

LEARNING SCILAB BY DOING

Philippe CASTAGLIOLA

25th July 2004

Scilab is a scientific software package for numerical computations providing a powerful open computing environment for engineering and scientific applications. It is developed since 1990 by researchers from INRIA and ENPC. Distributed freely via the Internet since 1994, Scilab is currently being used in educational and industrial environments around the world. Scilab includes hundreds of mathematical functions with the possibility to add interactively programs from various languages (C, Fortran...). It has sophisticated data structures (including lists, polynomials, rational functions, linear systems...), an interpreter and a high level programming language. Scilab has been conceived to be an open system where the user can define new data types and operations on these data types by using overloading. This scientific software will be intensively used during the course “Statistical Process Control & Design of Experiments”.

The best way for you learning Scilab is to execute step by step all the instructions below and try to understand what is the meaning of these instructions. Do not go step n until step $n - 1$ is not understood. Do not hesitate to ask the teacher if you do not understand something.

Have fun!

1 Basics

Creates the file `session.log` which contains a copy of the current Scilab session

```
-->diary("session.log")
```

Create two (2,3) matrices `x` and `y`


```
-->x=[7.05,7.61,6.73;7.12,7.37,7.04]
```

```
-->y=[7.63,6.92,7.34;7.51,7.55,7.53]
```

Some usefull keyboard tricks



 for *left*,



 for *right*,

 for *previous command*,

 for *next command*,

  for *begining of the line*,

  for *end of the line*,

  for *erase end of line*,

  for *erase the whole line*.

Create a (2,2) matrix **z**. The ; at the end means *do not print the result*

```
-->z=[4.91,5.19;5.16,5.33];  
-->z
```

Two basic ways for concatening matrices

```
-->[x,y]  
-->[x;y]
```

A more complex example of concatenation

```
-->[x,[0;0];y,[0;0];0,0,0,1]
```

Matrix (3,4) of ones, and matrix (2,5) of zeros

```
-->ones(3,4)  
-->zeros(2,5)
```

Matrix of ones and matrix of zeros having the same size as **x**

```
-->ones(x)  
-->zeros(x)
```

The ' means *transpose*

```
-->x'
```

Add and substract two matrices having the same size

-->x+y

-->x-y

Add 3 and subtract 5 to every entries of matrix x

-->x+3

-->x-5

Multiply matrix z by matrix x

-->z*x

Multiply matrix x by 3

-->3*x

Element-wise multiplication and division of matrices

-->x.*y

-->x./y

Divide 5 by every entries of matrix y. Caution : the () each side of 5 are mandatory in order to avoid confusion with (5.)/y

-->(5)./y

Matrix power operator

-->z^3

Element-wise power operator

-->x.^3

Guess what ?

-->2^x

Kronecker product

-->x.*.[1,2;3,4]

How to replicate vector [1,2,3] 10 times

-->ones(10,1)*[1,2,3]

-->[1,2,3] '*ones(1,10)

How to replicate matrix z 4 times using the Kronecker product

```
-->z.*ones(1,4)
-->ones(1,4).*z
```

Boolean operator <=. T is True, F is False

```
-->x<=y
```

Another example. Boolean operators are <, <=, >, >=, <> (or ~=)

```
-->x>7
```

Logical “or” and “and”

```
-->(x<=7.1)|(7.3<=x)
-->(7.1<=x)&(x<=7.3)
```

Logical “not”

```
-->~(x>7)
```

Find entries satisfying a condition

```
-->find((x<=7.1)|(7.3<=x))
-->find((7.1<=x)&(x<=7.3))
```

Matrix oriented “or” and “and”

```
-->or(x>7)
-->and(x>7)
```

Determinant, inverse, trace

```
-->det(z)
-->inv(z)
-->trace(z)
```

d is the eigenvalue matrix of matrix z. v is the eigenvector matrix of matrix z

```
-->[d,v]=bdiag(z)
-->v*d*inv(v)
```

Element-wise exponential, logarithm, square-root ... Also available cos, sin, tan, acos, asin, atan, cosh, sinh, tanh, acosh, asinh, atanh ...

```
-->exp(z)
-->log(x)
-->sqrt(x)
-->cos(x)
```

Matrix oriented exponential and logarithm

```
-->expm(z)
-->logm(z)
```

Real numbers into integers

```
-->ceil(x/3)
-->ceil(-x/3)
-->floor(x/3)
-->floor(-x/3)
-->round(x/3)
-->round(-x/3)
```

Identity matrix

```
-->eye(3,3)
-->eye(z)
```

Give the size of matrix **x**. "**r**" is for row, and "**c**" is for column

```
-->[r,c]=size(x);[r,c]
-->size(x,"c")
-->size(x,"r")
```

Total number of entries in matrix **x**

```
-->size(x,"*")
```

Sum of every entry of matrix **y**

```
-->sum(y)
```

Row sum. The result is in a column ("**c**") vector

```
-->sum(y,"c")
```

Column sum. The result is in a row ("**r**") vector

```
-->sum(y,"r")
```

Same for the product

```
-->prod(y)
-->prod(y,"c")
-->prod(y,"r")
```

Same for the cumulative sum and cumulative product

```
-->cumsum(y)
-->cumsum(y,"c")
-->cumprod(y)
-->cumprod(y,"r")
```

Same for the minimum and maximum

```
-->min(y)
-->min(y,"c")
-->max(y)
-->max(y,"r")
```

The row vector $[1, 2, 3, 4, 5]$

```
-->1:5
```

The column vector $[1; 1.5; \dots; 4.5; 5]$

```
-->(1:0.5:5)'
```

The row vector $[5, 4.5, \dots, 1.5, 1]$

```
-->5:-0.5:1
```

A row vector having 1 as first entry, 5 as last entry and 9 entries at all, i.e.
 $[1, 1.5, \dots, 4.5, 5]$

```
-->linspace(1,5,9)
```

Extract rows 1,2 and column 1,3 of matrix **x**

```
-->x([1,2],[1,3])
-->x(1:2,[1,3])
```

Extract every rows and column 1,3 of matrix **x**

```
-->x(:, [1,3])
```

Extract 1st row of matrix **x**

```
-->x(1,:)
```

Guess what ?

```
-->x(2:-1:1,3:-1:1)
```

Matrices are linearly row coded. Try this

```
-->x(1)
-->x(2)
-->x(6)
```

Transform matrix **x** in a (6,1) column vector using the row order. The **x'** is used in order to obtain a column order

```
-->matrix(x,6,1)
-->matrix(x',6,1)
```

Descending sort

```
-->sort(x)
```

sx contains matrix **x** sorted and **i** is the index of the sort

```
-->[sx,i]=sort(x)
```

A way of making an ascending sort

```
-->-sort(-x)
```

Sort by row and column

```
-->sort(x,"c")
-->sort(x,"r")
```

2 Inputs/Outputs

Simple display of matrix **x**

```
-->disp(x)
```

Write matrix **x** using a C type format

```
-->mprintf("%4.2f %5.3f %12.4e\n",x)
```

Write matrix **x** using a FORTRAN type format

```
-->write(%io(2),x,"(f4.2,1x,f5.3,1x,e12.4)")
```

Write matrix **x** into file **x.dat** using a C type format

```
-->f=fopen("x.dat","w");
-->fprintf(f,"%4.2f %5.3f %12.4e\n",x)
-->fclose(f);
```

Write matrix `x` into file `x.dat` using a FORTRAN type format

```
-->f=fopen("x.dat","w");
-->fprintf(f,"%4.2f %5.3f %12.4e\n",x)
-->fclose(f);
```

Read matrix from file `x.dat`

```
-->read("x.dat",2,3)
```

3 Graphics

Define abscissae

```
-->u=linspace(-%pi,%pi,100)';
```

Compute curves $\cos(u)$ and $2\sin(4u)$

```
-->yc=cos(u);
-->ys=2*sin(4*u);
```

Clear graphic screen

```
-->xbasc()
```

Plot two curves. `-9` and `-8` are dot styles and `2` is a line style. `xselect()` puts the graphic window on top

```
-->plot2d([u,u,u],[yc,ys,ys],[-9,2,-8]);xselect()
```

Another example. Here the `frameflag=0` means “keep” previous coordinates

```
-->xbasc()
-->plot2d(u,yc,-9)
-->plot2d([u,u],[ys,ys],[2,-8],frameflag=0);xselect()
```

Another example. Here the `rect=[-2,-1,2,1]` means “use defined coordinate rectangle”

```
-->xbasc()
-->plot2d([u,u,u],[yc,ys,ys],[-9,2,-8],rect=[-2,-1,2,1]);xselect()
```


If you want to save the plot, use the “File” menu of the plot window. Then you can

- use the “Export” sub-menu. The file can be saved as a postscript, xfig or gif file.
- for MS Windows version, you can also use the “Copy to clipboard” sub-menu.

4 Programming

Open a text processor (like xemacs), and type the following macro. Remark: this macro is not optimized at all and it is created only for pedagogic purpose. At the same time you type it, try to understand what this macro is supposed to do. The // are used for comments

```
function [s,p,d]=myfun1(x,y)
//
[argout,argin]=argn()
if argin~=2
    error("incorrect number of arguments")
end
[rx,cx]=size(x)
if cx~=1
    error("1st argument x must be a column vector")
end
[ry,cy]=size(y)
if cy~=1
    error("2nd argument y must be a column vector")
end
if rx~=ry
    error("1st argument x and 2nd argument y must have the same size")
end
// For loop
for i=1:rx
    s(i)=x(i)+y(i)
end
// While loop
j=1
while j<=rx
    p(j)=x(j)*y(j)
```

```

        j=j+1
    end
    // While loop with break
    k=1
    while %t
        d(k)=x(k)/y(k)
        if k==rx
            break
        else
            k=k+1
        end
    end
end

```

Save this macro in a file (I assume this file is `c:/tmp/myfun1.sci`). The macro `myfun1` can be loaded by typing

```
-->getf("c:/tmp/myfun1.sci")
```

Now you can use this new (powerfull :)) macro. For example

```
[s,p,d]=myfun1([1;2;3],[2;3;4])
```

Type this second macro `myfun2`. I assume you save it in the file `c:/tmp/myfun2.sci`. Again, this macro is useless and only created for pedagogic purpose.

```

function y=myfun2(x,t1,t2)
//
[argout,argin]=argn()
if (argin<2)|(argin>3)
    error("incorrect number of arguments")
end
// if elseif structure
if argin==2
    t2="sum"
end
if t1=="exp"
    x=exp(x)
elseif t1=="log"
    x=log(x)
elseif t1=="sqrt"
    x=sqrt(x)
else
    error("2nd argument t1 must be ""exp"", ""log"" or ""sqrt""")
end

```

```

end
// select structure
select t2
  case "sum"
    y=sum(x)
  case "prod"
    y=prod(x)
  case "mean"
    y=mean(x)
  case "std"
    y=std(x)
  else
    error("3rd argument t2 must be ""sum"", ""prod"", ""mean"" or ""std""")
end

```

Load this macro

```
-->getf("c:/tmp/myfun2.sci")
```

And try it. For example

```

-->myfun2(1:0.1:2,"log")
-->myfun2(1:0.1:2,"sqrt","mean")

```

5 Misc

How to get help on commands or functions (`sum` in this case)

```
-->help sum
```

If you want more help, use the “Help” menu on the main Scilab window and select “Help Dialog”.