The input event is directed once every n times to output. The input is driven by an event clock.

## Dialog box




- **Phase**

  positive scalar.

  Properties : Type 'vec' of size 1

- **Division factor**

an integer greater than 1.

Properties : Type 'vec' of size 1

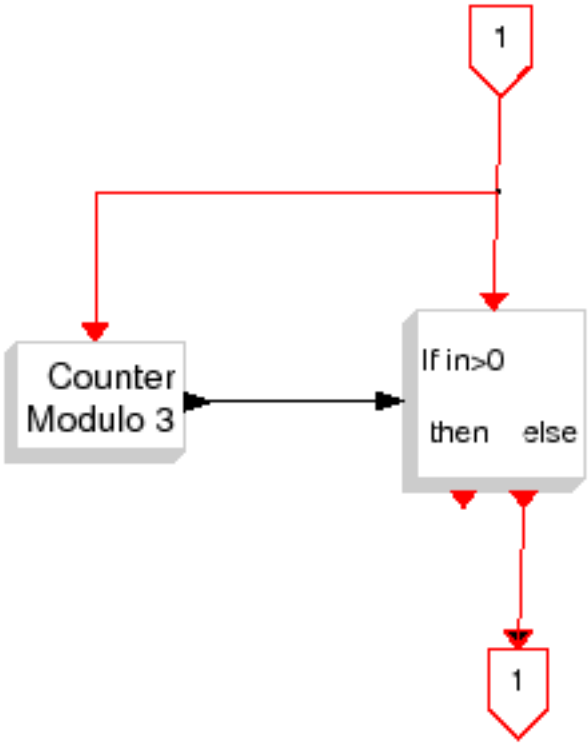
## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/freq_div.sci

## Compiled Super Block content



## Authors

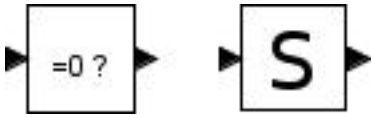
**Ramine Nikoukhah** - INRIA

# 9. Implicit palette

# Nome

Implicit_pal — Implicit palette

## Block Screenshot




## Module

- xcos

## Description

In the Implicit palette, you can find blocks used to model implicit systems.

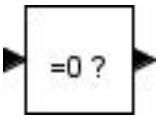
## Blocks

- CONSTRAINT_f - Constraint
- DIFF_f - Sum

# Nome

CONSTRAINT_f — Constraint

## Block Screenshot




## Contents


- Constraint
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Implicit palette

## Description

Defines implicit algebraic relations.

## Dialog box

- **Set number of constraints**

  no of algebraic relations to be defined.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** yes
- **direct-feedthrough:** no

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *constraint*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/CONSTRAINT_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/constraint.f (Type 10001)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

DIFF_f — Derivative

## Block Screenshot

## Contents


- Derivative
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

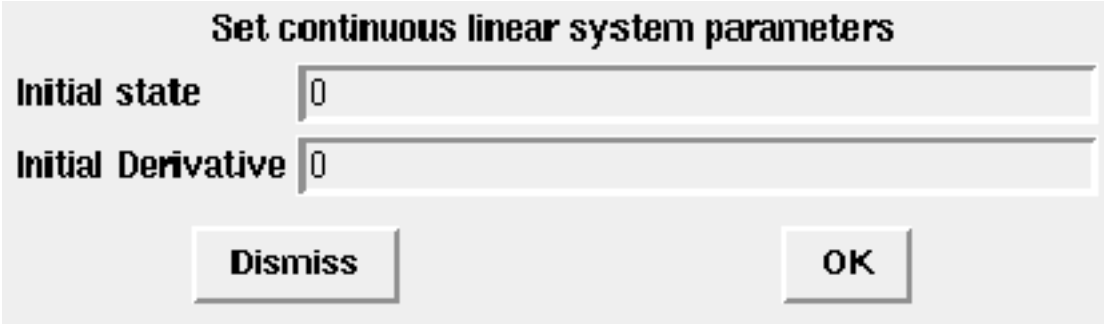

## Palette

- Implicit palette

## Description

This block computes the derivative of the input.

## Dialog box




- **Initial state**

  The initial continuous state.

  Properties : Type 'vec' of size 1.

- **Initial Derivative**

  The initial derivative state.

Properties : Type 'vec' of size 1.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** yes
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *diffblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/DIFF_f.sci

## Computational function

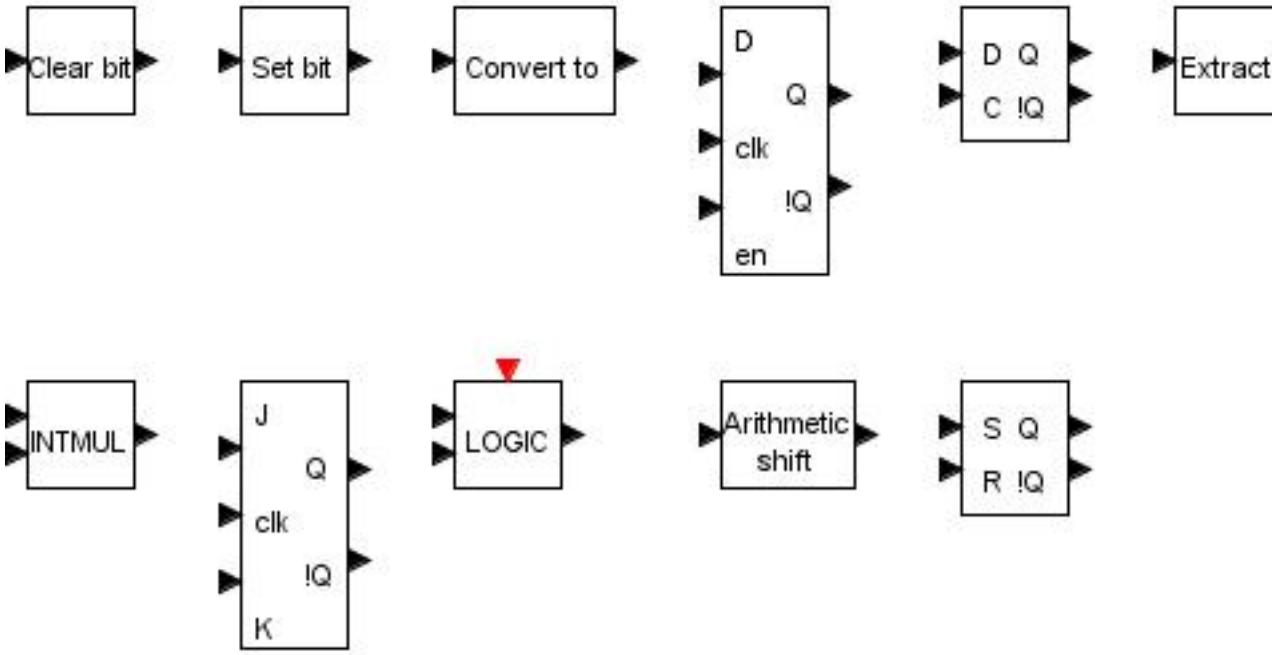
- SCI/modules/scicos_blocks/src/fortran/diffblk.f (Type 10001)

# 10. Integer palette

# Nome

Integer_pal — Integer palette

## Block Screenshot




## Module

- xcos

## Description

That palette is dedicated to handle integer numbers. Various basic operators for the management of bit fields and for logic are implemented as well as common gates encountered in digital circuits.

## Blocks

- BITCLEAR - BITCLEAR Clear a Bit
- BITSET - BITSET Set a Bit
- CONVERT - CONVERT Data Type Conversion
- DFLIPFLOP - D flip-flop
- DLATCH - D latch flip-flop
- EXTRACTBITS - EXTRACTBITS
- INTMUL - INTMUL integer matrix multiplication
- JKFLIPFLOP - JK flip-flop
- LOGIC - Combinational Logic
- SHIFT - SHIFT Shift Bits
- SRFLIPFLOP - SR flip-flop

# Nome

BITCLEAR — Clear a Bit

## Block Screenshot

## Contents


- Clear a Bit
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

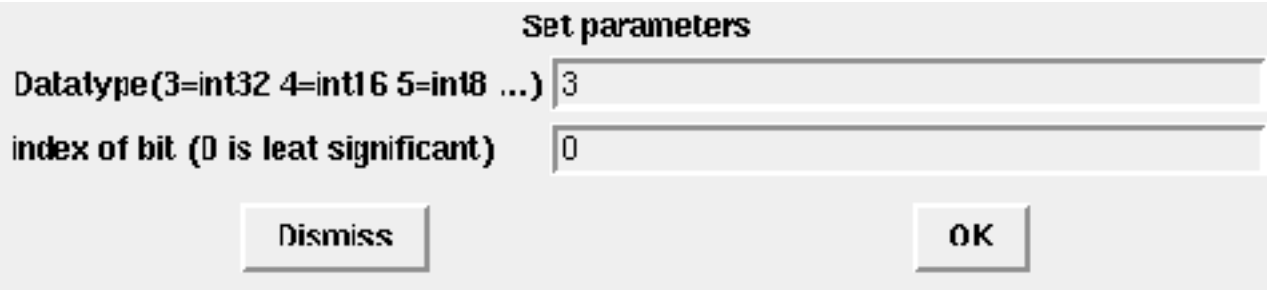

## Palette

- Integer palette

## Description

This blocks set the specified bit of the integer input to 0. The user can specify the bit in the field:"index of bit". Bit 0 is the least significant bit.

## Dialog box




- **Datatype(3=int32 4=int16 5=int8 ...)**

  It indicates the type of the input/output data. It support all the integer datatype, number must be between 3 and 8.

  Properties : Type 'vec' of size 1.

- **index of bit (0 is least significant)**

  It indicate the index of the bit to clear. When the type is int32 or uint32 the number must be positive and less than 32. When the type is int16 or uint16 the number must be positive and less than 16. When the type is int8 or uint8 the number must be positive and less than 8.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 3**
- **regular outputs:**

  **- port 1 : size [1,1] / type 3**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *bit_clear_32*

## Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/BITCLEAR.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/bit_clear_32.c
- SCI/modules/scicos_blocks/src/c/bit_clear_16.c
- SCI/modules/scicos_blocks/src/c/bit_clear_8.c

## See also

- BITSET - BITSET Set a Bit

## Authors

**Fady NASSIF** - INRIA

# Nome

BITSET — Set a Bit

## Block Screenshot

## Contents


- Set a Bit
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

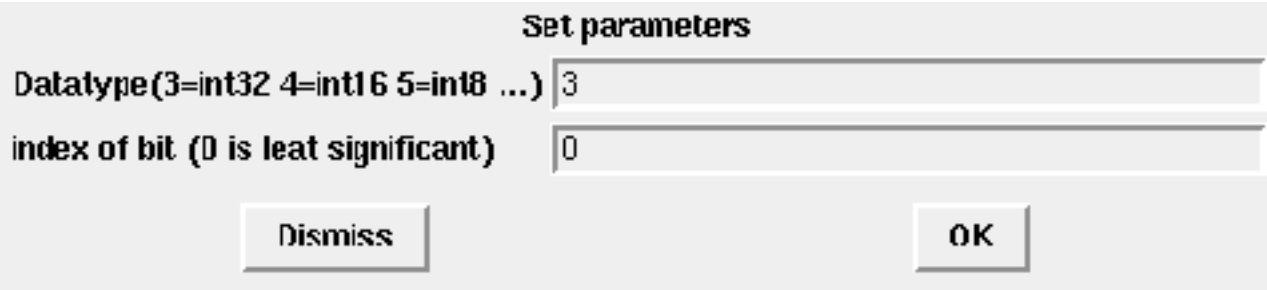

## Palette

- Integer palette

## Description

This blocks set the specified bit of the integer input to 1. The user can specify the bit in the field:"index of bit". Bit 0 is the least significant bit.

## Dialog box




- **Datatype(3=int32 4=int16 5=int8 ...)**

  It indicates the type of the input/output data. It support all the integer datatype, number must be between 3 and 8.

  Properties : Type 'vec' of size 1.

- **index of bit (0 is leat significant)**

  It indicate the index of the bit to clear. When the type is int32 or uint32 the number must be positive and less than 32. When the type is int16 or uint16 the number must be positive and less than 16. When the type is int8 or uint8 the number must be positive and less than 8.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 3**
- **regular outputs:**

  **- port 1 : size [1,1] / type 3**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *bit_set_32*

## Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/BITSET.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/bit_set_32.c
- SCI/modules/scicos_blocks/src/c/bit_set_16.c
- SCI/modules/scicos_blocks/src/c/bit_set_8.c

## See also

- BITCLEAR - BITCLEAR Clear a Bit

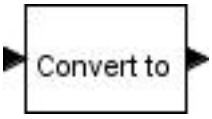
## Authors

**Fady NASSIF** - INRIA

# Nome

CONVERT — Data Type Conversion

## Block Screenshot




## Contents


- CONVERT Data Type Conversion
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Integer palette

## Description

This block converts an input signal of any data type to a specified data type. The input can be real, complex or integer. When the output is an integer and when overflow occurs the block three different forms of results :

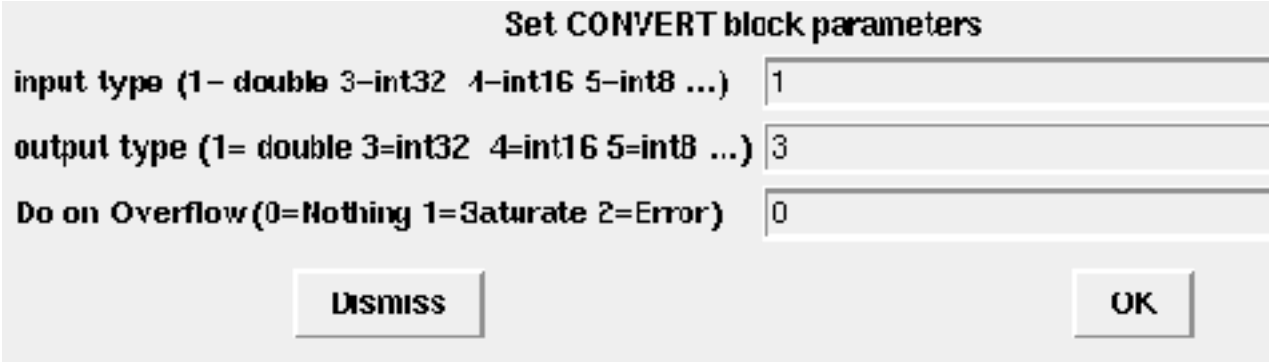
1- A normal non saturated result.

2- A saturated result.

3- An error message warning the user about the overflow..

The user can select one of these three forms by setting the "DO ON OVERFLOW" field to 0,1 or 2.

## Dialog box



- **input type (1= double 3=int32 4=int16 5=int8 ...)**

  It indicates the input data type, it can be a double or an integer.

  Properties : Type 'vec' of size 1.

- **output type (1= double 3=int32 4=int16 5=int8 ...)**

  It indicates the output data type, it can be a double or an integer.

  Properties : Type 'vec' of size 1.

- **Do on Overflow(0=Nothing 1=Saturate 2=Error)**

  When this parameter is set to zero the result is similar to a normal multiplication of two integer matrix. When it is set to 1, on overflow the block saturate the result. When it is set to 2, on overflow an error message box appears.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 3**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *convert*

## Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/CONVERT.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/convert.c

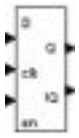
## Authors

**Fady NASSIF** - INRIA

# Nome

DFLIPFLOP — D flip-flop

## Block Screenshot




## Contents


- D flip-flop
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “See also”
  - “Authors”


## Palette

- Integer palette

## Description

The DFLIPFLOP block outputs the input state when the enable is set and on the rising edge of the clock. The input is D the enable is en and the clock is clk. Q and !Q are the outputs of this block. This block is almostly used with digital number, the input data type is int8.

The truth table of this block is

| en | D | Q | !Q |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

**- port 1 : size [1,1] / type 5**

**- port 2 : size [1,1] / type 1**

**- port 3 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 5**

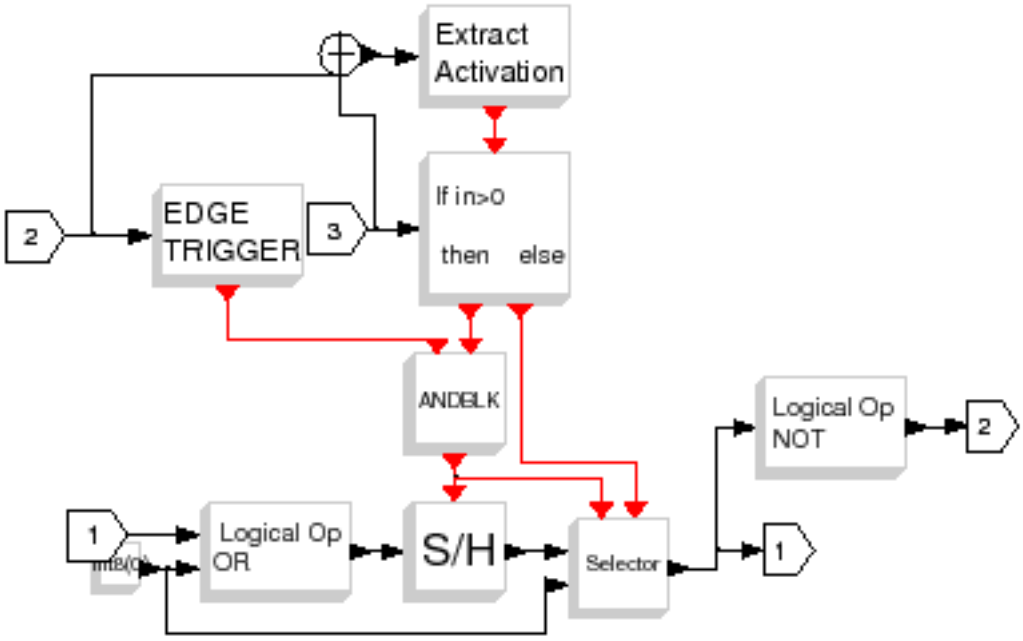
  **- port 2 : size [1,1] / type 5**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/DFLIPFLOP.sci

## Compiled Super Block content




## See also

- DLATCH - D latch flip-flop
- SRFLIPFLOP - SR flip-flop
- JKFLIPFLOP - JK flip-flop

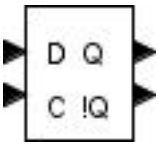
## Authors

**Fady NASSIF** - INRIA

# Nome

DLATCH — D latch flip-flop

## Block Screenshot




## Contents


- D latch flip-flop
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “See also”
  - “Authors”


## Palette

- Integer palette

## Description

This block outputs the input state when the input gate is high. The input is D the enable is C. Q and !Q are the outputs of this block. This block is almost used with digital number, the input data type is int8.

The truth table of this block is

| C | D | Q | !Q |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

**- port 1 : size [1,1] / type 5**

**- port 2 : size [1,1] / type -1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 5**

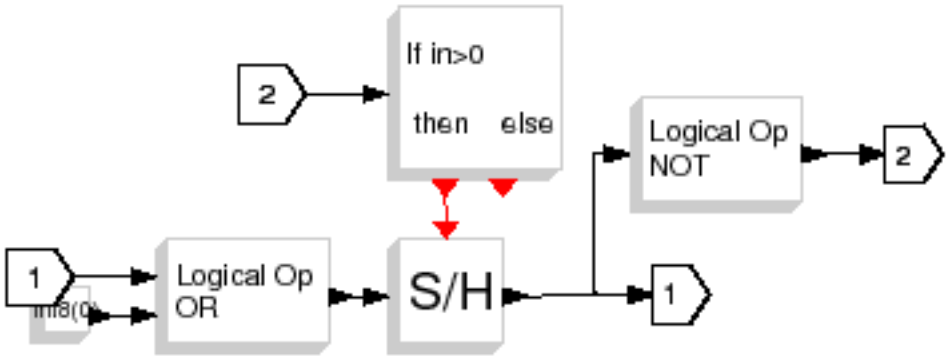
  **- port 2 : size [1,1] / type 5**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/DLATCH.sci

## Compiled Super Block content




## See also

- DFLIPFLOP - D flip-flop
- SRFLIPFLOP - SR flip-flop
- JKFLIPFLOP - JK flip-flop

## Authors

**Fady NASSIF** - INRIA

# Nome

EXTRACTBITS — EXTRACTBITS

## Block Screenshot

## Contents


- EXTRACTBITS
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Integer palette

## Description

This block outputs a contiguous selection of bits from the input integer number.The Bits to extract defines the method by which the user select the output bits.

1- When "Upper Half" is selected the block outputs the half of the input that contain the most significant bit. In this case the third parameter "number of bits or index of bit" is ignored.

2- When "Lower Half" is selected the block outputs the half of the input that contain the least significant bit. In this case the third parameter "number of bits or index of bit" is ignored.

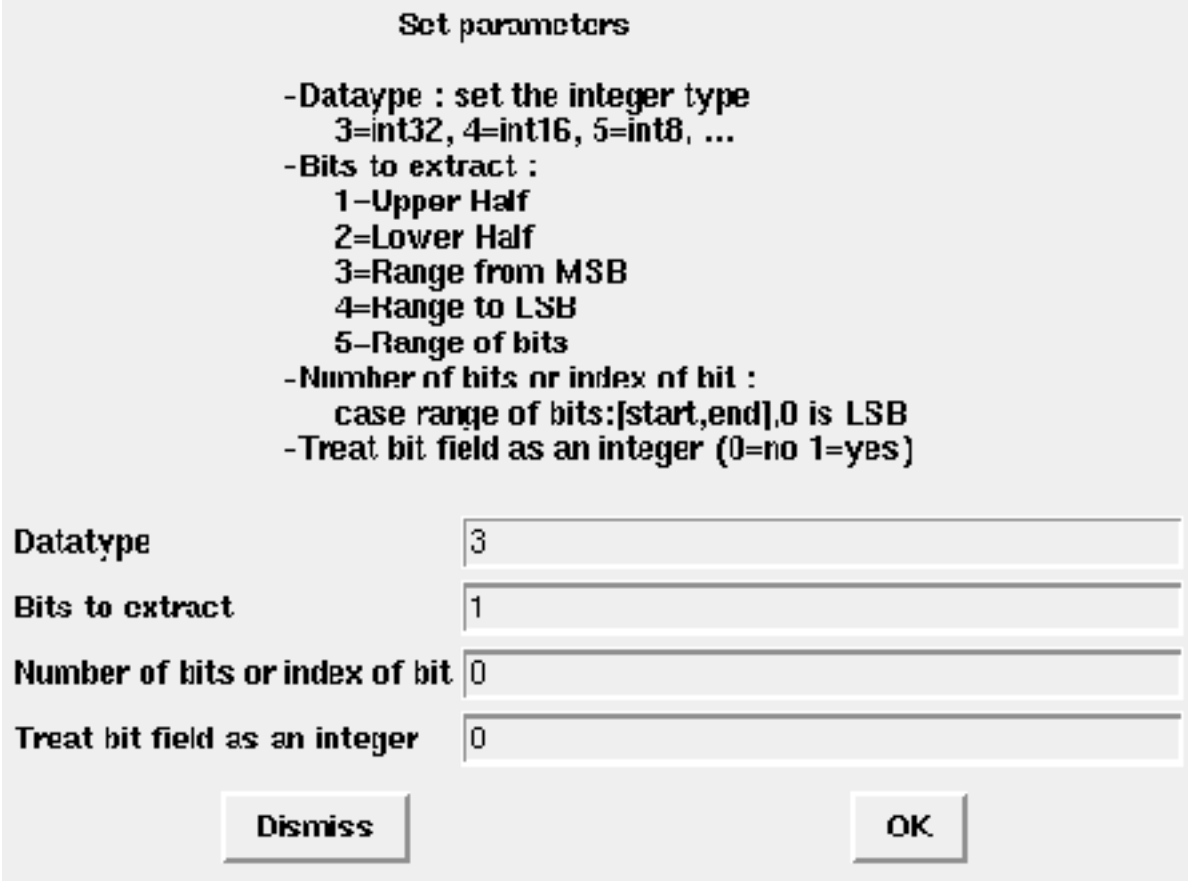
3- When "Range starting with most significant bit" is selected the block outputs certain number of bits of the input that contain the most significant bit. In this case the third parameter "number of bits or index of bit" defines the number of bits to extract.

4- When "Range ending with least significant bit" is selected the block outputs certain number of bits of the input that contain the least significant bit. In this case the third parameter "number of bits or index of bit" defines the number of bits to extract.

5- When "Range of bits" is selected the block outputs a range of bits of the input. In this case the third parameter "number of bits or index of bit" defines the range of bits to extract, it must be a vector with the format [start,end].

The extracted value depends on the forth parameter "Treat bit field as an integer". When it is set to 0 the input scaling is used to determine the output scaling. When it is set to 1, only the extracted bits forms the output number.

## Dialog box




- **Datatype(3=int32 4=int16 5=int8 ...)**

  It indicates the type of the input/output data. It support all the integer datatype, number must be between 3 and 8.

  Properties : Type 'vec' of size 1.

- **Bits to extract(1=Upper Half 2=Lower Half 3=Range starting with most significant bit 4=Range ending with least significant bit 5=Range of bits)**

  It indicates the mode used to extract bits from the input data.

  Properties : Type 'vec' of size 1.

- **number of bits or index of bit (case range of bits:[start,end],0 is least significant bit)**

  When the "Bits to extract" field is set to 3 or 4, this parameter is used to determine the number of bits to extract and it must be a number. When the "Bits to extract" field is set to 5 ,this parameter is used to determine range of bits to extract and it must have the [start,end] form vector. When the "Bits to extract" field is set to 1 or 2, this parameter is ignored.

  Properties : Type 'vec' of size -1.

- **Treat bit field as an integer(0=no 1=yes)**

  It indicates the scaling mode to use on the output bits selection.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 3**
- **regular outputs:**

  **- port 1 : size [1,1] / type 3**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *extract_bit_32_UH0*

## Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/EXTRACTBITS.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/extract_bit_32_UH0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_UH1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u32_UH1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_LH.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_MSB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u32_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_LSB.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_RB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_32_RB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u32_RB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_UH0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_UH1.c

- SCI/modules/scicos_blocks/src/c/extract_bit_u16_UH1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_LH.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_MSB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u16_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_LSB.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_RB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_16_RB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u16_RB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_UH0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_UH1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u8_UH1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_LH.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_MSB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u8_MSB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_LSB.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_RB0.c
- SCI/modules/scicos_blocks/src/c/extract_bit_8_RB1.c
- SCI/modules/scicos_blocks/src/c/extract_bit_u8_RB1.c

## See also

- BITSET - BITSET Set a Bit
- BITCLEAR - BITCLEAR Clear a Bit

## Authors

**Fady NASSIF** - INRIA

# Nome

INTMUL — Integer matrix multiplication

## Block Screenshot

## Contents


- Integer matrix multiplication
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Integer palette

## Description

The INTMUL block computes the matrix multiplication of two integers inputs matrices.The number of rows of the second matrix must be equal to the number of columns of the first matrix. The output is a matrix where the number of rows is equal to the number of rows of the first input matrix and the number of columns is equal to the number of columns of the second input matrix. This block support all the integer data type.

On overflow, the result can take different forms:

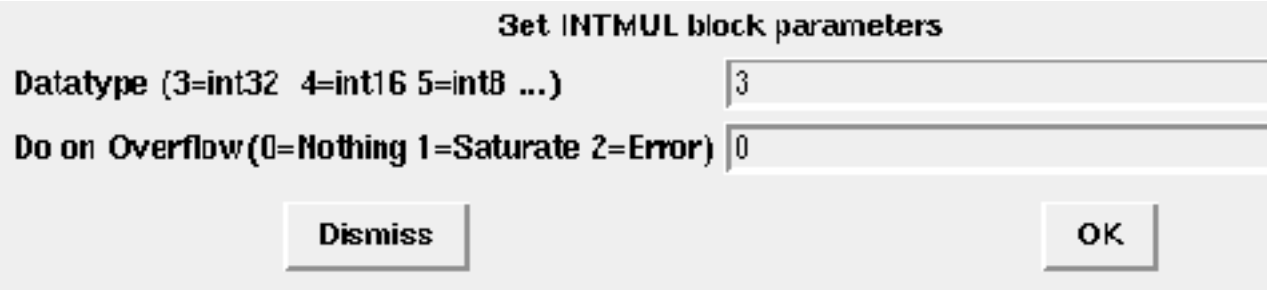
1- A normal non saturated result.

2- A saturated result.

3- An error message warning the user about the overflow.

The user can select one of these three forms by setting the "DO ON OVERFLOW" field to 0,1 or 2.

## Dialog box




- **Datatype (3=int32 4=int16 5=int8 ...)**

  It indicates the type of the input/output data. It support all the integer datatype, number must be between 3 and 8.

  Properties : Type 'vec' of size 1.

- **Do on Overflow(0=Nothing 1=Saturate 2=Error)**

  When this parameter is set to zero the result is similar to a normal multiplication of two integer matrix. When it is set to 1, on overflow the block saturate the result. When it is set to 2, on overflow an error message box appears.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 3**

  **- port 2 : size [-2,-3] / type 3**

- **regular outputs:**

  **- port 1 : size [-1,-3] / type 3**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *matmul_i32*

## Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/INTMUL.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/matmul_i32n.c
- SCI/modules/scicos_blocks/src/c/matmul_i16n.c
- SCI/modules/scicos_blocks/src/c/matmul_i8n.c
- SCI/modules/scicos_blocks/src/c/matmul_ui32n.c
- SCI/modules/scicos_blocks/src/c/matmul_ui16n.c
- SCI/modules/scicos_blocks/src/c/matmul_ui8n.c
- SCI/modules/scicos_blocks/src/c/matmul_i32s.c
- SCI/modules/scicos_blocks/src/c/matmul_i16s.c
- SCI/modules/scicos_blocks/src/c/matmul_i8s.c
- SCI/modules/scicos_blocks/src/c/matmul_ui32s.c
- SCI/modules/scicos_blocks/src/c/matmul_ui16s.c
- SCI/modules/scicos_blocks/src/c/matmul_ui8s.c
- SCI/modules/scicos_blocks/src/c/matmul_i32e.c
- SCI/modules/scicos_blocks/src/c/matmul_i16e.c
- SCI/modules/scicos_blocks/src/c/matmul_i8e.c
- SCI/modules/scicos_blocks/src/c/matmul_ui32e.c
- SCI/modules/scicos_blocks/src/c/matmul_ui16e.c
- SCI/modules/scicos_blocks/src/c/matmul_ui8e.c

## See also

- MATMUL - MATMUL Matrix Multiplication

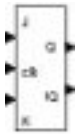
## Authors

**Fady NASSIF** - INRIA

# Nome

JKFLIPFLOP — JK flip-flop

## Block Screenshot



## Contents


- JK flip-flop
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “See also”
  - “Authors”


## Palette

- Integer palette

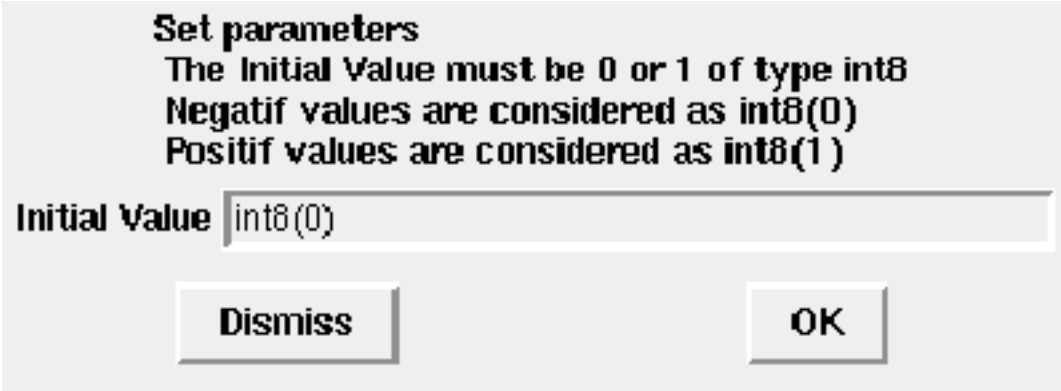
## Description

The JK flip flop is the most versatile of the basic flip-flops. It has two inputs traditionally labeled J and K. When J and K are different, the output takes the value of J at the next falling edge. When J and K are both low, no change occurs in the output state, when they are both high the output will toggle from one state to other. It can perform the functions of the set/reset flip-flop and has the advantage that there are no ambiguous states. It can also act as a T flip-flop to accomplish toggling action if J and K are tied together. This toggle application finds extensive use in binary counters. This block is almost used with digital number, the input data type is int8.

The truth table of this block is

| J | K | Q(t) | !Q(t) |
|---|---|---|---|
| 0 | 0 | Q(t-1) | !Q(t-1) |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | !Q(t-1) | Q(t-1) |

## Dialog box




- **Initial Value**

  Initial Value of the state Q.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 5**

  **- port 2 : size [1,1] / type 1**

  **- port 3 : size [1,1] / type 5**
- **regular outputs:**

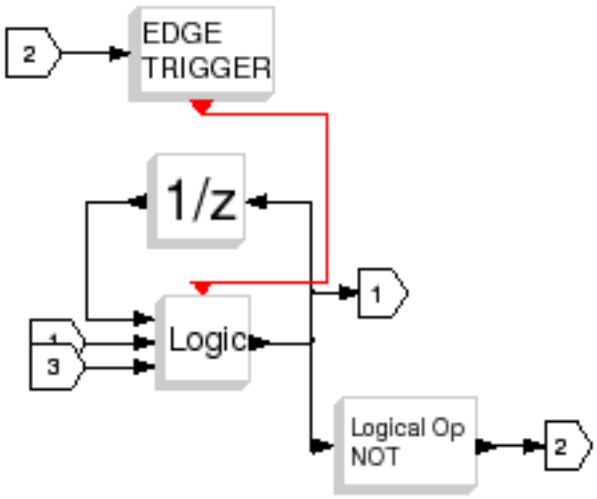
  **- port 1 : size [1,1] / type 5**

  **- port 2 : size [1,1] / type 5**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/JKFLIPFLOP.sci

## Compiled Super Block content




## See also

- DLATCH - D latch flip-flop
- DFLIPFLOP - D flip-flop
- SRFLIPFLOP - SR flip-flop

## Authors

**Fady NASSIF** - INRIA

# Nome

LOGIC — Combinational Logic

## Block Screenshot

## Contents


- Combinational Logic
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Integer palette

## Description

This block implements a standard truth table for modeling programming array, digital circuit and any other boolean expressions. The user can specify a matrix that defines all the possible block output in the Truth table field. Each row of the matrix contains the output of different combination of input elements. The number of rows must be a power of two, it defines the number of inputs using the equation:

number of row = 2 (number of input)

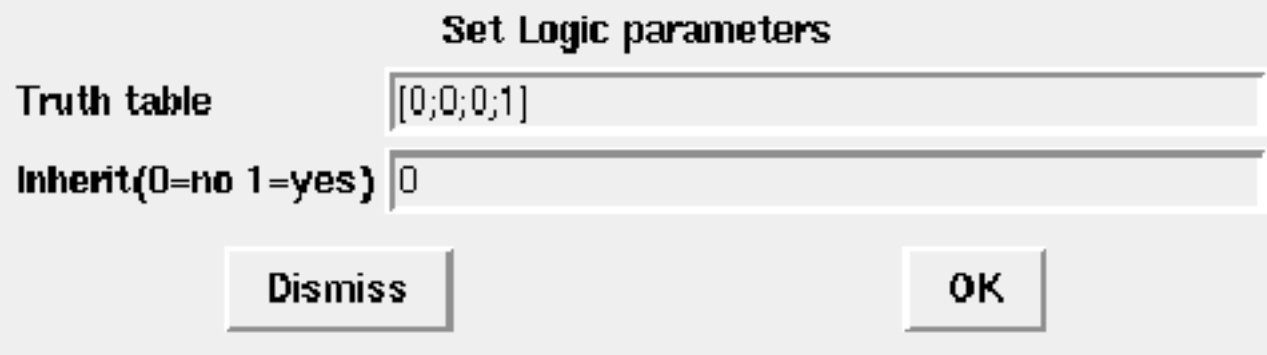
The number of outputs is equal to the number of columns of the matrix.

This block support only the int8 data type. When the input is positive, the input is considered as logical 1, When it is negative or zero it is considered as logical 0.

This block can be activated by an implicit input event or it can inherit the clock from the regular input.

This block is used to implement SR and JK flip-flops.

## Dialog box




- **Truth table**

  The matrix of outputs. For more information see the description part.

  Properties : Type 'mat' of size [-1,-2].

- **Inherit(0=no 1=yes)**

  Specifies if the clock is inherit or not.

  Properties : Type 'vec' of size 1.

## Example

```
The easiest example to consider is the OR example. In this case we have two inpu
|-----------|-----------|-----------|
|  input 1  |  input 2  |  output   |
|-----------|-----------|-----------|
|     0     |     0     |     0     |
|-----------|-----------|-----------|
|     0     |     1     |     1     |
|-----------|-----------|-----------|
|     1     |     0     |     1     |
|-----------|-----------|-----------|
|     1     |     1     |     1     |
|-----------|-----------|-----------|
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 5**

  **- port 2 : size [1,1] / type 5**

- **regular outputs:**

  **- port 1 : size [1,1] / type 5**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *logic*

## Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/LOGIC.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/logic.c

## See also

- SRFLIPFLOP - SR flip-flop
- JKFLIPFLOP - JK flip-flop

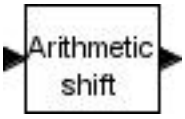
## Authors

**Fady NASSIF** - INRIA

# Nome

SHIFT — Shift Bits

## Block Screenshot




## Contents


- Shift Bits
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Integer palette

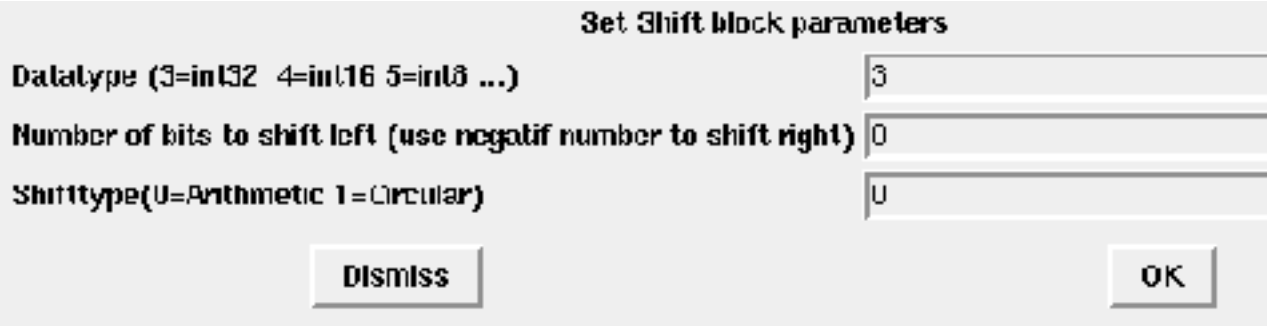
## Description

This block shifts the bits of the input signal. In this operation the digits are moved to the right or to the left. The user can choose the rule to shifts the bits. It can be normal or cycle by setting the "Shifttype" parameter to "0" or "1".

When the Shifttype is 0, an arithmetic shift is applied to the input signal. In this case, the bits that are shifted out of either end are discarded. Zeros are shifted in on the right, in the case of left shift; in the case of right shifts, copies of the sign bit is shifted in on the left.

When the "Shifttype" is 1,a circular shift is applied to the input signal. In this case, the bits are rotated as if the left and right ends of the register are joined. The value that is shifted in on the right during a left-shift is whatever values was shifted out on the left, and vice versa.

## Dialog box




- **Datatype (3=int32 4=int16 5=int8 ...)**

  It indicates the type of the input/output data. It support all the integer datatype, number must be between 3 and 8.

  Properties : Type 'vec' of size 1.

- **Number of bits to shift left (use negative number to shift right)**

  It indicates the number of bits the input signal is shifted. A positive value indicates a shift left, negative values indicates shift right.

  Properties : Type 'vec' of size 1.

- **Shifttype(0=Arithmetic 1=Circular)**

  It indicate the rule used to shift the bits. It can be arithmetic or circular. When the Shifttype is normal, an arithmetic shift is applied to the input signal. In this case, the bits that are shifted.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 3**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 3**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *shift_32_LA*

## Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/SHIFT.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/shift_32_LA.c
- SCI/modules/scicos_blocks/src/c/shift_32_LC.c
- SCI/modules/scicos_blocks/src/c/shift_32_RA.c
- SCI/modules/scicos_blocks/src/c/shift_u32_RA.c
- SCI/modules/scicos_blocks/src/c/shift_32_RC.c
- SCI/modules/scicos_blocks/src/c/shift_16_LA.c
- SCI/modules/scicos_blocks/src/c/shift_16_LC.c
- SCI/modules/scicos_blocks/src/c/shift_16_RA.c
- SCI/modules/scicos_blocks/src/c/shift_u16_RA.c
- SCI/modules/scicos_blocks/src/c/shift_16_RC.c
- SCI/modules/scicos_blocks/src/c/shift_8_LA.c
- SCI/modules/scicos_blocks/src/c/shift_8_LC.c
- SCI/modules/scicos_blocks/src/c/shift_8_RA.c
- SCI/modules/scicos_blocks/src/c/shift_u8_RA.c
- SCI/modules/scicos_blocks/src/c/shift_8_RC.c

## See also

- BITSET - BITSET Set a Bit
- BITCLEAR - BITCLEAR Clear a Bit

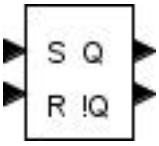
## Authors

**Fady NASSIF** - INRIA

# Nome

SRFLIPFLOP — SR flip-flop

## Block Screenshot




## Contents


- SR flip-flop
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “See also”
  - “Authors”


## Palette

- Integer palette

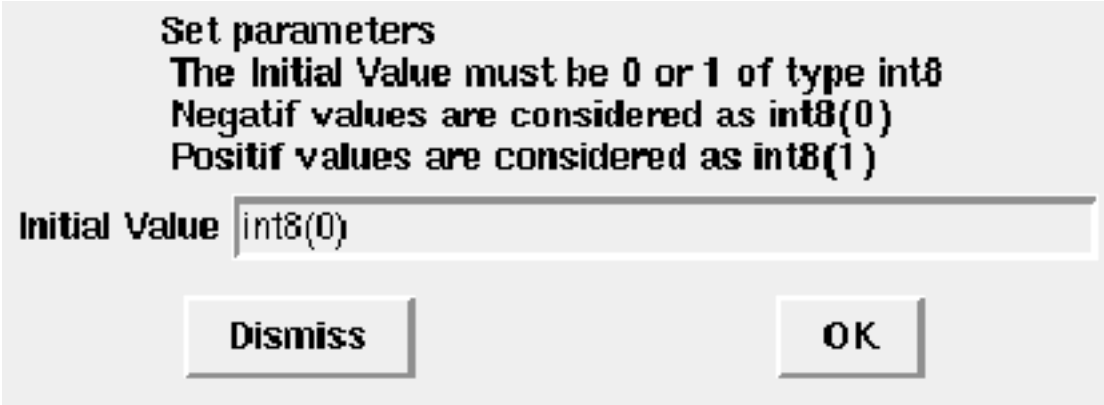
## Description

This block describe the simplest and the most fundamental latch the SR flip flop. Where S and R are the input and Q and !Q are the outputs.If S (Set) is pulsed high while R is held low, then the Q output is forced high, and stays high when S returns low; similarly, if R (Reset) is pulsed high while S is held low, then the Q output is forced low, and stays low when R returns low. When both are low, Q(t) takes the same state as Q(t-1). When they are both high, both Q and !Q take the low values we are in an unstable state. Practically we have to avoid this case.This block is almost used with digital number, the input data type is int8.

The truth table of this block is

| S | R | Q(t) | !Q(t) |
|---|---|---|---|
| 0 | 0 | Q(t-1) | !Q(t-1) |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| -> This case is to avoid | | | |

## Dialog box




- **Initial Value**

  Initial Value of the state Q.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 5**

  **- port 2 : size [1,1] / type 5**
- **regular outputs:**

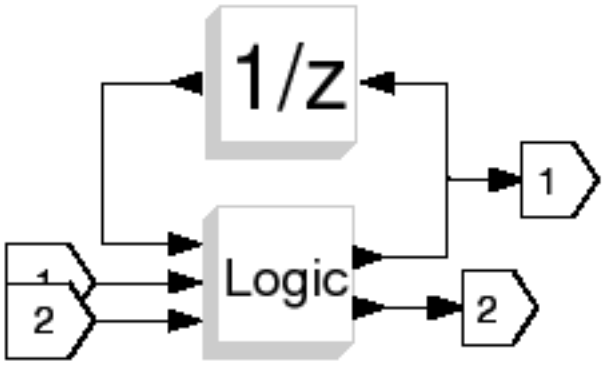
  **- port 1 : size [1,1] / type 5**

  **- port 2 : size [1,1] / type 5**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/IntegerOp/SRFLIPFLOP.sci

## Compiled Super Block content




## See also

- DLATCH - D latch flip-flop
- DFLIPFLOP - D flip-flop
- JKFLIPFLOP - JK flip-flop

## Authors

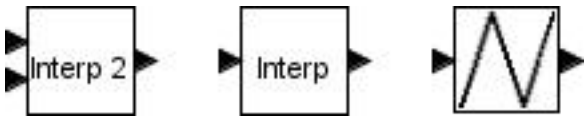
**Fady NASSIF** - INRIA

# 11. Lookup tables palette

# Nome

Lookuptables_pal — Lookup tables palette

## Block Screenshot




## Module

- xcos

## Description

The lookup tables palette includes blocks that compute output approximations from inputs.

## Blocks

- INTRP2BLK_f — 2D interpolation
- INTRPLBLK_f — Interpolation
- LOOKUP_f — Lookup table

# Nome

INTRP2BLK_f — 2D interpolation

## Block Screenshot

## Contents


- 2D interpolation
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

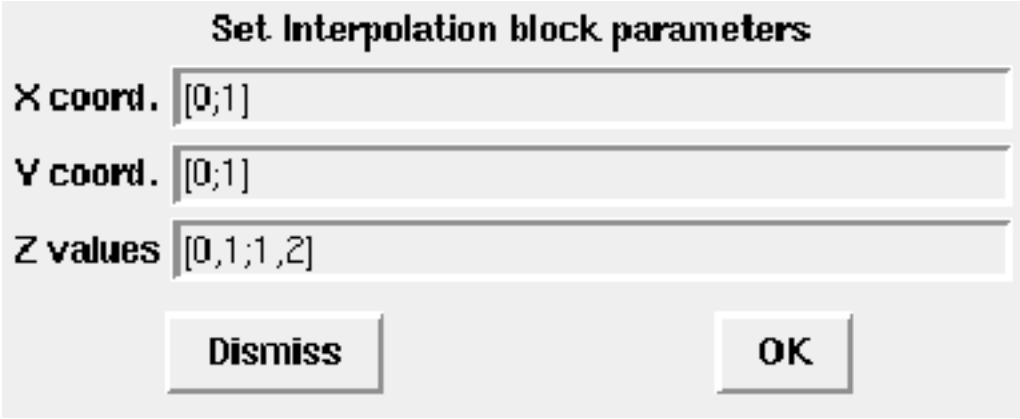

## Palette

- Lookup tables palette

## Description

The output of this block is a function of the inputs obtained by bilinear interpolation. This block has two scalar inputs and a single scalar output. The and give respectively the coordinate and the coordinate of the -th data point to be interpolated and its value.

## Dialog box




- **X coord.**

  an n-vector (strictly increasing).

Properties : Type 'vec' of size -1

- **Y coord.**

  an m-vector (strictly increasing).

  Properties : Type 'vec' of size -1

- **Z values**

  an $m \times n$ matrix.

  Properties : Type 'mat' of size [-1,-1]

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *intrp2*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/INTRP2BLK_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/intrp2.f (Type 1)

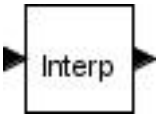
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

INTRPLBLK_f — Interpolation

## Block Screenshot




## Contents


- Interpolation
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

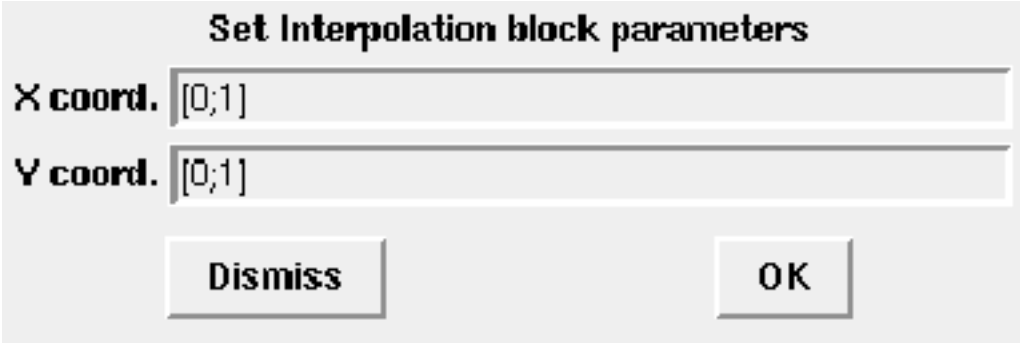

## Palette

- Lookup tables palette

## Description

The output of this block is a function of the input obtained by linear interpolation. This block has a single scalar input and a single scalar output port. The coord. and coord. give respectively the coordinate and the coordinate of the data points to be interpolated. coord must be strictly increasing.

## Dialog box




- **X coord.**

  A vector (strictly increasing).

  Properties : Type 'vec' of size -1

- **Y coord.**

  A vector (same size as $X$ coord).

  Properties : Type 'vec' of size -1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *intrpl*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/INTRPLBLK_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/intrpl.f (Type 0)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

LOOKUP_f — Lookup table

## Block Screenshot

## Contents


- Lookup table
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Lookup tables palette

## Description

This block realizes a non-linear function defined using a graphical editor.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *lookup*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/LOOKUP_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/lookup.f (Type 0)

## Authors

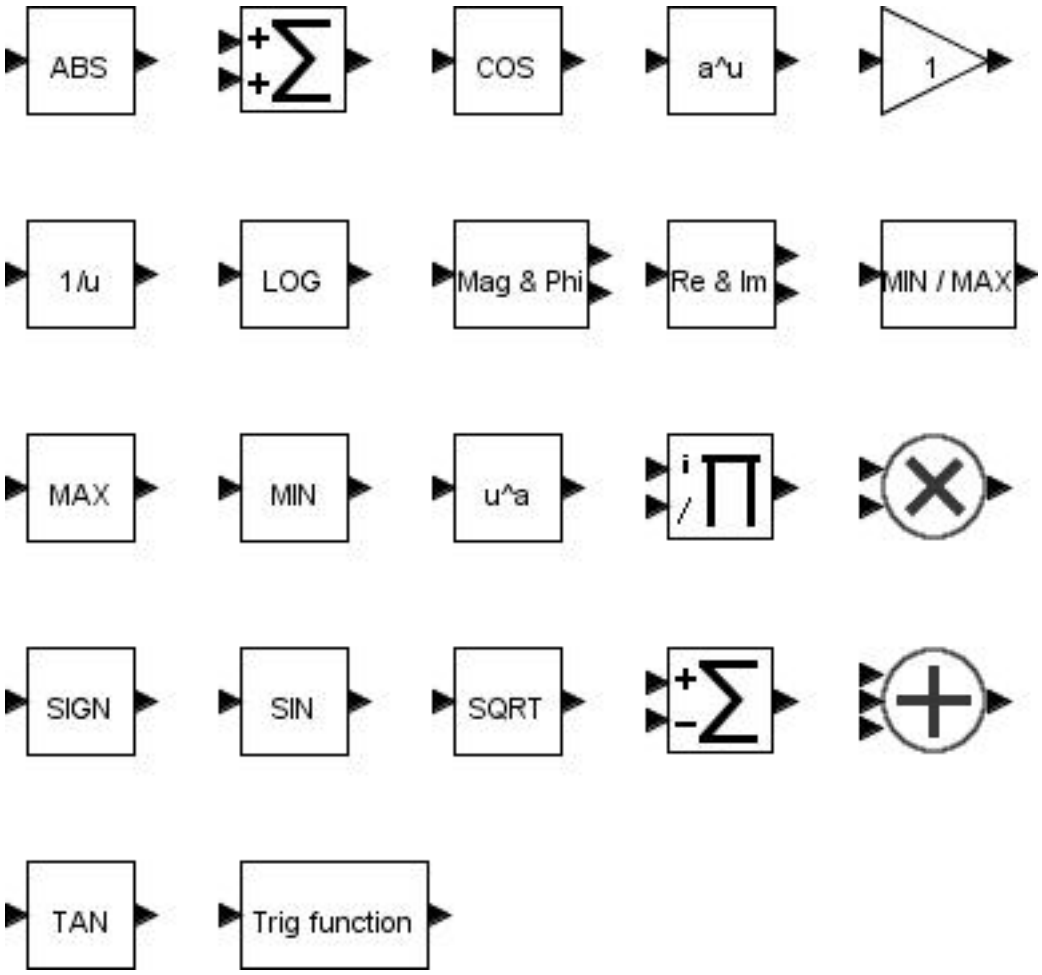
**Ramine Nikoukhah** - INRIA

# 12. Math operations palette

# Nome

Mathoperations_pal — Math operations palette

## Block Screenshot




## Module

- xcos

## Description

The Math operations palette contains blocks that model general mathematical functions.

## Blocks

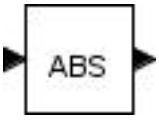
- ABS_VALUE — Absolute value
- BIGSOM_f — Sum
- COSBLK_f — Cosinus Block
- EXPBLK_m — Exponential
- GAINBLK_f — Gain
- INVBLK — Inverse

- LOGBLK_f — Common logarithm
- MATMAGPHI — Complex to Magnitude and Angle Conversion
- MATZREIM — Complex decomposition
- MAX_f — MAX
- MAXMIN — Max and Min
- MIN_f — MIN
- POWBLK_f — Array power
- PROD_f — Multiplication
- PRODUCT — Product
- SIGNUM — Signum
- SINBLK_f — Sinus block
- SQRT — Square root
- SUM_f — Addition
- SUMMATION — Summation
- TANBLK_f — Tangent block
- TrigFun — Trigonometric function

# Nome

ABS_VALUE — Absolute value

## Block Screenshot




## Contents


- Absolute value
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

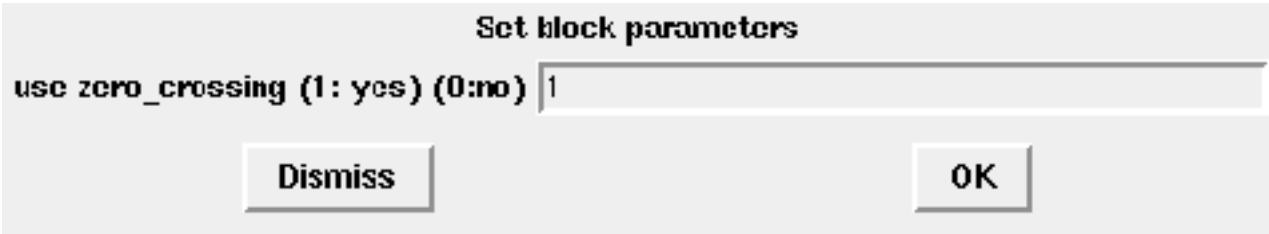

## Palette

- Math operations palette

## Description

The Abs block outputs the absolute value of the input.

## Dialog box




- **use zero_crossing**

  Select to enable zero crossing detection.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *absolute_value*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/ABS_VALUE.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/absolute_value.c (Type 4)

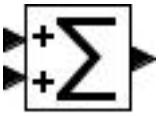
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

BIGSOM_f — Sum

## Block Screenshot



## Contents


- Sum
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

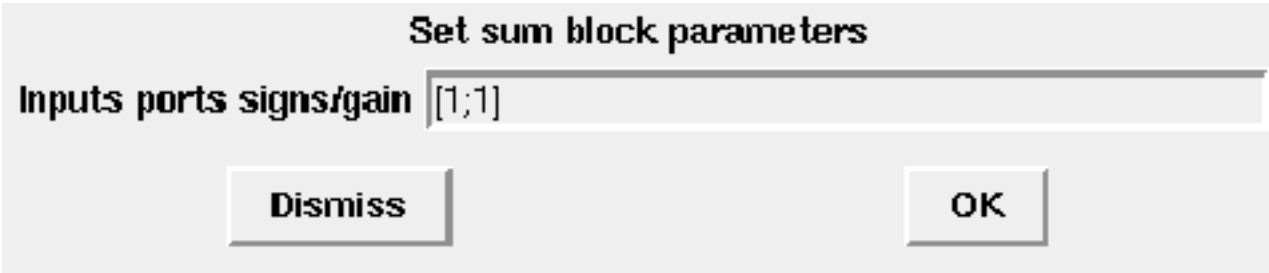

## Palette

- Math operations palette

## Description

The Sum block performs addition on its inputs. This block can add scalar or vector inputs.

## Dialog box




- **Inputs ports signs/gain**

  Set sign and a gain for each inputs.

  Properties : Type 'vec' of size -1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no

- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**

  **- port 2 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sum*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/BIGSOM_f.sci

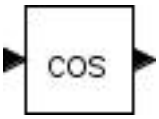
## Computational function

- SCI/modules/scicos_blocks/src/c/sum.c (Type 2)

# Nome

COSBLK_f — COSBLK

## Block Screenshot




## Contents


- COSBLK
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”


## Palette

- Math operations palette

## Description

Description

$$y = \cos(u)$$

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *cosblk*

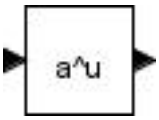
## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/COSBLK_f.sci

# Nome

EXPBLK_m — Exponential

## Block Screenshot




## Contents


- Exponential
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”


## Palette

- Math operations palette

## Description

This block realizes . The input and output port sizes are determined by the compiler.

## Dialog box

- **a**

  A real positive scalar.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *expblk_m*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/EXPBLK_m.sci

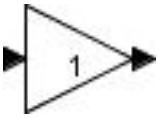
## Computational function

- SCI/modules/scicos_blocks/src/c/expblk_m.c (Type 4)

# Nome

GAINBLK_f — Gain

## Block Screenshot




## Contents


- Gain
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

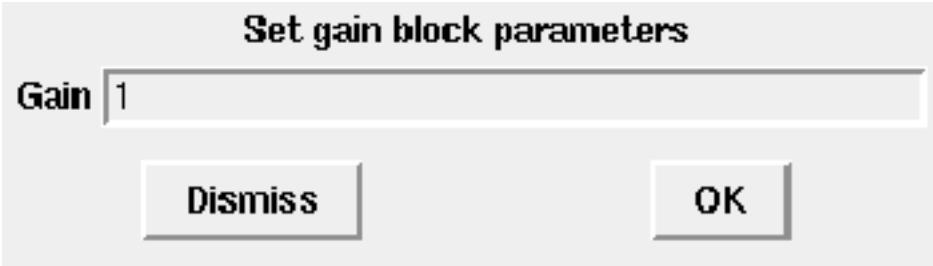

## Palette

- Math operations palette

## Description

The GAINBLK computes the product of a square matrix A by the input matrix U, where the number of rows/cols of A is equal to the number of rows of U.

## Dialog box




- **Gain**

  This parameter defined the square matrix A.

  Properties : Type 'mat' of size [-1,-1].

## Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *gain*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/GAINBLK_f.sci

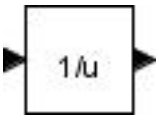
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

INVBLK — Inverse

## Block Screenshot




## Contents


- Inverse
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”


## Palette

- Math operations palette

## Description

This block computes . The input (output) size is determined by the context.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no

- **name of computational function:** *invblk4*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/INVBLK.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/invblk4.c (Type 4)

# Nome

LOGBLK_f — Log

## Block Screenshot

## Contents


- Log
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

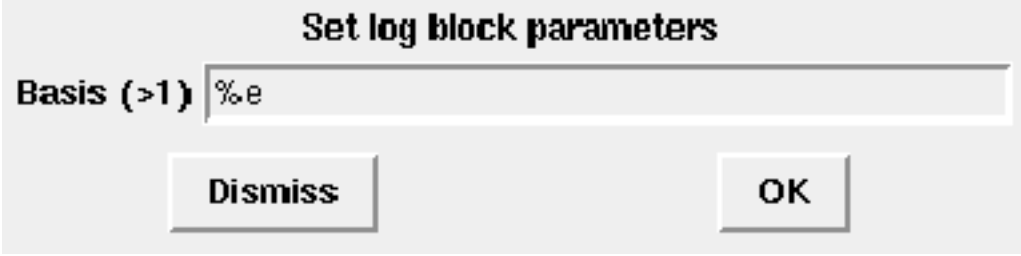

## Palette

- Math operations palette

## Description

This block realizes . The input and output port sizes are determined by the context.

## Dialog box




- **Basis**

  A real scalar greater than 1.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *logblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/LOGBLK_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/logblk.f (Type 0)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

MATMAGPHI — Complex to Magnitude and Angle Conversion

## Block Screenshot

## Contents


- Complex to Magnitude and Angle Conversion
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Math operations palette

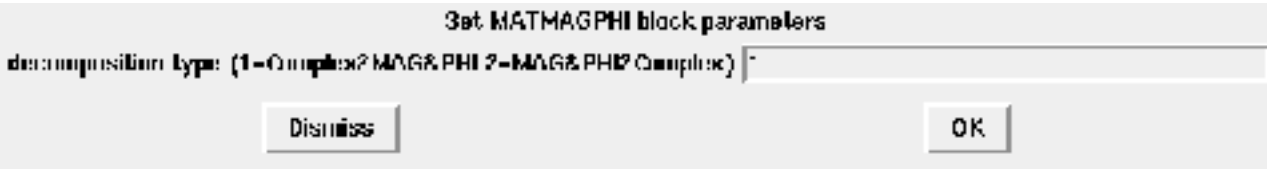
## Description

MATMAGPHI Block has two types of decomposotions.

When the type is set to one, the block converts a complex number to the magnitude and the radian angle, in this case the input is complex and the outputs are real double. If the input is real double, the angle will be zero or PI and the magnitude will be equal to the absolute of the input number.

When the type is set to two, the block outputs a complex number given the magnitude and the radian angle. In this case the inputs are real double and the output is complex.

## Dialog box




- **decomposition type (1=Complex2MAG&PHI 2=MAG&PHI2Complex)**

  It indicates the rule of the conversion.

Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 2**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**

  **- port 2 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *matz_abs*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATMAGPHI.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/matz_abs.c
- SCI/modules/scicos_blocks/src/c/matz_absc.c

## See also

- MATZREIM - Complex decomposition

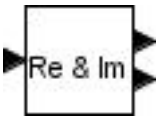
## Authors

**Fady NASSIF** - INRIA

# Nome

MATZREIM — Complex decomposition

## Block Screenshot




## Contents


- Complex decomposition
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Math operations palette

## Description

This block decomposes a complex number by separating the real and imaginary parts or compose a complex number by joining the two parts. The user can select even to separate or to join real and imaginary part by setting the decomposition type to 1 or 2. When it is set to 1, the input is a complex matrix and the outputs are the real and imaginary parts of the input. When it set to 2, The inputs are two real matrices, the output is a complex number with real part the first input and imaginary part the second input.

## Dialog box

- **decomposition type (1=Complex2Real&Imag 2=Real&Imag2Complex)**

  Indicates the type to use for the decomposition. See the description part for more information.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 2**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**

  **- port 2 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *matz_reim*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATZREIM.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/matz_reim.c
- SCI/modules/scicos_blocks/src/c/matz_reimc.c

## See also

- MATMAGPHI - MATMAGPHI Complex to Magnitude and Angle Conversion

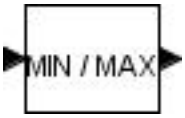
## Authors

**Fady NASSIF** - INRIA

# Nome

MAXMIN — Max and Min

## Block Screenshot




## Contents


- Max and Min
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

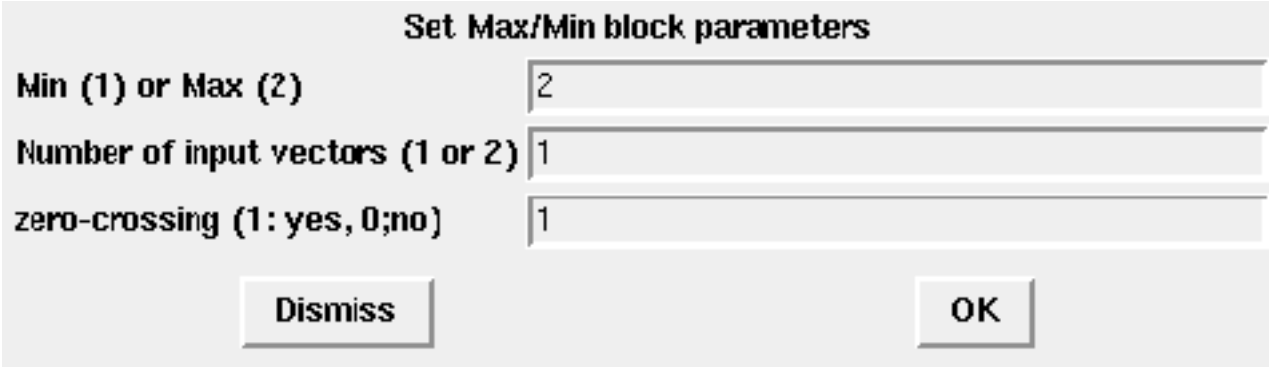

## Palette

- Math operations palette

## Description

The MinMax block outputs either the minimum or the maximum element or elements of the inputs. You can choose the function to apply by selecting one of the choices from the Function parameter list.

## Dialog box




- **Min or Max**

  The function (min or max) to apply to the input.

  Properties : Type 'vec' of size 1.

- **Number of input vectors**

The number of inputs to the block.

Properties : Type 'vec' of size 1.

- **zero-crossing**

  Select to enable zero crossing detection to detect minimum and maximum values.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *minmax*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/MAXMIN.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/minmax.c (Type 4)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

MAX_f — MAX

## Block Screenshot

## Contents


- MAX
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”


## Palette

- Math operations palette

## Description

That block find the max value in the elements of its input vector.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no

- **name of computational function:** *maxblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/MAX_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/maxblk.f (Type 0)

# Nome

MIN_f — MIN

## Block Screenshot

## Contents


- MIN
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”


## Palette

- Math operations palette

## Description

That block find the min value in the elements of its input vector.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no

- **name of computational function:** *minblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/MIN_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/minblk.f (Type 0)

# Nome

POWBLK_f — Power

## Block Screenshot

## Contents


- Power
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

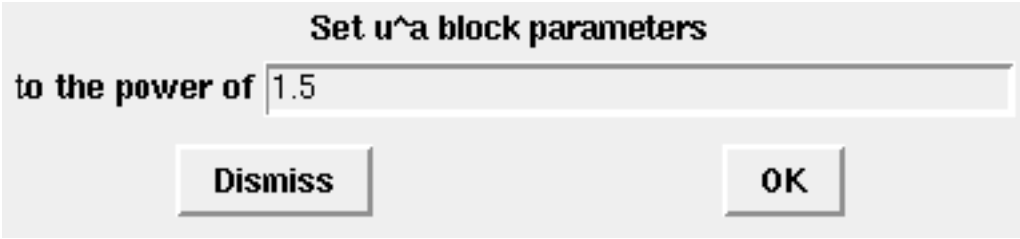

## Palette

- Math operations palette

## Description

This block realizes . The input and output port sizes are determined by the compiler according to the connected blocks port sizes.

## Dialog box




- **to the power of**

  real scalar.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *powblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/POWBLK_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/powblk.f (Type 0)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

PRODUCT — Product

## Block Screenshot

## Contents


- Product
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

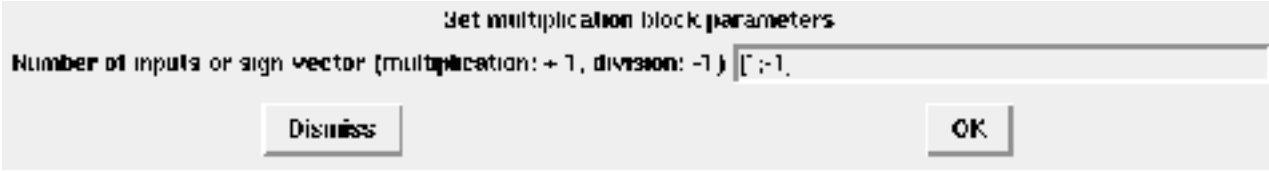

## Palette

- Math operations palette

## Description

The Product block performs multiplication or division of its inputs. This block produces outputs using either element-wise or matrix multiplication, depending on the value of the Multiplication parameter. You specify the operations with the Number of inputs parameter. Multiply(+1) and divide (-1) characters indicate the operations to be performed on the inputs.

## Dialog box




- **Number of inputs or sign vector**

  Enter the number of inputs.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**

  **- port 2 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *product*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/PRODUCT.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/product.c (Type 4)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

PROD_f — Multiplication

## Block Screenshot

## Contents


- Multiplication
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Math operations palette

## Description

The output is the element wise product of the inputs.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**

  **- port 2 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *prod*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/PROD_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/prod.c (Type 2)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

SIGNUM — Signum

## Block Screenshot

## Contents


- Signum
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

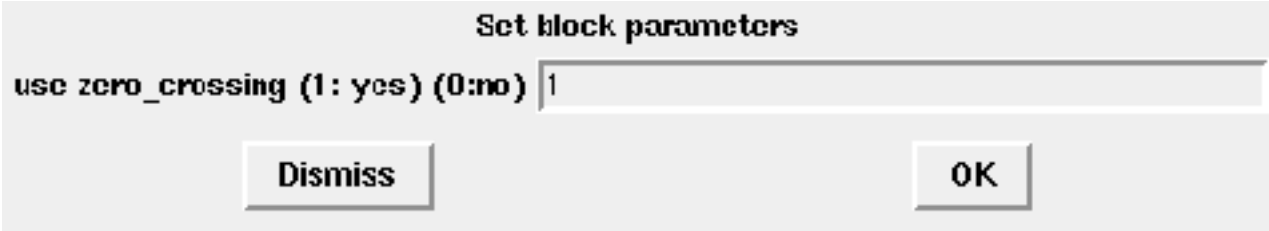
- Math operations palette

## Description

The Sign block indicates the sign of the input:

- The output is 1 when the input is greater than zero.
- The output is 0 when the input is equal to zero.
- The output is -1 when the input is less than zero.

## Dialog box




- **use zero_crossing**

  Select to enable zero crossing detection.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *signum*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/SIGNUM.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/signum.c (Type 4)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

SINBLK_f — SINBLK

## Block Screenshot

## Contents


- SINBLK
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”


## Palette

- Math operations palette

## Description

Description

$$y = \sin(u)$$

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sinblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/SINBLK_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/sinblk.f (Type 0)

# Nome

SQRT — Square root

## Block Screenshot

## Contents


- Square root
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Math operations palette

## Description

This block computes the square root of each element of the input matrix. It supported real and complex data types.

## Dialog box

- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Scicos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_sqrt*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/SQRT.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_sqrt.c
- SCI/modules/scicos_blocks/src/c/matz_sqrt.c

## See also

- POWBLK_f - Pow

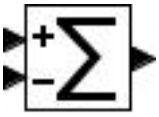
## Authors

**Fady NASSIF** - INRIA

# Nome

SUMMATION — Matrix Summation

## Block Screenshot



## Contents


- Matrix Summation
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Math operations palette

## Description

The Sum block performs addition or subtraction on its inputs. This block can add or subtract scalar, vector, or matrix inputs. It can also collapse the elements of a single input vector.

The number of inputs is given by the second parameter. This parameter can be a vector of +1 and -1 or it can be a positive value. In the first case the size of the vector indicates the number of inputs and the signs indicates whether it is a summation or a subtraction. In the second case, the block is a summation block and the value indicates the number of inputs.

On overflow, the result can take different forms:

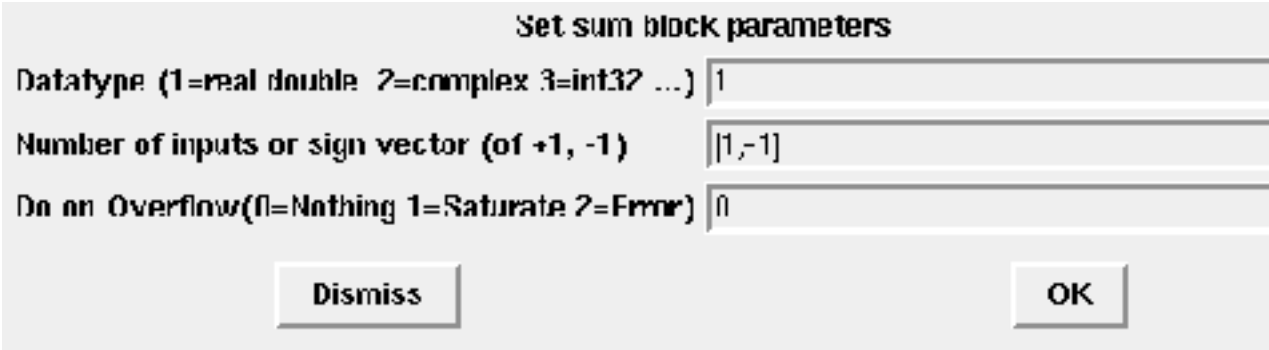
1- A normal non saturated result.

2- A saturated result.

3- An error message warning the user about the overflow.

The user can select one of these three forms by setting the "DO ON OVERFLOW" field to 0,1 or 2.

## Dialog box




- **Datatype (1=real double 2=complex 3=int32 ...)**

  It indicates the type of the input/output data. It support all datatype, number must be between 1 and 8.

  Properties : Type 'vec' of size 1.

- **Number of inputs or sign vector (of +1, -1)**

  It indicates the number of inputs and the operation see the description for more detail.

  Properties : Type 'vec' of size -1.

- **Do on Overflow(0=Nothing 1=Saturate 2=Error)**

  When this parameter is set to zero the result is similar to a normal summation of two integer matrix. When it is set to 1, on overflow the block saturate the result. When it is set to 2, on overflow an error message box appears. If the Data type is double or complex this parameter is not used.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

  **- port 2 : size [-1,-2] / type 1**

- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *summation*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/SUMMATION.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/summation.c
- SCI/modules/scicos_blocks/src/c/summation_z.c
- SCI/modules/scicos_blocks/src/c/summation_i32n.c
- SCI/modules/scicos_blocks/src/c/summation_i16n.c
- SCI/modules/scicos_blocks/src/c/summation_i8n.c
- SCI/modules/scicos_blocks/src/c/summation_ui32n.c
- SCI/modules/scicos_blocks/src/c/summation_ui16n.c
- SCI/modules/scicos_blocks/src/c/summation_ui8n.c
- SCI/modules/scicos_blocks/src/c/summation_i32s.c
- SCI/modules/scicos_blocks/src/c/summation_i16s.c
- SCI/modules/scicos_blocks/src/c/summation_i8s.c
- SCI/modules/scicos_blocks/src/c/summation_ui32s.c
- SCI/modules/scicos_blocks/src/c/summation_ui16s.c
- SCI/modules/scicos_blocks/src/c/summation_ui8s.c
- SCI/modules/scicos_blocks/src/c/summation_i32e.c
- SCI/modules/scicos_blocks/src/c/summation_i16e.c
- SCI/modules/scicos_blocks/src/c/summation_i8e.c
- SCI/modules/scicos_blocks/src/c/summation_ui32e.c
- SCI/modules/scicos_blocks/src/c/summation_ui16e.c
- SCI/modules/scicos_blocks/src/c/summation_ui8e.c

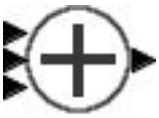
## Authors


- **Fady NASSIF** INRIA
- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA

# Nome

SUM_f — Addition

## Block Screenshot



## Contents


- Addition
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Math operations palette

## Description

The Sum block performs addition on its inputs. This block can add scalar, vector, or matrix inputs.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**

  **- port 2 : size [-1,1] / type 0**

  **- port 3 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0

- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *plusblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/SUM_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/plusblk.c (Type 2)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

TANBLK_f — TANBLK

## Block Screenshot

## Contents


- TANBLK
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”


## Palette

- Math operations palette

## Description

Description

$$y = \tan(u)$$

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *tanblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/TANBLK_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/tanblk.f (Type 0)

# Nome

TrigFun — Trigonometric function

## Block Screenshot

## Contents

- Trigonometric function
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

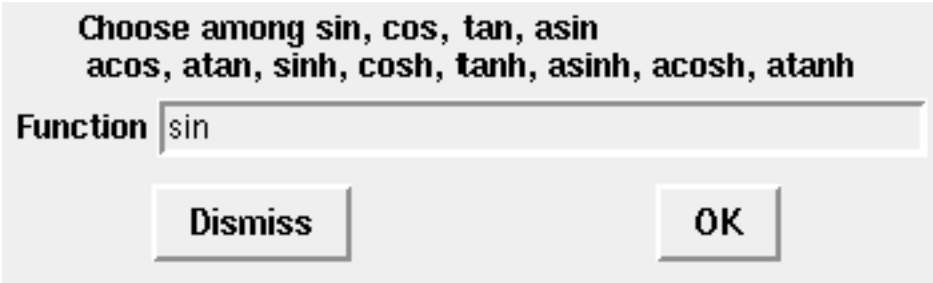
## Palette

- Math operations palette

## Description

The Trigonometric Function block performs numerous common trigonometric functions. You can select one of these functions from the Function list: sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, and tanh. The block output is the result of the operation of the function on the input or inputs.

## Dialog box




- **Function**

  The trigonometric function.

  Properties : Type 'str' of size 1

## Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sin_blk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/TrigFun.sci

## Computational function

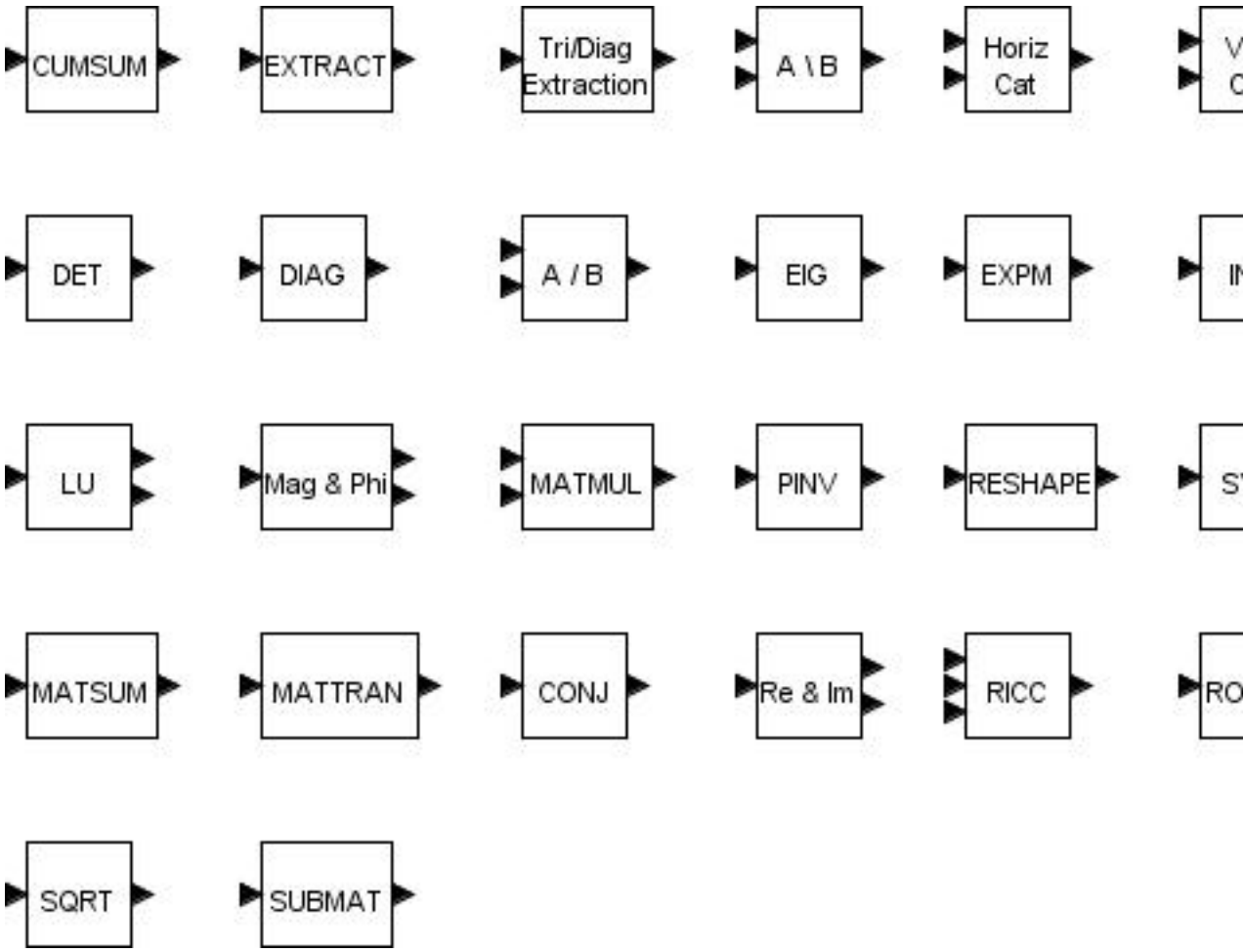
- SCI/modules/scicos_blocks/src/c/sin_blk.c (Type 4)

# 13. Matrix operation palette

# Nome

Matrix_pal — Matrix operation palette

## Block Screenshot




## Module

- xcos

## Description

Matrix palette contains all blocks that you need to do simple and complex matrix operations.

## Blocks

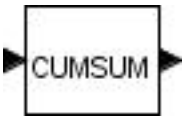
- CUMSUM - CUMSUM: Cumulative Sum
- EXTRACT - EXTRACT: Matrix Extractor
- EXTTRI - EXTTRI: Triangular or Diagonal extraction
- MATBKSL - MATBKSL:left matrix division
- MATCATH - MATCATH: Horizontal Concatenation
- MATCATV - MATCATV Vertical Concatenation

- MATDET - MATDET Matrix Determinant
- MATDIAG - MATDIAG Create Diagonal Matrix
- MATDIV - MATDIV Matrix division
- MATEIG - MATEIG Matrix Eigenvalues
- MATEXPM - MATEXPM Matrix Exponential
- MATINV - MATINV Matrix Inverse
- MATLU - MATLU LU Factorization
- MATMAGPHI - MATMAGPHI Complex to Magnitude and Angle Conversion
- MATMUL - MATMUL Matrix Multiplication
- MATPINV - MATPINV Matrix PseudoInverse
- MATRESH - MATRESH Matrix Reshape
- MATSING - MATSING SVD decomposition
- MATSUM - Matrix Sum
- MATTRAN - Matrix Transpose
- MATZCONJ - Matrix Conjugate
- MATZREIM - Complex decomposition
- RICC - RICC Equation de Riccati
- ROOTCOEF - Coefficient computation
- SQRT - SQRT Square root
- SUBMAT - SUBMAT Sub-matrix extraction

# Nome

CUMSUM — Cumulative Sum

## Block Screenshot




## Contents


- Cumulative Sum
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

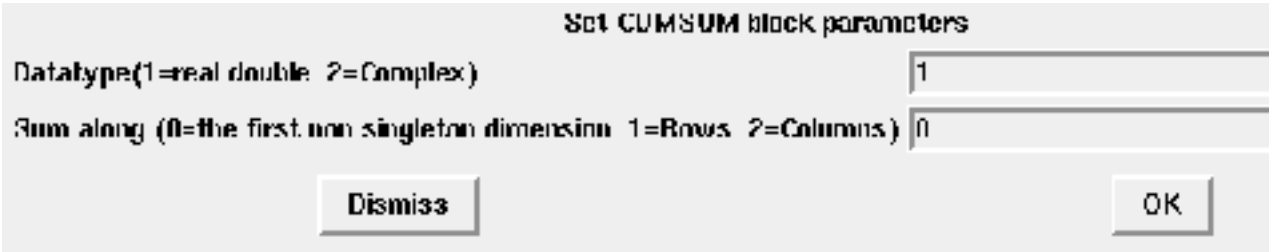
## Description

The CUMSUM block sums the element of an input m*n matrix u along either the rows, the columns or the first non singleton dimension. When the "Sum along" parameter is set to "1", the block sums across the elements of each row. The result will be displayed as a m*1 matrix.

When the "Sum along" parameter is set to "2", the block sums across the elements of each column. The result will be display as a 1*n matrix.

When the "Sum along" parameter is set to "0", the block sums across the first non singleton dimension. The result will be displayed as one element. This block is equivalent to cumsum in scilab.

## Dialog box



- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Scicos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

- **Sum along (0=the first non singleton dimension 1=Rows 2=Columns)**

  Indicate whether to sum across the rows, the columns or the first non singleton dimension.

  Properties : Type 'vec' of size 1.

## Example

```
A=[1 2 3;4 5 6;7 8 9]
If the sum is along the row the result will be
B=[12;15;18]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cumsum_m*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/CUMSUM.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/cumsum_m.c
- SCI/modules/scicos_blocks/src/c/cumsum_r.c

- SCI/modules/scicos_blocks/src/c/cumsum_c.c
- SCI/modules/scicos_blocks/src/c/cumsumz_m.c
- SCI/modules/scicos_blocks/src/c/cumsumz_r.c
- SCI/modules/scicos_blocks/src/c/cumsumz_c.c

## See also

- MATSUM - Matrix Sum (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

EXTRACT — Matrix Extractor

## Block Screenshot

## Contents


- Matrix Extractor
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

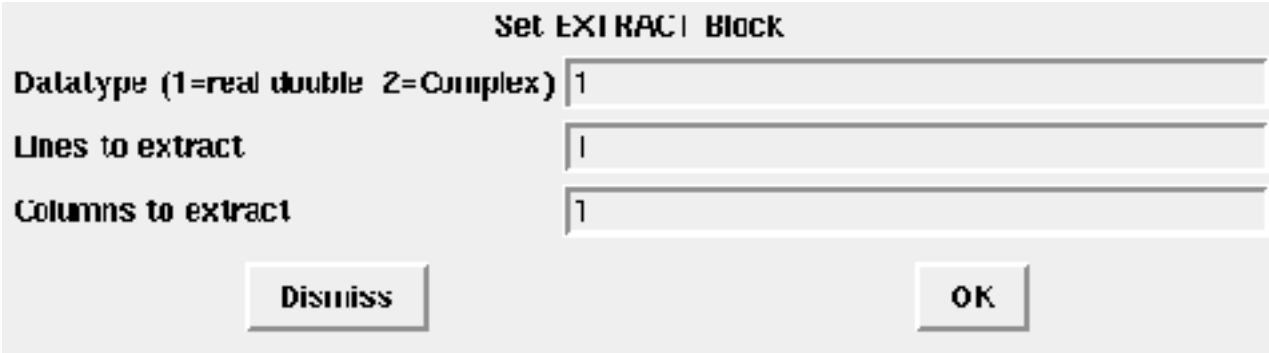

## Palette

- Matrix operation palette

## Description

The EXTRACT block extracts some elements from the matrix. The size of the output depends on the number of rows and number of columns to extract.

## Dialog box




- **Datatype (1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **Lines to extract**

  It indicates the numbers of the lines to extract.

  Properties : Type 'mat' of size [1,-1].

- **Columns to extract**

  It indicates the numbers of the columns to extract.

  Properties : Type 'mat' of size [1,-1].

## Example

```
A=[1 2 3;4 5 6;7 8 9]
If the "Lines to extract" is 1 and 2 and the "Column to extract" is 1 and 3 the
B=[1 3;4 6]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *extract*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/EXTRACT.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/extract.c

- SCI/modules/scicos_blocks/src/c/extractz.c

## See also

- EXTTRI - EXTTRI: Triangular or Diagonal extraction (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

EXTTRI — Triangular or Diagonal extraction

## Block Screenshot

## Contents


- Triangular or Diagonal extraction
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

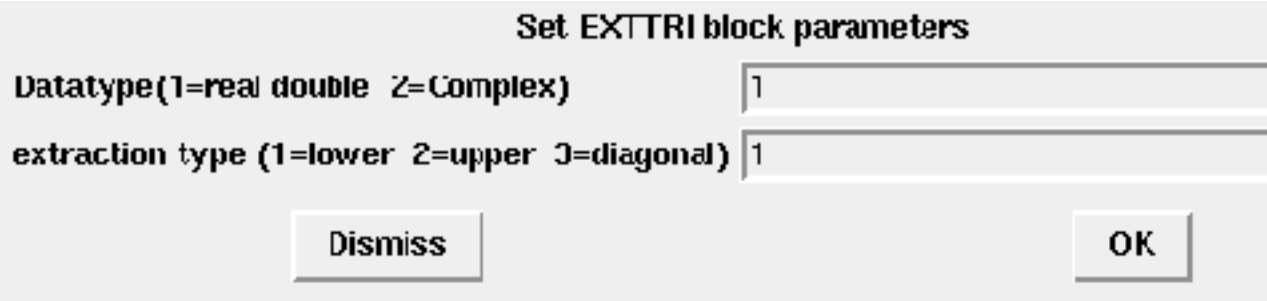
The EXTTRI block extracts some elements from the input matrix u.

When the "Extraction type" is set to "1", the block copies the elements on and above the main diagonal to an output matrix of the same size. The elements below the main diagonal are set to zero.

When the "Extraction type" is set to "2", the block copies the elements on and below the main diagonal to an output matrix of the same size. The elements above the main diagonal are set to zero.

When the "Extraction type" is set to "3", the block copies the elements on the main diagonal to an output matrix of the same size. The elements above and below the main diagonal are set to zero.

## Dialog box




- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

- **extraction type (1=lower 2=upper 3=diagonal)**

  It indicates the form of the output matrix. It can be an upper triangle, a lower triangle or a diagonal matrix.

  Properties : Type 'vec' of size 1.

## Example

```
A=[1 2 3;4 5 6;7 8 9;10 11 12]
If the extraction type is 2 then the output is
B=[1 0 0;4 5 0;7 8 9;10 11 12]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *extrilz*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/EXTTRI.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/exttril.c
- SCI/modules/scicos_blocks/src/c/exttriu.c
- SCI/modules/scicos_blocks/src/c/extdiag.c
- SCI/modules/scicos_blocks/src/c/exttrilz.c
- SCI/modules/scicos_blocks/src/c/exttriuz.c
- SCI/modules/scicos_blocks/src/c/extdiagz.c

## See also

- EXTRACT - EXTRACT: Matrix Extractor (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATBKSL — left matrix division

## Block Screenshot

## Contents


- left matrix division
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

The MATBKSL block outputs the left matrix division. It is a solution to A*x=B. The higher input is the A matrix, the lower one is the B matrix, and the output is x. If A is an M-by-N1 matrix, B must be a M-by-N2 where N1 and N2 can be different or equal. The output x is a N1-by-N2 matrix.

The equivalent of BACKSLASH is "ïn Scilab.

## Dialog box

- **Datatype (1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

## Example

```
A=[1 7 3;23 32 29]
B=[21 18;13 10]
then the result of the A*x=B equation is (A\B):
x=[-4.504 -3.922;3.643 3.132;0.000 0.000]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

  **- port 2 : size [-1,-3] / type 1**
- **regular outputs:**

  **- port 1 : size [-2,-3] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_bksl*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATBKSL.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_bksl.c
- SCI/modules/scicos_blocks/src/c/matz_bksl.c

## See also

- MATDIV - MATDIV Matrix division (xcos Block)
- MATMUL - MATMUL Matrix Multiplication (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATCATH — Horizontal Concatenation

## Block Screenshot

## Contents


- Horizontal Concatenation
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

The MATCATH Block outputs the horizontal concatenation of multiple matrices. It is also called concatenation according to the columns. The inputs U1,U2,...,Un must have the same number of rows. The output is a M-by-(N1+N2+...+Nn) matrix, where N1,N2,...,Nn are the numbers of columns of the inputs matrices, and M is the number of rows.

The equivalent of MATCATH in Scilab is y=[U1 U2 ... Un].

## Dialog box

- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

- **number of columns of each matrix**

  It indicates the number of columns of the inputs matrices.

  Properties : Type 'mat' of size [-1,1].

## Example

```
A=[1 2 3;4 5 6]
B=[7 8;9 10]
The result of the horizental concatenation is:
C=[1 2 3 7 8;4 5 6 9 10]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

  **- port 2 : size [-1,-3] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_cath*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATCATH.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_cath.c

- SCI/modules/scicos_blocks/src/c/matz_cath.c

## See also

- MATCATV - MATCATV Vertical Concatenation (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATCATV — Vertical Concatenation

## Block Screenshot

## Contents


- Vertical Concatenation
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

The MATCATV Block outputs the vertical concatenation of multiple matrices. It is also called concatenation according to the rows. The inputs U1,U2,...,Un must have the same number of columns. The output is a (M1+M2+...+Mn)-by-N matrix, where M1,M2,...,Mn are the numbers of rows of the inputs matrices, and N is the number of columns.

The equivalent of MATCATH in Scilab is y=[U1;U2;...;Un]

## Dialog box

- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

- **number of line of each matrix**

  It indicates the number of rows of the inputs matrices.

  Properties : Type 'mat' of size [-1,1].

## Example

```
A=[1 2;3 4;5 6]
B=[7 8;9 10]
The result of the horizental concatenation is:
C=[1 2;3 4;5 6;7 8;9 10]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

  **- port 2 : size [-1,-3] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_catv*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATCATV.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_catv.c

- SCI/modules/scicos_blocks/src/c/matz_catv.c

## See also

- MATCATH - MATCATH: Horizontal Concatenation (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATDET — Matrix Determinant

## Block Screenshot

## Contents


- Matrix Determinant
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

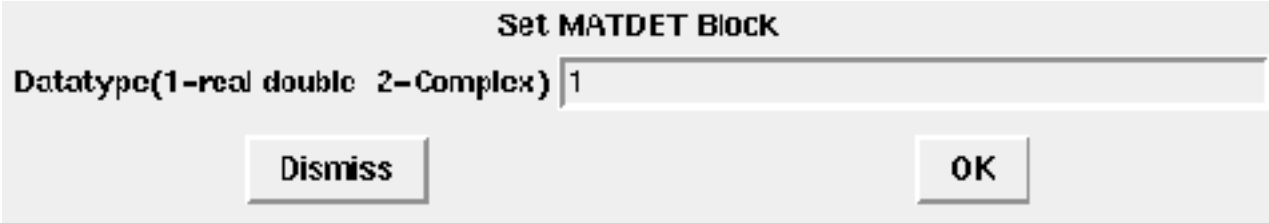
- Matrix operation palette

## Description

The MATDET outputs the determinant of a square input matrix. If the input A=[A11 A12 A13;A21 A22 A23;A31 A32 A33] then the output of the block has the form of: y=A11*(A22*A33-A23*A32)-A12*(A21*A33-A23*A31)+A13*(A21*A32-A22*A31).

The equivalent of MATDET in Scilab is "det"

## Dialog box




- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

## Example

```
U=[1 0 63;2 -2 5;9 9 4]
y=2215
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_det*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATDET.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_det.c
- SCI/modules/scicos_blocks/src/c/matz_det.c

## See also

- MATINV - MATINV Matrix Inverse (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATDIAG — Create Diagonal Matrix

## Block Screenshot

## Contents


- Create Diagonal Matrix
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

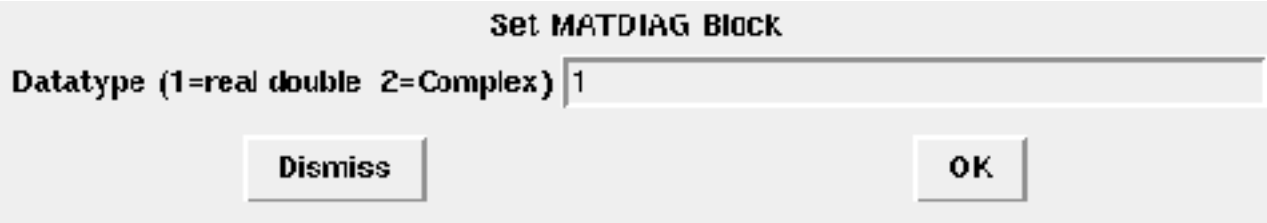

## Palette

- Matrix operation palette

## Description

The MATDIAG block create a diagonal matrix from a 1D vector. If the input is a M-by-1 vector than the output is an M-by-M matrix.

## Dialog box




- **Datatype (1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

## Example

```
if the input of the block is U=ones(5,1) then the output is:
y=[1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;0 0 0 1 0;0 0 0 0 1]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_diag*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATDIAG.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_diag.c
- SCI/modules/scicos_blocks/src/c/matz_diag.c

## See also

- EXTTRI - EXTTRI: Triangular or Diagonal extraction (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATDIV — Matrix division

## Block Screenshot

## Contents


- Matrix division
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

The MATDIV block outputs the right matrix division. It is a solution to x*B=A. The higher input is the A matrix, the lower one is the B matrix, and the output is x. If A is an M1-by-N matrix, B must be a M2-by-N where M1 and M2 can be different or equal. The output x is a M1-by-M2 matrix.

The equivalent of BACKSLASH is "/" in Scilab.

## Dialog box

- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

## Example

```
A=[1 7 3;23 32 29]
B=[21 18 34;13 10 19;11 54 36]
then the result of the x*B=A equation is (A/B):
x=[-0.475 0.712 0.156;-4.350 8.381 0.491]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-3] / type 1**

  **- port 2 : size [-2,-3] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_div*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATDIV.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_div.c
- SCI/modules/scicos_blocks/src/c/matz_div.c

## See also

- MATBKSL - MATBKSL:left matrix division
- MATMUL - MATMUL Matrix Multiplication

## Authors

**Fady NASSIF** - INRIA

# Nome

MATEIG — Matrix Eigenvalues

## Block Screenshot

## Contents


- Matrix Eigenvalues
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

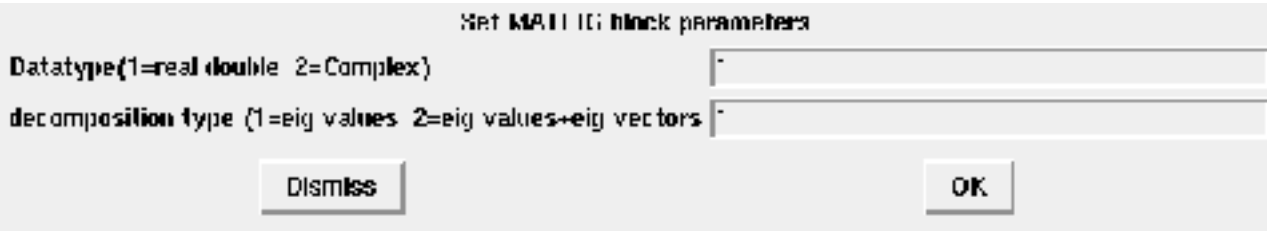
The MATEIG calculate the eigenvalues and the eigenvectors of a square input matrix U.

When the "Decomposition type" is set to 1, the block outputs the eigenvalues in a vector form, if the input is a M-by-M matrix the output is a M-by-1 vector.

When the "Decomposition type" is set to 2, the block outputs two matrices. for an M-by-M input matrix,the first output is a M-by-M diagonal matrix composed by the eigenvalues, and the second is a M-by-M matrices composed by the eigenvectors; the eigenvectors are represented by the columns of the matrix.

The equivalent of the MATEIG block in Scilab is "spec(A)"

## Dialog box



- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

- **decomposition type (1=eig values 2=eig values+eig vectors**

  To select the form of the output.

  Properties : Type 'vec' of size 1.

## Example

```
A=[1 12 32;21 35 46;70 8 19]
When the "Decomposition type" is set to one the output is y=[-35.649;14.279;76.
When the "Decomposition type" is set to two the outputs are:
y1=[-35.649 0.000 0.000;0.000 14.279 0.000;0.000 0.000 76.369]
y2=[0.557 0.080 0.349;0.330 -0.922 0.770;-0.762 0.379 0.533]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 2**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_vps*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATEIG.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_vps.c

- SCI/modules/scicos_blocks/src/c/mat_vpv.c
- SCI/modules/scicos_blocks/src/c/matz_vps.c
- SCI/modules/scicos_blocks/src/c/matz_vpv.c

## See also

- MATSING - MATSING SVD decomposition (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATEXPM — Matrix Exponential

## Block Screenshot

## Contents


- Matrix Exponential
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

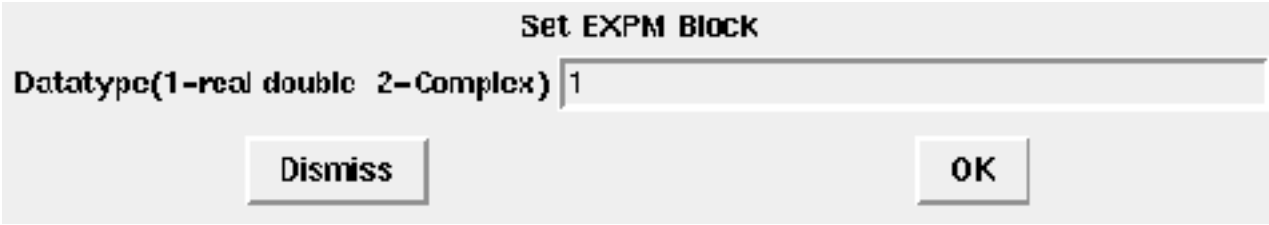

## Palette

- Matrix operation palette

## Description

The MATEXPM outputs the matrix exponential of a square matrix input by the pade's approximants. The output is a square matrix with the same size of the input. The equivalent of this block in Scilab is "expm".

## Dialog box




- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

## Example

```
u=[1 2 3;2 3 1;4 2 1]
y=[182.612 196.518 141.735;172.973 190.770 133.577;204.677 220.063 159.067]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_expm*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATEXPM.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_expm.c
- SCI/modules/scicos_blocks/src/c/matz_expm.c

## See also

- MATMUL - MATMUL Matrix Multiplication (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATINV — Matrix Inverse

## Block Screenshot

## Contents


- Matrix Inverse
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

The MATINV Block outputs the inverse of a square input matrix using the LU factorization. A warning message is printed if the input is badly scaled or nearly singular. The equivalent function of this block in Scilab is "inv".

## Dialog box

- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_inv*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATINV.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_inv.c
- SCI/modules/scicos_blocks/src/c/matz_inv.c

## See also

- MATLU - MATLU LU Factorization (xcos Block)
- MATPINV - MATPINV Matrix PseudoInverse (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATLU — LU Factorization

## Block Screenshot

## Contents


- LU Factorization
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

The MATLU Block outputs two matrices L and U, with row pivoting, from the LU factorization of a square input matrix. If A is the input matrix then E*A=L*U where E is the permutation matrix.The equivalent function of this block in Scilab is "[l,u,e]=lu(A)"

## Dialog box

- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-1] / type 1**

  **- port 2 : size [-1,-1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_lu*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATLU.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_lu.c
- SCI/modules/scicos_blocks/src/c/matz_lu.c

## See also

- MATINV - MATINV Matrix Inverse (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATMUL — Matrix Multiplication

## Block Screenshot

## Contents


- Matrix Multiplication
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

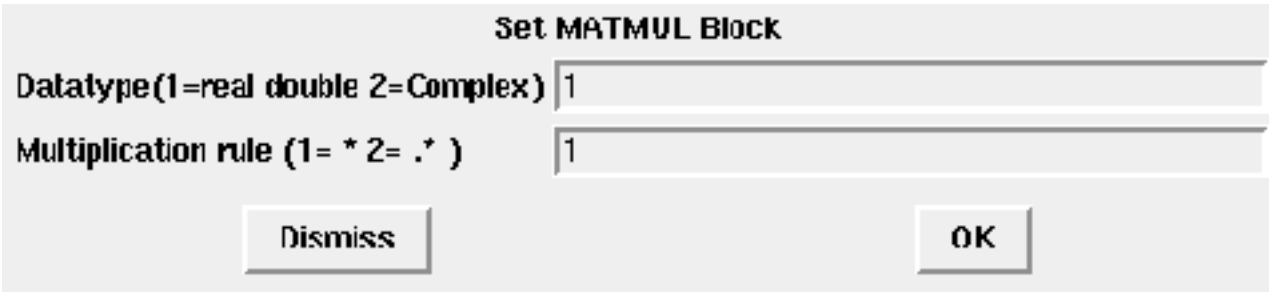

## Palette

- Matrix operation palette

## Description

The MATMUL block computes the matrix multiplication of two inputs matrices.The number of rows of the second matrix must be equal to the number of columns of the first matrix. The output is a matrix where the number of rows is equal to that of the first input matrix and the number of columns is equal to that of the second input matrix.

## Dialog box




- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **Multiplication rule (1= * 2= .* )**

  ?

  Properties : Type 'vec' of size 1.

## Example

```
A=[1 2 3;4 5 6]
B=[9 8 7 6;5 4 3 2;9 7 5 3]
y=[46 37 28 19;115 94 73 52]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

  **- port 2 : size [-2,-3] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-3] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *matmul_m*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATMUL.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/matmul_m.c
- SCI/modules/scicos_blocks/src/c/matzmul_m.c

- SCI/modules/scicos_blocks/src/c/matmul2_m.c
- SCI/modules/scicos_blocks/src/c/matzmul2_m.c

## See also

- INTMUL - INTMUL integer matrix multiplication (xcos Block)
- MATDIV - MATDIV Matrix division (Scicos Block)
- MATBKSL - MATBKSL:left matrix division (xcos Block)
- MATEXPM - MATEXPM Matrix Exponential (xcos Block)
- SUMMATION - SUMMATION Matrix Summation (xcos Block)

## Authors


- **Fady NASSIF** INRIA
- **Alan Layec** INRIA

# Nome

MATPINV — Matrix PseudoInverse

## Block Screenshot

## Contents


- Matrix PseudoInverse
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

The MATPINV Block outputs the inverse of a non square input matrix using the SVD theory.if the SVD decomposition of A is equal to:

A=USV'

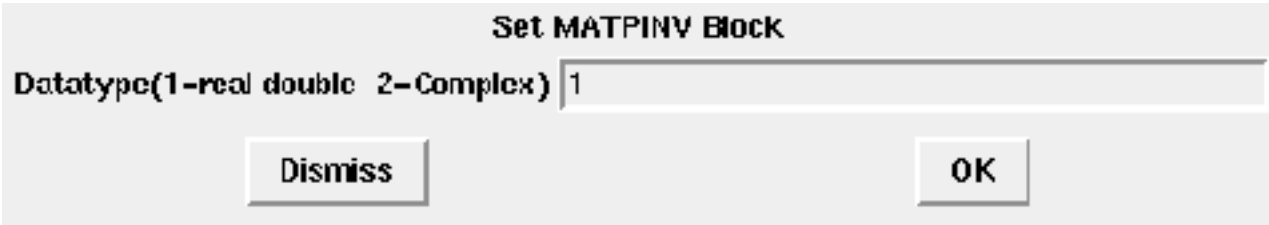
The pseudoinverse x of A is given by:

X=VS"U' where S"(i,j)=1/S(i,j) (if S(i,j) =0), U' and V are respectively the transpose of U and V'.

and we have A*X*A=A and X*A*X=X. Both A*X and X*A are Hermitian . A warning message is printed if the input is badly scaled or nearly singular.

When the input is a M-by-N matrix the output is a N-by-M matrix. The equivalent function of this block in Scilab is "pinv".

## Dialog box



- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **regular outputs:**

  **- port 1 : size [-2,-1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_pinv*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATPINV.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_pinv.c
- SCI/modules/scicos_blocks/src/c/matz_pinv.c

## See also

- MATINV - MATINV Matrix Inverse (xcos Block)
- MATSING - MATSING SVD decomposition (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATRESH — Matrix Reshape

## Block Screenshot

## Contents


- Matrix Reshape
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

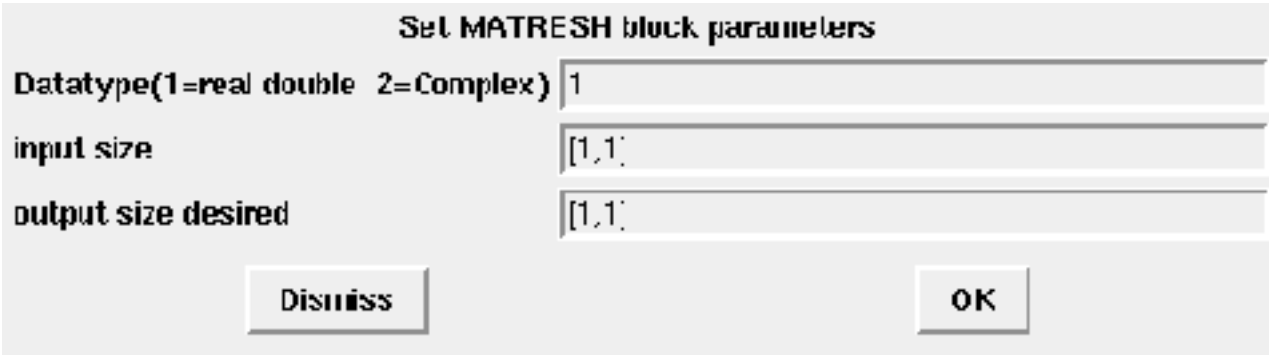

## Palette

- Matrix operation palette

## Description

The RESHAPE block changes the dimensions of a matrix or a vector to another dimensions specified by the user in the "output size desired" label. The output size must be less or equal to the input size.

## Dialog box




- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size -1.

- **input size**

  It indicates the size of the input matrix.

  Properties : Type 'vec' of size -1.

- **output size desired**

  It indicates the desired output's size.

  Properties : Type 'vec' of size -1.

## Example

```
u=[1 2 3 4;5 6 7 8]
When the output desired is [1;6] the output is:
y=[1 2 3 4 5 6]
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_reshape*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATRESH.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_reshape.c

- SCI/modules/scicos_blocks/src/c/matz_reshape.c

## See also

- EXTRACT - EXTRACT: Matrix Extractor (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATSING — SVD decomposition

## Block Screenshot

## Contents


- SVD decomposition
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

The MATSING block computes the economy sized SVD of the M-by-N input matrix A by finding U,S and V such that

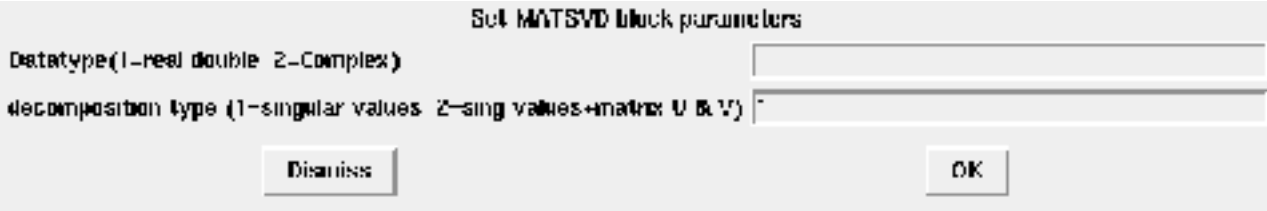
A=U*S*V'.

When the decomposition type is set to one, the output is a vector composed by the singular values.

When the decomposition type is set to two, we have three outputs: the second output is a diagonal matrix S composed by the singular values and the other two outputs are the unitary matrices U and V.

The equivalent function of this block in Scilab is "svd".

## Dialog box



- **Datatype(1=real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

- **decomposition type (1=singular values 2=sing values+matrix U & V)**

  It indicates the form of the output. When it is set to one, we have a unique vector output (singular values). When it is set to two we have three same sizes matrices(U,S,V).

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_sing*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATSING.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_sing.c
- SCI/modules/scicos_blocks/src/c/mat_svd.c
- SCI/modules/scicos_blocks/src/c/matz_sing.c
- SCI/modules/scicos_blocks/src/c/matz_svd.c

## See also

- MATEIG - MATEIG Matrix Eigenvalues (xcos Block)

- MATLU - MATLU LU Factorization (xcos Block)

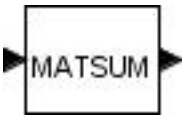
## Authors

**Fady NASSIF** - INRIA

# Nome

MATSUM — Matrix Sum

## Block Screenshot




## Contents


- Matrix Sum
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

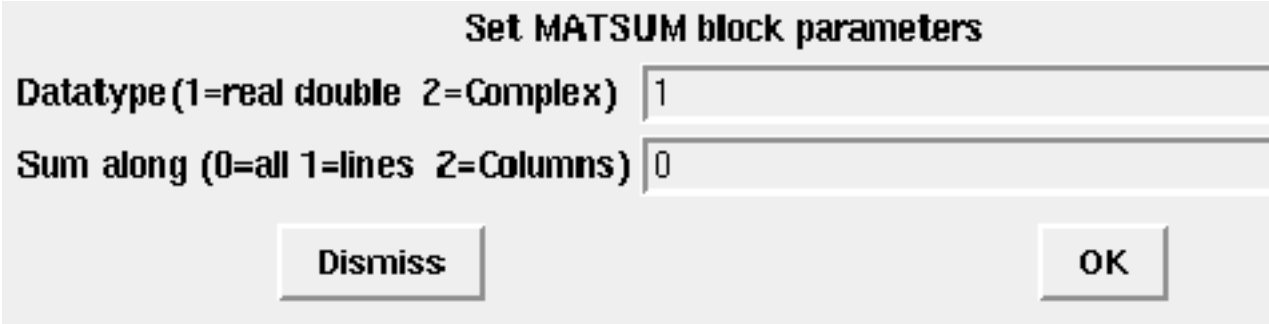

## Palette

- Matrix operation palette

## Description

The MATSUM block returns the sum of the element of an input matrix/vector. When the Sum along is set to all the block outputs the sum of all the elements of the matrix. The output is then a scalar. When the Sum along is set to lines the block is a row-wise sum. The output is a row vector. When the Sum along is set to Columns the block is a column-wise sum. The output is a column vector. The equivalent function of this block in Scilab is: "sum".

## Dialog box




- **Datatype(1=real double 2=Complex)**

It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

- **Sum along (0=all 1=lines 2=Columns)**

  Indicates the used rule to sum. For more information see the description part.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mat_sum*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATSUM.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mat_sum.c
- SCI/modules/scicos_blocks/src/c/mat_suml.c
- SCI/modules/scicos_blocks/src/c/mat_sumc.c
- SCI/modules/scicos_blocks/src/c/matz_sum.c
- SCI/modules/scicos_blocks/src/c/matz_suml.c
- SCI/modules/scicos_blocks/src/c/matz_sumc.c

## See also

- SUBMAT - SUBMAT Sub-matrix extraction (xcos Block)

- SUMMATION - SUMMATION Matrix Summation (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

MATTRAN — Matrix Transpose

## Block Screenshot

## Contents


- Matrix Transpose
- • “Palette”
  - “Description”
  - “Dialog box”
  - Example
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

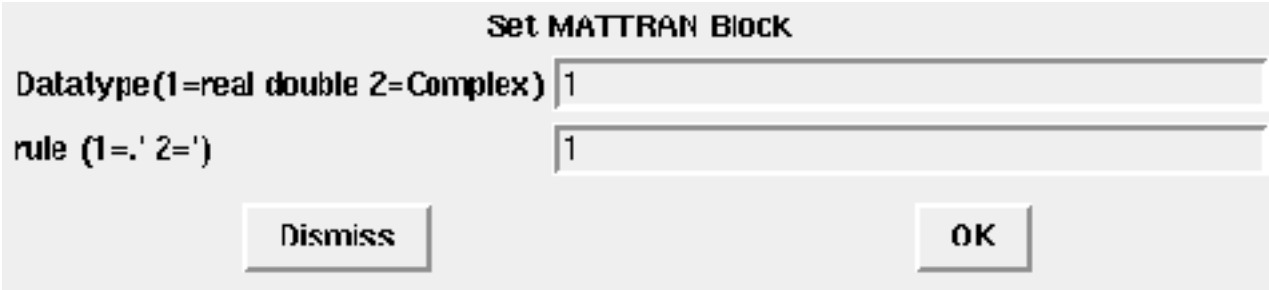

## Palette

- Matrix operation palette

## Description

This Block transposes an MxN matrix to a NxM matrix. For complex data type it uses the hermitian transpose. The equivalent of this block in Scilab is y=u'.

## Dialog box




- **Datatype(1=real double 2=Complex)**

  It indicates the data type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label xcos will print the message "Datatype is not supported".

Properties : Type 'vec' of size 1.

## Example

```
.                        Complex
--                  --               --           --
| 1+i   2+3i    3+2i |   Tranpose    | 1- i   4- i |
| 4+i   5-8i    6-2i | ----------->  | 2-3i   5+8i |
--                  --               | 3-2i   6+2i |
.                                    --           --
.                   Real
--           --                --      --
| 1   -2   3 |   Tranpose    | 1    4 |
| 4   5    6 | ----------->  |- 2   5 |
--           --               | 3    6 |
.                             --      --
```

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **regular outputs:**

  **- port 1 : size [-2,-1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mattran_m*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATTRAN.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mattran_m.c
- SCI/modules/scicos_blocks/src/c/matztran_m.c

## See also

- Matrix_pal - Matrix operation palette (xcos Palette)

## Authors

- **Fady NASSIF** INRIA
- **Alan Layec** INRIA

# Nome

MATZCONJ — Matrix Conjugate

## Block Screenshot

## Contents


- Matrix Conjugate
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

This blocks computes the conjugate of a complex input matrix.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 2**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 2**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *matz_conj*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/MATZCONJ.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/matz_conj.c

## See also

- Matrix_pal - Matrix operation palette (xcos Palette)

## Authors

**Fady NASSIF** - INRIA

# Nome

RICC — Riccati Equation

## Block Screenshot

## Contents


- Riccati Equation
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Matrix operation palette

## Description

This block computes the solution of Riccati equation using different method and for both case continuous and discrete.

The Riccati equation in continuous time is:

A*X+X*A+C-X*D*X=0

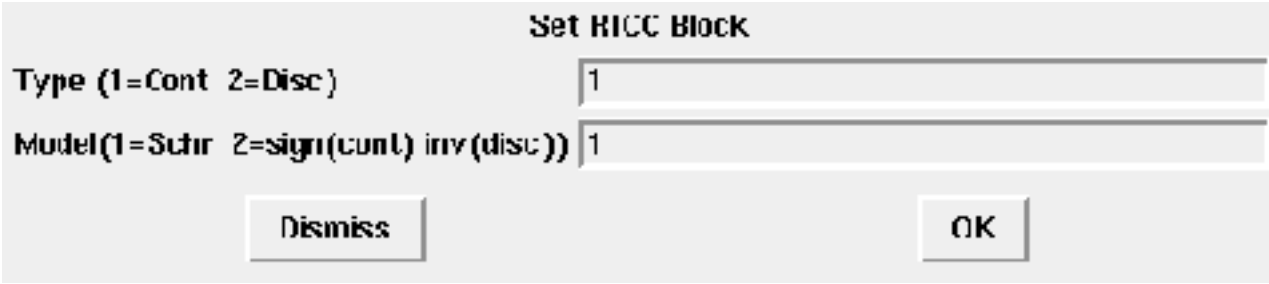
The Riccati equation in discrete time is:

A*X*(inv(In-D))*A-X+C=0

where A is an NxN matrix, it is the first input of the block, C and D are two NxN symetrics matrices and are respectively the second and third input of the RICC block. X represent the output of the block, it is also a NxN matrix.

The user can choose between two methods of computation. For the continuous time he can use even the Schur method or the matrix sign function approach method, by setting the Model parameter to 1 or 2. For the discrete time, the models are the Schur method and the inverse free spectral decomposition method.

## Dialog box




- **Type (1=Cont 2=Disc)**

  For continuous time signal set this parameter to 1. For discrete input time set it to 2.

  Properties : Type 'vec' of size 1.

- **Model(1=Schr 2=sign(cont) inv(disc))**

  To use the Shur method in computation set this parameter to 1. To use matrix sign function approach in continuous case or the inverse free spectral decomposition method in discrete case set this parameter to 2.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-1] / type 1**

  **- port 2 : size [-1,-1] / type 1**

  **- port 3 : size [-1,-1] / type 1**

- **regular outputs:**

  **- port 1 : size [-1,-1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *ricc_m*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/RICC.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/ricc_m.c

## See also

- Matrix_pal - Matrix operation palette (xcos Palette)

## Authors

**Fady NASSIF** - INRIA

# Nome

ROOTCOEF — Coefficient computation

## Block Screenshot

## Contents


- Coefficient computation
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

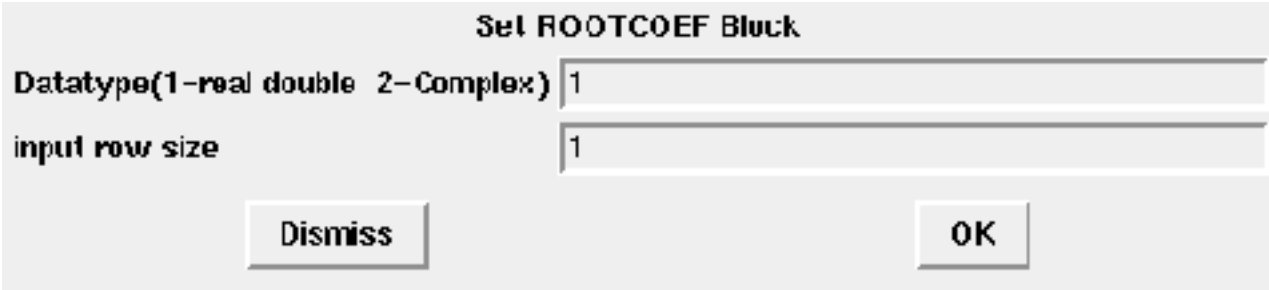

## Palette

- Matrix operation palette

## Description

This block computes the coefficients of a polynomial given its root values.

## Dialog box




- **Datatype(1=real double 2=Complex)**

  This block can only support double inputs values. These values can be real or complex.

  Properties : Type 'vec' of size 1.

- **input row size**

  The input row size.

Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-2,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *root_coef*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/ROOTCOEF.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/root_coef.c
- SCI/modules/scicos_blocks/src/c/rootz_coef.c

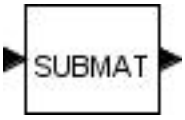
## Authors

**Fady NASSIF** - INRIA

# Nome

SUBMAT — Sub-matrix extraction

## Block Screenshot




## Contents


- Sub-matrix extraction
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

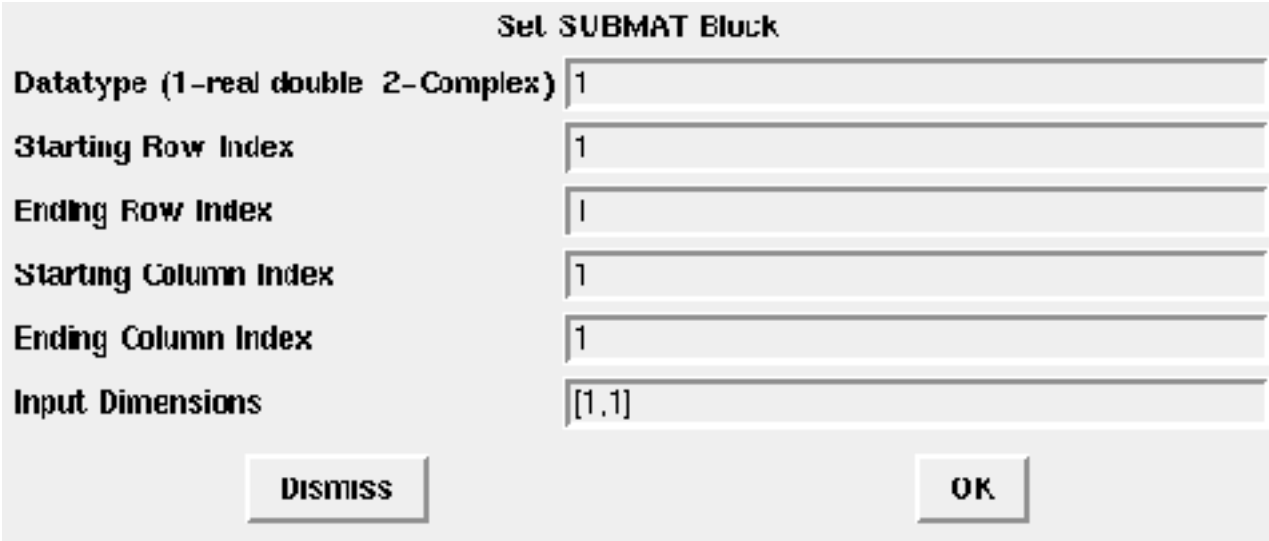

## Palette

- Matrix operation palette

## Description

This block outputs a sub matrix of the input matrix. The output matrix will be defining by using the parameters of this block.

## Dialog box



- **Datatype (1=real double 2=Complex)**

  Type of the output matrix. It can be double or complex.

  Properties : Type 'vec' of size 1.

- **Starting Row Index**

  The first row of the submatrix.

  Properties : Type 'vec' of size 1.

- **Ending Row Index**

  The last row of the Submatrix.

  Properties : Type 'vec' of size 1.

- **Starting Column Index**

  The first column of the submatrix.

  Properties : Type 'vec' of size 1.

- **Ending Column Index**

  The last row of the submatrix.

  Properties : Type 'vec' of size 1.

- **Input Dimension**

  The Matrix input dimensions.

  Properties : Type 'vec' of size 2.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no

- **name of computational function:** *submat*

## Interfacing function

- SCI/modules/scicos_blocks/macros/MatrixOp/SUBMAT.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/submat.c
- SCI/modules/scicos_blocks/src/c/submatz.c

## Authors

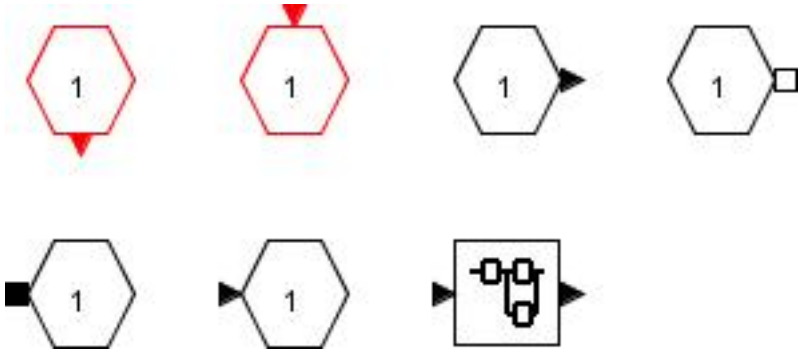
**Fady NASSIF** - INRIA

# 14. Port & Subsystem palette

# Nome

Portaction_pal — Port & Subsystem palette

## Block Screenshot



## Module

- xcos

## Description

The Port & Subsystem palette includes blocks for creating subsystems.

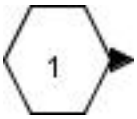
## Blocks

- CLKINV_f — Input activation port Event-Select
- CLKOUTV_f — Output activation port Event-Select
- IN_f - Input Port
- INIMPL_f — Input implicit port
- OUTIMPL_f — Output implicit port
- OUT_f - Output Port
- SUPER_f — Super block

# Nome

IN_f — Input Port

## Block Screenshot




## Contents

- Input Port
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

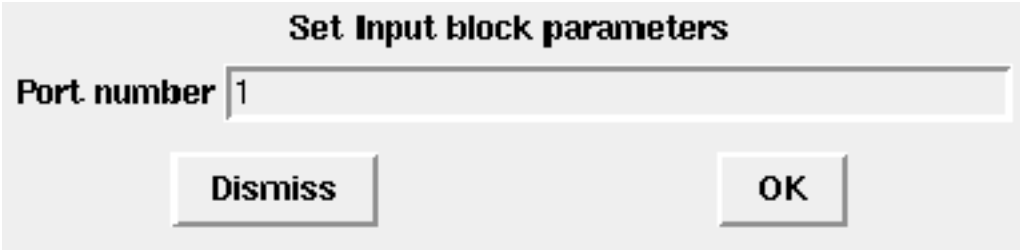
## Palette

- Port & Subsystem palette

## Description

This block must only be used inside xcos Super Blocks to represent a regular input port. The input size is determined by the context. In a Super Block, regular input ports must be numbered from 1 to the number of regular input ports.

## Dialog box




- **Port number**

  an integer defining the port number.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no

- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [-1,-2] / type -1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *input*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/IN_f.sci

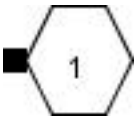
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

OUTIMPL_f — Output implicit port

## Block Screenshot




## Contents

- Output implicit port
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

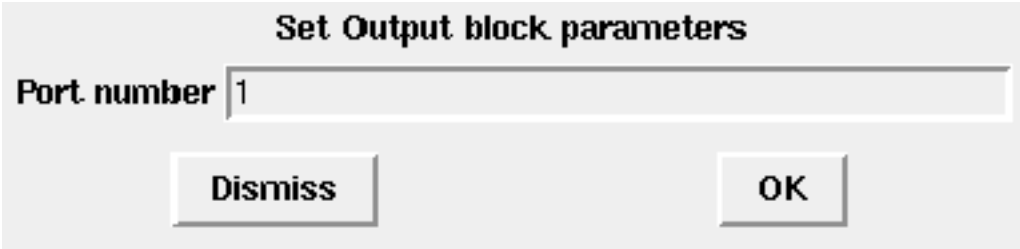
## Palette

- Port & Subsystem palette

## Description

Outport blocks are the links from a system to a destination outside the system.

## Dialog box




- **Port number**

  Specify the port number of the Outport block.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no

- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *outimpl*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/OUTIMPL_f.sci

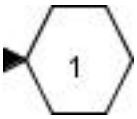
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

OUT_f — Output Port

## Block Screenshot




## Contents

- Output Port
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

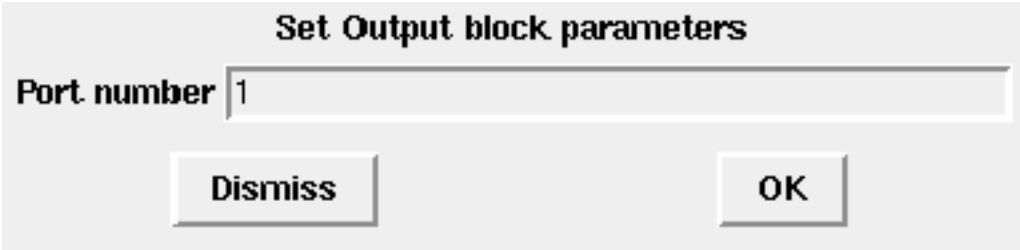
## Palette

- Port & Subsystem palette

## Description

This block must only be used inside Xcos Super Blocks to represent a regular output port. In a Super Block, regular output ports must be numbered from 1 to the number of regular output ports. Size of the output is determined by the compiler according to the connected blocks port sizes.

## Dialog box




- **Port number**

  an integer defining the port number.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type -1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *output*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/OUT_f.sci

## Authors

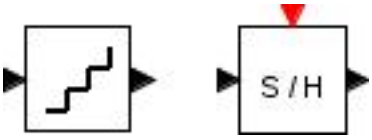
**Ramine Nikoukhah** - INRIA

# 15. Signal processing palette

# Nome

Signalprocessing_pal — Signal processing palette

## Block Screenshot




## Module

- xcos

## Description

The signal processing palette contains blocks designed specifically for signal processing applications.

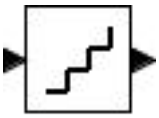
## Blocks

- MCLOCK_f
- MFCLCK_f
- QUANT_f — Quantization
- SAMPHOLD_m — Sample and hold

# Nome

QUANT_f — Quantization

## Block Screenshot



## Contents


- Quantization
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

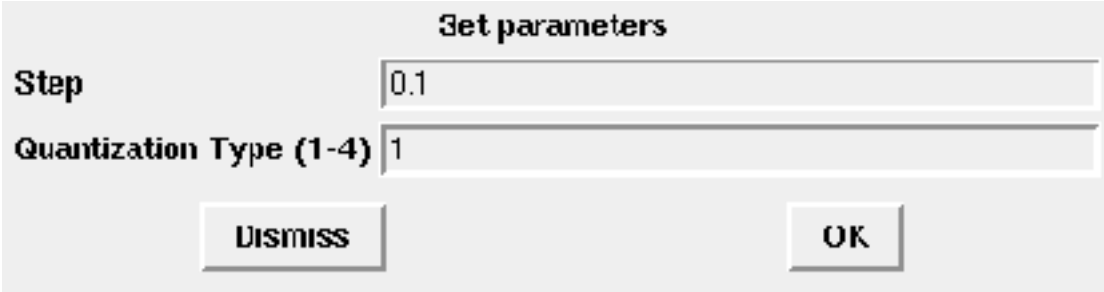

## Palette

- Signal processing palette

## Description

This block outputs the quantization of the input according to a choice of methods:

## Dialog box




- **Step**

  scalar, Quantization step

  Properties : Type 'vec' of size 1.

- **Quantization Type**

  scalar with possible values 1,2,3 or 4 Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *qzrnd*

## Interfacing function

- SCI/modules/scicos_blocks/macros/NonLinear/QUANT_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/qzrnd.f
- SCI/modules/scicos_blocks/src/fortran/qztrn.f
- SCI/modules/scicos_blocks/src/fortran/qzflr.f
- SCI/modules/scicos_blocks/src/fortran/qzcel.f

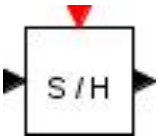
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

SAMPHOLD_m — Sample and hold

## Block Screenshot




## Contents


- Sample and hold
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

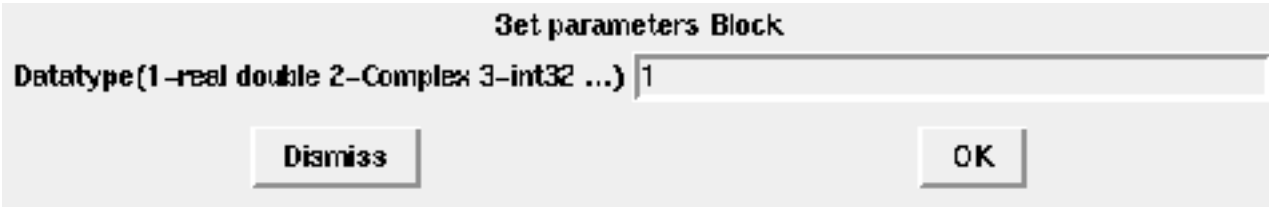

## Palette

- Signal Processing palette

## Description

Each time an input event is received block copy its input on the output and hold it until input event. For periodic Sample and hold, event input must be generated by a **Clock**.

## Dialog box




- **Datatype(1=real double 2=Complex 3=int32 ...)**

  Output datatype. This block can support all data types.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes

- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *samphold4_m*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Linear/SAMPHOLD_m.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/samphold4_m.c (Type 4)

## Authors

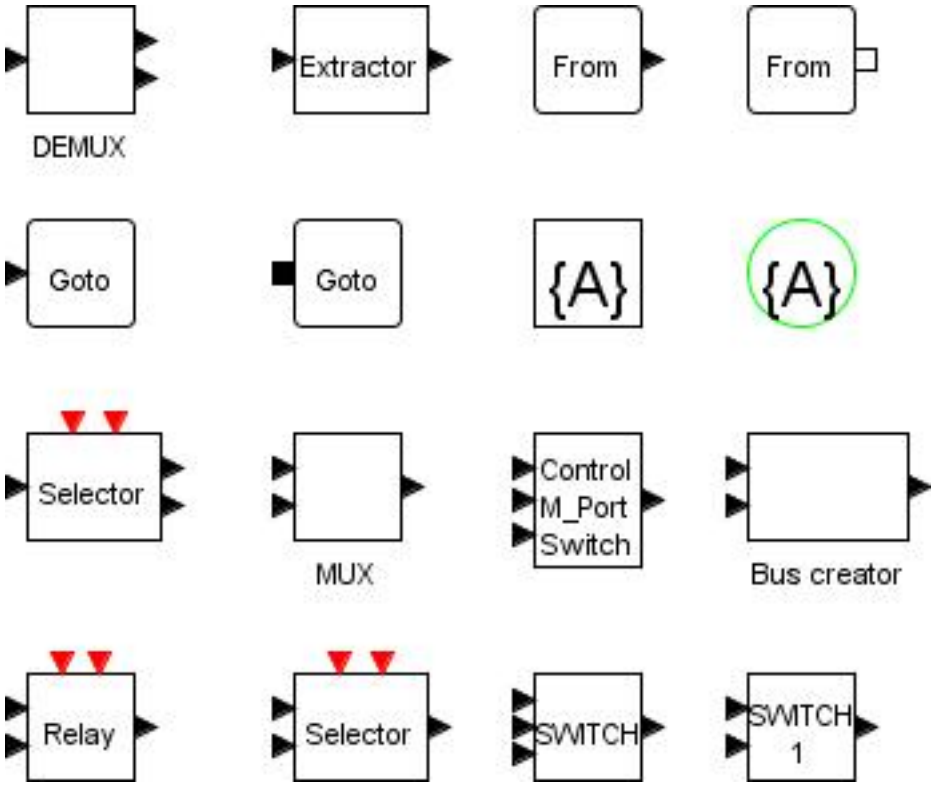
**Fady NASSIF** - INRIA

# 16. Signal routing palette

# Nome

Signalrouting_pal — Signal routing palette

## Block Screenshot




## Module

- xcos

## Description

The Signal routing palette includes blocks that transport signals from one point in a block diagram to another.

## Blocks

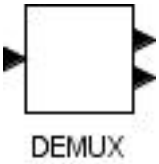
- DEMUX — Demultiplexer
- EXTRACTOR — Extractor
- FROM — FROM Receives data from a corresponding GOTO
- FROMMO — Receives data from a corresponding GOTOMO
- GOTO — GOTO Pass block input to From block
- GOTOMO — Pass block input to FROMMO block
- GotoTagVisibility — Define Scope of GOTO tag visibility
- GotoTagVisibilityMO — Define Scope of GOTOMO tag visibility
- ISELECT_m — Iselect

- MUX — Multiplexer
- M_SWITCH — Multi-port switch
- NRMSOM_f — Merge data
- RELAY_f — Relay
- SELECT_m — Select
- SWITCH2_m — Switch2
- SWITCH_f — Switch

# Nome

DEMUX — Demultiplexer

## Block Screenshot




## Contents


- Demultiplexer
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Signal routing palette

## Description

Given a vector valued input this block splits inputs over vector valued outputs. So , where are numbered from top to bottom. Input and Output port sizes are determined by the context.

## Dialog box

- **number of output ports or vector of sizes**

  positive integer less than or equal to 8 .

  Properties : Type 'vec' of size -1

## Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [0,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**

  **- port 2 : size [-2,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *multiplex*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/DEMUX.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/multiplex.c (Type 4)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

EXTRACTOR — Extractor

## Block Screenshot

## Contents


- Extractor
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

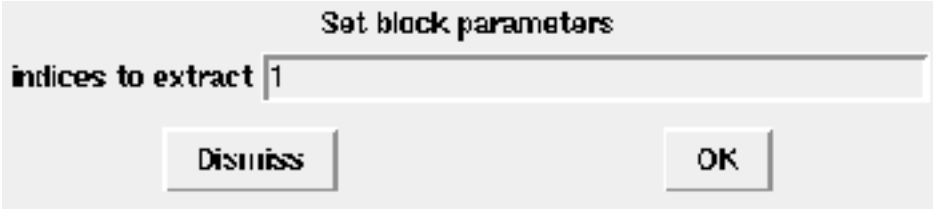

## Palette

- Signal routing palette

## Description

Extracts or select a regular input from a multiple regular input.

## Dialog box




- **indices to extract**

  a regular input to be extracted from the multiple regular inputs.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no

- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *extractor*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/EXTRACTOR.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/extractor.c (Type 4)

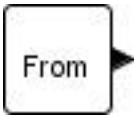
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

FROM — FROM Receives data from a corresponding GOTO

## Block Screenshot




## Contents

- FROM Receives data from a corresponding GOTO
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “See also”
  - “Authors”

## Palette

- Signal routing palette

## Description

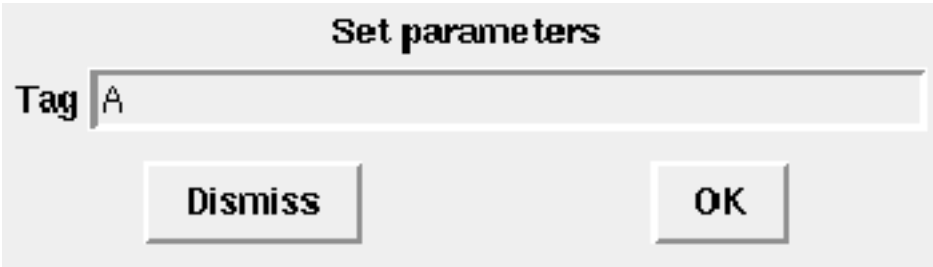
The main role of the GOTO/FROM blocks is to transport signals from a block to another block without connecting them physically. The FROM block transports its received data (from the corresponding GOTO) to its output. Multiples FROM blocks can receive data from one GOTO, although a GOTO can send data to multiple FROM.

The GOTO and FROM blocks are connected by the tag parameter.

For information on the visibility and limitation of these blocks please refer to the GOTO documentation.

This block can support all the data types.

## Dialog box



- **Tag**

  The tag of the GOTO block passing the signal to this FROM block.

  Properties : Type 'str' of size -1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [-1,-2] / type -1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *from*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/FROM.sci

## See also

- GOTO - GOTO Pass block input to From block
- GotoTagVisibility - Define Scope of GOTO tag visibility

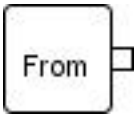
## Authors

**Fady NASSIF** - INRIA

# Nome

FROMMO — Receives data from a corresponding GOTOMO

## Block Screenshot




## Contents

- Receives data from a corresponding GOTOMO
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “See also”
  - “Authors”

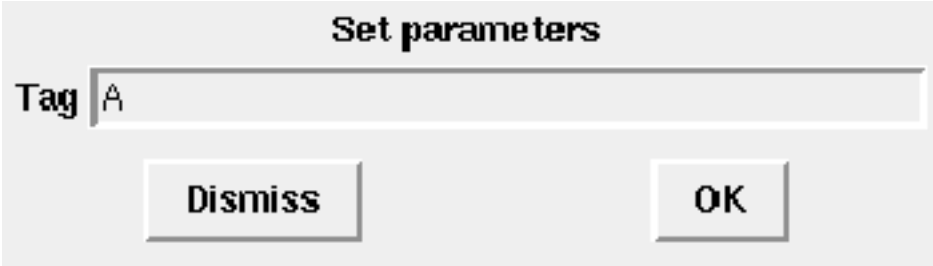
## Palette

- Signal Routing palette

## Description

This block is used to connect Modelica blocks. For more information on how it works please refer to the documentation of the FROM block by clicking on the link in the "See also" field.

## Dialog box




- **Tag**

  The tag of the GOTOMO block passing the signal to this FROMMO block.

  Properties : Type 'str' of size -1.

## Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [-1,-2] / type -1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *frommo*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/FROMMO.sci

## See also

- FROM - FROM Receives data from a corresponding GOTO (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

GOTO — GOTO Pass block input to From block

## Block Screenshot

## Contents


- GOTO Pass block input to From block
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “See also”
  - “Authors”


## Palette

- Signal routing palette

## Description

The main role of the GOTO/FROM blocks is to transport signals from a block to another block without connecting them physically. The GOTO block transports its input data to its corresponding FROM block. A simple GOTO block can send data to multiple FROM, although a FROM can receive data from only one GOTO.

The GOTO and FROM blocks are connected by the tag parameter.

The "Tag Visibility" parameter indicates if the location of the FROM block is limited:

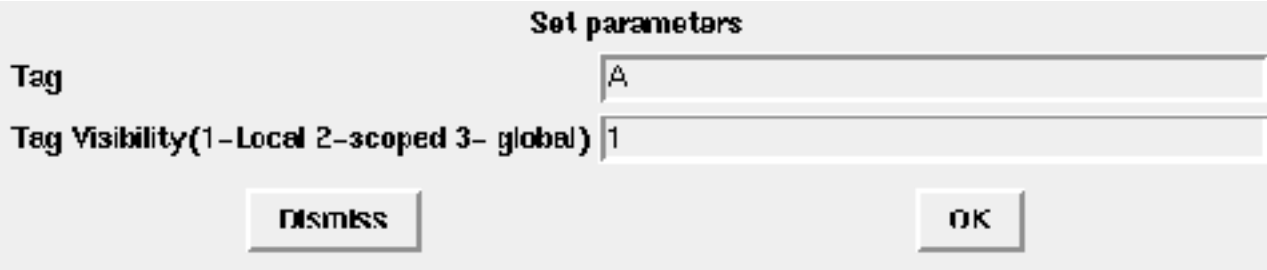
- Local: means that the corresponding FROM of that GOTO must be in the same subsystem.

- Scoped: means that the corresponding FROM of that GOTO must be in the same subsystem or in any subsystem below the GotoTagVisibility block in the model hierarchy.

- Global: means that the corresponding FROM of that GOTO can be anywhere in the model.

This block can support all the data types.

## Dialog box




- **Tag**

  This parameter identifies the Goto block whose scope is defined in this block.

  Properties : Type 'str' of size -1.

- **Tag Visibility(1=Local 2=scoped 3= global)**

  This parameter identifies the visibility of the block. It can be local(1), scoped(2) or global(3).

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type -1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *goto*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/GOTO.sci

## See also

- FROM - FROM Receives data from a corresponding GOTO
- GotoTagVisibility - Define Scope of GOTO tag visibility

## Authors

**Fady NASSIF** - INRIA

# Nome

GOTOMO — Pass block input to FROMMO block

## Block Screenshot

## Contents


- Pass block input to FROMMO block
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “See also”
  - “Authors”

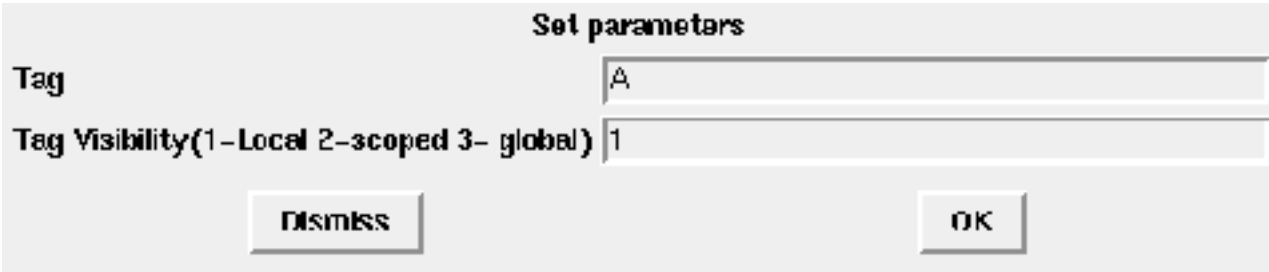

## Palette

- Signal Routing palette

## Description

This block is used to connect Modelica blocks. For more information on how it works please refer to the documentation of the GOTO block by clicking on the link in the "See also" field.

## Dialog box




- **Tag**

  This parameter identifies the Goto block whose scope is defined in this block.

  Properties : Type 'str' of size -1.

- **Tag Visibility(1=Local 2=scoped 3= global)**

  This parameter identifies the visibility of the block. It can be local(1), scoped(2) or global(3).

Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *gotomo*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/GOTOMO.sci

## See also

- GOTO - GOTO Pass block input to From block (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

GotoTagVisibility — Define Scope of GOTO tag visibility

## Block Screenshot

## Contents

- Define Scope of GOTO tag visibility
- 
  - “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “See also”
  - “Authors”

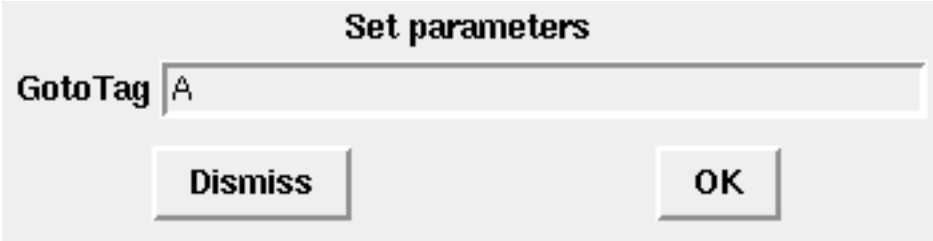

## Palette

- Signal routing palette

## Description

This block defines the accessibility of the GOTO block when it is configure as "scoped". The FROM block corresponding to that GOTO must be in the same subsystem of the GotoTagVisibility or in subsystems below it in the model hierarchy.

## Dialog box




- **GotoTag**

  The Goto block tag whose visibility is defined by the location of this block.

  Properties : Type 'str' of size -1.

## Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *gototagvisibility*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/GotoTagVisibility.sci

## See also

- GOTO - GOTO Pass block input to From block
- FROM - FROM Receives data from a corresponding GOTO

## Authors

**Fady NASSIF** - INRIA

# Nome

GotoTagVisibilityMO — Define Scope of GOTOMO tag visibility

## Block Screenshot

## Contents

- Define Scope of GOTOMO tag visibility
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “See also”
  - “Authors”

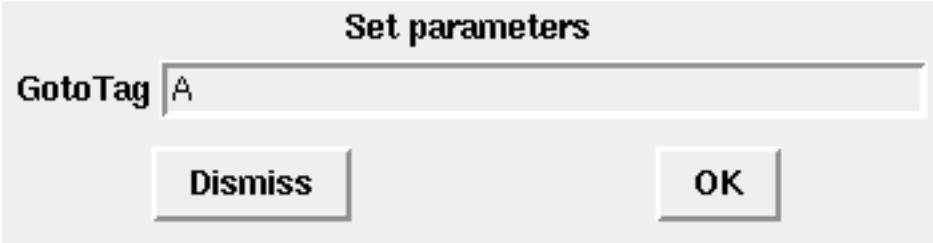
## Palette

- Signal Routing palette

## Description

This block is used in case of Modelica connection. For more information on how it works please refer to the GotoTagVisibility block by clicking on the link in the "See also" field.

## Dialog box




- **GotoTag**

  The Goto block tag whose visibility is defined by the location of this block.

  Properties : Type 'str' of size -1.

## Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *gototagvisibilitymo*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/GotoTagVisibilityMO.sci

## See also

- GotoTagVisibility - Define Scope of GOTO tag visibility (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

ISELECT_m — Iselect

## Block Screenshot

## Contents

- Iselect
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

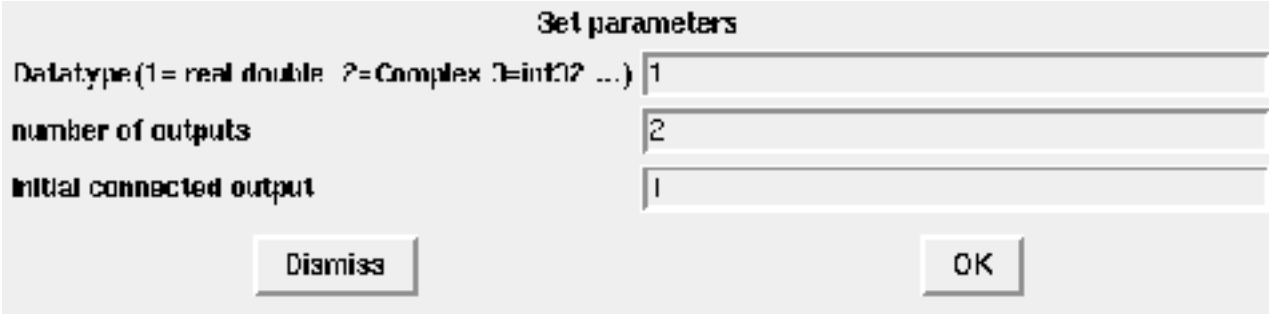
## Palette

- Signal routing palette

## Description

Select signals from an incoming events. This block has one regular input port.

## Dialog box




- **Datatype(1= real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Xcos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1

- **number of outputs**

  a scalar. Number of regular and event outputs.

Properties : Type 'vec' of size 1

- **initial connected output**

  an integer. It must be between 1 and the number of inputs.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**

  **- port 2 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 2
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *selector_m*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/ISELECT_m.sci

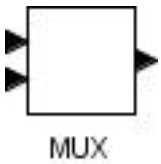
## Computational function

- SCI/modules/scicos_blocks/src/c/selector_m.c (Type 4)

# Nome

MUX — Multiplexer

## Block Screenshot




## Contents


- Multiplexer
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Signal routing palette

## Description

Given vector valued inputs this block merges inputs in an single output vector. So , where are numbered from top to bottom. Input and Output port sizes are determined by the context.

## Dialog box

- **number of input ports or vector of sizes**

  integer greater than or equal to 1 and less than 8.

  Properties : Type 'vec' of size -1.

## Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**

  **- port 2 : size [-2,1] / type 1**
- **regular outputs:**

  **- port 1 : size [0,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *multiplex*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/MUX.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/multiplex.c (Type 4)

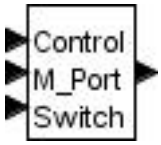
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

M_SWITCH — Multi-port switch

## Block Screenshot




## Contents

- Multi-port switch
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

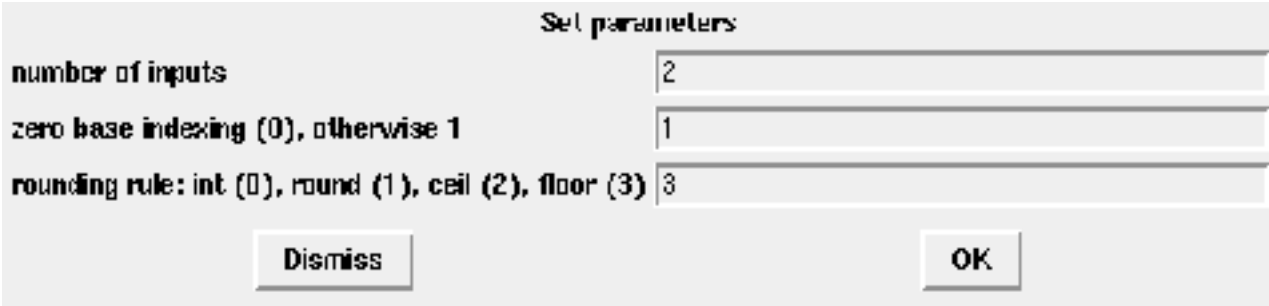
## Palette

- Signal routing palette

## Description

The Multi-Port Switch block chooses between a number of inputs. The first (top) input is called the control input, while the rest of the inputs are called data inputs. The value of the control input determines which data input is passed through to the output port.

## Dialog box




- **number of inputs**

  Specify the number of data inputs to the block.

  Properties : Type 'vec' of size 1

- **zero base indexing**

If selected, the block uses zero-based indexing. Otherwise, the block uses one-based indexing.

Properties : Type 'vec' of size 1

- **rounding rule: int**

  Select the rounding mode for the output.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [-1,1] / type 0**

  **- port 3 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *mswitch*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/M_SWITCH.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/mswitch.c (Type 4)

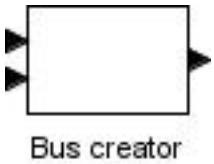
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

NRMSOM_f — Merge data

## Block Screenshot




## Contents


- Merge data
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

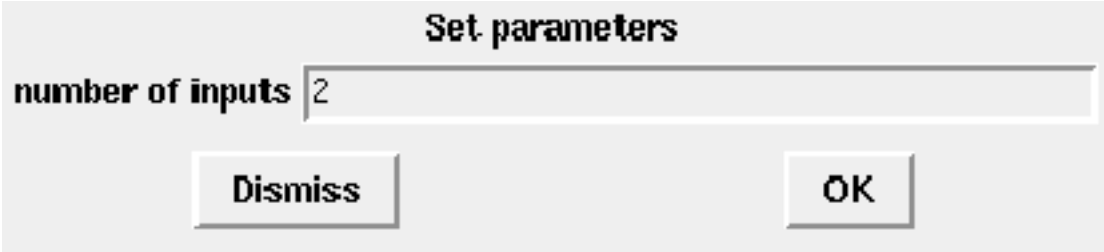

## Palette

- Signal routing palette

## Description

The Merge block combines its inputs into a single output line whose value at any time is equal to the most recently computed output of its driving blocks. You can specify any number of inputs by setting the block's Number of inputs parameter.

## Dialog box




- **number of inputs**

  The number of input ports to be merged.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**

  **- port 2 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *junk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/NRMSOM_f.sci

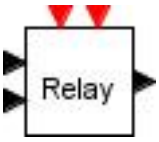
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

RELAY_f — Relay

## Block Screenshot




## Contents


- Relay
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

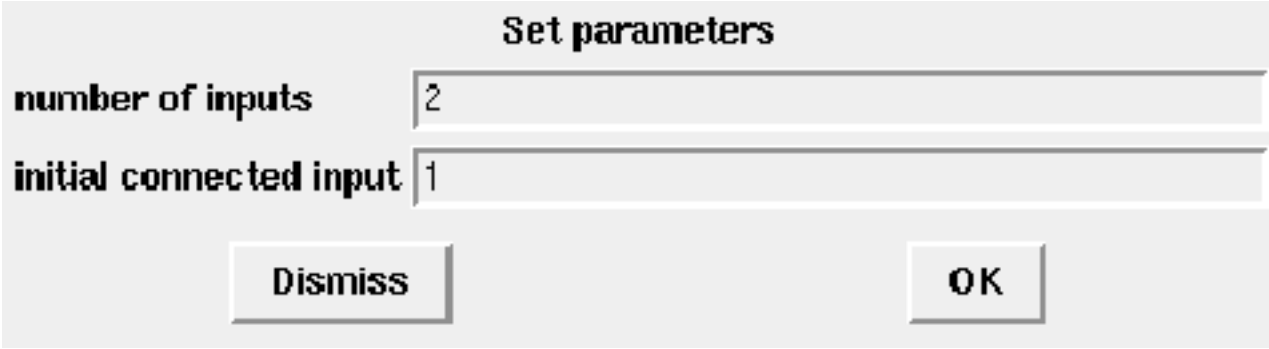

## Palette

- Signal routing palette

## Description

This block routes one of the regular inputs to the unique regular output. the choice of which input is to be routed is done, initially by the "initial connected input" parameter. Then, every time an input event arrives on the i-th input event port, the i-th regular input port is routed to the regular output.

## Dialog box




- **number of inputs**

  a scalar. Number of regular and event inputs.

  Properties : Type 'vec' of size 1

- **initial connected input**

  an integer. It must be between 1 and the number of inputs.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** yes
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**

  **- port 2 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 2
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *relay*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/RELAY_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/relay.c (Type 2)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

SELECT_m — Select

## Block Screenshot

## Contents


- Select
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

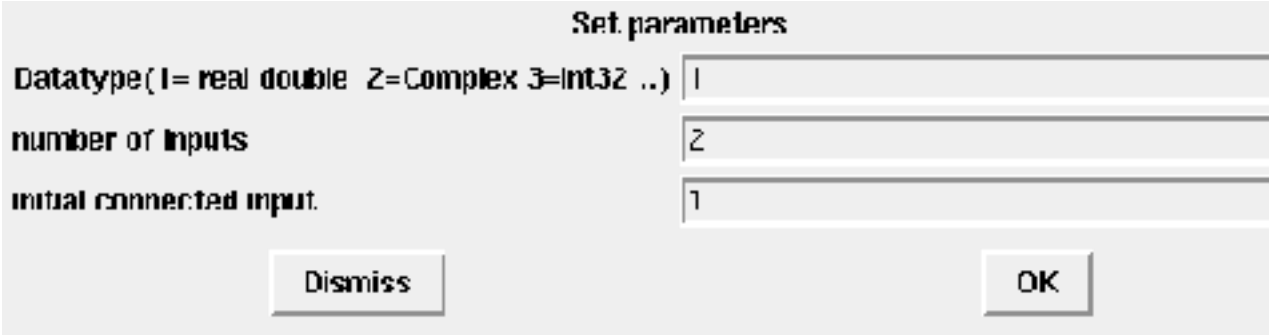

## Palette

- Signal routing palette

## Description

A Selector block accepts either vector or matrix signals as input. Set the Input Type parameter to the type of signal (vector or matrix) that the block should accept in your model. The parameter dialog box and the block's appearance change to reflect the type of input that you select. The way the block determines the elements to select differs slightly, depending on the type of input.

## Dialog box




- **Datatype(1= real double 2=Complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Xcos will print the message "Datatype is not supported". Properties : Type 'vec' of size 1

- **number of inputs**

  a scalar. Number of regular and event inputs.

  Properties : Type 'vec' of size 1

- **initial connected input**

  an integer. It must be between 1 and the number of inputs.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

  **- port 2 : size [-1,-2] / type 1**

- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**

- **number/sizes of activation inputs:** 2
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *selector_m*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/SELECT_m.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/selector_m.c (Type 4)

## Authors

- **Fady NASSIF** INRIA
- **Ramine Nikoukhah** INRIA

# Nome

SWITCH2_m — Switch2

## Block Screenshot

## Contents


- Switch2
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

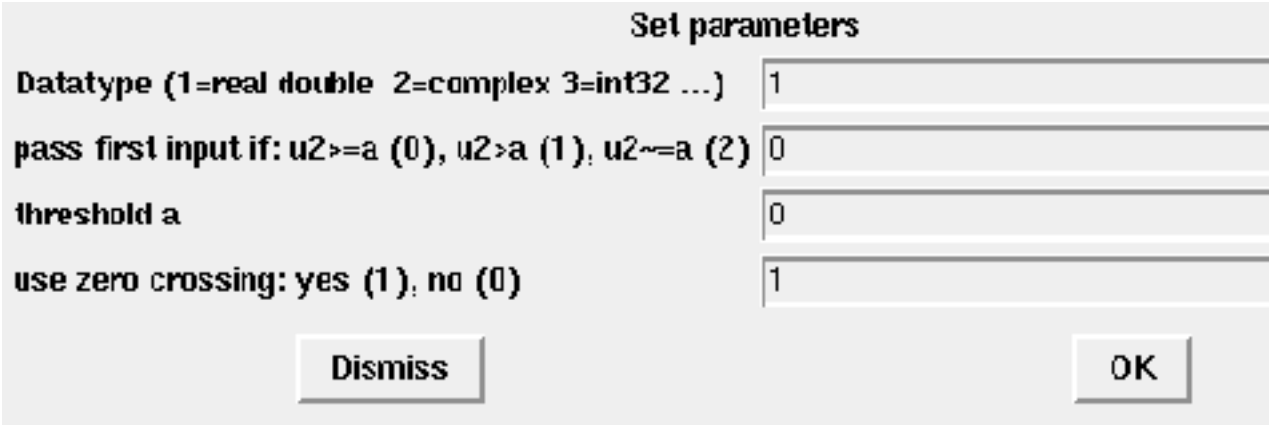

## Palette

- Signal routing palette

## Description

The Switch block passes through the first (top) input or the third (bottom) input based on the value of the second (middle) input. The first and third inputs are called data inputs. The second input is called the control input. You select the conditions under which the first input is passed with the Criteria for passing first input parameter. You can make the block check whether the control input is greater than or equal to the threshold value, purely greater than the threshold value, or nonzero. If the control input meets the condition set in the Criteria for passing first input parameter, then the first input is passed. Otherwise, the third input is passed.

## Dialog box



- **Datatype(1= real double 2=Complex)**

  a scalar. Give the datatype of the inputs/output.

  Properties : Type 'vec' of size 1

- **pass first input if: u2 >= a**

  Select the conditions under which the first input is passed. You can make the block check whether the control input is greater than or equal to the threshold value, purely greater than the threshold value, or nonzero. If the control input meets the condition set in this parameter, then the first input is passed. Otherwise, the third input is passed.

  Properties : Type 'vec' of size 1.

- **threshold a**

  Assign the switch threshold that determines which input is passed to the output.

  Properties : Type 'vec' of size 1.

- **use zero crossing: yes**

  Select to enable zero crossing detection.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

  **- port 2 : size [1,1] / type 0**

  **- port 3 : size [-1,-2] / type 1**

- **regular outputs:**

  **- port 1 : size [-1,-2] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *switch2_m*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/SWITCH2_m.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/switch2_m.c (Type 4)

## Authors


- **Fady NASSIF** INRIA
- **Ramine Nikoukhah** INRIA

# Nome

SWITCH_f — Switch

## Block Screenshot

## Contents


- Switch
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

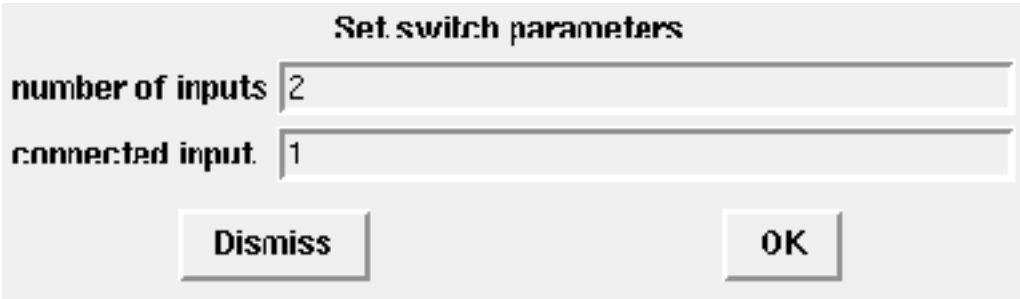

## Palette

- Signal routing palette

## Description

This is a Manual Switch block. It selects one of its inputs to pass through to the output. The selected input is propagated to the output, while the unselected inputs are discarded.

## Dialog box




- **number of inputs**

  Specify the number of data inputs to the block.

  Properties : Type 'vec' of size 1

- **connected input**

  an integer. It must be between 1 and the number of inputs.

Properties : Type 'vec' of size 1

## Default properties

- **always active:** yes
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**

  **- port 2 : size [-1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *switchn*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Branching/SWITCH_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/switchn.c (Type 2)

## Authors

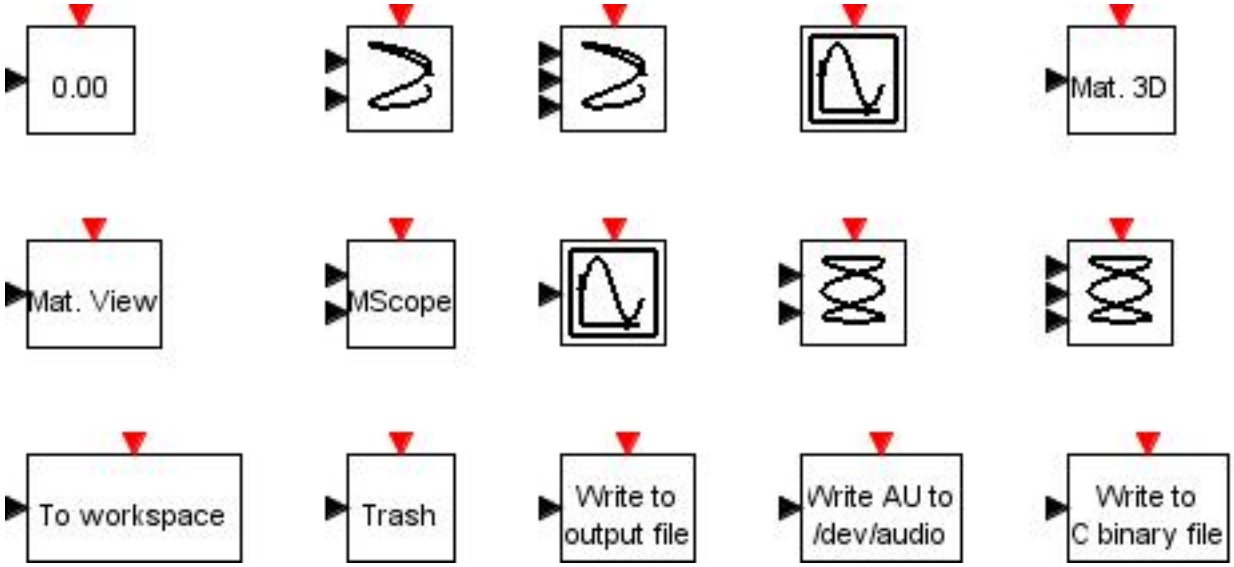
**Ramine Nikoukhah** - INRIA

# 17. Sinks palette

# Nome

Sinks_pal — Sinks palette

## Block Screenshot




## Module

- xcos

## Description

In Sinks palette, you can find a variety of blocks used to display (Scope) and write data during simulation and also some output ports used in superblocks.

The blocks of that palette doesn't have regular output ports.

## Blocks

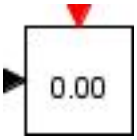
- AFFICH_m - Display
- CANIMXY - y=f(x) animated viewer
- CANIMXY3D - z=f(x,y) animated viewer
- CFSCOPE - Floating point scope
- CLKOUTV_f - Output activation port
- CMAT3D - Matrix z values 3D viewer
- CMATVIEW - Matrix Colormapped viewer
- CMSCOPE - Multi display scope
- CSCOPE - Single Display Scope
- CSCOPXY - y=f(x) permanent viewer
- CSCOPXY3D - z=f(x,y) permanent viewer
- ENDBLK - END block viewer

- END_c - END_c block viewer
- HALT_f - Halt block viewer
- OUTIMPL_f - Output implicit port viewer
- OUT_f - Output port viewer
- TOWS_c - Data to Scilab worspace
- TRASH_f — Trash block
- WFILE_f — Write to file
- WRITEAU_f — Write AU sound file
- Write binary data

# Nome

AFFICH_m — Display

## Block Screenshot




## Contents

- Display
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

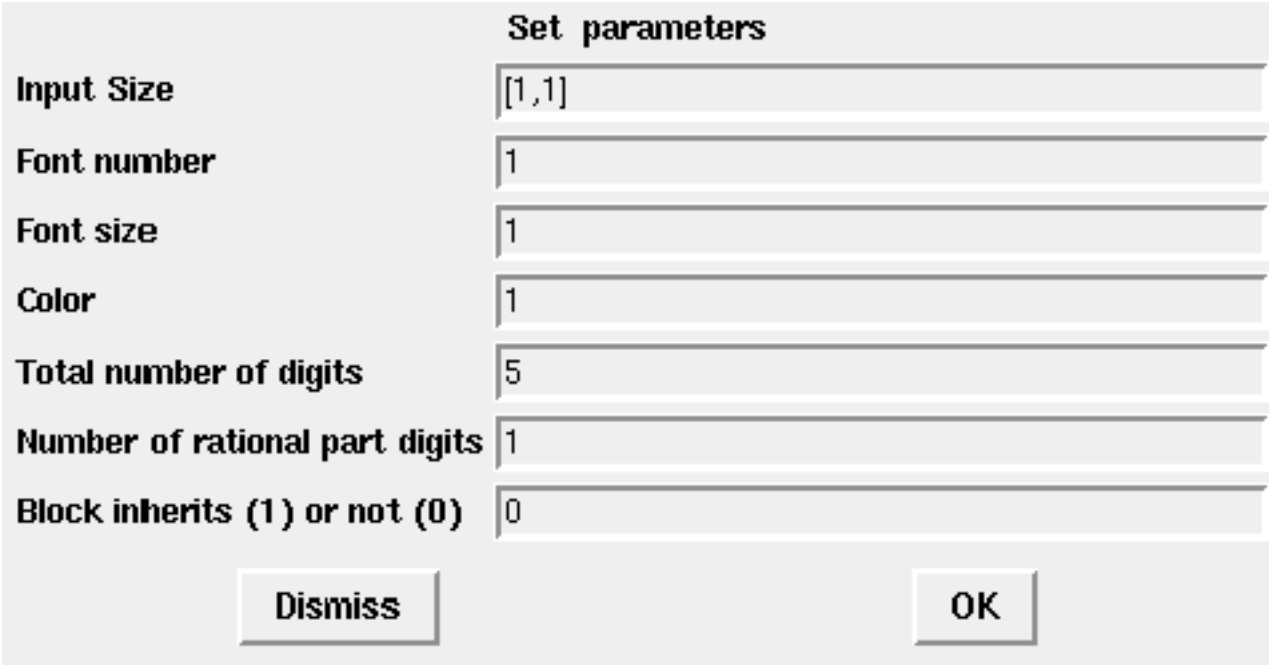
## Palette

- Sinks palette

## Description

This block displays the value of its unique input inside the block (in the diagram) during simulation.

## Dialog box



- **Input Size**

  Explicitly define the size of the input port.

  Properties : Type 'mat' of size [1,2].

- **Font number**

  integer, the selected font number (see xset).

  Properties : Type 'vec' of size 1

- **Font size**

  integer, the selected font size (set xset)

  Properties : Type 'vec' of size 1

- **Color**

  integer, the selected color for the text (see xset)

  Properties : Type 'vec' of size 1

- **Total number of digits**

  an integer greater than 3, the maximum number of digits used to represent the number (sign, integer part and rational part)

  Properties : Type 'vec' of size 1

- **Number of rational part digits**

  n integer greater than or equal 0, the number of digits used to represent the rational part

  Properties : Type 'vec' of size 1

- **Block inherits**

  Options to choose event inheritance from regular input or from explicit event input (0).

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *affich2*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/AFFICH_m.sci

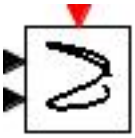
## Computational function

- SCI/modules/scicos_blocks/src/fortran/affich2.f (Type 0)

# Nome

CANIMXY — y=f(x) animated viewer

## Block Screenshot



## Contents


- y=f(x) animated viewer
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

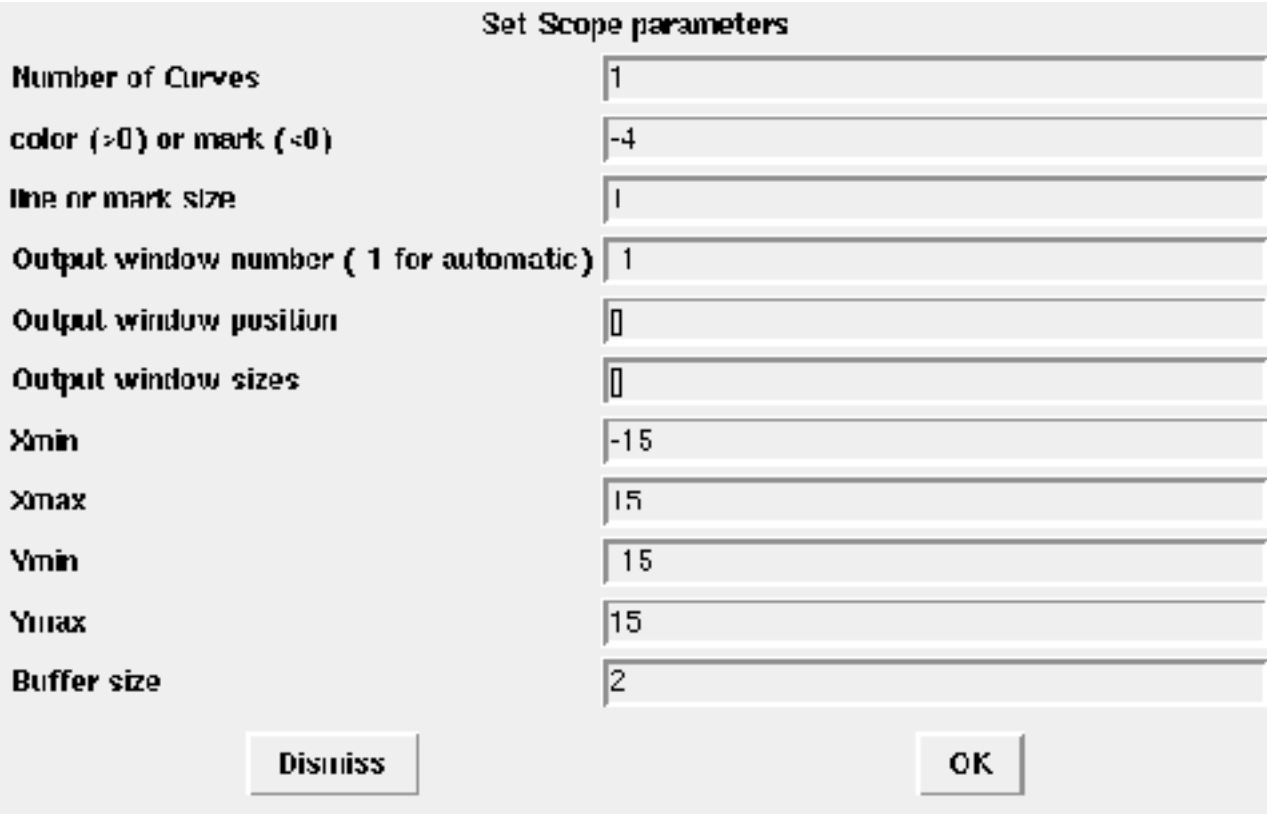

## Palette

- Sinks palette

## Description

This block realizes the visualization of the evolution of the two regular input signals by drawing the second input as a function of the first at instants of events on the event input port. When a point is drawn on screen it stays until the buffer length is reached. This scope is useful to make simple animations.

## Dialog box




- **Number of Curves**

  Set the number of curves.

  Properties : Type 'vec' of size 1

- **color**

  an integer. It is the color number ( $>= 0$ ) or marker type ( $< 0$ ) used to draw the evolution of the input port signal. See**xset()** for color (dash type) definitions.

  Properties : Type 'vec' of size 1

- **line or mark size**

  an integer.

  Properties : Type 'vec' of size 1

- **Output window number**

  The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).

  Properties : Type 'vec' of size 1

- **Output window position**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Output window sizes**

  a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

  Properties : Type 'vec' of size -1

- **Xmin**

  Minimum values of the first input; used to set up the X-axis of the plot in the graphics window.

  Properties : Type 'vec' of size 1

- **Xmax**

  Maximum values of the first input; used to set up the X-axis of the plot in the graphics window.

  Properties : Type 'vec' of size 1

- **Ymin**

  Minimum and maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.

  Properties : Type 'vec' of size 1

- **Ymax**

  Maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.

  Properties : Type 'vec' of size 1

- **Buffer size**

  An integer value. In order to minimize the number of graphics outputs, data may buffered.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0

- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *canimxy*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CANIMXY.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/canimxy.c (Type 4)

## See also

- CANIMXY3D - z=f(x,y) animated viewer

## Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

# Nome

CANIMXY3D — z=f(x,y) animated viewer

## Block Screenshot

## Contents


- z=f(x,y) animated viewer
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

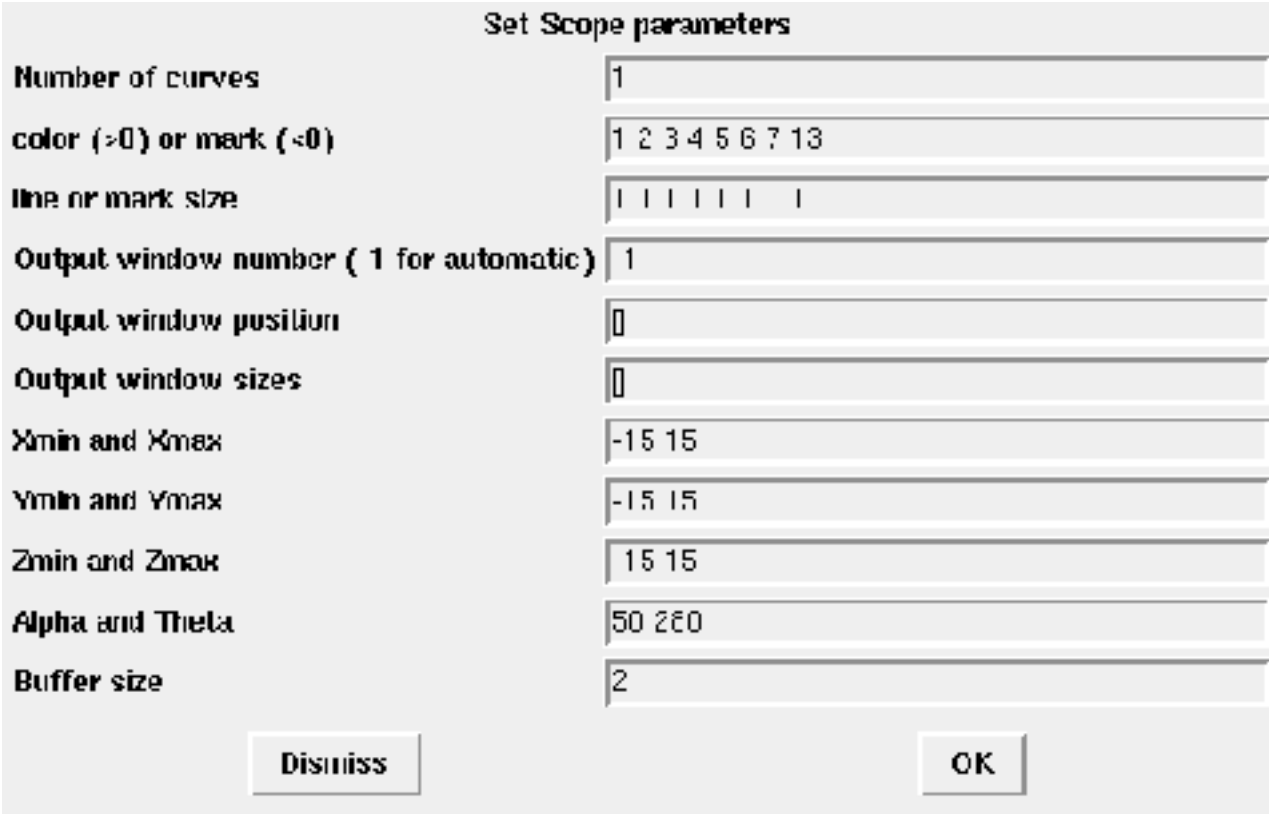

## Palette

- Sinks palette

## Description

This block realizes the visualization of the evolution of the three regular input signals by drawing the third input as a function of the two others at instants of events on the event input port. When a point is drawn on screen it stays until the buffer length is reached. This scope is useful to make simple animations.

## Dialog box




- **Number of Curves**

  Set the number of curves.

  Properties : Type 'vec' of size -1

- **color**

  an integer. It is the color number ( $>= 0$ ) or marker type ( $< 0$ ) used to draw the evolution of the input port signal. See**xset()** for color (dash type) definitions.

  Properties : Type 'vec' of size -1

- **line or mark size**

  an integer.

  Properties : Type 'vec' of size -1

- **Output window number**

  The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).

  Properties : Type 'vec' of size -1

- **Output window position**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Output window sizes**

  a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

  Properties : Type 'vec' of size -1

- **Xmin and Xmax**

  Minimum and Maximum values of the first input; used to set up the X-axis of the plot in the graphics window.

  Properties : Type 'vec' of size -1

- **Ymin and Ymax**

  Minimum and Maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.

  Properties : Type 'vec' of size -1

- **Zmin and Zmax**

  Minimum and Maximum values of the third input; used to set up the Z-axis of the plot in the graphics window.

  Properties : Type 'vec' of size -1

- **Alpha and Theta**

  Set Alpha and Theta for the 3D view.

  Properties : Type 'vec' of size -1

- **Buffer size**

  An integer value. In order to minimize the number of graphics outputs, data may buffered.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**

  **- port 3 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *canimxy3d*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CANIMXY3D.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/canimxy3d.c (Type 4)

## See also

- CANIMXY - y=f(x) animated viewer

## Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

# Nome

CFSCOPE — Floating point scope

## Block Screenshot

## Contents


- Floating point scope
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

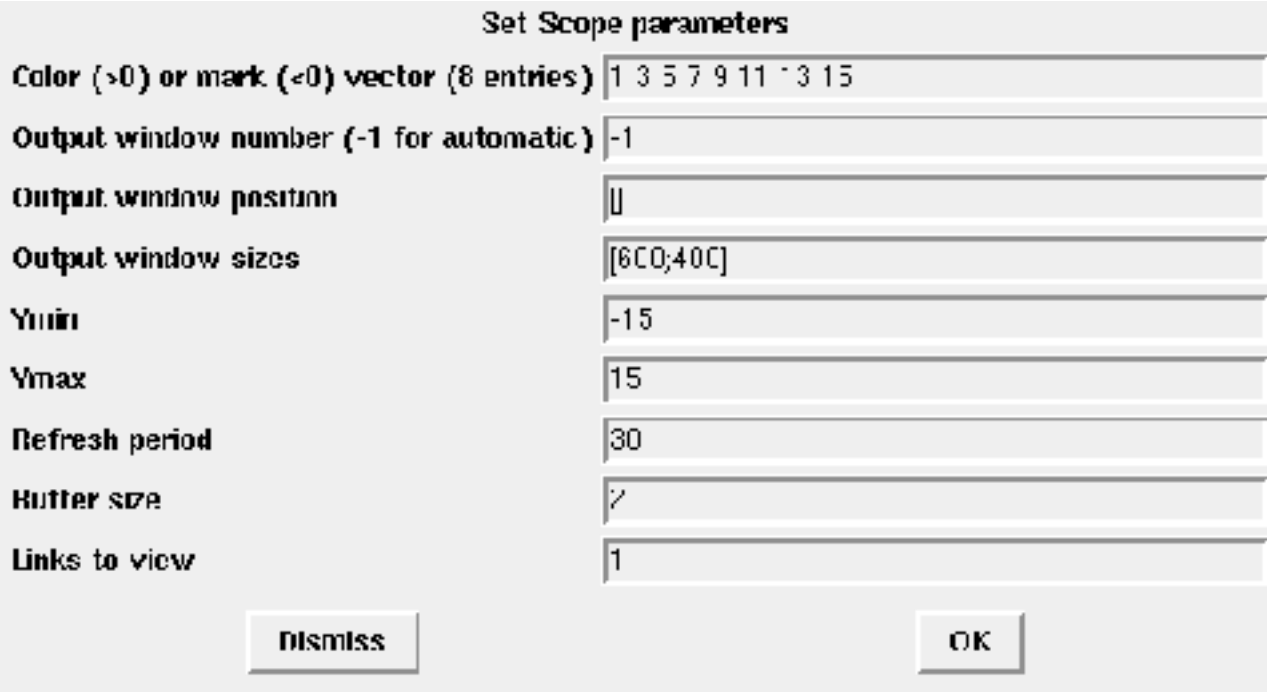

## Palette

- Sinks palette

## Description

This is a floating scope block.

## Dialog box




- **Color**

  a vector of integers. The i-th element is the color number ( $> 0$ ) or dash type ( $< 0$ ) used to draw the evolution of the i-th input port signal. See**plot2d** for color (dash type) definitions.

  Properties : Type 'vec' of size 8

- **Output window number**

  The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If default value is used**(1)** , Xcos define the output window number.

  Properties : Type 'vec' of size 1

- **Output window position**

  a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

  Properties : Type 'vec' of size -1

- **Output window sizes**

  a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

  Properties : Type 'vec' of size -1

- **Ymin**

  Minimum values of the input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size -1

- **Ymax**

  Maximum values of the input; used to set up the Y-axis of the plot in the graphics window.

  Properties : Type 'vec' of size 1

- **Refresh period**

  Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.

  Properties : Type 'vec' of size

- **Buffer size**

  To improve efficiency it is possible to buffer the input data. The drawing is only done after each-**Buffer size** call to the block.

  Properties : Type 'vec' of size 1

- **Links to view**

  This parameter allows you to display the output of specified link.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cfscope*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CFSCOPE.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/cfscope.c (Type 4)

## Authors

- **Ramine Nikoukhah** INRIA

- **Benoit Bayol**

# Nome

CMAT3D — Matrix z values 3D viewer

## Block Screenshot

## Contents


- Matrix z values 3D viewer
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

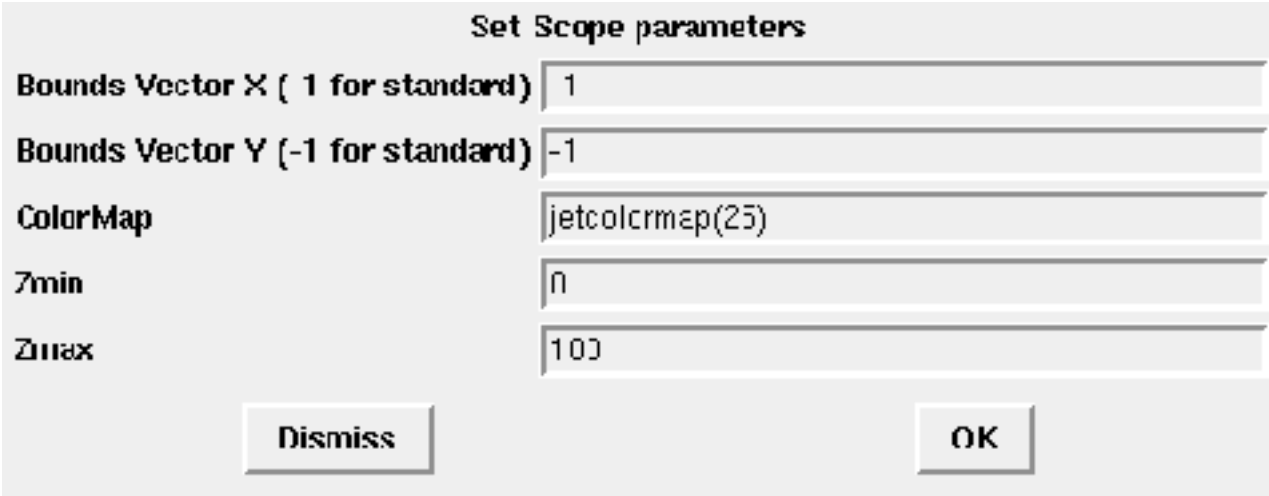

## Palette

- Sinks palette

## Description

CMAT3D is a scope that shows matrix values as z values on a xy grid.

## Dialog box



- **Bounds Vector X (-1 for standard)**

  If you let -1 the x ticks would be between 0 and 1 else you can put your own vector.

  Properties : Type 'vec' of size -1.

- **Bounds Vector Y (-1 for standard)**

  If you let -1 the x ticks would be between 0 and 1 else you can put your own vector.

  Properties : Type 'vec' of size -1.

- **ColorMap**

  The colormap is a range color linked to the window output of the scope. You can put a jetcolormap or hotcolormap or graycolormap or your own (see colormap help).

  Properties : Must be a mx3 matrix and m > = 3

- **Zmin**

  Minimum value in Z values

  Properties : Type 'vec' of size 1.

- **Zmax**

  Maximum values in Z values

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cmat3d*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CMAT3D.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/cmat3d.c (Type 4)

## See also

- CMATVIEW - Matrix Colormapped viewer

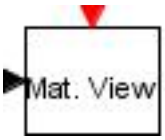
## Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

# Nome

CMATVIEW — Matrix Colormapped viewer

## Block Screenshot




## Contents


- Matrix Colormapped viewer
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

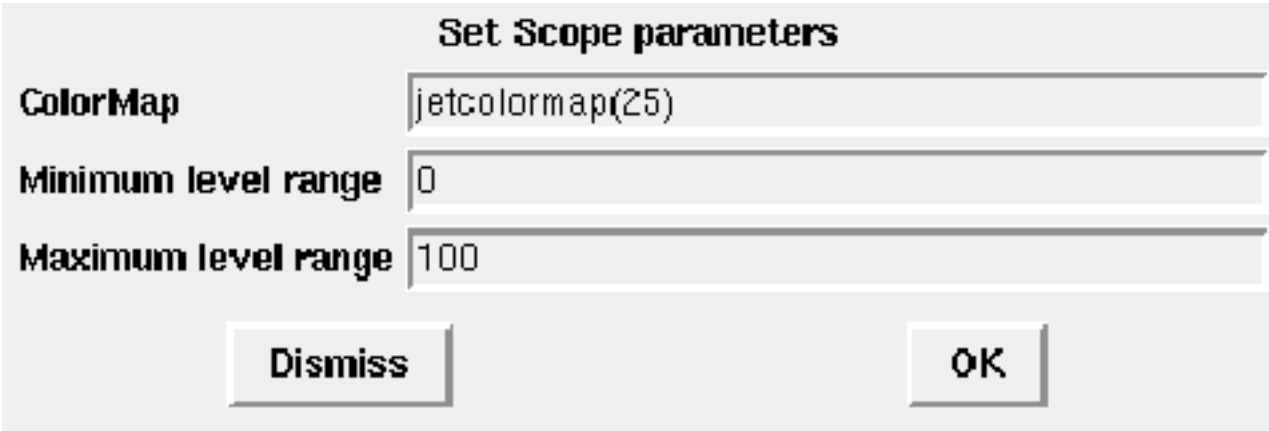

## Palette

- Sinks palette

## Description

CMATVIEW is a scope that shows matrix values on a colormapped grid.

## Dialog box




- **ColorMap**

The colormap is a range color linked to the window output of the scope. You can put a jetcolormap or hotcolormap or graycolormap or your own (see colormap help).

Properties : Must be a mx3 matrix and m > = 3

- **Minimum level range**

  The minimum level range is the minimum value who comes in the regular input port. It would be linked to the 'cold value' of the colormap.

  Properties : A scalar

- **Maximum level range**

  The maximum level range is the maximum value who comes in the regular input port. It would be linked to the 'hot value' of the colormap.

  Properties : A scalar

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,-2] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cmatview*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CMATVIEW.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/cmatview.c (Type 4)

## See also

- CMAT3D - Matrix z values 3D viewer

## Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

# Nome

CMSCOPE — Multi display scope

## Block Screenshot

## Contents

- Multi display scope
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

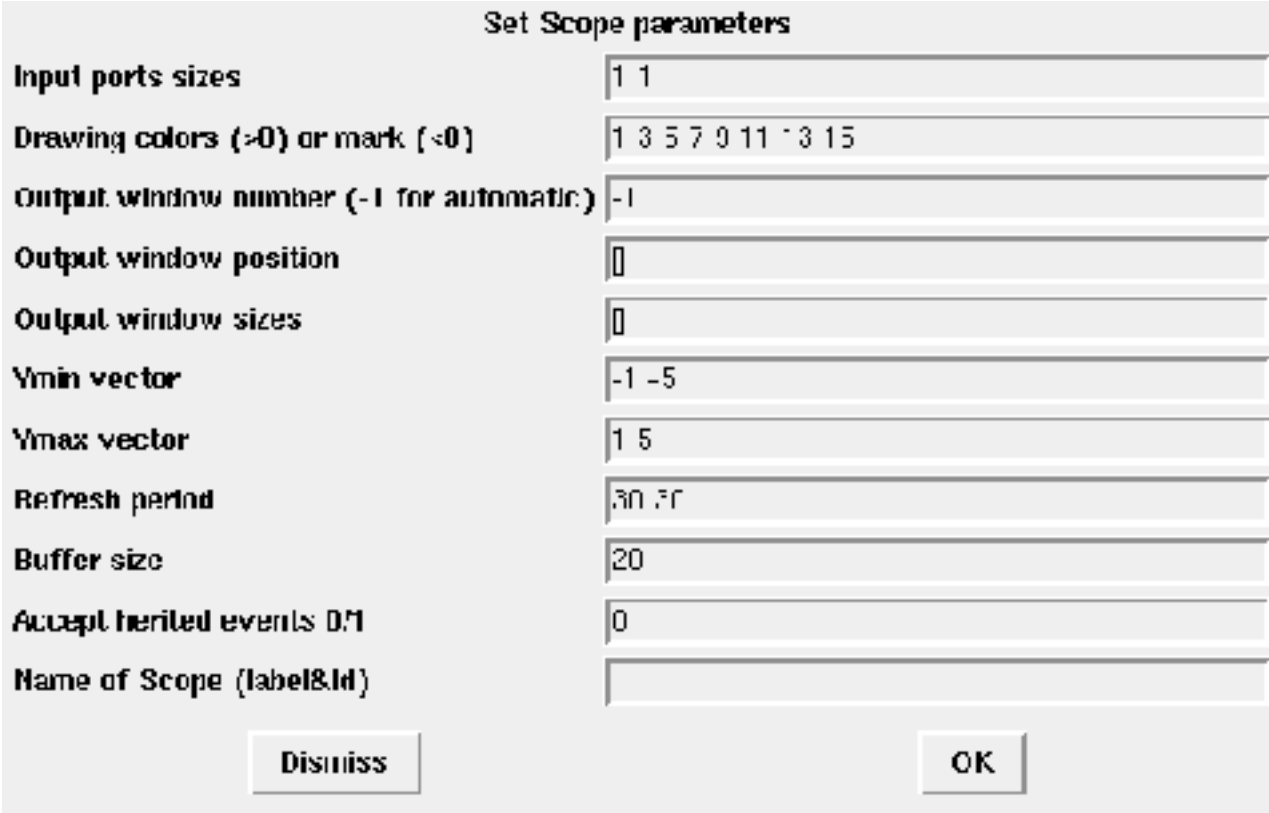
## Palette

- Sinks palette

## Description

When you start a simulation, Xcos open Scope windows. The Scope block displays its input with respect to simulation time. The Scope block can have multiple axes (one per port); all axes have a common time range with independent y-axes. The Scope allows you to adjust the amount of time and the range of input values displayed.

## Dialog box




- **Input ports sizes**

  It allows multiple input ports.

  Properties : Type 'vec' of size -1

- **Drawing colors**

  a vector of integers. The i-th element is the color number ( $> 0$ ) or dash type ( $< 0$ ) used to draw the evolution of the i-th input port signal. See**plot2d** for color (dash type) definitions.

  Properties : Type 'vec' of size -1

- **Output window number**

  The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If default value is used**(1)** , Xcos define the output window number.

  Properties : Type 'vec' of size 1

- **Output window position**

  a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

  Properties : Type 'vec' of size -1

- **Output window sizes**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size size -1

- **Ymin vector**

  Minimum values of the input; used to set up the Y-axis of the plot in the graphics window.

  Properties : Type 'vec' of size size(%1,'*')

- **Ymax vector**

  Maximum values of the input; used to set up the Y-axis of the plot in the graphics window.

  Properties : Type 'vec' of size size(%1,'*')

- **Refresh period**

  Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.

  Properties : Type 'vec' of size size(%1,'*')

- **Buffer size**

  To improve efficiency it is possible to buffer the input data. The drawing is only done after each-**Buffer size** call to the block.

  Properties : Type 'vec' of size 1

- **Accept herited events 0/1**

  if 0**CSCOPE_f** draws a new point only when an event occurs on its event input port. if 1**CSCOPE_f** draws a new point when an event occurs on its event input port and when it's regular input changes due to an event on an other upstrem block (herited events).

  Properties : Type 'vec' of size 1

- **Name of Scope**

  Name/label of the block.

  Properties : Type 'str' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1

- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cmscope*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CMSCOPE.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/cmscope.c (Type 4)

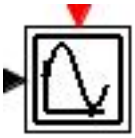
## Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

# Nome

CSCOPE — Single Display Scope

## Block Screenshot



## Contents


- Single Display Scope
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

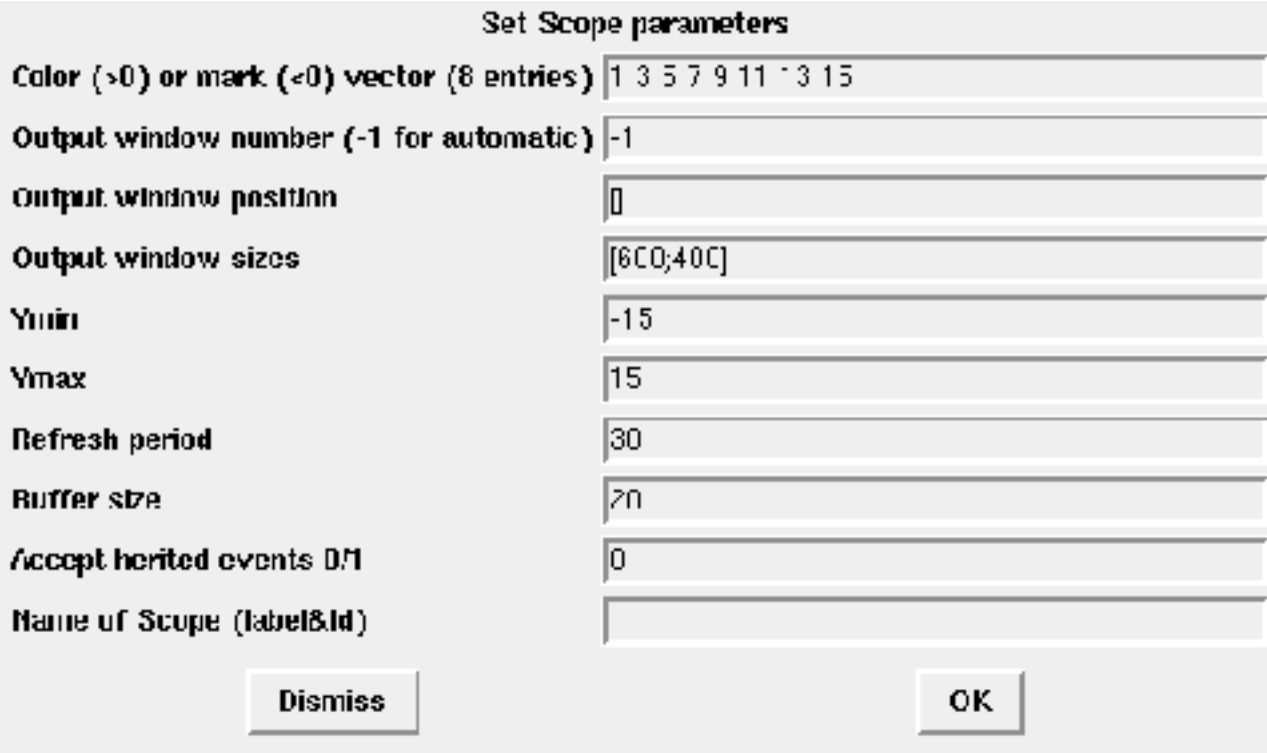

## Palette

- Sinks palette

## Description

The Scope block displays its input with respect to simulation time. Both axes have a common range. The Scope allows you to adjust the amount of time and the range of input values displayed.

## Dialog box




- **Color**

  a vector of integers. The i-th element is the color number ( $> 0$ ) or dash type ( $< 0$ ) used to draw the evolution of the i-th input port signal. See**plot2d** for color (dash type) definitions.

  Properties : Type 'vec' of size 8

- **Output window number**

  The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If default value is used**(1)** , Scicos define the output window number.

  Properties : Type 'vec' of size 1

- **Output window position**

  a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

  Properties : Type 'vec' of size 1

- **Output window sizes**

  a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

  Properties : Type 'vec' of size -1

- **Ymin**

Minimum values of the input; used to set up the Y-axis of the plot in the graphics window.

Properties : Type 'vec' of size 1

- **Ymax**

  Maximum values of the input; used to set up the Y-axis of the plot in the graphics window.

  Properties : Type 'vec' of size 1

- **Refresh period**

  Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.

  Properties : Type 'vec' of size 1

- **Buffer size**

  To improve efficiency it is possible to buffer the input data. The drawing is only done after each-**Buffer size** call to the block.

  Properties : Type 'vec' of size 1

- **Accept herited events 0/1**

  if 0**CSCOPE_f** draws a new point only when an event occurs on its event input port. If 1**CSCOPE_f** draws a new point when an event occurs on its event input port and when it's regular input changes due to an event on an other upstream block (herited events).

  Properties : Type 'vec' of size 1

- **Name of Scope**

  Name/label of the block.

  Properties : Type 'str' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cscope*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CSCOPE.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/cscope.c (Type 4)

## See also

- CMSCOPE - Multi display scope

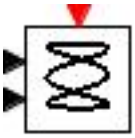
## Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

# Nome

CSCOPXY — y=f(x) permanent viewer

## Block Screenshot



## Contents


- y=f(x) permanent viewer
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

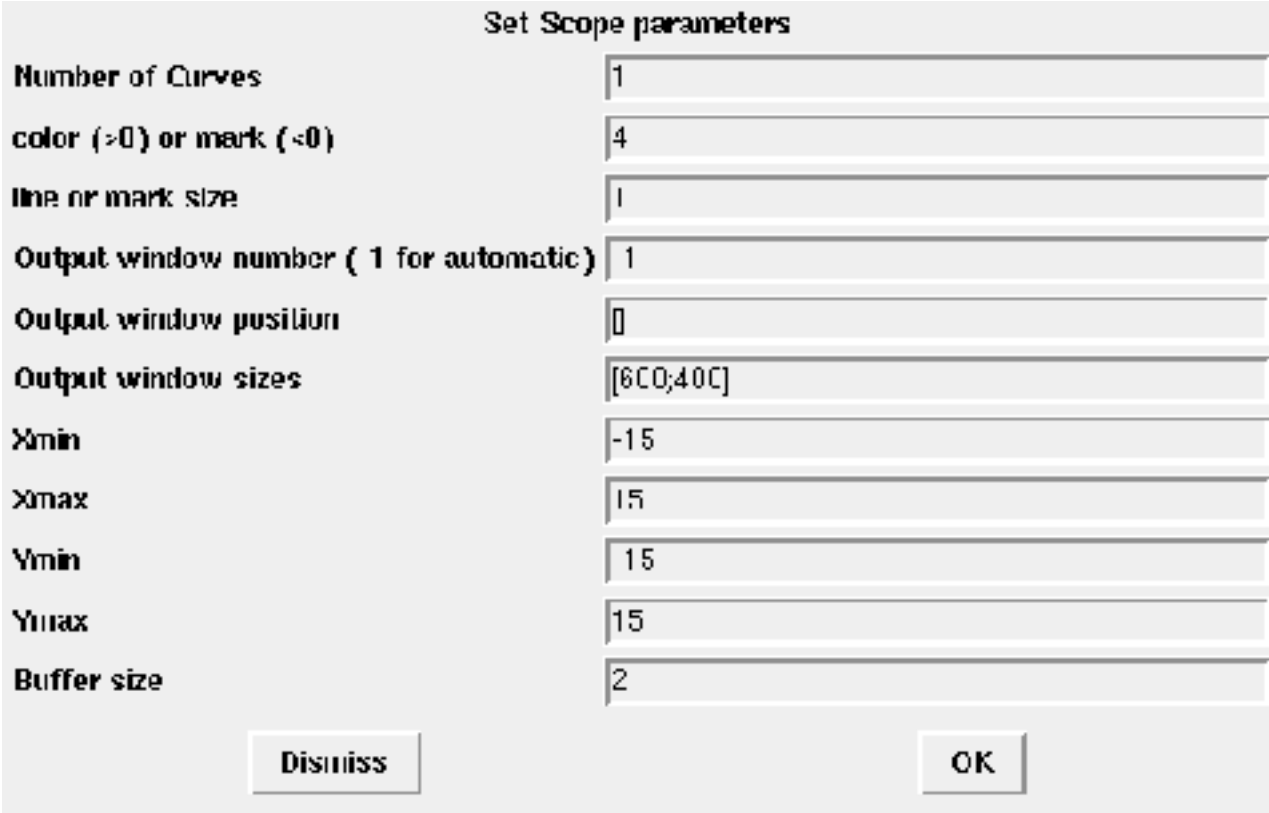

## Palette

- Sinks palette

## Description

This block realizes the visualization of the evolution of the two regular inputs signals by drawing the second input as a function of the first at instants of events on the event input port. When a point is drawn on screen it stays until the simulation is finished.

## Dialog box




- **Number of Curves**

  Set the number of curves. Properties : Type 'vec' of size 1

- **color**

  $> 0$ $< 0$

  an integer. It is the color number ( ) or dash type ( ) used to draw the evolution of the input port signal. See**plot2d** for color (dash type) definitions.

  Properties : Type 'vec' of size 1

- **line or mark size**

  an integer.

  Properties : Type 'vec' of size 1

- **Output window number**

  The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).

  Properties : Type 'vec' of size 1

- **Output window position**

  a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Output window sizes**

  a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

  Properties : Type 'vec' of size -1

- **Xmin**

  Minimum values of the first input; used to set up the X-axis of the plot in the graphics window.

  Properties : Type '' of size

- **Xmax**

  Maximum values of the first input; used to set up the X-axis of the plot in the graphics window.

  Properties : Type 'vec' of size 1

- **Ymin**

  Minimum values of the second input; used to set up the Y-axis of the plot in the graphics window.

  Properties : Type 'vec' of size 1

- **Ymax**

  Maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.

  Properties : Type 'vec' of size 1

- **Buffer size**

  To improve efficiency it is possible to buffer the input data. The drawing is only done after each Buffer size call to the block.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no

- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cscopxy*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CSCOPXY.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/cscopxy.c (Type 4)

## See also

- CSCOPXY3D - z=f(x,y) permanent viewer

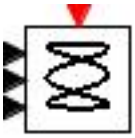
## Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

# Nome

CSCOPXY3D — z=f(x,y) permanent viewer

## Block Screenshot



## Contents


- z=f(x,y) permanent viewer
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

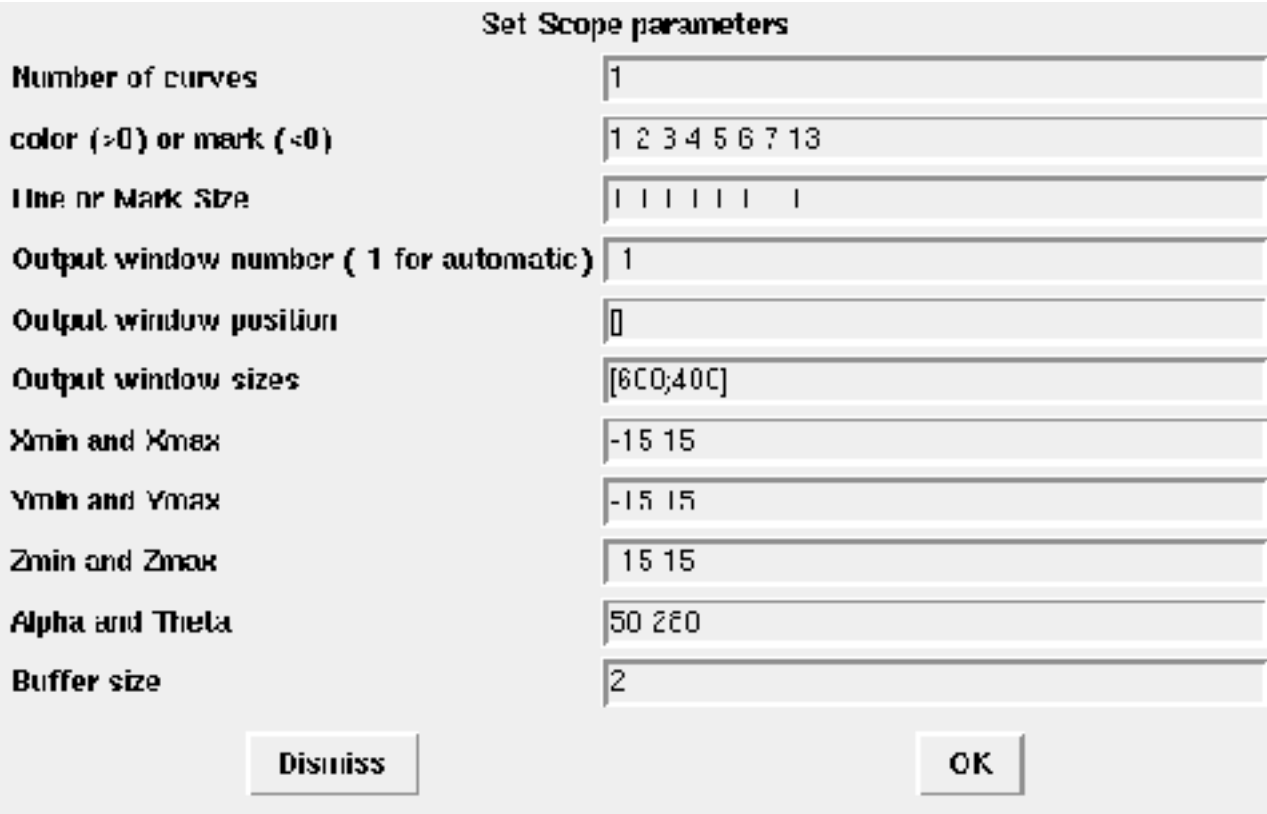

## Palette

- Sinks palette

## Description

This block realizes the visualization of the evolution of the three regular inputs signals by drawing the third input as a function of the two others at instants of events on the event input port. When a point is drawn on screen it stays until the simulation is finished.

## Dialog box




- **Number of Curves**

  Set the number of curves.

  Properties : Type 'vec' of size -1

- **color**

  $>= 0$ $< 0$

  an integer. It is the color number ( ) or marker type ( ) used to draw the evolution of the input port signal. See**xset()** for color (dash type) definitions.

  Properties : Type 'vec' of size -1

- **line or mark size**

  an integer.

  Properties : Type 'vec' of size -1

- **Output window number**

  The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).

  Properties : Type 'vec' of size -1

- **Output window position**

a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Properties : Type 'vec' of size -1

- **Output window sizes**

  a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

  Properties : Type 'vec' of size -1

- **Xmin and Xmax**

  Minimum and Maximum values of the first input; used to set up the X-axis of the plot in the graphics window.

  Properties : Type 'vec' of size -1

- **Ymin and Ymax**

  Minimum and Maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.

  Properties : Type 'vec' of size -1

- **Zmin and Zmax**

  Minimum and Maximum values of the third input; used to set up the Z-axis of the plot in the graphics window.

  Properties : Type 'vec' of size -1

- **Alpha and Theta**

  Set Alpha and Theta for the 3D view.

  Properties : Type 'vec' of size -1

- **Buffer size**

  An integer value. In order to minimize the number of graphics outputs, data may buffered.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**

  **- port 3 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cscopxy3d*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/CSCOPXY3D.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/cscopxy3d.c (Type 4)

## See also

- CSCOPXY - y=f(x) permanent viewer

## Authors

- **Ramine Nikoukhah** INRIA
- **Benoit Bayol**

# Nome

ENDBLK — END block

## Block Screenshot

## Contents


- END block
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “See also”
  - “Authors”

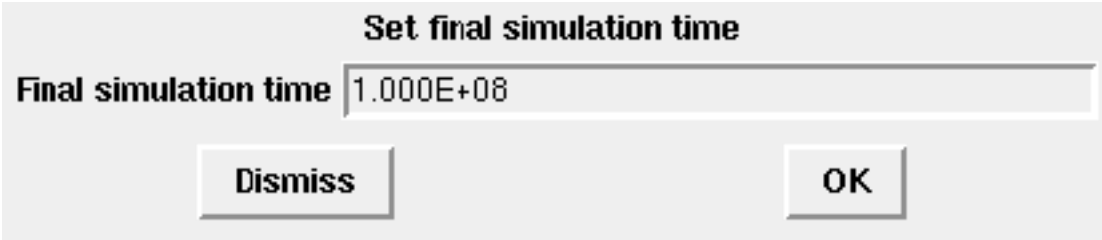

## Palette

- Sinks - Sinks palette

## Description

That block can be used to set the final time of the simulation. When that block is truely parametrized then the simulator will jump to the 'final integration time' defined in the Setup item of the simulate Menu from the time defined by the parameter 'Final simulation time' of the dialog box. That parameter can be a numerical value or a symbolic variable defined in the scicos context.

## Dialog box




- **Final simulation time**

  Set the final time of the simulation.

  When simulator reaches that value then the current time will jump to the final integration time.

  Properties : Type 'vec' of size 1.

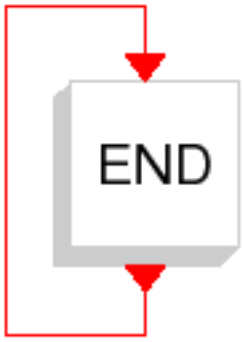
## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/ENDBLK.sci

## Compiled Super Block content




## See also

- END_c - END_c block (Scicos Block)

## Authors

**Alan Layec** - INRIA

# Nome

END_c — END_c block

## Block Screenshot

## Contents


- END_c block
- • “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Sinks - Sinks palette

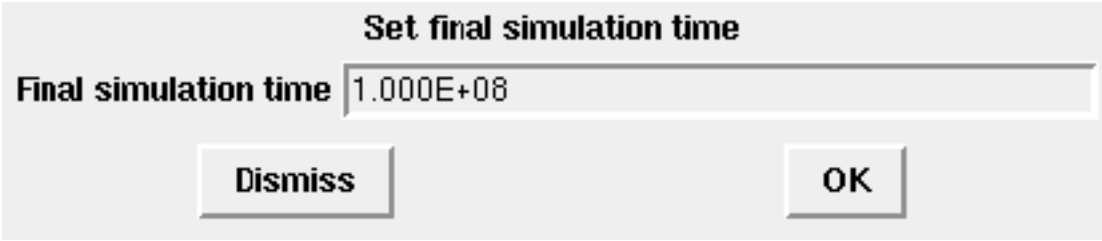
## Description

That block increases the current time to the final integration time of the simulation when it is activated :

$$T_f = T_{cur}$$

with **Tcur** the activation date of the block and **Tf** the final integration time defined in the in the Setup item of the simulate Menu (scs_m.props.tf).

## Dialog box




- **Final simulation time**

That parameter is a date for an initial output event. By using a feed back from the event output port to the event input port, then that block can himself end the simulation at the time defined by this parameter.

Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *scicosexit*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Events/END_c.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/scicosexit.c (Type 4)

## See also

- ENDBLK - END block (Scicos Block)

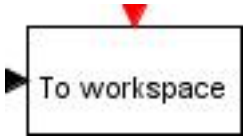
## Authors

**Alan Layec** - INRIA

# Nome

TOWS_c — Data to Scilab worspace

## Block Screenshot




## Contents


- Data to Scilab worspace
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

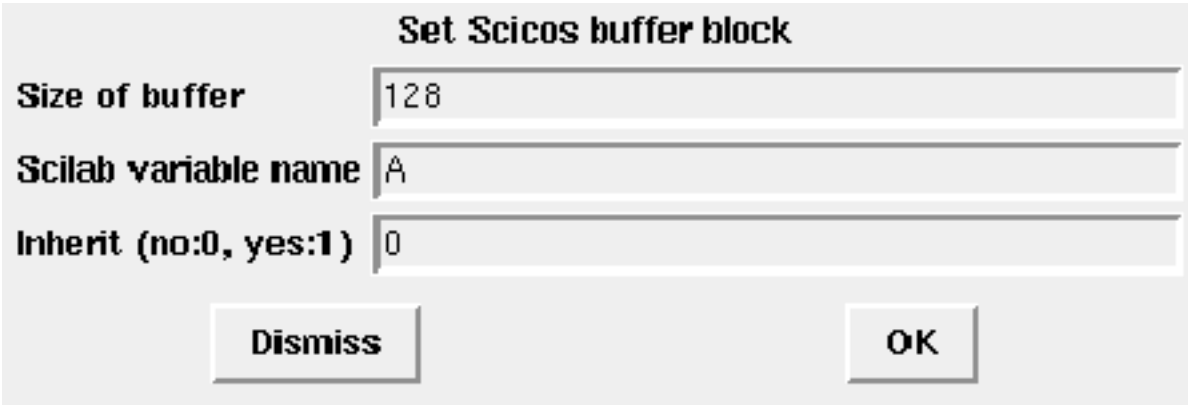
- Sinks palette

## Description

That block is used to put simulated data in the scilab workspace.

Each sampling time, both dates and values of input are recorded.

## Dialog box




- **Size of buffer**

  Set the size of the input buffer. That gives the total number of samples recorded during the simulation.

That buffer is a circulate buffer.

Properties : Type 'vec' of size 1.

- **Scilab variable name**

  Set the name of the Scilab variable. This must be a valid variable name.

  The simulation must be finished to retrieve that variable in the Scilab workspace.

  Properties : Type 'str' of size 1.

- **Inherit (no:0, yes:1)**

  Options to choose event inheritance from regular input or from explicit event input (0).

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type -1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *tows_c*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/TOWS_c.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/tows_c.c (Type 4)

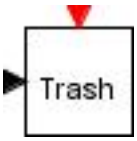
## Authors

**Alan Layec** - INRIA

# Nome

TRASH_f — Trash block

## Block Screenshot




## Contents


- Trash block
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”


## Palette

- Sinks palette

## Description

That block is an end-block. It do nothing.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *trash*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/TRASH_f.sci

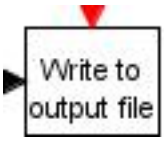
## Computational function

- SCI/modules/scicos_blocks/src/fortran/trash.f (Type 0)

# Nome

WFILE_f — Write to file

## Block Screenshot




## Contents


- Write to file
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Sinks palette

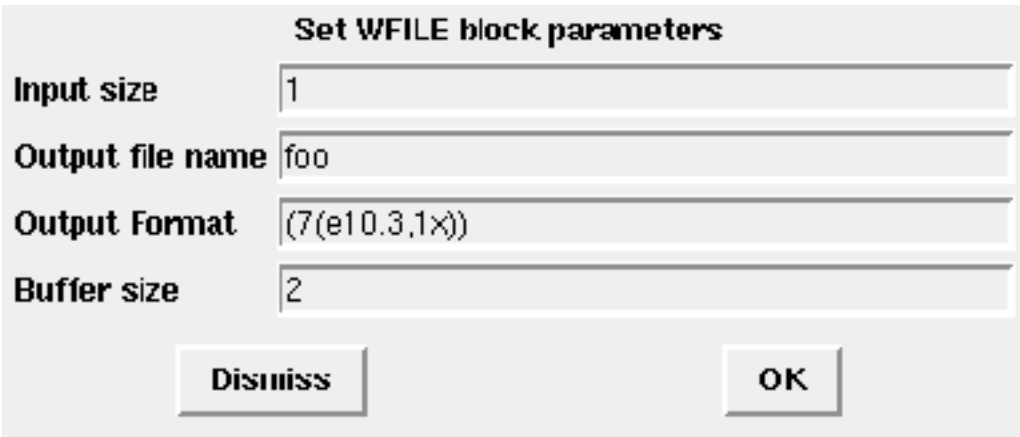
## Description

This block allows user to save data in a file, in formatted and binary mode. Each call to the block corresponds to a record in the file. Each record has the following form: where is the value of time when block is called and is the ith input value.

This block does not manage UTF filename.

The pair block is .

## Dialog box



- **Input size**

  a scalar. This fixes the input size.

  Properties : Type 'vec' of size 1.

- **Output file name**

  a character string defining the path of the file.

  Properties : Type 'str' of size 1.

- **Output Format**

  a character string defining the Fortran format to use or nothing for an unformatted (binary) write. If given, the format must began by a left parenthesis and end by a right parenthesis. example: $(e10.3)$.

  Properties : Type 'str' of size 1.

- **Buffer size**

  To improve efficiency it is possible to buffer the input data. Write on the file is only done after each**Buffer size** call to the block.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *writef*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/WFILE_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/writef.f (Type 0)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

WRITEAU_f — Write AU sound file

## Block Screenshot

## Contents


- Write AU sound file
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

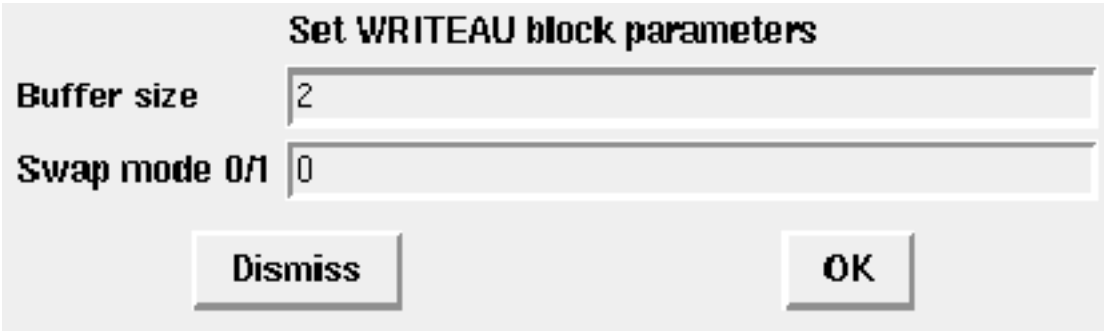

## Palette

- Sinks palette

## Description

This block writes a sound file specified by the string *.au file. The data should be arranged with one channel per column. Amplitude values outside the range [-1,+1] are clipped prior to writing. auwrite supports multichannel data for 8-bit mu-law and 8- and 16-bit linear formats.

## Dialog box




- **Buffer size**

  To improve efficiency it is possible to buffer the input data. read on the file is only done after each Buffer size call to the block.

  Properties : Type 'vec' of size 1

- **Swap mode 0/1**

  With**Swap mode=1** the file is supposed to be coded in "little endian IEEE format" and data are swaped if necessary to match the IEEE format of the processor. If**Swap mode=0** then automatic bytes swap is disabled.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *writeau*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/WRITEAU_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/writeau.c (Type 2)

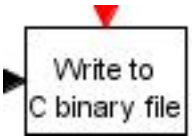
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

WRITEC_f — Write binary data

## Block Screenshot




## Contents


- Write binary data
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

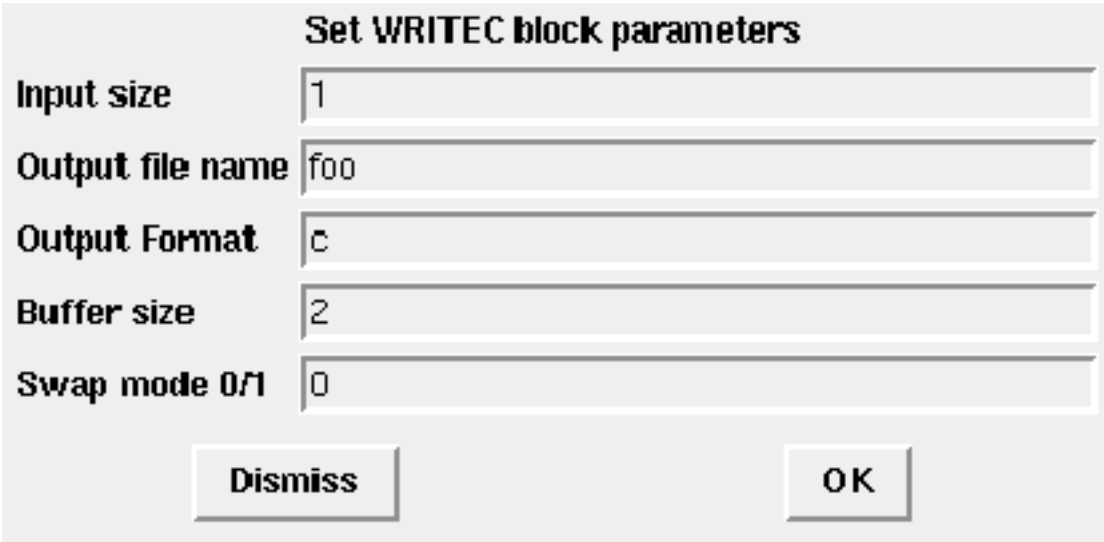

## Palette

- Sinks palette

## Description

This block allows user to write data in a C binary file.

## Dialog box




- **Input size**

a scalar, the size of the input.

Properties : Type 'vec' of size 1

- **Output file name**

  a character string defining the output file name.

  Properties : Type 'str' of size 1

- **Output Format**

  a character string defining the format to use.

  Properties : Type 'str' of size 1

- **Buffer size**

  To improve efficiency it is possible to buffer the input data. read on the file is only done after each**Buffer size** call to the block.

  Properties : Type 'vec' of size 1

- **Swap mode 0/1**

  With**Swap mode=1** the file is supposed to be coded in ``little endian IEEE format'' and data are swaped if necessary to match the IEEE format of the processor. If**Swap mode=0** then automatic bytes swap is disabled.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *writec*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sinks/WRITEC_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/writec.c (Type 2)

## Authors

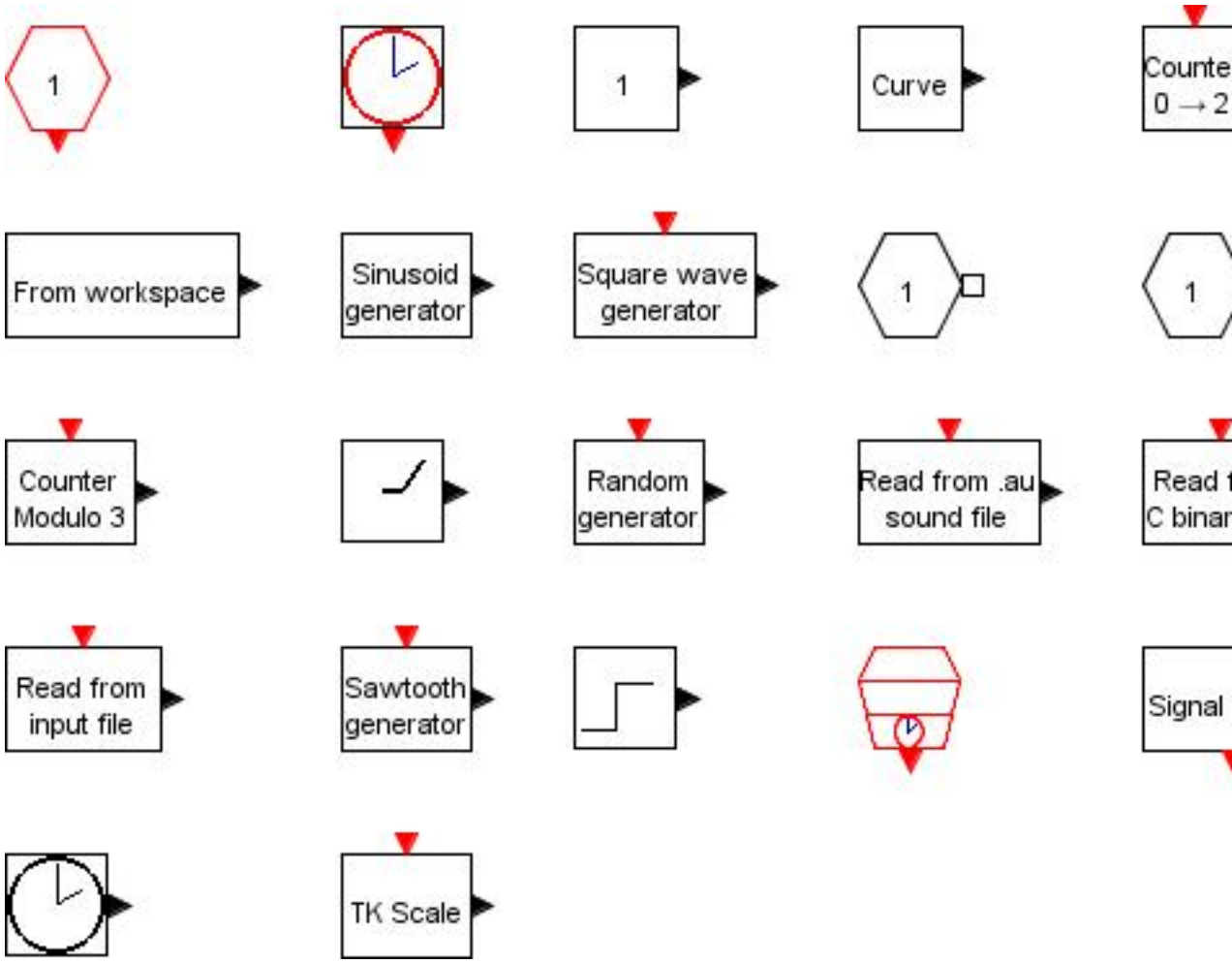
**Ramine Nikoukhah** - INRIA

# 18. Sources palette

# Nome

Sources_pal — Sources palette

## Block Screenshot




## Module

- xcos

## Description

Most of blocks of the Source palette can be viewed as data generators. That palette also contains blocks to read data from files and input ports used in superblocks.

The blocks of that palette doesn't have regular input ports.

## Blocks

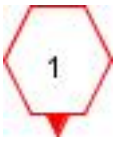
- CLKINV_f - Input activation port
- CLOCK_c - Activation clock
- CONST_m - Constant
- Counter - Counter

- CURV_f - Curve
- FROMWSB - Data from Scilab workspace to Xcos
- GENSIN_f - Sin generator
- GENSQR_f - Square wave generator
- IN_f - Input Port
- INIMPL_f - Input implicit port
- Modulo_Count - Modulo counter
- RAMP - Ramp
- RAND_m - Random generator
- READAU_f - Read AU sound file
- READC_f - Read binary data
- RFILE_f - Read from file
- SampleCLK - Sample Time Clock
- SAWTOOTH_f - Sawtooth generator
- Sigbuilder - Signal creator/generator
- STEP_FUNCTION - Step function generator
- TIME_f - Time
- TKSCALE - Adjust constant value with a tk widget

# Nome

CLKINV_f — Input activation port

## Block Screenshot




## Contents

- Input activation port
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

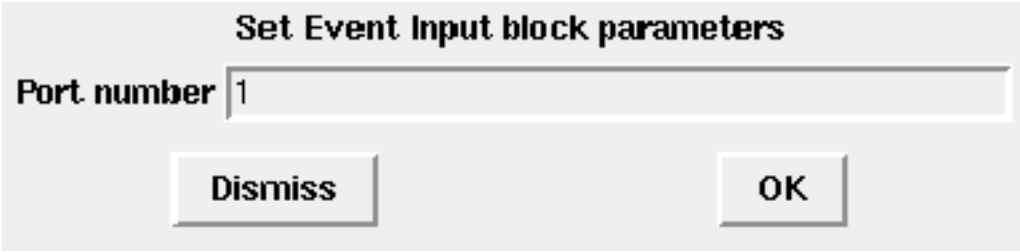
## Palette

- Sources palette

## Description

This block must only be used inside Xcos Super Blocks to represent an event input port. In a Super Block, the event input ports must be numbered from 1 to the number of event input ports.

## Dialog box




- **Port number**

  an integer defining the port number.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no

- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *input*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/CLKINV_f.sci

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

CLOCK_c — Activation clock

## Block Screenshot

## Contents


- Activation clock
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “See also”
  - “Authors”


## Palette

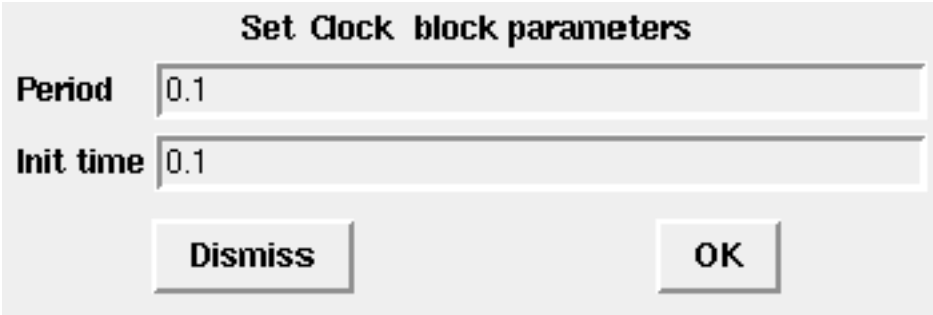
- Sources palette

## Description

This block is a Super Block constructed by feeding back the output of the block into its input event port.

The unique output of this block generates a regular train of events that's scheduled by the dialog parameter **Period**.

## Dialog box




- **Period**

  scalar.

  One over the frequency of the clock.

Period is the time that separates two output events.

Properties : Type 'vec' of size 1.

- **Init time**

  scalar.

  Starting date.

  If negative the clock never starts.

  Properties : Type 'vec' of size 1.

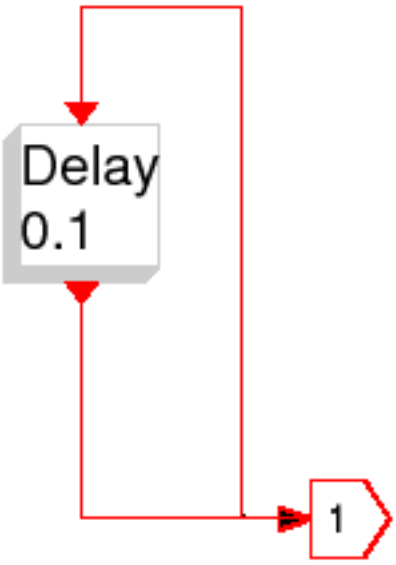
## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/CLOCK_c.sci

## Compiled Super Block content



## See also

- EVTDLY_c - Event delay

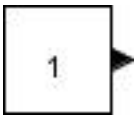
## Authors

**Alan Layec** - INRIA

# Nome

CONST_m — Constant

## Block Screenshot




## Contents


- Constant
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Sources palette

## Description

This block is a constant value generator.

## Dialog box

- **Constant**

  A matrix of any type.

  The size of the matrix gives the size of the regular output port.

  The constant(i,j) value is the component(i,j) value of the output port.

  From this value the block inherits its data type.

  Properties : Type 'mat' of size [-1,-2].

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *cstblk4*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/CONST_m.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/cstblk4.c (Type 4)

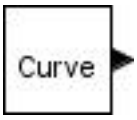
## Authors


- **Fady NASSIF** INRIA
- **Alan Layec** INRIA

# Nome

CURV_f — Curve

## Block Screenshot




## Contents


- Curve
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”


## Palette

- Sources palette

## Description

This block defines a tabulated function of time. Between mesh points block performs a linear interpolation. Outside tabulation block outputs last tabulated value. User may define the tabulation of the function using a curve editor.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no

- **object discrete-time state:** no
- **name of computational function:** *intplt*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/CURV_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/intplt.f (Type 0)

## See also

- Sigbuilder - Signal creator/generator

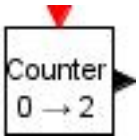
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

Counter — Counter

## Block Screenshot




## Contents


- Counter
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “See also”
  - “Authors”

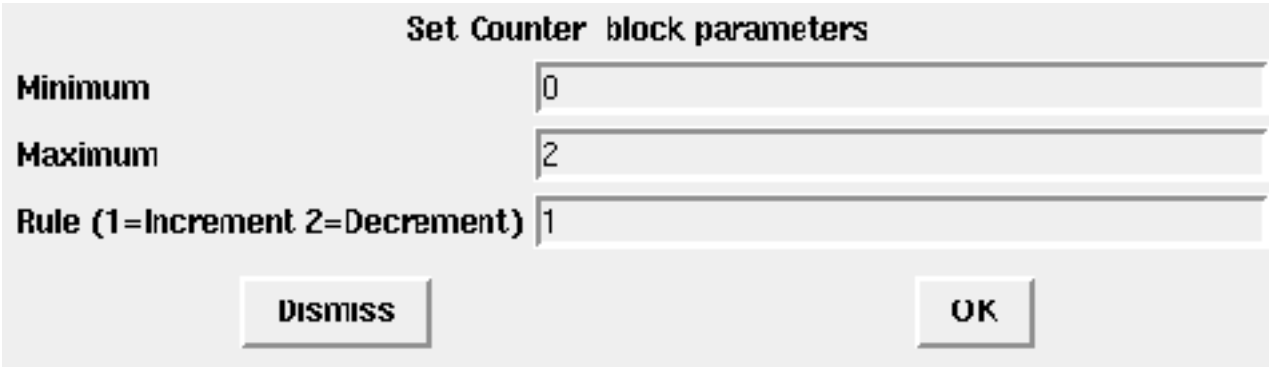

## Palette

- Sources palette

## Description

This block counts from minimal to maximal or from maximal to minimal depending on the choice of the rule.

## Dialog box




- **Minimum**

  The lowest number of the counter.

Properties : Type 'vec' of size 1.

- **Maximum**

  The highest number of the counter.

  Properties : Type 'vec' of size 1.

- **Rule (1=Increment 2=Decrement)**

  The rule of counting. If it is 1 then the counter counts from the lower number to the higher number. the count in this case is increasing. otherwise, if the rule is equal to 2 the counter will decrease and it will count from the higher number to the lower one.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *counter*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/Counter.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/counter.c (Type 4)

## See also

- Modulo_Count - Modulo counter (xcos Block)

## Authors

**Fady NASSIF** - INRIA

# Nome

FROMWSB — Data from Scilab workspace to Xcos

## Block Screenshot

## Contents

- Data from Scilab workspace to Xcos
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “See also”
  - “Authors”

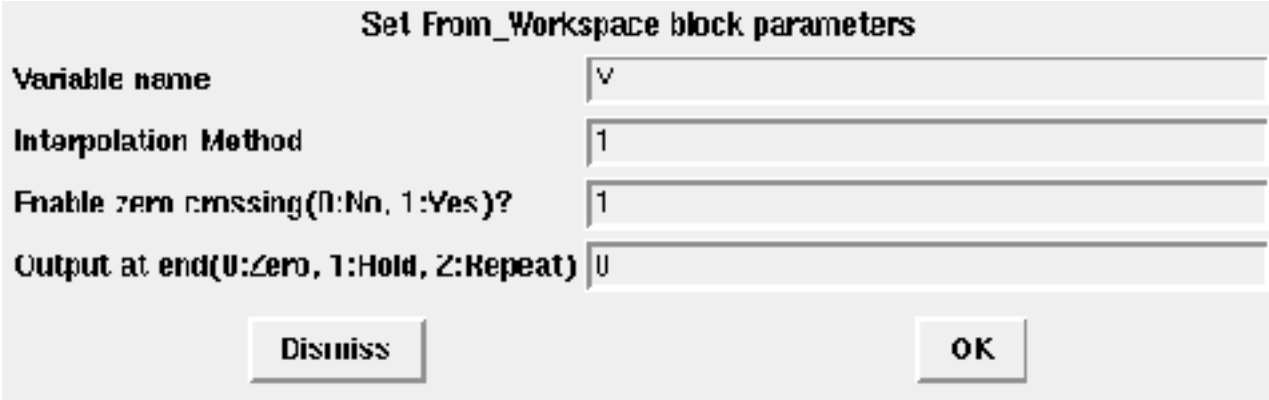
## Palette

- Sources palette

## Description

That block is used to get data defined in the Scilab workspace and use them in Xcos. Data should have "time" and "values" fields.

## Dialog box




- **Variable name:** This variable is defined in Scilab and should be a structure with two fields, i.e., a "time" field of size (Nx1) and a "values" filed of size (NxM). "time" is a column vector of size

Nx1 and "values" is a matrix of size "N*M". "time" filed can only be of Real type, whereas the "values" field can be , , , , , , , and .

- **Interpolation method:** Variables read by Xcos are data values read at discrete instants given by the time field. This option causes the block to interpolate at time steps for which no corresponding workspace data exists. There are four interpolation methods available.
  - **0: "Zero order method"**. This method generates a piecewise constant signal. i.e., for $t_i \leq t < t_{i+1}$, $y(t) = y_i$. This method is available for all data types.
  - **1: "Linear method"**. This method generates a piecewise linear signal, i.e., for $t_i \leq t < t_{i+1}$, $y(t) = y_i + (t - t_i)\frac{y_{i+1} - y_i}{t_{i+1} - t_i}$. For data types other than double and complex, the linear interpolation can be used, but the final output will be computed by casting interpolation result into the original data type.
  - **2:"NATURAL method"**. This cubic spline is computed by using the following conditions (considering $n$ points $x_1, \ldots, x_n$): . This method is only available for Real and complex data types.
  - **3:"NOT_A_KNOT method"**. The cubic spline is computed by using the following conditions (considering $n$ points $x_1, \ldots, x_n$): . This method is only available for Real and complex data types.
- **Enable zero crossing(0:No, 1:Yes)?:** Enables zero crossing detection. When and interpolation methods are chosen, the output signal will be discontinuous at data time instants. These possible discontinuities may cause problem for the numerical solver. In order to perform a reliable numerical integration, the zero crossing option is used. If output of the block affects data used by the numerical solver, at discontinuous points, a discrete event is generated and the numerical solver is cold restarted. The discrete event is also generated at the $t_1$ and $t_n$ for other interpolating methods.
- **Output at end(0:Zero, 1:Hold, 2:Repeat):** This option is for selecting method for generating output after the last time point for which data is available from the workspace.
  - **0 ("Zero"):** The output is set to zero.
  - **1 ("Hold"):** The output is hold.
  - **2 ("Repeat"):** The output is repeated from workspace.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

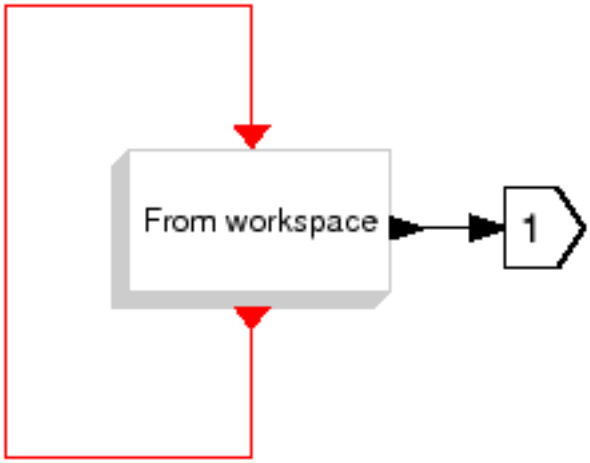
**- port 1 : size [-1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/FROMWSB.sci

## Compiled Super Block content




## See also

- TOWS_c - Data to Scilab worspace

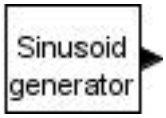
## Authors

**Masoud Najafi** - INRIA

# Nome

GENSIN_f — Sin generator

## Block Screenshot




## Contents


- Sin generator
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

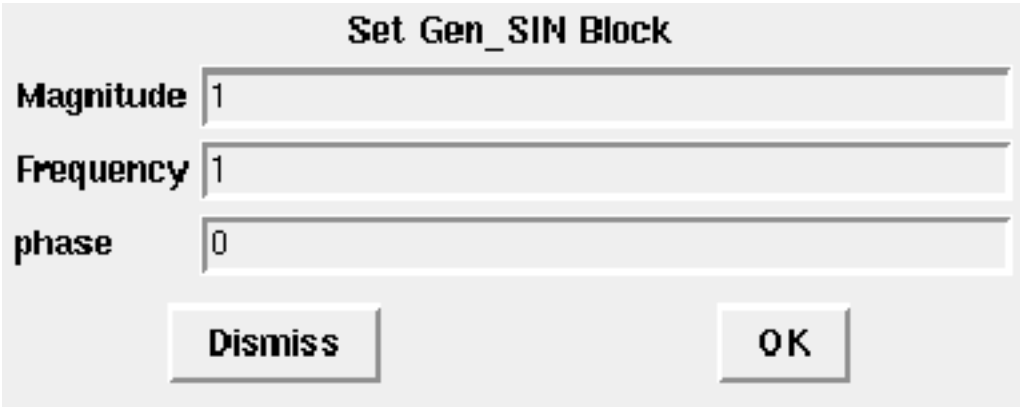

## Palette

- Sources palette

## Description

This block is a sine wave generator:

## Dialog box




- **Magnitude**

  a scalar. The magnitude M.

  Properties : Type 'vec' of size 1.

- **Frequency**

  a scalar. The frequency F.

  Properties : Type 'vec' of size 1.

- **phase**

  a scalar. The phase P.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *gensin*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/GENSIN_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/gensin.f (Type 0)

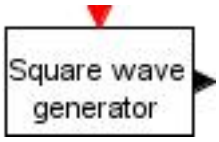
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

GENSQR_f — Square wave generator

## Block Screenshot




## Contents


- Square wave generator
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

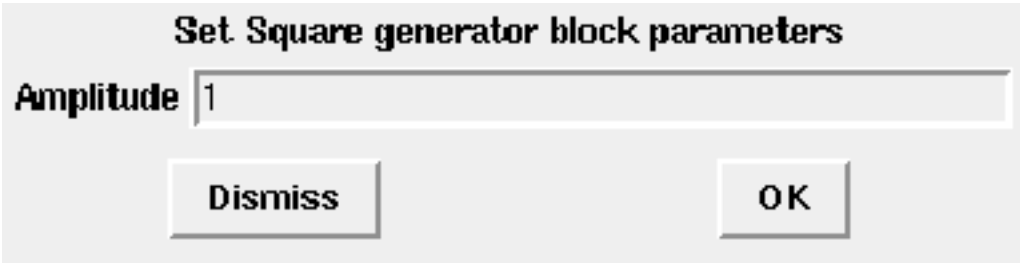

## Palette

- Sources palette

## Description

This block is a square wave generator: output takes values -M and M . Every time an event is received on the input event port, the output switches from -M to M , or M to -M .

## Dialog box




- **Amplitude**

  a scalar $M$ .

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no

- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *gensqr*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/GENSQR_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/gensqr.f (Type 0)

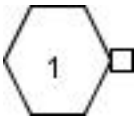
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

INIMPL_f — Input implicit port

## Block Screenshot




## Contents

- Input implicit port
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

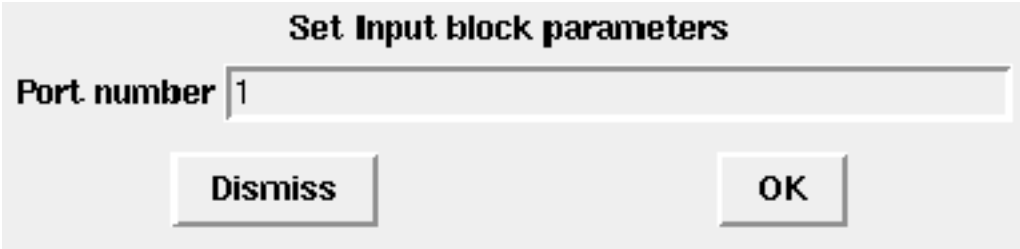
## Palette

- Sources palette

## Description

Inport blocks are the links from outside a system into the system.

## Dialog box




- **Port number**

  Specify the port number of the Inport block.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no

- **mode:** no
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *inimpl*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/INIMPL_f.sci

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

Modulo_Count — Modulo counter

## Block Screenshot

## Contents


- Modulo counter
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

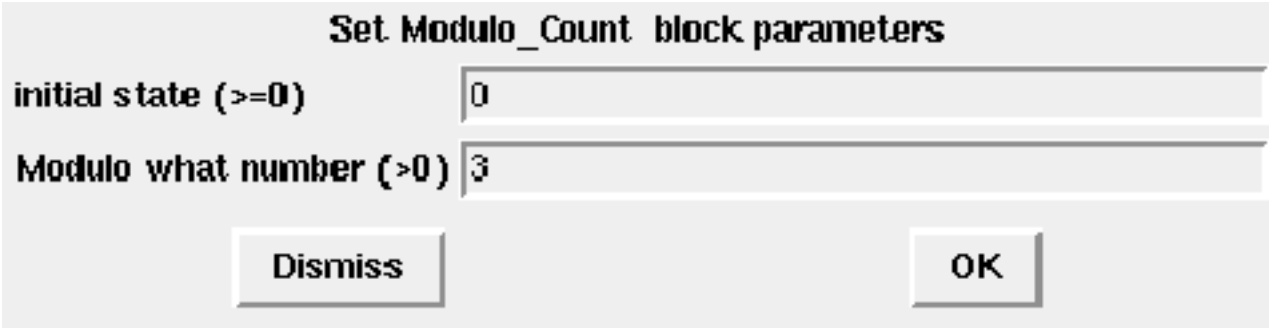

## Palette

- Sources palette

## Description

It is a discrete block. The block outputs a periodic scalar signal having a waveform that the user specifies.

## Dialog box




- **initial state**

  A scalar initial discrete state.

  Properties : Type 'vec' of size 1

- **Modulo what number**

Number of required discrete signals.

Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *modulo_count*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/Modulo_Count.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/modulo_count.c (Type 4)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

RAMP — Ramp

## Block Screenshot

## Contents


- Ramp
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

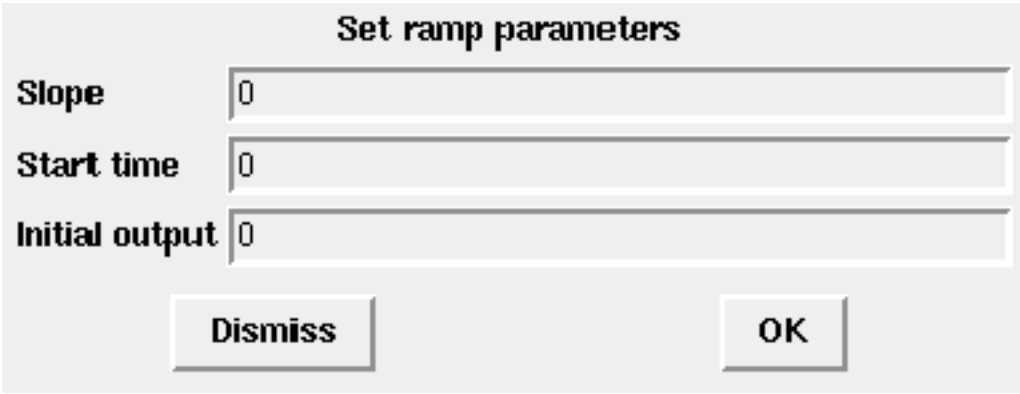

## Palette

- Sources palette

## Description

The Ramp block generates a signal that starts at a specified time and value and changes by a specified rate. The block's **Slope , Start time** and **Initial output** parameters determine the characteristics of the output signal. All must have the consistent dimensions after scalar expansion.

## Dialog box




- **Slope**

  The rate of change of the generated signal.

  Properties : Type 'vec' of size 1.

- **Start time**

  The time at which the signal begins to be generated.

  Properties : Type 'vec' of size 1.

- **Initial output**

  The initial value of the signal.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** yes
- **mode:** yes
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *ramp*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/RAMP.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/ramp.c (Type 4)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

RAND_m — Random generator

## Block Screenshot

## Contents


- Random generator
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

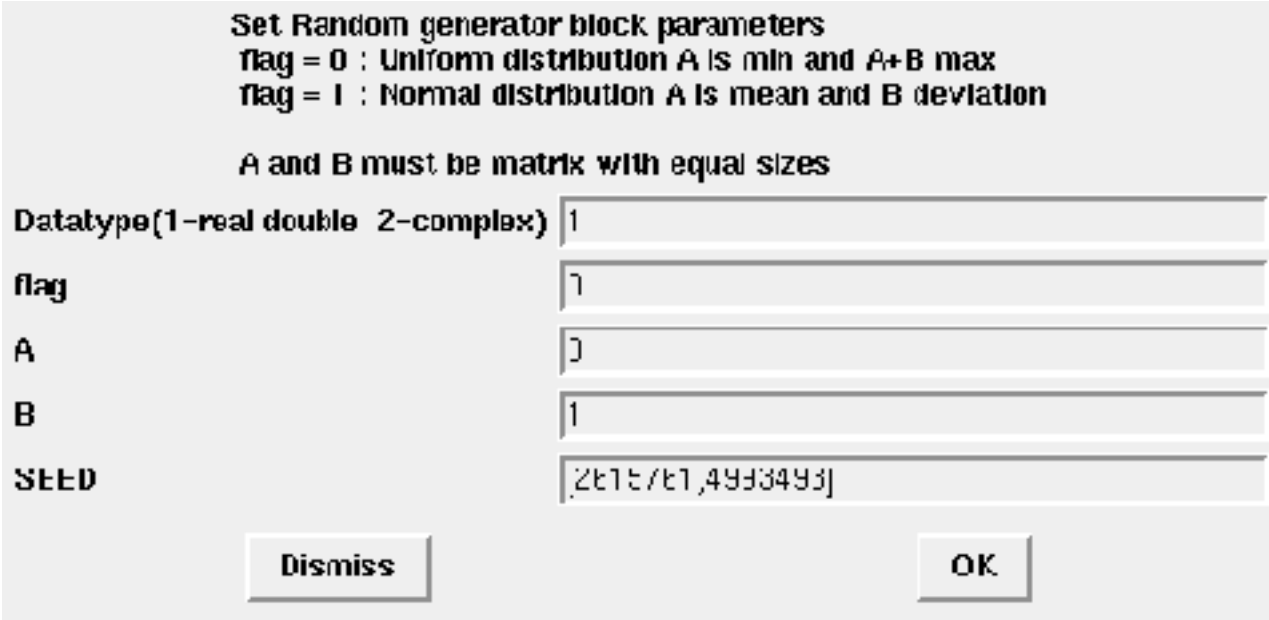
- Sources palette

## Description

This block is a random wave generator: each output component takes piecewise constant random values. Every time an event is received on the input event port, the outputs take new independent random values.

Output port size is given by the size of **A** and **B** matrices.

## Dialog box




- **Datatype(1=real double 2=complex)**

  It indicates the type of the output. It support only the two types double (1) and complex (2). If we input another entry in this label Scicos will print the message "Datatype is not supported".

  Properties : Type 'vec' of size 1.

- **flag**

  0 or 1.

  0 for uniform distribution on [A,A+B].

  1 for normal distribution.

  Properties : Type 'vec' of size 1.

- **A**

  matrix

  Properties : Type 'mat' of size [-1,-2].

- **B**

  matrix

  Properties : Type 'mat' of size [-1,-2].

- **seed**

  matrix

  Seed value for a sequence of random number.

  First number is for the real part and the second for the imaginary part.

  Properties : Type 'mat' of size [1,2].

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *rndblk_m*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/RAND_m.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/rndblk_m.c
- SCI/modules/scicos_blocks/src/c/rndblkz_m.c

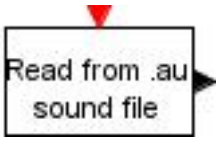
## Authors


- **Fady Nassif** INRIA
- **Ramine Nikoukhah** INRIA

# Nome

READAU_f — Read AU sound file

## Block Screenshot




## Contents


- Read AU sound file
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

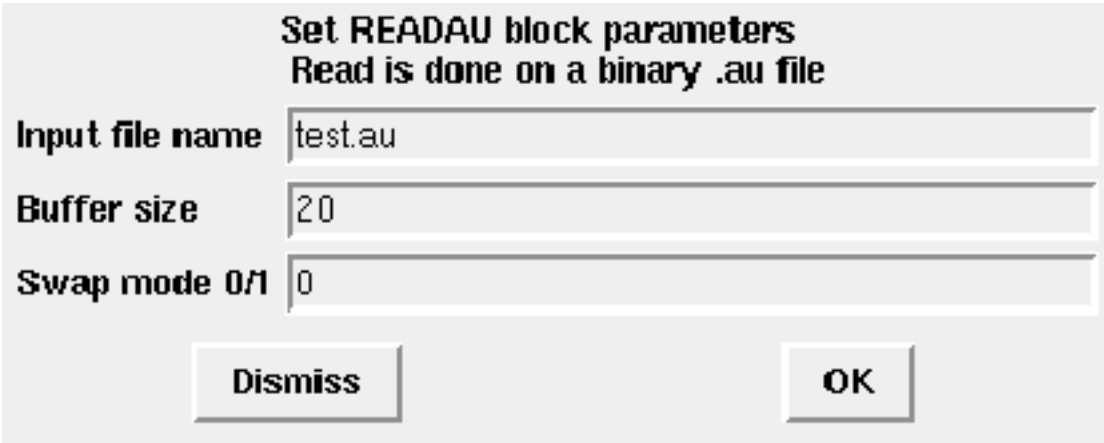
- Sources palette

## Description

Loads a sound file specified by the string *.au file, returning the sampled data in y. The .au extension is appended if no extension is given. Amplitude values are in the range [-1,+1]. auread supports multichannel data in the following formats:

- 8-bit mu-law
- 8-, 16-, and 32-bit linear
- Floating-point

## Dialog box



- **Input file name**

  a character string defining the path of the file.

  Properties : Type 'str' of size 1

- **Buffer size**

  To improve efficiency it is possible to buffer the input data. Read on the file is only done after each Buffer size call to the block.

  Properties : Type 'vec' of size 1

- **Swap mode 0/1**

  With**Swap mode=1** the file is supposed to be coded in "little endian IEEE format" and data are swaped if necessary to match the IEEE format of the processor. If**Swap mode=0** then automatic bytes swap is disabled.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *readau*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/READAU_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/readau.c (Type 2)

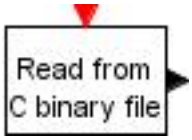
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

READC_f — Read binary data

## Block Screenshot




## Contents


- Read binary data
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

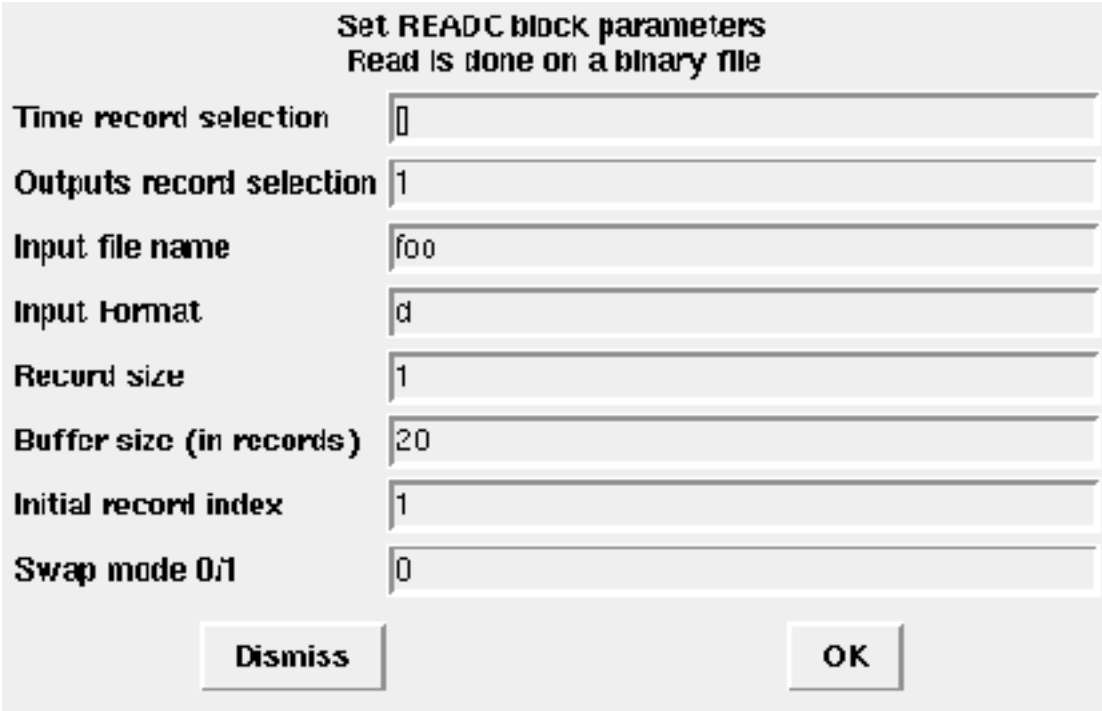

## Palette

- Sources palette

## Description

This block allows user to read data in a C file. **Output record selection** and **Time record Selection** allows the user to select data among file records. Each call to the block advance one record in the file.

## Dialog box




- **Time record selection**

  an empty matrix or a positive integer.

  If an integer i is given the i th element of the read record is assumed to be the date of the output event.

  If empty no output event exists.

  Properties : Type 'vec' of size -1

- **Outputs record selection**

  a vector of positive integer.

  $[k_1, \cdots, k_n]$ ,The $k_i$ th element of the read record gives the value of $i^{th}$ output.

  Properties : Type 'vec' of size -1

- **Input file name**

  a character string defining the path of the file.

  Properties : Type 'str' of size 1

- **Input Format**

  a character string defining the format to use.

  Properties : Type 'str' of size 1

- **Record size**

The file is supposed to be formed by a sequence of data with same format.

These data are organized in a sequence of record each of them containing Record size data.

Properties : Type 'vec' of size 1

- **Buffer size**

  To improve efficiency it is possible to buffer the input data. Read on the file is only done after each Buffer size call to the block.

  Properties : Type 'vec' of size 1

- **Initial record index**

  a scalar. This fixes the first record of the file to use.

  Properties : Type 'vec' of size 1

- **Swap mode 0/1**

  With**Swap mode=1** the file is supposed to be coded in "little endian IEEE format" and data are swaped if necessary to match the IEEE format of the processor. If**Swap mode=0** then automatic bytes swap is disabled.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *readc*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/READC_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/readc.c (Type 2)

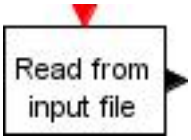
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

RFILE_f — Read from file

## Block Screenshot




## Contents


- Read from file
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

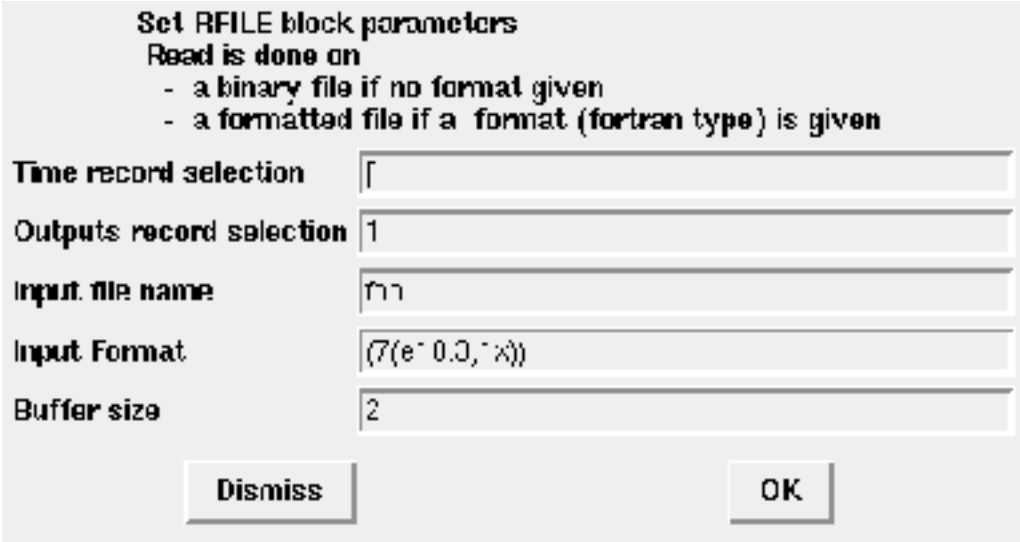
- Sources

## Description

This block allows user to read data in a file, in formatted or binary mode. **Output record selection** and **Time record selection** allows the user to select data among file records. Each call to the block advance one record in the file.

The pair block is .

## Dialog box



- **Time record selection**

  an empty matrix or a positive integer.

  If an integer $i$ is given the $i$ th element of the read record is assumed to be the date of the output event. If empty no output event exists.

  Properties : Type 'vec' of size -1.

- **Outputs record selection**

  a vector of positive integer.

  $[k_1, \cdots, k_n]$ $k_i$. The th element of the read record gives the value of i th output.

  Properties : Type 'vec' of size -1.

- **Input file name**

  a character string defining the path of the file.

  Properties : Type 'str' of size 1.

- **Input Format**

  a character string defining the Fortran format to use or nothing for an unformatted (binary) write. If given, the format must began by a left parenthesis and end by a right parenthesis. example: $(e10.3)$.

  Properties : Type 'str' of size 1.

- **Buffer size**

  To improve efficiency it is possible to buffer the input data. read on the file is only done after each**Buffer size** call to the block.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0

- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *readf*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/RFILE_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/readf.f (Type 0)

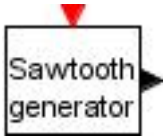
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

SAWTOOTH_f — Sawtooth generator

## Block Screenshot




## Contents


- Sawtooth generator
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Sources palette

## Description

This block is a sawtooth wave generator: output is from to where and denote the times of two successive input events.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** yes
- **object discrete-time state:** no
- **name of computational function:** *sawtth*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/SAWTOOTH_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/sawtth.f (Type 0)

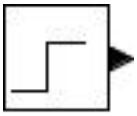
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

STEP_FUNCTION — Step function generator

## Block Screenshot



## Contents


- Step function generator
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “Authors”

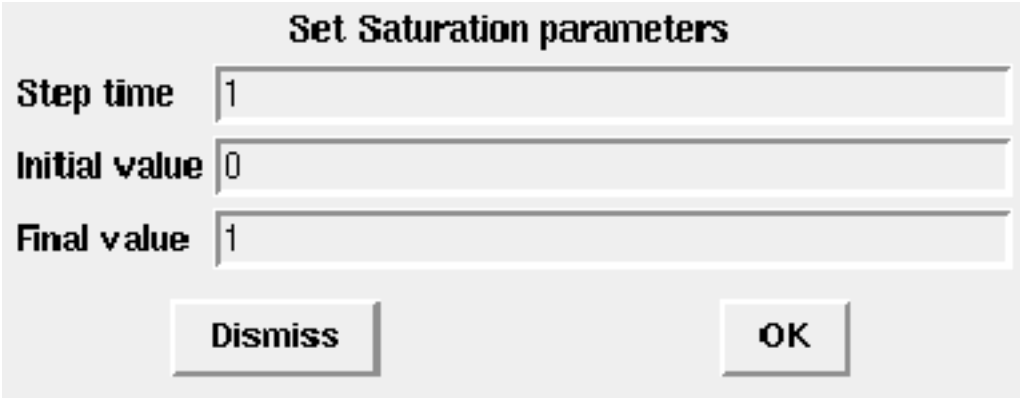

## Palette

- Sources palette

## Description

The Step block provides a step between two definable levels at a specified time. If the simulation time is less than the Step time parameter value, the block's output is the Initial value parameter value. For simulation time greater than or equal to the Step time, the output is the Final value parameter value.

## Dialog box




- **Step time**

  The time, in seconds, when the output jumps from the Initial value parameter to the Final value parameter.

  Properties : Type 'vec' of size 1.

- **Initial value**

  The block output until the simulation time reaches the Step time parameter.

  Properties : Type 'vec' of size -1.

- **Final value**

  The block output when the simulation time reaches and exceeds the Step time parameter.

  Properties : Type 'vec' of size -1.

## Default properties

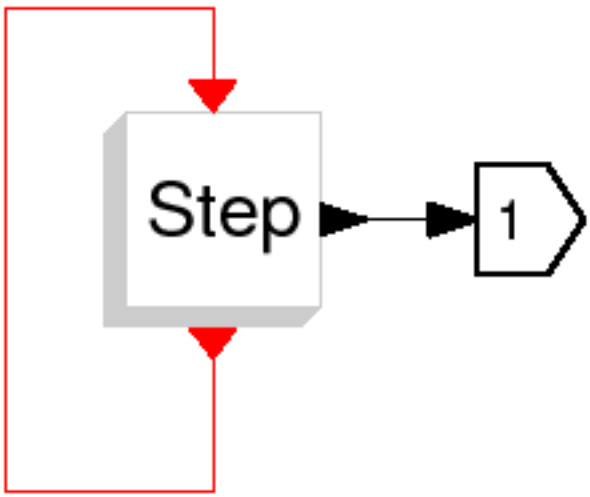
- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/STEP_FUNCTION.sci

## Compiled Super Block content



## Authors

**Ramine Nikoukhah** - INRIA

# Nome

SampleCLK — Sample Time Clock

## Block Screenshot

## Contents


- Sample Time Clock
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “See also”
  - “Authors”


## Palette

- Sources palette

## Description

The difference between the SampleCLK and the CLOCK_c is that all the SampleCLK blocks in our diagram are synchronous. The synchronism is done due to two different methods of computation in the compilation phase.

The first method consists of computing a clock that is faster than all the SampleCLK connected to a counter which activate the event select block.

The clock is calculated due to the following rule.

If all the blocks have the same offset then the frequency of the clock is the gcd of the sample time, and the offset of the clock is equal to the offset.

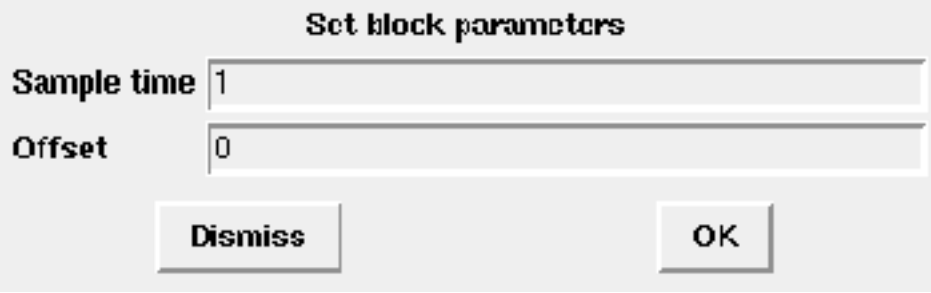
If the offsets are different, then the frequency of the clock is the gcd of the sample time and the offset, and the offset of the clock is equal to 0.

The Counter counts from one to the least common multiple of the sample time (lcm).

The number of outputs of the ESELECT_f block is equal to the lcm.

The second method uses the Multifrequency block it generates events only for specific time. Events in this method are not periodically generated as in the first one.

## Dialog box




- **Sample time**

  The Sample time value.

  Properties : Type 'vec' of size 1.

- **Offset**

  The offset value.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sampleclk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/SampleCLK.sci

## See also

- CLOCK_c - Activation clock
- Counter - Counter
- ESELECT_f - Synchronous block Event-Select
- M_freq - Multiple Frequencies

## Authors

**Fady NASSIF** - INRIA

# Nome

Sigbuilder — Signal creator/generator

## Block Screenshot

## Contents

- Signal creator/generator
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Compiled Super Block content
  - “Authors”

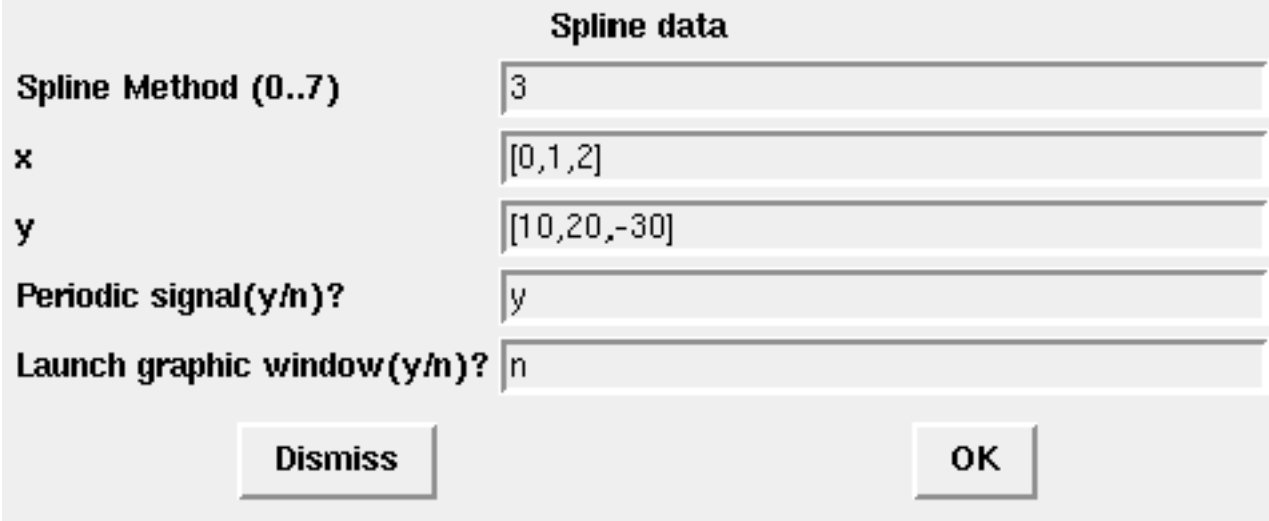
## Palette

- Sources palette

## Description

The Signal Builder block is a superblock containing a block whose output event port is connected to its input event port. This event feedback gives the possibility to generate events at discontinuous point of the signal. The generated events automatically restart the numerical solver and avoids numerical problems. The generated event is also made available to the user for possible use. Remind that if higher interpolation methods are used, the events are generated only at the beginning and at the end of the signal.

## Dialog box




The parameters of Sigbuilder block is the same as that of block.

## Default properties

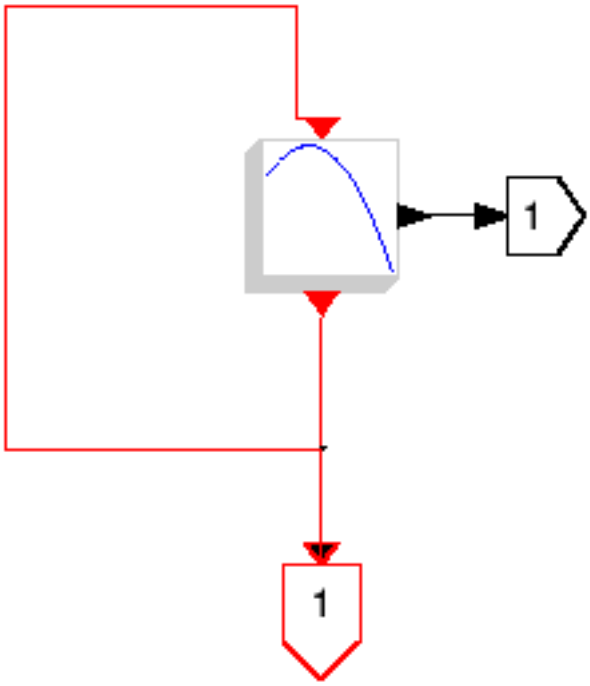
- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [-1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *csuper*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/Sigbuilder.sci

## Compiled Super Block content




## Authors

- Masoud Najafi, INRIA

# Nome

TIME_f — Time

## Block Screenshot

## Contents


- Time
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Sources palette

## Description

This block is a time generator.

The unique regular output is the current time.

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no

- **name of computational function:** *timblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/TIME_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/timblk.f (Type 0)

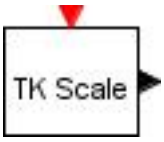
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

TKSCALE — Adjust constant value with a tk widget

## Block Screenshot




## Contents

- Adjust constant value with a tk widget
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

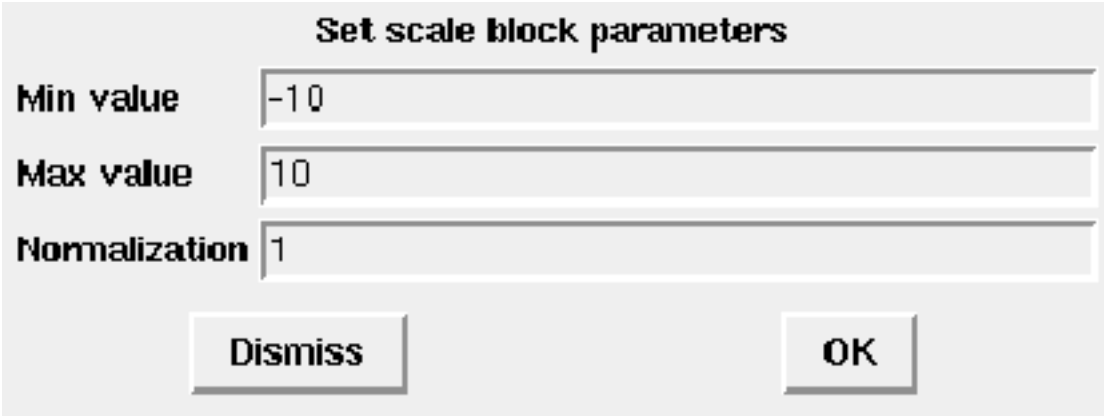
## Palette

- Sources palette

## Description

This source block generates a piecewise constant signal the value of which can be set interactively using a TK widget in the course of the simulation. The output value is the value set by the slider (an integer between Min value and Max value) divided by the Normalization factor. Increasing proportionaly all three block parameters does not change the output range, but it does increase precision.

## Dialog box




- **Min value**

An integer specifying the min value in the range of the scale.

Properties : Type 'vec' of size 1.

- **Max value**

  An integer specifying the max value in the range of the scale.

  Properties : Type 'vec' of size 1.

- **Normalization**

  The output of the block is the integer value specified by the slider (an integer between Min value and the Max value) divided by this Normalization factor.

  Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 1
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *tkscaleblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Sources/TKSCALE.sci

## Computational function

- SCI/modules/scicos/macros/scicos_scicos/tkscaleblk.sci (Type 5)

## Authors

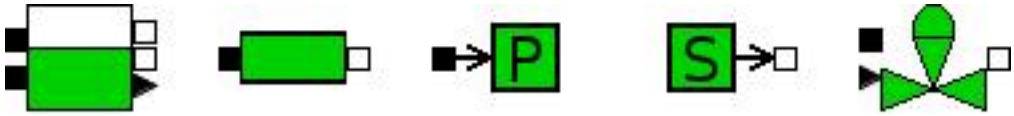
**Ramine Nikoukhah** - INRIA

# 19. Thermohydraulics palette

# Nome

ThermoHydraulics_pal — Thermal-Hydraulics toolbox

## Block Screenshot




## Module

- xcos

## Description

Thermal-Hydraulics toolbox contains some basic thermal-hydraulic components such as pressure source, pipe, control valves, etc.

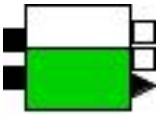
## Blocks

- Bache - Thermal-hydraulic tank (reservoir)
- PerteDP - Thermal-hydraulic pipe
- PuitsP - Thermal-hydraulic drain (well)
- SourceP - Thermal-hydraulic constant pressure source
- VanneReglante - Thermal-hydraulic control valve

# Nome

Bache — Thermal-hydraulic tank (reservoir)

## Block Screenshot



## Contents

- Thermal-hydraulic tank (reservoir)
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model

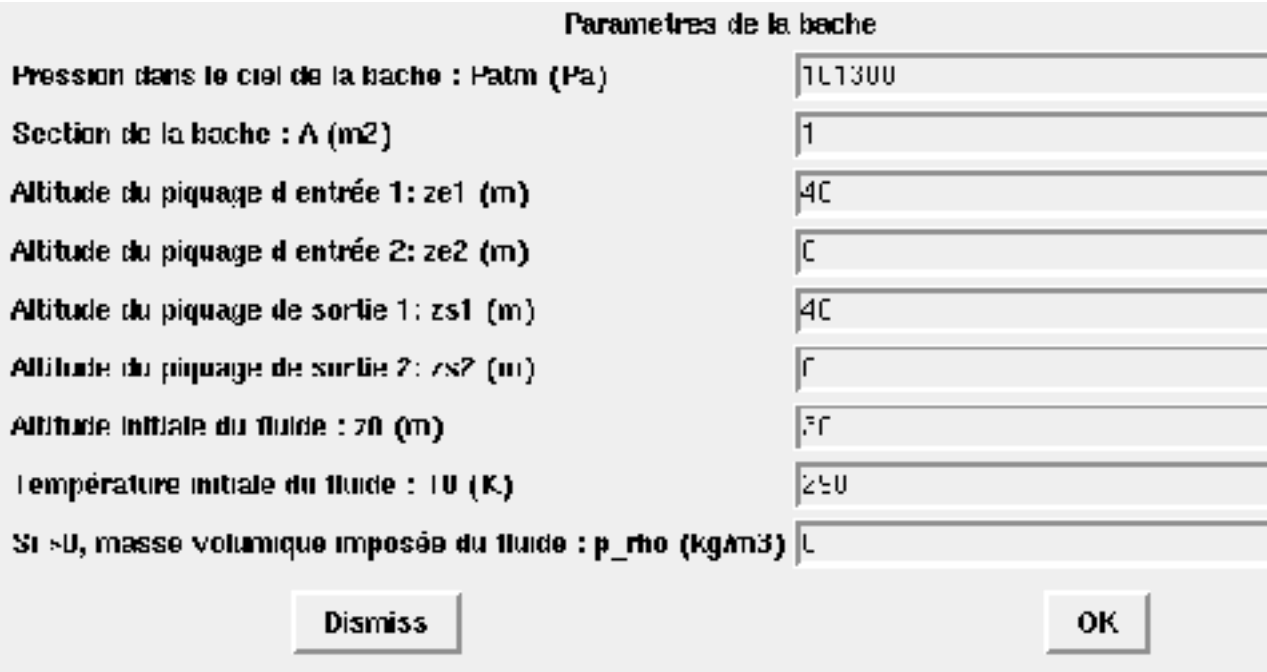
## Palette

- Thermal-Hydraulics palette

## Description

The **Bache** block represents a thermal-hydraulic tank or reservoir. This block has two inlets and two outlets whose altitudes can be changed by the user. Conventionally, for input ports (black ports) flow direction is positive when fluid flows into the tank. On the other hand, for output ports (white ports) flow direction is positive when fluid flows out of the tank. The user can set the surface area of the tank, the initial temperature and initial level of the fluid in the tank. If an input or output port is left unused, it should be blocked by a **stopper** block.

## Dialog box




- **Pression dans le ciel de la bache : Patm (Pa)**

  Atmospheric pressure inside the tank.

  Properties : Type 'vec' of size -1.

- **Section de la bache : A (m2)**

  Surface area of the tank.

  Properties : Type 'vec' of size -1.

- **Altitude du piquage d entrée 1: ze1 (m)**

  Altitude of the first input port

  Properties : Type 'vec' of size -1.

- **Altitude du piquage d entrée 2: ze2 (m)**

  Altitude of the second input port

  Properties : Type 'vec' of size -1.

- **Altitude du piquage de sortie 1: zs1 (m)**

  Altitude of the first output port.

  Properties : Type 'vec' of size -1.

- **Altitude du piquage de sortie 2: zs2 (m)**

  Altitude of the second output port.

  Properties : Type 'vec' of size -1.

- **Altitude initiale du fluide : z0 (m)**

  Initial fluid level in the tank.

  Properties : Type 'vec' of size -1.

- **Température initiale du fluide : T0 (K)**

  Temperature of fluid in the tank

  Properties : Type 'vec' of size -1.

- **Si 0, masse volumique imposée du fluide : p_rho (kg/m3)**

  Density of fluid

  Properties : Type 'vec' of size -1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'Ce1'

    Implicit **variable.**
  - **Modelica variable name :** 'Ce2'

    Implicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'Cs1'

    Implicit **variable.**
  - **Modelica variable name :** 'Cs2'

    Implicit **variable.**
  - **Modelica variable name :** 'yNiveau'

    Explicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'Patm'

    **Default value :** 101300

    **Is a state variable :** no.
  - **Modelica parameter name :** 'A'

    **Default value :** 1

    **Is a state variable :** no.
  - **Modelica parameter name :** 'ze1'

    **Default value :** 40

    **Is a state variable :** no.

- **Modelica parameter name :** 'ze2'

  **Default value :** 0

  **Is a state variable :** no.

- **Modelica parameter name :** 'zs1'

  **Default value :** 40

  **Is a state variable :** no.

- **Modelica parameter name :** 'zs2'

  **Default value :** 0

  **Is a state variable :** no.

- **Modelica parameter name :** 'z0'

  **Default value :** 30

  **Is a state variable :** no.

- **Modelica parameter name :** 'T0'

  **Default value :** 290

  **Is a state variable :** no.

- **Modelica parameter name :** 'p_rho'

  **Default value :** 0

  **Is a state variable :** no.

- **File name of the model :** Bache

## Interfacing function

- SCI/modules/scicos_blocks/macros/Hydraulics/Bache.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Hydraulics/Bache.mo

# Nome

PerteDP — Thermal-hydraulic pipe

## Block Screenshot

## Contents

- Thermal-hydraulic pipe
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model

## Palette

- Thermal-Hydraulics palette

## Description

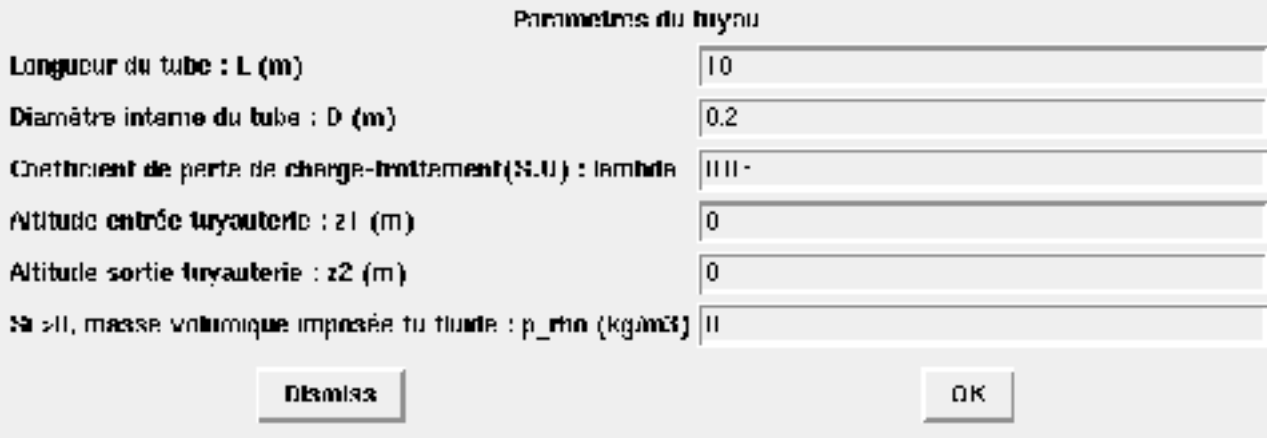
A **PertDP** block represents a hydraulic pipe with linear resistance losses. This component represents a hydraulic resistance and pressure loss is directly proportional to the flow rate. Conventionally, the flow direction is the positive when fluid flows from the black port to the white port. The pressure loss is obtained

with

$$P_{loss} = P_{black} - P_{white}$$

. The key parameters of this block are the pipes' length, the pipe's diameter, inlet and outlet altitudes, and some other thermal-hydraulic coefficients.

## Dialog box




- **Longueur du tube : L (m)**

  Length of pipe

  Properties : Type 'vec' of size -1.

- **Diamètre interne du tube : D (m)**

  Pipe diameter

  Properties : Type 'vec' of size -1.

- **Coefficient de perte de charge-frottement(S.U) : lambda**

  Coefficient of thermohydraluc resistance

  Properties : Type 'vec' of size -1.

- **Altitude entrée tuyauterie : z1 (m)**

  Altitude of the first port (z1)

  Properties : Type 'vec' of size -1.

- **Altitude sortie tuyauterie : z2 (m)**

  Altitude of the second port (z2)

  Properties : Type 'vec' of size -1.

- **Si 0, masse volumique imposée fu fluide : p_rho (kg/m3)**

  Fluid density

  Properties : Type 'vec' of size -1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'C1'

    Implicit **variable.**

- **Outputs :**
  - **Modelica variable name :** 'C2'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'L'

    **Default value :** 10

    **Is a state variable :** no.
  - **Modelica parameter name :** 'D'

    **Default value :** 0.2

    **Is a state variable :** no.
  - **Modelica parameter name :** 'lambda'

    **Default value :** 0.03

    **Is a state variable :** no.
  - **Modelica parameter name :** 'z1'

    **Default value :** 0

    **Is a state variable :** no.
  - **Modelica parameter name :** 'z2'

    **Default value :** 0

    **Is a state variable :** no.
  - **Modelica parameter name :** 'p_rho'

    **Default value :** 0

    **Is a state variable :** no.
- **File name of the model :** PerteDP

## Interfacing function

- SCI/modules/scicos_blocks/macros/Hydraulics/PerteDP.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Hydraulics/PerteDP.mo

# Nome

PuitsP — Thermal-hydraulic drain (well)

## Block Screenshot

## Contents

- Thermal-hydraulic drain (well)
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model

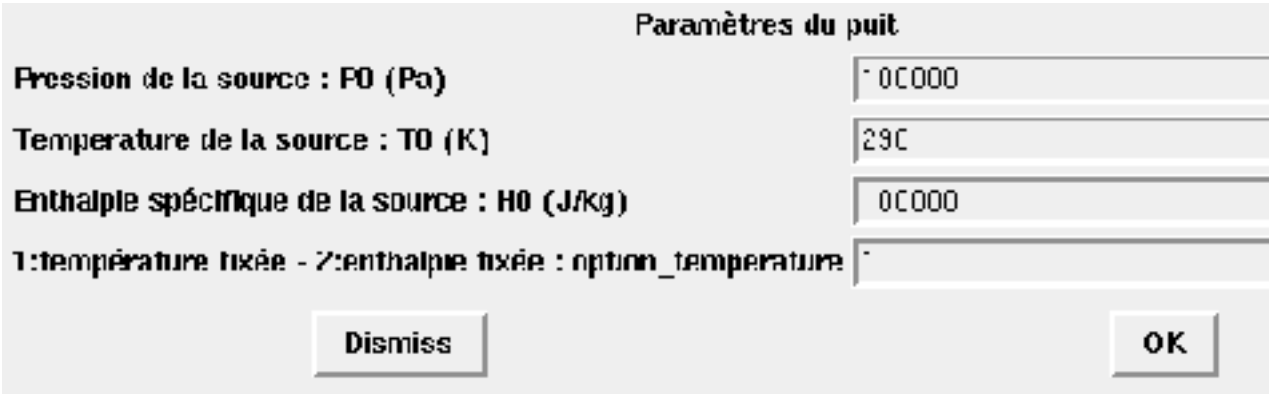
## Palette

- Thermal-Hydraulics palette

## Description

This thermal-hydraulic component represents a thermal-hydraulic constant pressure drain (well). This block is specified with its pressure and temperature. Conventionally, the flow direction is positive when fluid flows into the block.

## Dialog box




- **Pression de la source : P0 (Pa)**

  Pressure of the thermohydraulic source.

  Properties : Type 'vec' of size -1.

- **Temperature de la source : T0 (K)**

Temperature of the thermohydraulic source.

Properties : Type 'vec' of size -1.

- **Enthalpie spécifique de la source : H0 (J/kg)**

  Specific Enthaly of the thermohydraulic source.

  Properties : Type 'vec' of size -1.

- **1:température fixée - 2:enthalpie fixée : option_temperature**

  Temperature option. 1: fixed temperature - 2: fixed enthalpy

  Properties : Type 'vec' of size -1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'C'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'P0'

    **Default value :** 100000

    **Is a state variable :** no.
  - **Modelica parameter name :** 'T0'

    **Default value :** 290

    **Is a state variable :** no.
  - **Modelica parameter name :** 'H0'

    **Default value :** 100000

    **Is a state variable :** no.
  - **Modelica parameter name :** 'option_temperature'

    **Default value :** 1

    **Is a state variable :** no.
- **File name of the model :** Puits

## Interfacing function

- SCI/modules/scicos_blocks/macros/Hydraulics/PuitsP.sci

## Modelica model

- SCI/modules/scicos_blocks/macros/Hydraulics/Puits.mo

# Nome

SourceP — Thermal-hydraulic constant pressure source

## Block Screenshot

## Contents

- Thermal-hydraulic constant pressure source
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model

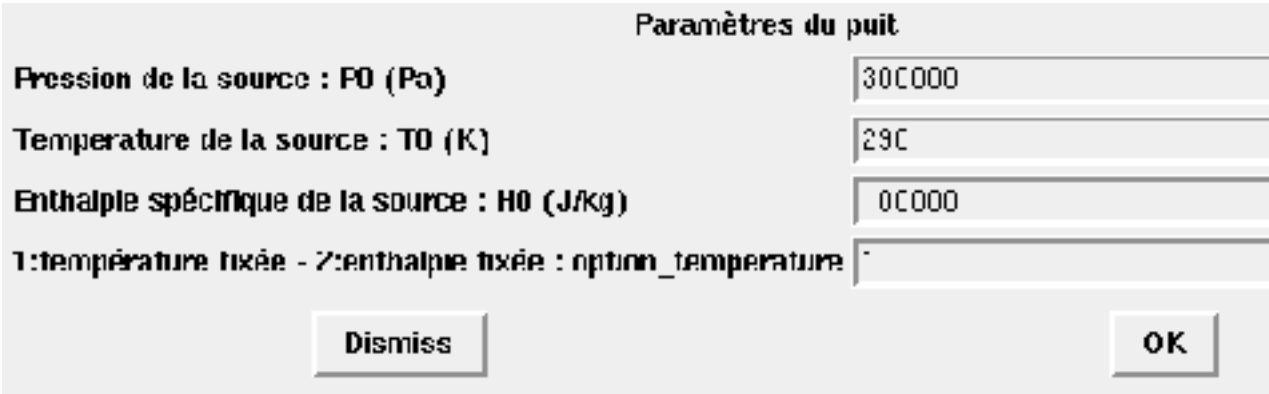
## Palette

- Thermal-Hydraulics palette

## Description

This thermal-hydraulic component represents a thermal-hydraulic constant pressure supply. This block is specified with its output pressure and temperature. Conventionally, the flow direction is positive when the fluid flows out of the block.

## Dialog box




- **Pression de la source : P0 (Pa)**

  Pressure of the thermohydraulic source

  Properties : Type 'vec' of size -1.

- **Temperature de la source : T0 (K)**

Temperature of the thermohydraulic source

Properties : Type 'vec' of size -1.

- **Enthalpie spécifique de la source : H0 (J/kg)**

  Specific enthalpie of the thermohydraulic source

  Properties : Type 'vec' of size -1.

- **1:température fixée - 2:enthalpie fixée : option_temperature**

  Temperature option. 1: fixed temperature - 2: fixed enthalpy

  Properties : Type 'vec' of size -1.

## Default properties

- **Outputs :**
  - **Modelica variable name :** 'C'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'P0'

    **Default value :** 300000

    **Is a state variable :** no.
  - **Modelica parameter name :** 'T0'

    **Default value :** 290

    **Is a state variable :** no.
  - **Modelica parameter name :** 'H0'

    **Default value :** 100000

    **Is a state variable :** no.
  - **Modelica parameter name :** 'option_temperature'

    **Default value :** 1

    **Is a state variable :** no.
- **File name of the model :** Source

## Interfacing function

- SCI/modules/scicos_blocks/macros/Hydraulics/SourceP.sci

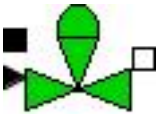
## Modelica model

- SCI/modules/scicos_blocks/macros/Hydraulics/Source.mo

# Nome

VanneReglante — Thermal-hydraulic control valve

## Block Screenshot



## Contents


- Thermal-hydraulic control valve
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - Modelica model


## Palette

- Thermal-Hydraulics palette

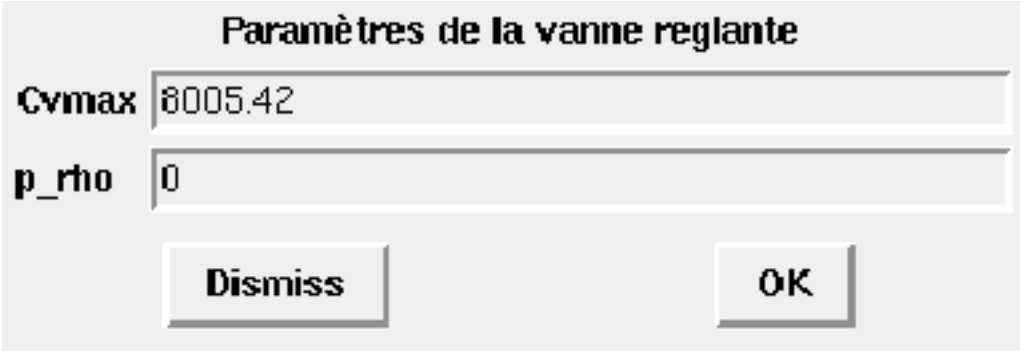
## Description

The **VanneReglante** block represents a variable orifice control valve. The flow rate through the valve is proportional to the valve opening, ,

$$\Delta P \times h \times |h| = k \times Q \times |Q|$$

where is the valve opening, is the pressure difference, and is the flow rate. This model is only used for the laminar flow regimes. is a constant depending on the valve geometry and mass density of fluid.

## Dialog box




- **Cvmax**

Cvmax (maximum opening of the valve)

Properties : Type 'vec' of size -1.

- **p_rho**

  Fluid density

  Properties : Type 'vec' of size -1.

## Default properties

- **Inputs :**
  - **Modelica variable name :** 'C1'

    Implicit **variable.**
  - **Modelica variable name :** 'Ouv'

    Explicit **variable.**
- **Outputs :**
  - **Modelica variable name :** 'C2'

    Implicit **variable.**
- **Parameters :**
  - **Modelica parameter name :** 'Cvmax'

    **Default value :** 8005.42

    **Is a state variable :** no.
  - **Modelica parameter name :** 'p_rho'

    **Default value :** 0

    **Is a state variable :** no.
- **File name of the model :** VanneReglante

## Interfacing function

- SCI/modules/scicos_blocks/macros/Hydraulics/VanneReglante.sci

## Modelica model

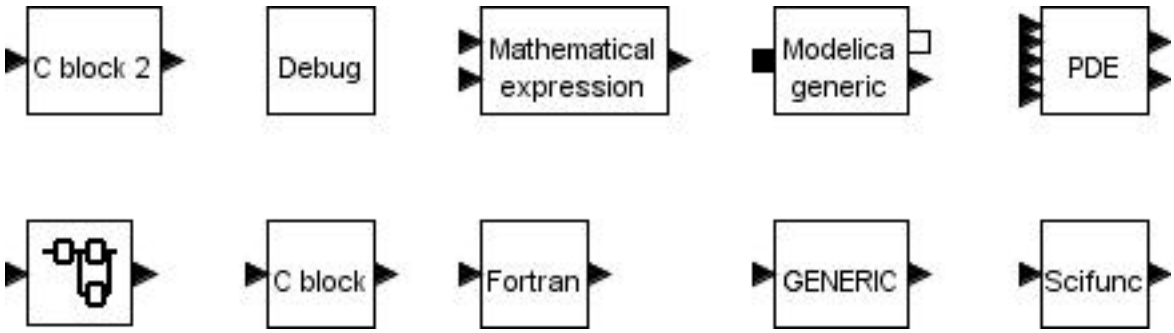
- SCI/modules/scicos_blocks/macros/Hydraulics/VanneReglante.mo

# 20. User defined functions palette

# Nome

Userdefinedfunctions_pal — User defined functions palette

## Block Screenshot




## Module

- xcos

## Description

The user defined function contains blocks that allow you to model the component behaviour. The output is expressed as a function of the input.

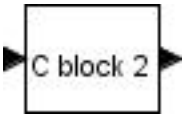
## Blocks

- c_block — C file function
- CBLOCK — New C
- EXPRESSION — Mathematical expression
- fortran_block — Fortran
- generic_block3 — Generic block
- MBLOCK — Modelica generic block
- PDE — 1D PDE block
- scifunc_block_m — Scilab function block
- SUPER_f — Super block
- TEXT_f — Text

# Nome

CBLOCK — New C

## Block Screenshot




## Contents

- New C
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

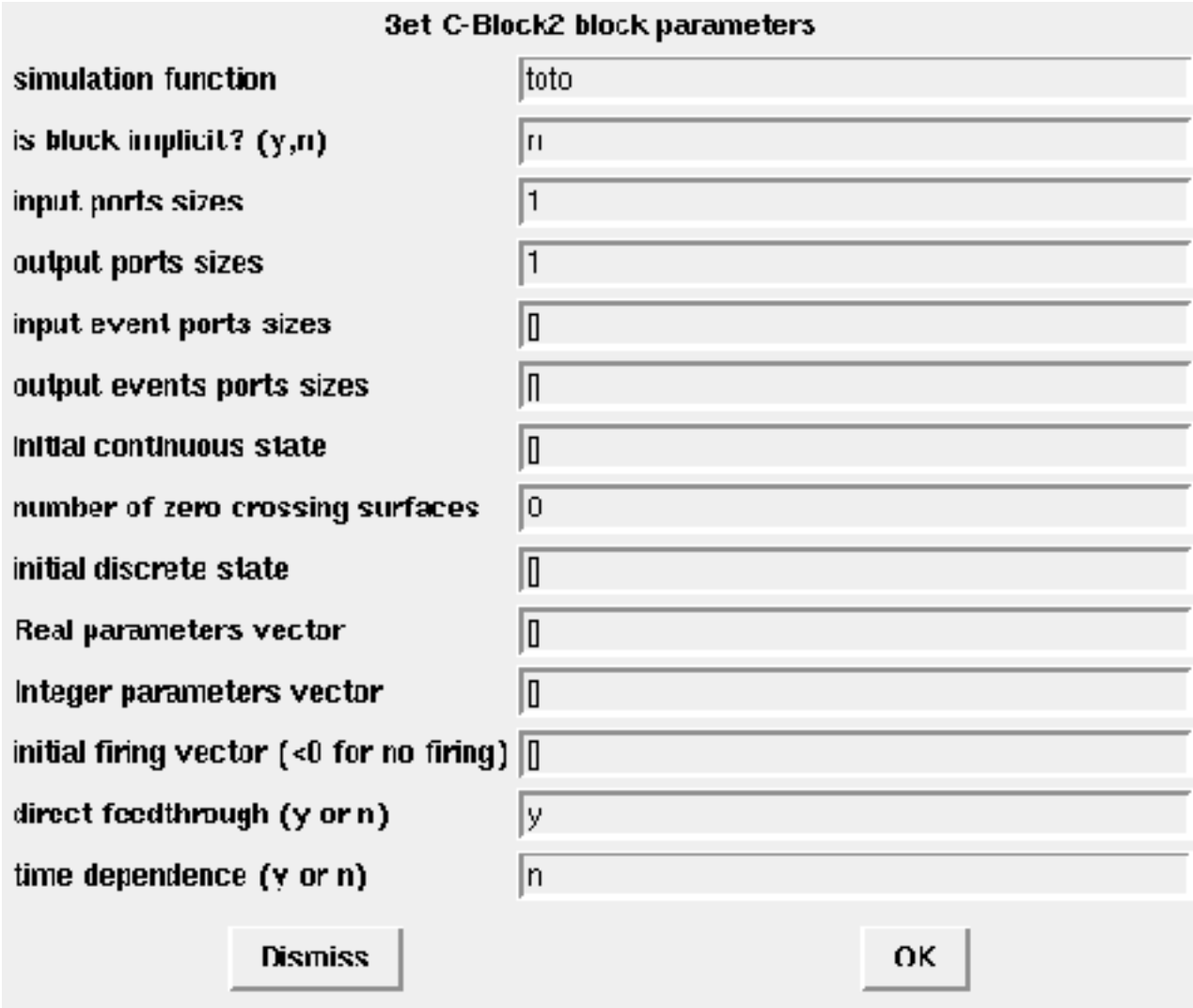
## Palette

- User defined functions palette

## Description

This block creates skeleton of the C-computing function. It also creates library file and object files.

## Dialog box




- **simulation function**

  Name of the function to be generated.

  Properties : Type 'str' of size 1

- **is block implicit?**

  If yes (y) is selected, it calls implicit solver (dasrt) else (n) it calls explicit solver, lsodar.

  Properties : Type 'str' of size 1

- **input ports sizes**

  Number of regular input ports

  Properties : Type 'vec' of size -1

- **output ports sizes**

  Number of regular output ports

  Properties : Type 'vec' of size -1

- **input event ports sizes**

  Number of event input ports

  Properties : Type 'vec' of size -1

- **output events ports sizes**

  Number of event output ports

  Properties : Type 'vec' of size -1

- **initial continuous state**

  Initial Conditions

  Properties : Type 'vec' of size -1

- **number of zero crossing surfaces**

  Select to enable zero crossing detection.

  Properties : Type 'vec' of size 1

- **initial discrete state**

  Initial conditions of the discrete states.

  Properties : Type 'vec' of size -1

- **Real parameters vector**

  Real Parameter vector that the function accepts.

  Properties : Type 'vec' of size -1

- **Integer parameters vector**

  Integer Parameter vector that the function accepts.

  Properties : Type 'vec' of size -1

- **initial firing vector**

  A vector. Size of this vector corresponds to the number of event outputs. The value of the $i^{th}$ entry specifies the time of the preprogrammed event firing on the $i^{th}$ output event port. If less than zero, no event is preprogrammed.

  Properties : Type 'vec' of size 'sum(%6)'

- **direct feedthrough**

  The input to the block at the current time determine the output of the block at the current time. This forces the input to feed through to the output, as if the system were operating at steady-state.

  Properties : Type 'str' of size 1

- **time dependence**

  Create a signal that specifies the time dependence.

  Properties : Type 'str' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:**

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/CBLOCK.sci

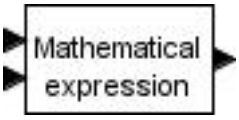
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

EXPRESSION — Mathematical expression

## Block Screenshot




## Contents


- Mathematical expression
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

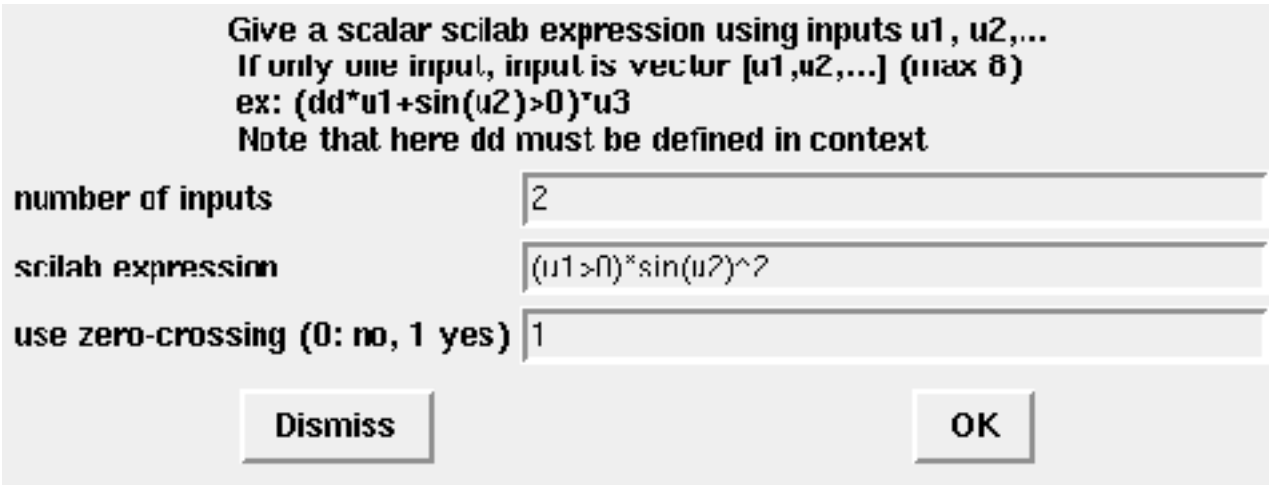

## Palette

- User defined functions palette

## Description

The Expression block applies the specified Scilab functions to its input.

## Dialog box




- **number of inputs**

  Block input can be a scalar or vector.

Properties : Type 'vec' of size 1

- **scilab expression**

  The Scilab expression applied to the input.

  Properties : Type 'vec' of size 1

- **use zero-crossing**

  Select to enable zero crossing detection.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** yes
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *evaluate_expr*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/EXPRESSION.sci

## Computational function

- SCI/modules/scicos_blocks/src/c/evaluate_expr.c (Type 4)

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

MBLOCK — Modelica generic block

## Block Screenshot

## Contents


- Modelica generic block
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

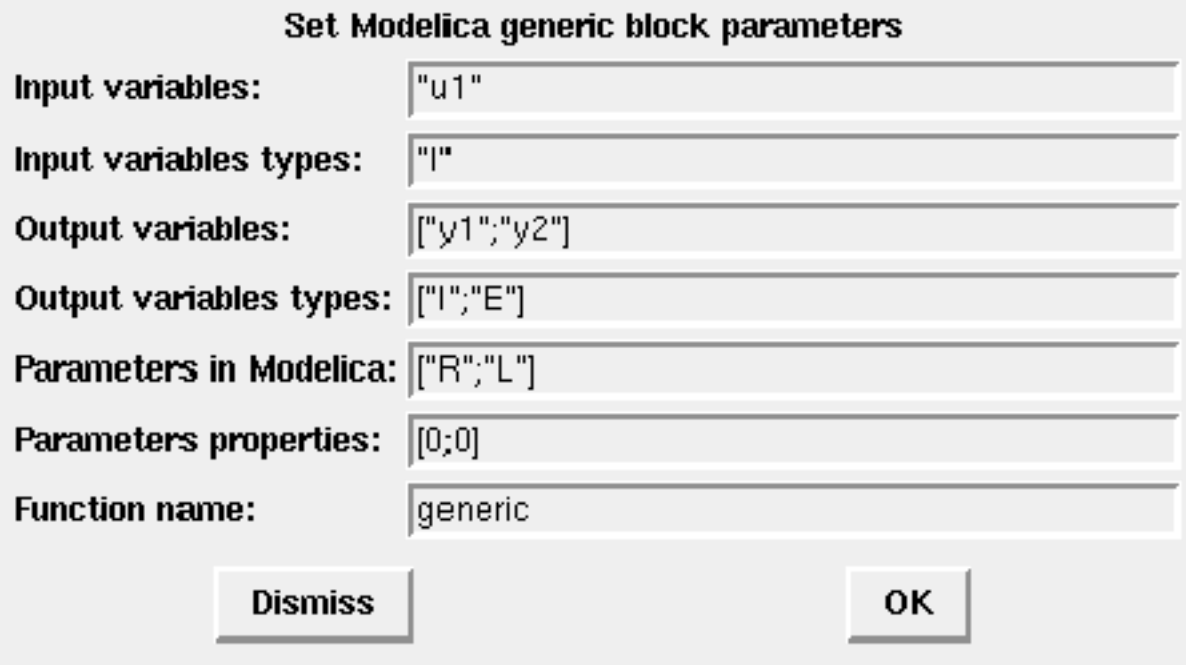

## Palette

- User Defined function palette

## Description

The block "MBlock" provides an easy way to build a xcos block whose behavior is specified by a Modelica program. Using this block, the user will be able to write and compile Modelica programs in xcos without creating any interfacing function. The associated Modelica program of this block can be either given in a file or written in the window opened by the block. In order to link this block to other xcos blocks that may be other Modelica blocks, the types of block ports' as well as their associated variables should be specified.

## Dialog box




- **Input variables**

  In this filed, the ports connected to the left hand side of the block are defined. If the port is an explicit port, it will be an input port. In this case, the variable should be declared in the Modelica program as Real. If the port is an implicit port, the variable desinating this port should be a "connector". Remind that for implicit port, the notion of input and output does not exist and specifying an implicit variable in this filed is just placing the port at the left hend side of the block.

- **Input variables types**

  In this filed, the type of ports are specified, i.e., 'I' for implicit ports and 'E' for explicit ports. The size of the vector of "input variables" and the vector of "input_vector_type" should be equal.

- **Output variables**

  Similar to the input variables vector, the explicit output variables and implicit variables which are displayed at the right hand side of the block are specified in this filed.

- **Output variables types**

  The type of variables given in the Output variable vector are specified, i.e., 'I' for implicit ports and 'E' for explicit ports.

- **Parameters in Modelica**

  The values of parameters declared in the Modelica program can be overloaded. To overload a parameter value, the name of parameters are given in this field and their corresponding values are given in the "parameter values" fields that are displayed in the second dialog box.

- **Parameters properties**

  The type of the Modelica parameters. For that time being, one can parametrize three types of Modelica variable :

  - **0 :** the parameter is set to be a **Modelica parameter** variable (scalar or vector).

- **1 :** the parameter is set to be an **initial condition of Modelica state** variable (scalar or vector).
- **2 :** the parameter is set to be an **initial condition of Modelica state** variable with the property **fixed=true** (scalar or vector).

- **Function name**

  The Modelica class name is specified in this filed. If the Modelica class name is specified without any path or extension, an interactive window is opened and the user can write or edit the Modelica program. This window is opened each time the user clicks on the block. If the Modelica class name is specified with path and '.mo' extension, the compiler looks for the file and if it is found, the file will be compiled, otherwise a window is opened and the user can write the Modelica program. This Modelica file will be saved with the given filename in the specified path. The next time, only input/output characteristics of the block can be changed, and the Modelica file should be edited with another text editor.

- **Parameter values**

  The value of Modelica parameters are given in the "Set parameters values" dialog box. These values that can be scalar or vector, can also be defined in the xcos context. In order to access the xcos context, click on the "Diagram" menu then click on the "Context" submenu. For instance, here is an example of overloading of parameters in a Modelica program. Parameters vector = ['Speed';'Position';"Length"]

  Parameters properties vector = [0;2;1] Speed value = [12.0]

  Position value = [0.0 ; 0.1 ; POS]

  Length value = [13.0 ; 12.1]

## Default properties

- **always active:** yes
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

  **- port 2 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *generic*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/MBLOCK.sci

## Authors

**M. Najafi/A. Layec** - INRIA

# Nome

SUPER_f — Super block

## Block Screenshot

## Contents


- Super block
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Authors”


## Palette

- User defined functions palette

## Description

This block opens up a new Scicos window for editing a new block diagram. This diagram describes the internal functions of the super block.

Super block inputs and outputs (regular or event) are designated by special (input or output) blocks.

Regular input blocks must be numbered from 1 to the number of regular input ports. Regular input ports of the super block are numbered from the top of the block shape to the bottom. Regular output ports must be numbered from 1 to the number of regular output ports. Regular output ports of the super block are numbered from the top of the block shape to the bottom. Event input blocks must be numbered from 1 to the number of event input ports. Event input ports of the super block are numbered from the left of the block shape to the right. Event output ports must be numbered from 1 to the number of event output ports. Event output ports of the super block are numbered from the left of the block shape to the right.

## Default properties

- **always active:** no
- **direct-feedthrough:** no
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **regular outputs:**

**- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *super*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/SUPER_f.sci

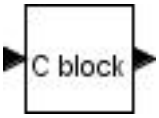
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

c_block — C language

## Block Screenshot




## Contents


- C language
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

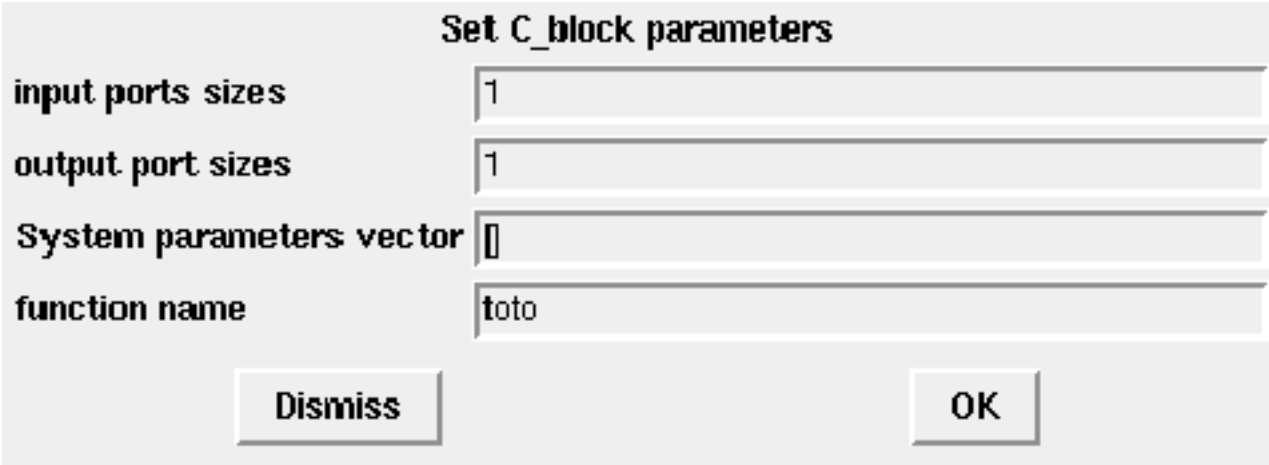

## Palette

- User defined functions palette

## Description

This block creates skeleton of the C computing function. Also it creates library file and object files.

## Dialog box




- **input ports sizes**

  Number of regular input ports.

  Properties : Type 'vec' of size -1.

- **output port sizes**

  Number of regular output ports.

  Properties : Type 'vec' of size -1.

- **System parameters vector**

  Number of parameters that this function accepts.

  Properties : Type 'vec' of size -1.

- **function name**

  Name of the function to be generated.

  Properties : Type 'str' of size -1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:**

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/c_block.sci

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

fortran_block — Fortran

## Block Screenshot

## Contents


- Fortran
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Authors”

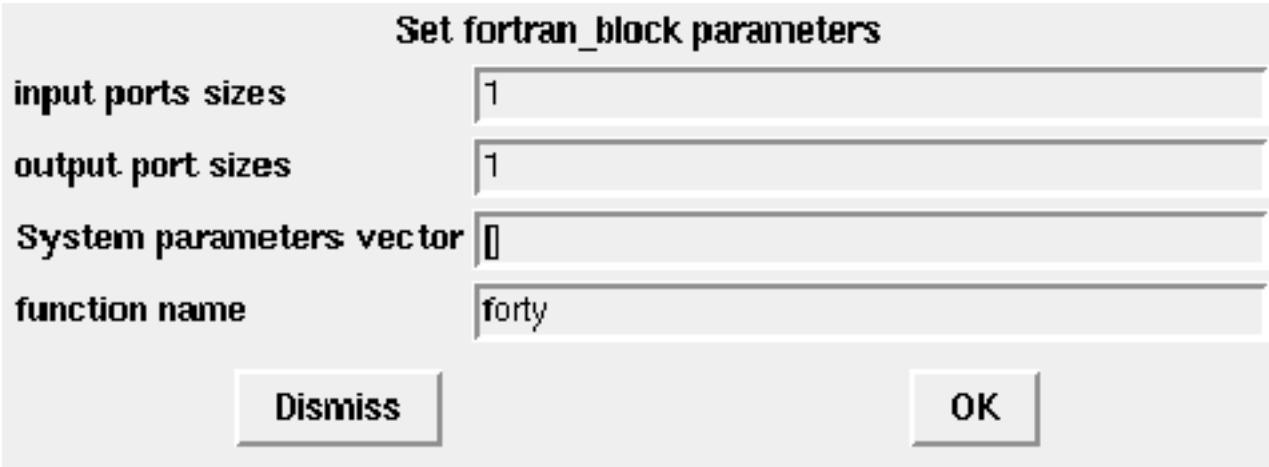

## Palette

- User defined functions palette

## Description

This block creates skeleton of the FORTRAN computing function. Also it creates library file and object files.

## Dialog box




- **input ports sizes**

  Number of regular input ports.

  Properties : Type 'vec' of size -1.

- **output port sizes**

  Number of regular output ports.

  Properties : Type 'vec' of size -1.

- **System parameters vector**

  Number of parameters that this function accepts.

  Properties : Type 'vec' of size -1.

- **function name**

  Name of the function to be generated.

  Properties : Type 'vec' of size -1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:**

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/fortran_block.sci

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

generic_block3 — Generic block

## Block Screenshot

## Contents


- Generic block
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

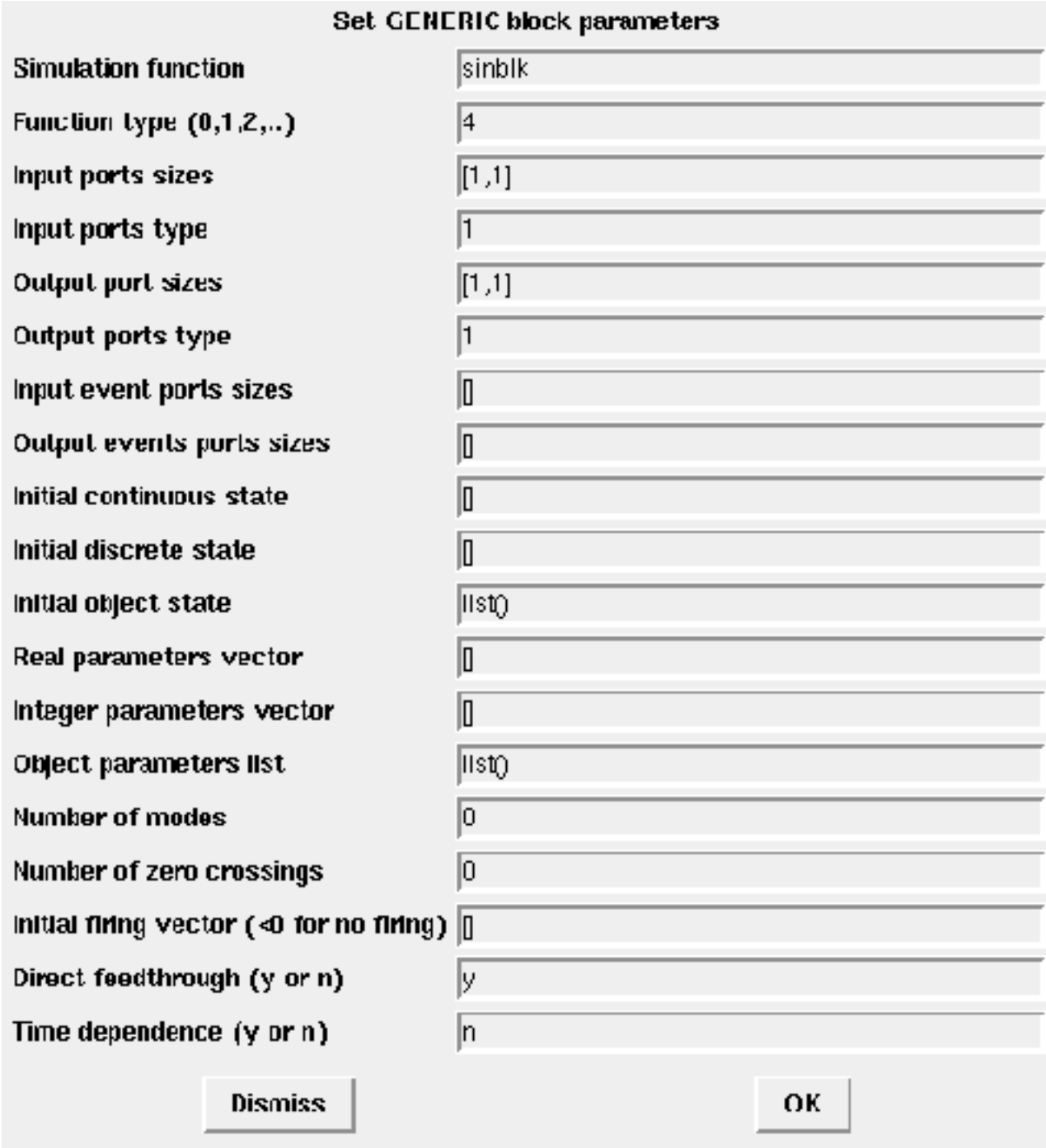

## Palette

- User defined functions palette

## Description

The block provides a generic interfacing function but the computational function needs to be defined separately, either as a Scilab function or a Fortran or a C function. Besides the name of the function, user should specify information such as the type, whether or not the block contains a direct feed-through term. The function realising computational functions of generic blocks of a Scicos diagram must be saved along with the diagram and loaded or dynamically linked before simulation.

## Dialog box




- **Simulation function**

  Name of the function to be loaded.

  Properties : Type 'str' of size 1

- **Function type**

  Type of the computational function supported by Scicos.

  Properties : Type 'vec' of size 1

- **Input ports sizes**

  Number of regular input ports.

  Properties : Type 'mat' of size [-1,2]

- **Input ports type**

  Set the datatype of the regular input ports.

  Properties : Type 'vec' of size -1

- **Iutput port sizes**

  Number of regular input ports.

  Properties : Type 'mat' of size [-1,2]

- **Output ports type**

  Set the datatype of the regular output ports.

  Properties : Type 'vec' of size -1

- **Input event ports sizes**

  a vector of ones, size of event input ports. The size of the vector gives the number of event input ports.

  Properties : Type 'vec' of size -1

- **Output events ports sizes**

  a vector of ones, size of event output ports. The size of the vector gives the number of of event output ports.

  Properties : Type 'vec' of size -1

- **Initial continuous state**

  A column vector of Initial State Conditions.

  Properties : Type 'vec' of size -1

- **Initial discrete state**

  A column vector Initial discrete Conditions.

  Properties : Type 'vec' of size -1

- **Initial object state**

  A Scilab list that defined the initial object state (oz).

  Properties : Type 'lis' of size -1

- **Real parameters vector**

  column vector. Any parameters used in the block can be defined here as a column vector.

  Properties : Type 'vec' of size -1

- **Integer parameters vector**

column vector. Any integer parameters used in the block can be defined here as a column vector.

Properties : Type 'vec' of size -1

- **Object parameters list**

  A Scilab list that defined the list of the Object parameters (opar).

  Properties : Type 'lis' of size -1

- **Number of modes**

  Number of Right hand side functions in the system.

  Properties : Type 'vec' of size 1

- **Number of zero_crossings**

  No. of zero-crossings

  Properties : Type 'vec' of size 1

- **Initial firing vector**

  vector. Size of this vector corresponds to the number of event outputs. The value of the i-th entry specifies the time of the preprogrammed event firing on the i-th output event port. If less than zero, no event is preprogrammed.

  Properties : Type 'vec' of size sum(%6)

- **Direct feedthrough**

  character "y" or "n", specifies if block has a direct input to output feedthrough.

  Properties : Type 'vec' of size 1

- **Time dependence**

  Time dependance : character "y" or "n", specifies if block output depends explicitly on time.

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0

- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *sinblk*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/generic_block3.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/sinblk.f (Type 4)

## Authors

- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA

# Nome

scifunc_block_m — Scilab function block

## Block Screenshot

## Contents


- Scilab function block
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

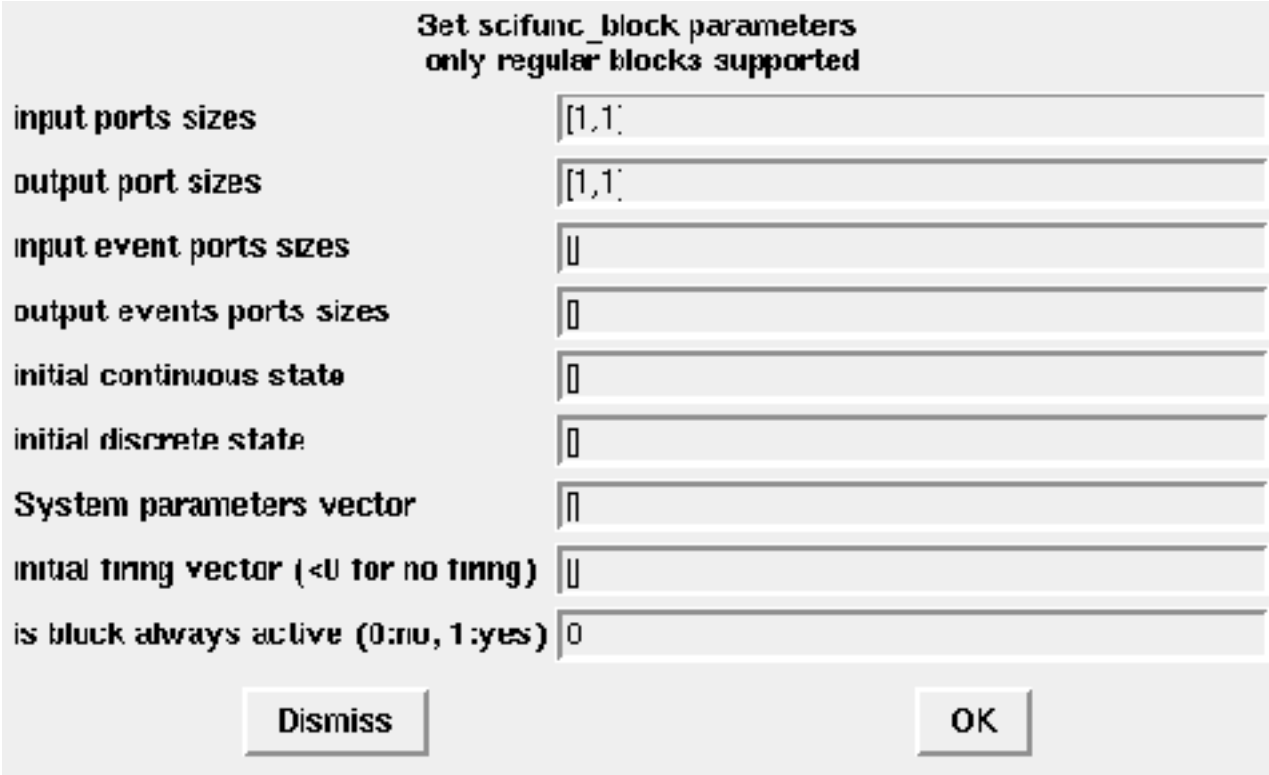

## Palette

- User defined functions palette

## Description

This block can realize any type of Scicos block. The function of the block is defined interactively using dialogue boxes and in Scilab language. During simulation, these instructions are interpreted by Scilab; the simulation of diagrams that include these types of blocks is slower. For more information see Scicos reference manual.

## Dialog box




- **input ports sizes**

  a scalar. Number of regular input ports

  Properties : Type 'vec' of size -1

- **output port sizes**

  a scalar. Number of regular output ports

  Properties : Type 'vec' of size -1

- **input event ports sizes**

  a scalar. Number of input event ports

  Properties : Type 'vec' of size -1

- **output events ports sizes**

  a scalar. Number of output event ports

  Properties : Type 'vec' of size -1

- **initial continuous state**

  a column vector.

  Properties : Type 'vec' of size -1

- **initial discrete state**

  a column vector.

Properties : Type 'vec' of size -1

- **System parameters vector**

  a string: c or d (**CBB** or **DBB** ), other types are not supported.

  Properties : Type 'vec' of size -1

- **initial firing vector**

  vector. Size of this vector corresponds to the number of event outputs. The value of the i-th entry specifies the time of the preprogrammed event firing on the i-th output event port. If less than zero, no event is preprogrammed.

  Properties : Type 'vec' of size sum(%4)

- **is block always active**

  other dialogues are opened consecutively where used may input Scilab code associated with the computations needed (block initialization, outputs, continuous and discrete state, output events date, block ending)

  Properties : Type 'vec' of size 1

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** no
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**

- **regular outputs:**

  **- port 1 : size [1,1] / type 1**

- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 0
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *scifunc*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Misc/scifunc_block_m.sci

## Computational function

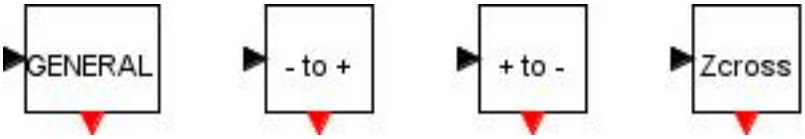
- SCI/modules/scicos/src/fortran/scifunc.f (Type 3)

# 21. Zero crossing detection palette

# Nome

Zerocrossingdetection_pal — Zero crossing detection palette

## Block Screenshot




## Module

- xcos

## Description

Zero crossing detection blocks are used to detect values crossing of state variables during the simulation. This blocks use the solvers (ODE or DAE) to do that operation.

## Blocks

- CLINDUMMY_f — Dummy
- GENERAL_f — Zero crossing
- NEGTOPOS_f - Threshold negative to positive
- POSTONEG_f - Threshold positive to negative
- ZCROSS_f - Threshold detection at zero

# Nome

GENERAL_f — GENERAL_f title

## Block Screenshot

## Contents

- GENERAL_f title
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”

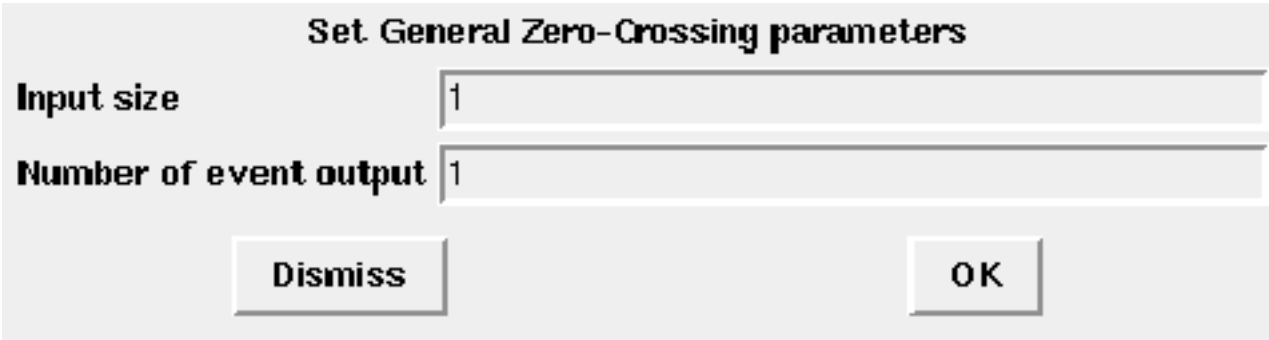
## Palette

- Zero crossing detection palette

## Description

Add here a paragraph of the function description

## Dialog box




- **Input size**

  The parameter description 1.

  Properties : Type 'vec' of size 1.

- **Number of event output**

  The parameter description 2.

Properties : Type 'vec' of size 1.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *zcross*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Threshold/GENERAL_f.sci

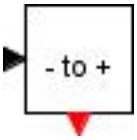
## Computational function

- SCI/modules/scicos_blocks/src/fortran/zcross.f (Type 1)

# Nome

NEGTOPOS_f — Threshold negative to positive

## Block Screenshot




## Contents


- Threshold negative to positive
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Zero crossing detection palette

## Description

An output event is generated when the unique input crosses zero with a positive slope.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *zcross*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Threshold/NEGTOPOS_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/zcross.f (Type 1)

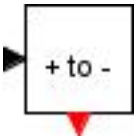
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

POSTONEG_f — Threshold positive to negative

## Block Screenshot




## Contents


- Threshold positive to negative
- • “Palette”
  - “Description”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”


## Palette

- Zero crossing detection palette

## Description

An output event is generated when the unique input crosses zero with a negative slope.

## Default properties

- **always active:** no
- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *zcross*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Threshold/POSTONEG_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/zcross.f (Type 1)

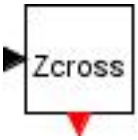
## Authors

**Ramine Nikoukhah** - INRIA

# Nome

ZCROSS_f — Threshold detection at zero

## Block Screenshot




## Contents

- Threshold detection at zero
- • “Palette”
  - “Description”
  - “Dialog box”
  - “Default properties”
  - “Interfacing function”
  - “Computational function”
  - “Authors”

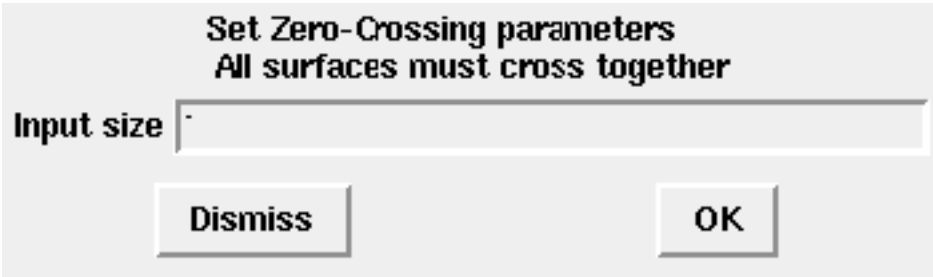
## Palette

- Zero crossing detection palette

## Description

An output event is generated when all inputs (if more than one) cross zero simultaneously.

## Dialog box




- **Input size**

  a positive integer.

  Property : Type 'vec' of size 1.

## Default properties

- **always active:** no

- **direct-feedthrough:** yes
- **zero-crossing:** yes
- **mode:** no
- **regular inputs:**

  **- port 1 : size [1,1] / type 1**
- **number/sizes of activation inputs:** 0
- **number/sizes of activation outputs:** 1
- **continuous-time state:** no
- **discrete-time state:** no
- **object discrete-time state:** no
- **name of computational function:** *zcross*

## Interfacing function

- SCI/modules/scicos_blocks/macros/Threshold/ZCROSS_f.sci

## Computational function

- SCI/modules/scicos_blocks/src/fortran/zcross.f (Type 1)

## Authors

**Ramine Nikoukhah** - INRIA

# Capítulo 8. Programming xcos Blocks

## 1. C Computational Functions

# Nome

C_macros — Utilities C macros

## Contents


- C_macros - Utilities C macros
- • “Module”
  - “Description”
  - “Inputs/outputs”
  - “Events”
  - “Parameters”
  - “States and work”
  - “Zero crossing surfaces and modes”
  - “Authors”


## Module

- xcos

## Description

The following C macros are avialable by including the file in a C computational function.

## Inputs/outputs

| Macro | Description |
|---|---|
| GetNin(blk) | Get number of regular input port. |
| GetInPortRows(blk,x) | Get number of rows (first dimension) of regular input port number x. |
| GetInPortCols(blk,x) | Get number of columns (second dimension) of regular input port number x. |
| GetInPortSize(blk,x,y) | Get regular input port size number x. (y=1 for the first dimension, y=2 for the second dimension) |
| GetInType(blk,x) | Get type of regular input port number x. |
| GetInPortPtrs(blk,x) | Get regular input port pointer of port number x. |
| GetRealInPortPtrs(blk,x) | Get pointer of real part of regular input port number x. |
| GetImagInPortPtrs(blk,x) | Get pointer of imaginary part of regular input port number x. |
| Getint8InPortPtrs(blk,x) | Get pointer of int8 typed regular input port number x. |
| Getint16InPortPtrs(blk,x) | Get pointer of int16 typed regular input port number x. |
| Getint32InPortPtrs(blk,x) | Get pointer of int32 typed regular input port number x. |

| | |
|---|---|
| Getuint8InPortPtrs(blk,x) | Get pointer of uint8 typed regular input port number x. |
| Getuint16InPortPtrs(blk,x) | Get pointer of uint16 typed regular input port number x. |
| Getuint32InPortPtrs(blk,x) | Get pointer of uint32 typed regular input port number x. |
| GetSizeOfIn(blk,x) | Get the sizeof of the regular input port number x. |
| GetNout(blk) | Get number of regular output port. |
| GetOutPortRows(blk,x) | Get number of rows (first dimension) of regular output port number x. |
| GetOutPortCols(blk,x) | Get number of columns (second dimension) of regular output port number x. |
| GetOutPortSize(blk,x,y) | Get regular output port size number x. (y=1 for the first dimension, y=2 for the second dimension) |
| GetOutType(blk,x) | Get type of regular output port number x. |
| GetOutPortPtrs(blk,x) | Get regular output port pointer of port number x. |
| GetRealOutPortPtrs(blk,x) | Get pointer of real part of regular output port number x. |
| GetImagOutPortPtrs(blk,x) | Get pointer of imaginary part of regular output port number x. |
| Getint8OutPortPtrs(blk,x) | Get pointer of int8 typed regular output port number x. |
| Getint16OutPortPtrs(blk,x) | Get pointer of int16 typed regular output port number x. |
| Getint32OutPortPtrs(blk,x) | Get pointer of int32 typed regular output port number x. |
| Getuint8OutPortPtrs(blk,x) | Get pointer of uint8 typed regular output port number x. |
| Getuint16OutPortPtrs(blk,x) | Get pointer of uint16 typed regular output port number x. |
| Getuint32OutPortPtrs(blk,x) | Get pointer of uint32 typed regular output port number x. |
| GetSizeOfOut(blk,x) | Get the sizeof of the regular output port number x. |

## Events

| **Macro** | **Description** |
|---|---|
| GetNevIn(blk) | Get the input event number. |
| GetNevOut(blk) | Get number of event output port. |
| GetNevOutPtrs(blk) | Get pointer of event output register. |

## Parameters

| **Macro** | **Description** |
|---|---|
| GetNipar(blk) | Get number of integer parameters. |

| | |
|---|---|
| GetIparPtrs(blk) | Get pointer of the integer parameters register |
| GetNrpar(blk) | Get number of real parameters. |
| GetRparPtrs(blk) | Get pointer of the real parameters register. |
| GetNopar(blk) | Get number of object parameters. |
| GetOparType(blk,x) | Get type of object parameters number x. |
| GetOparSize(blk,x,y) | Get size of object parameters number x. (y=1 for the first dimension, y=2 for the second dimension) |
| GetOparPtrs(blk,x) | Get pointer of object parameters number x. |
| GetRealOparPtrs(blk,x) | Get pointer of real object parameters number x. |
| GetImagOparPtrs(blk,x) | Get pointer of imaginary part of object parameters number x. |
| Getint8OparPtrs(blk,x) | Get pointer of int8 typed object parameters number x. |
| Getint16OparPtrs(blk,x) | Get pointer of int16 typed object parameters number x. |
| Getint32OparPtrs(blk,x) | Get pointer of int32 typed object parameters number x. |
| Getuint8OparPtrs(blk,x) | Get pointer of uint8 typed object parameters number x. |
| Getuint16OparPtrs(blk,x) | Get pointer of uint16 typed object parameters number x. |
| Getuint32OparPtrs(blk,x) | Get pointer of uint32 typed object parameters number x. |
| GetSizeOfOpar(blk,x) | Get the sizeof of the object parameters number x. |

## States and work

| **Macro** | **Description** |
|---|---|
| GetNstate(blk) | Get number of continuous state. |
| GetState(blk) | Get pointer of the continuous state register. |
| GetDstate(blk) | Get number of discrete state. |
| GetNdstate(blk) | Get pointer of the discrete state register. |
| GetNoz(blk) | Get number of object state. |
| GetOzType(blk,x) | Get type of object state number x. |
| GetOzSize(blk,x,y) | Get size of object state number x. (y=1 for the first dimension, y=2 for the second dimension) |
| GetOzPtrs(blk,x) | Get pointer of object state number x. |
| GetRealOzPtrs(blk,x) | Get pointer of real object state number x. |
| GetImagOzPtrs(blk,x) | Get pointer of imaginary part of object state number x. |
| Getint8OzPtrs(blk,x) | Get pointer of int8 typed object state number x. |
| Getint16OzPtrs(blk,x) | Get pointer of int16 typed object state number x. |
| Getint32OzPtrs(blk,x) | Get pointer of int32 typed object state number x. |
| Getuint8OzPtrs(blk,x) | Get pointer of uint8 typed object state number x. |

| | |
|---|---|
| Getuint16OzPtrs(blk,x) | Get pointer of uint16 typed object state number x. |
| Getuint32OzPtrs(blk,x) | Get pointer of uint32 typed object state number x. |
| GetSizeOfOz(blk,x) | Get the sizeof of the object state number x. |
| GetWorkPtrs(blk)(blk) | Get the pointer of the Work array. |

## Zero crossing surfaces and modes

| **Macro** | **Description** |
|---|---|
| GetNg(blk) | Get number of zero crossing surface. |
| GetGPtrs(blk) | Get pointer of the zero crossing register. |
| GetNmode(blk) | Get number of modes. |
| GetModePtrs(blk) | Get pointer of the mode register. |

## Authors

**Alan Layec** - INRIA

# Nome

C_struct — C Block structure of a computational function

## Contents


- C_struct - C Block structure of a computational function
- • “Module”
  - “Description”
  - “Inputs/outputs”
  - “Events”
  - “Parameters”
  - “States and work”
  - “Zero crossing surfaces and modes”
  - “Miscallaneous”
  - “Authors”


## Module

- xcos

## Description

The C structure of a xcos block defines all the fields to handle data provided by the simulator such inputs/outputs, parameters, states, ...

That structure of type is defined in the file , and user must include that header in each computational functions in the form :

The fields, that can be either C pointers or directly data, are then accessible via the structure :

This access is a approach and most of users should prefer the approach for facilities purpose.

In the current version of xcos, the structure is defined :

## Inputs/outputs

- **block->nin :** Integer that gives the number of regular input ports of the block.

  One can't override the index when reading sizes of input ports in the array and the index when reading data in the array with a C computational function.

  The number of regular input ports can also be got by the use of the C macros .

- **block->insz :** An array of integers of size that respectively gives the first dimensions, the second dimensions and the type of data driven by regular input ports.

Note that this array of size differs from the array and to provide full compatibilty with blocks that only use a single dimension.

Suppose that you have a block with three inputs : the first is an int32 matrix of size 3,2, the second a single complex number (matrix of size 1,1) and the last a real matrix of size 4,1.

In thescicos_model of such a block, the inputs will be defined : and the corresponding field at C computational function level will be coded as : Do the difference here in the type numbers defined at the (2,1,3) and the type numbers defined at the (84,11,10). The following table gives the correspondance for all xcos type:

- **block->inptr :** An array of pointers of size nin,1 that allow to directly acces to the data contained in the regular input matrices.

  Suppose the previous example (block with three inputs : an int32 matrix of size [3,2], a complex scalar and a real matrix of size [4,1]).

  contains three pointers, and should be viewed as arrays contained the data for the int32, the real and the complex matrices : For i.e., to directly access to the data, the user can use theses instructions : One can also use the set of C macros :

  , ,

  , ,

  , ,

  , ,

  to have the appropiate pointer of the data to handle and

  , ,

  , ,

  ,

  to handle number, dimensions and type of regular input ports. ().

  For the previous example that gives : Finally note that the regular input port registers are only accessible for reading.

- **block->nout :** Integer that gives the number of regular output ports of the block.

  One can't override the index when reading sizes of output ports in the array and the index when reading data in the array with a C computational function.

  The number of regular output ports can also be got by the use of the C macros .

- **block->outsz :** An array of integers of size that respectively gives the first dimensions, the second dimensions and the type of data driven by regular output ports.

  Note that this array of size differs from the array and to provide full compatibilty with blocks that only use a single dimension.

  Suppose that you have a block with two outputs : the first is an int32 matrix of size 3,2, the second a single complex number (matrix of size 1,1) and the last a real matrix of size 4,1.

  In thescicos_model of such a block, the outputs will be defined : and the corresponding field at C computational function level will be coded as : Do the difference here in the type numbers defined

at the (2,1,3) and the type numbers defined at the (84,11,10) and please report to the previous table to have the correspondence for all xcos type.

- **block->outptr :** An array of pointers of size nout,1 that allow to directly acces to the data contained in the regular output matrices.

  Suppose the previous example (block with three outputs : an int32 matrix of size [3,2], a complex scalar and a real matrix of size [4,1]).

  contains three pointers, and should be viewed as arrays contained the data for the int32, the real and the complex matrices : For i.e., to directly access to the data, the user can use theses instructions : One can also use the set of C macros :

  , ,

  , ,

  , ,

  , ,

  to have the appropiate pointer of the data to handle and

  , ,

  , ,

  ,

  to handle number, dimensions and type of regular output ports. (). For the previous example that gives : Finally note that the regular output port registers must be only written for =1.

## Events

- **block->nevprt :** Integer that gives the event input port number by which the block has been activated. This number is a binary coding. For i.e, if block have two event inputs ports, can take the value 1 if the block has been called by its first event input port, the value 2 if it has been called by the second event input port and 3 if it is called by the same event on both input port 1 and 2.

  Note that can be -1 if the block is internally called.

  One can also retrieve this number by using the C macros .

- **block->nevout :** Integer that gives the number of event output ports of the block (also called the length of the output event register).

  One can't override the index when setting value of events in the output event register .

  The number of event output ports can also be got by the use of the C macro .

- **block->evout :** Array of double of size nevout,1 corresponding to the output event register. That register is used to program date of events during the simulation.

  When setting values in that array, you must understand that you give a delay relative to the current time of simulator :

where $t_{event}$ is the date of the programmed event, $t_{cur}$ is the current time in the simulator and $T_{delay}$ the value that must be informed in the output event register. For i.e, suppose that you want generate an event with the first event output port, 1ms after each calls of the block, then you'll use :

Note that every events generated from output event register will be asynchronous with event coming from event input port (even if you set ).

The event output register must be only written for =3.

## Parameters

- **block->nrpar :** Integer that gives the length of the real parameter register.

  One can't override the index when reading value of real parameters in the register .

  The total number of real parameters can also be got by the use of the C macro .

- **block->rpar :** Array of double of size nrpar,1 corresponding to the real parameter register. That register is used to pass real parameters coming from the scilab/xcos environment to your block model.

  The C type of that array is (or C xcos type ).

  Suppose that you have defined the following real parameters in thescicos_model of a block : you can retrieve the previous data in the C computational function with : You can also use the C macro to get the pointer of the real parameter register. For i.e., if we define the following in an interfacing function of a xcos block : in the corresponding C computational function of that block, we'll use : Note that real parameters register is only accessible for reading.

- **block->nipar :** Integer that gives the length of the integer parameter register.

  One can't override the index when reading value of integer parameters in the register .

  The total number of integer parameters can also be got by the use of the C macro .

- **block->ipar :** Array of int of size nipar,1 corresponding to the integer parameter register. That register is used to pass integer parameters coming from the scilab/xcos environment to your block model.

  The C type of that array is (or C xcos type ).

  Suppose that you have defined the following integer parameters in thescicos_model of a block : you can retrieve the previous data in the C computational function with : You can also use the C macro to get the pointer of the real parameter register.

  Most of time in the xcos C block libraries, the integer register is used to parametrize the length of real parameters. For i.e. if you define the following in a block : the array of real parameters (parametrized by ipar) can be retrieved in the correspondig C computational function with : Note that integer parameters register is only accessible for reading.

- **block->nopar :** Integer that gives the number of the object parameters.

  One can't override the index when accessing data in the arrays , and in a C computational function.

This value is also accessible via the C macro .

- **block->oparsz :** An array of integer of size nopar,2 that contains the dimensions of matrices of object parameters.

  The first column is for the first dimension and the second for the second dimension. For i.e. if we want the dimensions of the last object parameters, we'll use the instructions : The dimensions of object parameters can be get with the following C macro : with an integer that gives the index of the object parameter, .

- **block->opartyp :** An array of integer of size nopar,1 that contains the type of matrices of object parameters.

  The following table gives the correspondence for xcos type expressed in Scilab number, in C number and also corresponding C pointers and C macros used for : The type of object parameter can also be got by the use of the C macro . For i.e, if we want the C number type of the first object parameter, we'll use the following C instructions:

- **block->oparptr :** An array of pointers of size nopar,1 that allow to directly acces to the data contained in the object parameter.

  Suppose that you have defined in the editor a block with the following**opar** field inscicos_model : Then we have two object parameters, one is an 32-bit integer matrix with two rows and two columns and the second is a vector of complex numbers that can be understand as a matrix of size 1,3.

  At the C computational function level, the instructions , , , will respectively return the values 2,1,2,3 and the instructions , the values 11 and 84.

  will contain then two pointers, and should be viewed as arrays contained data of object parameter as shown in the following figure : For i.e., to directly access to the data, the user can use theses instructions : One can also use the set of C macros :

  , ,

  , ,

  , ,

  ,

  to have the appropiate pointer of the data to handle ().

  For the previous example that gives : Note that object parameters register is only accessible for reading.

## States and work

- **block->nx :** Integer that gives the length of the continus state register.

  One can't override the index when reading or writing data in the array , or with a C computational function.

- **block->x :** Array of double of size nx,1 corresponding to the continuous state register.

  That gives the result of the computation of the state derivative.

  A value of a continuous state is readable (for i.e the first state) with the C instructions : Note that on =4, user can write some initial conditions in that register.

The pointer of that array can also be retrieve via the C macro .

- **block->xd :** Array of double of size nx,1 corresponding to the derivative of the continuous state register.

  When systems are explicitly given in terms of Ordinary Differential Equations (ODE), it can be explicitly expressed or implicitly used in the residual vector when systems are expressed in terms of Differantial Algebraic Equations (DAE).

  Both systems must be programmed with .

  For i.e the Lorentz attractor written as an ODE system with three state variables, of the form :

  will be defined :

- **block->res :** Array of double of size nx,1 corresponding to Differential Algebraic Equation (DAE) residual.

  It is used to write the vector of systems that have the following form :

  For i.e the Lorentz attractor written as a DAE system with three state variables, will be defined :

- **block->nz :** Integer that gives the length of the discrete state register.

  One can't override the index when reading data in the array with a C computational function.

  This value is also accessible via the C macros .

- **block->z :** Array of double of size nz,1 corresponding to the discrete state register.

  A value of a discrete state is directly readable (for i.e the second state) with the C instructions : Note that the state register should be only written for =4 and =2.

  The pointer of that array can also be retrieve via the C macro .

- **block->noz :** Integer that gives the number of the discrete object states.

  One can't override the index when accessing data in the arrays , and in a C computational function.

  This value is also accessible via the C macro .

- **block->ozsz :** An array of integer of size noz,2 that contains the dimensions of matrices of discrete object states.

  The first column is for the first dimension and the second for the second dimension. For i.e. if we want the dimensions of the last object state, we'll use the instructions : The dimensions of object discrete states can be get with the following C macro : with an integer that gives the index of the discrete object state, .

- **block->oztyp :** An array of integer of size noz,1 that contains the type of matrices of discrete object states.

  The following table gives the correspondence table for xcos type expressed in Scilab number, in C number and also corresponding C pointers and C macros used for : The type of discrete object state can also be got by the use of the C macro . For i.e, if we want the C number type of the first discrete object state, we'll use the following C instructions:

- **block->ozptr :** An array of pointers of size noz,1 that allow to directly acces to the data contained in the discrete object state.

  Suppose that you have defined in the editor a block with the following**odstate** field inscicos_model : Then we have two discrete object states, one is an 32-bit integer matrix with two rows and two

columns and the second is a vector of complex numbers that can be understand as a matrix of size 1,3.

At the C computational function level, the instructions , , , will respectively return the values 2,1,2,3 and the instructions , the values 11 and 84.

will contain then two pointers, and should be viewed as arrays contained data of discrete object state as shown in the following figure : For i.e., to directly access to the data, the user can use theses instructions : One can also use the set of C macros :

, ,

, ,

, ,

,

to have the appropiate pointer of the data to handle ().

For the previous example that gives : Finally note that the discrete objects state should be only written for =4 and =2.

- **block->work :** A free pointer to set a working array for the block.

  The work pointer must be firstly allocated when = 4 and finally be free in the = 5.

  Then a basic life cyle of that pointer in a C computational function should be : Note that if a block use a pointer, it will be called with =2 even if the block don't use discrete states.

  The pointer of that array can also be retrieve via the C macro .

## Zero crossing surfaces and modes

- **block->ng :** Integer that gives the number of zero crossing surface of the block.

  One can't override the index when reading/writing data in the array with a C computational function.

  The number of zero crossing surface can also be got by the use of the C macro .

- **block->g :** Array of double of size ng,1 corresponding to the zero crossing surface register.

  That register is used to detect zero crossing of state variable during time domain integration.

  Note that it is accessible for writting for = 9.

  The pointer of that array can also be retrieve via the C macro .

- **block->nmode :** Integer that gives the number of mode of the block.

  One can't override the index when reading/writing data in the array with a C computational function.

  The number of mode can also be got by the use of the C macro .

- **block->mode :** Array of integer of size nmode,1 corresponding to the mode register.

  That register is used to set the mode of state variable during time domain integration.

  It is typically accessible for writting for = 9.

  The pointer of that array can also be retrieve via the C macro .

## Miscallaneous

- **block->type :** Integer that gives the type of the computational function. For C blocks, this number is equal to 4.
- **block->label :** Strings array that allows to retrieve the label of the block.

## Authors

**Alan Layec** - INRIA

# Nome

C_utils — Utilities C functions

## Contents


- C_utils - Utilities C functions
- • “Module”
  - “Description”
  - “Authors”


## Module

- xcos

## Description

The header provides some utilities functions to interact with the simulator in the C computational functions.

- **void do_cold_restart();**

  This function forces the solver to do a cold restart. It should be used in situations where the block creates a non smooth signal. Note that in most situations, non smooth situations are detected by zero-crossings and this function is not needed. This block is used in very exceptional situations.

- **int get_phase_simulation();**

  That function returns an integer which says if the simulator is realizing time domain integration. It can returns :

  - **1 :** The simulator is on a discrete activation time.
  - **2 :** The simulator is realizing a continuous time domain integration.

- **double get_scicos_time();**

  That function returns the current time of simulator.

- **int get_block_number();**

  That function returns an integer : the block index in the compiled structure. Each block in the simulated diagram have a single index, and blocks are numbered from 1 to nblk (the total number of blocks in the compiled structure).

- **void set_block_error(int);**

  Function to set a specific error during the simulation for the current block. If it is used, then after the execution of the computational function of the block, the simulator will end and will return an error message associated to the number given in the integer argument.

  The following calls are allowed :

  - **set_block_error(-1); :** the block has been called with input out of its domain,
  - **set_block_error(-2); :** singularity in a block,
  - **set_block_error(-3); :** block produces an internal error,

  - **set_block_error(-16); :** cannot allocate memory in block.

- **void end_scicos_sim();**

  A very specific function to set the current time of the simulator to the final time integration.

  Only expert user should use this function.

- **void set_pointer_xproperty(int* pointer);**

  This function set a vector of integer to inform the type (algebraic or differential) of the continuous state variables of the block.

- **void * scicos_malloc(size_t);**

  That function must be used to do allocation of scicos pointers inside a C computational function and in particular for =4 for the work pointer .

- **void scicos_free(void *p);**

  That function must be used to free scicos pointers inside a C computational function and in particular for =5 for the work pointer .

## Authors


- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA


# 2. Scilab Computational Functions

# Nome

sci_struct — Scicos block structure of a scilab computational function

## Contents


- sci_struct - Scicos block structure of a scilab computational function
- • “Module”
  - “Description”
  - “Inputs/outputs”
  - “Events”
  - “Parameters”
  - “States”
  - “Zero crossing surfaces and modes”
  - “Miscallaneous”
  - “Authors”


## Module

- xcos

## Description

A Scicos computational function of type 5 can be realized by the use of a Scilab function. That function doesn't really differs from all other scilab function : one can use all functions and instructions of the scilab language inside that function to do the computation.

Such a function must be written in a file with extension .sci, must be loaded inside scilab by the common loading scilab function (, , , ,...) and must have two right hand side arguments and one left hand side argument, as the following calling sequence :

When the simulator is calling such a computational function, it build a scilab structure (in the previous exemple this is the named rhs/lhs arguments) from his own internal C reprensation of a block structure (see for more details about the C structure of scicos blocks).

That scilab structure is a scilab typed list variable that have the following fields :

Each fields are then accessible inside the scilab computational function by the use of :

## Inputs/outputs

- **block.nin :** a scalar that gives the number of regular input ports. This is a read only data.
- **block.insz :** a vector of size , that gives the dimensions and types of the regular input ports.
  - **:** are the first dimensions.
  - **:** are the second dimensions.
  - **:** are the type of data (C coding).

This is a read only data.

- **block.inptr :** a list of size that enclosed typed matrices for regular input ports. Each element correspond to only one regular input port. Then i-th matrix of the block.inptr list will have the dimensions [ , ] and the type .

  The data type that can be provided by regular input ports are :

  - **1 :** matrix of real numbers,
  - **2 :** matrix of complex numbers,
  - **3 :** matrix of int32 numbers,
  - **4 :** matrix of int16 numbers,
  - **5 :** matrix of int8 numbers,
  - **6 :** matrix of uint32 numbers,
  - **7 :** matrix of uint16 numbers,
  - **8 :** matrix of uint8 numbers.

  This is a read only data.

- **block.nout :** a scalar that gives the number of regular output ports. This is a read only data.
- **block.outsz :** a vector of size , that gives the dimensions and types of the regular output ports.
  - **:** are the first dimensions.
  - **:** are the second dimensions.
  - **:** are the type of data (C coding).

  This is a read only data.

- **block.outptr :** a list of size that enclosed typed matrices for regular output ports. Each element correspond to only one regular output port. Then i-th matrix of the block.outptr list will have the dimensions [ , ] and the type .

  The data type that can be provided by regular output ports are :

  - **1 :** matrix of real numbers,
  - **2 :** matrix of complex numbers,
  - **3 :** matrix of int32 numbers,
  - **4 :** matrix of int16 numbers,
  - **5 :** matrix of int8 numbers,
  - **6 :** matrix of uint32 numbers,
  - **7 :** matrix of uint16 numbers,
  - **8 :** matrix of uint8 numbers.

  Values of regular output ports will be saved in theC structure of the block only for =6 and =1.

## Events

- **block.nevprt :** a scalar given the event input port number (binary coding) which have activated the block. This is a read only data.
- **block.nevout :** a scalar given the number of output event port of the block. This is a read only data.
- **block.evout :** a vector of size corresponding to the register of output event. Values of output event register will be saved in theC structure of the block only for =3.

## Parameters

- **block.nrpar :** a scalar given the number of real parameters. This is a read only data.
- **block.rpar :** a vector of size corresponding to the real parameter register. This is a read only data.
- **block.nipar :** a scalar given the number of integer parameters. This is a read only data.
- **block.ipar :** a vector of size correspondig to the integer parameter register. This is a read only data.
- **block.nopar :** a scalar given the number of object parameters. This is a read only data.
- **block.oparsz :** a matrix of size , that respectively gives the first and the second dimension of object parameters. This is a read only data.
- **block.opartyp :** a vector of size given the C coding type of data. This is a read only data.
- **block.opar :** a list of size given the values of object parameters. Each element of can be either a typed matrix or a list. Only matrix that encloses numbers of type real, complex, int32, int16, int8, uint32, uint16 and uint8 are allowed, all other types of scilab data will be enclosed in a sub-list. This is a read only data.

## States

- **block.nz :** a scalar given the number of discrete state for the block. This is a read only data.
- **block.z :** a vector of size corresponding to the discrete state register. Values of discrete state register will be saved in theC structure of the block only for =4, =6, =2 and =5.
- **block.noz :** a scalar that gives the number of discrete object state. This is a read only data.
- **block.ozsz :** a matrix of size , that respectively gives the first and the second dimension of discrete object state. This is a read only data.
- **block.oztyp :** a vector of size given the C coding type of data.
- **block.oz :** a list of size given the values of discrete object states. Each element of can be either a typed matrix or a list. Only matrix that encloses numbers of type real, complex, int32, int16, int8, uint32, uint16 and uint8 are allowed, all other types of scilab data will be enclosed in a sub-list. Values of discrete object state will be saved in theC structure of the block only for =4, =6, =2 and =5.
- **block.nx :** a scalar given the number of continuous state for the block. This is a read only data.
- **block.x :** a vector of size given the value of the continuous state register. Values of the continuous state register will be saved in theC structure of the block only for =4, =6 and =2.

- **block.xd :** a vector of size given the value of the derivative continuous state register. Values of the derivative continuous state register will be saved in theC structure of the block only for =4, =6, =0 and =2.
- **block.res :** a vector of size corresponding to the Differential Algebraic Equation (DAE) residual. Values of that register will be saved in theC structure of the block only for =0, and =10.

## Zero crossing surfaces and modes

- **block.ng :** a scalar given the number of zero crossing surfaces for the block. This is a read only data.
- **block.g :** a vector of size corresponding to the zero crossing register. Values of that register will be saved in theC structure of the block only for =9.
- **block.nmode :** a scalar given the number of mode for the block. This is a read only data.
- **block.mode :** a vector of size that corresponds to the mode register. Values of that register will be saved in theC structure of the block only for =9, with =1.

## Miscallaneous

- **block.type :** a scalar given the type of the block. This is a read only data.
- **block.label :** a string given the label of the block. This is a read only data.

## Authors

**Alan Layec** - INRIA

# 3. Utilities Functions

# Nome

curblock — Return the current called xcos block during the simulation

## Module

- xcos

```
blk=curblock()
```

## Parameters

- **blk :** the current block number in the compiled structure.

## Authors

**Ramine Nikoukhah** - INRIA

# Nome

getblocklabel — Get the label of a scicos block

## Module

- xcos

```
[label]=getblocklabel(blk)
```

## Parameters

- **blk :** Integer parameter. Set the index of a block (in the compiled structure).
- **label :** String parameter. Gives the string of the label of the block numbered blk.

# Nome

getscicosvars — Supervisor utility function

```
[myvar]=getscicosvars(str) [myvar]=getscicosvars([str1;str2;...])
```

## Module

- xcos

## Description

That utility function is used to retrieve working arrays of Scicos simulator and compiler during simulation.

It can be used inside a Scilab block to get information of all type of blocks. That function is very useful to debug diagrams and to do prototypes of simulations.

```
[myvar]=getscicosvars(str)
[myvar]=getscicosvars([str1;str2;...])
```

## Parameters

- **str,str1,str2,... :** That parameter can be a string or a matrix of string. The following entries are allowed :

  "x" : get the continuous state register.

  "nx" : get the length of the continuous state register.

  "xptr" : get the pointers register of the continuous state register.

  "zcptr" : get the pointers register of the zero-crossing surfaces register.

  "z" : get the discrete state register.

  "nz" : get the length of the continuous state register.

  "zptr" : get the pointers register of the discrete state register.

  "noz" : get the number of elements of the discrete object state list.

  "oz" : get the discrete object state list.

  "ozsz" : get the size of the elements of the discrete object state list.

  "oztyp" : get the type of the elements of the discrete object state list.

  "ozptr" : get the pointers register of the discrete object state list.

  "rpar" : get the real parameter register.

  "rpptr" : get the pointers register of the real parameter register.

  "ipar" : get the integer parameter register.

  "ipptr" : get the pointers register of the integer parameter register.

  "opar" : get the object parameter list.

"oparsz" : get the size of the elements of the object parameter list.

"opartyp" : get the type of the elements of the object parameter list.

"opptr" : get the pointers register of the object parameter list.

"outtb" : get the output register.

"inpptr" : get the pointers register of the number of regular input ports.

"outptr" : get the pointers register of the number of regular output ports.

"inplnk" : get the pointers register of the links connected to regular input ports.

"outlnk" : get the pointers register of the links connected to regular output ports.

"subs" : not used

"tevts" : get the current date register of the agenda.

"evtspt" : get the current event register of the agenda.

"pointi" : get the next event to be activated.

"iord" : get the vector of blocks activated at the start of the simulation.

"oord" : get the vector of blocks whose outputs affects computation of continuous state derivatives.

"zord" : get the vector of blocks whose outputs affects computation of zero-crossing surfaces.

"funtyp" : get the vector of type of computational functions.

"ztyp" : get the pointers vector for blocks which use zero-crossing surfaces.

"cord" : get the vector of blocks whose outputs evolve continuously.

"ordclk" : get the matrix associated to blocks activated by output activation ports.

"clkptr" : get the pointers vector for output activation ports.

"ordptr" : get the pointers vector to ordclk designating the part of ordclk corresponding to a given activation.

"critev" : get the vector of the critical events.

"mod" : get the vector pointers of block modes.

"nmod" : get the length of the vector pointers of block modes.

"iz" : get the register that store pointers of block-work.

"izptr" : get the pointers vector of the register that store C pointers of block-work.

"nblk" : get the number of block.

"outtbptr" : get the register that store C pointers of outtb.

"outtbpsz" : get the register that store the size of the elements of outtb.

"outtbtyp" : get the register that store the type of the elements of outtb.

"nlnk" : get the number of output.

"ncord" : get the number of blocks whose outputs evolve continuously.

"nordptr" : get the number of blocks whose outputs evolve by activation.

"iwa" : n.d.

"blocks" : get a scilab list that contains all block structures contains in the diagram.

"ng" : get length of the zero-crossing surfaces register.

"g" : get the zero-crossing surfaces register.

"t0" : get the current time of the simulation.

"tf" : get the final time of the simulation.

"Atol" : get the integrator absolute tolerance for the numerical solver.

"rtol" : get the integrator relative tolerance for the numerical solver.

"ttol" : get the tolerance on time of the simulator.

"deltat" : get the maximum integration time interval.

"hmax" : get the maximum step size for the numerical solver.

"nelem" : get the number of elements in outtb.

"outtb_elem" : get the vector of the number of elements in outtb.

- **myvar :** That output parameter can be an int32 matrix, a double matrix or a Tlist. This is given by the input parameter.

## See Also

- DEBUG_SCICOS - Debug block (Scicos Block)

## Authors

- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA

# Nome

phase_simulation — Get the current simulation phase

```
[psim]=phase_simulation()
```

## Module

- xcos

## Description

That function says if the Scicos simulator is realizing the time domain integration.

```
[psim]=phase_simulation()
```

## Parameters

- **psim :** get the current phase of the simulation
  - **1 :** The simulator is on a discrete activation time.
  - **2 :** The simulator is realizing a continuous time domain integration.

# Nome

pointer_xproperty — Get the type of a continuous time state variable

```
[xprop]=pointer_xproperty
```

## Module

- xcos

## Description

This function returns a vector that informs the type (algebraic or differential) of the continuous state variables of a block.

```
[xprop]=pointer_xproperty
```

## Parameters

- **xprop**

  The value gives the type of the states :

  - -1 : an algebraic state.
  - 1 : a differential state.

## See Also

- set_xproperty - Set the type of a continuous time state variable (Scilab Function)

# Nome

scicos_time — Returns the current time during simulation

## Module

- xcos

```
t=scicos_time()
```

## Parameters

- **t :** that is the current simulated time returned in real number.

# Nome

set_blockerror — set the block error number

```
set_blockerror(n)
```

## Module

- xcos

## Description

Function to set a specific error during the simulation for the current block. If it is used, then after the execution of the computational function of the block, the simulator will end and will return an error message associated to the number given in argument.

```
set_blockerror(n)
```

## Parameters

- **n** : an error number. The following calls are allowed :
  - **set_blockerror(-1)**

    the block has been called with input out of its domain
  - **set_blockerror(-2)**

    singularity in a block
  - **set_blockerror(-3)**

    block produces an internal error
  - **set_blockerror(-16)**

    cannot allocate memory in block

## Authors

**Alan Layec** - INRIA

# Nome

set_xproperty — Set the type of a continuous time state variable

```
set_xproperty(xprop)
```

## Module

- xcos

## Description

This function set a vector to inform the type (algebraic or differential) of the continuous state variables of a block.

```
set_xproperty(xprop)
```

## Parameters

- **xprop**

  The value gives the type of the states :

  - -1 : an algebraic state.
  - 1 : a differential state.

## See Also

- pointer_xproperty - Get the type of a continuous time state variable (Scilab Function)

# Capítulo 9. Scilab Data Structures

## 1. Blocks

# Nome

scicos_block — Define a block structure

## Module

- xcos

## block

Basic structure that define a xcos block.

That structure includes fields graphics, model, gui and doc.

Size : 5.

Type : scilab list.

- **graphics**

  Scilab object including graphical information concerning the features of the block.

  Size : 14.

  Type : scilab list.

- **model**

  Scilab list that contains the features of the block used for the compilation.

  Size : 23.

  Type : Scilab list.

- **gui**

  The name of the Scilab GUI function associated with the block.

  Size : 1.

  Type : string.

- **doc**

  Field used for documentation of the block Size : 1.

  Type : string.

## File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_block.sci

# Nome

scicos_graphics — Define a graphics structure

## Module

- xcos

## graphics

Scilab object including graphical information concerning the features of the block.

Size : 14.

Type : scilab list.

- **orig**

  Vector [xo,yo], where xo is the x coordinate of the block origin and yo is the y coordinate of the block origin.

  [xo,yo] is the coordinate of down-left point of the block shape.

  Size : 2.

  Type : row vector of real.

- **sz**

  Vector [w,h], where w is the block width and h the block height.

  Size : 2.

  Type : row vector of real.

- **flip**

  Set the block orientation. If true the input ports are on the left of the box and output ports are on the right.

  If false the input ports are on the right of the box and output ports are on the left.

  Size : 1.

  Type : boolean.

- **theta**

  Set the angle of the Scicos object.

  This value is in degree and is included in [-360,360].

  Size : 1.

  Type : real.

- **exprs**

  Strings including formal expressions used in the dialog box of the block.

Size : number of formal expressions.

Type : column vector of strings.

- **pin**

  Vector. pin(i) is the number of the link connected to the ith regular input port (counting from one), or 0 if this port is not connected.

  Size : number of regular input ports.

  Type : column vector of integers.

- **pout**

  Vector. pout(i) is the number of the link connected to the ith regular output port (counting from one), or 0 if this port is not connected.

  Size : number of regular output ports.

  Type : column vector of integers.

- **pein**

  Vector. pein(i) is the number of the link connected to the ith event input port (counting from one), or 0 if this port is not connected.

  Size : number of events input ports.

  Type : column vector of integers.

- **peout**

  Vector. peout(i) is the number of the link connected to the ith event output port (counting from one), or 0 if this port is not connected.

  Size : number of events output ports.

  Type : column vector of integers.

- **gr_i**

  Strings including Scilab graphics expressions for customizing the block graphical aspect. This field may be set with Icon sub_menu.

  Size : -.

  Type : column vector of strings.

- **id**

  A string including an identification for the block. The string is displayed under the block in the diagram.

  Size : 1.

  Type : string.

- **in_implicit**

  A vector of strings including 'E' or 'I'.

'E' and 'I' stand respectively for explicit and implicit port, and this vector indicates the nature of each input port. For regular blocks (not implicit), this vector is empty or contains only "E".

Size : nul or number of regular input ports.

Type : column vector of strings.

- **out_implicit**

  A vector of strings including 'E' or 'I'.

  'E' and 'I' stand respectively for explicit and implicit port, and this vector indicates the nature of each output port. For regular blocks (not implicit), this vector is empty or contains only "E".

  Size : nul or number of regular output ports.

  Type : column vector of strings.

## File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_graphics.sci

# Nome

scicos_model — Define a model structure

# Module

- xcos

# model

Scilab list that contains the features of the block used for the compilation.

Size : 23.

Type : Scilab list.

- **sim**

  A list containing two elements. The first element is a string containing the name of the computational function (C, Fortran,or Scilab). The second element is an integer specifying the type of the computational function. Currently type 4 and 5 are used, but older types continue to work to ensure backward compatibility.

  For some older case, sim can be a single string and that means that the type is supposed to be 0.

  Size : 2.

  Type : Scilab list.

- **in**

  A vector specifying the number and size of the first dimension of regular input ports indexed from top to bottom of the block. If no input port exist in==[].

  The size can be negative, equal to zero or positive :

  - If a size is less than zero, the compiler will try to find the appropriate size.
  - If a size is equal to zero, the compiler will affect this dimension by added all positive size found in that vector
  - If a size is greater than zero, then the size is explicitly given.

  Size : number of regular input ports.

  Type : column vector of integer numbers.

- **in2**

  A vector specifying the second dimension of regular input ports indexed from top to bottom of the block.

  in with in2 formed then the regular input sizes matrix.

  For compatibility, this dimension can stay empty ([]).

  That means that the dimensions of input ports will be in,1 The size can be negative, equal to zero or positive :

- If a size is less than zero, the compiler will try to find the appropriate size.
- If a size is equal to zero, the compiler will affect this dimension by added all positive size found in that vector.
- If a size is greater than zero, then the size is explicitly given.

Size : number of regular input ports.

Type : column vector of integer numbers.

- **intyp**

  A vector specifying the types of regular input ports.

  Its sizes is equal to the sizes of in.

  The types of regular input ports can be :

  - 1 real matrix,
  - 2 complex matrix,
  - 3 int32 matrix,
  - 4 int16 matrix,
  - 5 int8 matrix,
  - 6 uint32 matrix,
  - 7 uint16 matrix,
  - 8 uint8 matrix.

  Size : number of regular input ports.

  Type : column vector of integer numbers.

- **out**

  A vector specifying the number and size of the first dimension of regular output ports indexed from top to bottom of the block. If no output port exist out==[].

  The size can be negative, equal to zero or positive :

  - If a size is less than zero, the compiler will try to find the appropriate size.
  - If a size is equal to zero, the compiler will affect this dimension by added all positive size found in that vector
  - If a size is greater than zero, then the size is explicitly given.

  Size : number of regular output ports.

Type : column vector of integer numbers.

- **out2**

  A vector specifying the second dimension of regular output ports indexed from top to bottom of the block.

  out with out2 formed then the regular output sizes matrix.

  For compatibility, this dimension can stay empty ([]). That means that the dimensions of output ports will be out,1 That dimension can be negative, equal to zero or positive :

  - If a size is less than zero, the compiler will try to find the appropriate size.
  - If a size is equal to zero, the compiler will affect this dimension by added all positive size found in that vector.
  - If a size is greater than zero, then the size is explicitly given.

  Size : number of regular output ports.

  Type : column vector of integer numbers.

- **outtyp**

  A vector specifying the types of regular output ports.

  Its sizes is equal to the sizes of out.

  The types of regular output ports can be :

  - 1 real matrix,
  - 2 complex matrix,
  - 3 int32 matrix,
  - 4 int16 matrix,
  - 5 int8 matrix,
  - 6 uint32 matrix,
  - 7 uint16 matrix,
  - 8 uint8 matrix.

  Size : number of regular output ports.

  Type : column vector of integer numbers.

- **evtin**

  A vector specifying the number and sizes of activation inputs. Currently activation ports can be only of size one.

  If no event input port exists evtin must be equal to [].

Size : number of input event ports.

Type : column vector of integer numbers.

- **evtout**

  A vector specifying the number and sizes of activation outputs.

  Currently activation ports can be only of size one.

  If no event output port exists evtout must be equal to [].

  Size : number of output event ports.

  Type : column vector of integer numbers.

- **state**

  Vector containing initial values of continuous-time state.

  Must be [] if no continuous state.

  Size : number of continuous-time state.

  Type : column vector of real numbers.

- **dstate**

  Vector containing initial values of discrete-time state.

  Must be [] if no discrete state.

  Size : number of discrete-time state.

  Type : column vector of real numbers.

- **odstate**

  List containing initial values of objects state.

  Must be list() if no objects state.

  Objects state can be any types of scilab variable.

  In the computational function case of type 4 (C blocks) only elements containing matrix of real, complex, int32, int16 ,int8 ,uint32, uit16 and uint8 will be correctly provided for readind/writing.

  Size : number of objects state.

  Type : scilab list of scilab objects.

- **rpar**

  The vector of floating point block parameters.

  Must be [] if no floating point parameters.

  Size : number of real parameters.

  Type : column vector of real numbers.

- **ipar**

The vector of integer block parameters.

Must be [] if no integer parameters.

Size : number of integer parameters.

Type : column vector of integer numbers.

- **opar**

  List of objects block parameters. Must be list() if no objects parameters.

  Objects parameters can be any types of scilab variable.

  In the computational function case of type 4 (C blocks) only elements containing matrix of real, complex, int32, int16 ,int8 ,uint32, uit16 and uint8 will be correctly provided for reading.

  Size : number of objetcs parameters.

  Type : list of scilab object.

- **blocktype**

  Character that can be set to 'c' or 'd' indifferently for standard blocks. 'x' is used if we want to force the computational function to be called during the simulation phase even if the block does not contribute to computation of the state derivative.

  'l', 'm' and 's' are reserved. Not to be used.

  Size : 1.

  Type : Character.

- **firing**

  Vector of initial event firing times of size equal to the number of activation output ports (see evout). It contains output initial event dates (Events generated before any input event arises). Negative values stands for no initial event on the corresponding port. Size : number of output event ports.

  Type : column vector of real numbers.

- **dep_ut**

  Boolean vector [dep_u, dep_t].

  - **dep_u**

    true if block is always active.

    (output depends continuously of the time)

  - **dep_t**

    true if block has direct feed-through, i.e., at least one of the outputs depends directly (not through the states) on one of the inputs. In other words, when the computational function is called with flag 1, the value of an input is used to compute the output.

  Size : 2.

Type : Boolean vector.

- **label**

  String that defines a label. It can be used to identify a block in order to access or modify its parameters during simulation.

  Size : 1.

  Type : string.

- **nzcross**

  Number of zero-crossing surfaces.

  Size : Number of zero-crossing surfaces.

  Type : column vector of integer numbers.

- **nmode**

  Length of the mode register. Note that this gives the size of the vector mode and not the total number of modes in which a block can operate in. Suppose a block has 3 modes and each mode can take two values, then the block can have up to 23=8 modes.

  Size : Number of modes.

  Type : column vector of integer numbers.

- **equations**

  Used in case of implicit blocks.

  Data structure of type modelica which contains modelica code description if any. That list contains four entries :

  - **model**

    a string given the name of the file that contains the modelica function.

  - **inputs**

    a colunm vector of strings that contains the names of the modelica variables used as inputs.

  - **outputs**

    a colunm vector of strings that contains the names of the modelica variables used as outputs.

  - **parameters**

    a list with two entries. The first is a vector of strings for the name of modelica variable names used as parameters and the second entries is a list that contains the value of parameters.

    Names of modelica states can also be informed with parameters. In that case a third entry is used to do the difference between parameters and states.

    For i,e : mo.parameters=list(['C','v'],list(C,v),[0,1]) means that 'C' is a parameter(0) of value C, and 'v' is a state(1) with initial value v.

Size : 5.

Type : scilab list.

## File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_model.sci

# 2. Compilation/Simulation

# Nome

scicos_cpr — Compiled Scicos structure

# Module

- xcos

# cpr

The Scilab object cpr contains the result of the compilation.

That structure includes fields state, sim, cor and corinv.

Size : 5.

Type : scilab list.

- **state**

  Scilab typed list of type xcs. It contains all the states of the model, that is, everything than can evolve during the simulation.

  state contains x, z, oz, iz, tevts, evtspt, pointi and outtb.

  Size : 9.

  Type : scilab tlist.

- **sim**

  Scilab typed list of type scs. It contains static arrays coming from the result of the compilation. That arrays does not evolve during the simulation.

  Size : 33.

  Type : scilab tlist.

- **cor**

  It is a list with same recursive structure as scs_m. Each leaf contains the index of associated block in sim data structure. Size : number of objects in scs_m.

  Type : scilab list.

- **corinv**

  corinv(i) is the path of i th block defined in sim data structure in the scs_m data structure.

  Size : number of blocks in the compiled structre.

  Type : scilab list.

# File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_cpr.sci

# Nome

scicos_sim — Define a sim structure

# Module

- xcos

# sim

Scilab typed list of type scs. It contains static arrays coming from the result of the compilation. That arrays does not evolve during the simulation.

Size : 33.

Type : scilab tlist.

- **funs**

  A list containing names of the computational functions or scilab functions.

  Size : number of blocks.

  Type : list of strings and/or scilab function.

- **xptr**

  A vector pointer to the continuous time state register x. The continuous-time state of block i is state.x(sim.xptr(i):sim.xptr(i+1)-1).

  Size : number of blocks + 1.

  Type : column vector of integers.

- **zptr**

  A vector pointer to the discrete time state register z. The discrete-time state of block i is state.z(sim.zptr(i):sim.zptr(i+1)-1).

  Size : number of blocks + 1.

  Type : column vector of integers.

- **ozptr**

  A vector pointer to the object discrete state register oz. The object discrete state of block i is state.oz(sim.ozptr(i):sim.ozptr(i+1)-1).

  Size : number of blocks + 1.

  Type : column vector of integers.

- **zcptr**

  A vector pointer to the zero-crossing surfaces.

  register. That vector gives by block the used number of the zero-crossing.

  Size : number of blocks + 1.

Type : column vector of integers.

- **inpptr**

  (sim.inpptr(i+1)-sim.inpptr(i)) gives the number of regular input ports of the i block.

  inpptr(i) points to the beginning of ith block inputs within the indirection table inplnk.

  Size : number of blocks + 1.

  Type : column vector of integers.

- **outptr**

  (sim.outptr(i+1)-sim.outptr(i)) gives the number of regular ouyput ports of the i block.

  outptr(i) points to the beginning of ith block outputs within the indirection table outlnk.

  Size : number of blocks + 1.

  Type : column vector of integers.

- **inplnk**

  (cpr.sim.inplnk(cpr.sim.inpptr(i)-1+j)) is the index of the link connected to the jth input port of the ith block where j goes from 1 to (cpr.sim.inpptr(i+1)-cpr.sim.inpptr(i)).

  Size : total number of regular input port.

  Type : column vector of integers.

- **outlnk**

  (cpr.sim.outlnk(cpr.sim.outptr(i)-1+j)) is the index of the link connected to the jth output port of the ith block where j goes from 1 to (cpr.sim.outptr(i+1)-cpr.sim.outptr(i)).

  Size : total number of regular output port.

  Type : column vector of integers.

- **rpar**

  Vector of real parameters that is obtained by concatenating the real parameters registers of all the blocks.

  Size : total number of real parameters.

  Type : column vector of real numbers.

- **rpptr**

  A vector pointer to the real parameters register rpar. The real parameters of block i are sim.rpar(sim.rpptr(i):sim.rpptr(i+1)-1).

  Size : number of blocks + 1.

  Type : column vector of integer.

- **ipar**

  Vector of integer parameters that is obtained by concatenating the integer parameters registers of all the blocks.

Size : total number of integer parameters.

Type : column vector of integer.

- **ipptr**

  A vector pointer to the integer parameters register ipar. The integer parameters of block i are sim.ipar(sim.ipptr(i):sim.ipptr(i+1)-1).

  Size : number of blocks + 1.

  Type : column vector of real numbers.

- **opar**

  List of object parameters that is obtained by concatenating the list of object parameters of all the blocks.

  Size : total number of object parameters.

  Type : list of scilab objects.

- **opptr**

  A vector pointer to the object parameters list opar. The object parameters of block i are sim.opar(sim.opptr(i):sim.opptr(i+1)-1).

  Size : number of blocks + 1.

  Type : column vector of integers.

- **clkptr**

  A vector pointer to output activation ports.

  (cpr.sim.clkptr(i):cpr.sim.clkptr(i+1)-1) gives the number of output event ports of the block i.

  Size : number of blocks + 1.

  Type : column vector of integers.

- **ordptr**

  A vector pointer to ordclk designating the part of ordclk corresponding to a given activation.

  (cpr.sim.ordptr(i):cpr.sim.ordptr(i+1)-1) points to the region within ordclk indicates the number of blocks activated by the output event ports numbered i.

  Size : number of sources of activation + 1.

  Type : column vector of integers.

- **execlk**

  Unused.

  Size : - Type : matrix of real.

- **ordclk**

  A matrix associated to blocks activated by output activation ports. The first column contains the block number, and the second, the event code by which the block should be called.

Size : total number of blocks summed by source of activations.

Type : matrix of integers.

- **cord**

  A matrix associated to always active blocks.

  The first column contains the block number, and the second, the event code by which the block should be called.

  Size : ncord.

  Type : matrix of integers.

- **oord**

  Subset of cord. Blocks of that matrix have outputs which affect computation of continuous state derivatives.

  Size : noord.

  Type : matrix of integers.

- **zord**

  Subset of zord. Blocks of that matrix have outputs which affect computation of zero-crossing surfaces.

  Size : nzord.

  Type : matrix of integers.

- **critev**

  A vector of size equal to the number of activations and containing zeros and ones. The value one indicates that the activation is critical in the sense that the continuous-time solver must be cold restarted.

  Size : number of source of activation.

  Type : column vector of integers.

- **nb**

  Number of blocks. Note that the number of blocks may differ from the original number of blocks in the diagram because c_pass2 may duplicate some conditional blocks.

  Size : 1.

  Type : integer.

- **ztyp**

  A vector of size equal to the number of blocks.

  A 1 entry indicates that the block may have zero-crossings, even if it doesn't in the context of the diagram. Usually not used by the simulator.

  Size : number of blocks.

  Type : column vector of integers.

- **nblk**

  Not used. Set to nb.

  Size : 1 Type : integer.

- **ndcblk**

  Not used.

  Size : - Type : integer.

- **subscr**

  Not used.

  Size : 0 Type : empty real.

- **funtyp**

  A vector of size equal to the number of blocks indicating the type of the computational function of the block. Block type can be 0 through 5.

  Currently only type 4 (C language) and type 5 (Scilab language) computational functions should be used. But older blocks can also be used.

  Size : number of blocks.

  Type : column vector of integer.

- **iord**

  A matrix associated to blocks that must be activated at the start of the simulation. This includes blocks inheriting from constant blocks and always active blocks.

  Size : niord.

  Type : matrix of integers.

- **labels**

  A string vector of size equal to the number of blocks containing block labels.

  Size : numbers of blocks.

  Type : column vector of strings.

- **modptr**

  A vector pointer to the block modes.

  Size : number of blocks + 1.

  Type : column vector of integer.

## File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_sim.sci

# Nome

scicos_state — Define a state structure

# Module

- xcos

# state

Scilab typed list of type xcs. It contains all the states of the model, that is, everything than can evolve during the simulation.

state contains x, z, oz, iz, tevts, evtspt, pointi and outtb.

Size : 9.

Type : scilab tlist.

- **x**

  The continuous-time state register, which is obtained by concatenating the continuous-time states of all the blocks.

  Size : total of all the size of continuous-time state registers.

  Type : column vector of real numbers.

- **z**

  The discrete-time state register, which is obtained by concatenating the discrete-time states of all the blocks.

  Size : total of all the size of discrete-time state registers.

  Type : column vector of real number.

- **oz**

  The list of the object discrete-time state, which is obtained by concatenating the object discrete-time states of all the blocks.

  Size : total of all the size of object state.

  Type : list of scilab object.

- **iz**

  Vector of size equal to the number of blocks.

  That vector is used to store pointers of the working state register (work). If a block needs to allocate memory at initialization (flag 4), the associated pointer is saved here.

  Size : number of blocks.

  Type : column vector of real numbers.

- **tevts**

Vector of size equal to the number of activation sources. It contains the scheduled times for programmed activations in evtspt.

Size : number of activation sources.

Type : column vector of integers.

- **evtspt**

  Vector of size equal to the number of activation sources. It is an event scheduler.

  Size : number of activation sources.

  Type : column vector of integers.

- **pointi**

  The number of the next programmed event.

  Size : 1.

  Type : integer.

- **outtb**

  Scilab list containing all output registers of blocks. Each element of that list contains typed matrix-based data.

  Size : number of regular output ports.

  Type : list of scilab matrix.

## File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_state.sci

# 3. Diagram

# Nome

scicos_diagram — Define a scs_m structure

## Module

- xcos

## diagram

Size : 4.

Type : scilab list.

- **props**

  Diagram properties.

  This entry contains various informations such some main diagram initials values.

  This variable is a tlist of type and contains wpar, title, tol, tf, context, options and doc.

  Size : 11.

  Type : Scilab tlist of type .

- **objs**

  List of objects included in the Scicos diagram.

  The objects used in scicos areblock ,link and Text.

  The objects can also be deleted object data structure.

  Deleted object data structure is marked list('Deleted').

  Size : total number of objects in the diagram.

  Type : Scilab tlist of type , or Text.

- **version**

  A string that gives the version of the Scicos diagram.

  This is used to provide compatibility with old diagram.

  Note that you can get the current version of Scicos by using the entry 'About scicos' in the help menu or by using the function get_scicos_version().

  Size : 1.

  Type : String.

## File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_diagram.sci

# Nome

scicos_params — Define a param structure

## Module

- xcos

## params

Size : 11.

Type : scilab list.

- **wpar**

  This vector is not currently used.

  It may be used in the future to code window sizes of the editor.

  Size : 6.

  Type : column vector or real.

- **title**

  Vector of character strings, where the first one is the diagram title and default name of save file name, and the second one is the path of the directory of the file name.

  Size : 2.

  Type : row vector of strings.

- **tol**

  A vector containing simulation parameters including various tolerances used by the solver:

  - **atol**

    Integrator absolute tolerance for the numerical solver.

  - **rtol**

    Integrator relative tolerance for the numerical solver.

  - **ttol**

    Tolerance on time.

    If an integration period is less than ttol, the numerical solver is not called.

  - **deltat**

    Maximum integration time interval.

    If an integration period is larger than deltat, the numerical solver is called more than once in such a way that for each call the integration period remains below deltat

  - **scale**

Real-time scaling; the value 0 corresponds to no real-time scaling.

It associates a Scicos simulation time to the real time in seconds.

A value of 1 means that each Scicos unit of time corresponds to one second.

- **solver**

  Choice of numerical solver.

  The value 0 implies an ODE solver and 100 implies a DAE solver.

- **hmax**

  Maximum step size for the numerical solver.

  0 means no limit.

Size : 7.

Type : column vector of real.

- **tf**

  Final time simulation.

  The simulation stops at this time.

  The default value is 100000.

  Size : 1.

  Type : real.

- **context**

  A vector of strings containing Scilab instructions defining Scilab variables to be used inside block's dialog box as symbolic parameters.

  All valid Scilab instructions can be used and also comments.

  Size : number of lines of the context.

  Type : column vector of strings.

- **void1**

  unused field.

  Size : -.

  Type : -.

- **options**

  Scilab object of type scsopt defining graphical properties of the editor such as background color and link color.

  The fields are the following:

- **3D**

  A list with two entries. The first one is a boolean indicating whether or not blocks should have 3D aspect.

  The second entry indicates the color in the current colormap to be used to create the 3D effect.

  The default is 33 which corresponds to gray added by Scicos to the standard colormap, which contains 32 colors.

  The default value is list(%t,33).

- **Background**

  Vector with two entries: background and foreground colors.

  The default value is [8,1].

- **link**

  Default link colors for regular and activation links.

  These colors are used only at link construction.

  Changing them does not affect already constructed links.

  The default value is [1,5], which corresponds to black and red if the standard Scilab colormap is used.

- **ID**

  A list of two vectors including font number and sizes.

  The default value is [5,1],[4,1].

- **Cmap**

  An n,3 matrix containing RGB values of colors to be added to the colormap.

  The default value is, [0.8,0.8,0.8] i.e., the color gray.

Size : 6.

Type : scilab tlist of type scsopt.

- **void2**

  unused field.

  Size : -.

  Type : -.

- **void3**

  unused field.

Size : -.

Type : -.

- **doc**

  User defined diagram documentation structure.

  Size : 1.

  Type : Strings.

## File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_params.sci

# 4. Links

# Nome

scicos_link — Define a link structure

# Module

- xcos

# link

Size : 8.

Type : scilab list.

- **xx**

  Vector of x coordinates of the link path.

  A link is defined as a polyline line.

  Size : number of points of the link.

  Type : column vector of real numbers.

- **yy**

  Vector of y coordinates of the link path.

  A link is defined as a polyline line.

  Size : number of points of the link.

  Type : column vector of real numbers.

- **id**

  Character string, the link identification.

  Size : 1.

  Type : string.

- **thick**

  Vector of size two defining line thickness.

  Size : 2.

  Type : row vector of integers.

- **ct**

  The first entry of this vector designates the color, and the second, the nature of the link. The second entry is 1 for a regular link, -1 for an activation link, and 2 for an implicit link.

  Size : 2.

  Type : row vector of integers.

- **from**

Vector of size three including the block number, port number, and port type (0 for output, 1 for input) at the origin of the link. Note that the third entry may be 1 if the link is implicit; otherwise it is zero.

Size : 3.

Type : row vector of integers.

- **to**

  Vector of size three including the block number, port number, and port type (0 for output, 1 for input) at the destination of the link. Note that the third entry may be 1 if the link is implicit; otherwise it is zero.

  Size : 3.

  Type : row vector of integers.

## File content

- SCI/modules/scicos/macros/scicos_scicos/scicos_link.sci

# Capítulo 10. Scilab Utilities Functions

# Nome

buildouttb — Build of the sublist %cpr.state.outtb

```
[outtb]=buildouttb(lnksz,lnktyp)
```

# Module

- xcos

# Description

Build an initialized outtb list.

```
[outtb]=buildouttb(lnksz,lnktyp)
```

# Parameters

- **outtb :** a list of size n.
- **lnksz :** That parameter gives the size of Scilab object in outtb.

  This matrix of integer or real numbers can have a n,2 or 2,n size.
- **lnktyp :** That parameters gives the type of Scilab object in outtb :

  1 : double

  2 : complex

  3 : int32

  4 : int16

  5 : int8

  6 : uint32

  7 : uint16

  8 : uint8

  else : double

  This matrix of integer or real numbers can have a n,1 or 1,n size.

# Authors

**Alan Layec** - INRIA

# Nome

create_palette — Palette generator

```
[routines,IntFunc] = create_palette(Path)
```

# Module

- xcos

# Description

This function generates a palette if Path is a string indicating the directoy in which the interfacing functions of the blocks are to be found. If Path is not present or if it is %t, then standard scicos palettes are regenerated. If %f, then only IntFunc (list of interfacing functions) is returned. List of routines is empty in this case.

```
[routines,IntFunc] = create_palette(Path)
```

# Parameters

- **Path :** a string given the directory path that contains Scicos blocks interfacing functions.
- **routines :** a vector of strings that contains names of computational functions used in scicos blocks.
- **IntFunc :** a vector of strings that contains names of interfacing functions used in scicos blocks.

# File content

- SCI/modules/scicos/macros/scicos_utils/create_palette.sci

# Nome

get_scicos_version — Get the current Scicos version

```
scicos_ver = get_scicos_version
```

# Module

- xcos

# Description

This function is used to know the current version number of Scicos.

```
scicos_ver = get_scicos_version
```

# Parameters

- **scicos_ver :** a string given the current number version of Scicos.

# Example

```
//Get the scicos version
get_scicos_version()
```

# File content

- SCI/modules/scicos/macros/scicos_utils/get_scicos_version.sci

# Authors

**Alan Layec** - INRIA

# Nome

scicos_debug — Set the level of the Scicos debugging

```
scicos_debug(level) level=scicos_debug()
```

# Module

- xcos

# Description

This Scilab function is used to set the debug level of a Scicos simulation.

One can used it in the "Calc" mode of the Scicos editor or as an instruction in a Scilab block or in an interfacing function.

```
scicos_debug(level)
level=scicos_debug()
```

# Parameters

- **level :** set/get the current level of the Scicos simulation debugging.
  - **0 :** no debugging.
  - **1 :** light debugging information printout.
  - **2 :** more information printout and execution of Debug Block if any in diagram.
  - **3 :** silent debugging mode (no information printout) and execution of Debug Block if any in diagram.

# See Also

- DEBUG_SCICOS - Debug block (Scicos Block)

# Authors

- **Alan Layec** INRIA
- **Ramine Nikoukhah** INRIA

# Nome

var2vec — Transform a scilab variable in a vector of double

```
[vec]=var2vec(var)
```

# Module

- xcos

# Description

var2vec / vec2var functions are used inside the interfacing functions of Scilab blocks to give the possibility to the user to handle Scilab objects with the real parameter (rpar) and with the discrete state register (z).

```
[vec]=var2vec(var)
```

# Parameters

- **var :** Input parameter. Can be any types of Scilab objects.
- **vec :** Output parameter. A vector of real numbers.

# Example

```
-->a=list("cos",[1.1,2])
a  =
a(1)
cos
a(2)
1.1    2.
-->b=var2vec(a)
b  =
4.244-314
1.273-313
8.488-314
2.122-314
4.941-324
8.488-314
5.093-313
2.122-314
2.122-314
9.881-324
1.1
2.
```

# See Also

- vec2var - Transform a vector of double in a scilab variable (Scilab Function)

# Nome

vec2var — Transform a vector of double in a scilab variable

```
[var]=vec2var(vec)
```

# Module

- xcos

# Description

var2vec / vec2var functions are used inside the interfacing functions of Scilab blocks to give the possibility to the user to handle Scilab objects with the register of real parameters (rpar) and with the discrete state register (z).

```
[var]=vec2var(vec)
```

# Parameters

- **vec :** Input parameter. A vector of real numbers.
- **var :** Output parameter. Can be any types of Scilab objects.

# Example

```
-->a=list("cos",[1.1,2])
a  =
a(1)
cos
a(2)
1.1    2.
-->b=var2vec(a)
b  =
4.244-314
1.273-313
8.488-314
2.122-314
4.941-324
8.488-314
5.093-313
2.122-314
2.122-314
9.881-324
1.1
2.
-->c=vec2var(b)
c  =
c(1)
cos
c(2)
1.1    2.
```

# See Also

- var2vec - Transform a scilab variable in a vector of double (Scilab Function)

# Nome

xcos — Block diagram editor and GUI for the hybrid simulator

```
xcos(filename)
xcos(scs_m_list)
```

# Module

- xcos

# Description

Xcos is a visual editor for constructing models of hybrid dynamical systems.

Invoking Xcos with no argument opens up an empty Xcos window. Models can then be assembled, loaded, saved, compiled, simulated, using GUI of Xcos. Xcos serves as an interface to the various block diagram compilers and the hybrid simulator scicosim.

```
xcos('mydiagram.xcos')
```

# Parameters

- **filename :** a character string containing the path of the diagram file (.cos, .cosf or .xcos extension). If no input argument is used, an empty diagram is opened (default name Untitled).
- **scs_m_list :** a Xcos diagram structure after edition.

# Example

```
// Open a new diagram
xcos();
xcos

// Load a diagram
xcos SCI/scicos/demos/bounce.cosf;

// Load a structure
load('mondiagr.cos');
xcos(scs_m);
```

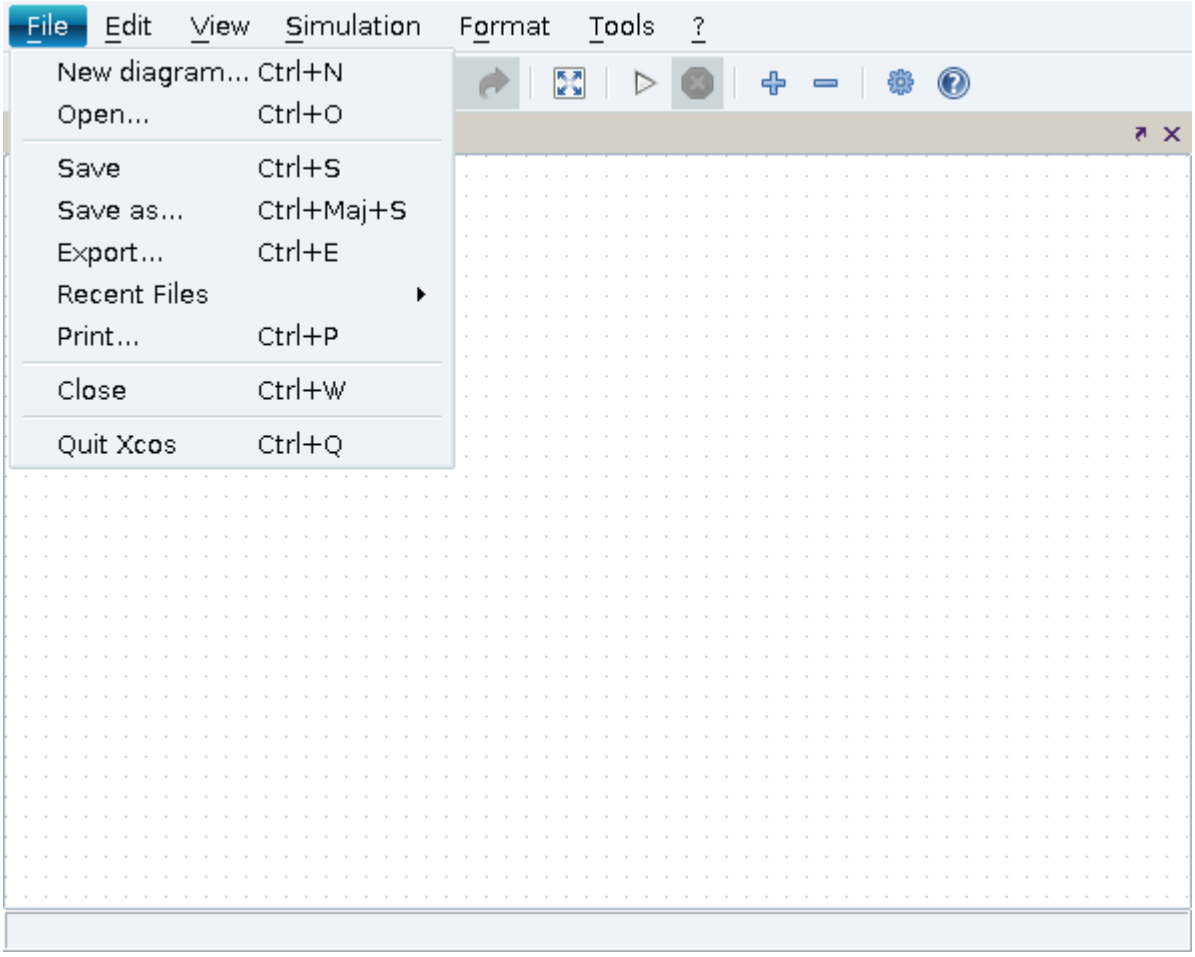
# See Also

- scicosim - Scicos (batch) simulation function (Scilab Function)
- scicos_simulate - Function for running xcos simulation in batch mode (Scilab Function)
- Menu entries

# Nome

Menu_entries — Editor menu entries

# File menu




- **File:New**

  Clicking on the New menu item loads an empty diagram in the active editor xcos window. If the previous content of the window is not saved, it will be lost. With this menu, you can open a new diagram or a new palette.

- **File:Open (Ctrl+o)**

  Select the Open menu item to load an ASCII or binary file containing a saved block diagram or palette. A dialog box allows user choosing the file.

- **File:Save (Ctrl+s)**

  Select the save menu item to save the block diagram in a binary file already selected by a previous select the Save As menu item. If you select this menu item and you have never clicked on the Save As menu item, the diagram is saved in the current directory as "window_name".cos where "window_name" is the name of the window appearing on top of the window (usually Untitled or Super Block). The .cos binary files are machine independent.

- **File:Save As (Ctrl+Shift+s)**

  Select the Save As menu item to save the block diagram or palette in a file. A dialog box allows choosing the file which must have a .cos or .cosf extension. The diagram takes the name of the file

(without the extension). If extension is ".cosf" an ASCII formatted save is performed instead of binary save. Formatted save is slower than regular save.

- **File:Export (Ctrl+e)**

  This menu is used to export a figure of the current xcos diagram. The export can be done directly in postscript format or done first in a graphic window to export in a second step in all the format that scilab can provide.

- **File:Recent Files**

  Via this menu, you have a quick access to the recent opened files.

- **File:Print (Ctrl+p)**

  Print the current diagram onto a printer.

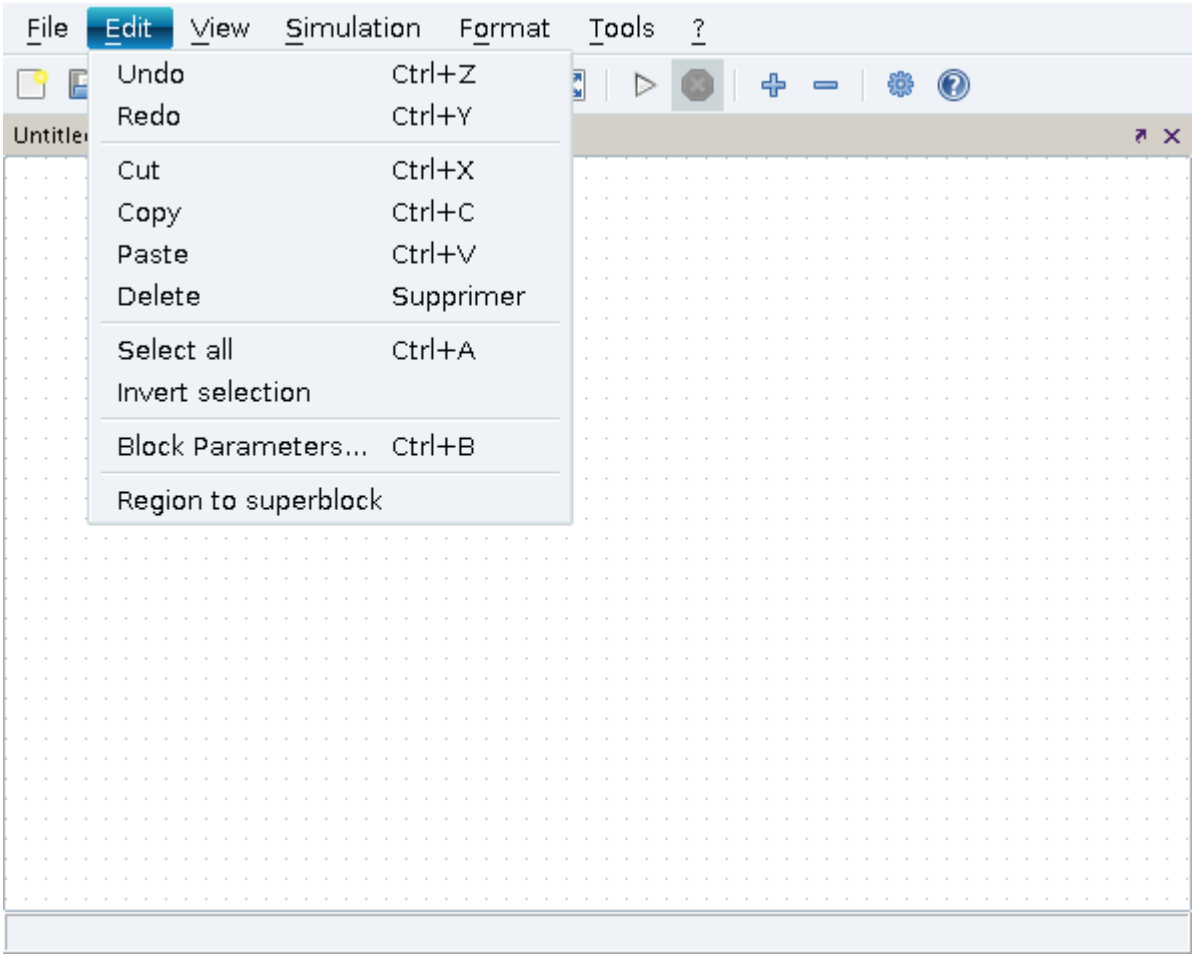
- **File:Close (Ctrl+w)**

  If several diagram are opened, the Close action closes the current diagram. If only one diagram is opened, the Close action will close xcos and closes the viewport and palettes windows if these windows are opened.

- **File:Quit (Ctrl+q)**

  If several diagram are opened, the Quit action will close xcos and closes the viewport and palettes windows if these windows are opened. It will close all the opened diagram.

# Edit menu



- **Edit:Undo (Ctrl+z)**

  Select the Undo menu item to undo the last edit operation.

- **Edit:Redo (Ctrl+y)**

  Select the Redo menu item to redo the last undo edit operation.

- **Edit:Cut (Ctrl+x)**

  Cut is used to remove the selected object from the diagram and keep a copy in the clipboard if the object is a block.

- **Edit:Copy (Ctrl+c)**

  Copy is used to place a copy of the selected object in the clipboard if the object is a block.

- **Edit:Paste (Ctrl+v)**

  Paste places the object in the Clipboard in the diagram.

- **Edit:Delete (Delete)**

  To delete blocks or a links, select first the Delete menu item, then click successively on the selected objects (with left button). When you delete a block all links connected to it are deleted as well.

- **Edit:Select all (Ctrl+a)**

  Select all the blocks in the current diagram.

- **Edit:Invert selection**

  Invert the current selection.

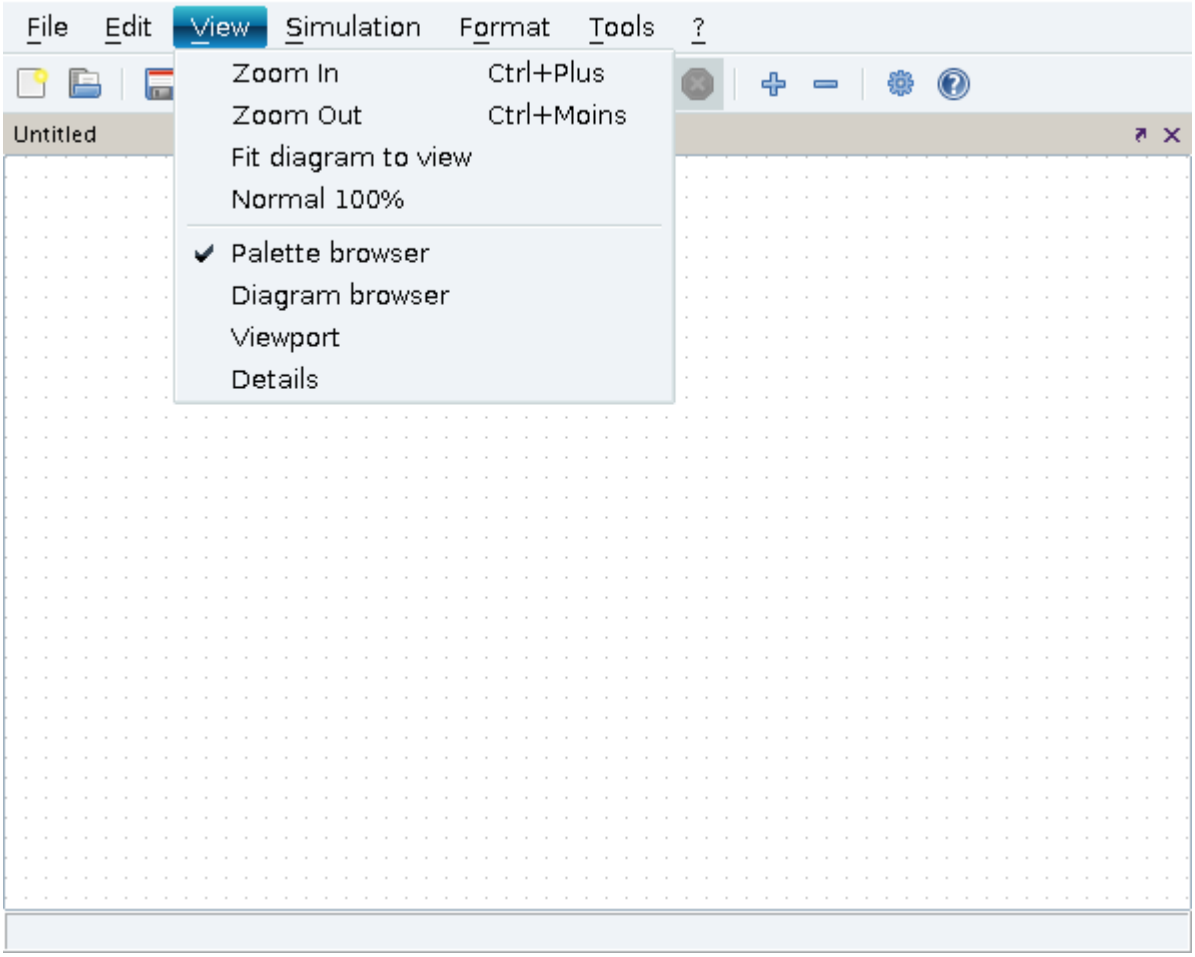
- **Edit:Block Parameters (Ctrl+b)**

  Open the block configuration window for the current selected block.

- **Edit:Region to superblock**

  Convert a selection of blocks into a superblock.

# View menu




- **View:Zoom in (Ctrl+plus)**

  When you select this menu item the diagram is zoomed in by a factor of 10%.

- **View:Zoom out (Ctrl+minus)**

  When you select this menu item the diagram is zoomed out by a factor of 10%.

- **View:Fit diagram to view**

  When you select this menu item the diagram is fit to the size of the current window.

- **View:Normal 100%**

  Resize the work area so as the diagram fits onto this work area.

- **View:Palette browser**

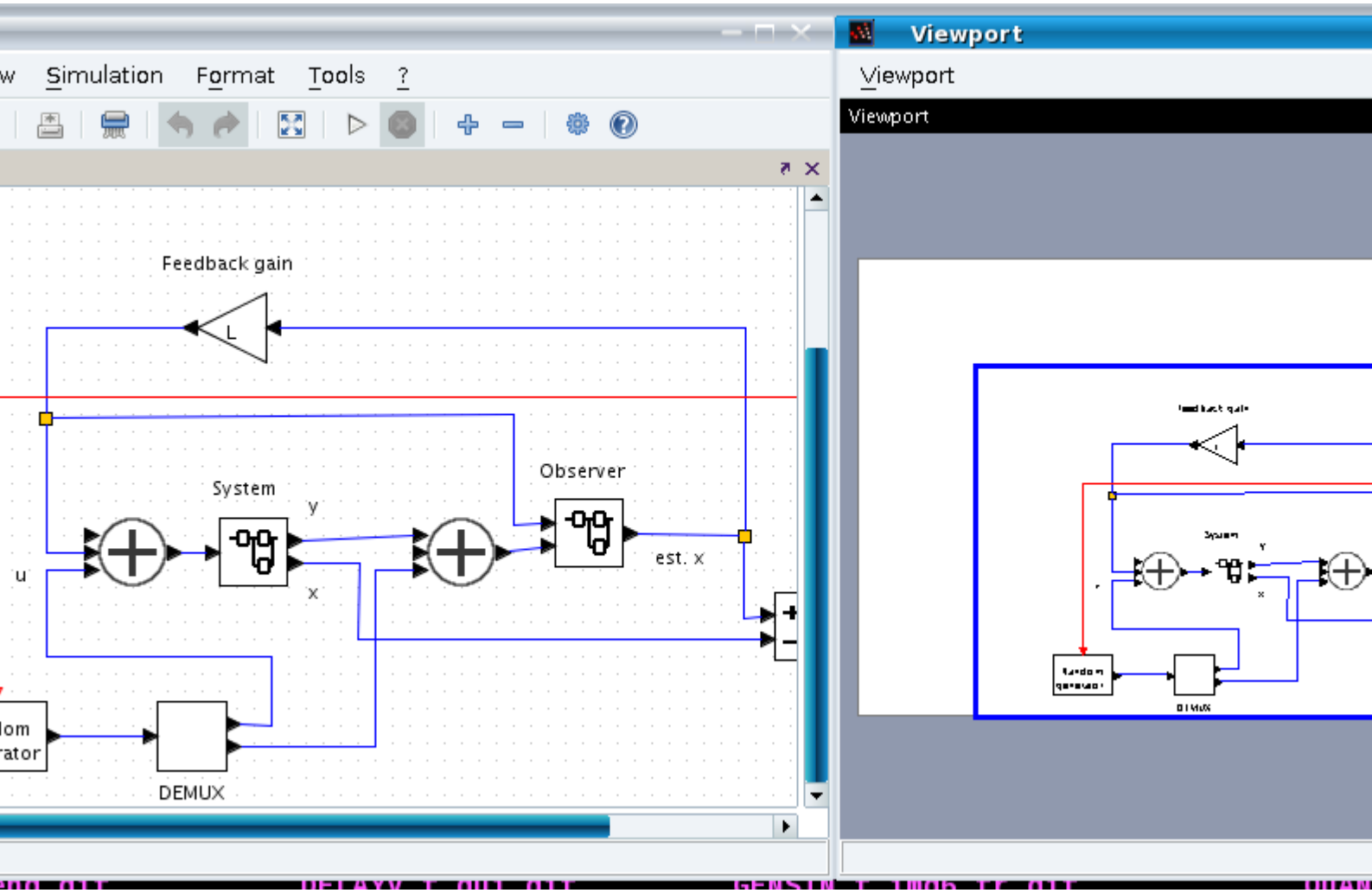
  Open the palette browser.

- **View:Diagram browser**

  Displays a window which lists all the blocks of a diagram and print some informations related to the scs_m structure of the blocks.

- **View:Viewport**

Display the viewport. The viewport is an image of the current diagram. With the viewport, you can move the working area onto a piece of the diagram. You can zoom and unzoom part of a diagram.

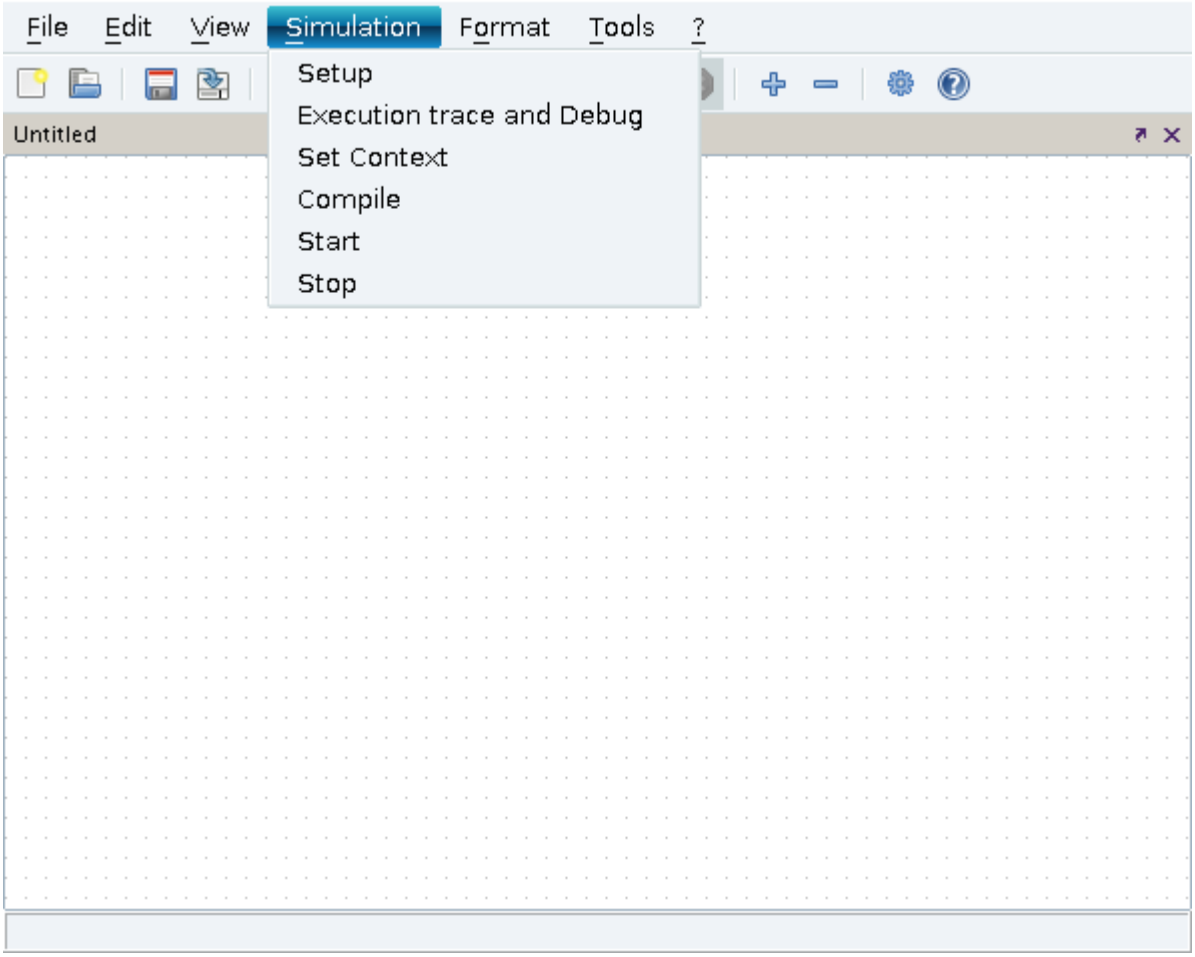



Above, you have an example of the viewport which is used to zoom on a part of a diagram, and on the right, the xcos window displays the zoomed part of the diagram.

- **View:Details**

  Displays a window which lists all the selected blocks of a diagram and print some informations related to the scs_m structure of these blocks.

# Simulation menu




- **Simulation:Setup**

  In the main Xcos window, clicking on the Setup menu item invokes a dialog box that allows you to change integration parameters:

  - Final integration time (integration ends at this time, it always starts from 0)
  - Real time scaling (forces real time simulation by setting xcos unit of time to 1 second)
  - Absolute and relative error tolerances (solver properties)
  - Time tolerance (the smallest time interval for which the ode solver is used to update continuous states)
  - Max integration time interval (the maximum time interval for each call to solver, it must be reduced if the error message "too many calls" is encountered)
  - Solver (choose the numerical solver to be used), Max step size (max time step taken by solver)

- **Execution trace and Debug**

  Set Xcos in debug mode. Allows diagram debugging.

- **Simulation:Set Context**

  When you select this menu item you obtain a dialog to enter Scilab instructions for defining symbolic Xcos parameters used in block definitions or to do whatever you want. These instructions will be

evaluated each time the diagram is loaded. If you change the value of a symbolic Xcos parameters in the context, all the blocks are updated (Eval is performed).

- **Simulate:Compile**

  Select the Compile menu item to compile the block diagram. This menu item need never be used since compilation is performed automatically, if necessary, before the beginning of every simulation (Run menu item). Normally, a new compilation is not needed if only system parameters and internal states are modified. In some cases however these modifications are not correctly updated and a manual compilation may be needed before a Restart or a Continue. Please report if you encounter such a case.

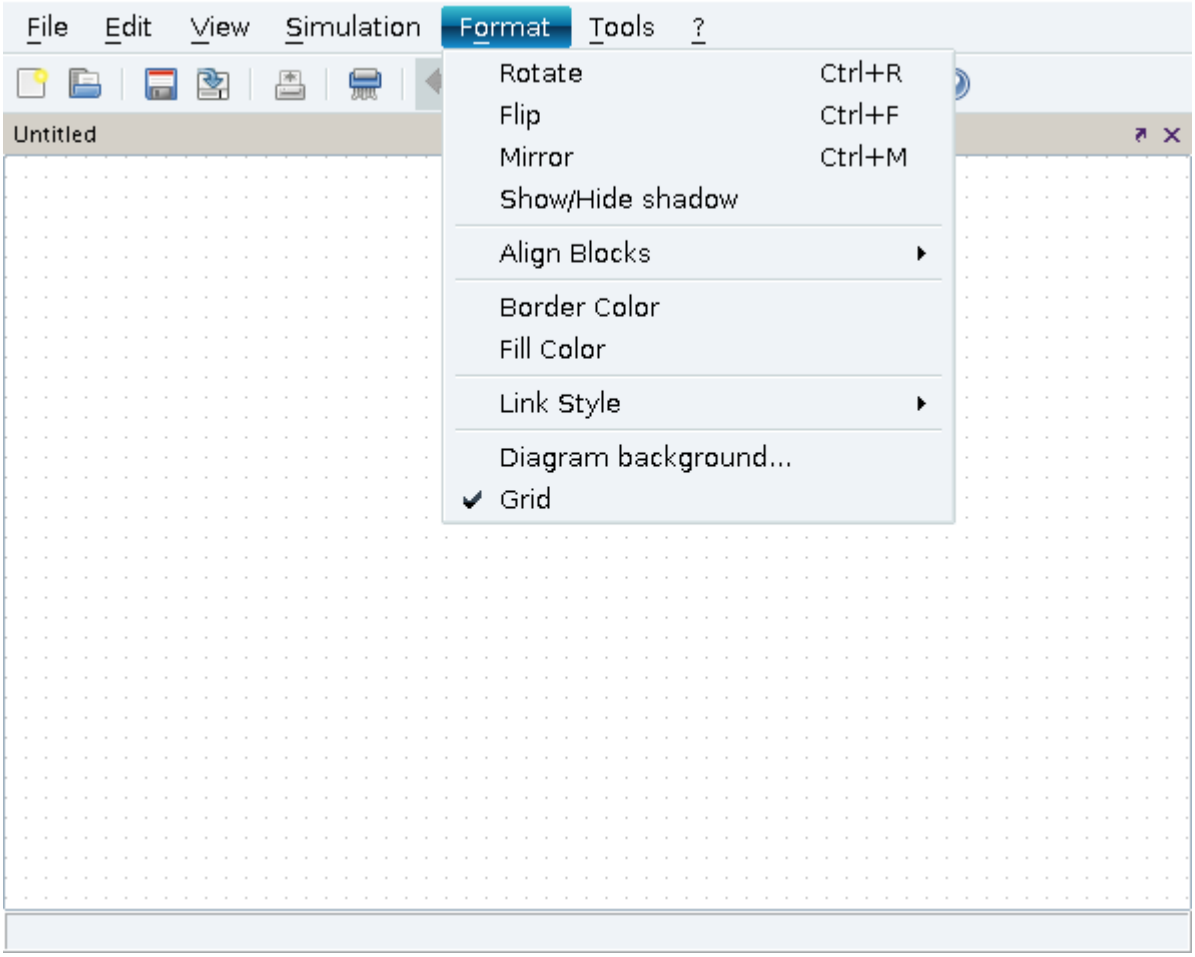
- **Simulate:start**

  Select the Run menu item to start the simulation. If the system has already been simulated, a dialog box appears where you can choose to Continue, Restart or End the simulation.

- **Simulation:Stop**

  You may interrupt the simulation by clicking on the "stop" button, change any of the block parameters and continue the simulation with the new values.

# Format menu




- **Format:Rotate (Ctrl+r)**

  Rotate allows to turn a block on the Left. Each time the block is turned left, his angle is decresead of 45 degrees. If no blocks or many blocks are selected, this is the block under the mouse pointer which turns.

- **Format:Flip (Ctrl+f)**

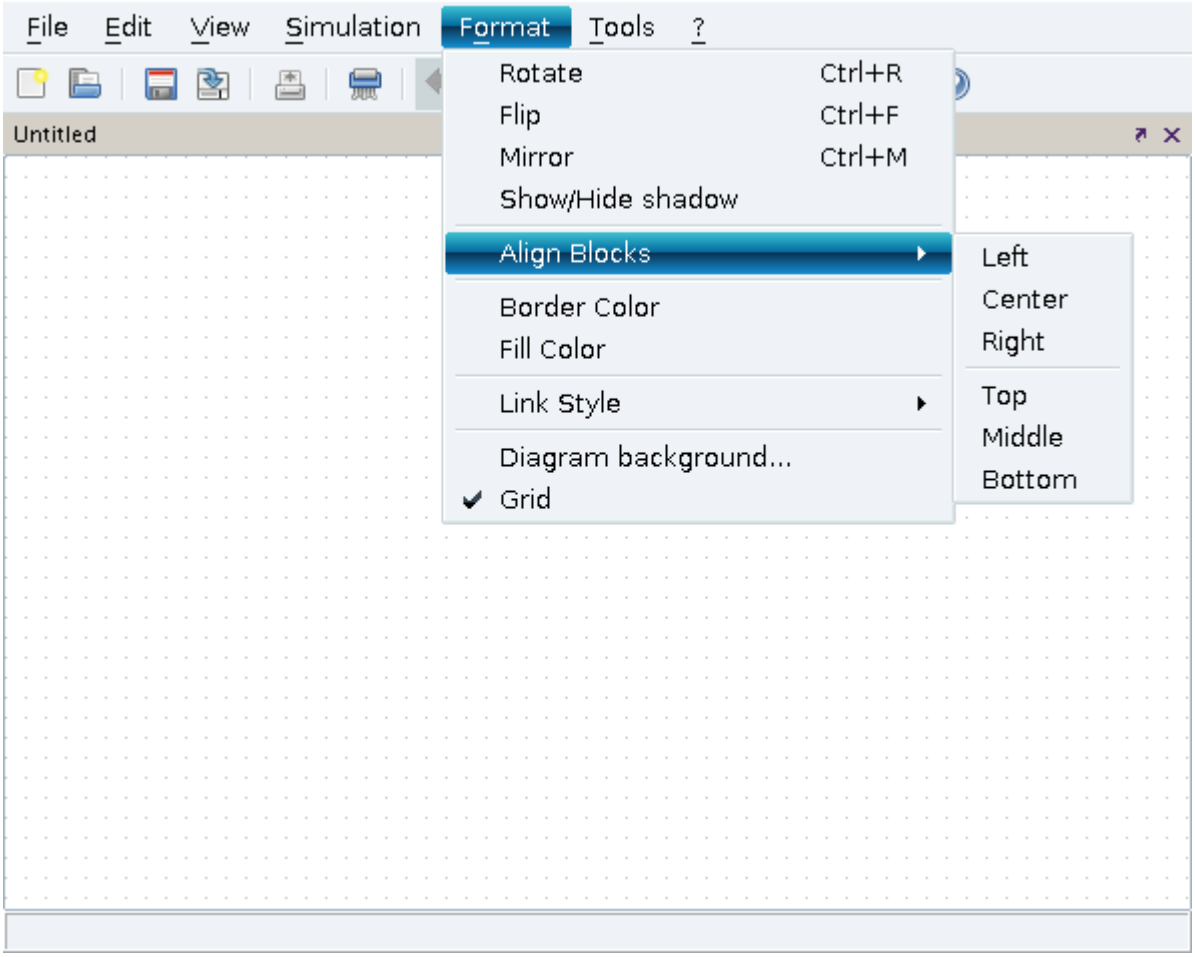
  To reverse the positions of the (regular) inputs and outputs of a block placed on its sides, select the Flip menu item first and then click on the selected block. This does not affect the order, nor the position of the input and output event ports which are numbered from left to right.

- **Format:Show/Hide shadow**

  This menu allows to select 3D shape for selected blocks and associated parameters.

- **Format:Align Blocks**

  When you select several blocks, it is possible to align them vertically (Top, Bottom and Middle) and horizontally (Left, Right, Center).



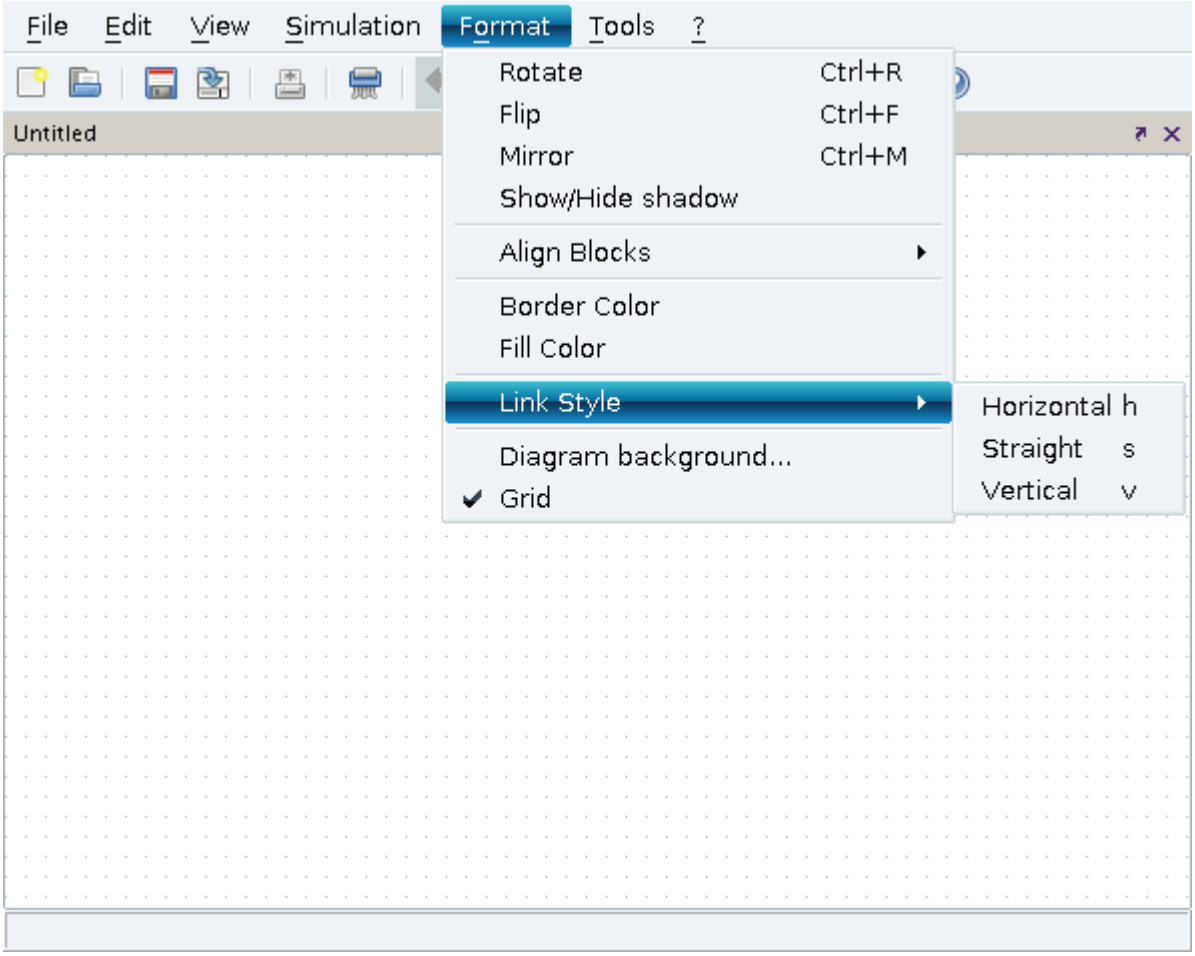

- **Format: Border Color**

  This menu allows to change the border color.

- **Format:Line Color**

  This menu allows to change the line color.

- **Format:Link Style**




This menu allows to the the style of the link:

- Horizontal
- Straight
- Vertical

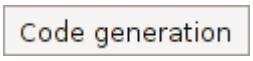
- **Format:Diagram background**

  This menu allows to change the background color.

- **Format:Grid**

  This menu allows to activate / desactivate the grid. Using a grid, it is easier to place a block on the working area.

# Tools menu

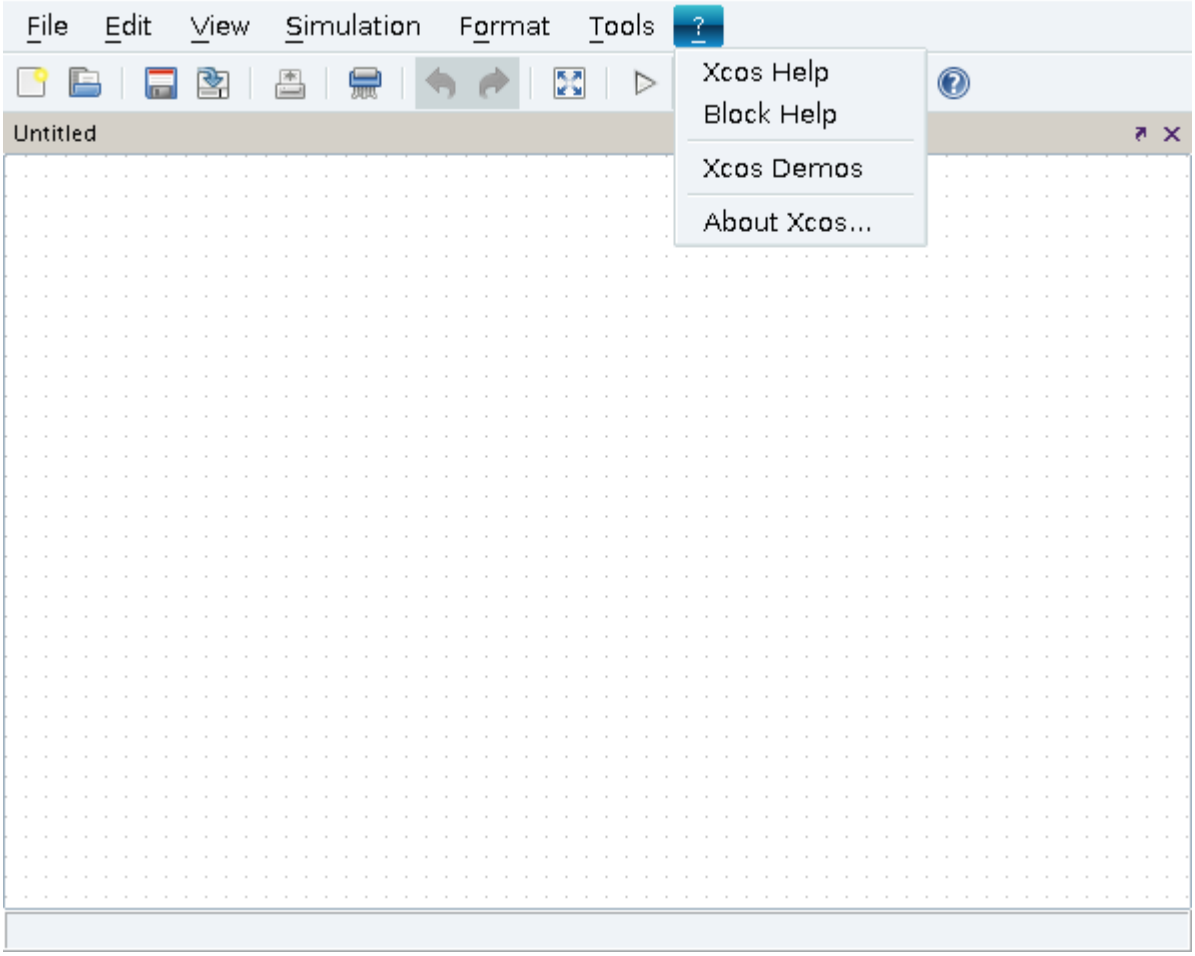



- **Tools:Code generation**

  This menu allows to generate the simulation code associated with a discrete time Super Block. The code generation is obtained simply by selecting this menu and then the desired Super Block. If the Super Block satisfies the required conditions, a dialog box pops up to ask for a block name, a directory where to put the generated files and for optional libraries requested by the linker. Given this information the code is generated, compiled and linked with Scilab. The Super Block is auto-

matically replaced by a new block which implements the generated code. It is then possible to run the modified diagram. The code for standalone use is also generated.

# Help menu




- **Help:Xcos Help**

  This menu opens the help browser.

- **Help:Block Help**

  To get help on a Xcos object (a block), select first an object (a block) and then click on this menuHelp menu item and then on the selected object or menu item.

- **Help:Xcos Demos**

  The Demos menu allows to open some examples of Xcos diagram.

- **Help:About Xcos**

  About xcos item display the current version of Xcos and gives some useful informations.

# Parte LVI. scilab editor

# Name

edit_error — opens in scilab editor the source of the last recorded error

```
answ = edit_error(clearerror)
```

# Parameters

clearerror
: boolean - if true the error condition is cleared, if false it is kept (as in lasterror)

answ
: a string stating which source file is open (or why no file was open)

# Description

This function opens in scilab editor the source of the function which caused the last recorded error.

This function works only for functions which are defined in libraries, i.e. not for internal functions, nor with functions defined online, nor loaded with individual exec or getd. This is since Scilab presently retains only the path to libraries and not to individual function sources.

Correspondance between the function name foo and function filename foo.sci is tacitly assumed.

# Examples

```
acosh abc
edit_error
```

# See Also

lasterror , errclear

# Authors

Enrico Segre

Allan CORNET

# Name

Editor — Embedded Scilab text editor

```
editor()
editor(file)
editor([file1, file2])
editor(file, line_number)
editor([file1, file2], [line_number1, line_number2])
```

# Parameters

file
: a string, the file we want to open.

[file1, file2]
: a matrix of string, files we want to open.

line_number
: an integer, number of the line we want to highlight at the opening of the file.

[line_number1, line_number2]
: a matrix of integer, each opened file will have it's corresponding line highlighted.

# Description

`Editor` is an embedded Scilab text editor.

It can be started with a fresh text buffer pressing the "Editor" button on top of the main Scilab window, or from Scilab command line with the instruction `editor()`, or it can open specific files if invoked with any of the calling sequences above (whithout any parameters, it opens editor with a blank file).

The same invocation adds further files to an already opened Editor.

Keyboard shortcuts are defined for most possible editing actions and reported by the menu entries.

editor can be started in the following ways :

- By the menu Applications. Choose Applications => Editor
- From the command line:
  - editor()
  - editor(file)
  - editor([file1, file2])
  - editor(file, line_number)
  - editor([file1, file2], [line_number1, line_number2])

# Menus and Shortcuts

- Menu File

| Commande | Shortcut | Description |
|---|---|---|
| New... | <CTRL-N> | Open a new file |
| Open... | <CTRL-O> | Open an existing file |
| Recent Files | | Display files recently opened |
| Save | <CTRL-S> | Save a file |
| Save as... | <CTRL-MAJ-S> | Save a file as |
| Page Setup | | Setup page for printing |
| Print Preview | <CTRL-MAJ-P> | Open a print preview window |
| Print... | <CTRL-P> | Print a file |
| Close | <CTRL-W> | Close a file |
| Quit | <CTRL-Q> | Close Editor |

- Menu Edit

| Commande | Shortcut | Description |
|---|---|---|
| Undo | <CTRL-Z> | Undo action |
| Redo | <CTRL-Y> | Redo action |
| Cut | <CTRL-X> | Cut the selection |
| Copy | <CTRL-C> | Copy the selection |
| Paste | <CTRL-V> | Paste the selection |
| Select All | <CTRL-A> | Select the entire document |
| Delete | | Delete the selection |
| Comment Selection | <CTRL-D> | Comment selected lines |
| Uncomment Selection | <CTRL-MAJ-D> | Uncomment selected lines |
| Tabify Selection | <TAB> | Tabify selected lines |
| Untabify Selection | <MAJ-TAB> | Untabify selected lines |
| Indent | <CTRL-I> | Indent selected lines |

- Menu Search

| Commande | Shortcut | Description |
|---|---|---|
| Find/Replace | <CTRL-F> | Find and/or Replace an element |
| Goto line | <CTRL-G> | Goto line |

- Menu View

| Commande | Shortcut | Description |
|---|---|---|
| Show/Hide Toolbar | | Option to show or hide the tool-bar |
| Highlight current line | <CTRL-J> | Highlight the current line |
| Line Numbers | <CTRL-B> | Display document's line num-bers |
| Set Colors... | | Color settings for documents |
| Set Fonts... | | Font settings for documents |
| Reset default font | | Reset default font settings for documents |

- Menu Document

| Commande | Shortcut | Description |
|---|---|---|
| Syntaxe Type | | Syntaxe type settings (default type is Scilab) |
| Encoding | | Encoding settings (default type is UTF-8 Encoding) |
| Colorize | | Colorize the document |
| Auto Indent | | Activate the automatic indentation |

- Menu Execute

| Commande | Shortcut | Description |
|---|---|---|
| Load Into Scilab | <CTRL-L> | Load the entire document into the Scilab console |
| Evaluate Selection | | Load the selection into the Scilab console |
| Execute Into Scilab | <CTRL-E> | If the file exist, execute the content of the file |

# Remarks

Document :

The default text colourisation is the Scilab's syntaxe colourisation.

The `auto-indent` mode indent a line according to Scilab's syntaxe (after a return action).

Bugs:

You can reported bugs in the Bugzilla http://bugzilla.scilab.org

There are still a few bugs that we are trying to fix, details can be found in the given link by filtering entries with the "Editor" element.

# Examples

```
// editor without parameters
editor();

// editor with a file name
editor('SCI/modules/time/macros/datenum.sci');

// editor with a matrix of files name
editor(['SCI/modules/time/macros/datenum.sci','SCI/modules/time/macros/datevec.

// editor with a file name and the line number to highlight
editor('SCI/modules/time/macros/datenum.sci', 5);

// editor with a matrix of files name and the corresponding matrix of lines to
// the files name matrix and the lines to highlight matrix should have the same
editor(['SCI/modules/time/macros/datenum.sci','SCI/modules/time/macros/datevec.
```

# Author

Sylvestre KOUMAR

# Parte LVII. API Scilab

# Índice


11. Scilab Gateway API ..... 2989
    1. How to ..... 2989
    CheckColumn ..... 3011
    CheckDimProp ..... 3012
    CheckDims ..... 3013
    CheckLength ..... 3014
    CheckLhs ..... 3015
    CheckRhs ..... 3016
    CheckRow ..... 3017
    CheckSameDims ..... 3018
    CheckScalar ..... 3020
    CheckSquare ..... 3021
    CheckVector ..... 3022
    CreateListVarFrom ..... 3023
    CreateListVarFromPtr ..... 3026
    CreateVar ..... 3029
    FindOpt ..... 3031
    FirstOpt ..... 3033
    GetListRhsVar ..... 3035
    GetRhsVar ..... 3037
    GetType ..... 3039
    IsOpt ..... 3040
    Lhs ..... 3042
    LhsVar ..... 3043
    NumOpt ..... 3045
    OverLoad ..... 3047
    Rhs ..... 3049
    Scierror ..... 3050
    Scilab C Types ..... 3051
    get_optionals ..... 3053
    istk ..... 3055
    sci_types ..... 3056
    sciprint ..... 3058
    stk ..... 3059
12. list_management ..... 3060
    Boolean reading (Scilab gateway) ..... 3061
    Boolean writing (Scilab gateway) ..... 3073
    Boolean sparse reading (Scilab gateway) ..... 3077
    Boolean sparse writing (Scilab gateway) ..... 3089
    Create List (Scilab gateway) ..... 3093
    Double reading (Scilab gateway) ..... 3097
    Double writing (Scilab gateway) ..... 3109
    Get child item (Scilab gateway) ..... 3113
    Item Number (Scilab gateway) ..... 3116
    Integer reading (Scilab gateway) ..... 3119
    Integer writing (Scilab gateway) ..... 3131
    Pointer reading (Scilab gateway) ..... 3136
    Pointer writing (Scilab gateway) ..... 3148
    Polynomial reading (Scilab gateway) ..... 3152
    Polynomial writing (Scilab gateway) ..... 3164
    Sparse reading (Scilab gateway) ..... 3168
    Sparse writing (Scilab gateway) ..... 3180
    String reading (Scilab gateway) ..... 3184
    String writing (Scilab gateway) ..... 3196

# Capítulo 11. Scilab Gateway API

## 1. How to

# Name

Calling a scilab function (macros) from a C gateway — Calling a scilab function (macros) from a C interface

## Calling a scilab function (macros) from a C interface

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

When you write a interface with scilab, you can need to call another function directly from your function. You can pass a pointer on scilab function to your function.

The source files of this example are stored in the directory examples/call_scifunction.

How to use this example:

```
exec call_scifunction.sce; // launch scilab
v = call_scifunc(30,12,scilabfoo);
```

you pass a pointer on a scilab function (scilabfoo macro) to your function 'call_scifunc'.

C2F(scifunction) calls another scilab function (here scilabfoo).

You need to indicate :

- position of the first element (Rhs) in the Scilab memory
- pointer on scilab function
- number of Lhs (output of scilab function called)
- number of Rhs (input of scilab function called)

The script call_scifunction.sce used to build and load the C interface into Scilab is the following:

```
files=['sci_call_scifunc.c']; // Defines the list of file
ilib_build('callscifunc',['call_scifunc','sci_call_scifunc'],files,[]); // Buil

exec loader.sce;

function r = scilabfoo(x,y)
  r = x + y;
endfunction

v = call_scifunc(30,12,scilabfoo);
disp('result : ' + string(v));

v = call_scifunc(300,120,scilabfoo);
disp('result : ' + string(v));

ulink(); // unload the dynamic library
```

The file sci_call_scifunc.c is the following:

```
#include "stack-c.h"
#include "Scierror.h"
#include "localization.h"
```

```
int sci_call_scifunc(char *fname)
{
  int m1    = 0, n1    = 0, l1    = 0;
  int m2    = 0, n2    = 0, l2    = 0;
  int m3    = 0, n3    = 0, l3    = 0;
  int rm1   = 0, rn1   = 0, rl1   = 0;
  int m_out = 1, n_out = 1, l_out = 0;
  double v1 = 0, v2 = 0, r = 0;
  int positionFirstElementOnStackForScilabFunction = 0;
  int numberOfRhsOnScilabFunction = 0;
  int numberOfLhsOnScilabFunction = 0;
  int pointerOnScilabFunction     = 0;

  CheckRhs(3,3);
  CheckLhs(1,1);

  if (GetType(1) != sci_matrix)
  {
    Scierror(999,_("%s: Wrong type for input argument #%d: A real expected.\n")
    return 0;
  }

  GetRhsVar(1, MATRIX_OF_DOUBLE_DATATYPE, &m1, &n1, &l1);
  if ( (m1 == n1) && (n1 == 1) )
  {
    v1 = *stk(l1);
  }
  else
  {
    Scierror(999,_("%s: Wrong size for input argument #%d: A scalar expected.\n
    return 0;
  }

  if (GetType(2) != sci_matrix)
  {
    Scierror(999,_("%s: Wrong type for input argument #%d: A real expected.\n")
    return 0;
  }

  GetRhsVar(2, MATRIX_OF_DOUBLE_DATATYPE, &m2, &n2, &l2);
  if ( (m2 == n2) && (n2 == 1) )
  {
    v2 = *stk(l2);
  }
  else
  {
    Scierror(999,_("%s: Wrong size for input argument #%d: A scalar expected.\n
    return 0;
  }

  if (GetType(3) != sci_c_function)
  {
    Scierror(999,_("%s: Wrong type for input argument #%d: A scilab function ex
    return 0;
  }

  // get pointer on external function (here scilabfoo)
  GetRhsVar(3, EXTERNAL_DATATYPE, &m3, &n3, &l3);
```

```
  // r = scilabfoo(x, y)
  // rhs eq. 2
  // lhs eq. 1

  // Position first element in the Scilab memory to use by Scilab Function
  // v = call_scifunc(300,120,scilabfoo);
  // On stack : 300 is on Top position (1)
  // 120 second position
  // scilabfoo third position
  // we want to pass 300 & 120 to scilab Function
  // First position is here : 1

  positionFirstElementOnStackForScilabFunction = 1;
  numberOfRhsOnScilabFunction = 2;
  numberOfLhsOnScilabFunction = 1;
  pointerOnScilabFunction     = l3;

  // r = scilabfoo(x, y)
  // Scifunction call a scilab function
  Scifunction(&positionFirstElementOnStackForScilabFunction,
              &pointerOnScilabFunction,
              &numberOfLhsOnScilabFunction,
              &numberOfRhsOnScilabFunction);

  // result r is now on position positionFirstElementOnStackForScilabFunction o
  GetRhsVar(1, MATRIX_OF_DOUBLE_DATATYPE, &rm1, &rn1, &rl1);
  r = *stk(rl1);

  CreateVar(Rhs+1, MATRIX_OF_DOUBLE_DATATYPE, &m_out, &n_out, &l_out);
  *stk(l_out) = r;

  LhsVar(1) = Rhs + 1;

  return 0;
}
```

The main function in this C file is Scifunction. It allows to call a Scilab function inside a C interface.

## See Also

CheckLhs, CheckRhs, stk, LhsVar, GetType, Scierror, Rhs, Lhs, sci_types

# Nome

How to access a matrix — How to access a matrix using the C gateway functions

## Description

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

The goal is to get a matrix of doubles send to a function written in C.

For this, we will wrote a C gateway function in which we will retrieve the matrix, we will perform some simple steps in this C function:

- First, we will get an access to the matrix in the Scilab memory
- We will perform some simple operations on the matrix (in this example, we will multiply by 2 each elements of the matrix)
- We will return the result to Scilab

This example is available in the directory SCI/modules/core/examples/ex1.

## The C function

```
#include <stack-c.h>

int sci_multiply_by_two(char * fname)
{
  int m_in_var, n_in_var, l_in_var;
  int m_out_var, n_out_var, l_out_var;
  int i_row, j_col;
  double * pMatrix = NULL;

  // First, access to the input variable (a matrix of doubles)
  GetRhsVar(1, MATRIX_OF_DOUBLE_DATATYPE, &m_in_var, &n_in_var, &l_in_var);

  // Create the returned variable (a matrix of doubles)
  m_out_var = m_in_var;
  n_out_var = n_in_var;
  CreateVar(2, MATRIX_OF_DOUBLE_DATATYPE, &m_out_var, &n_out_var, &l_out_var);

  pMatrix = stk(l_in_var);

  // Perform some simple operations on the matrix
  for(i_row=0; i_row<m_in_var; i_row++)
    {
      for(j_col=0; j_col<n_in_var; j_col++)
        {
          pMatrix[i_row + j_col * m_out_var] = 2 * pMatrix[i_row + j_col * m_in
        }
    }

  // Return the output variable
  LhsVar(1) = 2;

  return 0;
}
```

This file must be saved as "multiply_by_two.c".

The main thing to highlight is that, to build a C gateway function, we need to include the header stack-c.h. In this header, we find the prototypes and macros of the main C gateway functions.

To be able to build and link such a C function to scilab, we need to write a Scilab script which will compile this C function and then create a loader script which will link the C function to a Scilab function.

## The builder script

```
// This is the builder.sce
// must be run from this directory

lines(0);

ilib_name  = 'lib_multiply_by_two';

files = ['multiply_by_two.c'];

libs  = [];

table =['multiply_by_two', 'sci_multiply_by_two'];

ldflags = "";
cflags  = "";
fflags  = "";

ilib_build(ilib_name,table,files,libs,'Makelib',ldflags,cflags,fflags);
```

This file must be saved as "builder.sce".

This script will tell Scilab which files must be compiled (here, it's multiply_by_two.c), what will be the name of the shared library (here, it's lib_multiply_by_two) and which C symbol will be linked to a Scilab function (here, we will link the sci_multiply_by_two C symbol to the Scilab function "multiply_by_two").

To build this function, we just need to to:

```
exec builder.sce;
```

Now we are able to test our new C function. First, let's load this new function in scilab:

```
exec loader.sce;
```

The script loader.sce is normally automatically built by builder.sce.

## Testing our new function

We now write a simple example to test our new function.

```
A = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15];
```

```
B = multiply_by_two(A);

disp(B);
```

The script must be saved as "test.sce".

Let's run our scripts and see what is the result:

```
-->exec builder.sce;
   Generate a gateway file
   Generate a loader file
   Generate a Makefile
   ilib_gen_Make: Copy compilation files (Makefile*, libtool...) to TMPDIR
   ilib_gen_Make: Copy multiply_by_two.c to TMPDIR
   ilib_gen_Make: Copy lib_multiply_by_two.c to TMPDIR
   ilib_gen_Make: Modification of the Makefile in TMPDIR.
   Running the makefile

-->exec loader.sce;
Shared archive loaded.
Link done.

-->exec test.sce;

    2.     4.     6.     8.     10.
    12.    14.    16.    18.    20.
    22.    24.    26.    28.    30.
-->
```

This simple function has produced a new matrix which corresponds to the matrix transmitted as an input argument and for which each element of the matrix has been multiplied by 2.

## Rebuilding a gateway function

Let's imagine that our gateway function has already been build and we would like to make some changes in our function (multiply by 3 instead of 2).

How do we perform such a changes without restarting Scilab ?

First, we need to list all the dynamic libraries which has been loaded into Scilab. The can be done using the link('show') function:

```
-->link('show')
Number of entry points 1.
Shared libraries :
[ 0 ] : 1 libraries.
Entry point lib_multiply_by_two in shared library 0.
 ans  =

    0.
```

Here, in our current Scilab session, only 1 dynamic library has been loaded. This library has a reference number. For our library, it's "0". Now that we know the reference number of our library, we are able to:

- unload this library (using the function ulink(0) - 0 is the reference number of our library)

- perform some modification in the source code of our C gateway function (replace multiply by 2 by 3)
- rebuild the C gateway function (exec builder.sce;)
- load the modified C gateway function into scilab (exec loader.sce;)

This is what is done is the following example:

```
-->ulink(0)

-->exec builder.sce;
   Generate a gateway file
   Generate a loader file
   Generate a Makefile
   ilib_gen_Make: Copy compilation files (Makefile*, libtool...) to TMPDIR
   ilib_gen_Make: Copy multiply_by_two.c to TMPDIR
   ilib_gen_Make: Copy lib_multiply_by_two.c to TMPDIR
   ilib_gen_Make: Modification of the Makefile in TMPDIR.
   Running the makefile

-->exec loader.sce;
Shared archive loaded.
Link done.

-->exec test.sce;

    3.     6.     9.     12.    15.
    18.    21.    24.    27.    30.
    33.    36.    39.    42.    45.
```

## See Also

GetRhsVar, Scilab C Types, CreateVar, LhsVar, stk, ilib_build, link, ulink

# Nome

How to check parameters — how to check parameter send to an interface using the C gateway functions

## Description

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

The goal is to get a set of parameters via a set of C gateway functions and then to perform some checks in the C function.

This example is available in the directory core/examples/check_properties.

## The C function

```
#include <stack-c.h>
#include <sciprint.h>

int sci_check_properties_1(char * fname)
{
  int m1, n1, l1;
  int m2, n2, l2;
  int m3, n3, l3;
  int m4, n4, l4;
  int m5, n5, l5;

  CheckRhs(5,5);
  CheckLhs(0,1) ;

  ////////////////////////////
  // Getting first argument //
  ////////////////////////////

  GetRhsVar(1, "d", &m1, &n1, &l1);

  CheckVector(1,m1,n1);   // Check that first argument is a vector
  CheckLength(1,m1*n1,4); // Check vector length

  /////////////////////////////
  // Getting second argument //
  /////////////////////////////

  GetRhsVar(2, "d", &amp;m2, &amp;n2, &amp;l2);

  CheckRow(2,m2,n2); // Checks that second argument is a row vector
                     // CheckColumn can also be used

  CheckDimProp(1,2, m1 * n1 != n2); // Check compatibility beetween arg 1 and a

  ////////////////////////////
  // Getting third argument //
  ////////////////////////////

  GetRhsVar(3, "d", &m3, &n3, &l3);

  CheckSameDims(1,3,m1,n1,m3,n3); // Checks that arg 1 and arg3 have same dimen
```

```
  ////////////////////////////
  // Getting fourth argument //
  ////////////////////////////

  GetRhsVar(4,"d",&m4,&n4,&l4);

  CheckScalar(4,m4,n4); // arg 4 must be scalar

  ////////////////////////////
  // Getting fourth argument //
  ////////////////////////////

  GetRhsVar(5,"d",&m5,&n5,&l5);

  CheckSquare(5,m5,n5);   // square matrix
  CheckDims(5,m5,m5,5,5); // check dimensions

  LhsVar(1)=0;

  return 0;
}

// We must be careful on the scilab name function (8 chars max).

int sci_check_properties_2(char * fname)
{
  int m1,n1,l1;

  CheckRhs(1,1);
  CheckLhs(0,1) ;

  switch(VarType(1))
    {
    case 1:
      GetRhsVar(1, "d", &m1, &n1, &l1);
      sciprint("1 is a scalar matrix\n");
      break;
    case 10:
      GetRhsVar(1, "c", &m1, &n1, &l1);
      sciprint("1 is a string\n");
      break;
    case 5:
      sciprint("1 is a sparse trying to overload\n");
      OverLoad(1);
    }

  LhsVar(1) = 0;

  return 0;
}
```

This file must be saved as "check_properties.c".

The main thing to highlight is that, to build a C gateway function, we need to include the header stack-c.h. In this header, we find the prototypes and macros of the main C gateway functions. We also need to include sciprint.h because we use the sciprint function.

To be able to build and link such a C function to Scilab, we need to write a Scilab script which will compile this C function and then create a loader script which will link the C function to a Scilab function.

## The builder script

```
// This is the builder.sce
// must be run from this directory

lines(0);

ilib_name  = 'lib_check_properties';

files = ['check_properties.c'];

libs  = [];

table =['check_properties_1', 'sci_check_properties_1'; ...
        'chprop2',            'sci_check_properties_2'];

// We must be careful when we choose a scilab function name in case of overload
// We Scilab name function must be 8 char max.

ldflags = "";
cflags  = "";
fflags  = "";

ilib_build(ilib_name,table,files,libs,'Makelib',ldflags,cflags,fflags);
```

This file must be saved as "builder.sce".

This script will tell Scilab which files must be compiled (here, it's check_properties.c), what will be the name of the shared library (here, it's lib_check_properties) and which C symbol will be linked to a Scilab function (here, we will link the sci_check_properties_1 C symbol to the Scilab function "check_properties_1").

For the other C function, we must be careful on the name of the Scilab function we will choose. Because this function will be overloading, the current overloading process of Scilab works only on Scilab primitives (Scilab function wrote in C) which have a name which is maximum 8 char wide.

For this function, we will link the sci_check_properties_2 C symbol to the Scilab function "chprop2").

To build this function, we just need to to:

```
exec builder.sce;
```

Now we are able to test our new C function. First, let's load this new function in scilab:

```
exec loader.sce;
```

The script loader.sce is normally automatically built by builder.sce.

## Testing our new function

We now write a simple example to test our new functions.

```
// checks arguments compatibility

check_properties_1([1;2;3;4],[3,4,5,6],[6;7;8;9],90,rand(5,5))

// first argument can have different types

chprop2([1,2,2]);
chprop2('foo');

// overload case

deff('[]=%sp_chprop2(sp)','disp(''sparse overloaded'')');
chprop2(sparse([1,2,3]));

// tests which give an error message with check_properties_1

try
  check_properties_1([1;2;3;4]',[3,4,5,6],[6;7;8;9],90,rand(5,5))
catch
  disp(lasterror());
end

try
  check_properties_1([1;2;3;4],[3,4,5,6]',[6;7;8;9],90,rand(5,5))
catch
  disp(lasterror());
end

try
  check_properties_1([1;2;3;4],[3,4,5,6],[6;7;8;9]',90,rand(5,5))
catch
  disp(lasterror());
end

try
  check_properties_1([1;2;3;4],[3,4,5,6],[6;7;8;9],[],rand(5,5))
catch
  disp(lasterror());
end

try
  check_properties_1([1;2;3;4],[3,4,5,6],[6;7;8;9],90,rand(4,4))
catch
  disp(lasterror());
end
```

The script must be saved as "check_properties.sce".

Let's run our scripts and see what is the result:

```
-->exec builder.sce;
   Génère un fichier gateway
   Génère un fichier loader
   Génère un Makefile : Makelib
   Exécute le makefile
   Compilation de check_properties.c
```

```
   Construction de la bibliothèque partagée (soyez patient)

-->;exec loader.sce;
Bibliothèque partagée chargée.
Link done.

-->exec check_properties.sce;
1 is a scalar matrix
1 is a string
1 is a sparse trying to overload

 sparse overloaded

 check_properties_1 : Les paramètres first et third a des dimensions incompatib

 check_properties_1: second paramètre devrait être un vecteur ligne

 check_properties_1 : Les paramètres first et third a des dimensions incompatib

 check_properties_1: fourth paramètre devrait être un scalaire

 check_properties_1 : Argument numéro 5 n'a pas les bonnes dimensions (4,4),  (
```

## See Also

GetRhsVar, CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, OverLoad, ilib_build

# Nome

How to create and access a list — How to create and access a list using the C gateway functions

## Description

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

The goal is to get a [mt]list, to get some elements stored in that this and to send a new [mt]list using a function written in C.

For this, we will wrote two C gateway function in which we will retrieve the [mt]list, and we will create a new [mt]list.

## The C function - access to a [mt]list

This example is available in SCI/modules/core/example/print_list.

Let's initialize a mlist in Scilab.

```
A = mlist(['mytype','var1','var2'],'a string',[1 2; 3 4]);
```

This mlist is of type 'mytype' (typeof(A)=='mytype') and it has 2 elements:

- A('var1') which is equal to 'a string'
- A('var2') which is equal to [1 2; 3 4]

We now create a C function called sci_print_list which will print the elements stored in the list.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_print_list(char * fname)
{
  int m_list_in, n_list_in, l_list_in;
  int m_type,    n_type;
  int m_var1,    n_var1,    l_var1;
  int m_var2,    n_var2,    l_var2;
  char ** LabelList = NULL;

  CheckRhs(1,1); // We accept only 1 parameter

  GetRhsVar(1,"m",&m_list_in,&n_list_in,&l_list_in); // Get a mlist

  // Get the type and the name of the variables (the first element of the mlist
  GetListRhsVar(1,1,"S",&m_type,&n_type,&LabelList);

  if (strcmp(LabelList[0],"mytype")!=0)
    {
      sciprint("error, you must ship a mlist or type mytype\n");
      return 0;
    }

  // Get the first variable (a string)
  GetListRhsVar(1,2,"c",&m_var1,&n_var1,&l_var1);
```

```
  sciprint("var1 = %s\n",cstk(l_var1));

  // Get the second variable (a double matrix)
  GetListRhsVar(1,3,"d",&m_var2,&n_var2,&l_var2);
  sciprint("var2 = [%f %f %f %f]\n",*stk(l_var2+0),
                                    *stk(l_var2+1),
                                    *stk(l_var2+2),
                                    *stk(l_var2+3));

  return 0;
}
```

To be able to build and link such a C function to scilab, we need to write a Scilab script which will compile this C function and then create a loader script which will link the C function to a Scilab function. The C file is available in the example directory. It is named print_list.c.

## The builder script

```
// This is the builder.sce
// must be run from this directory

lines(0);

ilib_name  = 'lib_print_list';

files = ['print_list.c'];

libs  = [];

table =['print_list', 'sci_print_list'];

ldflags = "";
cflags  = "";
fflags  = "";

ilib_build(ilib_name,table,files,libs,'Makelib',ldflags,cflags,fflags);
  </programlisting>

    <para>This file must be saved as "builder.sce".</para>

    <para>This script will tell Scilab which files must be compiled (here,
    it's print_list.c), what will be the name of the shared library (here,
    it's lib_print_list) and which C symbol will be linked to a Scilab
    function (here, we will link the sci_print_list C symbol to the Scilab
    function "print_list").</para>

    <para>To build this function, we just need to to:</para>

    <programlisting role = ""><![CDATA[
exec builder.sce;
```

Now we are able to test our new C function. First, let's load this new function in scilab:

```
exec loader.sce;
```

The script loader.sce is normally automatically built by builder.sce. Let's test our new function:

```
exec builder.sce;
exec loader.sce;

A = mlist(['mytype','var1','var2'],'a string',[1 2; 3 4]);

print_list(A);
```

## The C function - creation of a [mt]list

This example is available in SCI/modules/core/example/create_list.

We now write a simple example to test our new function to create a [mt]list.

```
A = create_list();

disp(A);
```

First, let's write the C function:

```
#include <stack-c.h>
#include <string.h>

int sci_create_list(char * fname)
{
  int m_list_out, n_list_out;
  int m_var1,     n_var1,     l_var1,  l_list_var1;
  int m_var2,     n_var2,     l_var2,  l_list_var2;
  int m_mlist,    n_mlist,    l_mlist;

  // The labels of our mlist
  static const char * ListLabels [] = {"mylist","var1","var2"};

  // First, we create the variables using a classical way
  // The size of the Scilab variables
  m_var1  = 1; n_var1  = strlen("a string")+1; // a null terminated string
  m_var2  = 2; n_var2  = 2; // A 2x2 double matrix
  m_mlist = 3; n_mlist = 1; // A mlist with 3 elements

  // Creation of the Scilab variables
  // A('var1')
  CreateVar(1, "c", &m_var1,  &n_var1,  &l_var1);
  // A('var2')
  CreateVar(2, "d", &m_var2,  &n_var2,  &l_var2);
  // A
  CreateVar(3, "m", &m_mlist, &n_mlist, &l_mlist);

  // We store values in the create variables
  // The matrix will be stored in A('var2')
  *stk(l_var2+0) = 1;
  *stk(l_var2+1) = 2;
  *stk(l_var2+2) = 3;
  *stk(l_var2+3) = 4;
```

```
  // The string will be stored in A('var1')
  strncpy(cstk(l_var1),"a string\0",n_var1);

  m_list_out = 3; n_list_out = 1;

  // now, affect the variable  to the mlist
  // The labels (it corresponds to A = mlist(['mylist','var1','var2'], ...
  CreateListVarFromPtr(3, 1, "S", &m_list_out, &n_list_out, ListLabels);
  // The value stored in A('var1') (it corresponds to A = ...,'a string', ...
  CreateListVarFrom(3, 2, "c", &m_var1, &n_var1, &l_list_var1, &l_var1);
  // The value stored in A('var2') (it corresponds to A = ...,[1 2,3 4]);
  CreateListVarFrom(3, 3, "d", &m_var2, &n_var2, &l_list_var2, &l_var2);

  // We return only the mlist which has been created at position 3
  LhsVar(1) = 3;

  return 0;
}
```

Some important comments related to the CreateVar(Pos,"m",&m, &n, &l) function. When called on a mlist, only the m parameter is taken in account, the n parameter is not used. So, be careful:

```
m_list = 3; n_list = 1;
CreateVar(1, "m", &m_list, &n_list, &l_list);
```

creates a mlist with 3 elements but:

```
m_list = 1; n_list = 3;
CreateVar(1, "m", &m_list, &n_list, &l_list);
```

creates a mlist with only 1 element !

Another important thing: when we create a list element using CreateListVarFrom, it is not recommended to access the created variable using, for example, stk(l_list_var2) because CreateListVarFrom performs type transformation on the list variables.

## The builder script

```
// This is the builder.sce
// must be run from this directory

lines(0);

ilib_name  = 'lib_create_list';

files = ['create_list.c'];

libs  = [];

table =['create_list', 'sci_create_list'];

ldflags = "";
```

```
cflags  = "";
fflags  = "";

ilib_build(ilib_name,table,files,libs,'Makelib',ldflags,cflags,fflags);
```

This file must be saved as "builder.sce".

This script will tell Scilab which files must be compiled (here, it's create_list.c), what will be the name of the shared library (here, it's lib_create_list) and which C symbol will be linked to a Scilab function (here, we will link the sci_create_list C symbol to the Scilab function "create_list").

To build this function, we just need to to:

```
exec builder.sce;
```

Now we are able to test our new C function. First, let's load this new function in scilab:

```
exec loader.sce;
```

The script loader.sce is normally automatically built by builder.sce. Let's test our new function:

```
exec builder.sce;
exec loader.sce;

A = create_list();

disp(typeof(A))
disp(getfield(1,A))
disp(A('var1'))
disp(A('var2'))
```

## See Also

GetRhsVar, GetListRhsVar, CreateVar, CreateListVarFrom, CreateListVarFromPtr, LhsVar, stk, ilib_build

# Nome

How to deal with optional parameters — how to deal with optional parameters send to an interface using the C gateway functions

## Description

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

The goal is to get a set of optional parameters via a C gateway and then to perform some checks in the C function (number of optional parameters, does an optional parameters exists, etc.).

This example is available in the directory core/examples/optional_parameters

## The C function

```
#include <stack-c.h>

int ex2c(double * a, int * ma, int * na,
         double * b, int * mb, int * nb)
{
  int i;

  for(i=0;i<(*ma)*(*na);i++) a[i] = 2*a[i];
  for(i=0;i<(*mb)*(*nb);i++) b[i] = 3*b[i];

  return(0);
}

int sci_optional_parameters(char * fname)
{
  int m1,n1,l1;

  // optional names must be stored in alphabetical order in opts
  static rhs_opts opts[]= {{-1,"v1","d",0,0,0},
                           {-1,"v2","d",0,0,0},
                           {-1,NULL,NULL,0,0}};

  int minrhs = 1, maxrhs = 1;
  int minlhs = 1, maxlhs = 3;
  int nopt, iopos, res;
  char buffer_name[csiz]; // csiz used for character coding

  nopt = NumOpt();

  CheckRhs(minrhs,maxrhs+nopt);
  CheckLhs(minlhs,maxlhs);

  // first non optional argument
  GetRhsVar( 1, "c", &m1, &n1, &l1);

  if (get_optionals(fname,opts)==0) return 0;

  // default values if optional arguments are not given:  v1=[99] and v2=[3]

  sciprint("number of optional parameters = %d\n", NumOpt());
  sciprint("first optional parameters = %d\n", FirstOpt());
  sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
```

```
  sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));

  if (IsOpt(1,buffer_name))
    sciprint("parameter 1 is optional: %s\n", buffer_name);
  if (IsOpt(2,buffer_name))
    sciprint("parameter 2 is optional: %s\n", buffer_name);
  if (IsOpt(3,buffer_name))
    sciprint("parameter 3 is optional: %s\n", buffer_name);

  iopos = Rhs;

  if (opts[0].position==-1)
    {
      iopos++;
      opts[0].position = iopos;
      opts[0].m = 1; opts[0].n = 1;
      opts[0].type = "d";
      CreateVar(opts[0].position, opts[0].type, &opts[0].m, &opts[0].n, &opts[0
      *stk(opts[0].l) = 99.0;
    }

  if (opts[1].position==-1)
    {
      iopos++ ;
      opts[1].position = iopos;
      opts[1].m = 1; opts[1].n = 1;
      opts[1].type = "d";
      CreateVar(opts[1].position, opts[1].type, &opts[1].m, &opts[1].n, &opts[1
      *stk(opts[1].l) = 3;
    }

  ex2c(stk(opts[0].l),&opts[0].m,&opts[0].n,
       stk(opts[1].l),&opts[1].m,&opts[1].n);

  // return the first argument (unchanged ) then v1 and v2

  LhsVar(1) = 1;
  LhsVar(2) = opts[0].position;
  LhsVar(3) = opts[1].position;

  return 0;
}
```

This file must be saved as "optional_parameters.c".

The main thing to highlight is that, to build a C interface function, we need to include the header stack-c.h. In this header, we find the prototypes and macros of the main C interface functions. We also need to include sciprint.h because we use the sciprint function.

To be able to build and link such a C function to scilab, we need to write a Scilab script which will compile this C function and then create a loader script which will link the C function to a Scilab function.

## The builder script

```
// This is the builder.sce
// must be run from this directory
```

```
lines(0);

ilib_name  = 'lib_optional_parameters';

files = ['optional_parameters.c'];

libs  = [];

table =['optional_parameters', 'sci_optional_parameters'];

ldflags = "";
cflags  = "";
fflags  = "";

ilib_build(ilib_name,table,files,libs,'Makelib',ldflags,cflags,fflags);
```

This file must be saved as "builder.sce".

This script will tell Scilab which files must be compiled (here, it's optional_parameters.c), what will be the name of the shared library (here, it's lib_optional_parameters) and which C symbol will be linked to a Scilab function (here, we will link the sci_optional_parameters C symbol to the Scilab function "optional_parameters").

To build this function, we just need to to:

```
exec builder.sce;
```

Now we are able to test our new C function. First, let's load this new function in scilab:

```
exec loader.sce;
```

The script loader.sce is normally automatically built by builder.sce.

## Testing our new function

We now write a simple example to test our new functions.

```
// Example with optional argument specified with the 'arg=value syntax'
// [a,b,c] = ex12c(x1, [v1 = arg1, v2 = arg2]), arg1 default value 99
//                                              arg2 default value 3
// only v1 and v2 are recognized as optional argument names
// the return value are a = x1, b = 2*v2, c = 3*v2

[a,b,c] = optional_parameters('test');
disp('a = ' + a + ' b = ' + string(b) + ' c = ' + string(c));

[a,b,c] = optional_parameters('test',v1=[10,20]);
disp('a = ' + a + ' b = ' + string(b) + ' c = ' + string(c));

[a,b,c] = optional_parameters('test',v1=[10,20],v2=8);
disp('a = ' + a + ' b = ' + string(b) + ' c = ' + string(c));
```

```
[a,b,c] = optional_parameters('test',v2=8,v1=[10]);
disp('a = ' + a + ' b = ' + string(b) + ' c = ' + string(c));
```

The script must be saved as "optional_parameters.sce".

Let's run our scripts and see what is the result:

```
-->;exec builder.sce;
   Génère un fichier gateway
   Génère un fichier loader
   Génère un Makefile : Makelib
   Exécute le makefile
   Compilation de optional_parameters.c
   Construction de la bibliothèque partagée (soyez patient)

-->;exec loader.sce;
Bibliothèque partagée chargée.
Link done.

-->;exec optional_parameters.sce;
number of optional parameters = 0
first optional parameters = 2
FindOpt(v1) = 0
FindOpt(v2) = 0

 a = test b = 198 c = 9
number of optional parameters = 1
first optional parameters = 2
FindOpt(v1) = 2
FindOpt(v2) = 0
parameter 2 is optional: v1

!a = test b = 20 c = 9  a = test b = 40 c = 9  !
number of optional parameters = 2
first optional parameters = 2
FindOpt(v1) = 2
FindOpt(v2) = 3
parameter 2 is optional: v1
parameter 3 is optional: v2

!a = test b = 20 c = 24  a = test b = 40 c = 24  !
number of optional parameters = 2
first optional parameters = 2
FindOpt(v1) = 3
FindOpt(v2) = 2
parameter 2 is optional: v2
parameter 3 is optional: v1

 a = test b = 20 c = 24
```

## See Also

GetRhsVar, CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, OverLoad, ilib_build

# Name

CheckColumn — C interface function which checks if a parameter send to the C function is a column vector or not

```
CheckColumn(StackPos,m_var,n_var)
```

# Parameters

StackPos
: the position in the Scilab memory of the argument for which we want to perform the check (input parameter)

m_var
: the number of lines of the parameter at position StackPos in the Scilab memory

n_var
: the number of columns of the parameter at position StackPos in the Scilab memory

# Description

C interface function which checks if a parameter send to the C function is a column vector or not. You must include stack-c.h to benefit from this function.

# Examples

In this example, the C interface function takes one input parameters and prints the integer corresponding to the type of the variable sent as parameter in the Scilab console. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
  int m1, n1, l1;

  CheckRhs(1,1);

  GetRhsVar(1, "d", &m1, &n1, &l1);

  CheckColumn(1,m1,n1); // Check that first argument is a column vector

  return 0;
}
```

# See Also

CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Name

CheckDimProp — C interface function which checks the comatibility between 2 arguments send to the C function

```
CheckColumn(StackPos_var1,StackPos_var2,expression)
```

# Parameters

StackPos_var1
: the position in the Scilab memory of the first parameter for which we want to perform the check (input parameter)

StackPos_var2
: the position in the Scilab memory of the second variable for which we want to perform the check (input parameter)

expression
: a boolean expression which represent the size check we want to perform

# Description

C interface function which checks the compatibility between 2 arguments send to the C function. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

# Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
  int m1, n1, l1;
  int m2, n2, l2;

  CheckRhs(2,2);

  GetRhsVar(1, "d", &m1, &n1, &l1);
  GetRhsVar(2, "d", &m2, &n2, &l2);

  // Check compatibility beetween arg 1 and arg 2. We want m1*n1 == n2
  CheckDimProp(1,2, m1 * n1 != n2);

  return 0;
}
```

# See Also

CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Name

CheckDims — C interface function which checks if a parameter send to the C function has the required dimensions

```
CheckDims(StackPos,m_var,n_var,m_required,n_required)
```

# Parameters

StackPos
: the position in the Scilab memory of the parameter for which we want to know the type (input parameter)

m_var
: the number of lines of the parameter at position StackPos in the Scilab memory (input parameter)

n_var
: the number of columns of the parameter at position StackPos in the Scilab memory (input parameter)

m_required
: the required number of lines (input parameter)

n_required
: the required number of columns (input parameter)

# Description

C interface function which checks if a parameter send to the C function has the required dimensions. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

# Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
  int m1, n1, l1;

  CheckRhs(1,1);

  GetRhsVar(1, "d", &m1, &n1, &l1);

  CheckDims(1,m1,n1,1,4); // Check that argument is a 1x4 matrix

  return 0;
}
```

# See Also

CheckColumn, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Name

CheckLength — C interface function which checks the length of a vector send as a parameter to the C function

```
CheckLength(StackPos,m_var,m_required)
```

# Parameters

StackPos
: the position in the Scilab memory of the parameter for which we want to check (input parameter)

m_var
: the number of lines of the parameter at position StackPos in the Scilab memory (input parameter)

m_required
: the required number of lines (input parameter)

# Description

C interface function which checks the length of a vector send as a parameter to the C function. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

# Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
  int m1, n1, l1;

  CheckRhs(1,1);

  GetRhsVar(1, "d", &m1, &n1, &l1);

  CheckLength(1,m1*n1,4); // Check that vector has 4 elements

  return 0;
}
```

# See Also

CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckSameDims, CheckSquare, How to check parameters

# Name

CheckLhs — C macro which checks the number of output arguments present in the calling Scilab function.

```
CheckLhs(nb_min_params,nb_max_params)
```

# Parameters

nb_min_params
: the minimum number of output arguments which must be present in the calling Scilab function

nb_max_params
: the maximum number of output arguments which must be present in the calling Scilab function

# Description

C macro which checks the number of output arguments present in the calling Scilab function. You must include stack-c.h to benefit from this function.

If the number of arguments is not between nb_min_params and nb_max_params, we quit the C interface (`return 0;`) and an error is returned in the Scilab console.

Since `CheckLhs` is doing a `return 0;` within the gateway function, it is important to call this macro before any memory allocation in order to avoid any memory leak.

# Examples

In this example, the C gateway function checks for a number of output arguments which must be between 2 and 3.

```
#include <stack-c.h>

int sci_mychecklhs(char * fname)
{
  CheckLhs(2,3);

  return 0;
}
```

Now, some functions testing this interface:

```
[A,B] = mychecklhs(); // OK, 2 output arguments
[A,B,C] = mychecklhs(); // OK, 3 output arguments
[A] = mychecklhs(); // ERROR, 1 output argument
[A,B,C,D] = mychecklhs(); // ERROR, 4 output arguments
```

# See Also

CheckRhs

# Name

CheckRhs — C macro which checks the number of input arguments present in the calling Scilab function.

```
CheckRhs(nb_min_params,nb_max_params)
```

# Parameters

nb_min_params
: The minimum number of input arguments which must be present in the calling Scilab function

nb_max_params
: the maximum number of input arguments which must be present in the calling Scilab function

# Description

C macro which checks the number of input arguments present in the calling Scilab function. You must include stack-c.h to benefit from this function.

If the number of input arguments is not between nb_min_params and nb_max_params, we quit the C interface (`return 0;`) and an error is returned in the Scilab console.

Since `CheckRhs` is doing a `return 0;` within the gateway function, it is important to call this macro before any memory allocation in order to avoid any memory leak.

# Examples

In this example, the C gateway function checks for a number of input arguments which must be between 2 and 3.

```
#include <stack-c.h>

int sci_mycheckrhs(char * fname)
{
  CheckRhs(2,3);

  return 0;
}
```

Now, some functions testing this interface:

```
mycheckrhs(A,B); // OK, 2 output arguments
mycheckrhs(A,B,C); // OK, 3 output arguments
mycheckrhs(A); // ERROR, 1 output argument
mycheckrhs(A,B,C,D); // ERROR, 4 output arguments
```

# See Also

CheckLhs

# Name

CheckRow — C interface function which checks if a parameter send to the C function is a row vector or not

```
CheckRow(StackPos,m_var,n_var)
```

# Parameters

StackPos
: the position in the Scilab memory of the parameter for which we want to know the type (input parameter)

m_var
: the number of lines of the parameter at position StackPos in the Scilab memory

n_var
: the number of columns of the parameter at position StackPos in the Scilab memory

# Description

C interface function which checks if a parameter send to the C function is a row vector or not. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

# Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
  int m1, n1, l1;

  CheckRhs(1,1);

  GetRhsVar(1, "d", &m1, &n1, &l1);

  CheckRow(1,m1,n1); // Check that first argument is a row vector

  return 0;
}
```

# See Also

CheckColumn, CheckDims, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Name

CheckSameDims — C interface function which checks if two parameters send to the C function have the same size

```
CheckSameDims(StackPos_var1,StackPos_var2,m_var1,n_var1,m_var2,n_var2)
```

# Parameters

StackPos_var1
: the position in the Scilab memory of the first parameter for which we want to perform the check (input parameter)

StackPos_var2
: the position in the Scilab memory of the second parameter for which we want to perform the check (input parameter)

m_var1
: the number of lines of the parameter at position StackPos_var1 in the Scilab memory

n_var1
: the number of columns of the parameter at position StackPos_var1 in the Scilab memory

m_var2
: the number of lines of the parameter at position StackPos_var2 in the Scilab memory

n_var2
: the number of columns of the parameter at position StackPos_var2 in the Scilab memory

# Description

C interface function which checks if two parameters send to the C function have the same size. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

# Examples

In this example, the C interface function takes one input parameters and prints the integer corresponding to the type of the variable sent as parameter in the Scilab console.

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
  int m1, n1, l1;
  int m2, n2, l2;

  CheckRhs(2,2);

  GetRhsVar(1, "d", &m1, &n1, &l1);
  GetRhsVar(2, "d", &m2, &n2, &l2);

  CheckSameDims(1,2,m1,n1,m2,n2); // Check that both vectors have the same size

  return 0;
}
```

# See Also

CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSquare, How to check parameters

# Name

CheckScalar — C interface function which checks if a parameter send to the C function is a scalar or not

```
CheckScalar(StackPos,m_var,n_var)
```

# Parameters

StackPos
: the position in the Scilab memory of the parameter for which we want to perform the check (input parameter)

m_var
: the number of lines of the parameter at position StackPos in the Scilab memory

n_var
: the number of columns of the parameter at position StackPos in the Scilab memory

# Description

C interface function which checks if a parameter send to the C function is a scalar or not. You must include stack-c.h to benefit from this function.

# Examples

In this example, the C interface function takes one input parameters and prints the integer corresponding to the type of the variable sent as parameter in the Scilab console. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
  int m1, n1, l1;

  CheckRhs(1,1);

  GetRhsVar(1, "d", &m1, &n1, &l1);

  CheckScalar(1,m1,n1); // Check that first argument is a scalar

  return 0;
}
```

# See Also

CheckColumn, CheckDims, CheckRow, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Name

CheckSquare — C interface function which checks if a parameter send to the C function is a square matrix or not

```
CheckSquare(StackPos,m_var,n_var)
```

# Parameters

StackPos
: the position in the Scilab memory of the parameter for which we want to perform the check (input parameter)

m_var
: the number of lines of the parameter at position StackPos in the Scilab memory

n_var
: the number of columns of the parameter at position StackPos in the Scilab memory

# Description

C interface function which checks if a parameter send to the C function is a square matrix or not. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

# Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
  int m1, n1, l1;

  CheckRhs(1,1);

  GetRhsVar(1, "d", &m1, &n1, &l1);

  CheckSquare(1,m1,n1); // Check that first argument is a square matrix

  return 0;
}
```

# See Also

CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, How to check parameters

# Name

CheckVector — C interface function which checks if a parameter send to the C function is a vector (column or row) or not

```
CheckVector(StackPos,m_var,n_var)
```

# Parameters

StackPos
: the position in the Scilab memory of the parameter for which we want to perform the check (input parameter)

m_var
: the number of lines of the parameter at position StackPos in the Scilab memory

n_var
: the number of columns of the parameter at position StackPos in the Scilab memory

# Description

C interface function which checks if a parameter send to the C function is a vector (column or row) or not. You must include stack-c.h to benefit from this function. If the test fails, we return from the C interface and an adequate error message is printed in the Scilab console.

# Examples

```
#include <stack-c.h>

int sci_check_properties(char * fname)
{
  int m1, n1, l1;

  CheckRhs(1,1);

  GetRhsVar(1, "d", &m1, &n1, &l1);

  CheckVector(1,m1,n1); // Check that first argument is a vector (column or row

  return 0;
}
```

# See Also

CheckColumn, CheckDims, CheckRow, CheckScalar, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Nome

CreateListVarFrom — a C interface function which allows to create a new Scilab parameter in a [mt]list

```
CreateListVarFrom(StackPos,Type, &m_rows, &n_cols, &l_stack_list_pos, &l_stack_
```

# Parameters

StackPos
: the rank of the parameter to be created (input parameter)

Type
: the Scilab type of the parameter to be created (input parameter). Can be (see Scilab C Type for more informations):

- STRING_DATATYPE "c"
- MATRIX_OF_STRING_DATATYPE "S"
- MATRIX_OF_DOUBLE_DATATYPE "d"
- MATRIX_OF_RATIONAL_DATATYPE "r"
- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE "I"
- MATRIX_OF_INTEGER_DATATYPE "i"
- MATRIX_OF_BOOLEAN_DATATYPE "b"
- MATRIX_OF_COMPLEX_DATATYPE "z"
- SPARSE_MATRIX_DATATYPE "s"
- TYPED_LIST_DATATYPE "t"
- MATRIX_ORIENTED_TYPED_LIST_DATATYPE "m"
- SCILAB_POINTER_DATATYPE "p"
- GRAPHICAL_HANDLE_DATATYPE "h"
- EXTERNAL_DATATYPE "f"
- MATRIX_OF_POLYNOMIAL_DATATYPE "x"

m_rows
: the number of lines of the matrix to be created (input parameter)

n_cols
: the number of columns of the matrix to be created (input parameter)

l_stack_list_pos
: the position in the Scilab memory of the created variable in the list(output parameter)

l_stack_pos
: the position in the Scilab memory of the created variable (input parameter)

# Description

A C interface function which allows to create a new Scilab variable in a [mt]list

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

# Examples

```
#include <stack-c.h>
#include <string.h>

int sci_create_list(char * fname)
{
  int m_list_out, n_list_out;
  int m_var1,     n_var1,     l_var1,  l_list_var1;
  int m_var2,     n_var2,     l_var2,  l_list_var2;
  int m_mlist,    n_mlist,    l_mlist;

  // The labels of our mlist
  static const char * ListLabels [] = {"mylist","var1","var2"};

  // First, we create the variables using a classical way
  // The size of the Scilab variables
  m_var1  = 1; n_var1  = strlen("a string")+1; // a null terminated string
  m_var2  = 2; n_var2  = 2; // A 2x2 double matrix
  m_mlist = 3; n_mlist = 1; // A mlist with 3 elements

  // Creation of the Scilab variables
  // A('var1')
  CreateVar(1, "c", &m_var1,  &n_var1,  &l_var1);
  // A('var2')
  CreateVar(2, "d", &m_var2,  &n_var2,  &l_var2);
  // A
  CreateVar(3, "m", &m_mlist, &n_mlist, &l_mlist);

  // We store values in the create variables
  // The matrix will be stored in A('var2')
  *stk(l_var2+0) = 1;
  *stk(l_var2+1) = 2;
  *stk(l_var2+2) = 3;
  *stk(l_var2+3) = 4;

  // The string will be stored in A('var1')
  strncpy(cstk(l_var1),"a string\0",n_var1);

  m_list_out = 3; n_list_out = 1;

  // now, affect the variable  to the mlist
  // The labels (it corresponds to A = mlist(['mylist','var1','var2'], ...
  CreateListVarFromPtr(3, 1, "S", &m_list_out, &n_list_out, ListLabels);
  // The value stored in A('var1') (it corresponds to A = ...,'a string', ...
  CreateListVarFrom(3, 2, "c", &m_var1, &n_var1, &l_list_var1, &l_var1);
  // The value stored in A('var2') (it corresponds to A = ...,[1 2,3 4]);
  CreateListVarFrom(3, 3, "d", &m_var2, &n_var2, &l_list_var2, &l_var2);

  // We return only the mlist which has been created at position 3
  LhsVar(1) = 3;

  return 0;
}
```

This example is available in SCI/modules/core/example/create_list.

# See Also

Scilab C Type, istk, LhsVar, CreateVar

# Nome

CreateListVarFromPtr — a C interface function which allows to create a new Scilab parameter from a pointer in a [mt]list

```
CreateListVarFrom(StackPos, Type, &m_rows, &n_cols, &l_stack_list_pos, void * P
```

# Parameters

StackPos
: the rank of the parameter to be created (input parameter)

Type
: the Scilab type of the parameter to be created (input parameter). Can be (see Scilab C Type for more informations):

- STRING_DATATYPE "c"
- MATRIX_OF_STRING_DATATYPE "S"
- MATRIX_OF_DOUBLE_DATATYPE "d"
- MATRIX_OF_RATIONAL_DATATYPE "r"
- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE "I"
- MATRIX_OF_INTEGER_DATATYPE "i"
- MATRIX_OF_BOOLEAN_DATATYPE "b"
- MATRIX_OF_COMPLEX_DATATYPE "z"
- SPARSE_MATRIX_DATATYPE "s"
- TYPED_LIST_DATATYPE "t"
- MATRIX_ORIENTED_TYPED_LIST_DATATYPE "m"
- SCILAB_POINTER_DATATYPE "p"
- GRAPHICAL_HANDLE_DATATYPE "h"
- EXTERNAL_DATATYPE "f"
- MATRIX_OF_POLYNOMIAL_DATATYPE "x"

m_rows
: the number of lines of the matrix to be created (input parameter)

n_cols
: the number of columns of the matrix to be created (input parameter)

l_stack_list_pos
: the position in the Scilab memory of the created parameter in the list (output parameter)

Pointer
: the pointer to the data area (input parameter)

# Description

A C interface function which allows to create a new Scilab parameter from a pointer in a [mt]list

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

# Examples

```
#include <stack-c.h>
#include <string.h>

int sci_create_list(char * fname)
{
  int m_list_out, n_list_out;
  int m_var1,     n_var1,     l_var1,  l_list_var1;
  int m_var2,     n_var2,     l_var2,  l_list_var2;
  int m_mlist,    n_mlist,    l_mlist;

  // The labels of our mlist
  static const char * ListLabels [] = {"mylist","var1","var2"};

  // First, we create the variables using a classical way
  // The size of the Scilab variables
  m_var1  = 1; n_var1  = strlen("a string")+1; // a null terminated string
  m_var2  = 2; n_var2  = 2; // A 2x2 double matrix
  m_mlist = 3; n_mlist = 1; // A mlist with 3 elements

  // Creation of the Scilab variables
  // A('var1')
  CreateVar(1, "c", &m_var1,  &n_var1,  &l_var1);
  // A('var2')
  CreateVar(2, "d", &m_var2,  &n_var2,  &l_var2);
  // A
  CreateVar(3, "m", &m_mlist, &n_mlist, &l_mlist);

  // We store values in the create variables
  // The matrix will be stored in A('var2')
  *stk(l_var2+0) = 1;
  *stk(l_var2+1) = 2;
  *stk(l_var2+2) = 3;
  *stk(l_var2+3) = 4;

  // The string will be stored in A('var1')
  strncpy(cstk(l_var1),"a string\0",n_var1);

  m_list_out = 3; n_list_out = 1;

  // now, affect the variable  to the mlist
  // The labels (it corresponds to A = mlist(['mylist','var1','var2'], ...
  CreateListVarFromPtr(3, 1, "S", &m_list_out, &n_list_out, ListLabels);
  // The value stored in A('var1') (it corresponds to A = ...,'a string', ...
  CreateListVarFrom(3, 2, "c", &m_var1, &n_var1, &l_list_var1, &l_var1);
  // The value stored in A('var2') (it corresponds to A = ...,[1 2,3 4]);
  CreateListVarFrom(3, 3, "d", &m_var2, &n_var2, &l_list_var2, &l_var2);

  // We return only the mlist which has been created at position 3
  LhsVar(1) = 3;

  return 0;
}
```

This example is available in SCI/modules/core/example/create_list.

# See Also

Scilab C Type, istk, LhsVar, CreateVar

# Nome

CreateVar — a C gateway function which allows to create a new Scilab parameter

```
CreateVar(StackPos, Type, &m_rows, &n_cols, &l_stack_pos);
```

# Parameters

StackPos
: The rank of the parameter to be created (input argument)

Type
: The Scilab C Type of the parameter to be created (input argument).

- STRING_DATATYPE "c"
- MATRIX_OF_STRING_DATATYPE "S"
- MATRIX_OF_DOUBLE_DATATYPE "d"
- MATRIX_OF_RATIONAL_DATATYPE "r"
- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE "I"
- MATRIX_OF_INTEGER_DATATYPE "i"
- MATRIX_OF_BOOLEAN_DATATYPE "b"
- MATRIX_OF_COMPLEX_DATATYPE "z"
- SPARSE_MATRIX_DATATYPE "s"
- TYPED_LIST_DATATYPE "t"
- MATRIX_ORIENTED_TYPED_LIST_DATATYPE "m"
- SCILAB_POINTER_DATATYPE "p"
- GRAPHICAL_HANDLE_DATATYPE "h"
- EXTERNAL_DATATYPE "f"
- MATRIX_OF_POLYNOMIAL_DATATYPE "x"

m_rows
: the number of lines of the matrix to be created (input argument)

n_cols
: the number of columns of the matrix to be created (input argument)

l_stack_pos
: the position in the Scilab memory of the created parameter (output argument)

# Description

A C gateway function which allows to create a new Scilab parameter

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

# Examples

```
#include <stack-c.h>

int sci_myones(char * fname)
{
  int m_row, n_col, l_pos;

  m_row = 1; n_col = 1; // We create a scalar
  CreateVar(1, MATRIX_OF_INTEGER_DATATYPE, &m_row, &n_col, &l_pos);

  *istk(l_pos) = 1;

  LhsVar(1) = 1;

  return 0;
}
```

# See Also

Scilab C Type, istk, LhsVar

# Name

FindOpt — C gateway function find the position of an optional argument given its name

```
Pos FindOpt(varname, opts)
```

# Parameters

varname
: the name of the optional parameter

opts
: a C list of optional parameters

```
typedef struct rhs_opts__ {
  int position ; // stack position : -1 if not present
  char *name; // the name of the variable
  char *type; // a Scilab type (like "d") representing the type of the varia
  int m,n; // the size of the variable
  unsigned long int l; // a pointer to the Scilab stack
} rhs_opts;
```

Pos
: the rank of the optional parameter if it has been found in the parameters sent to the C function, 0 otherwise.

# Description

A C gateway function which find the position of an optional argument given its name. You must include stack-c.h to benefit from this function.

# Examples

A more complete example is available in the directory SCI/modules/core/example/optional_parameters.

```
#include <stack-c.h>

int sci_optional_parameters(char * fname)
{
  int m1,n1,l1;

  // optional names must be stored in alphabetical order in opts
  static rhs_opts opts[]= {{-1,"v1","d",0,0,0},
                           {-1,"v2","d",0,0,0},
                           {-1,NULL,NULL,0,0}};

  int minrhs = 1, maxrhs = 1;
  int minlhs = 1, maxlhs = 3;
  int nopt, iopos, res;
  char buffer_name[csiz]; // csiz used for character coding

  nopt = NumOpt();
```

```
  CheckRhs(minrhs,maxrhs+nopt);
  CheckLhs(minlhs,maxlhs);

  // first non optional argument
  GetRhsVar( 1, "c", &m1, &n1, &l1);

  if (get_optionals(fname,opts)==0) return 0;

  sciprint("number of optional parameters = %d\n", NumOpt());
  sciprint("first optional parameters = %d\n", FirstOpt());
  sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
  sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));

  if (IsOpt(1,buffer_name))
    sciprint("parameter 1 is optional: %s\n", buffer_name);
  if (IsOpt(2,buffer_name))
    sciprint("parameter 2 is optional: %s\n", buffer_name);
  if (IsOpt(3,buffer_name))
    sciprint("parameter 3 is optional: %s\n", buffer_name);

  return 0;
}
```

# See Also

CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Name

FirstOpt — C gateway function which returns the position of the first optional parameter

```
Pos FirstOpt()
```

# Parameters

Pos
: the position of the first optional parameter, Rhs + 1 if no optional parameters have been given to the function

# Description

A C gateway function which returns the position of the first optional parameter. You must include stack-c.h to benefit from this function.

# Examples

A more complete example is available in the directory SCI/modules/core/example/optional_parameters.

```
#include <stack-c.h>

int sci_optional_parameters(char * fname)
{
  int m1,n1,l1;

  // optional names must be stored in alphabetical order in opts
  static rhs_opts opts[]= {{-1,"v1","d",0,0,0},
                           {-1,"v2","d",0,0,0},
                           {-1,NULL,NULL,0,0}};

  int minrhs = 1, maxrhs = 1;
  int minlhs = 1, maxlhs = 3;
  int nopt, iopos, res;
  char buffer_name[csiz]; // csiz used for character coding

  nopt = NumOpt();

  CheckRhs(minrhs,maxrhs+nopt);
  CheckLhs(minlhs,maxlhs);

  // first non optional argument
  GetRhsVar( 1, "c", &m1, &n1, &l1);

  if (get_optionals(fname,opts)==0) return 0;

  sciprint("number of optional parameters = %d\n", NumOpt());
  sciprint("first optional parameters = %d\n", FirstOpt());
  sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
  sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));

  if (IsOpt(1,buffer_name))
    sciprint("parameter 1 is optional: %s\n", buffer_name);
  if (IsOpt(2,buffer_name))
```

```
    sciprint("parameter 2 is optional: %s\n", buffer_name);
  if (IsOpt(3,buffer_name))
    sciprint("parameter 3 is optional: %s\n", buffer_name);

  return 0;
}
```

# See Also

CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Nome

GetListRhsVar — a C gateway function which allows to access a parameter stored in a [mt]list transmitted to a Scilab function

```
GetListRhsVar(StackPos,ListPos, Type, &m_rows, &n_cols, &l_stack_pos);
```

# Parameters

StackPos
: the rank of the [mt]list to be accessed (input parameter)

ListPos
: the rank in the list of the parameter to be accessed (input parameter)

Type
: the Scilab type of the parameter to be accessed (input parameter). Can be (see Scilab C Type for more informations):

- STRING_DATATYPE "c"
- MATRIX_OF_STRING_DATATYPE "S"
- MATRIX_OF_DOUBLE_DATATYPE "d"
- MATRIX_OF_RATIONAL_DATATYPE "r"
- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE "I"
- MATRIX_OF_INTEGER_DATATYPE "i"
- MATRIX_OF_BOOLEAN_DATATYPE "b"
- MATRIX_OF_COMPLEX_DATATYPE "z"
- SPARSE_MATRIX_DATATYPE "s"
- TYPED_LIST_DATATYPE "t"
- MATRIX_ORIENTED_TYPED_LIST_DATATYPE "m"
- SCILAB_POINTER_DATATYPE "p"
- GRAPHICAL_HANDLE_DATATYPE "h"
- EXTERNAL_DATATYPE "f"
- MATRIX_OF_POLYNOMIAL_DATATYPE "x"

m_rows
: the number of lines of the accessed parameter (output parameter)

n_cols
: the number of columns of the accessed parameter (output parameter)

l_stack_pos
: the position on the stack of the accessed parameter (output parameter)

# Description

A C gateway function which allows to access a parameter stored in a [mt]list transmitted to a Scilab function

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

# Examples

In this example, the function has one input parameter. It gets a mlist and the prints some informations related to the content of the mlist.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_print_list(char * fname)
{
  int m_list_in, n_list_in, l_list_in;
  int m_type,    n_type;
  int m_var1,    n_var1,    l_var1;
  int m_var2,    n_var2,    l_var2;
  char ** LabelList = NULL;

  CheckRhs(1,1); // We accept only 1 parameter

  GetRhsVar(1,"m",&m_list_in,&n_list_in,&l_list_in); // Get a mlist

  // Get the type and the name of the variables (the first element of the mlist
  GetListRhsVar(1,1,"S",&m_type,&n_type,&LabelList);

  if (strcmp(LabelList[0],"mytype")!=0)
    {
      sciprint("error, you must ship a mlist or type mytype\n");
      return 0;
    }

  // Get the first variable (a string)
  GetListRhsVar(1,2,"c",&m_var1,&n_var1,&l_var1);
  sciprint("var1 = %s\n",cstk(l_var1));

  // Get the second variable (a double matrix)
  GetListRhsVar(1,3,"d",&m_var2,&n_var2,&l_var2);
  sciprint("var2 = [%f %f %f %f]\n",*stk(l_var2+0),
                                    *stk(l_var2+1),
                                    *stk(l_var2+2),
                                    *stk(l_var2+3));

  return 0;
}
```

This example is available in SCI/modules/core/example/print_list.

# See Also

Scilab C Type, istk, LhsVar

# Nome

GetRhsVar — a C gateway function which allows to access an argument transmitted to a Scilab function

```
GetRhsVar(StackPos, Type, &m_rows, &n_cols, &l_stack_pos);
```

# Parameters

StackPos
: The rank of the variable to be accessed (input argument)

Type
: The Scilab C Type of the parameter to be accessed (input argument).

- STRING_DATATYPE "c"
- MATRIX_OF_STRING_DATATYPE "S"
- MATRIX_OF_DOUBLE_DATATYPE "d"
- MATRIX_OF_RATIONAL_DATATYPE "r"
- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE "I"
- MATRIX_OF_INTEGER_DATATYPE "i"
- MATRIX_OF_BOOLEAN_DATATYPE "b"
- MATRIX_OF_COMPLEX_DATATYPE "z"
- SPARSE_MATRIX_DATATYPE "s"
- TYPED_LIST_DATATYPE "t"
- MATRIX_ORIENTED_TYPED_LIST_DATATYPE "m"
- SCILAB_POINTER_DATATYPE "p"
- GRAPHICAL_HANDLE_DATATYPE "h"
- EXTERNAL_DATATYPE "f"
- MATRIX_OF_POLYNOMIAL_DATATYPE "x"

m_rows
: the number of lines of the accessed parameter (output argument)

n_cols
: the number of columns of the accessed parameter (output argument)

l_stack_pos
: the position in the Scilab memory of the accessed parameter (output argument)

# Description

A C gateway function which allows to access a argument transmitted to a Scilab function

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

# Examples

In this example, the function has two input arguments:

- the number of columns (first argument)
- the number of lines (second argument)

The goal of this function is to created a matrix of integers equal to 1.

```
#include <stack-c.h>
#include <string.h>

int sci_myones(char * fname)
{
  int m_param_1, n_param_1, l_param_1;
  int m_param_2, n_param_2, l_param_2;
  int m_out_row, n_out_col, l_out_pos;
  int i;
  int * pOutPos = NULL;

  GetRhsVar(1, MATRIX_OF_INTEGER_DATATYPE, &m_param_1, &n_param_1, &l_param_1);
  GetRhsVar(2, MATRIX_OF_INTEGER_DATATYPE, &m_param_2, &n_param_2, &l_param_2);

  // We create a matrix of ints equal to 1
  m_out_row = *istk(l_param_1); // The first dimension of the matrix to be crea
                                // is stored in the first input parameter
  n_out_col = *istk(l_param_2); // The second dimension of the matrix to be cre
                                // is stored in the second input parameter

  CreateVar(3, MATRIX_OF_INTEGER_DATATYPE, &m_out_row, &n_out_col, &l_out_pos);

  pOutPos = istk(l_out_pos); // Get a pointer to the area allocated by CreateVa
  for(i=0;i<m_out_row*n_out_row;i++) pOutPos[i] = 1;

  // A concise way to write the preceding line of code:
  // for(i=0;i<m_out_row*n_out_row;i++) *istk(l_out_pos+i) = 1;

  LhsVar(1) = 3; // We return the 3rd Scilab variable of our gateway

  return 0;
}
```

# See Also

Scilab C Type, istk, LhsVar

# Name

GetType — C gateway function which returns the type of a parameter in the Scilab memory

```
SciType GetType(StackPos)
```

# Parameters

StackPos
: the position on the Scilab memory of the parameter for which we want to know the type (input argument)

SciType
: the type (defined in the sci_types enum which you can find in stack-c.h) of the parameter at position StackPos in the Scilab memory

# Description

GetType is a C gateway function which returns the type of a parameter in the Scilab memory. You must include stack-c.h to benefit from this function.

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

# Examples

In this example, the C gateway function takes one input argument and prints the integer corresponding to the type of the variable sent as argument in the Scilab console.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_mygettype(char * fname)
{
  sciprint("The type of the first argument is %d\n", GetType(1));

  return 0;
}
```

# See Also

sciprint, sci_types

# Name

IsOpt — C gateway function which checks if a parameter is optional and returns the name of the parameter

```
Res IsOpt(Pos,buffer_name)
```

# Parameters

Pos
: the position in the Scilab memory of the parameter to be checked (input parameter)

buffer_name
: an array of char (of size csiz which correspond to the maximum length of a Scilab variable) in which the name of the parameter will be copied if the position Pos is optional (output parameter)

Res
: 1 if Pos has an optional parameter, 0 otherwise

# Description

C gateway function which checks if a parameter is optional and returns the name of the parameter. You must include stack-c.h to benefit from this function.

# Examples

A more complete example is available in the directory SCI/modules/core/example/optional_parameters.

```
#include <stack-c.h>

int sci_optional_parameters(char * fname)
{
  int m1,n1,l1;

  // optional names must be stored in alphabetical order in opts
  static rhs_opts opts[]= {{-1,"v1","d",0,0,0},
                           {-1,"v2","d",0,0,0},
                           {-1,NULL,NULL,0,0}};

  int minrhs = 1, maxrhs = 1;
  int minlhs = 1, maxlhs = 3;
  int nopt, iopos, res;
  char buffer_name[csiz]; // csiz used for character coding

  nopt = NumOpt();

  CheckRhs(minrhs,maxrhs+nopt);
  CheckLhs(minlhs,maxlhs);

  // first non optional argument
  GetRhsVar( 1, "c", &m1, &n1, &l1);

  if (get_optionals(fname,opts)==0) return 0;

  sciprint("number of optional parameters = %d\n", NumOpt());
```

```
  sciprint("first optional parameters = %d\n", FirstOpt());
  sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
  sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));

  if (IsOpt(1,buffer_name))
    sciprint("parameter 1 is optional: %s\n", buffer_name);
  if (IsOpt(2,buffer_name))
    sciprint("parameter 2 is optional: %s\n", buffer_name);
  if (IsOpt(3,buffer_name))
    sciprint("parameter 3 is optional: %s\n", buffer_name);

  return 0;
}
```

# See Also

CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Name

Lhs — A C gateway function which provides the number of output arguments present in the calling Scilab function

```
nb_params Lhs
```

# Parameters

nb_params
: the number of output arguments present in the calling Scilab function

# Description

Lhs provides a C gateway function which provides the number of output arguments present in the calling Scilab function. You must include stack-c.h to benefit from this function.

Note: Lhs means Left Hand Side.

# Examples

In this example, the C gateway function can take several output arguments and prints in the Scilab console the integer corresponding to the number of output arguments detected in the calling Scilab function.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_mylhs(char * fname)
{
  sciprint("The number of output arguments is %d\n", Lhs);

  return 0;
}
```

# See Also

sciprint, Rhs

# Nome

LhsVar — a C gateway function which specifies which parameters created inside the C gateway will be returned as an output argument into Scilab.

```
LhsVar(RankPos) = RankVar;
```

# Parameters

RankPos
: as integer providing the rank of the output argument

RankVar
: the rank of the parameter created inside the C gateway to be returned as an Scilab output argument

# Description

A C gateway function which specifies which variables created inside the C interface will be returned as an output argumen into Scilab.

# Examples

This example takes a matrix of doubles as input and returns:

- the number of lines (first output argument)
- the number of rows (second output argument)

We create an intermediate Scilab parameter which will handle an integer but will neither be used nor returned as an output argument.

TODO: insert an example in the Scilab language

```
#include <stack-c.h>

int sci_mysizedouble(char * fname)
{
  int m_in_row,          n_in_col,          l_in_pos;
  int m_out_lines_row,   n_out_lines_col,   l_out_lines_pos;
  int m_out_columns_row, n_out_columns_col, l_out_columns_pos;
  int m_nop,             n_nop,             l_nop;

  GetRhsVar(1, MATRIX_OF_DOUBLE_DATATYPE, &m_in_row, &n_in_col, &l_in_pos);

  m_out_lines_row   = 1; n_out_lines_col   = 1; // We create a scalar
  m_out_columns_row = 1; n_out_columns_col = 1; // We create a scalar
  m_nop             = 1; n_nop             = 1; // We create a scalar

  CreateVar(2, MATRIX_OF_INTEGER_DATATYPE, &m_out_lines_row,   &n_out_lines_col
  CreateVar(3, MATRIX_OF_INTEGER_DATATYPE, &m_nop,             &n_nop,
  CreateVar(4, MATRIX_OF_INTEGER_DATATYPE, &m_out_columns_row, &n_out_columns_c

  *istk(l_out_lines_pos)   = m_in_row; // the out_lines_pos parameter handles t
  *istk(l_nop)             = 1; // store a mere value, but will neither be used
  *istk(l_out_columns_pos) = n_in_col; // the out_columns_pos parameter handles

  LhsVar(1) = 2; // We set the parameter 2 as an output argument
```

```
  LhsVar(2) = 4; // We set the parameter 4 as an output argument

  return 0;
}
```

# See Also

Scilab C Type, istk, CreateVar, GetRhsVar

# Name

NumOpt — C gateway function which returns the number of optional parameters sent to a C function

```
Res = NumOpt()
```

# Parameters

Res
: the number of optional parameters detected

# Description

A C gateway function which returns the number of optional parameters sent to a C function. You must include stack-c.h to benefit from this function.

# Examples

A more complete example is available in the directory SCI/modules/core/example/optional_parameters.

```
#include <stack-c.h>

int sci_optional_parameters(char * fname)
{
  int m1,n1,l1;

  // optional names must be stored in alphabetical order in opts
  static rhs_opts opts[]= {{-1,"v1","d",0,0,0},
                           {-1,"v2","d",0,0,0},
                           {-1,NULL,NULL,0,0}};

  int minrhs = 1, maxrhs = 1;
  int minlhs = 1, maxlhs = 3;
  int nopt, iopos, res;
  char buffer_name[csiz]; // csiz used for character coding

  nopt = NumOpt();

  CheckRhs(minrhs,maxrhs+nopt);
  CheckLhs(minlhs,maxlhs);

  // first non optional argument
  GetRhsVar( 1, "c", &m1, &n1, &l1);

  if (get_optionals(fname,opts)==0) return 0;

  sciprint("number of optional parameters = %d\n", NumOpt());
  sciprint("first optional parameters = %d\n", FirstOpt());
  sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
  sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));

  if (IsOpt(1,buffer_name))
    sciprint("parameter 1 is optional: %s\n", buffer_name);
  if (IsOpt(2,buffer_name))
    sciprint("parameter 2 is optional: %s\n", buffer_name);
```

```
  if (IsOpt(3,buffer_name))
    sciprint("parameter 3 is optional: %s\n", buffer_name);

  return 0;
}
```

# See Also

CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Name

OverLoad — C gateway function which tells Scilab to look for another overloaded function

```
OverLoad(StackPos)
```

# Parameters

StackPos
: the position in the Scilab memory of the parameter for which we want to take into account for the overloading process (input argument)

# Description

A C gateway function which tells Scilab to look for another overloaded function. Scilab then appends to the name of the function a prefix (like %sp_ if the parameter taken into account is sparse) and look for the overloaded function. You must include stack-c.h to benefit from this function.

Be careful with the Scilab name of the function. Indeed, the current overloading process of Scilab works only on Scilab primitives (Scilab function wrote in C) which must have a Scilab name which is maximum 8 char wide.

# Examples

In this example, the C interface function takes one input parameters and prints the integer corresponding to the type of the variable sent as parameter in the Scilab console.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_check_properties_2(char * fname)
{
  int m1,n1,l1;

  CheckRhs(1,1);
  CheckLhs(0,1) ;

  switch(VarType(1))
    {
    case sci_matrix:
      GetRhsVar(1, "d", &m1, &n1, &l1);
      sciprint("1 is a scalar matrix\n");
      break;
    case sci_strings:
      GetRhsVar(1, "c", &m1, &n1, &l1);
      sciprint("1 is a string\n");
      break;
    case sci_sparse:
      sciprint("1 is a sparse trying to overload\n");
      OverLoad(1);
    }

  LhsVar(1) = 0;

  return 0;
}
```

```
 </programlisting>

    <para>The builder.sce script look like this:</para>

    <programlisting role = "example"><![CDATA[
// This is the builder.sce
// must be run from this directory

lines(0);

ilib_name  = 'lib_check_properties';

files = ['check_properties.c'];

libs  = [];

table =['chprop2', 'sci_check_properties_2'];

// We must be careful when we choose a scilab function name in case of overload.
// We Scilab name function must be 8 char max.

ldflags = "";
cflags  = "";
fflags  = "";

ilib_build(ilib_name,table,files,libs,'Makelib',ldflags,cflags,fflags);
```

And now, an example of use of this new function:

```
chprop2([1,2,2]);
chprop2('foo');

// overload case

deff('[]=%sp_chprop2(sp)','disp(''sparse overloaded'')');
chprop2(sparse([1,2,3]));
```

# See Also

CheckColumn, CheckDims, CheckRow, CheckScalar, CheckVector, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Name

Rhs — A C gateway function which provides the number of input arguments present in the calling Scilab function

```
nb_params Rhs
```

# Parameters

nb_params
: the number of input arguments present in the calling Scilab function

# Description

A C gateway function which provides the number of input arguments present in the calling Scilab function. You must include stack-c.h to benefit from this function.

Note: Rhs means Right Hand Side.

# Examples

In this example, the C interface function can take several input arguments and prints in the Scilab console the integer corresponding to the number of input arguments detected in the calling Scilab function.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_myrhs(char * fname)
{
  sciprint("The number of input parameters is %d\n", Rhs);

  return 0;
}
```

# See Also

sciprint, Lhs

# Name

Scierror — C gateway function which displays an error message to the user (same profil as the printf function) and returns an integer value specifying an error level

```
void Scierror(error_level,format,value_1,..,value_n)
```

# Parameters

error_level
: an integer value specifying an error level

format
: a char* string. Specifies a character string combining literal characters with conversion specifications.

value_i
: Specifies the data to be converted according to the format parameter.

returns
: If the operation is successfull, this function returns the number of characters printed (not including the trailing '\0' used to end output to strings).

  If an error occurred, a negative value is returned.

# Description

Scierror is a C gateway function which displays an error message to the user (same profil as the printf function) and returns an integer value specifying an error level. You must include Scierror.h to benefit from this function. This header is provided in the output_stream module (this directory should be included by default).

# Examples

In this example, the C gateway function prints an error message and returns the error level 133.

```
#include <stack-c.h>
#include <Scierror.h>

int sci_myscierror(char * fname)
{
  Scierror(133,"An error has occured: %d\n", 1);

  return 0;
}
```

# See Also

printf_conversion, printf, sciprint

# Nome

Scilab C Types — the C types available in a C gateway

# Description

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

The string/char * datatype in the Scilab memory:

- STRING_DATATYPE
- "c"

The string/char ** datatype in the Scilab memory:

- MATRIX_OF_STRING_DATATYPE
- "S"

A matrix of double * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_DOUBLE_DATATYPE
- "d"

A matrix of rational * if the size of the matrix is 1,1, it is a single value

- MATRIX_OF_RATIONAL_DATATYPE
- "r"

A matrix of integer * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_VARIABLE_SIZE_INTEGER_DATATYPE
- "I"

A matrix of 'little' integer * 'little' because in reality, this int is a complex with the imaginary * part set to 0 * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_INTEGER_DATATYPE
- "i"

A matrix of boolean * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_BOOLEAN_DATATYPE
- "b"

A matrix of complex * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_COMPLEX_DATATYPE
- "z"

A sparse matrix * if the size of the matrix is 1,1, it is a single value

- SPARSE_MATRIX_DATATYPE
- "s"

A list:

- LIST_DATATYPE
- "l"

A typed list:

- TYPED_LIST_DATATYPE
- "t"

A Matrix oriented typed list * mlist object are very similar to tlist objects. But * if M is an mlist, for any index i which is not a field name, * M(i) is not the i th field of the list but is interpreted as * the i th entry of M seen as a vector. * This is the only difference between mlist and tlist:

- MATRIX_ORIENTED_TYPED_LIST_DATATYPE
- "m"

The scilab pointer datatype in the Scilab memory:

- SCILAB_POINTER_DATATYPE
- "p"

The scilab graphic handle datatype in the Scilab memory:

- GRAPHICAL_HANDLE_DATATYPE
- "h"

An "external" is a function or routine which is used as an argument * of some high-level primitives (such as ode, optim, schur...):

- EXTERNAL_DATATYPE
- "f"

A matrix of polynomial coeff * if the size of the matrix is 1,1, it is a single value:

- MATRIX_OF_POLYNOMIAL_DATATYPE
- "x"

# See Also

mlist, list, tlist, CreateVar, GetRhsVar

# Name

get_optionals — C gateway function which initialize the list of optional parameters

```
res get_optionals(fname, opts)
```

# Parameters

fname
: the name passed to the C gateway. The name of the calling function (of type char *) (input parameter)

opts
: a C list of optional parameters

```
typedef struct rhs_opts__ {
  int position ; // stack position : -1 if not present
  char *name; // the name of the variable
  char *type; // a Scilab type (like "d") representing the type of the varia
  int m,n; // the size of the variable
  unsigned long int l; // a pointer to the Scilab stack
} rhs_opts;
```

res
: if no optional parameters has been sent, the res = 0. Otherwise, res = 1.

# Description

A C gateway function which initialize the list of optional parameters. You must include stack-c.h to benefit from this function.

# Examples

A more complete example is available in the directory SCI/modules/core/example/optional_parameters.

```
#include <stack-c.h>

int sci_optional_parameters(char * fname)
{
  int m1,n1,l1;

  // optional names must be stored in alphabetical order in opts
  static rhs_opts opts[]= {{-1,"v1","d",0,0,0},
                           {-1,"v2","d",0,0,0},
                           {-1,NULL,NULL,0,0}};

  int minrhs = 1, maxrhs = 1;
  int minlhs = 1, maxlhs = 3;
  int nopt, iopos, res;
  char buffer_name[csiz]; // csiz used for character coding

  nopt = NumOpt();
```

```
  CheckRhs(minrhs,maxrhs+nopt);
  CheckLhs(minlhs,maxlhs);

  // first non optional argument
  GetRhsVar( 1, "c", &m1, &n1, &l1);

  if (get_optionals(fname,opts)==0) return 0;

  sciprint("number of optional parameters = %d\n", NumOpt());
  sciprint("first optional parameters = %d\n", FirstOpt());
  sciprint("FindOpt(v1) = %d\n", FindOpt("v1", opts));
  sciprint("FindOpt(v2) = %d\n", FindOpt("v2", opts));

  if (IsOpt(1,buffer_name))
    sciprint("parameter 1 is optional: %s\n", buffer_name);
  if (IsOpt(2,buffer_name))
    sciprint("parameter 2 is optional: %s\n", buffer_name);
  if (IsOpt(3,buffer_name))
    sciprint("parameter 3 is optional: %s\n", buffer_name);

  return 0;
}
```

# See Also

CheckDims, CheckRow, CheckScalar, CheckVector, CheckOverLoad, CheckDimProp, CheckLength, CheckSameDims, CheckSquare, How to check parameters

# Nome

istk — Return a pointer to an integer to access data stored at a given position in the Scilab memory

```
istk(l_stack_pos)
```

# Parameters

l_stack_pos
: the position in the Scilab memory of the parameter to be accessed (input argument)

# Description

This C gateway function returns a pointer to an integer to access data stored at a given position in the Scilab memory.

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

# Examples

```
#include <stack-c.h>

int sci_myones(char * fname)
{
  int m_row, n_col, l_pos;

  m_row = 1; n_col = 1; // We create a scalar
  CreateVar(1, MATRIX_OF_INTEGER_DATATYPE, &m_row, &n_col, &l_pos);

  *istk(l_pos) = 1; // We store the value 1 in the memory allocated for output
                    // parameter 1

  LhsVar(1) = 1;

  return 0;
}
```

# See Also

Scilab C Type, CreateVar, LhsVar

# Name

sci_types — a C enumeration which defines the types available for a variable

# Description

A C enumeration which defines the types available for a variable. You must include stack-c.h to benefit from this type definition.

The list of available types is the following:

- 1 - *sci_matrix*: a matrix of doubles
- 2 - *sci_poly*: a polynomials matrix
- 4 - *sci_boolean*: a boolean matrix
- 5 - *sci_sparse*: a sparse matrix
- 6 - *sci_boolean_sparse*: a sparse boolean matrix
- 7 - *sci_matlab_sparse*: a sparse matlab matrix
- 8 - *sci_ints*: a matrix of integers
- 9 - *sci_handles*: a graphical handle
- 10 - *sci_strings*: a matrix of strings
- 11 - *sci_u_function*: an uncompiled Scilab function
- 13 - *sci_c_function*: a compiled Scilab function
- 14 - *sci_lib*: a library of Scilab functions
- 15 - *sci_list*: a Scilab list
- 16 - *sci_tlist*: a Scilab tlist
- 17 - *sci_mlist*: a Scilab mlist
- 128 - *sci_pointer (was: sci_lufact_pointer before Scilab 5.2)*: a pointer

Integers or enumeration types can be used to check the type of the variables. Using the enumeration type is recommended because of the explicit meaning of the value of the enumeration type.

If this function is used, it is probable that GetType will also be used.

# Examples

In this example, the C gateway function takes one argument. Through a switch case structure, we display the type of the variable sent as a parameter.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_mysci_typesrhs(char * fname)
{
  switch(GetType(1))
  {
    case sci_matrix:
      sciprint("A matrix of doubles\n");
```

```
      break;
    case sci_poly:
      sciprint("A matrix of polynomials\n");
      break;
    case sci_boolean:
      sciprint("A matrix of booleans\n");
      break;
    case sci_sparse:
      sciprint("A sparse matrix of doubles\n");
      break;
    case sci_boolean_sparse:
      sciprint("A sparse matrix of booleans\n");
      break;
    case sci_matlab_sparse:
      sciprint("A sparse matlab matrix\n");
      break;
    case sci_ints:
      sciprint("A matrix of integers\n");
      break;
    case sci_handles:
      sciprint("A graphic handle\n");
      break;
    case sci_strings:
      sciprint("A matrix of strings\n");
      break;
    case sci_u_function:
      sciprint("An uncompiled Scilab function\n");
      break;
    case sci_c_function:
      sciprint("A compiled Scilab function\n");
      break;
    case sci_lib:
      sciprint("A library of Scilab functions\n");
      break;
    case sci_list:
      sciprint("A Scilab list\n");
      break;
    case sci_tlist:
      sciprint("A Scilab tlist\n");
      break;
    case sci_mlist:
      sciprint("A Scilab mlist\n");
      break;
    case sci_pointer:
    case sci_lufact_pointer: /* Renamed in version 5.2 */
      sciprint("A pointer\n");
      break;
    default:
      sciprint("Unknown type !\n"); // Should never happen
  }

  return 0;
}
```

# See Also

sciprint, GetType

# Name

sciprint — A C gateway function which displays standard messages to the user (same profil as the C printf function)

```
void sciprint(format,value_1,..,value_n)
```

# Parameters

format
: a char* string. Specifies a character string combining literal characters with conversion specifications.

value_i
: Specifies the data to be converted according to the format parameter (%s, %d, ...).

# Description

This C gateway function provides the capabilities to display messages to the Scilab user. Basically; it emulates the C language printf function. You must include sciprint.h to benefit from this function. This header is provided in the output_stream module (this directory should be included by default).

Note that if you want to trigger an error, the function Scierror is more appropriate.

# Examples

In this example, the C gateway function prints several messages illustrating the use of the sciprint function in the Scilab console.

```
#include <stack-c.h>
#include <sciprint.h>

int sci_mysciprint(char * fname)
{
  sciprint("printing an integer: %d\n", 1);
  sciprint("printing a double:   %f\n", 2.1);
  sciprint("printing a string:   %s\n", "test");

  return 0;
}
```

# See Also

printf_conversion, printf, Scierror

# Nome

stk — Return a pointer to a double to access data stored at a given position in the Scilab memory

```
stk(l_stack_pos)
```

# Parameters

l_stack_pos
: the position in the Scilab memory of the variable to be accessed (input argument)

# Description

This C gateway function returns a pointer to a double to access data stored at a given position in the Scilab memory .

WARNING: this API is deprecated. It will be removed in Scilab 6. Please use the new API instead.

# Examples

```
#include <stack-c.h>

int sci_myones(char * fname)
{
  int m_row, n_col, l_pos;

  m_row = 1; n_col = 1; // We create a scalar
  CreateVar(1, MATRIX_OF_DOUBLE_DATATYPE, &m_row, &n_col, &l_pos);

  *stk(l_pos) = 1.0; // We store the value 1.0 in the area allocated by CreateVa

  LhsVar(1) = 1; // We set the parameter 1 created by CreateVar as an output pa
                   // of the gateway function

  return 0;
}
```

# See Also

Scilab C Type, CreateVar, LhsVar

# Capítulo 12. list_management

# Name

Boolean reading (Scilab gateway) — How to read matrix of boolean in a list.

Input argument profile:

```
SciErr getMatrixOfBooleanInList(void* _pvCtx, int* _piParent, int _iItemPos, in
```

Named variable profile:

```
SciErr readMatrixOfBooleanInNamedList(void* _pvCtx, char* _pstName, int* _piPar
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piParent
: Address of the parent of the new item.

_pstName
: Name of the variable for "named" functions.

_iItemPos
: Position of the new item in the list.

_piRows
: Return number of rows of the variable.

_piCols
: Return number of columns of the variable.

_piBool
: Return address of data array (size: _iRows * _iCols).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to read matrix of boolean in a list.

# Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
 int i = 0;
 for(i = 0 ; i < iTab ; i++)
 {
  sciprint("\t");
 }
}


int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
```

```
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int iItem       = 0;
 int iRet        = 0;
 int *piAddr     = NULL;

 CheckRhs(1,1);

 sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 get_info(1, NULL, piAddr, 0);

 LhsVar(1) = 0;
 return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRet    = 0;
 int iType   = 0;

 sciErr = getVarType(pvApiCtx, _piAddr, &iType);
 switch(iType)
 {
 case sci_matrix :
  iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_poly :
  iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean :
  iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_sparse :
  iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean_sparse :
  iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_ints :
  iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_strings :
```

```
  iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_list :
  insert_indent();
  sciprint("List ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_tlist :
  insert_indent();
  sciprint("TList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_mlist :
  insert_indent();
  sciprint("MList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_pointer :
  iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 default :
  insert_indent();
  sciprint("Unknow type\n");
  return 1;
 }
 return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRet        = 0;
 int iItem       = 0;
 int* piChild    = NULL;

 sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciprint("(%d)\n", iItem);
 for(i = 0 ; i < iItem ; i++)
 {
  sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  iTab++;
  iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
  iTab--;
 }
```

```
 return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblRea
  }
  else
  {
   sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRow
  }
  else
  {
   sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Double (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iLen            = 0;
 int iRows           = 0;
 int iCols           = 0;

 char pstVar[16];
 int* piCoeff        = NULL;
 double** pdblReal   = NULL;
```

```
double** pdblImg    = NULL;

sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

if(_iItemPos == 0)
{//not in list
 sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 piCoeff     = (int*)malloc(sizeof(int) * iRows * iCols);
 sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
 pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
  pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
 }

 if(isVarComplex(pvApiCtx, _piAddr))
 {
  sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, 
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
 else
 {
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblRea
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
}
else
{
 sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
 if(sciErr.iErr)
 {
```

```
   printError(&sciErr, 0);
   return 0;
  }

  piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
  pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
   pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
  }

  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Poly  (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

 for(i = 0 ; i < iRows * iCols ; i++)
 {
  free(pdblReal[i]);
  free(pdblImg[i]);
 }
 free(pdblReal);
 free(pdblImg);
 free(piCoeff);
 return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;

 int* piBool     = NULL;
```

```
 if(_iItemPos == 0)
 {
  sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
 }
 else
 {
  sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean (%d x %d)\n", iRows, iCols);
 return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;
 int iItem           = 0;

 int* piNbRow        = NULL;
 int* piColPos       = NULL;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//Not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &p
  }
  else
  {
   sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  }
 }

 insert_indent();
 sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
```

```
 return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;
 int iItem       = 0;

 int* piNbRow    = NULL;
 int* piColPos   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &p
 }
 else
 {
  sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows,
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iPrec               = 0;
 int iRows               = 0;
 int iCols               = 0;

 char* pcData            = NULL;
 short* psData           = NULL;
 int* piData             = NULL;
 unsigned char* pucData  = NULL;
 unsigned short* pusData = NULL;
 unsigned int* puiData   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  switch(iPrec)
  {
```

```
 case SCI_INT8 :
  sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
  break;
 case SCI_INT16 :
  sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
  break;
 case SCI_INT32 :
  sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
  break;
 case SCI_UINT8 :
  sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &puc
  break;
 case SCI_UINT16 :
  sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pu
  break;
 case SCI_UINT32 :
  sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &pu
  break;
 default :
  return 1;
 }
}
else
{
 sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 switch(iPrec)
 {
 case SCI_INT8 :
  sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
  break;
 case SCI_INT16 :
  sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, 
  break;
 case SCI_INT32 :
  sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, 
  break;
 case SCI_UINT8 :
  sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &
  break;
 case SCI_UINT16 :
  sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, 
  break;
 case SCI_UINT32 :
  sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, 
  break;
 default :
  return 1;
 }
}

if(sciErr.iErr)
{
```

```
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 if(iPrec > 10)
 {
  sciprint("Unsigned ");
 }

 sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

 return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRows       = 0;
 int iCols       = 0;
 int* piLen      = NULL;

 char **pstData  = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }

  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData)
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
 else
 {
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
```

```
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }

  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Strings (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 void* pvPtr     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
 }
 else
 {
  sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
```

```
 }

 insert_indent();
 sciprint("Pointer : 0x%08X\n", pvPtr);

 return 0;
}
```

## Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2,-1,0,0,0;-1,2,-1,0,0;0,-1,2,-1,0;0,0,-1,2,-1;0,0,0,-1,2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with","you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

# Name

Boolean writing (Scilab gateway) — How to add matrix of boolean in a list.

Create from existing data.

Input argument profile:

```
SciErr createMatrixOfBooleanInList(void* _pvCtx, int _iVar, int* _piParent, int
```

Named variable profile:

```
SciErr createMatrixOfBooleanInNamedList(void* _pvCtx, char* _pstName, int* _piP
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_piBool
: Address of data array (size: _iRows * _iCols).

SciErr
: Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocMatrixOfBooleanInList(void* _pvCtx, int _iVar, int* _piParent, int
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_piBool
: Returns address of data array (size: _iRows * _iCols).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to add matrix of boolean in a list.

# Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int *piAddr             = NULL;
 int* piChild            = NULL;
 double pdblData1[]      = {1,3,5,2,4,6};
 double pdblData2[]      = {6,4,2,5,3,1};
 char *pstData[]         = {"may","be","the","with","puffin","you"};
 short psData[]          = {1,4,2,5,3,6};
 double pdblPoly1[]      = {1};
 double pdblPoly2[]      = {-2,-1};
 double pdblPoly3[]      = {1,2,3};
 double pdblPoly4[]      = {-4,-3,-2,-1};
 double pdblPoly5[]      = {1,2,3,4,5};
 double pdblPoly6[]      = {-6,-5,-4,-3,-2,-1};
 double *pdblPoly[]      = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPol
 int piCoef[]            = {1,3,5,2,4,6};
 int piNbItemRow[]       = {1,2,1};
 int piColPos[]          = {8,4,7,2};
 double pdblSReal[]      = {1,2,3,4};
 double pdblSImg[]       = {4,3,2,1};
 int piBool[]            = {1,0,1,0,1,0,1,0,1};
 double* pdblDataPtr     = NULL;


 sciErr = createList(pvApiCtx, 1, 8, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }
```

```
sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstData
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psDa
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblDa
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
```

```
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblDataPtr     = (double*)malloc(sizeof(double) * 4);
 pdblDataPtr[0]  = 1;
 pdblDataPtr[1]  = 2;
 pdblDataPtr[2]  = 3;
 pdblDataPtr[3]  = 4;

 sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = 1;
 return 0;
}
```

## Scilab test script

```
size_ref    = 8;
type_ref    = ["constant","string","int16","polynomial", "sparse", "boolean", "
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

# Name

Boolean sparse reading (Scilab gateway) — How to read boolean sparse in a list.

Input argument profile:

```
SciErr getBooleanSparseMatrixInList(void* _pvCtx, int* _piParent, int _iItemPos
```

Named variable profile:

```
SciErr readBooleanSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int* _p
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_piParent
: Address of the parent of the new item.

_pstName
: Name of the variable for "named" functions.

_iItemPos
: Position of the new item in the list.

_piRows
: Return number of rows of the variable.

_piCols
: Return number of columns of the variable.

_piNbItem
: Return number of non zero value.

_piNbItemRow
: Return number of item in each rows (size: _iRows).

_piColPos
: Return column position for each item (size: _iNbItem).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to read boolean sparse in a list.

# Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
 int i = 0;
 for(i = 0 ; i < iTab ; i++)
 {
  sciprint("\t");
 }
```

```
}


int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int iItem       = 0;
 int iRet        = 0;
 int *piAddr     = NULL;

 CheckRhs(1,1);

 sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 get_info(1, NULL, piAddr, 0);

 LhsVar(1) = 0;
 return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRet    = 0;
 int iType   = 0;

 sciErr = getVarType(pvApiCtx, _piAddr, &iType);
 switch(iType)
 {
 case sci_matrix :
  iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_poly :
  iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean :
  iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_sparse :
  iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean_sparse :
```

```
  iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_ints :
  iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_strings :
  iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_list :
  insert_indent();
  sciprint("List ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_tlist :
  insert_indent();
  sciprint("TList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_mlist :
  insert_indent();
  sciprint("MList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_pointer :
  iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 default :
  insert_indent();
  sciprint("Unknow type\n");
  return 1;
 }
 return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRet        = 0;
 int iItem       = 0;
 int* piChild    = NULL;

 sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciprint("(%d)\n", iItem);
 for(i = 0 ; i < iItem ; i++)
 {
  sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
```

```
  iTab++;
  iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
  iTab--;
 }

 return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblRea
  }
  else
  {
   sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRow
  }
  else
  {
   sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Double (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iLen            = 0;
```

```
int iRows           = 0;
int iCols           = 0;

char pstVar[16];
int* piCoeff        = NULL;
double** pdblReal   = NULL;
double** pdblImg    = NULL;

sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

if(_iItemPos == 0)
{//not in list
 sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 piCoeff     = (int*)malloc(sizeof(int) * iRows * iCols);
 sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
 pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
  pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
 }

 if(isVarComplex(pvApiCtx, _piAddr))
 {
  sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, p
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
 else
 {
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblRea
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
```

```
 }
 else
 {
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
  pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
   pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
  }

  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Poly  (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

 for(i = 0 ; i < iRows * iCols ; i++)
 {
  free(pdblReal[i]);
  free(pdblImg[i]);
 }
 free(pdblReal);
 free(pdblImg);
 free(piCoeff);
 return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
```

```
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;

 int* piBool     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
 }
 else
 {
  sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean (%d x %d)\n", iRows, iCols);
 return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;
 int iItem           = 0;

 int* piNbRow        = NULL;
 int* piColPos       = NULL;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//Not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &
  }
  else
  {
   sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
```

```
   sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  }
 }

 insert_indent();
 sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;
 int iItem       = 0;

 int* piNbRow    = NULL;
 int* piColPos   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &p
 }
 else
 {
  sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows,
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iPrec               = 0;
 int iRows               = 0;
 int iCols               = 0;

 char* pcData            = NULL;
 short* psData           = NULL;
 int* piData             = NULL;
 unsigned char* pucData  = NULL;
 unsigned short* pusData = NULL;
 unsigned int* puiData   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
  if(sciErr.iErr)
  {
```

```
   printError(&sciErr, 0);
   return 0;
  }

  switch(iPrec)
  {
  case SCI_INT8 :
   sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
   break;
  case SCI_INT16 :
   sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
   break;
  case SCI_INT32 :
   sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
   break;
  case SCI_UINT8 :
   sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &puc
   break;
  case SCI_UINT16 :
   sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pu
   break;
  case SCI_UINT32 :
   sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &pu
   break;
  default :
   return 1;
  }
 }
 else
 {
  sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  switch(iPrec)
  {
  case SCI_INT8 :
   sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
   break;
  case SCI_INT16 :
   sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, 
   break;
  case SCI_INT32 :
   sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, 
   break;
  case SCI_UINT8 :
   sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &
   break;
  case SCI_UINT16 :
   sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, 
   break;
  case SCI_UINT32 :
   sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, 
   break;
  default :
```

```
   return 1;
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 if(iPrec > 10)
 {
  sciprint("Unsigned ");
 }

 sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

 return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRows       = 0;
 int iCols       = 0;
 int* piLen      = NULL;

 char **pstData  = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }

  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData)
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
```

```
   return 0;
  }
 }
 else
 {
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }

  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Strings (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 void* pvPtr     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
 }
 else
 {
  sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
```

```
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Pointer : 0x%08X\n", pvPtr);

 return 0;
}
```

## Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2,-1,0,0,0;-1,2,-1,0,0;0,-1,2,-1,0;0,0,-1,2,-1;0,0,0,-1,2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with","you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

# Name

Boolean sparse writing (Scilab gateway) — How to add boolean sparse matrix in a list.

Input argument profile:

```
SciErr createBooleanSparseMatrixInList(void* _pvCtx, int _iVar, int* _piParent,
```

Named variable profile:

```
SciErr createBooleanSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int* 
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_iNbItem
: Number of non zero itmes in the sparse.

_piNbItemRow
: Number of item in each rows (size: _iRows).

_piColPos
: Column position for each item (size: _iNbItem).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to add boolean sparse matrix in a list.

# Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int *piAddr             = NULL;
 int* piChild            = NULL;
 double pdblData1[]      = {1,3,5,2,4,6};
 double pdblData2[]      = {6,4,2,5,3,1};
```

```
char *pstData[]         = {"may","be","the","with","puffin","you"};
short psData[]          = {1,4,2,5,3,6};
double pdblPoly1[]      = {1};
double pdblPoly2[]      = {-2,-1};
double pdblPoly3[]      = {1,2,3};
double pdblPoly4[]      = {-4,-3,-2,-1};
double pdblPoly5[]      = {1,2,3,4,5};
double pdblPoly6[]      = {-6,-5,-4,-3,-2,-1};
double *pdblPoly[]      = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPol
int piCoef[]            = {1,3,5,2,4,6};
int piNbItemRow[]       = {1,2,1};
int piColPos[]          = {8,4,7,2};
double pdblSReal[]      = {1,2,3,4};
double pdblSImg[]       = {4,3,2,1};
int piBool[]            = {1,0,1,0,1,0,1,0,1};
double* pdblDataPtr     = NULL;


sciErr = createList(pvApiCtx, 1, 8, &piAddr);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstDat
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psD
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10,
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
```

```
 }

 sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, 
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 //add list in list
 sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblDataPtr     = (double*)malloc(sizeof(double) * 4);
 pdblDataPtr[0]  = 1;
 pdblDataPtr[1]  = 2;
 pdblDataPtr[2]  = 3;
 pdblDataPtr[3]  = 4;

 sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = 1;
 return 0;
}
```

## Scilab test script

```
size_ref    = 8;
type_ref    = ["constant","string","int16","polynomial", "sparse", "boolean", "
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

# Name

Create List (Scilab gateway) — How to get create a list in Scilab memory.

Input argument profile:

```
SciErr createList(void* _pvCtx, int _iVar, int _iNbItem, int** _piAddress)
```

```
SciErr createMList(void* _pvCtx, int _iVar, int _iNbItem, int** _piAddress)
```

```
SciErr createTList(void* _pvCtx, int _iVar, int _iNbItem, int** _piAddress)
```

Named variable profile:

```
SciErr createNamedList(void* _pvCtx, char* _pstName, int _iNbItem, int** _piAdd
```

```
SciErr createNamedTList(void* _pvCtx, char* _pstName, int _iNbItem, int** _piAd
```

```
SciErr createNamedMList(void* _pvCtx, char* _pstName, int _iNbItem, int** _piAd
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_iNbItem
: Number of items in the list.

_piAddress
: Return address of the list.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to get the address of a list in a list.

# Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int *piAddr             = NULL;
 int* piChild            = NULL;
 double pdblData1[]      = {1,3,5,2,4,6};
 double pdblData2[]      = {6,4,2,5,3,1};
 char *pstData[]         = {"may","be","the","with","puffin","you"};
 short psData[]          = {1,4,2,5,3,6};
 double pdblPoly1[]      = {1};
 double pdblPoly2[]      = {-2,-1};
 double pdblPoly3[]      = {1,2,3};
 double pdblPoly4[]      = {-4,-3,-2,-1};
```

```
double pdblPoly5[]      = {1,2,3,4,5};
double pdblPoly6[]      = {-6,-5,-4,-3,-2,-1};
double *pdblPoly[]      = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPol
int piCoef[]            = {1,3,5,2,4,6};
int piNbItemRow[]       = {1,2,1};
int piColPos[]          = {8,4,7,2};
double pdblSReal[]      = {1,2,3,4};
double pdblSImg[]       = {4,3,2,1};
int piBool[]            = {1,0,1,0,1,0,1,0,1};
double* pdblDataPtr     = NULL;


sciErr = createList(pvApiCtx, 1, 8, &piAddr);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstDat
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psD
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10,
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
 printError(&sciErr, 0);
```

```
  return 0;
 }

 sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, 
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 //add list in list
 sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblDataPtr     = (double*)malloc(sizeof(double) * 4);
 pdblDataPtr[0]  = 1;
 pdblDataPtr[1]  = 2;
 pdblDataPtr[2]  = 3;
 pdblDataPtr[3]  = 4;

 sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = 1;
 return 0;
}
```

## Scilab test script

```
size_ref    = 8;
type_ref    = ["constant","string","int16","polynomial", "sparse", "boolean", "
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);
```

```
l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

# Name

Double reading (Scilab gateway) — How to read matrix of double in a list.

Input argument profile:

```
SciErr getMatrixOfDoubleInList(void* _pvCtx, int* _piParent, int _iItemPos, int
```

```
SciErr getComplexMatrixOfDoubleInList(void* _pvCtx, int* _piParent, int _iItemP
```

Named variable profile:

```
SciErr readMatrixOfDoubleInNamedList(void* _pvCtx, char* _pstName, int* _piPare
```

```
SciErr readComplexMatrixOfDoubleInNamedList(void* _pvCtx, char* _pstName, int* 
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piParent
: Address of the parent of the new item.

_pstName
: Name of the variable for "named" functions.

_iItemPos
: Position of the new item in the list.

_piRows
: Return number of rows of the variable.

_piCols
: Return number of columns of the variable.

_pdblReal
: Return address of real part data array (size: _iRows * _iCols).

_pdblImg
: Return address of imaginary part data array (size: _iRows * _iCols).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to read matrix of double in a list.

# Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
 int i = 0;
 for(i = 0 ; i < iTab ; i++)
 {
  sciprint("\t");
 }
```

```
}


int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int iItem       = 0;
 int iRet        = 0;
 int *piAddr     = NULL;

 CheckRhs(1,1);

 sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 get_info(1, NULL, piAddr, 0);

 LhsVar(1) = 0;
 return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRet    = 0;
 int iType   = 0;

 sciErr = getVarType(pvApiCtx, _piAddr, &iType);
 switch(iType)
 {
 case sci_matrix :
  iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_poly :
  iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean :
  iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_sparse :
  iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean_sparse :
```

```
  iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_ints :
  iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_strings :
  iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_list :
  insert_indent();
  sciprint("List ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_tlist :
  insert_indent();
  sciprint("TList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_mlist :
  insert_indent();
  sciprint("MList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_pointer :
  iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 default :
  insert_indent();
  sciprint("Unknow type\n");
  return 1;
 }
 return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRet        = 0;
 int iItem       = 0;
 int* piChild    = NULL;

 sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciprint("(%d)\n", iItem);
 for(i = 0 ; i < iItem ; i++)
 {
  sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
```

```
  iTab++;
  iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
  iTab--;
 }

 return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblRea
  }
  else
  {
   sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRow
  }
  else
  {
   sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Double (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iLen            = 0;
```

```
int iRows           = 0;
int iCols           = 0;

char pstVar[16];
int* piCoeff        = NULL;
double** pdblReal   = NULL;
double** pdblImg    = NULL;

sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

if(_iItemPos == 0)
{//not in list
 sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 piCoeff     = (int*)malloc(sizeof(int) * iRows * iCols);
 sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
 pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
  pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
 }

 if(isVarComplex(pvApiCtx, _piAddr))
 {
  sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, p
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
 else
 {
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblRea
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
```

```
 }
 else
 {
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
  pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
   pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
  }

  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Poly  (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

 for(i = 0 ; i < iRows * iCols ; i++)
 {
  free(pdblReal[i]);
  free(pdblImg[i]);
 }
 free(pdblReal);
 free(pdblImg);
 free(piCoeff);
 return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
```

```
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;

 int* piBool     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
 }
 else
 {
  sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean (%d x %d)\n", iRows, iCols);
 return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;
 int iItem           = 0;

 int* piNbRow        = NULL;
 int* piColPos       = NULL;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//Not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &
  }
  else
  {
   sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
```

```
   sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  }
 }

 insert_indent();
 sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;
 int iItem       = 0;

 int* piNbRow    = NULL;
 int* piColPos   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &p
 }
 else
 {
  sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows,
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iPrec               = 0;
 int iRows               = 0;
 int iCols               = 0;

 char* pcData            = NULL;
 short* psData           = NULL;
 int* piData             = NULL;
 unsigned char* pucData  = NULL;
 unsigned short* pusData = NULL;
 unsigned int* puiData   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
  if(sciErr.iErr)
  {
```

```
   printError(&sciErr, 0);
   return 0;
  }

  switch(iPrec)
  {
  case SCI_INT8 :
   sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
   break;
  case SCI_INT16 :
   sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
   break;
  case SCI_INT32 :
   sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
   break;
  case SCI_UINT8 :
   sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &puc
   break;
  case SCI_UINT16 :
   sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pu
   break;
  case SCI_UINT32 :
   sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &pu
   break;
  default :
   return 1;
  }
 }
 else
 {
  sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  switch(iPrec)
  {
  case SCI_INT8 :
   sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
   break;
  case SCI_INT16 :
   sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, 
   break;
  case SCI_INT32 :
   sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, 
   break;
  case SCI_UINT8 :
   sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &
   break;
  case SCI_UINT16 :
   sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, 
   break;
  case SCI_UINT32 :
   sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, 
   break;
  default :
```

```
   return 1;
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 if(iPrec > 10)
 {
  sciprint("Unsigned ");
 }

 sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

 return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRows       = 0;
 int iCols       = 0;
 int* piLen      = NULL;

 char **pstData  = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }

  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData)
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
```

```
   return 0;
  }
 }
 else
 {
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }

  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Strings (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 void* pvPtr     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
 }
 else
 {
  sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
```

```
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Pointer : 0x%08X\n", pvPtr);

 return 0;
}
```

## Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2,-1,0,0,0;-1,2,-1,0,0;0,-1,2,-1,0;0,0,-1,2,-1;0,0,0,-1,2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with","you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

# Name

Double writing (Scilab gateway) — How to add matrix of double in a list.

Create from existing data.

Input argument profile:

```
SciErr createMatrixOfBooleanInList(void* _pvCtx, int _iVar, int* _piParent, int
```

Named variable profile:

```
SciErr createMatrixOfBooleanInNamedList(void* _pvCtx, char* _pstName, int* _piP
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_pdblReal
: Address of real data array (size: _iCols * _iRows).

_pdblImg
: Address of imaginary data array (size: _iCols * _iRows).

SciErr
: Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocMatrixOfDoubleInList(void* _pvCtx, int _iVar, int* _piParent, int _
```

```
SciErr allocComplexMatrixOfDoubleInList(void* _pvCtx, int _iVar, int* _piParent
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_pdblReal
: Return address of real data array (size: _iCols * _iRows).

_pdblImg
: Return address of imaginary data array (size: _iCols * _iRows).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to add matrix of double in a list.

Two types of functions can be used to write in the memory of Scilab.

# Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int *piAddr             = NULL;
 int* piChild            = NULL;
 double pdblData1[]      = {1,3,5,2,4,6};
 double pdblData2[]      = {6,4,2,5,3,1};
 char *pstData[]         = {"may","be","the","with","puffin","you"};
 short psData[]          = {1,4,2,5,3,6};
 double pdblPoly1[]      = {1};
 double pdblPoly2[]      = {-2,-1};
 double pdblPoly3[]      = {1,2,3};
 double pdblPoly4[]      = {-4,-3,-2,-1};
 double pdblPoly5[]      = {1,2,3,4,5};
 double pdblPoly6[]      = {-6,-5,-4,-3,-2,-1};
 double *pdblPoly[]      = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPol
 int piCoef[]            = {1,3,5,2,4,6};
 int piNbItemRow[]       = {1,2,1};
 int piColPos[]          = {8,4,7,2};
 double pdblSReal[]      = {1,2,3,4};
 double pdblSImg[]       = {4,3,2,1};
 int piBool[]            = {1,0,1,0,1,0,1,0,1};
 double* pdblDataPtr     = NULL;


 sciErr = createList(pvApiCtx, 1, 8, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }
```

```
sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstData
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psDa
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblDa
```

```
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblDataPtr     = (double*)malloc(sizeof(double) * 4);
 pdblDataPtr[0]  = 1;
 pdblDataPtr[1]  = 2;
 pdblDataPtr[2]  = 3;
 pdblDataPtr[3]  = 4;

 sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = 1;
 return 0;
}
```

## Scilab test script

```
size_ref    = 8;
type_ref    = ["constant","string","int16","polynomial", "sparse", "boolean", "
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

# Name

Get child item (Scilab gateway) — How to get the address of a list child.

```
SciErr getListItemAddress(void* _pvCtx, int* _piAddress, int _iItemNum, int** _
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piAddress
: Address of the list.

_iItemNum
: Item number.

_piItemAddress
: Return address of the item.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to get the address of a list child.

# Gateway Source

```
int get_list_info(int* _piAddress);
void insert_indent(void);

static int iLocalTab = 0;

int common_list(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr     = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_list_info(piAddr);

    LhsVar(1) = 0;
    return 0;
}

int get_list_info(int* _piAddress)
{
    SciErr sciErr;
```

```
    int i       = 0;
    int iRet    = 0;
    int iItem   = 0;

    //get list item number, failed if variable is not a kind of list
    sciErr = getListItemNumber(pvApiCtx, _piAddress, &iItem);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        sciprint("This variable is not a list");
        return 0;
    }

    sciprint("List (%d items) -> address : 0x%08X) : \n", iItem, _piAddress);
    for(i = 0 ; i < iItem ; i++)
    {
        int iType           = 0;
        int* piAddrChild    = NULL;
        sciErr = getListItemAddress(pvApiCtx, _piAddress, i + 1, &piAddrChild);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        sciErr = getVarType(pvApiCtx, piAddrChild, &iType);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        if(iType == sci_list || iType == sci_tlist || iType == sci_mlist)
        {
            insert_indent();
            sciprint("Child %d -> ", i + 1);
            iLocalTab++;
            iRet = get_list_info(piAddrChild);
            iLocalTab--;
            if(iRet)
            {
                return 1;
            }
        }
        else
        {
            insert_indent();
            sciprint("Child %d -> address : 0x%08X\n", i + 1, piAddrChild);
        }
    }
    return 0;
}

void insert_indent(void)
{
    int i = 0;
    for(i = 0 ; i < iLocalTab ; i++)
    {
```

```
        sciprint("\t");
    }
}
```

## Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];
l2 = ["may","the";"puffin","be";"with","you"];
l3 = int8([1,2,3]);
l5 = list(l1,l2,l3);
l4 = list(l5, list(l5,l5));
l6 = uint16([1000,2000,3000]);
l = list(l1,l2,l3,l6,l4,l5);
common_list(l)
```

# Name

Item Number (Scilab gateway) — How to get the number of items in a list (list, mlist, tlist).

```
SciErr getListItemNumber(void* _pvCtx, int* _piAddress, int* _piNbItem)
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piAddress
: Address of the list.

_piNbItem
: Return the number of items.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to get the number of items in a list (list, mlist, tlist).

# Gateway Source

```
int get_list_info(int* _piAddress);
void insert_indent(void);

static int iLocalTab = 0;

int common_list(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int *piAddr     = NULL;

    CheckRhs(1,1);

    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    get_list_info(piAddr);

    LhsVar(1) = 0;
    return 0;
}

int get_list_info(int* _piAddress)
{
    SciErr sciErr;
    int i       = 0;
    int iRet    = 0;
```

```
    int iItem   = 0;

    //get list item number, failed if variable is not a kind of list
    sciErr = getListItemNumber(pvApiCtx, _piAddress, &iItem);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        sciprint("This variable is not a list");
        return 0;
    }

    sciprint("List (%d items) -> address : 0x%08X) : \n", iItem, _piAddress);
    for(i = 0 ; i < iItem ; i++)
    {
        int iType           = 0;
        int* piAddrChild    = NULL;
        sciErr = getListItemAddress(pvApiCtx, _piAddress, i + 1, &piAddrChild);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        sciErr = getVarType(pvApiCtx, piAddrChild, &iType);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            return 0;
        }

        if(iType == sci_list || iType == sci_tlist || iType == sci_mlist)
        {
            insert_indent();
            sciprint("Child %d -> ", i + 1);
            iLocalTab++;
            iRet = get_list_info(piAddrChild);
            iLocalTab--;
            if(iRet)
            {
                return 1;
            }
        }
        else
        {
            insert_indent();
            sciprint("Child %d -> address : 0x%08X\n", i + 1, piAddrChild);
        }
    }
    return 0;
}

void insert_indent(void)
{
    int i = 0;
    for(i = 0 ; i < iLocalTab ; i++)
    {
        sciprint("\t");
    }
```

```
}
```

## Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];
l2 = ["may","the";"puffin","be";"with","you"];
l3 = int8([1,2,3]);
l5 = list(l1,l2,l3);
l4 = list(l5, list(l5,l5));
l6 = uint16([1000,2000,3000]);
l = list(l1,l2,l3,l6,l4,l5);
common_list(l)
```

# Name

Integer reading (Scilab gateway) — How to read matrix of integer in a list.

Input argument profile:

Signed integer :

```
SciErr getMatrixOfInteger8InList(void* _pvCtx, int* _piParent, int _iItemPos, i
SciErr getMatrixOfInteger16InList(void* _pvCtx, int* _piParent, int _iItemPos,
SciErr getMatrixOfInteger32InList(void* _pvCtx, int* _piParent, int _iItemPos,
```

Unsigned integer :

```
SciErr getMatrixOfUnsignedInteger8InList(void* _pvCtx, int* _piParent, int _iIt
SciErr getMatrixOfUnsignedInteger16InList(void* _pvCtx, int* _piParent, int _iI
SciErr getMatrixOfUnsignedInteger32InList(void* _pvCtx, int* _piParent, int _iI
```

Named variable profile:

Signed integer :

```
SciErr readMatrixOfIntger8InNamedList(void* _pvCtx, char* _pstName, int* _piPar
SciErr readMatrixOfIntger16InNamedList(void* _pvCtx, char* _pstName, int* _piPa
SciErr readMatrixOfIntger32InNamedList(void* _pvCtx, char* _pstName, int* _piPa
```

Unsigned integer :

```
SciErr readMatrixOfUnsignedInteger8InNamedList(void* _pvCtx, char* _pstName, in
SciErr readMatrixOfUnsignedInteger16InNamedList(void* _pvCtx, char* _pstName, i
SciErr readMatrixOfUnsignedInteger32InNamedList(void* _pvCtx, char* _pstName, i
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piParent
: Address of the parent of the new item.

_pstName
: Name of the variable for "named" functions.

_iItemPos
: Position of the new item in the list.

_piRows
: Return number of rows of the variable.

_piCols
: Return number of columns of the variable.

_pcData, _pucData, _psData, _pusData, _piData, _puiData
: Return address of data array (size: _iRows * _iCols).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to read matrix of integer in a list.

# Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
 int i = 0;
 for(i = 0 ; i < iTab ; i++)
 {
  sciprint("\t");
 }
}


int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int iItem       = 0;
 int iRet        = 0;
 int *piAddr     = NULL;

 CheckRhs(1,1);

 sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 get_info(1, NULL, piAddr, 0);

 LhsVar(1) = 0;
 return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRet    = 0;
```

```
 int iType   = 0;

 sciErr = getVarType(pvApiCtx, _piAddr, &iType);
 switch(iType)
 {
 case sci_matrix :
  iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_poly :
  iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean :
  iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_sparse :
  iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean_sparse :
  iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_ints :
  iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_strings :
  iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_list :
  insert_indent();
  sciprint("List ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_tlist :
  insert_indent();
  sciprint("TList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_mlist :
  insert_indent();
  sciprint("MList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_pointer :
  iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 default :
  insert_indent();
  sciprint("Unknow type\n");
  return 1;
 }
 return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRet        = 0;
 int iItem       = 0;
```

```
 int* piChild    = NULL;

 sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciprint("(%d)\n", iItem);
 for(i = 0 ; i < iItem ; i++)
 {
  sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  iTab++;
  iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
  iTab--;
 }

 return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblRea
  }
  else
  {
   sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRo
  }
  else
  {
   sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
  }
 }
```

```
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Double (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iLen            = 0;
 int iRows           = 0;
 int iCols           = 0;

 char pstVar[16];
 int* piCoeff        = NULL;
 double** pdblReal   = NULL;
 double** pdblImg    = NULL;

 sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 if(_iItemPos == 0)
 {//not in list
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piCoeff     = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
  pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
   pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
  }
```

```
 if(isVarComplex(pvApiCtx, _piAddr))
 {
  sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, p
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
 else
 {
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblRea
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
}
else
{
 sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
 sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
 pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
  pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
 }

 if(isVarComplex(pvApiCtx, _piAddr))
 {
  sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows
 }
 else
 {
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
 }
}

if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
```

```
 }

 insert_indent();
 sciprint("Poly  (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

 for(i = 0 ; i < iRows * iCols ; i++)
 {
  free(pdblReal[i]);
  free(pdblImg[i]);
 }
 free(pdblReal);
 free(pdblImg);
 free(piCoeff);
 return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;

 int* piBool     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
 }
 else
 {
  sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean (%d x %d)\n", iRows, iCols);
 return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;
 int iItem           = 0;

 int* piNbRow        = NULL;
 int* piColPos       = NULL;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
```

```
 {//Not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &p
  }
  else
  {
   sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  }
 }

 insert_indent();
 sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;
 int iItem       = 0;

 int* piNbRow    = NULL;
 int* piColPos   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &pi
 }
 else
 {
  sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows,
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
```

```
{
 SciErr sciErr;
 int iPrec               = 0;
 int iRows               = 0;
 int iCols               = 0;

 char* pcData            = NULL;
 short* psData           = NULL;
 int* piData             = NULL;
 unsigned char* pucData  = NULL;
 unsigned short* pusData = NULL;
 unsigned int* puiData   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  switch(iPrec)
  {
  case SCI_INT8 :
   sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
   break;
  case SCI_INT16 :
   sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
   break;
  case SCI_INT32 :
   sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
   break;
  case SCI_UINT8 :
   sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &puc
   break;
  case SCI_UINT16 :
   sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pu
   break;
  case SCI_UINT32 :
   sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &pu
   break;
  default :
   return 1;
  }
 }
 else
 {
  sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  switch(iPrec)
  {
  case SCI_INT8 :
```

```
   sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
   break;
  case SCI_INT16 :
   sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, 
   break;
  case SCI_INT32 :
   sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, 
   break;
  case SCI_UINT8 :
   sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &
   break;
  case SCI_UINT16 :
   sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, 
   break;
  case SCI_UINT32 :
   sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, 
   break;
  default :
   return 1;
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 if(iPrec > 10)
 {
  sciprint("Unsigned ");
 }

 sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

 return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRows       = 0;
 int iCols       = 0;
 int* piLen      = NULL;

 char **pstData  = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
```

```
 piLen = (int*)malloc(sizeof(int) * iRows * iCols);
 sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
 }

 sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData)
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }
}
else
{
 sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 piLen = (int*)malloc(sizeof(int) * iRows * iCols);
 sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
 }

 sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }
}

if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}
```

```
 insert_indent();
 sciprint("Strings (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 void* pvPtr     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
 }
 else
 {
  sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Pointer : 0x%08X\n", pvPtr);

 return 0;
}
```

## Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2,-1,0,0,0;-1,2,-1,0,0;0,-1,2,-1,0;0,0,-1,2,-1;0,0,0,-1,2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with","you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

# Name

Integer writing (Scilab gateway) — How to add matrix of integer in a list.

Create from existing data.

Input argument profile:

Signed integer :

```
SciErr createMatrixOfInteger8InList(void* _pvCtx, int _iVar, int* _piParent, in
SciErr createMatrixOfInteger16InList(void* _pvCtx, int _iVar, int* _piParent, i
SciErr createMatrixOfInteger32InList(void* _pvCtx, int _iVar, int* _piParent, i
```

Unsigned integer :

```
SciErr createMatrixOfUnsignedInteger8InList(void* _pvCtx, int _iVar, int* _piPa
SciErr createMatrixOfUnsignedInteger16InList(void* _pvCtx, int _iVar, int* _piP
SciErr createMatrixOfUnsignedInteger32InList(void* _pvCtx, int _iVar, int* _piP
```

Named variable profile:

Signed integer :

```
SciErr createMatrixOfInteger8InNamedList(void* _pvCtx, char* _pstName, int* _pi
SciErr createMatrixOfInteger16InNamedList(void* _pvCtx, char* _pstName, int* _p
SciErr createMatrixOfInteger32InNamedList(void* _pvCtx, char* _pstName, int* _p
```

Unsigned integer :

```
SciErr createMatrixOfUnsignedInteger8InNamedList(void* _pvCtx, char* _pstName,
SciErr createMatrixOfUnsignedInteger16InNamedList(void* _pvCtx, char* _pstName,
SciErr createMatrixOfUnsignedInteger32InNamedList(void* _pvCtx, char* _pstName,
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_pcData8, _psData16, _piData32, _pucData8, _pusData16, _puiData32
: Address of data array (size: _iCols * _iRows)

SciErr
: Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

Signed integer :

```
SciErr allocMatrixOfInteger8InList(void* _pvCtx, int _iVar, int* _piParent, int
```

```
SciErr allocMatrixOfInteger16InList(void* _pvCtx, int _iVar, int* _piParent, in
```

```
SciErr allocMatrixOfInteger32InList(void* _pvCtx, int _iVar, int* _piParent, in
```

Unsigned integer :

```
SciErr allocMatrixOfUnsignedInteger8InList(void* _pvCtx, int _iVar, int* _piPar
```

```
SciErr allocMatrixOfUnsignedInteger16InList(void* _pvCtx, int _iVar, int* _piPa
```

```
SciErr allocMatrixOfUnsignedInteger32InList(void* _pvCtx, int _iVar, int* _piPa
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_pcData8, _psData16, _piData32, _pucData8, _pusData16, _puiData32
: Return address of data array (size: _iCols * _iRows)

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to add matrix of integer in a list.

Two types of functions can be used to write in the memory of Scilab.

## Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int *piAddr             = NULL;
 int* piChild            = NULL;
 double pdblData1[]      = {1,3,5,2,4,6};
 double pdblData2[]      = {6,4,2,5,3,1};
 char *pstData[]         = {"may","be","the","with","puffin","you"};
 short psData[]          = {1,4,2,5,3,6};
 double pdblPoly1[]      = {1};
 double pdblPoly2[]      = {-2,-1};
 double pdblPoly3[]      = {1,2,3};
 double pdblPoly4[]      = {-4,-3,-2,-1};
 double pdblPoly5[]      = {1,2,3,4,5};
 double pdblPoly6[]      = {-6,-5,-4,-3,-2,-1};
 double *pdblPoly[]      = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPol
 int piCoef[]            = {1,3,5,2,4,6};
 int piNbItemRow[]       = {1,2,1};
 int piColPos[]          = {8,4,7,2};
 double pdblSReal[]      = {1,2,3,4};
 double pdblSImg[]       = {4,3,2,1};
 int piBool[]            = {1,0,1,0,1,0,1,0,1};
 double* pdblDataPtr     = NULL;


 sciErr = createList(pvApiCtx, 1, 8, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstDat
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psD
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }
```

```
sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

pdblDataPtr     = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0]  = 1;
pdblDataPtr[1]  = 2;
pdblDataPtr[2]  = 3;
pdblDataPtr[3]  = 4;

sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
if(sciErr.iErr)
```

```
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = 1;
 return 0;
}
```

## Scilab test script

```
size_ref    = 8;
type_ref    = ["constant","string","int16","polynomial", "sparse", "boolean", "
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

# Name

Pointer reading (Scilab gateway) — How to read pointer in a list.

Input argument profile:

```
SciErr getPointerInList(void* _pvCtx, int* _piParent, int _iItemPos, void** _pv
```

Named variable profile:

```
SciErr readPointerInNamedList(void* _pvCtx, char* _pstName, int* _piParent, int
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_pvPtr
: Return pointer on data.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to read pointer in a list.

# Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
 int i = 0;
 for(i = 0 ; i < iTab ; i++)
 {
  sciprint("\t");
 }
}


int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
```

```
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int iItem       = 0;
 int iRet        = 0;
 int *piAddr     = NULL;

 CheckRhs(1,1);

 sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 get_info(1, NULL, piAddr, 0);

 LhsVar(1) = 0;
 return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRet    = 0;
 int iType   = 0;

 sciErr = getVarType(pvApiCtx, _piAddr, &iType);
 switch(iType)
 {
 case sci_matrix :
  iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_poly :
  iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean :
  iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_sparse :
  iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean_sparse :
  iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_ints :
  iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_strings :
  iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
```

```
 case sci_list :
  insert_indent();
  sciprint("List ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_tlist :
  insert_indent();
  sciprint("TList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_mlist :
  insert_indent();
  sciprint("MList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_pointer :
  iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 default :
  insert_indent();
  sciprint("Unknow type\n");
  return 1;
 }
 return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRet        = 0;
 int iItem       = 0;
 int* piChild    = NULL;

 sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciprint("(%d)\n", iItem);
 for(i = 0 ; i < iItem ; i++)
 {
  sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  iTab++;
  iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
  iTab--;
 }

 return 0;;
}
```

```
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblRea
  }
  else
  {
   sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRo
  }
  else
  {
   sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Double (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iLen            = 0;
 int iRows           = 0;
 int iCols           = 0;

 char pstVar[16];
 int* piCoeff        = NULL;
 double** pdblReal   = NULL;
 double** pdblImg    = NULL;
```

```
sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

if(_iItemPos == 0)
{//not in list
 sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 piCoeff     = (int*)malloc(sizeof(int) * iRows * iCols);
 sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
 pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
  pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
 }

 if(isVarComplex(pvApiCtx, _piAddr))
 {
  sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, p
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
 else
 {
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblRea
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
}
else
{
 sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
```

```
  }

  piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
  pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
   pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
  }

  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Poly  (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

 for(i = 0 ; i < iRows * iCols ; i++)
 {
  free(pdblReal[i]);
  free(pdblImg[i]);
 }
 free(pdblReal);
 free(pdblImg);
 free(piCoeff);
 return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;

 int* piBool     = NULL;

 if(_iItemPos == 0)
 {
```

```
  sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
 }
 else
 {
  sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean (%d x %d)\n", iRows, iCols);
 return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;
 int iItem           = 0;

 int* piNbRow        = NULL;
 int* piColPos       = NULL;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//Not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &
  }
  else
  {
   sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
  }
 }

 insert_indent();
 sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}
```

```
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;
 int iItem       = 0;

 int* piNbRow    = NULL;
 int* piColPos   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &p
 }
 else
 {
  sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows,
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iPrec               = 0;
 int iRows               = 0;
 int iCols               = 0;

 char* pcData            = NULL;
 short* psData           = NULL;
 int* piData             = NULL;
 unsigned char* pucData  = NULL;
 unsigned short* pusData = NULL;
 unsigned int* puiData   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  switch(iPrec)
  {
  case SCI_INT8 :
   sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
```

```
  break;
 case SCI_INT16 :
  sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
  break;
 case SCI_INT32 :
  sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
  break;
 case SCI_UINT8 :
  sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &puc
  break;
 case SCI_UINT16 :
  sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pu
  break;
 case SCI_UINT32 :
  sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &pu
  break;
 default :
  return 1;
 }
}
else
{
 sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 switch(iPrec)
 {
 case SCI_INT8 :
  sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
  break;
 case SCI_INT16 :
  sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, 
  break;
 case SCI_INT32 :
  sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, 
  break;
 case SCI_UINT8 :
  sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &
  break;
 case SCI_UINT16 :
  sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, 
  break;
 case SCI_UINT32 :
  sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, 
  break;
 default :
  return 1;
 }
}

if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
```

```
 }

 insert_indent();
 if(iPrec > 10)
 {
  sciprint("Unsigned ");
 }

 sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

 return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRows       = 0;
 int iCols       = 0;
 int* piLen      = NULL;

 char **pstData  = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }

  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData)
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
 else
 {
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
```

```
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }

  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Strings (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 void* pvPtr     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
 }
 else
 {
  sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }
```

```
 insert_indent();
 sciprint("Pointer : 0x%08X\n", pvPtr);

 return 0;
}
```

## Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2,-1,0,0,0;-1,2,-1,0,0;0,-1,2,-1,0;0,0,-1,2,-1;0,0,0,-1,2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with","you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

# Name

Pointer writing (Scilab gateway) — How to add pointer in a list.

Input argument profile:

```
SciErr createPointerInList(void* _pvCtx, int _iVar, int* _piParent, int _iItemP
```

Named variable profile:

```
SciErr createPointerInNamedList(void* _pvCtx, char* _pstName, int* _piParent, i
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_pvPtr
: Address of the pointer.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to add pointer in a list.

# Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int *piAddr             = NULL;
 int* piChild            = NULL;
 double pdblData1[]      = {1,3,5,2,4,6};
 double pdblData2[]      = {6,4,2,5,3,1};
 char *pstData[]         = {"may","be","the","with","puffin","you"};
 short psData[]          = {1,4,2,5,3,6};
 double pdblPoly1[]      = {1};
 double pdblPoly2[]      = {-2,-1};
 double pdblPoly3[]      = {1,2,3};
 double pdblPoly4[]      = {-4,-3,-2,-1};
 double pdblPoly5[]      = {1,2,3,4,5};
 double pdblPoly6[]      = {-6,-5,-4,-3,-2,-1};
 double *pdblPoly[]      = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPol
 int piCoef[]            = {1,3,5,2,4,6};
```

```
int piNbItemRow[]       = {1,2,1};
int piColPos[]          = {8,4,7,2};
double pdblSReal[]      = {1,2,3,4};
double pdblSImg[]       = {4,3,2,1};
int piBool[]            = {1,0,1,0,1,0,1,0,1};
double* pdblDataPtr     = NULL;


sciErr = createList(pvApiCtx, 1, 8, &piAddr);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstData
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psDa
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, 
```

```
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 //add list in list
 sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblDataPtr     = (double*)malloc(sizeof(double) * 4);
 pdblDataPtr[0]  = 1;
 pdblDataPtr[1]  = 2;
 pdblDataPtr[2]  = 3;
 pdblDataPtr[3]  = 4;

 sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = 1;
 return 0;
}
```

## Scilab test script

```
size_ref    = 8;
type_ref    = ["constant","string","int16","polynomial", "sparse", "boolean", "
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
```

```
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

# Name

Polynomial reading (Scilab gateway) — How to read matrix of polynomial in a list.

Input argument profile:

```
SciErr getMatrixOfPolyInList(void* _pvCtx, int* _piParent, int _iItemPos, int* _
```

```
SciErr getComplexMatrixOfPolyInList(void* _pvCtx, int* _piParent, int _iItemPos
```

Named variable profile:

```
SciErr readMatrixOfPolyInNamedList(void* _pvCtx, char* _pstName, int* _piParent
```

```
SciErr readComplexMatrixOfPolyInNamedList(void* _pvCtx, char* _pstName, int* _p
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piParent
: Address of the parent of the new item.

_pstName
: Name of the variable for "named" functions.

_iItemPos
: Position of the new item in the list.

_piRows
: Return number of rows of the variable.

_piCols
: Return number of columns of the variable.

_piNbCoef
: Return number of coefficient for each polynomial. (must be allocated)

_pdblReal
: Address of array of double* with imaginary part of coefficient (size: _iCols * _iRows, must be allocated)

_pdblImg
: Address of array of double* with imaginary part of coefficient (size: _iCols * _iRows, must be allocated)

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to read matrix of polynomial in a list.

# Gateway Source

```


static int iTab = 0;

void insert_indent(void)
{
```

```
 int i = 0;
 for(i = 0 ; i < iTab ; i++)
 {
  sciprint("\t");
 }
}


int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int iItem       = 0;
 int iRet        = 0;
 int *piAddr     = NULL;

 CheckRhs(1,1);

 sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 get_info(1, NULL, piAddr, 0);

 LhsVar(1) = 0;
 return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRet    = 0;
 int iType   = 0;

 sciErr = getVarType(pvApiCtx, _piAddr, &iType);
 switch(iType)
 {
 case sci_matrix :
  iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_poly :
  iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean :
  iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
```

```
  break;
 case sci_sparse :
  iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean_sparse :
  iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_ints :
  iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_strings :
  iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_list :
  insert_indent();
  sciprint("List ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_tlist :
  insert_indent();
  sciprint("TList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_mlist :
  insert_indent();
  sciprint("MList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_pointer :
  iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 default :
  insert_indent();
  sciprint("Unknow type\n");
  return 1;
 }
 return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRet        = 0;
 int iItem       = 0;
 int* piChild    = NULL;

 sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciprint("(%d)\n", iItem);
 for(i = 0 ; i < iItem ; i++)
 {
  sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
```

```
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  iTab++;
  iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
  iTab--;
 }

 return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblRea
  }
  else
  {
   sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRow
  }
  else
  {
   sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Double (%d x %d)\n", iRows, iCols);

 return 0;;
}
```

```
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iLen            = 0;
 int iRows           = 0;
 int iCols           = 0;

 char pstVar[16];
 int* piCoeff        = NULL;
 double** pdblReal   = NULL;
 double** pdblImg    = NULL;

 sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 if(_iItemPos == 0)
 {//not in list
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piCoeff     = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
  pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
   pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
  }

  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, 
   if(sciErr.iErr)
   {
    printError(&sciErr, 0);
    return 0;
   }
  }
  else
  {
   sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblRea
   if(sciErr.iErr)
```

```
   {
    printError(&sciErr, 0);
    return 0;
   }
  }
 }
 else
 {
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
  pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
   pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
  }

  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Poly  (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

 for(i = 0 ; i < iRows * iCols ; i++)
 {
  free(pdblReal[i]);
  free(pdblImg[i]);
 }
 free(pdblReal);
 free(pdblImg);
 free(piCoeff);
```

```
 return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;

 int* piBool     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
 }
 else
 {
  sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean (%d x %d)\n", iRows, iCols);
 return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;
 int iItem           = 0;

 int* piNbRow        = NULL;
 int* piColPos       = NULL;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//Not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &p
  }
  else
  {
   sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
```

```
  {
   sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
  }
 }

 insert_indent();
 sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;
 int iItem       = 0;

 int* piNbRow    = NULL;
 int* piColPos   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &p
 }
 else
 {
  sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows,
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iPrec               = 0;
 int iRows               = 0;
 int iCols               = 0;

 char* pcData            = NULL;
 short* psData           = NULL;
 int* piData             = NULL;
 unsigned char* pucData  = NULL;
 unsigned short* pusData = NULL;
 unsigned int* puiData   = NULL;
```

```
if(_iItemPos == 0)
{//Not in list
 sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 switch(iPrec)
 {
 case SCI_INT8 :
  sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
  break;
 case SCI_INT16 :
  sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
  break;
 case SCI_INT32 :
  sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
  break;
 case SCI_UINT8 :
  sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &puc
  break;
 case SCI_UINT16 :
  sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pu
  break;
 case SCI_UINT32 :
  sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &pu
  break;
 default :
  return 1;
 }
}
else
{
 sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 switch(iPrec)
 {
 case SCI_INT8 :
  sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
  break;
 case SCI_INT16 :
  sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
  break;
 case SCI_INT32 :
  sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
  break;
 case SCI_UINT8 :
  sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &
  break;
 case SCI_UINT16 :
  sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &
```

```
   break;
  case SCI_UINT32 :
   sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, 
   break;
  default :
   return 1;
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 if(iPrec > 10)
 {
  sciprint("Unsigned ");
 }

 sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

 return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRows       = 0;
 int iCols       = 0;
 int* piLen      = NULL;

 char **pstData  = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }
```

```
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData)
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
 else
 {
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }

  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Strings (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 void* pvPtr     = NULL;

 if(_iItemPos == 0)
 {
```

```
  sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
 }
 else
 {
  sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Pointer : 0x%08X\n", pvPtr);

 return 0;
}
```

## Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2,-1,0,0,0;-1,2,-1,0,0;0,-1,2,-1,0;0,0,-1,2,-1;0,0,0,-1,2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with","you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

# Name

Polynomial writing (Scilab gateway) — How to add matrix of polynomial in a list.

Input argument profile:

```
SciErr createMatrixOfPolyInList(void* _pvCtx, int _iVar, int* _piParent, int _i
```

```
SciErr createComplexMatrixOfPolyInList(void* _pvCtx, int _iVar, int* _piParent,
```

Named variable profile:

```
SciErr createMatrixOfPolyInNamedList(void* _pvCtx, char* _pstName, int* _piPare
```

```
SciErr createComplexMatrixOfPolyInNamedList(void* _pvCtx, char* _pstName, int* 
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_pstVarName
: Variable name in polynomials (Scilab5: 4 characters max).

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_piNbCoef
: Number of coefficient for each polynomial.

_pdblReal
: Address of array of double* with real part of coefficient (size: _iCols * _iRows)

_pdblImg
: Address of array of double* with imaginary part of coefficient (size: _iCols * _iRows)

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to add matrix of polynomial in a list.

# Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int *piAddr             = NULL;
 int* piChild            = NULL;
 double pdblData1[]      = {1,3,5,2,4,6};
 double pdblData2[]      = {6,4,2,5,3,1};
 char *pstData[]         = {"may","be","the","with","puffin","you"};
 short psData[]          = {1,4,2,5,3,6};
 double pdblPoly1[]      = {1};
 double pdblPoly2[]      = {-2,-1};
 double pdblPoly3[]      = {1,2,3};
 double pdblPoly4[]      = {-4,-3,-2,-1};
 double pdblPoly5[]      = {1,2,3,4,5};
 double pdblPoly6[]      = {-6,-5,-4,-3,-2,-1};
 double *pdblPoly[]      = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPol
 int piCoef[]            = {1,3,5,2,4,6};
 int piNbItemRow[]       = {1,2,1};
 int piColPos[]          = {8,4,7,2};
 double pdblSReal[]      = {1,2,3,4};
 double pdblSImg[]       = {4,3,2,1};
 int piBool[]            = {1,0,1,0,1,0,1,0,1};
 double* pdblDataPtr     = NULL;


 sciErr = createList(pvApiCtx, 1, 8, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstDat
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psD
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }
```

```
sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

pdblDataPtr     = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0]  = 1;
pdblDataPtr[1]  = 2;
pdblDataPtr[2]  = 3;
pdblDataPtr[3]  = 4;

sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
if(sciErr.iErr)
```

```
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = 1;
 return 0;
}
```

## Scilab test script

```
size_ref    = 8;
type_ref    = ["constant","string","int16","polynomial", "sparse", "boolean", "
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

# Name

Sparse reading (Scilab gateway) — How to read sparse in a list.

Input argument profile:

```
SciErr getSparseMatrixInList(void* _pvCtx, int* _piParent, int _iItemPos, int* _
```

```
SciErr getComplexSparseMatrixInList(void* _pvCtx, int* _piParent, int _iItemPos
```

Named variable profile:

```
SciErr readSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int* _piParent
```

```
SciErr readComplexSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int* _p
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piParent
: Address of the parent of the new item.

_pstName
: Name of the variable for "named" functions.

_iItemPos
: Position of the new item in the list.

_piRows
: Return number of rows of the variable.

_piCols
: Return number of columns of the variable.

_piNbItem
: Return number of non zero value.

_piNbItemRow
: Return number of item in each rows (size: _iRows).

_piColPos
: Return column position for each item (size: _iNbItem).

_pdblReal
: Return array on real part data (size: _iNbItem).

_pdblImg
: Return array on imaginary part data (size: _iNbItem).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to read sparse in a list.

## Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
 int i = 0;
 for(i = 0 ; i < iTab ; i++)
 {
  sciprint("\t");
 }
}


int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int iItem       = 0;
 int iRet        = 0;
 int *piAddr     = NULL;

 CheckRhs(1,1);

 sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 get_info(1, NULL, piAddr, 0);

 LhsVar(1) = 0;
 return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRet    = 0;
 int iType   = 0;

 sciErr = getVarType(pvApiCtx, _piAddr, &iType);
 switch(iType)
```

```
 {
 case sci_matrix :
  iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_poly :
  iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean :
  iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_sparse :
  iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean_sparse :
  iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_ints :
  iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_strings :
  iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_list :
  insert_indent();
  sciprint("List ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_tlist :
  insert_indent();
  sciprint("TList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_mlist :
  insert_indent();
  sciprint("MList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_pointer :
  iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 default :
  insert_indent();
  sciprint("Unknow type\n");
  return 1;
 }
 return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRet        = 0;
 int iItem       = 0;
 int* piChild    = NULL;

 sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
 if(sciErr.iErr)
```

```
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciprint("(%d)\n", iItem);
 for(i = 0 ; i < iItem ; i++)
 {
  sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  iTab++;
  iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
  iTab--;
 }

 return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblRea
  }
  else
  {
   sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRo
  }
  else
  {
   sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
```

```
  return 0;
 }

 insert_indent();
 sciprint("Double (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iLen            = 0;
 int iRows           = 0;
 int iCols           = 0;

 char pstVar[16];
 int* piCoeff        = NULL;
 double** pdblReal   = NULL;
 double** pdblImg    = NULL;

 sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 if(_iItemPos == 0)
 {//not in list
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piCoeff     = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
  pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
   pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
  }

  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff,
   if(sciErr.iErr)
```

```
   {
    printError(&sciErr, 0);
    return 0;
   }
  }
  else
  {
   sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblRea
   if(sciErr.iErr)
   {
    printError(&sciErr, 0);
    return 0;
   }
  }
 }
 else
 {
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
  pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
   pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
  }

  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Poly  (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);
```

```
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  free(pdblReal[i]);
  free(pdblImg[i]);
 }
 free(pdblReal);
 free(pdblImg);
 free(piCoeff);
 return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;

 int* piBool     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
 }
 else
 {
  sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean (%d x %d)\n", iRows, iCols);
 return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;
 int iItem           = 0;

 int* piNbRow        = NULL;
 int* piColPos       = NULL;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//Not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &
```

```
  }
  else
  {
   sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
  }
 }

 insert_indent();
 sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;
 int iItem       = 0;

 int* piNbRow    = NULL;
 int* piColPos   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &p
 }
 else
 {
  sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows,
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iPrec               = 0;
 int iRows               = 0;
```

```
int iCols               = 0;

char* pcData            = NULL;
short* psData           = NULL;
int* piData             = NULL;
unsigned char* pucData  = NULL;
unsigned short* pusData = NULL;
unsigned int* puiData   = NULL;

if(_iItemPos == 0)
{//Not in list
 sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 switch(iPrec)
 {
 case SCI_INT8 :
  sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
  break;
 case SCI_INT16 :
  sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
  break;
 case SCI_INT32 :
  sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
  break;
 case SCI_UINT8 :
  sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &puc
  break;
 case SCI_UINT16 :
  sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pu
  break;
 case SCI_UINT32 :
  sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &pu
  break;
 default :
  return 1;
 }
}
else
{
 sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 switch(iPrec)
 {
 case SCI_INT8 :
  sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
  break;
 case SCI_INT16 :
  sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
```

```
   break;
  case SCI_INT32 :
   sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
   break;
  case SCI_UINT8 :
   sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &i
   break;
  case SCI_UINT16 :
   sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &
   break;
  case SCI_UINT32 :
   sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &
   break;
  default :
   return 1;
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 if(iPrec > 10)
 {
  sciprint("Unsigned ");
 }

 sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

 return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRows       = 0;
 int iCols       = 0;
 int* piLen      = NULL;

 char **pstData  = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
  if(sciErr.iErr)
  {
```

```
  printError(&sciErr, 0);
  return 0;
 }

 pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
 }

 sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData)
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }
}
else
{
 sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 piLen = (int*)malloc(sizeof(int) * iRows * iCols);
 sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
 }

 sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }
}

if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

insert_indent();
sciprint("Strings (%d x %d)\n", iRows, iCols);

return 0;;
```

```
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 void* pvPtr     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
 }
 else
 {
  sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Pointer : 0x%08X\n", pvPtr);

 return 0;
}
```

## Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2,-1,0,0,0;-1,2,-1,0,0;0,-1,2,-1,0;0,0,-1,2,-1;0,0,0,-1,2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with","you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

# Name

Sparse writing (Scilab gateway) — How to add sparse matrix in a list.

Create from existing data.

Input argument profile:

```
SciErr createSparseMatrixInList(void* _pvCtx, int _iVar, int* _piParent, int _i
```

```
SciErr createComplexSparseMatrixInList(void* _pvCtx, int _iVar, int* _piParent,
```

Named variable profile:

```
SciErr createSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int* _piPare
```

```
SciErr createComplexSparseMatrixInNamedList(void* _pvCtx, char* _pstName, int* 
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_iNbItem
: Number of non zero itmes in the sparse.

_piNbItemRow
: Number of item in each rows (size: _iRows).

_piColPos
: Column position for each item (size: _iNbItem).

_pdblReal
: Address of real data array (size: _iNbItem).

_pdblImg
: Address of imaginary data array (size: _iNbItem).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to add sparse matrix in a list.

# Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int *piAddr             = NULL;
 int* piChild            = NULL;
 double pdblData1[]      = {1,3,5,2,4,6};
 double pdblData2[]      = {6,4,2,5,3,1};
 char *pstData[]         = {"may","be","the","with","puffin","you"};
 short psData[]          = {1,4,2,5,3,6};
 double pdblPoly1[]      = {1};
 double pdblPoly2[]      = {-2,-1};
 double pdblPoly3[]      = {1,2,3};
 double pdblPoly4[]      = {-4,-3,-2,-1};
 double pdblPoly5[]      = {1,2,3,4,5};
 double pdblPoly6[]      = {-6,-5,-4,-3,-2,-1};
 double *pdblPoly[]      = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPol
 int piCoef[]            = {1,3,5,2,4,6};
 int piNbItemRow[]       = {1,2,1};
 int piColPos[]          = {8,4,7,2};
 double pdblSReal[]      = {1,2,3,4};
 double pdblSImg[]       = {4,3,2,1};
 int piBool[]            = {1,0,1,0,1,0,1,0,1};
 double* pdblDataPtr     = NULL;


 sciErr = createList(pvApiCtx, 1, 8, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstDat
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psD
 if(sciErr.iErr)
 {
```

```
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//add list in list
sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

pdblDataPtr     = (double*)malloc(sizeof(double) * 4);
pdblDataPtr[0]  = 1;
pdblDataPtr[1]  = 2;
pdblDataPtr[2]  = 3;
```

```
 pdblDataPtr[3]  = 4;

 sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = 1;
 return 0;
}
```

## Scilab test script

```
size_ref    = 8;
type_ref    = ["constant","string","int16","polynomial", "sparse", "boolean", "
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

# Name

String reading (Scilab gateway) — How to read matrix of string in a list.

Input argument profile:

```
SciErr getMatrixOfStringInList(void* _pvCtx, int* _piParent, int _iItemPos, int
```

Named variable profile:

```
SciErr readMatrixOfStringInNamedList(void* _pvCtx, char* _pstName, int* _piPare
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piParent
: Address of the parent of the new item.

_pstName
: Name of the variable for "named" functions.

_iItemPos
: Position of the new item in the list.

_piRows
: Return number of rows of the variable.

_piCols
: Return number of columns of the variable.

_piLength
: Address of array of strings length (must be allocated size: _piRows * _piCols)

_pstStrings
: Address of array of char* (must be allocated size: _piRows * _piCols)

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to read matrix of string in a list.

# Gateway Source

```
static int iTab = 0;

void insert_indent(void)
{
 int i = 0;
 for(i = 0 ; i < iTab ; i++)
 {
  sciprint("\t");
 }
}
```

```
int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);
int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos);

int common_read(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int iItem       = 0;
 int iRet        = 0;
 int *piAddr     = NULL;

 CheckRhs(1,1);

 sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 get_info(1, NULL, piAddr, 0);

 LhsVar(1) = 0;
 return 0;
}

int get_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRet    = 0;
 int iType   = 0;

 sciErr = getVarType(pvApiCtx, _piAddr, &iType);
 switch(iType)
 {
 case sci_matrix :
  iRet = get_double_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_poly :
  iRet = get_poly_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean :
  iRet = get_boolean_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_sparse :
  iRet = get_sparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_boolean_sparse :
  iRet = get_bsparse_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_ints :
```

```
  iRet = get_integer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_strings :
  iRet = get_string_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_list :
  insert_indent();
  sciprint("List ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_tlist :
  insert_indent();
  sciprint("TList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_mlist :
  insert_indent();
  sciprint("MList ");
  iRet = get_list_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 case sci_pointer :
  iRet = get_pointer_info(_iRhs, _piParent, _piAddr, _iItemPos);
  break;
 default :
  insert_indent();
  sciprint("Unknow type\n");
  return 1;
 }
 return iRet;
}

int get_list_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRet        = 0;
 int iItem       = 0;
 int* piChild    = NULL;

 sciErr = getListItemNumber(pvApiCtx, _piAddr, &iItem);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciprint("(%d)\n", iItem);
 for(i = 0 ; i < iItem ; i++)
 {
  sciErr = getListItemAddress(pvApiCtx, _piAddr, i + 1, &piChild);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  iTab++;
  iRet = get_info(_iRhs, _piAddr, piChild, i + 1);
```

```
  iTab--;
 }

 return 0;;
}

int get_double_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblRea
  }
  else
  {
   sciErr = getMatrixOfDouble(pvApiCtx, _piAddr, &iRows, &iCols, &pdblReal);
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRo
  }
  else
  {
   sciErr = getMatrixOfDoubleInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Double (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_poly_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iLen            = 0;
 int iRows           = 0;
 int iCols           = 0;
```

```
char pstVar[16];
int* piCoeff        = NULL;
double** pdblReal   = NULL;
double** pdblImg    = NULL;

sciErr = getPolyVariableName(pvApiCtx, _piAddr, pstVar, &iLen);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

if(_iItemPos == 0)
{//not in list
 sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 piCoeff     = (int*)malloc(sizeof(int) * iRows * iCols);
 sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, NULL);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
 pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
  pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
 }

 if(isVarComplex(pvApiCtx, _piAddr))
 {
  sciErr = getComplexMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, 
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
 else
 {
  sciErr = getMatrixOfPoly(pvApiCtx, _piAddr, &iRows, &iCols, piCoeff, pdblRea
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
}
else
{
```

```
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piCoeff = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCols
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
  pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pdblReal[i] = (double*)malloc(sizeof(double) * piCoeff[i]);
   pdblImg[i]  = (double*)malloc(sizeof(double) * piCoeff[i]);
  }

  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getMatrixOfPolyInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Poly  (%d x %d), varname : \'%s\'\n", iRows, iCols, pstVar);

 for(i = 0 ; i < iRows * iCols ; i++)
 {
  free(pdblReal[i]);
  free(pdblImg[i]);
 }
 free(pdblReal);
 free(pdblImg);
 free(piCoeff);
 return 0;;
}

int get_boolean_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;
```

```
 int* piBool     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getMatrixOfBoolean(pvApiCtx, _piAddr, &iRows, &iCols, &piBool);
 }
 else
 {
  sciErr = getMatrixOfBooleanInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iC
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean (%d x %d)\n", iRows, iCols);
 return 0;
}

int get_sparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows           = 0;
 int iCols           = 0;
 int iItem           = 0;

 int* piNbRow        = NULL;
 int* piColPos       = NULL;

 double* pdblReal    = NULL;
 double* pdblImg     = NULL;

 if(_iItemPos == 0)
 {//Not in list
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &p
  }
  else
  {
   sciErr = getSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &piNbRow
  }
 }
 else
 {
  if(isVarComplex(pvApiCtx, _piAddr))
  {
   sciErr = getComplexSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows
  }
  else
  {
   sciErr = getSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCol
  }
 }
```

```
 insert_indent();
 sciprint("Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_bsparse_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iRows       = 0;
 int iCols       = 0;
 int iItem       = 0;

 int* piNbRow    = NULL;
 int* piColPos   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getBooleanSparseMatrix(pvApiCtx, _piAddr, &iRows, &iCols, &iItem, &p
 }
 else
 {
  sciErr = getBooleanSparseMatrixInList(pvApiCtx, _piParent, _iItemPos, &iRows,
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Boolean Sparse (%d x %d), Item(s) : %d \n", iRows, iCols, iItem);
 return 0;;
}

int get_integer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int iPrec               = 0;
 int iRows               = 0;
 int iCols               = 0;

 char* pcData            = NULL;
 short* psData           = NULL;
 int* piData             = NULL;
 unsigned char* pucData  = NULL;
 unsigned short* pusData = NULL;
 unsigned int* puiData   = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
```

```
 switch(iPrec)
 {
 case SCI_INT8 :
  sciErr = getMatrixOfInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &pcData);
  break;
 case SCI_INT16 :
  sciErr = getMatrixOfInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &psData);
  break;
 case SCI_INT32 :
  sciErr = getMatrixOfInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &piData);
  break;
 case SCI_UINT8 :
  sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, _piAddr, &iRows, &iCols, &puc
  break;
 case SCI_UINT16 :
  sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, _piAddr, &iRows, &iCols, &pu
  break;
 case SCI_UINT32 :
  sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, _piAddr, &iRows, &iCols, &pu
  break;
 default :
  return 1;
 }
}
else
{
 sciErr = getMatrixOfIntegerPrecision(pvApiCtx, _piAddr, &iPrec);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 switch(iPrec)
 {
 case SCI_INT8 :
  sciErr = getMatrixOfInteger8InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
  break;
 case SCI_INT16 :
  sciErr = getMatrixOfInteger16InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
  break;
 case SCI_INT32 :
  sciErr = getMatrixOfInteger32InList(pvApiCtx, _piParent, _iItemPos, &iRows, &
  break;
 case SCI_UINT8 :
  sciErr = getMatrixOfUnsignedInteger8InList(pvApiCtx, _piParent, _iItemPos, &
  break;
 case SCI_UINT16 :
  sciErr = getMatrixOfUnsignedInteger16InList(pvApiCtx, _piParent, _iItemPos, &
  break;
 case SCI_UINT32 :
  sciErr = getMatrixOfUnsignedInteger32InList(pvApiCtx, _piParent, _iItemPos, &
  break;
 default :
  return 1;
 }
}
```

```
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 if(iPrec > 10)
 {
  sciprint("Unsigned ");
 }

 sciprint("Integer %d bits (%d x %d)\n", (iPrec % 10) * 8, iRows, iCols);

 return 0;;
}

int get_string_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 int i;
 int iRows       = 0;
 int iCols       = 0;
 int* piLen      = NULL;

 char **pstData  = NULL;

 if(_iItemPos == 0)
 {//Not in list
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, NULL, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, NULL);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }

  sciErr = getMatrixOfString(pvApiCtx, _piAddr, &iRows, &iCols, piLen, pstData)
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }
```

```
 else
 {
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  piLen = (int*)malloc(sizeof(int) * iRows * iCols);
  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null ter
  }

  sciErr = getMatrixOfStringInList(pvApiCtx, _piParent, _iItemPos, &iRows, &iCo
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Strings (%d x %d)\n", iRows, iCols);

 return 0;;
}

int get_pointer_info(int _iRhs, int* _piParent, int *_piAddr, int _iItemPos)
{
 SciErr sciErr;
 void* pvPtr     = NULL;

 if(_iItemPos == 0)
 {
  sciErr = getPointer(pvApiCtx, _piAddr, &pvPtr);
 }
 else
 {
  sciErr = getPointerInList(pvApiCtx, _piParent, _iItemPos, &pvPtr);
 }

 if(sciErr.iErr)
```

```
 {
  printError(&sciErr, 0);
  return 0;
 }

 insert_indent();
 sciprint("Pointer : 0x%08X\n", pvPtr);

 return 0;
}
```

## Scilab test script

```
function read_all()
d = [1,2,3;4,5,6;7,8,9];common_read(d);
s=poly(0,"x");p=1+s+2*s^2;p = [(p * 2),(p * s + 3);(p * 2 * s ** 2 - 6),(12 - 4
b = [%t,%f;%t,%f;%f,%t];common_read(b);
sp=sparse([2,-1,0,0,0;-1,2,-1,0,0;0,-1,2,-1,0;0,0,-1,2,-1;0,0,0,-1,2]);common_r
bsp=sparse([1,2;4,5;3,10],[%t,%t,%t]);common_read(bsp);
i8 = int8([1,2,3]);common_read(i8);
ui32 = uint32([3;2;1]);common_read(ui32);
str = ["may", "the", "puffin"; "be", "with","you"];common_read(str);
if with_module('umfpack') then
    Cp = taucs_chfact(sp);
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str), Cp);
else
    l = list(list(d, p, list(b, sp)), list(i8, bsp), list(ui32, str));
end
common_read(l)
endfunction
read_all;
```

# Name

String writing (Scilab gateway) — How to add matrix of string in a list.

Input argument profile:

```
SciErr createMatrixOfStringInList(void* _pvCtx, int _iVar, int* _piParent, int _
```

Named variable profile:

```
SciErr createMatrixOfStringInNamedList(void* _pvCtx, char* _pstName, int* _piPa
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_piParent
: Address of the parent of the new item.

_iItemPos
: Position of the new item in the list.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_pstStrings
: Address of array of char* (size: _iCols * _iRows).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to add matrix of string in a list.

# Gateway Source

```
int list_createlist(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int *piAddr             = NULL;
 int* piChild            = NULL;
 double pdblData1[]      = {1,3,5,2,4,6};
 double pdblData2[]      = {6,4,2,5,3,1};
 char *pstData[]         = {"may","be","the","with","puffin","you"};
 short psData[]          = {1,4,2,5,3,6};
 double pdblPoly1[]      = {1};
 double pdblPoly2[]      = {-2,-1};
 double pdblPoly3[]      = {1,2,3};
```

```
double pdblPoly4[]      = {-4,-3,-2,-1};
double pdblPoly5[]      = {1,2,3,4,5};
double pdblPoly6[]      = {-6,-5,-4,-3,-2,-1};
double *pdblPoly[]      = {pdblPoly1, pdblPoly3, pdblPoly5, pdblPoly2, pdblPol
int piCoef[]            = {1,3,5,2,4,6};
int piNbItemRow[]       = {1,2,1};
int piColPos[]          = {8,4,7,2};
double pdblSReal[]      = {1,2,3,4};
double pdblSImg[]       = {4,3,2,1};
int piBool[]            = {1,0,1,0,1,0,1,0,1};
double* pdblDataPtr     = NULL;


sciErr = createList(pvApiCtx, 1, 8, &piAddr);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piAddr, 1, 3, 2,
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfStringInList(pvApiCtx, Rhs + 1, piAddr, 2, 2, 3, pstDat
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfInteger16InList(pvApiCtx, Rhs + 1, piAddr, 3, 2, 3, psD
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfPolyInList(pvApiCtx, Rhs + 1, piAddr, 4, "x", 3, 2, piC
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createComplexSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 5, 3, 10, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = createMatrixOfBooleanInList(pvApiCtx, Rhs + 1, piAddr, 6, 3, 3, piBoo
if(sciErr.iErr)
{
```

```
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createBooleanSparseMatrixInList(pvApiCtx, Rhs + 1, piAddr, 7, 3, 10, 
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 //add list in list
 sciErr = createListInList(pvApiCtx, Rhs + 1, piAddr, 8, 3, &piChild);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createMatrixOfDoubleInList(pvApiCtx, Rhs + 1, piChild, 1, 3, 2, pdblD
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciErr = createSparseMatrixInList(pvApiCtx, Rhs + 1, piChild, 2, 3, 10, 4, piN
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 pdblDataPtr     = (double*)malloc(sizeof(double) * 4);
 pdblDataPtr[0]  = 1;
 pdblDataPtr[1]  = 2;
 pdblDataPtr[2]  = 3;
 pdblDataPtr[3]  = 4;

 sciErr = createPointerInList(pvApiCtx, Rhs + 1, piChild, 3, pdblDataPtr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = 1;
 return 0;
}
```

## Scilab test script

```
size_ref    = 8;
type_ref    = ["constant","string","int16","polynomial", "sparse", "boolean", "
```

```
dim_ref     = list([3,2],[2,3],[2,3],[3,2],[3,10],[3,3],[3,10],3);

l = list_createlist();
if size(l) <> size_ref then error("failed"), end
for i = 1 : size_ref
    if typeof(l(i)) <> type_ref(i) then error("failed"), end
    if size(l(i)) <> dim_ref(i) then error("failed"), end
end
```

# Name

Boolean reading (Scilab gateway) — How to read matrix of boolean.

Input argument profile:

```
SciErr getMatrixOfBoolean(void* _pvCtx, int* _piAddress, int* _piRows, int* _pi
```

Named variable profile:

```
SciErr readNamedMatrixOfBoolean(void* _pvCtx, char* _pstName, int* _piRows, int
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piAddress
: Address of the Scilab variable.

_pstName
: Name of the variable for "named" functions.

_piRows
: Return number of rows of the variable.

_piCols
: Return number of columns of the variable.

_piBool
: Return address of data array (size: _iRows * _iCols).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to read matrix of boolean.

# Gateway Source

```
int read_write_boolean(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    //first variable info : real matrix of double
    int iRows     = 0;
    int iCols     = 0;
    int *piAddr    = NULL;
    int* piBool    = NULL;

    //check input and output arguments
    CheckRhs(1,1);
    CheckLhs(1,1);
```

```
    //get variable address of the first input argument
    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    //get size and data from Scilab memory
    sciErr = getMatrixOfBoolean(pvApiCtx, piAddr, &iRows, &iCols, &piBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    //Do something with data
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        piBool[i] = piBool[i] == 0 ? 1 : 0;
    }

    sciErr = createMatrixOfBoolean(pvApiCtx, Rhs + 1, iRows, iCols, piBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    LhsVar(1) = Rhs + 1;
    return 0;
}
```

# Scilab test script

```
a = [%t, %f, %t ; %f, %t, %f ; %t, %f, %t];
a_ref = [%f, %t, %f ; %t, %f, %t ; %f, %t, %f];
b = read_write_boolean(a);

if or(b <> a_ref) then error("failed"), end
```

# Name

Boolean writing (Scilab gateway) — How to write matrices of boolean.

Input argument profile:

```
SciErr createMatrixOfBoolean(void* _pvCtx, int _iVar, int _iRows, int _iCols, i
```

Named variable profile:

```
SciErr createNamedMatrixOfBoolean(void* _pvCtx, char* _pstName, int _iRows, int
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_piRows
: Return number of rows of the variable.

_piCols
: Return number of columns of the variable.

_piBool
: Return address of data array (size: _iRows * _iCols).

SciErr
: Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocMatrixOfBoolean(void* _pvCtx, int _iVar, int _iRows, int _iCols, in
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_iRows
: Number of rows of the new variable.

_iCols
: Numbers of columns of the new variable.

_piBool
: Returns address of real data array (size: _iCols * _iRows).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to write matrix of boolean.

# Gateway Source

```
int read_write_boolean(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    //first variable info : real matrix of double
    int iRows     = 0;
    int iCols     = 0;
    int *piAddr    = NULL;
    int* piBool    = NULL;

    //check input and output arguments
    CheckRhs(1,1);
    CheckLhs(1,1);


    //get variable address of the first input argument
    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    //get size and data from Scilab memory
    sciErr = getMatrixOfBoolean(pvApiCtx, piAddr, &iRows, &iCols, &piBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    //Do something with data
    for(i = 0 ; i < iRows * iCols ; i++)
    {
        piBool[i] = piBool[i] == 0 ? 1 : 0;
    }

    sciErr = createMatrixOfBoolean(pvApiCtx, Rhs + 1, iRows, iCols, piBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    LhsVar(1) = Rhs + 1;
    return 0;
```

```
}
```

## Scilab test script

```
a = [%t, %f, %t ; %f, %t, %f ; %t, %f, %t];
a_ref = [%f, %t, %f ; %t, %f, %t ; %f, %t, %f];
b = read_write_boolean(a);

if or(b <> a_ref) then error("failed"), end
```

# Name

Boolean sparse reading (Scilab gateway) — How to read boolean sparse in a gateway.

Input argument profile:

```
SciErr getBooleanSparseMatrix(void* _pvCtx, int* _piAddress, int* _piRows, int*
```

Named variable profile:

```
SciErr readNamedBooleanSparseMatrix(void* _pvCtx, char* _pstName, int* _piRows,
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_piAddress
: Address of the Scilab variable.

_pstName
: Name of the variable for "named" functions.

_piRows
: Return number of rows of the variable.

_piCols
: Return number of columns of the variable.

_piNbItem
: Return number of non zero value.

_piNbItemRow
: Return number of item in each rows (size: _iRows).

_piColPos
: Return column position for each item (size: _iNbItem).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to read boolean sparse.

# Gateway Source

```
int read_write_bsparse(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i                   = 0;
    int j                   = 0;
    int k                   = 0;

    //first variable info : real matrix of double
    int iRows               = 0;
    int iCols               = 0;
    int *piAddr             = NULL;
    int iNbItem             = 0;
```

```
    int* piNbItemRow        = NULL;
    int* piColPos           = NULL;

    int iCol                = 0;
    int iNewCol             = 0;

    int iNewItem            = 0;
    int* piNewRow           = NULL;
    int* piNewCol           = NULL;


    //check input and output arguments
    CheckRhs(1,1);
    CheckLhs(1,1);


    //get variable address of the first input argument
    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    //get size and data from Scilab memory
    sciErr = getBooleanSparseMatrix(pvApiCtx, piAddr, &iRows, &iCols, &iNbItem,
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    //Do something with data
    //convert %T -> %F and %F -> %T

    iNewItem = (iRows * iCols) - iNbItem;
    piNewRow = (int*)MALLOC(sizeof(int) * iRows);
    piNewCol = (int*)MALLOC(sizeof(int) * iNewItem);

    for(i = 0 ; i < iRows ; i++)
    {
        piNewRow[i] = iCols - piNbItemRow[i];
        for(j = 0 ; j < iCols ; j++)
        {
            int iFind = 0;
            for(k = 0 ; k < piNbItemRow[i] ; k++)
            {
                if(piColPos[iCol + k] == (j + 1))
                {
                    iFind = 1;
                    break;
                }
            }
            if(iFind == 0)
            {
                piNewCol[iNewCol++] = (j + 1);
            }
        }
```

```
        iCol += piNbItemRow[i];
    }

    sciErr = createBooleanSparseMatrix(pvApiCtx, Rhs + 1, iRows, iCols, iNewIte
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    LhsVar(1) = Rhs + 1;
    return 0;
}
```

# Scilab test script

```
a = sparse([%t, %f, %t ; %f, %t, %f ; %t, %f, %t]);
a_ref = sparse([%f, %t, %f ; %t, %f, %t ; %f, %t, %f]);

b = read_write_bsparse(a);
if or(b <> a_ref) then error("failed");end
```

# Name

Boolean sparse writing (Scilab gateway) — How to add boolean sparse matrix in a gateway.

Input argument profile:

```
SciErr createBooleanSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCols
```

Named variable profile:

```
SciErr createNamedBooleanSparseMatrix(void* _pvCtx, char* _pstName, int _iRows,
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_iNbItem
: Number of non zero itmes in the sparse.

_piNbItemRow
: Number of item in each rows (size: _iRows).

_piColPos
: Column position for each item (size: _iNbItem).

SciErr
: Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocBooleanSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCols
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_iNbItem
: Number of non zero itmes in the sparse.

_piNbItemRow
: Number of item in each rows (size: _iRows).

_piColPos
: Column position for each item (size: _iNbItem).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to add boolean sparse matrix in a list.

# Gateway Source

```
int read_write_bsparse(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i                   = 0;
    int j                   = 0;
    int k                   = 0;

    //first variable info : real matrix of double
    int iRows               = 0;
    int iCols               = 0;
    int *piAddr             = NULL;
    int iNbItem             = 0;
    int* piNbItemRow        = NULL;
    int* piColPos           = NULL;

    int iCol                = 0;
    int iNewCol             = 0;

    int iNewItem            = 0;
    int* piNewRow           = NULL;
    int* piNewCol           = NULL;

    //check input and output arguments
    CheckRhs(1,1);
    CheckLhs(1,1);


    //get variable address of the first input argument
    sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    //get size and data from Scilab memory
    sciErr = getBooleanSparseMatrix(pvApiCtx, piAddr, &iRows, &iCols, &iNbItem,
    if(sciErr.iErr)
```

```
    {
        printError(&sciErr, 0);
        return 0;
    }

    //Do something with data
    //convert %T -> %F and %F -> %T

    iNewItem = (iRows * iCols) - iNbItem;
    piNewRow = (int*)MALLOC(sizeof(int) * iRows);
    piNewCol = (int*)MALLOC(sizeof(int) * iNewItem);

    for(i = 0 ; i < iRows ; i++)
    {
        piNewRow[i] = iCols - piNbItemRow[i];
        for(j = 0 ; j < iCols ; j++)
        {
            int iFind = 0;
            for(k = 0 ; k < piNbItemRow[i] ; k++)
            {
                if(piColPos[iCol + k] == (j + 1))
                {
                    iFind = 1;
                    break;
                }
            }
            if(iFind == 0)
            {
                piNewCol[iNewCol++] = (j + 1);
            }
        }
        iCol += piNbItemRow[i];
    }

    sciErr = createBooleanSparseMatrix(pvApiCtx, Rhs + 1, iRows, iCols, iNewIte
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    LhsVar(1) = Rhs + 1;
    return 0;
}
```

# Scilab test script

```
a = sparse([%t, %f, %t ; %f, %t, %f ; %t, %f, %t]);
a_ref = sparse([%f, %t, %f ; %t, %f, %t ; %f, %t, %f]);

b = read_write_bsparse(a);
if or(b <> a_ref) then error("failed");end
```

# Name

Variable Reference (Scilab gateway) — How to get the address of an argument or a variable in a gateway.

Input argument profile:

```
SciErr getVarAddressFromPosition(void* _pvCtx, int _iVar, int** _piAddress)
```

Named variable profile:

```
SciErr getVarAddressFromName(void* _pvCtx, char* _pstName, int** _piAddress)
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_iVar
: Position of the argument in the function call.

_pstName
: Scilab variable name.

_piAddress
: Return address of the Scilab variable.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This function retrieves the address of an argument in a gateway.

# Gateway Source

```
SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }

    //1 for true, 0 for false
    iBool = sciErr.iErr == 0 ? 1 : 0;
```

```
    sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
    //assign allocated variables to Lhs position
    LhsVar(1) = 1;

    return 0;
}

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr     = NULL;
    int iType       = 0;
    int iRows       = 0;
    int iCols       = 0;
    int iItem       = 0;
    int iComplex    = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :
        {
            char pstSigned[]    = "signed";
            char pstUnsigned[]  = "unsigned";
```

```
            char* pstSign       = pstSigned;

            int iPrec           = 0;
            sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
            if(sciErr.iErr)
            {
                return sciErr;
            }

            if(iPrec > 10)
            {
                pstSign = pstUnsigned;
            }

            sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
        }
        break;
        case sci_strings :
            sciprint("strings\n");
            break;
        case sci_list :
            sciprint("list\n");
            break;
        case sci_tlist :
            sciprint("tlist\n");
            break;
        case sci_mlist :
            sciprint("mlist\n");
            break;
        default :
            sciprint("Not manage by this function\n");
            return sciErr;
    }

    if(isVarComplex(pvApiCtx, piAddr))
    {
        sciprint("\tComplex: Yes\n");
    }

    sciprint("\tDimensions: ");
    if(isVarMatrixType(pvApiCtx, piAddr))
    {
        sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
        if(sciErr.iErr)
        {
            return sciErr;
        }

        sciprint("%d x %d", iRows, iCols);
    }
    else
    {
        sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
        if(sciErr.iErr)
        {
            return sciErr;
        }
        sciprint("%d", iItem);
```

```
    }
    return sciErr;
}
```

## Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];
l2 = ["may","the";"puffin","be";"with","you"];
l3 = int8([1,2,3]);
l4 = uint16([1000,2000,3000]);
l5 = list(l1,l2,l3);
l = list(l1,l2,l3,l4,l5);
common_function(l(1:$))
```

# Name

Variable dimension (Scilab gateway) — How to get the dimensions of an argument or a variable stored as matrix.

Input argument profile:

```
SciErr getVarDimension(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCols
```

Named variable profile:

```
SciErr getNamedVarDimension(void* _pvCtx, char *_pstName, int* _piRows, int* _p
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_piAddress
: The address of the variable.

_pstName
: Scilab variable name.

_piRows
: Return number of rows.

_piCols
: Return number of columns.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to get the dimensions of a variable in a gateway.

# Gateway Source

```
SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }
```

```
    //1 for true, 0 for false
    iBool = sciErr.iErr == 0 ? 1 : 0;
    sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
    //assign allocated variables to Lhs position
    LhsVar(1) = 1;

    return 0;
}

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr     = NULL;
    int iType       = 0;
    int iRows       = 0;
    int iCols       = 0;
    int iItem       = 0;
    int iComplex    = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :
```

```
        {
            char pstSigned[]    = "signed";
            char pstUnsigned[]  = "unsigned";
            char* pstSign       = pstSigned;

            int iPrec           = 0;
            sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
            if(sciErr.iErr)
            {
                return sciErr;
            }

            if(iPrec > 10)
            {
                pstSign = pstUnsigned;
            }

            sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
        }
        break;
        case sci_strings :
            sciprint("strings\n");
            break;
        case sci_list :
            sciprint("list\n");
            break;
        case sci_tlist :
            sciprint("tlist\n");
            break;
        case sci_mlist :
            sciprint("mlist\n");
            break;
        default :
            sciprint("Not manage by this function\n");
            return sciErr;
    }

    if(isVarComplex(pvApiCtx, piAddr))
    {
        sciprint("\tComplex: Yes\n");
    }

    sciprint("\tDimensions: ");
    if(isVarMatrixType(pvApiCtx, piAddr))
    {
        sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
        if(sciErr.iErr)
        {
            return sciErr;
        }

        sciprint("%d x %d", iRows, iCols);
    }
    else
    {
        sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
        if(sciErr.iErr)
        {
```

```
            return sciErr;
        }
        sciprint("%d", iItem);
    }
    return sciErr;
}
```

# Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];
l2 = ["may","the";"puffin","be";"with","you"];
l3 = int8([1,2,3]);
l4 = uint16([1000,2000,3000]);
l5 = list(l1,l2,l3);
l = list(l1,l2,l3,l4,l5);
common_function(l(1:$))
```

# Name

Variable Type (Scilab gateway) — How to get the type of an argument or a variable within a gateway.

Input argument profile:

```
SciErr getVarType(void* _pvCtx, int* _piAddress, int* _piType)
```

Named variable profile:

```
SciErr getNamedVarType(void* _pvCtx, char* _pstName, int* _piType)
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_piAddress
: The address of the variable

_pstName
: Scilab variable name.

_piType
: Scilab type of the variable (sci_matrix, sci_strings, sci_ints, ...).

  Note that the list of the different variable types is available as an enum in stack-c.h.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to retrieve the Scilab type of a variable in a gateway.

# Gateway Source

```


SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }

    //1 for true, 0 for false
```

```
    iBool = sciErr.iErr == 0 ? 1 : 0;
    sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }
    //assign allocated variables to Lhs position
    LhsVar(1) = 1;

    return 0;
}

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr     = NULL;
    int iType       = 0;
    int iRows       = 0;
    int iCols       = 0;
    int iItem       = 0;
    int iComplex    = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :
        {
            char pstSigned[]    = "signed";
```

```
            char pstUnsigned[]  = "unsigned";
            char* pstSign       = pstSigned;

            int iPrec           = 0;
            sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
            if(sciErr.iErr)
            {
                return sciErr;
            }

            if(iPrec > 10)
            {
                pstSign = pstUnsigned;
            }

            sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
        }
        break;
        case sci_strings :
            sciprint("strings\n");
            break;
        case sci_list :
            sciprint("list\n");
            break;
        case sci_tlist :
            sciprint("tlist\n");
            break;
        case sci_mlist :
            sciprint("mlist\n");
            break;
        default :
            sciprint("Not manage by this function\n");
            return sciErr;
    }

    if(isVarComplex(pvApiCtx, piAddr))
    {
        sciprint("\tComplex: Yes\n");
    }

    sciprint("\tDimensions: ");
    if(isVarMatrixType(pvApiCtx, piAddr))
    {
        sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
        if(sciErr.iErr)
        {
            return sciErr;
        }

        sciprint("%d x %d", iRows, iCols);
    }
    else
    {
        sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
        if(sciErr.iErr)
        {
            return sciErr;
        }
```

```
        sciprint("%d", iItem);
    }
    return sciErr;
}
```

# Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];
l2 = ["may","the";"puffin","be";"with","you"];
l3 = int8([1,2,3]);
l4 = uint16([1000,2000,3000]);
l5 = list(l1,l2,l3);
l = list(l1,l2,l3,l4,l5);
common_function(l(1:$))
```

# Name

Variable Complexity (Scilab gateway) — How to get the argument or variable complexity.

Input argument profile:

```
int isVarComplex(void* _pvCtx, int* _piAddress)
```

Named variable profile:

```
int isNamedVarComplex(void* _pvCtx, char *_pstName)
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_piAddress
: Address of the variable

_pstName
: Scilab variable name.

Returned value
: 0 for real variables and 1 for complex variables.

# Description

This help describes how to retrieve the variable complexity.

# Gateway Source

```
SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }

    //1 for true, 0 for false
    iBool = sciErr.iErr == 0 ? 1 : 0;
    sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
    if(sciErr.iErr)
    {
```

```
        printError(&sciErr, 0);
        return 0;
    }
    //assign allocated variables to Lhs position
    LhsVar(1) = 1;

    return 0;
}

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr     = NULL;
    int iType       = 0;
    int iRows       = 0;
    int iCols       = 0;
    int iItem       = 0;
    int iComplex    = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :
        {
            char pstSigned[]    = "signed";
            char pstUnsigned[]  = "unsigned";
            char* pstSign       = pstSigned;

            int iPrec           = 0;
```

```
            sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
            if(sciErr.iErr)
            {
                return sciErr;
            }

            if(iPrec > 10)
            {
                pstSign = pstUnsigned;
            }

            sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
        }
        break;
        case sci_strings :
            sciprint("strings\n");
            break;
        case sci_list :
            sciprint("list\n");
            break;
        case sci_tlist :
            sciprint("tlist\n");
            break;
        case sci_mlist :
            sciprint("mlist\n");
            break;
        default :
            sciprint("Not manage by this function\n");
            return sciErr;
    }

    if(isVarComplex(pvApiCtx, piAddr))
    {
        sciprint("\tComplex: Yes\n");
    }

    sciprint("\tDimensions: ");
    if(isVarMatrixType(pvApiCtx, piAddr))
    {
        sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
        if(sciErr.iErr)
        {
            return sciErr;
        }

        sciprint("%d x %d", iRows, iCols);
    }
    else
    {
        sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
        if(sciErr.iErr)
        {
            return sciErr;
        }
        sciprint("%d", iItem);
    }
    return sciErr;
}
```

# Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];
l2 = ["may","the";"puffin","be";"with","you"];
l3 = int8([1,2,3]);
l4 = uint16([1000,2000,3000]);
l5 = list(l1,l2,l3);
l = list(l1,l2,l3,l4,l5);
common_function(l(1:$))
```

# Name

Matrix Type (Scilab gateway) — How to know if an argument or a variable is stored as a matrix.

Input argument profile:

```
int isVarMatrixType(void* _pvCtx, int* _piAddress)
```

Named variable profile:

```
int isNamedVarMatrixType(void* _pvCtx, char *_pstName)
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_piAddress
: Address of the variable

Returned value
: 1 if the variable is stored as matrix otherwise 0.

# Description

This help describes how to know if a variable is stored as a matrix.

In some cases (exemple: list), a variable is not stored as a standard Scilab matrix. This function provides a way to handle both cases.

# Gateway Source

```
SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }

    //1 for true, 0 for false
    iBool = sciErr.iErr == 0 ? 1 : 0;
    sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
    if(sciErr.iErr)
    {
```

```
        printError(&sciErr, 0);
        return 0;
    }
    //assign allocated variables to Lhs position
    LhsVar(1) = 1;

    return 0;
}

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr     = NULL;
    int iType       = 0;
    int iRows       = 0;
    int iCols       = 0;
    int iItem       = 0;
    int iComplex    = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :
        {
            char pstSigned[]    = "signed";
            char pstUnsigned[]  = "unsigned";
            char* pstSign       = pstSigned;

            int iPrec           = 0;
```

```
            sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
            if(sciErr.iErr)
            {
                return sciErr;
            }

            if(iPrec > 10)
            {
                pstSign = pstUnsigned;
            }

            sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
        }
        break;
        case sci_strings :
            sciprint("strings\n");
            break;
        case sci_list :
            sciprint("list\n");
            break;
        case sci_tlist :
            sciprint("tlist\n");
            break;
        case sci_mlist :
            sciprint("mlist\n");
            break;
        default :
            sciprint("Not manage by this function\n");
            return sciErr;
    }

    if(isVarComplex(pvApiCtx, piAddr))
    {
        sciprint("\tComplex: Yes\n");
    }

    sciprint("\tDimensions: ");
    if(isVarMatrixType(pvApiCtx, piAddr))
    {
        sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
        if(sciErr.iErr)
        {
            return sciErr;
        }

        sciprint("%d x %d", iRows, iCols);
    }
    else
    {
        sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
        if(sciErr.iErr)
        {
            return sciErr;
        }
        sciprint("%d", iItem);
    }
    return sciErr;
}
```

## Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];
l2 = ["may","the";"puffin","be";"with","you"];
l3 = int8([1,2,3]);
l4 = uint16([1000,2000,3000]);
l5 = list(l1,l2,l3);
l = list(l1,l2,l3,l4,l5);
common_function(l(1:$))
```

# Name

Double reading (Scilab gateway) — How to read matrices of double in a gateway.

Input argument profile:

```
SciErr getMatrixOfDouble(void* _pvCtx, int* _piAddress, int* _piRows, int* _piC
SciErr getComplexMatrixOfDouble(void* _pvCtx, int* _piAddress, int* _piRows, in
```

Named variable profile:

```
SciErr readNamedMatrixOfDouble(void* _pvCtx, char* _pstName, int* _piRows, int*
SciErr readNamedComplexMatrixOfDouble(void* _pvCtx, char* _pstName, int* _piRow
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_piAddress
: Address of the Scilab variable.

_pstName
: Name of the variable for "named" functions.

_piRows
: Return number of rows.

_piCols
: Return number of columns.

_pdblReal
: Return address of real data array (size: _iCols * _iRows).

_pdblImg
: Return address of imaginary data array (size: _iCols * _iRows).

  This argument does not exist with getMatrixOfDouble and readNamedMatrixOfDouble.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how matrix of doubles can be handled through the Scilab API.

# Gateway Source

```
int read_double(char *fname,unsigned long fname_len)
{
 int i;
 //first variable info : real matrix of double
 int iType   = 0;
 int iRows   = 0;
 int iCols   = 0;
 int iComplex  = 0;
 int *piAddr   = NULL;
 double* pdblReal = NULL;
```

```
double* pdblImg  = NULL;

SciErr sciErr;
//check input and output arguments
CheckRhs(1,1);
CheckLhs(1,1);


/************************
*    First variable    *
************************/

//get variable address of the first input argument
sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//check type
sciErr = getVarType(pvApiCtx, piAddr, &iType);
if(sciErr.iErr || iType != sci_matrix)
{
 printError(&sciErr, 0);
 return 0;
}

//get complexity
iComplex = isVarComplex(pvApiCtx, piAddr);

//check complexity
if(iComplex)
{
 //get size and data from Scilab memory
 sciErr = getComplexMatrixOfDouble(pvApiCtx, piAddr, &iRows, &iCols, &pdblReal
}
else
{
 //get size and data from Scilab memory
 sciErr = getMatrixOfDouble(pvApiCtx, piAddr, &iRows, &iCols, &pdblReal);
}

if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//Do something with data
//if variable is complex, switch real part and imaginary part otherwise multip

if(iComplex)
{
 sciErr = createComplexMatrixOfDouble(pvApiCtx, Rhs + 1, iRows, iCols, pdblImg
}
else
{
```

```
  for(i = 0 ; i < iRows * iCols ; i++)
  {
   pdblReal[i] = pdblReal[i] * -1;
  }
  sciErr = createMatrixOfDouble(pvApiCtx, Rhs + 1, iRows, iCols, pdblReal);
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = Rhs + 1;
 return 0;
}
```

## Scilab test script

```
a = [   0 1 2 3; ..
        4 5 6 7; ..
        8 9 10 11];
b = [   23*%i,      1+22*%i,    2+21*%i,    3+20*%i,    4+19*%i,    5+18*%i; ..
        6+17*%i,    7+16*%i,    8+15*%i,    9+14*%i,     10+13*%i,  11+12*%i; .
        12+11*%i,   13+10*%i,   14+9*%i,    15+8*%i,    16+7*%i,    17+6*%i; ..
        18+5*%i,    19+4*%i,    20+3*%i,    21+2*%i,    22+1*%i,    23];

a2 = read_double(a);
b2 = read_double(b);

if or(a2 <> a * -1) then error("failed"), end
if or(b2 <> (imag(b) + real(b) * %i)) then error("failed"), end
```

# Name

Double writing (Scilab gateway) — How to write matrices of doubles in a gateway.

Create from existing data.

Input argument profile:

```
SciErr createMatrixOfDouble(void* _pvCtx, int _iVar, int _iRows, int _iCols, do
```

```
SciErr createComplexMatrixOfDouble(void* _pvCtx, int _iVar, int _iRows, int _iC
```

Named variable profile:

```
SciErr createNamedMatrixOfDouble(void* _pvCtx, char* _pstName, int _iRows, int
```

```
SciErr createNamedComplexMatrixOfDouble(void* _pvCtx, char* _pstName, int _iRow
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_iRows
: Number of rows of the new variable.

_iCols
: Numbers of columns of the new variable.

_pdblReal
: Address of real data array (size: _iCols * _iRows).

_pdblImg
: Address of imaginary data array (size: _iCols * _iRows). This argument does not exist with createMatrixOfDouble and createNamedMatrixOfDouble.

SciErr
: Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocMatrixOfDouble(void* _pvCtx, int _iVar, int _iRows, int _iCols, dou
```

```
SciErr allocComplexMatrixOfDouble(void* _pvCtx, int _iVar, int _iRows, int _iCo
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_iRows
: Number of rows of the new variable.

_iCols
: Numbers of columns of the new variable.

_pdblReal
: Returns address of real data array (size: _iCols * _iRows).

_pdblImg
: Returns address of imaginary data array (size: _iCols * _iRows). This argument does not exist with allocMatrixOfDouble.

SciErr
: Error structure where is stored errors messages history and first error number.

# Gateway Source

```
int write_double(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int i,j;
 //first variable info : real matrix of double 3 x 4
 int iRows1   = 3;
 int iCols1   = 4;
 double* pdblReal1 = NULL;

 //second variable info : complex matrix of double 4 x 6
 int iRows2   = 4;
 int iCols2   = 6;
 double* pdblReal2 = NULL;
 double* pdblImg2 = NULL;

 /************************
 *    First variable    *
 ************************/
 //alloc array of data in OS memory
 pdblReal1 = (double*)malloc(sizeof(double) * iRows1 * iCols1);

 //fill array with incremental values
 //[ 0   1   2   3
 //  4   5   6   7
 //  8   9   10  11]
 for(i = 0 ; i < iRows1 ; i++)
 {
  for(j = 0 ; j < iCols1 ; j++)
  {
   pdblReal1[i + iRows1 * j] = i * iCols1 + j;
  }
 }
 //can be written in a single loop
 //for(i = 0 ; i < iRows1 * iCols1; i++)
 //{
 //  pdblReal1[i] = i;
 //}
```

```
 //create a variable from a existing data array
 sciErr = createMatrixOfDouble(pvApiCtx, Rhs + 1, iRows1, iCols1, pdblReal1);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 //after creation, we can free memory.
 free(pdblReal1);

 /*************************
 *    Second variable    *
 *************************/
 //reserve space in scilab memory and fill it
 sciErr = allocComplexMatrixOfDouble(pvApiCtx, Rhs + 2, iRows2, iCols2, &pdblRea
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 //fill array with incremental values for real part and decremental for imagina
 //[ 23i     1+22i       2+21i       3+20i       4+19i       5+18i
 //  6+17i   7+16i       8+15i       9+14i       10+13i      11+12i
 //  12+11i  13+10i      14+9i       15+8i       16+7i       17+6i
 //  18+5i   19+4i       20+3i       21+2i       22+1i       23  ]
 for(i = 0 ; i < iRows2 ; i++)
 {
  for(j = 0 ; j < iCols2 ; j++)
  {
   pdblReal2[i + iRows2 * j] = i * iCols2 + j;
   pdblImg2 [i + iRows2 * j] = (iRows2 * iCols2 - 1) - (i * iCols2 + j);
  }
 }
 //can be written in a single loop
 //for(i = 0 ; i < iRows2 * iCols2; i++)
 //{
 //  pdblReal2[i] = i;
 //  pdblImg2 [i] = (iRows2 * iCols2 - 1) - i;
 //}

 // /!\ DO NOT FREE MEMORY, in this case, it's the Scilab memory


 //assign allocated variables to Lhs position
 LhsVar(1) = Rhs + 1;
 LhsVar(2) = Rhs + 2;
 return 0;
}
```

# Scilab test script

```
a_ref = [   0 1 2 3; ..
            4 5 6 7; ..
            8 9 10 11];
```

```
b_ref = [   23*%i,      1+22*%i,    2+21*%i,    3+20*%i,    4+19*%i,    5+18*%i
            6+17*%i,    7+16*%i,    8+15*%i,    9+14*%i,     10+13*%i,  11+12*%
            12+11*%i,   13+10*%i,   14+9*%i,    15+8*%i,    16+7*%i,    17+6*%i
            18+5*%i,    19+4*%i,    20+3*%i,    21+2*%i,    22+1*%i,    23];
[a,b] = write_double();
if or(a <> a_ref) then error("failed");end
if or(b <> b_ref) then error("failed");end
```

# Name

Integer Precision (Scilab gateway) — How to get precision of an integer matrix.

```
SciErr getMatrixOfIntegerPrecision(void* _pvCtx, int* _piAddress, int* _piPreci
```

```
SciErr getNamedMatrixOfIntegerPrecision(void* _pvCtx, char* _pstName, int* _piP
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_piAddress
: Address of the variable.

_pstName
: Name of the variable for "named" functions.

_piPrecision
: Return precision of an integer variable. ( SCI_INT8, SCI_UINT8, SCI_INT16, ... )

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to get precision of an integer matrix.

# Gateway Source

```
SciErr printf_info(int _iVar);

int common_function(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
    int i;
    int *piAddr1    = NULL;
    int iBool       = 0;

    for(i = 0 ; i < Rhs ; i++)
    {
        sciErr = printf_info(i + 1);
        if(sciErr.iErr)
        {
            printError(&sciErr, 0);
            break;
        }
        sciprint("\n\n");
    }

    //1 for true, 0 for false
    iBool = sciErr.iErr == 0 ? 1 : 0;
    sciErr = createMatrixOfBoolean(pvApiCtx, 1, 1, 1, &iBool);
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
```

```
        return 0;
    }
    //assign allocated variables to Lhs position
    LhsVar(1) = 1;

    return 0;
}

SciErr printf_info(int _iVar)
{
    SciErr sciErr;
    int* piAddr     = NULL;
    int iType       = 0;
    int iRows       = 0;
    int iCols       = 0;
    int iItem       = 0;
    int iComplex    = 0;

    sciErr = getVarAddressFromPosition(pvApiCtx, _iVar, &piAddr);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("Variable %d information:\n", _iVar);

    sciErr = getVarType(pvApiCtx, piAddr, &iType);
    if(sciErr.iErr)
    {
        return sciErr;
    }

    sciprint("\tType: ");
    switch(iType)
    {
        case sci_matrix :
            sciprint("double\n");
            break;
        case sci_poly :
            sciprint("polynomial\n");
            break;
        case sci_boolean :
            sciprint("boolean\n");
            break;
        case sci_sparse :
            sciprint("sparse\n");
            break;
        case sci_boolean_sparse :
            sciprint("boolean_sparse\n");
            break;
        case sci_ints :
        {
            char pstSigned[]    = "signed";
            char pstUnsigned[]  = "unsigned";
            char* pstSign       = pstSigned;

            int iPrec           = 0;
            sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr, &iPrec);
```

```
            if(sciErr.iErr)
            {
                return sciErr;
            }

            if(iPrec > 10)
            {
                pstSign = pstUnsigned;
            }

            sciprint("%s integer %d bits\n", pstSign, (iPrec % 10) * 8);
        }
        break;
        case sci_strings :
            sciprint("strings\n");
            break;
        case sci_list :
            sciprint("list\n");
            break;
        case sci_tlist :
            sciprint("tlist\n");
            break;
        case sci_mlist :
            sciprint("mlist\n");
            break;
        default :
            sciprint("Not manage by this function\n");
            return sciErr;
    }

    if(isVarComplex(pvApiCtx, piAddr))
    {
        sciprint("\tComplex: Yes\n");
    }

    sciprint("\tDimensions: ");
    if(isVarMatrixType(pvApiCtx, piAddr))
    {
        sciErr = getVarDimension(pvApiCtx, piAddr, &iRows, &iCols);
        if(sciErr.iErr)
        {
            return sciErr;
        }

        sciprint("%d x %d", iRows, iCols);
    }
    else
    {
        sciErr = getListItemNumber(pvApiCtx, piAddr, &iItem);
        if(sciErr.iErr)
        {
            return sciErr;
        }
        sciprint("%d", iItem);
    }
    return sciErr;
}
```

## Scilab test script

```
l1 = [1,2*%i,3;%i,2,3*%i];
l2 = ["may","the";"puffin","be";"with","you"];
l3 = int8([1,2,3]);
l4 = uint16([1000,2000,3000]);
l5 = list(l1,l2,l3);
l = list(l1,l2,l3,l4,l5);
common_function(l(1:$))
```

# Name

Integer reading (Scilab gateway) — How to read matrices of integer in a gateway.

Input argument profile:

Signed integer :

```
SciErr getMatrixOfInteger8(void* _pvCtx, int* _piAddress, int* _piRows, int* _p
SciErr getMatrixOfInteger16(void* _pvCtx, int* _piAddress, int* _piRows, int* _
SciErr getMatrixOfInteger32(void* _pvCtx, int* _piAddress, int* _piRows, int* _
```

Unsigned integer :

```
SciErr getMatrixOfUnsignedInteger8(void* _pvCtx, int* _piAddress, int* _piRows,
SciErr getMatrixOfUnsignedInteger16(void* _pvCtx, int* _piAddress, int* _piRows
SciErr getMatrixOfUnsignedInteger32(void* _pvCtx, int* _piAddress, int* _piRows
```

Named variable profile:

Signed integer :

```
SciErr readNamedMatrixOfInteger8(void* _pvCtx, char* _pstName, int* _piRows, in
SciErr readNamedMatrixOfInteger16(void* _pvCtx, char* _pstName, int* _piRows, i
SciErr readNamedMatrixOfInteger32(void* _pvCtx, char* _pstName, int* _piRows, i
```

Unsigned integer :

```
SciErr readNamedMatrixOfUnsignedInteger8(void* _pvCtx, char* _pstName, int* _pi
SciErr readNamedMatrixOfUnsignedInteger16(void* _pvCtx, char* _pstName, int* _p
SciErr readNamedMatrixOfUnsignedInteger32(void* _pvCtx, char* _pstName, int* _p
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piAddress
: Address of the Scilab variable.

_pstName
: Name of the variable for "named" functions.

_piRows
: Return number of rows.

_piCols
: Return number of columns.

_pcData8, _pucData8, _psData16, _pusData16, _piData32, _puiData32
: Returns address of array ( size: _piRows * _piCols).

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how matrices of integer can be handled through the Scilab API.

# Gateway Source

```
void* create_output(int _iCoeff, int _iSize, int _iRows, int _iCols, void* _pvD

int read_integer(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 //output variable info
 int iRows8     = 0;
 int iCols8     = 0;
 int iRows16     = 0;
 int iCols16     = 0;
 int iRows32     = 0;
 int iCols32     = 0;
 int iRowsu8     = 0;
 int iColsu8     = 0;
 int iRowsu16    = 0;
 int iColsu16    = 0;
 int iRowsu32    = 0;
 int iColsu32    = 0;
 int iPrec     = 0;

 int* piAddr8    = NULL;
 int* piAddr16    = NULL;
 int* piAddr32    = NULL;
 int* piAddru8    = NULL;
 int* piAddru16    = NULL;
 int* piAddru32    = NULL;

 char* pcData    = NULL;
 short* psData    = NULL;
 int* piData     = NULL;
 unsigned char* pucData  = NULL;
 unsigned short* pusData  = NULL;
 unsigned int* puiData  = NULL;

 char* pcDataOut    = NULL;
 short* psDataOut   = NULL;
 int* piDataOut    = NULL;
 unsigned char* pucDataOut = NULL;
 unsigned short* pusDataOut = NULL;
 unsigned int* puiDataOut = NULL;

 //check input/ouput arguments count
 CheckRhs(6,6);
 CheckLhs(6,6);

 //get varialbe address
 sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr8);
 if(sciErr.iErr)
 {
```

```
 printError(&sciErr, 0);
 return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 2, &piAddru8);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 3, &piAddr16);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 4, &piAddru16);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 5, &piAddr32);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 6, &piAddru32);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr8, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT8)
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddru8, &iPrec);
if(sciErr.iErr || iPrec != SCI_UINT8)
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr16, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT16)
```

```
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddru16, &iPrec);
if(sciErr.iErr || iPrec != SCI_UINT16)
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr32, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT32)
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddru32, &iPrec);
if(sciErr.iErr || iPrec != SCI_UINT32)
{
 printError(&sciErr, 0);
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger8(pvApiCtx, piAddr8, &iRows8, &iCols8, &pcData);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, piAddru8, &iRowsu8, &iColsu8, &p
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger16(pvApiCtx, piAddr16, &iRows16, &iCols16, &psData)
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, piAddru16, &iRowsu16, &iColsu1
if(sciErr.iErr)
{
 printError(&sciErr, 0);
```

```
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger32(pvApiCtx, piAddr32, &iRows32, &iCols32, &piData)
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, piAddru32, &iRowsu32, &iColsu3
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//alloc and fill new variable
pcDataOut   = (char*)create_output(2, 1, iRows8, iCols8, (void*)pcData);
pucDataOut  = (unsigned char*)create_output(4, 1, iRowsu8, iColsu8, (void*)puc
psDataOut   = (short*)create_output(8, 2, iRows16, iCols16, (void*)psData);
pusDataOut  = (unsigned short*)create_output(16, 2, iRowsu16, iColsu16, (void*
piDataOut   = (int*)create_output(32, 4, iRows32, iCols32, (void*)piData);
puiDataOut  = (unsigned int*)create_output(64, 4, iRowsu32, iColsu32, (void*)p

//create new variable
sciErr = createMatrixOfInteger8(pvApiCtx, Rhs + 1, iRows8, iCols8, pcDataOut);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//create new variable
sciErr = createMatrixOfUnsignedInteger8(pvApiCtx, Rhs + 2, iRowsu8, iColsu8, p
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//create new variable
sciErr = createMatrixOfInteger16(pvApiCtx, Rhs + 3, iRows16, iCols16, psDataOu
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//create new variable
sciErr = createMatrixOfUnsignedInteger16(pvApiCtx, Rhs + 4, iRowsu16, iColsu16
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}
```

```
 //create new variable
 sciErr = createMatrixOfInteger32(pvApiCtx, Rhs + 5, iRows32, iCols32, piDataOu
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 //create new variable
 sciErr = createMatrixOfUnsignedInteger32(pvApiCtx, Rhs + 6, iRowsu32, iColsu32
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 //assign allocated variables to Lhs position
 LhsVar(1) = Rhs + 1;
 LhsVar(2) = Rhs + 2;
 LhsVar(3) = Rhs + 3;
 LhsVar(4) = Rhs + 4;
 LhsVar(5) = Rhs + 5;
 LhsVar(6) = Rhs + 6;
 return 0;
}

void* create_output(int _iCoeff, int _iSize, int _iRows, int _iCols, void* _pvD
{
 int i = 0;
 void* pvDataOut = (void*)malloc(_iSize * _iRows * _iCols);
 for(i = 0 ; i < _iRows * _iCols ; i++)
 {
  int iVal = 0;
  memcpy(&iVal, (char*)_pvDataIn + i * _iSize, _iSize);
  iVal *= _iCoeff;
  memcpy((char*)pvDataOut + i * _iSize, &iVal, _iSize);
 }
 return pvDataOut;
}
```

# Scilab test script

```
a8  = int8([    1   -2  3   -4  5; ..
                -6  7   -8  9   -10; ..
                11  -12 13  -14 15]);

au8 = uint8([   1   2   3   4   5; ..
                6   7   8   9   10; ..
                11  12  13  14  15]);

a16  = int16([  1   -2  3   -4  5; ..
                -6  7   -8  9   -10; ..
                11  -12 13  -14 15]);
```

```
au16 = uint16([ 1   2   3   4   5; ..
                6   7   8   9   10; ..
                11  12  13  14  15]);

a32 = int32([   1   -2  3   -4  5; ..
                -6  7   -8  9   -10; ..
                11  -12 13  -14 15]);

au32 = uint32([ 1   2   3   4   5; ..
                6   7   8   9   10; ..
                11  12  13  14  15]);

[c8, cu8, c16, cu16, c32, cu32] = read_integer(a8, au8, a16, au16, a32, au32);

if or(c8 <> a8 * 2) then error("failed"), end
if or(cu8 <> au8 * 4) then error("failed"), end
if or(c16 <> a16 * 8) then error("failed"), end
if or(cu16 <> au16 * 16) then error("failed"), end
if or(c32 <> a32 * 32) then error("failed"), end
if or(cu32 <> au32 * 64) then error("failed"), end
```

# Name

Integer writing (Scilab gateway) — How to write matrices of integers in a gateway.

Create from existing data.

Input argument profile:

Signed integer :

```
SciErr createMatrixOfInteger8(void* _pvCtx, int _iVar, int _iRows, int _iCols, 
SciErr createMatrixOfInteger16(void* _pvCtx, int _iVar, int _iRows, int _iCols,
SciErr createMatrixOfInteger32(void* _pvCtx, int _iVar, int _iRows, int _iCols,
```

Unsigned integer :

```
SciErr createMatrixOfUnsignedInteger8(void* _pvCtx, int _iVar, int _iRows, int 
SciErr createMatrixOfUnsignedInteger16(void* _pvCtx, int _iVar, int _iRows, int
SciErr createMatrixOfUnsignedInteger32(void* _pvCtx, int _iVar, int _iRows, int
```

Named variable profile:

Signed integer :

```
SciErr createNamedMatrixOfInteger8(void* _pvCtx, char* _pstName, int _iRows, in
SciErr createNamedMatrixOfInteger16(void* _pvCtx, char* _pstName, int _iRows, i
SciErr createNamedMatrixOfInteger32(void* _pvCtx, char* _pstName, int _iRows, i
```

Unsigned integer :

```
SciErr createNamedMatrixOfUnsignedInteger8(void* _pvCtx, char* _pstName, int _i
SciErr createNamedMatrixOfUnsignedInteger16(void* _pvCtx, char* _pstName, int _
SciErr createNamedMatrixOfUnsignedInteger32(void* _pvCtx, char* _pstName, int _
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable

_pstName
: Name of the variable for "named" functions.

_iRows
: Number of rows of the new variable

_iCols
: Numbers of columns of the new variable

_pcData8, _psData16, _piData32, _pucData8, _pusData16, _puiData32
: Address of data array (size: _iCols * _iRows)

SciErr
: Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

Signed integer :

```
SciErr allocMatrixOfInteger8(void* _pvCtx, int _iVar, int _iRows, int _iCols, c
SciErr allocMatrixOfInteger16(void* _pvCtx, int _iVar, int _iRows, int _iCols,
SciErr allocMatrixOfInteger32(void* _pvCtx, int _iVar, int _iRows, int _iCols,
```

Unsigned integer :

```
SciErr allocMatrixOfUnsignedInteger8(void* _pvCtx, int _iVar, int _iRows, int _
SciErr allocMatrixOfUnsignedInteger16(void* _pvCtx, int _iVar, int _iRows, int
SciErr allocMatrixOfUnsignedInteger32(void* _pvCtx, int _iVar, int _iRows, int
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable

_iRows
: Number of rows of the new variable

_iCols
: Numbers of columns of the new variable

_pcData8, _psData16, _piData32, _pucData8, _pusData16, _puiData32
: Returns address of data array (size: _iCols * _iRows)

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how matrix of integers can be handled through the Scilab API.

Two types of functions can be used to write in the memory of Scilab.

# Gateway Source

```
void* create_output(int _iCoeff, int _iSize, int _iRows, int _iCols, void* _pvD

int read_integer(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 //output variable info
 int iRows8     = 0;
 int iCols8     = 0;
 int iRows16     = 0;
 int iCols16     = 0;
 int iRows32     = 0;
 int iCols32     = 0;
 int iRowsu8     = 0;
 int iColsu8     = 0;
```

```
int iRowsu16    = 0;
int iColsu16    = 0;
int iRowsu32    = 0;
int iColsu32    = 0;
int iPrec     = 0;

int* piAddr8    = NULL;
int* piAddr16    = NULL;
int* piAddr32    = NULL;
int* piAddru8    = NULL;
int* piAddru16    = NULL;
int* piAddru32    = NULL;

char* pcData    = NULL;
short* psData    = NULL;
int* piData     = NULL;
unsigned char* pucData  = NULL;
unsigned short* pusData  = NULL;
unsigned int* puiData  = NULL;

char* pcDataOut    = NULL;
short* psDataOut   = NULL;
int* piDataOut    = NULL;
unsigned char* pucDataOut = NULL;
unsigned short* pusDataOut = NULL;
unsigned int* puiDataOut = NULL;

//check input/ouput arguments count
CheckRhs(6,6);
CheckLhs(6,6);

//get varialbe address
sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr8);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 2, &piAddru8);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 3, &piAddr16);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 4, &piAddru16);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
```

```
}

sciErr = getVarAddressFromPosition(pvApiCtx, 5, &piAddr32);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

sciErr = getVarAddressFromPosition(pvApiCtx, 6, &piAddru32);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr8, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT8)
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddru8, &iPrec);
if(sciErr.iErr || iPrec != SCI_UINT8)
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr16, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT16)
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddru16, &iPrec);
if(sciErr.iErr || iPrec != SCI_UINT16)
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddr32, &iPrec);
if(sciErr.iErr || iPrec != SCI_INT32)
{
 printError(&sciErr, 0);
 return 0;
}

//check variable precision
sciErr = getMatrixOfIntegerPrecision(pvApiCtx, piAddru32, &iPrec);
```

```
if(sciErr.iErr || iPrec != SCI_UINT32)
{
 printError(&sciErr, 0);
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger8(pvApiCtx, piAddr8, &iRows8, &iCols8, &pcData);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger8(pvApiCtx, piAddru8, &iRowsu8, &iColsu8, &p
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger16(pvApiCtx, piAddr16, &iRows16, &iCols16, &psData)
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger16(pvApiCtx, piAddru16, &iRowsu16, &iColsu1
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfInteger32(pvApiCtx, piAddr32, &iRows32, &iCols32, &piData)
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//retrieve dimensions and data
sciErr = getMatrixOfUnsignedInteger32(pvApiCtx, piAddru32, &iRowsu32, &iColsu3
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//alloc and fill new variable
pcDataOut   = (char*)create_output(2, 1, iRows8, iCols8, (void*)pcData);
pucDataOut  = (unsigned char*)create_output(4, 1, iRowsu8, iColsu8, (void*)puc
psDataOut   = (short*)create_output(8, 2, iRows16, iCols16, (void*)psData);
```

```
pusDataOut  = (unsigned short*)create_output(16, 2, iRowsu16, iColsu16, (void*
piDataOut   = (int*)create_output(32, 4, iRows32, iCols32, (void*)piData);
puiDataOut  = (unsigned int*)create_output(64, 4, iRowsu32, iColsu32, (void*)p

//create new variable
sciErr = createMatrixOfInteger8(pvApiCtx, Rhs + 1, iRows8, iCols8, pcDataOut);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//create new variable
sciErr = createMatrixOfUnsignedInteger8(pvApiCtx, Rhs + 2, iRowsu8, iColsu8, p
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//create new variable
sciErr = createMatrixOfInteger16(pvApiCtx, Rhs + 3, iRows16, iCols16, psDataOu
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//create new variable
sciErr = createMatrixOfUnsignedInteger16(pvApiCtx, Rhs + 4, iRowsu16, iColsu16
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//create new variable
sciErr = createMatrixOfInteger32(pvApiCtx, Rhs + 5, iRows32, iCols32, piDataOu
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//create new variable
sciErr = createMatrixOfUnsignedInteger32(pvApiCtx, Rhs + 6, iRowsu32, iColsu32
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//assign allocated variables to Lhs position
LhsVar(1) = Rhs + 1;
LhsVar(2) = Rhs + 2;
LhsVar(3) = Rhs + 3;
LhsVar(4) = Rhs + 4;
LhsVar(5) = Rhs + 5;
```

```
 LhsVar(6) = Rhs + 6;
 return 0;
}

void* create_output(int _iCoeff, int _iSize, int _iRows, int _iCols, void* _pvDa
{
 int i = 0;
 void* pvDataOut = (void*)malloc(_iSize * _iRows * _iCols);
 for(i = 0 ; i < _iRows * _iCols ; i++)
 {
  int iVal = 0;
  memcpy(&iVal, (char*)_pvDataIn + i * _iSize, _iSize);
  iVal *= _iCoeff;
  memcpy((char*)pvDataOut + i * _iSize, &iVal, _iSize);
 }
 return pvDataOut;
}
```

## Scilab test script

```
a8  = int8([    1   -2  3   -4  5; ..
                -6  7   -8  9   -10; ..
                11  -12 13  -14 15]);

au8 = uint8([   1   2   3   4   5; ..
                6   7   8   9   10; ..
                11  12  13  14  15]);

a16  = int16([  1   -2  3   -4  5; ..
                -6  7   -8  9   -10; ..
                11  -12 13  -14 15]);

au16 = uint16([ 1   2   3   4   5; ..
                6   7   8   9   10; ..
                11  12  13  14  15]);

a32 = int32([   1   -2  3   -4  5; ..
                -6  7   -8  9   -10; ..
                11  -12 13  -14 15]);

au32 = uint32([ 1   2   3   4   5; ..
                6   7   8   9   10; ..
                11  12  13  14  15]);

[c8, cu8, c16, cu16, c32, cu32] = read_integer(a8, au8, a16, au16, a32, au32);

if or(c8 <> a8 * 2) then error("failed"), end
if or(cu8 <> au8 * 4) then error("failed"), end
if or(c16 <> a16 * 8) then error("failed"), end
if or(cu16 <> au16 * 16) then error("failed"), end
if or(c32 <> a32 * 32) then error("failed"), end
if or(cu32 <> au32 * 64) then error("failed"), end
```

# Name

Pointer reading (Scilab gateway) — How to read pointer in a gateway.

Input argument profile:

```
SciErr createNamedPointer(void* _pvCtx, char* _pstName, void* _pvPtr)
```

Named variable profile:

```
SciErr readNamedPointer(void* _pvCtx, char* _pstName, void** _pvPtr)
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_piAddress
: Address of the Scilab variable.

_pvPtr
: Return address of pointer.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how pointer can be handled through the Scilab API.

# Gateway Source

```
int read_pointer(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 CheckRhs(0,1);
 CheckLhs(1,1);

 if(Rhs == 0)
 {//create mode
  double* pdblData = (double*)malloc(sizeof(double) * 2 * 2);
  pdblData[0]   = 1;
  pdblData[1]   = 3;
  pdblData[2]   = 2;
  pdblData[3]   = 4;

  sciErr = createPointer(pvApiCtx, Rhs + 1, (void*)pdblData);
 }
 else if(Rhs == 1)
 {//read mode
  int iType   = 0;
  int* piAddr   = NULL;
  void* pvPtr   = NULL;
  double* pdblData = NULL;

  sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
```

```
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  sciErr = getPointer(pvApiCtx, piAddr, &pvPtr);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblData = (double*)pvPtr;

  sciErr = createMatrixOfDouble(pvApiCtx, Rhs + 1, 2, 2, pdblData);
 }
 else
 {
  return 0;
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = Rhs + 1;
 return 0;
}
```

# Scilab test script

```
b_ref = [1,2;3,4];
a = read_pointer();
b = read_pointer(a);
if or(b <> b_ref) then error("failed"), end
```

# Name

Pointer writing (Scilab gateway) — How to write pointer in a gateway.

Create from existing data.

Input argument profile:

```
SciErr getPointer(void* _pvCtx, int* _piAddress, void** _pvPtr)
```

Named variable profile:

```
SciErr createNamedPointer(void* _pvCtx, char* _pstName, void** _pvPtr)
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_piAddress
: Address of the Scilab variable.

_pvPtr
: Address of pointer.

SciErr
: Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

```
SciErr allocPointer(void* _pvCtx, int _iVar, void** _pvPtr)
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piAddress
: Address of the Scilab variable.

_pvPtr
: Return address of pointer.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how pointer can be handled through the Scilab API.

Two types of functions can be used to write in the memory of Scilab.

# Gateway Source

```
int read_pointer(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
```

```
 CheckRhs(0,1);
 CheckLhs(1,1);

 if(Rhs == 0)
 {//create mode
  double* pdblData = (double*)malloc(sizeof(double) * 2 * 2);
  pdblData[0]   = 1;
  pdblData[1]   = 3;
  pdblData[2]   = 2;
  pdblData[3]   = 4;

  sciErr = createPointer(pvApiCtx, Rhs + 1, (void*)pdblData);
 }
 else if(Rhs == 1)
 {//read mode
  int iType   = 0;
  int* piAddr   = NULL;
  void* pvPtr   = NULL;
  double* pdblData = NULL;

  sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  sciErr = getPointer(pvApiCtx, piAddr, &pvPtr);
  if(sciErr.iErr)
  {
   printError(&sciErr, 0);
   return 0;
  }

  pdblData = (double*)pvPtr;

  sciErr = createMatrixOfDouble(pvApiCtx, Rhs + 1, 2, 2, pdblData);
 }
 else
 {
  return 0;
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 LhsVar(1) = Rhs + 1;
 return 0;
}
```

## Scilab test script

```
```

```
b_ref = [1,2;3,4];
a = read_pointer();
b = read_pointer(a);
if or(b <> b_ref) then error("failed"), end
```

# Name

Polynomial Symbolic Variable (Scilab gateway) — How to get the symbolic variable name.

Input argument profile:

```
SciErr getPolyVariableName(void* _pvCtx, int* _piAddress, char* _pstVarName, in
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piAddress
: Address of the variable.

_pstVarName
: Return the symbolic varaible name

_piVarNameLen
: Return the length of _pstVarName

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to get the symbolic variable name.

# Gateway Source

```
int read_poly(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int i,j;
 //variable info
 int iRows   = 0;
 int iCols   = 0;
 int iVarLen   = 0;
 int* piAddr   = NULL;
 int* piNbCoef  = NULL;
 double** pdblReal = NULL;
 double** pdblImg = NULL;
 char* pstVarname = NULL;

 //check input and output arguments
 CheckRhs(1,1);
 CheckLhs(1,1);

 sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 if(isVarComplex(pvApiCtx, piAddr) == FALSE)
 {
```

```
 //Error
 return 0;
}

//get variable name length
sciErr = getPolyVariableName(pvApiCtx, piAddr, NULL, &iVarLen);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//alloc buff to receive variable name
pstVarname = (char*)malloc(sizeof(char) * (iVarLen + 1));//1 for null terminat
//get variable name
sciErr = getPolyVariableName(pvApiCtx, piAddr, pstVarname, &iVarLen);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//First call: retrieve dimmension
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, NULL, NULL, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//alloc array of coefficient
piNbCoef = (int*)malloc(sizeof(int) * iRows * iCols);
//Second call: retrieve coefficient
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, piNbCoef, NU
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//alloc arrays of data
pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
 pdblReal[i] = (double*)malloc(sizeof(double) * piNbCoef[i]);
 pdblImg[i] = (double*)malloc(sizeof(double) * piNbCoef[i]);
}

//Third call: retrieve data
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, piNbCoef, pd
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}
```

```
//Do something with Data

//Invert polynomials in the matrix and invert coefficients

for(i = 0 ; i < (iRows * iCols) / 2 ; i++)
{
 int iPos1   = iRows * iCols - 1 - i;
 double* pdblSave = NULL;
 int iNbCoefSave  = 0;

 //switch array of coefficient
 pdblSave   = pdblReal[i];
 pdblReal[i]   = pdblReal[iPos1];
 pdblReal[iPos1]  = pdblSave;

 pdblSave   = pdblImg[i];
 pdblImg[i]   = pdblImg[iPos1];
 pdblImg[iPos1]  = pdblSave;

 //switch number of coefficient
 iNbCoefSave   = piNbCoef[i];
 piNbCoef[i]   = piNbCoef[iPos1];
 piNbCoef[iPos1]  = iNbCoefSave;
}

//switch coefficient
for(i = 0 ; i < iRows * iCols ; i++)
{
 for(j = 0 ; j < piNbCoef[i] /2 ; j++)
 {
  int iPos2   = piNbCoef[i] - 1 - j;
  double dblVal  = pdblReal[i][j];
  pdblReal[i][j]  = pdblReal[i][iPos2];
  pdblReal[i][iPos2] = dblVal;

  dblVal    = pdblImg[i][j];
  pdblImg[i][j]  = pdblImg[i][iPos2];
  pdblImg[i][iPos2] = dblVal;
 }
}

sciErr = createComplexMatrixOfPoly(pvApiCtx, Rhs + 1, pstVarname, iRows, iCols
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//free OS memory
free(pstVarname);
free(piNbCoef);

for(i = 0 ; i < iRows * iCols ; i++)
{
 free(pdblReal[i]);
 free(pdblImg[i]);
}
```

```
 free(pdblReal);
 free(pdblImg);

 //assign allocated variables to Lhs position
 LhsVar(1) = Rhs + 1;
 return 0;
}
```

# Scilab test script

```
coeff1 = [ ..
29*%i,22*%i,16*%i,11*%i,7*%i,30,23,17,12,8,-31*%i,-24*%i,-18*%i,-13*%i,-9*%i,32
4*%i,2*%i,%i,22,16,5,-3,0,-23*%i,-17*%i,-6*%i,0,0,24,18,0,0,0,-25*%i,-19*%i,0,0
11,7,4,2,1,-12*%i,-8*%i,-5*%i,3*%i,0,13,9,6,0,0,-14*%i,-10*%i,0,0,0,15,0,0,0,0,

x = poly(0, "x");
p1  = 1;
p2  = 2 * x + 3 * %i;
p3  = 4 * x**2 - 5 * %i * x + 6;
p4  = 7 * x**3 - 8 * %i * x**2 + 9 * x - 10 * %i;
p5  = 11 * x**4 - 12 * %i * x**3 + 13 * x**2 - 14 * %i * x + 15;
p6  = 16 * x**5 - 17 * %i * x**4 + 18 * x**3 - 19 * %i * x**2 + 20 * x  - 21 * 
p7  = 22 * x**6 - 23 * %i * x**5 + 24 * x**4 - 25 * %i * x**3 + 26 * x**2 - 27
p8  = %i;
p9  = 2 * %i * x - 3;
p10 = 4 * %i * x**2 + 5 * x - 6 * %i;
p11 = 7 * %i * x**3 + 8 * x**2 - 9 * %i * x + 10;
p12 = 11 * %i * x**4 + 12 * x**3 - 13 * %i * x**2 + 14 * x - 15 * %i;
p13 = 16 * %i * x**5 + 17 * x**4 - 18 * %i * x**3 + 19 * x**2 - 20 * %i * x  +
p14 = 22 * %i * x**6 + 23 * x**5 - 24 * %i * x**4 + 25 * x**3 - 26 * %i * x**2
p15 = 29 * %i * x**7 + 30 * x**6 - 31 * %i * x**5 + 32 * x**4 - 33 * %i * x**3
p = [p1, p2, p3, p4, p5 ; p6, p7, p8, p9 ,p10 ; p11, p12, p13, p14, p15];

p1 = read_poly(p);
coeff2 = coeff(p1);
if or(coeff2 <> coeff1) then error("failed"), end
```

# Name

Polynomial reading (Scilab gateway) — How to read matrices of polynomials in a gateway.

Input argument profile:

```
SciErr getMatrixOfPoly(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCol
```

```
SciErr getComplexMatrixOfPoly(void* _pvCtx, int* _piAddress, int* _piRows, int*
```

Named variable profile:

```
SciErr readNamedMatrixOfPoly(void* _pvCtx, char* _pstName, int* _piRows, int* _
```

```
SciErr readNamedComplexMatrixOfPoly(void* _pvCtx, char* _pstName, int* _piRows,
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piAddress
: Address of the Scilab variable.

_pstName
: Name of the variable for "named" functions.

_piRows
: Return number of rows.

_piCols
: Return number of columns.

_piNbCoef
: Return number of coefficient for each polynomial. (must be allocated)

_pdblReal
: Address of array of double* with imaginary part of coefficient (size: _iCols * _iRows, must be allocated)

_pdblImg
: Address of array of double* with imaginary part of coefficient (size: _iCols * _iRows, must be allocated)

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how matrix of polynomials can be handled through the Scilab API.

# Gateway Source

```
int read_poly(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int i,j;
 //variable info
 int iRows   = 0;
 int iCols   = 0;
```

```
int iVarLen   = 0;
int* piAddr   = NULL;
int* piNbCoef  = NULL;
double** pdblReal = NULL;
double** pdblImg = NULL;
char* pstVarname = NULL;

//check input and output arguments
CheckRhs(1,1);
CheckLhs(1,1);

sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

if(isVarComplex(pvApiCtx, piAddr) == FALSE)
{
 //Error
 return 0;
}

//get variable name length
sciErr = getPolyVariableName(pvApiCtx, piAddr, NULL, &iVarLen);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//alloc buff to receive variable name
pstVarname = (char*)malloc(sizeof(char) * (iVarLen + 1));//1 for null terminat
//get variable name
sciErr = getPolyVariableName(pvApiCtx, piAddr, pstVarname, &iVarLen);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//First call: retrieve dimmension
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, NULL, NULL, 
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//alloc array of coefficient
piNbCoef = (int*)malloc(sizeof(int) * iRows * iCols);
//Second call: retrieve coefficient
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, piNbCoef, NU
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
```

```
}

//alloc arrays of data
pdblReal    = (double**)malloc(sizeof(double*) * iRows * iCols);
pdblImg     = (double**)malloc(sizeof(double*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
 pdblReal[i] = (double*)malloc(sizeof(double) * piNbCoef[i]);
 pdblImg[i] = (double*)malloc(sizeof(double) * piNbCoef[i]);
}

//Third call: retrieve data
sciErr = getComplexMatrixOfPoly(pvApiCtx, piAddr, &iRows, &iCols, piNbCoef, pd
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}


//Do something with Data

//Invert polynomials in the matrix and invert coefficients

for(i = 0 ; i < (iRows * iCols) / 2 ; i++)
{
 int iPos1   = iRows * iCols - 1 - i;
 double* pdblSave = NULL;
 int iNbCoefSave  = 0;

 //switch array of coefficient
 pdblSave   = pdblReal[i];
 pdblReal[i]   = pdblReal[iPos1];
 pdblReal[iPos1]  = pdblSave;

 pdblSave   = pdblImg[i];
 pdblImg[i]   = pdblImg[iPos1];
 pdblImg[iPos1]  = pdblSave;

 //switch number of coefficient
 iNbCoefSave   = piNbCoef[i];
 piNbCoef[i]   = piNbCoef[iPos1];
 piNbCoef[iPos1]  = iNbCoefSave;
}

//switch coefficient
for(i = 0 ; i < iRows * iCols ; i++)
{
 for(j = 0 ; j < piNbCoef[i] /2 ; j++)
 {
  int iPos2   = piNbCoef[i] - 1 - j;
  double dblVal  = pdblReal[i][j];
  pdblReal[i][j]  = pdblReal[i][iPos2];
  pdblReal[i][iPos2] = dblVal;

  dblVal    = pdblImg[i][j];
  pdblImg[i][j]  = pdblImg[i][iPos2];
  pdblImg[i][iPos2] = dblVal;
```

```
  }
 }

 sciErr = createComplexMatrixOfPoly(pvApiCtx, Rhs + 1, pstVarname, iRows, iCols
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 //free OS memory
 free(pstVarname);
 free(piNbCoef);

 for(i = 0 ; i < iRows * iCols ; i++)
 {
  free(pdblReal[i]);
  free(pdblImg[i]);
 }

 free(pdblReal);
 free(pdblImg);

 //assign allocated variables to Lhs position
 LhsVar(1) = Rhs + 1;
 return 0;
}
```

# Scilab test script

```
coeff1 = [ ..
29*%i,22*%i,16*%i,11*%i,7*%i,30,23,17,12,8,-31*%i,-24*%i,-18*%i,-13*%i,-9*%i,32
4*%i,2*%i,%i,22,16,5,-3,0,-23*%i,-17*%i,-6*%i,0,0,24,18,0,0,0,-25*%i,-19*%i,0,0
11,7,4,2,1,-12*%i,-8*%i,-5*%i,3*%i,0,13,9,6,0,0,-14*%i,-10*%i,0,0,0,15,0,0,0,0,

x = poly(0, "x");
p1  = 1;
p2  = 2 * x + 3 * %i;
p3  = 4 * x**2 - 5 * %i * x + 6;
p4  = 7 * x**3 - 8 * %i * x**2 + 9 * x - 10 * %i;
p5  = 11 * x**4 - 12 * %i * x**3 + 13 * x**2 - 14 * %i * x + 15;
p6  = 16 * x**5 - 17 * %i * x**4 + 18 * x**3 - 19 * %i * x**2 + 20 * x  - 21 * 
p7  = 22 * x**6 - 23 * %i * x**5 + 24 * x**4 - 25 * %i * x**3 + 26 * x**2 - 27
p8  = %i;
p9  = 2 * %i * x - 3;
p10 = 4 * %i * x**2 + 5 * x - 6 * %i;
p11 = 7 * %i * x**3 + 8 * x**2 - 9 * %i * x + 10;
p12 = 11 * %i * x**4 + 12 * x**3 - 13 * %i * x**2 + 14 * x - 15 * %i;
p13 = 16 * %i * x**5 + 17 * x**4 - 18 * %i * x**3 + 19 * x**2 - 20 * %i * x  +
p14 = 22 * %i * x**6 + 23 * x**5 - 24 * %i * x**4 + 25 * x**3 - 26 * %i * x**2
p15 = 29 * %i * x**7 + 30 * x**6 - 31 * %i * x**5 + 32 * x**4 - 33 * %i * x**3
p = [p1, p2, p3, p4, p5 ; p6, p7, p8, p9 ,p10 ; p11, p12, p13, p14, p15];

p1 = read_poly(p);
coeff2 = coeff(p1);
if or(coeff2 <> coeff1) then error("failed"), end
```

# Name

Polynomial writing (Scilab gateway) — How to write matrices of polynomials in a gateway.

Input argument profile:

```
SciErr createMatrixOfPoly(void* _pvCtx, int _iVar, char* _pstVarName, int _iRow
```

```
SciErr createMatrixOfPoly(void* _pvCtx, int _iVar, char* _pstVarName, int _iRow
```

Named variable profile:

```
SciErr createNamedMatrixOfPoly(void* _pvCtx, char* _pstName, char* _pstVarName,
```

```
SciErr createNamedComplexMatrixOfPoly(void* _pvCtx, char* _pstName, char* _pstV
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable

_pstName
: Name of the variable for "named" functions.

_pstVarName
: Variable name in polynomials (Scilab5: 4 characters max)

_iRows
: Number of rows of the new variable

_iCols
: Numbers of columns of the new variable

_piNbCoef
: Number of coefficient for each polynomial.

_pdblReal
: Address of array of double* with real part of coefficient (size: _iCols * _iRows)

_pdblImg
: Address of array of double* with imaginary part of coefficient (size: _iCols * _iRows)

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how matrix of polynomials can be handled through the Scilab API.

# Gateway Source

```
int write_poly(char *fname,unsigned long fname_len)
{
    SciErr sciErr;
```

```
    //output variable info : polinomial matrix 2 x 4
    //[ x + 2                   x^2 - 4x + 5    4x^3 - 14x^2 + 18 ;
    //  2x^3 - 12x^2 + 64       1               8x^5 + 32x^3]
    int iRows               = 2;
    int iCols               = 3;

    //varname "x"
    char pstVarName[2]      = {"x"};

    //coeficient array
    int piNbCoef[6]         = {2,4,3,1,4,6};

    //data array
    double *pdblReal[6]     = {0};
    double pdblPoly0[2]     = {2,1};
    double pdblPoly1[4]     = {64,0,-12,2};
    double pdblPoly2[3]     = {5,-4,1};
    double pdblPoly3[1]     = {1};
    double pdblPoly4[4]     = {18,0,-14,4};
    double pdblPoly5[6]     = {0,0,0,32,0,8};

    pdblReal[0]             = pdblPoly0;
    pdblReal[1]             = pdblPoly1;
    pdblReal[2]             = pdblPoly2;
    pdblReal[3]             = pdblPoly3;
    pdblReal[4]             = pdblPoly4;
    pdblReal[5]             = pdblPoly5;

    sciErr = createMatrixOfPoly(pvApiCtx, Rhs + 1, pstVarName, iRows, iCols, pi
    if(sciErr.iErr)
    {
        printError(&sciErr, 0);
        return 0;
    }

    //assign allocated variables to Lhs position
    LhsVar(1) = Rhs + 1;
    return 0;
}
```

# Scilab test script

```
p_ref = [2 5 18 1 -4 0 0 1 -14 0 0 4 0 0 0 0 0 0;64 1 0 0 0 0 -12 0 0 2 0 32 0 
l = list();
a = write_poly();
p = coeff(a);
if or(p <> p_ref) then error("failed"), end
```

# Name

Sparse matrix reading (Scilab gateway) — How to read sparse matric in a gateway.

Input argument profile:

```
SciErr getSparseMatrix(void* _pvCtx, int* _piAddress, int* _piRows, int* _piCol
```

```
SciErr getComplexSparseMatrix(void* _pvCtx, int* _piAddress, int* _piRows, int*
```

Named variable profile:

```
SciErr readNamedSparseMatrix(void* _pvCtx, char* _pstName, int* _piRows, int* _
```

```
SciErr readNamedComplexSparseMatrix(void* _pvCtx, char* _pstName, int* _piRows,
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piAddress
: Address of the Scilab variable.

_pstName
: Name of the variable for "named" functions.

_piRows
: Return number of rows.

_piCols
: Return number of columns.

_piNbItem
: Return number of non zero value.

_piNbItemRow
: Return number of item in each rows (size: _iRows).

_piColPos
: Return column position for each item (size: _iNbItem).

_pdblReal
: Return address of real data array (size: _iCols * _iRows)

_pdblImg
: Return address of imaginary data array (size: _iCols * _iRows)

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how sparse matrix can be handled through the Scilab API.

# Gateway Source

```
int read_sparse(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
```

```
 int i,j,k;
 int* piAddr   = NULL;

 int iRows   = 0;
 int iCols   = 0;
 int iNbItem   = 0;
 int* piNbItemRow = NULL;
 int* piColPos  = NULL;

 double* pdblReal = NULL;
 double* pdblImg  = NULL;

 CheckRhs(1,1);

 sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 if(isVarComplex(pvApiCtx, piAddr))
 {
  sciErr = getComplexSparseMatrix(pvApiCtx, piAddr, &iRows, &iCols, &iNbItem, &p
 }
 else
 {
  sciErr = getSparseMatrix(pvApiCtx, piAddr, &iRows, &iCols, &iNbItem, &piNbItem
 }

 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 sciprint("Sparse %d item(s)\n", iNbItem);

 k = 0;
 for(i = 0 ; i < iRows ; i++)
 {
  for(j = 0 ; j < piNbItemRow[i] ; j++)
  {
   sciprint("(%d,%d) = %f", i+1, piColPos[k], pdblReal[k]);
   if(isVarComplex(pvApiCtx, piAddr))
   {
    sciprint(" %+fi", pdblImg[k]);
   }
   sciprint("\n");
   k++;
  }
 }

 //assign allocated variables to Lhs position
 LhsVar(1) = 0;
 return 0;
}
```

# Scilab test script

```
sp=sparse([1,2;4,5;3,10],[1 + 2*%i,2 - 3*%i,-3 + 4*%i]);
read_sparse(sp);
```

# Name

Sparse writing (Scilab gateway) — How to write sparse matrix in a gateway.

Create from existing data.

Input argument profile:

```
SciErr createSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCols, int _
```

```
SciErr createComplexSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCol
```

Named variable profile:

```
SciErr createNamedSparseMatrix(void* _pvCtx, char* _pstName, int _iRows, int _i
```

```
SciErr createNamedComplexSparseMatrix(void* _pvCtx, char* _pstName, int _iRows,
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h

_iVar
: Position in the Scilab memory where you want to put the variable.

_pstName
: Name of the variable for "named" functions.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_iNbItem
: Number of non zero itmes in the sparse.

_piNbItemRow
: Number of item in each rows (size: _iRows).

_piColPos
: Column position for each item (size: _iNbItem).

_pdblReal
: Address of real data array (size: _iNbItem).

_pdblImg
: Address of imaginary data array (size: _iNbItem).

  This argument does not exist with createSparseMatrix and createNamedSparseMatrix.

SciErr
: Error structure where is stored errors messages history and first error number.

Write directly in Scilab memory.

Input argument profile:

```
SciErr allocSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCols, int _
```

```
SciErr allocComplexSparseMatrix(void* _pvCtx, int _iVar, int _iRows, int _iCols
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable.

_iRows
: Number of rows of the new variable.

_iCols
: Number of columns of the new variable.

_iNbItem
: Number of non zero itmes in the sparse.

_piNbItemRow
: Return address of number of item in each rows (size: _iRows).

_piColPos
: Return address of column position for each item (size: _iNbItem).

_pdblReal
: Address of real data array (size: _iNbItem).

_pdblImg
: Address of imaginary data array (size: _iNbItem).

This argument does not exist with allocSparseMatrix.

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how to add sparse matrix.

Two types of functions can be used to write in the memory of Scilab.

# Gateway Source

```
int write_sparse(char *fname,unsigned long fname_len)
{
 SciErr sciErr;

 int piNbItemRow[] = {1,2,1};
 int piColPos[]  = {8,4,7,2};
 double pdblSReal[] = {1,2,3,4};
 double pdblSImg[] = {4,3,2,1};
 int iNbItem   = 4;

 sciErr = createComplexSparseMatrix(pvApiCtx, Rhs + 1, 3, 10, iNbItem, piNbItem
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
```

```
 }

 LhsVar(1) = 1;
 return 0;
}
```

## Scilab test script

```
sp_ref = sparse([1,8;2,4;2,7;3,2],[1+4*%i,2+3*%i,3+2*%i,4+%i], [3,10]);
sp = write_sparse();
if or(sp <> sp_ref) then error("failed"), end
```

# Name

String reading (Scilab gateway) — How to read matrices of strings in a gateway.

Input argument profile:

```
SciErr getMatrixOfString(void* _pvCtx, int* _piAddress, int* _piRows, int* _piC
```

```
SciErr getMatrixOfWideString(void* _pvCtx, int* _piAddress, int* _piRows, int* 
```

Named variable profile:

```
SciErr createNamedMatrixOfString(void* _pvCtx, char* _pstName, int _iRows, int 
```

```
SciErr createNamedMatrixOfWideString(void* _pvCtx, char* _pstName, int _iRows, 
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_piAddress
: Address of the Scilab variable.

_pstName
: Name of the variable for "named" functions.

_piRows
: Return number of rows.

_piCols
: Return number of columns.

_piLength
: Address of array of strings length (must be allocated size: _piRows * _piCols)

_pstStrings
: Address of array of char* (must be allocated size: _piRows * _piCols)

_pwstStrings
: Address of array of wchar_t* (must be allocated size: _piRows * _piCols)

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how matrix of strings can be handled through the Scilab API.

# Gateway Source

```
int read_string(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 int i,j;
 int iLen  = 0;
 //variable info
 int iRows  = 0;
 int iCols  = 0;
 int* piAddr  = NULL;
```

```
int* piLen  = NULL;
char** pstData = NULL;

//output variable
int iRowsOut = 1;
int iColsOut = 1;
char* pstOut = NULL;

//check input and output arguments
CheckRhs(1,1);
CheckLhs(1,1);

//get variable address
sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//fisrt call to retrieve dimensions
sciErr = getMatrixOfString(pvApiCtx, piAddr, &iRows, &iCols, NULL, NULL);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

piLen = (int*)malloc(sizeof(int) * iRows * iCols);
//second call to retrieve length of each string
sciErr = getMatrixOfString(pvApiCtx, piAddr, &iRows, &iCols, piLen, NULL);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
 pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1));//+ 1 for null term
}
//third call to retrieve data
sciErr = getMatrixOfString(pvApiCtx, piAddr, &iRows, &iCols, piLen, pstData);
if(sciErr.iErr)
{
 printError(&sciErr, 0);
 return 0;
}

//computer length of all strings
for(i = 0 ; i < iRows * iCols ; i++)
{
 iLen += piLen[i];
}

//alloc output variable
pstOut = (char*)malloc(sizeof(char) * (iLen + iRows * iCols));
```

```
 //initialize string to 0x00
 memset(pstOut, 0x00, sizeof(char) * (iLen + iRows * iCols));

 //concat input strings in output string
 for(i = 0 ; i < iRows ; i++)
 {
  for(j = 0 ; j < iCols ; j++)
  {
   int iCurLen = strlen(pstOut);
   if(iCurLen)
   {
    strcat(pstOut, " ");
   }
   strcpy(pstOut + strlen(pstOut), pstData[j * iRows + i]);
  }
 }

 //create new variable
 sciErr = createMatrixOfString(pvApiCtx, Rhs + 1, iRowsOut, iColsOut, &pstOut);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 //free memory
 free(piLen);
 for(i = 0 ; i < iRows * iCols ; i++)
 {
  free(pstData[i]);
 }
 free(pstData);
 free(pstOut);

 LhsVar(1) = Rhs + 1;
 return 0;
}
```

# Scilab test script

```
a_ref = ["may the puffin be with you"];
a = ["may", "the", "puffin"; "be","with","you"];
b = read_string(a);
if a_ref <> b then error("failed"), end
```

# Name

String writing (Scilab gateway) — How to write matrices of string in a gateway.

Input argument profile:

```
SciErr createMatrixOfString(void* _pvCtx, int _iVar, int _iRows, int _iCols, cha
```

```
SciErr createMatrixOfWideString(void* _pvCtx, int _iVar, int _iRows, int _iCols
```

Named variable profile:

```
SciErr createNamedMatrixOfString(void* _pvCtx, char* _pstName, int _iRows, int _
```

```
SciErr createNamedMatrixOfWideString(void* _pvCtx, char* _pstName, int _iRows, i
```

# Parameters

_pvCtx
: Scilab environment pointer, pass in "pvApiCtx" provided by api_scilab.h.

_iVar
: Position in the Scilab memory where you want to put the variable

_pstName
: Name of the variable for "named" functions.

_iRows
: Number of rows of the new variable

_iCols
: Numbers of columns of the new variable

_pstStrings
: Address of array of char* (size: _iCols * _iRows)

SciErr
: Error structure where is stored errors messages history and first error number.

# Description

This help describes how matrix of strings can be handled through the Scilab API.

# Gateway Source

```
int write_string(char *fname,unsigned long fname_len)
{
 SciErr sciErr;
 //variable info : matrix of string 2 x 3
 int iRows  = 2;
 int iCols  = 3;
 char** pstData = NULL;

 //data to put in the new variable
 char string11[] = "may";
 char string21[] = "be";
```

```
 char string12[] = "the";
 char string22[] = "with";
 char string13[] = "puffin";
 char string23[] = "you";

 //alloc new array
 pstData   = (char**)malloc(sizeof(char*) * iRows * iCols);

 //copy data address to the "main" array
 pstData[0]  = string11;
 pstData[1]  = string21;
 pstData[2]  = string12;
 pstData[3]  = string22;
 pstData[4]  = string13;
 pstData[5]  = string23;

 //create the variable
 sciErr = createMatrixOfString(pvApiCtx, Rhs + 1, iRows, iCols, pstData);
 if(sciErr.iErr)
 {
  printError(&sciErr, 0);
  return 0;
 }

 //free container
 free(pstData);

 //assign allocated variables to Lhs position
 LhsVar(1) = Rhs + 1;
 return 0;
}
```

# Scilab test script

```
a_ref = "may the puffin be with you ";
b = [];
a = write_string();
for i=1:size(a,"r")
    for j=1:size(a,"c")
        b = b + a(i,j);
        b = b + " ";
    end
end
if b <> a_ref then error("failed"), end
```

# Parte LVIII. Gerenciamento de ajuda online

# Name

add_help_chapter — adiciona uma entrada na lista de ajudas

```
add_help_chapter(title,path[,mode])
```

# Parâmetros

title
: string, o título da ajuda

path
: string, o endereço do diretório contendo os arquivos de ajuda

mode
: booleano, %T se o diretório pertence à lista de módulos do Scilab, %F em caso contrário (toolboxes). O valor padrão é %F.

# Descrição

Esta função adiciona uma nova entrada na lista de ajudas. Os arquivos do capítulo de ajuda devem estar localizados em um único diretório. Se o dado título `title` já existir na lista de ajudas associado com o mesmo endereço, nada é feito. A função verifica se o diretório existe.

# Ver Também

help, add_demo

# Autor

Serge Steer , INRIA

# Name

apropos — procura por palavras-chave na ajuda do Scilab

```
apropos(key)
apropos(regexp)
```

# Parâmetros

key
: string. Fornece a seqüência de caracteres a ser encontrada.

regexp
: string. Fornece a expressão regular a ser encontrada (apenas com "Scilab Browser")

# Descrição

`apropos(key)` procura por arquivos de ajuda Scilab contendo a palavra-chave `key` na seção de descrições breves.

`apropos(regexp)` procura por arquivos de ajuda Scilab contendo a expressão regular `regexp` na seção de descrições breves.

# Exemplos

```
apropos('ode')
apropos ode
apropos "list of"
apropos "sin.*hyperbolic"
apropos "^ab"  //procura por ajuda começando com os dois caracteres "ab"
apropos "quadratic.*solver"
```

# Ver Também

help, man

# Name

foo — descrição breve de foo

```
[y] = foo(x)
```

# Parâmetros

x
: o que puder ser x

y
: o que puder ser y

# Descrição

Um primeiro parárgrafo que explica o que a função foo computa. Se quiser enfatizar um parâmetro, então você utiliza o seguinte tag `x`, se quiser enfatizar parte do texto *encerre-o nestes tags* e então utilize estes **para obter uma fonte em negrito** e finalmente `para um estilo em máquina de escrever`.

Um segundo parágrafo... Aqui está um exemplo de um link para outra página : man.

first
: parágrafo simples

second
: toto é o foo francês...

# Exemplos

```
deff("y=foo(x)","y=x"); // define a função foo como a função identidade.
foo("toto")
```

# Ver Também

man, apropos

# Autor

B. P.

# Name

help — comando de ajuda on-line

```
help(key)
help
```

# Parâmetros

key
: string. Fornece a página de ajuda a ser encontrada

# Descrição

`help` sem argumentos fornece a página de hipertextos dos capítulos de ajuda.

`help(key)` exibe o arquivo de ajuda Scilab associado ao dado `key`. Se nenhum arquivo for encontrado, `help(key)` chama automaticamente `apropos(key)`.

Ver man para mais explicações sobre como escrever novas páginas.

# Ver Também

apropos, man

# Name

help_from_sci — Geração de arquivos de ajuda e de arquivos de demonstração a partir da seção de comentários de cabeçalho de um arquivo-fonte .sci

```
help_from_sci() // gera um modelo de função vazio
help_from_sci(funname,helpdir) // gera helpdir/funname.xml a partir de fu
help_from_sci(dirname,helpdir) // processa dirname/*.sci e cria helpdir/*
help_from_sci(dirname,helpdir,helpdir) // como acima, mas também cria o a
[helptxt,demotxt]=help_from_sci(funname) // retorna funname.xml e funname
```

# Parâmetros

funname:
: o nome de um único arquivo .sci a ser processado

dirname:
: nome do diretório onde todos os arquivos .sci serão processados

helpdir:
: endereço opcional onde o arquivo de ajuda .xml será criado

demodir:
: endereço opcional onde os arquivos de demonstração .dem.sce serão criados baseados nos códigos da seção Examples

helptxt:
: retorna o código da ajuda XML se helpdir for vazio, ou o endereço para o novo arquivo .xml

demotxt:
: retorna o código de demonstração se demodir for vazio, ou o endereço para o novo arquivo .dem.sc

# Descrição

help_from_sci é uma versão revisada da função help_skeleton. Seu objetivo é gerar arquivos de ajuda .xml baseados na seção de comentários de cabeçalho dos arquivos-fontes .sci. Opcionalmente, os arquivos de demonstração .dem.sce podem ser gerados baseados nos códigos da seção Examples na seção de cometários de cabeçalho dos arquivos .sci.

Para que help_from_sci formate o arquivo .xml propriamente, os a seção de comentários de cabeçalho deve concordar com algumas regras simples de formatação.

A primeira linha de comentário seguinte à definição de função deve conter uma descrição breve da função.

Os comentários restantes são formatados de acordo com os seguintes cabeçalhos (opcionais): "Calling Sequence", "Parameters", "Description", "Examples", "See also", "Used functions", "Authors" e "Bibliography".

As seguintes diretrizes devem ser seguidas ao se escrever os comentários de código fonte:

- `Calling Sequence` - um exemplo por linha.
- `Parameters` - separe o nome do parâmetro e a descrição por um ":". Mantenha a descrição de cada parâmetro na mesma linha.
- `Description` - a formatação do texto pode ser feita utilizando comandos XML. Adicionar uma linha de comentário vazia na seção Description é interpretado como começo de um novo parágrafo.

- `See also` - liste um nome de função por linha.
- `Authors` - escreva um autor em cada linha após o cabeçalho Authors. Use ";" para separar os autores de qualquer informação adicional.
- `Bibliography` - escreva uma referência por linha seguindo o cabeçalho References.

# Exemplos

```
help_from_sci()   // abrindo um modelo de código-fonte vazio no editor.
// salve este modelo como test_fun.sci no diretório corrente antes de executar
// os próximos comandos do exemplo

help_from_sci('test_fun')        // retornando o esqueleto xml como um string d

help_from_sci('test_fun','.')    // criando o arquivo de ajuda xml no diretório

// criando ambos os arquivos de ajuda e demonstração no diretório corrente.
help_from_sci('test_fun','.','.')

// de um diretório raiz de um toolbox, uma seqüência de chamamento típica seria
// help_from_sci('macros','help\pt_BR','demos')
// este comando processaria todos os arquivos .sci no diretório de macros
// e utilizaria a seção de comentários de cabeçalho para atualizar as ajudas .x
// diretório help\en_US e reconstruiria os arquivos .dem.sce no diretório demos
```

# Ver Também

help, help_skeleton, xmltohtml

# Autor

T. Pettersen
torbjorn.pettersen@broadpark.no

# Name

help_skeleton — constrói o esqueleto do arquivo de ajuda xml associado à função Scilab

```
txt = help_skeleton(funname [,path [,language]])
```

# Parâmetros

funname
: string : o nome da função

path
: string : o endereço onde se criará o arquivo, se for requerido. Se este argumento não for fornecido, o esqueleto é retornado como um string.

language
: string :com valor possível "fr_FR" ou "en_US" o padrão é "en_US"

txt
: o código xml ou o endereço do arquivo xml completo

# Descrição

`txt = help_skeleton(funname)` gera um vetor de strings contendo o esqueleto do código XML descrevendo a ajuda da função `funname`.

`fullpath = help_skeleton(funname,dirpath)` gera o código XML descrevendo a ajuda da função `funname` em um arquivo nomeado `funname.xml` no diretório especificado pelo endereço `dirpath`. Neste caso, a função retorna o endereço do arquivo.

# Exemplos

```
function [y,z]=foo(a,b),y=a+b,z=1,endfunction
p=help_skeleton('foo',TMPDIR)
if (isdef('editor') | (funptr('editor')<>0)) then
  editor(p);
end
```

# Ver Também

help

# Autor

Serge Steer, INRIA

# Name

make_index — cria um novo arquivo de índice para ajuda on-line

```
make_index()
```

# Descrição

A ajuda on-line lê primeiro o arquivo `index.html` , que contém a lista dos capítulos. Este arquivo vem com o Scilab e está no diretório `SCIDIR/man/<language>` (ver man). É possível modificar este arquivo de índice enquanto se adiciona interativamente novos arquivos. Para isto, modifique a variável %helps e então utilize a função `make_index`.

# Ver Também

%helps, man

# Name

man — Descrição do formato do arquivo de ajuda XML

# Descrição

Os arquivos-fontes da ajuda Scilab são escritos em formato XML

Arquivos-fontes (com extensão .xml) podem ser encontrados nos diretórios `<SCIDIR>/man/<language>/*`. O nome do arquivo é geralmente associado à palavra-chave (correspondente a um nome de função na maioria dos casos) que descreve .

# Breves palavras sobre XML

Um arquivo XML lembra um arquivo HTML, mas possui uma sintaxe tanto mais rígida quanto mais livre. Livre porque você pode construir os seus próprios tags (marcação): o conjunto de tags, junto com suas regras, deve ser descrito em algum lugar, geralmente em outro arquivo (`<SCIDIR>/man/manrev.dtd` para o Scilab), e rígida porque, uma vez que os tags e regras são definidas (as quais são chamadas Definição de Tipo de Documento (Document Type Definition): DTD) , você deve respeitá-las (em particular, para todos os tags `<MY_TAG>` abertos, deve corresponder um tag fechado `</MY_TAG>`).

O DTD manrev.dtd é escrito em SGML e precisa a exata sintaxe requerida por uma página de ajuda XML do Scilab. Então se você conhece esta linguagem, você pode ler este arquivo. O seguinte exemplo anotado (veja a próxima seção) mostra algumas possibilidades oferecidas por este DTD e pode ser suficiente para escrever páginas de ajuda simples.

Uma vez que a página XML é escrita e está em conformidade com o DTD, ela pode ser transformada em um HTML para ser lida por algum navegador de internet, ou pelo navegador tcltk do Scilab (ver a seção escolha do navegador nesta página). A tradução XML -> HTML é controlada por um conjunto de regras escritas no arquivo (XML) `<SCIDIR>/man/language/html.xsl`. Estas regras estão correntemente mais ou menos restritas para se adequarem aos recursos do navegador tcltk do Scilab (que pode exibir corretamente apenas HTML básico): Se você utilizar um navegador de HTML e quiser melhorar a aparência, terá que modificar este arquivo.

# Como escrever uma página de ajuda XML do Scilab simples: a maneira lazzy

Caso se deseje escrever um arquivo XML associado a uma nova função Scilab, pode-se utilizar a função help_skeleton para produzir o esqueleto do arquivo XML. Na maior parte dos casos, o usuário não precisará saber a sintaxe XML.

# Como escrever uma página de ajuda XML do Scilab simples: um exemplo

Aqui está um exemplo simples de uma página de ajuda XML que descreve uma função do Scilab hipotética chamada "foo". No seguinte, o arquivo XML é exibido em uma `fonte de máquina-de-escrever` e recortado em várias partes, cada parte precedida por algumas explicações associadas. O arquivo XML inteiro `foo.xml` está no diretório `<SCIDIR>/man/eng/utility` e o resultado pode ser exibido clicando-se em foo (você pode encontrar outros exemplos no diretório `<SCIDIR>/examples/man-examples-xml`). **Finalmente** note que alguns pares de tags `<TAG>`, `</TAG>` foram renomeados aqui `<ATAG>`, `</ATAG>`. Isto porque alguns scripts do Scilab que realizam trabalhos sobre ou a partir dos arquivos XML não verificam se um tag está dentro de uma entrada `VERBATIM`.

As primeiras três linhas do arquivo são mandatórias, a segunda precisa o endereço para o arquivo DTD e a terceira, formada pelo tag `<MAN>`, inicia a descrição hierárquica (o arquivo deve terminar

com o tag `</MAN>`). As quatro entradas seguintes : `LANGUAGE`, `TITLE`, `TYPE` e `DATE`, também são mandatórias (nesta ordem), o texto correspondente a `<TYPE>` sendo geralmente 'função do Scilab' (na maior parte dos casos), mas pode ser simplesmente 'palavra-chave do Scilab' ou 'tipo de dados do Scilab', ..., dependendo do que explica a página de ajuda.

```
<!DOCTYPE MAN SYSTEM "<SCIDIR>/man/manrev.dtd">
<MAN>
   <LANGUAGE>eng</LANGUAGE>
   <TITLE>foo</TITLE>
   <TYPE>função do Scilab</TYPE>
   <DATE>$LastChangedDate$</DATE>
```

A primeira destas duas entradas seguintes (`SHORT_DESCRIPTION`) é mandatória e importante desde que as palavras do texto da descrição curta (short description), são utilizadas pelo comando apropos para buscar páginas de ajuda a partir de uma palavra-chave: a descrição curta é utilizada para construir o arquivo `whatis.html` correspondente ao seu toolbox e o comando `apropos keyword` faz uma pesquisa em todos os arquivos whatis e propõe os links para cada página onde a palavra `keyword` se encontra em sua descrição curta (na verdade, os tags reais associados são `<SHORT_DESCRIPTION>` e `</SHORT_DESCRIPTION>` e não `<ASHORT_DESCRIPTION>` e `</ASHORT_DESCRIPTION>`). A próxima entrada (`CALLING_SEQUENCE`) deve ser utilizada ao se descrever uma função (mas não é estritamente mandatória). Se a sua função possui várias seqüências de chamamento, utilize várias entradas `CALLING_SEQUENCE_ITEM`.

```
<ASHORT_DESCRIPTION name="foo">descrição curta de foo</ASHORT_DESCRIPTION>
<CALLING_SEQUENCE>
    <CALLING_SEQUENCE_ITEM>[y] = foo(x)</CALLING_SEQUENCE_ITEM>
    </CALLING_SEQUENCE>
```

A entrada seguinte (`PARAM`) não é estritamente mandatória, mas é a boa para descrever cada parâmetro (de entrada e saída) no caso de uma função.

```
<PARAM>
   <PARAM_INDENT>
      <PARAM_ITEM>
         <PARAM_NAME>x</PARAM_NAME>
         <PARAM_DESCRIPTION>
            <SP>: o que pode ser x</SP>
         </PARAM_DESCRIPTION>
      </PARAM_ITEM>
      <PARAM_ITEM>
         <PARAM_NAME>y</PARAM_NAME>
         <PARAM_DESCRIPTION>
            <SP>: o que pode ser y</SP>
         </PARAM_DESCRIPTION>
      </PARAM_ITEM>
   </PARAM_INDENT>
</PARAM>
```

A entrada `DESCRIPTION` é talvez a mais significativa (mas não é estritamente mandatória) e pode ser mais sofisticada que neste exemplo (por exemplo, você pode ter sub-entradas `DESCRIPTION_ITEM`). Aqui, você vê como escrever vários parágrafos (cada um encerrado pelos tags `<P>` e `</P>`), como enfatizar uma variável, ou um nome de função (encerrando-os por tags `<VERB>` e `</VERB>`), como enfatizar uma parte o texto (`<EM>` ou `<BD>` e `<TT>` para colocar em

fonte de máquina de escrever)), e, finalmente, como colocar um link para outra página de ajuda (na verdade, os tags associados são `<LINK>` e `</LINK>` e não `<ALINK>` e `</ALINK>`).

```
<DESCRIPTION>
    <P>
       Um primeiro parágrafo explica o que a função foo computa.
       Se você deseja enfatizar um nome de parâmetro, você pode utilizar o segu
       tag <VERB>x</VERB>, se você desejar enfatizar parte do texto
       <EM>encerre-a nestes tags</EM> e use estes
       <BD>para uma fonte em negrito</BD> e finalmente <TT>para um estilo em má
    </P>
    <P>
       Um segundo parágrafo... Aqui vai um exemplo de um link para outra página
       <ALINK>man</ALINK>.
    </P>
</DESCRIPTION>
```

Aqui está como escrever a sua própria entrada, por exemplo, para escrever observações e/ou notas sobre a sua maravilhosa função.

```
<SECTION label='Notes'>
    <P>
       Aqui está uma lista de notas :
    </P>
    <ITEM label='first'><SP>blablabla...</SP></ITEM>
    <ITEM label='second'><SP>toto é o francês de foo...</SP></ITEM>
</SECTION>
```

Uma entrada importante é `EXAMPLE` que é reservada para mostrar usos no Scilab da sua função (comece com exemplos simples!). Note que você deve fechar esta entrada com `]]></EXAMPLE>` , e não como aqui com `}}></EXAMPLE>` (mais uma vez, este é um truque ruim para evitar problemas de interpretação).

```
<EXAMPLE><![CDATA[
    deff("y=foo(x)","y=x"); // definindo a função foo como a função identidade
    foo("toto")
}}></EXAMPLE>
```

Esta última parte explica como colocar links para outras páginas de ajuda relacionadas (como já foi dito antes, os bons tags são `<LINK>` e `</LINK>` e não `<ALINK>` e `</ALINK>` ) e finalmente como revelar o seu nome, se você quiser (utilize uma entrada `AUTHOR_ITEM` por autor). Talvez seja uma boa idéia colocar um endereço de email, se você procura por relatos de bugs !

```
<SEE_ALSO>
  <SEE_ALSO_ITEM> <ALINK>man</ALINK> </SEE_ALSO_ITEM>
  <SEE_ALSO_ITEM> <ALINK>apropos</ALINK> </SEE_ALSO_ITEM>
</SEE_ALSO>
 <AUTHOR>
  <AUTHOR_ITEM>B. P.</AUTHOR_ITEM>
</AUTHOR>
</MAN>
```

# Como criar um capítulo de ajuda

Crie um diretório e escreva um conjunto de arquivos XML construídos como descrito acima. Então, inicie o Scilab e execute `xmltohtml(dir)`, onde é um `dir` string fornecendo o endereço do diretório (ver xmltohtml para mais detalhes) .

# Como fazer o Scilab reconhecer um novo capítulo de ajuda

Isto pode ser feito através da função add_help_chapter.

# Exemplos

```
function y=foo(a,b,c),y=a+2*b+c,endfunction
path=help_skeleton('foo',TMPDIR)
if (isdef('editor') | (funptr('editor')<>0)) then
  editor(path)
end
```

# Ver Também

apropos, help, help_skeleton

# Name

manedit — editando item do manual

```
manedit(manitem)
```

# Parâmetros

manitem
: string (geralmente, o nome de uma função)

# Descrição

`edit(manitem)` abre o arquivo xml associado a `manitem` no editor.

Se não há arquivo xml associado a `manitem` e `manitem` é o nome de uma função Scilab, o xpad abre com o esqueleto do arquivo xml produzido por help_skeleton. Este arquivo está localizado em TMPDIR.

# Exemplos

```
manedit('manedit')

function [x,y,z]=foo123(a,b),
x=a+b,y=a-b,z=a==b
endfunction
manedit foo123
```

# Ver Também

help, help_skeleton

# Name

%helps — variável definindo o endereço dos diretórios de ajuda

# Descrição

A variável global `%helps` é uma matriz N x 2 de strings. A k-ésima linha de `%helps`, `%helps(k,:)` representa o k-ésimo capítulo do manual e é feita de dois strings:

`%helps(k,1)` é o nome de endereço absoluto de um diretório.

`%helps(k,2)` é um título para este diretório. Por exemplo, para k=2, nós temos o capítulo de gráficos `%helps(2,:)`.

A variável `%helps` é definida no arquivo de inicialização do Scilab `SCI+"/scilab.start"`.

Para adicionar um novo diretório de ajuda, o usuário deve adicionar uma linha a `%helps`. (Uma linha para cada diretório).

Por exemplo, `%helps=[%helps; "Path-Of-My-Help-Dir","My-Title"];` habilita o navegador de ajuda do Scilab a procurar por itens do manual de ajuda no diretório com endereço "Path-Of-My-Help-Dir".

"My-Title" é, então, o título do novo capítulo de ajuda.

Um diretório válido deve conter:

1- Um conjunto de arquivos `.html` (ex.: `item1.html, item2.html` etc). Os arquivos `.html` são geralmente construídos de arquivos XML.

2- Um arquivo `whatis.html`, que deve conter um formato especial. Cada linha de `whatis` deve ser como segue:

```
<BR><A HREF="item.html">item</A> - rápida descrição
```

`item` é o item da ajuda, i.e. o comando `help item` exibe o conteúdo do arquivo `item.html`.

O comando `apropos keyword` retorna as linhas de todos os arquivos `whatis.html` nos quais a palavra-chave keyword aparece.

Em plataformas Linux, o Scilab provê um Makefile para transformar páginas `.xml` pages em páginas `.html` (ver SCIDIR/examples/man-examples).

# Ver Também

apropos, help, man

# Name

xmltohtml — converte arquivos de ajuda xml do Scilab para formato HTML

```
xmltohtml(dirs [,titles [,dir_language [default_language]]]])
```

# Parâmetros

dirs
: vetor de strings: um conjunto de endereços de diretórios para os quais os manuais html devem ser gerados ou `[]`

titles
: vetor de strings: títulos associados a endereços de diretórios ou []

dir_language
: vetor de strings: idiomas associados a endereços de diretórios ou []

default_language
: vetor de strings: idiomas padrões associados aos endereços de diretórios ou []. Se um arquivo XML estiver faltando em dir_language, ele é copiado de default_language.

# Descrição

Converte arquivos de ajuda Scilab contidos em um conjunto de diretórios para formato HTML.

# Exemplos

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Construindo a ajuda em francês
// ==============================================================================
my_french_help_dir    = my_module_path+'/help/fr_FR';
my_french_help_title  = 'Example 1 [fr_FR]';
my_french_html_dir    = xmltohtml(my_french_help_dir,my_french_help_title,'fr_F

// Construindo a ajuda em inglês
// ==============================================================================
my_english_help_dir   = my_module_path+'/help/en_US';
my_english_help_title = 'Example 1 [en_US]';
my_english_html_dir   = xmltohtml(my_english_help_dir,my_english_help_title,'en_
```

```
// Construindo a ajuda em chinês
// ==========================================================================
my_chinese_help_dir   = my_module_path+'/help/zh_TW';
my_chinese_help_title = 'Example 1 [zh_TW]';
my_chinese_html_dir   = xmltohtml(my_chinese_help_dir,my_chinese_help_title,'zh
```

# Ver Também

help, add_help_chapter

# Name

xmltojar — converte arquivos de ajuda xml do Scilab para formato javaHelp

```
xmltojar(dirs [,titles [,dir_language [default_language]]]])
```

# Parâmetros

dirs
: vetor de strings: um conjunto de endereços de diretórios para os quais os manuais html devem ser gerados ou `[]`

titles
: vetor de strings: títulos associados a endereços de diretórios ou []

dir_language
: vetor de strings: idiomas associados a endereços de diretórios ou []

default_language
: vetor de strings: idiomas padrões associados aos endereços de diretórios ou []. Se um arquivo XML estiver faltando em dir_language, ele é copiado de default_language.

# Descrição

Converte arquivos de ajuda Scilab contidos em um conjunto de diretórios para formato jar.

# Exemplos

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Construindo a ajuda em francês
// ==============================================================================
my_french_help_dir    =  my_module_path+'/help/fr_FR';
my_french_help_title  =  'Example 1 [fr_FR]';
xmltojar(my_french_help_dir,my_french_help_title,'fr_FR');

// Construindo a ajuda em inglês
// ==============================================================================
my_english_help_dir   =  my_module_path+'/help/en_US';
my_english_help_title =  'Example 1 [en_US]';
xmltojar(my_english_help_dir,my_english_help_title,'en_US');
```

```
// Construindo a ajuda em chinês
// ==============================================================================
my_chinese_help_dir   =  my_module_path+'/help/zh_TW';
my_chinese_help_title =  'Example 1 [zh_TW]';
xmltojar(my_chinese_help_dir,my_chinese_help_title,'zh_TW');

// Adicionando capítulos de ajuda em francês, inglês e chinês
// ==============================================================================

if getlanguage() == 'fr_FR' then
  add_help_chapter(my_french_help_title,my_module_path+"/jar");

elseif getlanguage() == 'zh_TW' then
  add_help_chapter(my_chinese_help_title,my_module_path+"/jar");

else
  add_help_chapter(my_english_help_title,my_module_path+"/jar");
end

// Ver o resultado no navegador de ajuda
// ==============================================================================
help();

// Deletando capítulos de ajuda em francês e inglês
// ==============================================================================
if getlanguage() == 'fr_FR' then
  del_help_chapter(my_french_help_title);
else
  del_help_chapter(my_english_help_title);
end
```

# Ver Também

help, add_help_chapter

# Name

xmltopdf — converte arquivos de ajuda xml do Scilab para formato PDF

```
xmltopdf(dirs [,titles [,dir_language [default_language]]]])
```

# Parâmetros

dirs
: vetor de strings: um conjunto de endereços de diretórios para os quais os manuais pdf devem ser gerados ou `[]`

titles
: vetor de strings: títulos associados a endereços de diretórios ou []

dir_language
: vetor de strings: idiomas associados a endereços de diretórios ou []

default_language
: vetor de strings: idiomas padrões associados aos endereços de diretórios ou []. Se um arquivo XML estiver faltando em dir_language, ele é copiado de default_language.

# Descrição

Converte arquivos de ajuda Scilab contidos em um conjunto de diretórios para formato PDF

# Exemplos

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Construindo a ajuda em francês
// ==============================================================================
my_french_help_dir    = my_module_path+'/help/fr_FR';
my_french_help_title  = 'Example 1 [fr_FR]';
my_french_pdf         = xmltopdf(my_french_help_dir,my_french_help_title,'fr_FR

// Construindo a ajuda em inglês
// ==============================================================================
my_english_help_dir   = my_module_path+'/help/en_US';
my_english_help_title = 'Example 1 [en_US]';
my_english_pdf        = xmltopdf(my_english_help_dir,my_english_help_title,'en_
```

```
// Construindo a ajuda em chinês
// ==========================================================================
my_chinese_help_dir   = my_module_path+'/help/zh_TW';
my_chinese_help_title = 'Example 1 [zh_TW]';
my_chinese_pdf        = xmltopdf(my_chinese_help_dir,my_chinese_help_title,'zh_
```

# Ver Também

help, add_help_chapter

# Name

xmltops — converte arquivos de ajuda xml do Scilab para formato Postscript

```
xmltops(dirs [,titles [,dir_language [default_language]]]])
```

# Parâmetros

dirs
: vetor de strings: um conjunto de endereços de diretórios para os quais os manuais postscript devem ser gerados ou `[]`

titles
: vetor de strings: títulos associados a endereços de diretórios ou []

dir_language
: vetor de strings: idiomas associados a endereços de diretórios ou []

default_language
: vetor de strings: idiomas padrões associados aos endereços de diretórios ou []. Se um arquivo XML estiver faltando em dir_language, ele é copiado de default_language.

# Descrição

Converte arquivos de ajuda Scilab contidos em um conjunto de diretórios para formato PS.

# Exemplos

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Construindo a ajuda em francês
// ==============================================================================
my_french_help_dir    = my_module_path+'/help/fr_FR';
my_french_help_title  = 'Example 1 [fr_FR]';
my_french_ps          = xmltops(my_french_help_dir,my_french_help_title,'fr_FR'

// Construindo a ajuda em inglês
// ==============================================================================
my_english_help_dir   = my_module_path+'/help/en_US';
my_english_help_title = 'Example 1 [EN_US]';
my_english_ps         = xmltops(my_english_help_dir,my_english_help_title,'en_US
```

```
// Construindo a ajuda em chinês
// ==========================================================================
my_chinese_help_dir   = my_module_path+'/help/zh_TW';
my_chinese_help_title = 'Example 1 [zh_TW]';
my_chinese_ps         = xmltops(my_chinese_help_dir,my_chinese_help_title,'zh_TW
```

# Ver Também

help, add_help_chapter

# Name

del_help_chapter — Delete an entry in the helps list

```
del_help_chapter(title[,mode])
```

# Parameters

title
: String array, Help chapters title

mode
: A boolean, if TRUE, the chapter is considered as belongs to a internal modules, otherwise, the chapter is added as external module. It's an optional input argument and its default value is %F.

# Description

This function deletes an entry in the helps list.

# See Also

help , add_help_chapter

# Name

xmltochm — converts xml Scilab help files to Microsoft Compressed HTML format (Windows)

```
xmltochm(dirs [,titles [,dir_language [default_language]]]])
```

# Parameters

dirs
: vector of strings: a set of directory paths for which html manuals are to be generated or `[]`

titles
: vector of strings: titles associated to directory paths or []

dir_language
: vector of strings: languages associated to directory paths or []

default_language
: vector of strings: default languages associated to directory paths or []. If an XML file is missing in the dir_language, it's copied from the default_language.

# Description

converts xml Scilab help files contained in a set of directories into chm files.

Microsoft HTML Help Downloads (Windows) [http://msdn.microsoft.com/en-us/library/ms669985(VS.85).aspx]

# Examples

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Build the french help
// ==============================================================================
my_french_help_dir    =  my_module_path+'/help/fr_FR';
my_french_help_title  =  'Example 1 [fr_FR]';
res = xmltochm(my_french_help_dir,my_french_help_title,'fr_FR');
if MSDOS then
 dos('start ' + res);
end
```

```
// Build the english help
// ==============================================================================
my_english_help_dir   =  my_module_path+'/help/en_US';
my_english_help_title =  'Example 1 [en_US]';
res = xmltochm(my_english_help_dir,my_english_help_title,'en_US');
if MSDOS then
 dos('start ' + res);
end

// Build the chinese help
// ==============================================================================
my_chinese_help_dir   =  my_module_path+'/help/zh_TW';
my_chinese_help_title =  'Example 1 [zh_TW]';
res = xmltochm(my_chinese_help_dir,my_chinese_help_title,'zh_TW');
if MSDOS then
 dos('start ' + res);
end
```

# See Also

help , add_help_chapter