scilab enterprises

Scilab for very beginners

This document has been co-written by Scilab Enterprises and Christine Gomez, mathematics teacher at Lycée Descartes (Descartes High School) in Antony, Hauts-de-Seine (France).

Table of content

Introduction

About this document

The purpose of this document is to guide you step by step in exploring the various basic features of Scilab for a user who has never used numerical computation software. This presentation is voluntarily limited to the essential to allow easier handling of Scilab.

Computations, graphs and illustrations are made with Scilab 5.4.0. You can reproduce all those commands from this version.

Install Scilab

Scilab is numerical computation software that anybody can freely download. Available under Windows, Linux and Mac OS X, Scilab can be downloaded at the following address: http://www.scilab.org/

You can be notified of new releases of Scilab software by subscribing to our channel notification at the following address: http://lists.scilab.org/mailman/listinfo/release

Mailing list

To facilitate the exchange between Scilab users, dedicated mailing lists exist (list in French, list for the education world, international list in English). The principle is simple: registrants can communicate with each other by e-mail (questions, answers, sharing of documents, feedbacks...).

To browse the available lists and to subscribe, go to the following address:
http://www.scilab.org/communities/user_zone/mailing_list

Complementary resources

If you have an Internet connection, we invite you to visit Scilab website where you will find a section dedicated to the use of Scilab (http://www.scilab.org/support/documentation), with links and relevant documents which can be freely downloaded and printed.

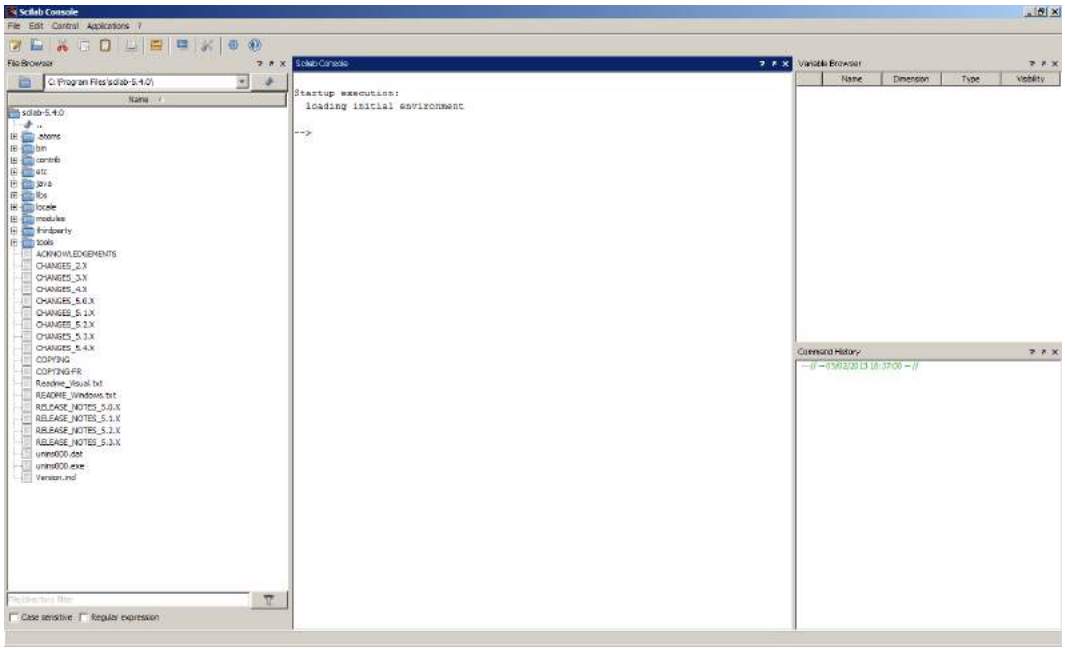
Chapter 1 – Become familiar with Scilab

The useful workspace in Scilab consists of several windows:

- The console for making calculations,
- The editor for writing programs,
- The graphics windows for displaying graphics,
- The embedded help.

The general environment and the console

After double-clicking the icon to launch Scilab, Scilab environment by default consists of the following docked windows – console, files and variables browsers, command history (see “Windows management and workspace customization”, page 11):



In the console after the prompt “ **-->** “, just type a command and press the Enter key (Windows and Linux) or Return key (Mac OS X) on the keyboard to obtain the corresponding result.

```
--> 57/4
 ans  =
    14.25
--> (2+9)^5
 ans  =
    161051.
```

Mention
Before the result, **ans** is displayed for “answer”.

It is possible to come back at any moment with the keyboard's arrow keys ← ↑ → ↓ or with the mouse. The left and right keys are used to change the instructions and the up and down keys are used to come back on a previously executed command.

Simple numerical calculations

All computations done with Scilab are numerical. Scilab performs computations with matrices (see chapter 2, page 23).

Operations are written with " **+** " for addition, " **–** " for subtraction, " ***** " for multiplication, " **/** " for division, " **^** " for exponents. For example:

```
-->2+3.4
 ans  =
    5.4
```

The case is sensitive. It is thus necessary to respect uppercase and lowercase for the calculations to be performed properly. For example, with **sqrt** command (which calculates the square root):

```
-->sqrt(9)
 ans  =
    3.
```

while:

```
-->SQRT(9)
         !--error 4
Undefined variable: SQRT
```

Particular numbers

%e and **%pi** represent respectively e and π:

```
--> %e
 %e  =
    2.7182818
```

```
--> %pi
 %pi  =
    3.1415927
```

%i represents the **i** of complexes in input and is displayed **i** in output:

```
--> 2+3*%i
 ans  =
    2. + 3.i
```

For not displaying the results

In adding a semi colon " **;** " at the end of a command line, the calculation is done but the result is not displayed.

```
-->(1+sqrt(5))/2;
```

```
--> (1+sqrt(5))/2
 ans  =
    1.618034
```

To remind the name of a function

The names of commonly used functions are summarized in Chapter 3 of this document (page 32). For example:

```
--> exp(10)/factorial(10)
 ans  =
    0.0060699
```

Mention
All available functions are also listed in the embedded help by clicking in the menu bar on **? > Scilab Help**.

The tab key →| on the keyboard can be used to complete the name of a function or a variable by giving its first few letters.

For example, after typing in the console the command:

```
-->fact
```

and then pressing the tab key, a window is displayed with all the functions and variables names beginning with **`fact`**, such as **`factorial`** and **`factor`**. Just double click on the required function or select it with the mouse or with the keys ↑ ↓ and press Enter (Windows and Linux) or Return (Mac OS X) to insert it in the command line.

The menu bar

The menus listed below are particularly useful.

Applications

- The command history allows you to find all the commands from previous sessions to the current session.
- The variables browser allows you to find all variables previously used during the current session.

Edit

Preferences (in **Scilab** menu under Mac OS X) allows you to set and customize colors, fonts and font size in the console and in the editor, which is very useful for screen projection.

Clicking on **Clear Console** clears the entire content of the console. In this case, the command history is still available and calculations made during the session remain in memory. Commands that have been erased are still available through the keyboard's arrow keys.

Control

To interrupt a running program, you can:

- Type **`pause`** in the program or click on **Control> Interrupt** in the menu bar (Ctrl X under Windows and Linux or Command X under Mac OS X), if the program is already running. In all cases, the prompt " **`-->`** " will turn into " **`-1->`** ", then into " **`-2->`** "…, if the operation is repeated.
- To return to the time prior to the program interruption, type **`resume`** in the console or click on **Control > Resume**.
- To quit for good a calculation without any possibility of return, type **`abort`** in the console or click on **Control > Abort** in the menu bar.

The editor

Typing directly into the console has two disadvantages: it is not possible to save the commands and it is not easy to edit multiple lines of instruction. The editor is the appropriate tool to run multiple instructions.

Opening the editor

To open the editor from the console, click on the first icon in the toolbar  or on **Applications > SciNotes** in the menu bar.

The editor opens with a default file named " **`Untitled 1`** ".

Writing in the editor

Typing in the editor is like as in any word processor.

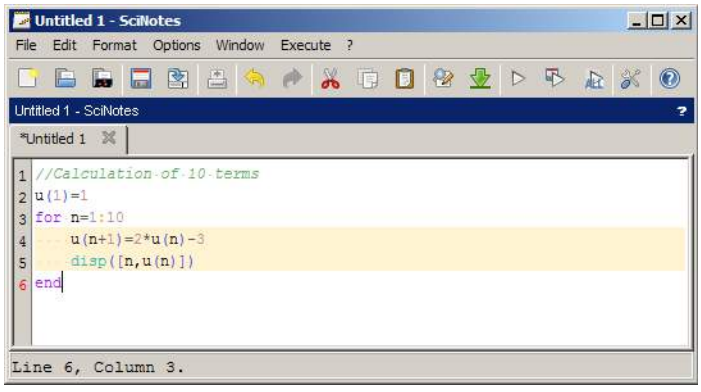
In the text editor, opening and closing parentheses, end loops, function and test commands are added automatically. However, these features can be disabled in **Options > Auto-completion on** menu, in clicking on the two below entries enabled by default:

- **(,[,…**
- **if,function,…**

While in principle each instruction should be entered on a separate line, it is possible to type multiple statements on a same line separating each statement with a semicolon " **`;`** ".

A space at the start of the line called indentation is automatic when a loop or a test is started.

In the following example, we calculate 10 terms of the sequence (u_n) defined by:
$\begin{cases} u_1 = 1 \\ u_{n+1} = 2u_n - 3 \end{cases}$



Mention
- Comments preceded with " **`//`** " will not be taken into account in the calculations.
- To change the font, click on **Options > Preferences**.
- When writing a program, indentation is automatic. If this is not the case, click on **Format > Correct indentation** to restore it (Ctrl I under Windows and Linux or Command I under Mac OS X).

Saving

Any file can be saved by clicking on **File > Save as**.

The extension " .sce " at the end of a file name will launch automatically Scilab when opening it (except under Linux and Mac OS X).

Copying into the console, executing a program

In clicking on Execute in the menu bar, three options are available:

- Execute " **…file with no echo** " (Ctrl Shift E under Windows and Linux, Cmd Shift E under Mac OS X): the file is executed without writing the program in the console (saving the file first is mandatory).
- Execute **" … file with echo** " (Ctrl L under Windows and Linux, Cmd L under Mac OS X): rewrite the file into the console and executes it.
- Execute " **…until the caret, with echo** " (Ctrl E under Windows and Linux, Cmd E under Mac OS X): rewrite the selection chosen with the mouse into the console and executes it or execute the file data until the caret position defined by the user.

Standard copy/paste can also be used.

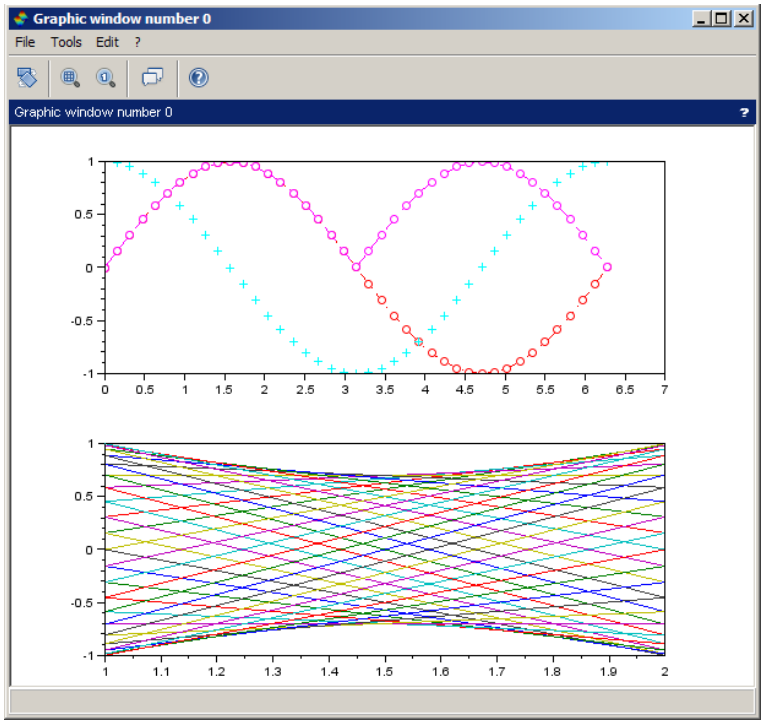
The graphics window

Opening a graphics window

A graphics window opens automatically when making any plot. It is possible to plot curves, surfaces, sequences of points (see chapter 2, page 18).

To obtain an example of curve, type in the console:

```
-->plot
```



Mention
- To erase a previous plot, type **`clf`**
(" clear figure ").
- To open another graphics window, type **`scf;`** (" set current figure ").
If several graphic windows are open, you can choose in which the plot will be drawn by typing **`scf(n)`**; n for the graphics window number (mentioned on the top left).

Modifying a plot

The magnifying glass allows zooming. To zoom in two dimensions, click on the tool and with the mouse create a rectangle which will constitute the new enlarged view. To zoom in three dimensions, click on the tool and create a parallelepiped which will constitute the new enlarged view. It is also possible to zoom in using the mouse wheel. To return to the initial screen, click on the other magnifying glass.

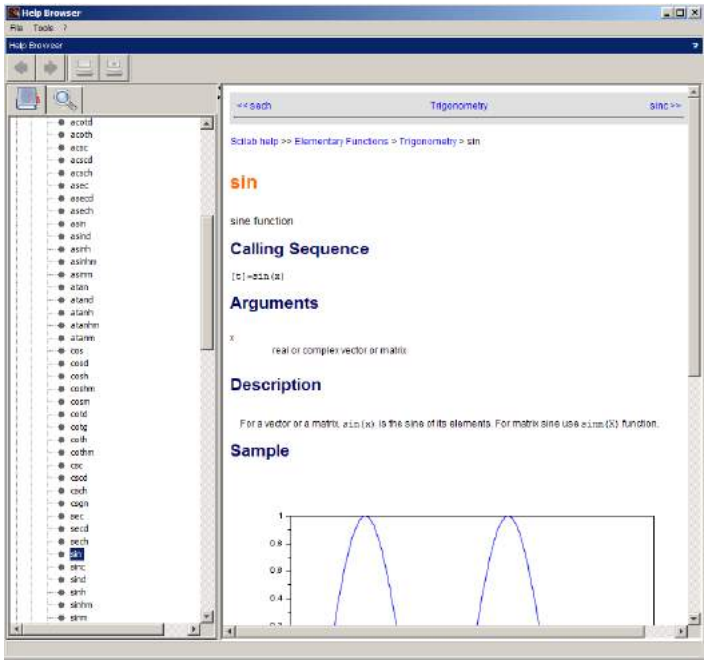
The icon enables rotation of the figure (particularly useful in 3-D) with right click actions which are guided by messages in the bottom of the graphics window.

For more precise modifications, click on **Edit > Figure properties** or **Axes properties** and let yourselves be guided (this option is not yet available under Mac OS X).

Online help

To access the online help, click on **? > Scilab Help** in the menu bar, or type in the console:

```
-->help
```



Mention
Examples of use can be executed in Scilab and edited in SciNotes in using the associated buttons in the example framework.

To get help with any function, type help in the console followed by the name of the appropriate function. For example:

```
-->help sin
```

displays help for **`sin`** (sine) function.

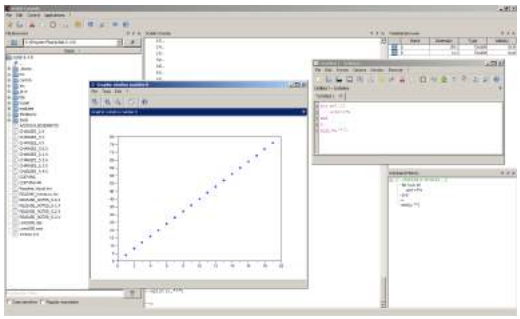
Windows management and workspace customization

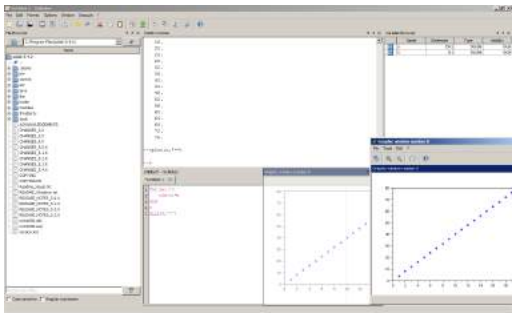
As in the default Scilab environment, where the console, files and variables browsers and command history are all together docked windows, all other windows in Scilab can be repositioned in a single one. For example, the user can choose to position the editor in the default environment of Scilab.

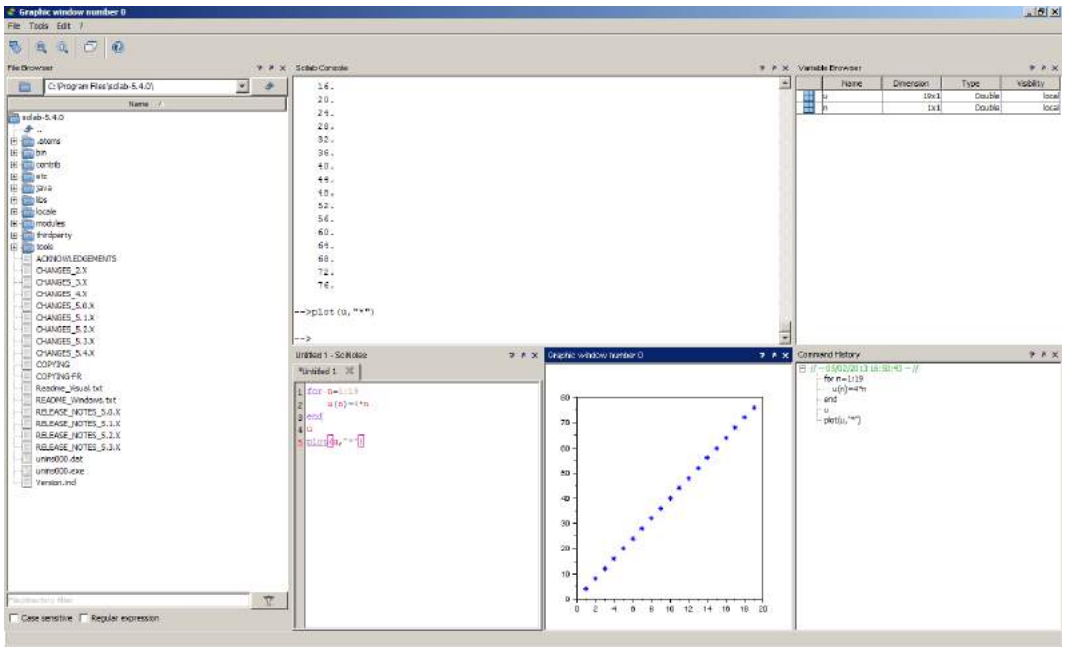
To dock a window in another one, first identify the blue horizontal bar under Windows, or black under Mac OS X and Linux, at the top of the window in the toolbar containing a question mark on the right.

- Under Windows and Linux, click on this bar with the left mouse button and, while maintaining the click, move the mouse pointer in the desired window.
- Under Mac OS X, click on this bar and while maintaining the click, move it in the desired window.

A rectangle appears indicating the future positioning of the window. When the position is the one you want, release the mouse button. To cancel and bring out the window, click on the small arrow on the right of the same bar.







Chapter 2 - Programming

In the examples given in this document, any line preceded by " **-->** " is a command, the other lines are the returns from commands (calculation results, error messages…). Do not write " **-->** " in the editor. They are introduced here to make the distinction between command lines and calculation results as it is displayed in the console after copying/pasting. When presented in a table (without " **-->** " and no calculation result), the commands are depicted exactly as they should be typed in the editor.

Variables, assignment and display

Variables

Scilab is not a computer algebra system. It calculates only with numbers. All calculations are done with matrices, although this may go unnoticed. Even if the concept of matrices is unknown, vectors and sequences of numbers can explain it, as they are, in fact, matrices of dimension $1 \times n$ or $n \times 1$ and a number is itself a matrix of dimension 1×1.

Variables do not need to be declared in advance, but any variable must have a value. For example, obtaining the value of a variable which has not been given a value produces an error:

```
-->a
  !--error 4
Undefined variable : a
```

If a value is assigned to the variable **a**, there is no longer an error:

```
--> a=%pi/4
 a  =
    0.7853982
--> a
 a  =
    0.7853982
```

Variables may take any name that is not already defined by the system:

```
--> Piby2=%pi/2
 Piby2  =
    1.5707963
```

Mention
Like the Scilab functions, a variable name must not have accents or special characters.

The result of a calculation that is not assigned to a variable is automatically assigned to the variable called **ans**:

```
-->3*(4-2)
 ans  =
    6.


-->ans
 ans  =
    6.
```

Functions

Functions are the easiest and most natural way to make computations from variables and for obtaining results from variables.

A function definition begins with **function** and ends with **endfunction**. For example, to convert euros (e) in dollars (d) with an exchange rate (t), the **dollars** function is defined. The variables are **e** and **t** and the image is **d**.

```
-->function d=dollars(e,t); d=e*t; endfunction


-->dollars(200,1.4)
 ans  =
    280.
```

Very usually numerical functions are functions of one real variable. For example, two functions **f** and **g** are defined using the following commands:

```
-->function y=f(x); y=36/(8+exp(-x)); endfunction


-->function y=g(x); y=4*x/9+4; endfunction
```

Mention
The variables **x** and **y** are dummy variables, their names can be reused when defining other functions or in Scilab.

The defined functions can then be used to calculate values:

```
--> f(10)
 ans  =
    4.4999745
--> g(12.5)
 ans  =
    9.5555556
```

Requesting the assignment of a variable

Assignment is made easily using the " **=** " operator.

Display

Writing

Typing the name of a variable displays its value, except when there is " **;** " at the end of the command line.

Brackets

Matrices are defined using square brackets (see page 23). As mentioned before, matrix computation is the basis of calculations in Scilab. A space or comma is used to separate columns and semicolons are used to separate rows.

To define a column vector and obtain a column display:

```
-->v=[3;-2;5]
 v   =
    3.
  - 2.
    5.
```

To define a row vector and obtain a row display:

```
-->v=[3,-2,5]
 v   =
    3.   - 2.    5.
```

Mention
This command can also be typed under the form:
v=[3 -2 5]

To define a matrix and obtain a tabular display:

```
-->m=[1 2 3;4 5 6;7 8 9]
 m   =
    1.    2.    3.
    4.    5.    6.
    7.    8.    9.
```

Mention
This command can also be typed under the form:
m=[1,2,3;4,5,6;7,8,9]

disp function

disp must always be used with parentheses.

With the vector **v** previously defined:

```
-->v(2)
 ans  =
  - 2.


-->disp(v(2))
  - 2.
```

To display a string (usually a sentence), put it between quotes:

```
-->disp("Bob won")
  Bob won
```

To display a combination of words and values use the **string** command which converts values to character strings using a " **+** " between the different parts:

```
-->d=500;


-->disp("Bob won "+string(d)+" dollars")
 Bob won 500 dollars
```

If the sentence contains an single quote, the latter needs to be doubled in the string to be displayed properly:

```
-->disp("It''s fair")
 It's fair
```

Loops

Incrementation

The " **:** " operator allows to define vectors of numbers whose coordinates are in arithmetic sequence. We give: << beginning value: step: ending value>>. It is possible that "ending value" is not reached. If the step is not mentioned, its default value is 1.

For example, to define a row vector of integers which increments in steps of 1 from 3 and 10:

```
-->3:10
 ans  =
    3.    4.    5.    6.    7.    8.    9.    10.
```

or which increments in steps of 2 from 1 to 10:

```
-->1:2:10
 ans  =
    1.    3.    5.    7.    9.
```

or which decreases in steps of 4 from 20 to 2:

```
-->u=20:-4:2
 u  =
    20.    16.    12.    8.    4.
```

For

The easiest loop structure for a fixed number of iterations is written with **`for … end`**.

Example: Calculation of 20 terms of a sequence defined by recurrence by: $\begin{cases} u_1 = 4 \\ u_{n+1} = u_n + 2n + 3 \end{cases}$

Algorithm	Scilab Editor
Put 4 in u(1)	`u(1)=4;`
For n from 1 to 20	`for n=1:20`
u(n+1) takes the value u(n)+2n+3	`u(n+1)= u(n)+2*n+3;`
and u(n)	`disp([n,u(n)])`
Display n and u(n)	`end`
End for	

While

To stop a loop when a given goal is reached, **`while … end`** is used.

I planted a Christmas tree in 2005 measuring 1.20 m. It grows by 30 cm per year. I decided to cut it when it exceeds 7 m. In what year will I cut the tree?

Algorithm	Scilab Editor
Put 1.2 in h (h = tree height)	`h=1.2;`
Put 2005 in y (y = year)	`y=2005;`
While h<7	`while h<7`
h takes the value h+0.3 (the tree grows)	`h=h+0.3;`
y takes the value y+1 (one year passes)	`y=y+1;`
End while	`end`
Display y (the final year)	`disp("I will cut the..`
	`tree in "+string(y))`

Mention

When a command is too long to be written in one line, the editor will wrap lines. One can also wrap lines using " `..` " (two dots) before going to the next line.

Tests

Comparison operators

Useful tests include comparing numbers or determining whether a statement is true or false. Below the corresponding commands:

Equal	Different	Less	Greater	Less or equal	Greater or equal
==	**<>**	**<**	**>**	**<=**	**>=**
True	False	And	Or	No	
%T	**%F**	**&**	**|**	**~**	

Mention
Be cautious with calculation accuracy. Calculations are approached and " == " will sometimes give wrong results (see Calculation accuracy, page 30).

To compare two vectors (or two matrices), the tests " **==** " and " **<>** " will compare term by term. For example:

```
-->X=[1,2,5]; Y=[5,3,5];


-->X==Y
 ans  =
  F F T
```

To test if two vectors are equal, **isequal** is used and **~isequal** is used if they are different:

```
-->isequal(X,Y)
 ans  =
  F


-->~isequal(X,Y)
 ans  =
  T
```

If…then

The basic conditional statements are the following:

- **if … then … else … end**,
- **if … then … elseif … then … else … end**.

if … then must be written on the same line and likewise with **elseif … then**.

Example: Alice throws three dice.

- If she gets three 6's she wins $20,
- If she gets three identical numbers different from 6 she wins $10,
- If she gets two identical numbers she wins $5,
- Otherwise she wins nothing.

Simulate a trial and calculate Alice's winnings using the functions:

- **grand** (see page 22),
- **unique(D)** which keeps only once the values that appears several times in **D**,
- **length(unique(D))** which returns the size of the obtained vector, that is to say 1 if three dice are equal, and 2 if two dice are equal.

Algorithm	Scilab Editor

Put the three values in D
If Alice gets three 6, then
Alice wins 20 dollars
Else if she receives three identical values, then
Alice wins 10 dollars
Else, if she receives two identical values, then
Alice wins 5 dollars
Otherwise
Alice wins nothing
End if
Display Alice's winnings
	`D=grand(1,3,"uin",1,6);`
	`if D==[6,6,6] then`
	`  W=20;`
	`elseif length(unique(D))==1 then`
	`  W=10;`
	`elseif length (unique(D))==2 then`
	`  W=5;`
	`else`
	`  W=0;`
	`end`
	`disp("Alice wins "+..`
	`string(W)+ " dollars")`

2-D and 3-D plots

plot command is used to create plots in the plane. Color and appearance can be specified by putting indications of color and points style within quotes:

- Colors

"b" = blue (by default), **"k"** = black, **"r"** = red, **"g"** = green, **"c"** = cyan, **"m"** = magenta, **"y"** = yellow, **"w"** = white.

- Point styles

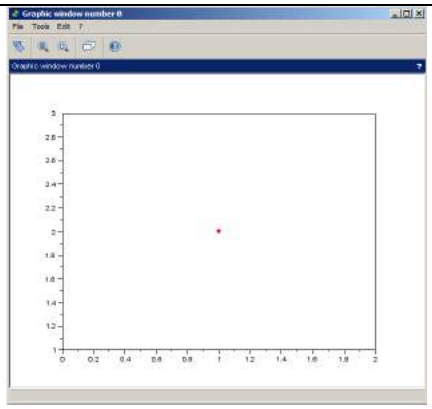
Joined (by default), or **"."**, **"+"**, **"o"**, **"x"**, **"*"**.

Other options are available with: **"s"**, **"d"**, **"v"**, **"<"**, and **">"**.

Basic plots

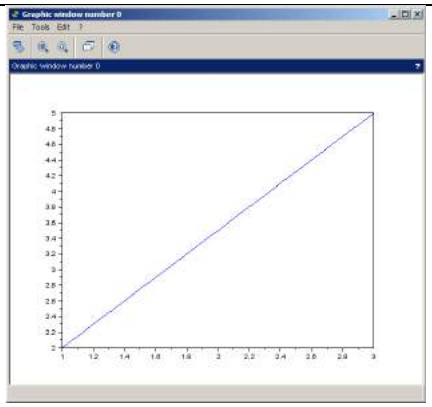
To plot a point

Plot the point A(1 ; 2) with a red point.

Scilab Editor	Graphics Window
`plot(1,2,".r")`	

To plot a segment

Plot the segment [AB] in blue (by default) with A(1 ; 2) and B(3 ; 5).

Scilab Editor	Graphics Window
`plot([1,3],[2,5])`	

Plots of plane curves defined by functions y=f(x)

For a function $x \to f(x)$ the values of x are specified using the **`linspace`** command by writing: **`x=linspace(a,b,n);`** in which **a** is the smallest value of the variable x, **b** the highest value of x, and **n** the number of values calculated between **a** and **b**.

Do not forget the " **`;`** " sympbol otherwise all n values of x will be displayed.

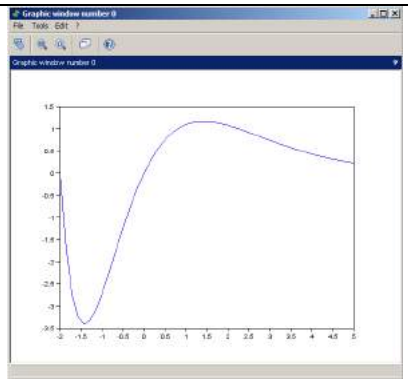
For example, consider two functions f and g defined over the interval [-2 ; 5] by:

$f(x) = (x^2 + 2x)e^{-x}$, and $g(x) = \sin\left(\frac{x}{2}\right)$

With the program below, we plot the curve defined by f, in blue by default:

Scilab Editor	Graphics Window

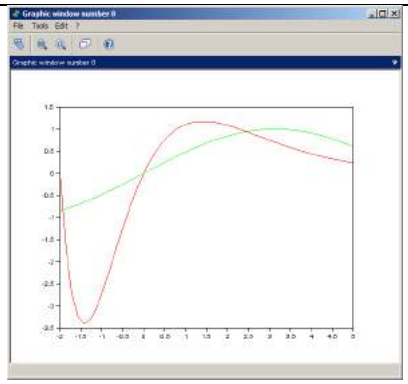
```
function y=f(x)
  y=(x^2+2*x)*exp(-x)
endfunction
x=linspace(-2,5,50);
plot(x,f)
```



By adding the program below, we obtain the plot of two curves, that of **f** in red and that of **g** in green. The previous plot was cleared through the **clf** command (" clear figure ").

Scilab Editor	Graphics Window

```
function y=g(x)
  y=sin(x/2)
endfunction
x=linspace(-2,5,50);
clf
plot(x,f,"r",x,g,"g")
```



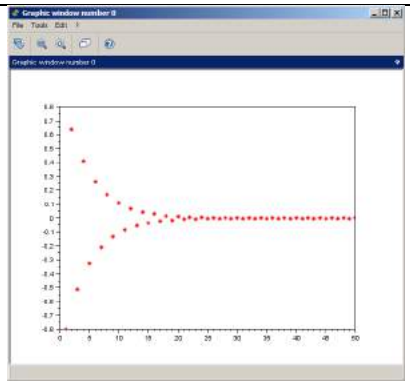
Plots of sequences of points

Terms of a sequence

The most common case is to plot the sequences of points $M(n,u(n))$ after calculating the coordinates $u(n)$ of a vector u. **plot(u,"*r")** specifies the style and color of the points in quotes: red and non-connected stars. By default, points are plotted in blue and are connected.

Scilab Editor	Graphics Window

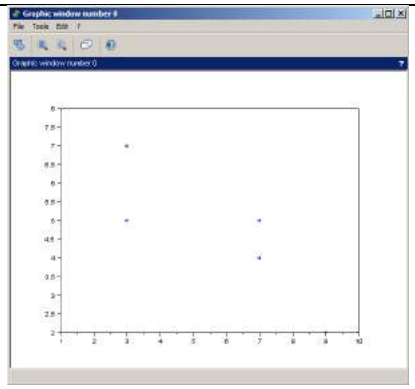
```
for n=1:50
  u(n)=(-0.8)^n;
end
clf; plot(u,"*r")
```



Bivariate statistical data

Bivariate statistical data are given in the form of two vectors: let's call them X and Y. **`plot(X,Y,"<")`** will plot a scatter plot of $M(X_i; Y_i)$ with blue triangles.

Scilab Editor	Graphics Window

`X=[1,3,3,7,7,9,10];`
`Y=[8,7,5,5,4,2,2];`
`clf; plot(X,Y,"<")`
	

Plots in 3 dimensions

Scilab can be used to plot surfaces and curves in space, with many options for the treatment of hidden faces, face colors, points of view, etc. The following examples will illustrate 3-D plots.

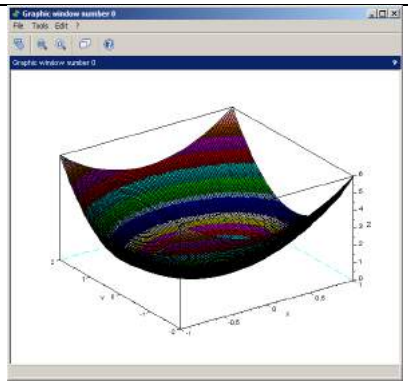
The **`surf`** function can be used for plotting surfaces. This function has three input variables, **`x`**, **`y`** and **`z`**. **`x`** and **`y`** are respectively vectors of size m and n corresponding to points on the axes (Ox) and (Oy). **`z`** is a matrix of dimension $n \times m$ with element z_{ij} corresponding to the height of the point with X-coordinate x_i and Y-coordinate y_j.

To plot the surface defined by a function of the form $z = f(x, y)$, it is necessary to :

- Define function f
- Calculate **`z=feval(x,y,f)'`**
  `feval(x,y,f)` returns the $m \times n$ matrix whose ij element is $f(x_i, y_j)$ which will be transposed by using the single quote symbol " ' "
- Execute **`surf(x,y,z).`**

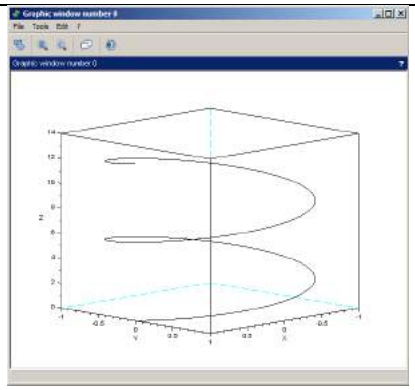
To plot the surface $z = 2x^2 + y^2$ (elliptic paraboloid):

Scilab Editor	Graphics Window

`function z=f(x,y)`
`  z=2*x^2+y^2;`
`endfunction`
`x=linspace(-1,1,100);`
`y=linspace(-2,2,200);`
`z=feval(x,y,f)';`
`clf`
`surf(x,y,z)`
	

Curves in space may be plotted using the **param3d** function. **param3d** has three arguments, **x**, **y** and **z**, each vectors has the same dimension and corresponds to the points (x_i, y_i, z_i) on the curve.

To plot the helix defined by $(x = \cos(t), y = \sin(t), z = t)$:

Scilab Editor	Graphics Window
`t=linspace(0,4*%pi,100);` `param3d(cos(t),sin(t),t)`	

Simulations and statistics

Several functions are available in Scilab to perform simulations quickly and efficiently.

Random sequences

- **grand(1,p,"uin",m,n)** returns a vector of p random integer sequences between m and n with p positive integer, m and n integers and $m \leq n$.

```
-->t= grand(1,4,"uin",1,6)
 t  =
    3.    1.    3.    6.
```

- **grand(1,p,"unf",a,b)** returns a vector of p random real sequences between a and b with p positive integer, a and b real and $a \leq b$.

```
-->tr= grand(1,2,"unf",-1,1)
 tr  =
  - 0.7460264    0.9377355
```

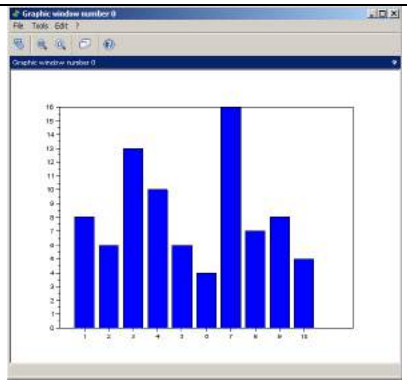
Statistics

All the standard statistical functions are listed on page 32.

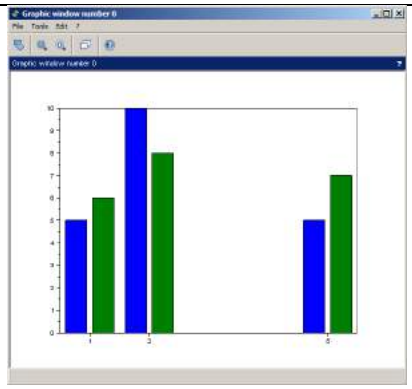
Keep particularly in mind:

- The function **bar(x,n,color)** which draws bar graphs:

Scilab Editor	Graphics Window

`x=[1:10];`
`n=[8,6,13,10,6,4,16,7,8,5];`
`clf; bar(x,n)`
	

- For a bar graph representing two sets side by side: consider the series of values X and the two series of numbers n1 and n2. For plotting, n1 and n2 must be column vectors that is why transposes are used in the example below:

Scilab Editor	Graphics Window
`X=[1,2,5];n1=[5,10,5];n2=[6,8,7];` `bar(X,[n1',n2'])`	

The **color** optional argument defines the color as in the **plot** function.

Additional information on matrices and vectors

Accessing elements

Square braces are used to define matrices. A space or a comma is used to switch from one column to another and semicolons are used to switch from one line to another.

```
-->m=[1 2 3;4 5 6]
 m  =
    1.    2.    3.
    4.    5.    6.
```

Mention
This command can also be typed under the form:
m=[1,2,3;4,5,6]

Parentheses are used to access elements or to modify them.

```
-->m(2,3)
 ans  =
    6.


-->m(2,3)=23
 m  =
    1.    2.    3.
    4.    5.    23.
```

The " : " operator is used to designate all the rows or all columns of a matrix.

To view the second row of the matrix **m**, type:

```
-->m(2,:)
 ans  =
    4.    5.    23.
```

And the third column:

```
-->m(:,3)
 ans  =
    3.
    23.
```

To obtain the transpose of a matrix or a vector use the single quote symbol " ' ":

```
-->m'
 ans  =
    1.    4.
    2.    5.
    3.    23.
```

Operations

The operations " * ", " / " are matrix operations. To make element wise operations, we need to put a dot before the operating sign: " .* ", " ./ ".

```
-->A=[1,2,3;4,5,6]
 A  =
    1.    2.    3.
    4.    5.    6.
```

```
-->B=[1;1;2]
```

```
 B  =
    1.
    1.
    2.
```

```
-->A*B
 ans  =
    9.
    21.
```

Matrix multiplication

```
-->A*A
    !--error 10
Inconsistent multiplication.
```

Dimensions are not consistent

```
-->A.*A
 ans  =
    1.     4.     9.
    16.    25.    36.
```

Element wise multiplication

```
-->2*(A+2)
 ans  =
    6.     8.     10.
    12.    14.    16.
```

The operation is performed on each element because 2 is a number

```
-->A/A
 ans  =
    1.            1.518D-16
    3.795D-15     1.
```

Gives the matrix X for which X*A = A
The exact result is:
1. 0
0 1.

For reasons of calculation accuracy, the result can be slightly different depending on your version of Scilab and your operating system (see calculation accuracy, page 29).

```
-->A./A
 ans  =
    1.    1.    1.
    1.    1.    1.
```

Gives the matrix divided element wise

In the case of vectors:

```
-->C=1:4
 C  =
    1.    2.    3.    4.
```

```
-->C*C
    !--error 10
Inconsistent multiplication.
```

Dimensions are not consistent

```
-->C.*C
 ans  =
    1.    4.    9.    16.
```

It is also possible to write **C^2** because, for vectors, exponents are interpreted as an operation element wise. This is not the case with matrices.

```
-->1/C
 ans  =
    0.0333333
    0.0666667
    0.1
    0.1333333
```

In this particular case with vectors, we find the vector X for which C*X = 1

```
-->(1)./C
 ans  =
    1.    0.5    0.3333333
0.25
```

Reverse element wise
As previously, **C^(-1)** is possible. The parentheses around 1 are necessary so that the point is not considered as a comma as part of the number 1. It is also possible to write **1 ./C** with a space between **1** and " **.** "

Solving linear systems

To solve the linear system AX = Y, in which A is a square matrix, use the backslash " \ "

X = A \ Y.

Be cautious, the operation Y / A will give (only if the dimensions are correct) another result, that is to say the matrix Z for which Z A = Y.

To solve the system: $\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix} \times X = \begin{pmatrix}1\\ 1\end{pmatrix}$

```
-->A=[1 2 3;4 5 6];

-->Y=[1;1];

-->X=A\Y
 X  =
  - 0.5
    0.
    0.5

-->A*X
 ans  =
    1.
    1.
```

Some useful functions

Sort

The **gsort** function is used to sort the elements of a vector in ascending or descending order.

```
-->v=[2,6,9,6,-4,0,2]
 v  =
    2.    6.    9.    6.  - 4.    0.    2.


--> gsort(v,"g","i")
 ans  =
  - 4.    0.    2.    2.    6.    6.    9.


--> gsort(v)
 ans  =
    9.    6.    6.    2.    2.    0.  - 4.
```

Length

The **length** function returns the number of coordinates for a vector. The **size** function returns the dimensions (rows, columns) for a matrix.

```
-->U=[1:10]
 U  =
    1.    2.    3.    4.    5.    6.    7.    8.    9.    10.


-->length(U)
 ans  =
    10.


-->m=[1 2 3;4 5 6];


-->size(U)
 ans  =
    2.    3.
```

Sum and product

The **sum** and **prod** functions respectively calculate the sum and the product of their argument elements.

```
-->U=[1:10];


-->sum(U)
 ans  =

    55.


-->prod(U)
 ans  =

    3628800.
```

Unique

The **unique** function keeps only once the elements of a vector (even if they are repeated several times) and sorts them in ascending order.

```
-->v=[2,6,9,6,-4,0,2]
 v  =

    2.    6.    9.    6.  - 4.    0.    2.


-->unique(v)
 ans  =

  - 4.    0.    2.    6.    9.
```

Find

The **find** function searches for elements in a vector or a matrix and returns a vector containing the corresponding indices.

To find all the elements of the vector *w* smaller than 5:

```
-->w=[1,5,3,8,14,7,3,2,12,6]; find(w<5)
 ans  =

    1.    3.    7.    8.
```

The resulting vector (1,3,7,8) indicates that elements w_1, w_3, w_7 and w_8 are smaller than 5.

To find all the elements of the vector w equal to 3:

```
-->w=[1,5,3,8,14,7,3,2,12,6]; find(w==3)
 ans  =
    3.    7.
```

The resulting vector (3,7) indicates that elements w_3 and w_7 are equal to 3.

Accuracy computation

Computation

Numbers have an absolute value between about 2.2×10^{-308} and $1.8\times10^{+308}$.

The number **%eps** equals to **2.220446049D-16** gives the smallest relative precision that can be obtained in computations, which therefore gives about 16 decimal digits.

Example 1: Computation of sin(π)

```
-->sin(%pi)
 ans  =
1.225D-16
```

The value of $sin(\pi)$ above is not 0, but it is considered as zero. Indeed, compared to the maximum value of the sine function (i.e. 1), it is equal to 0 with a value less than **%eps**.

Example 2: Testing if the triangle with sides $\sqrt{3}$, 1 et 2 is a right-angled triangle:

```
-->a=sqrt(3)
 a  =
    1.7320508
```

```
-->b=1
 b  =
    1.
```

```
-->c=2
 c  =
    2.
```

```
-->a^2+b^2==c^2
 ans  =
  F
```

The program responds false
because the value of a^2 + b ^ 2 is approximate

```
-->abs(a^2+b^2-c^2)<%eps
 ans  =
  F
```

The program responds false because absolute precision is called for

```
-->abs(a^2+b^2-c^2)/c^2<%eps
 ans  =
  T
```

The program responds true because relative precision is called for

Display

Results are displayed, by default, with 10 characters, including the decimal point and the sign. The **`format`** function is used to display more digits. To display 20 digits, type **`format(20)`**.

Reconsidering $a = \sqrt{3}$:

```
-->a^2
 ans  =
    3.
```

Here there are 7 decimal places, and we do not see the error.

```
-->format(20)

-->a^2
 ans  =
    2.99999999999999956
```

Here there are 17 decimal places, and we can see the error

Solving differential equations

To find the solutions of an explicit system of differential equations, simply reduce to differential equations of order 1.

$\begin{cases} y'(t) = f\big(t, y(t)\big) \\ y(t_0) = y_0 \end{cases}$ in which t_0 and t represent time and y_0 and $y(t)$ are vectors,

then apply the **`ode`** function: y=ode(y0,t0,t,f), with:

- **`y0`**: initial condition, vector of dimension n,
- **`t0`**: initial time,
- **`t`**: vector of dimension T corresponding to the times in which the solution is computed. This vector must begin with **`t0`**.
- **`f`**: function defining the system under the form:

```
function yprim=f(t,y)
    yprim(1)=...
    yprim(2)=...
    ....
    yprim(n)=...
endfunction
```

The solution y is a matrix of dimension $n \times T$: $\begin{pmatrix} y_1(1) & y_1(2) & \vdots & y_1(T) \\ y_2(1) & y_2(2) & \vdots & y_2(T) \\ \vdots & \vdots & & \vdots \\ y_n(1) & y_n(2) & \ldots & y_n(T) \end{pmatrix}$

Example: Solving the differential equation $\begin{cases} y'' = -4y \\ y(0) = 3, y'(0) = 0 \end{cases}$

This equation of order 2 is reduced to a system of two equations of order 1 as defined by:

$Y = \begin{pmatrix} Y(1) \\ Y(2) \end{pmatrix} = \begin{pmatrix} y \\ y' \end{pmatrix}, Yprim = \begin{pmatrix} Yprim(1) \\ Yprim(2) \end{pmatrix} = \begin{pmatrix} y' \\ y'' \end{pmatrix}$ and $\begin{matrix} Yprim(1) = (2) \\ Yprim(2) = -4 \times Y(1) \end{matrix}$

Comment	Scilab Editor

We define the function returning output vector y' from input variables t and y (y is a vector).
We define the values of t for plotting.
(the solver chooses itself the good values of t for the internal computation of the solution).
We define the initial conditions.
We execute ode.
We plot the integral curve of y with respect to t.
	`function yprim=f(t,y)`
	`  yprim(1)=y(2);`
	`  yprim(2)=-4*y(1) ;`
	`endfunction`
	`t0=0; tmax=5;`
	`t=t0:0.05:tmax;`
	`y0=3; yprim0=0;`
	`y=ode([y0;yprim0],t0,t,f);`
	`clf; plot(t,y(1,:))`

Chapter 3 – Useful Scilab functions

Analysis

- **`sqrt(x)`** returns the square root of x with x real positive or zero, and the complex root of real positive part otherwise.
- **`log(x)`** returns the natural logarithm of x with x real or complex number.
- **`exp(x)`** returns the exponential of x with x real or complex number.
- **`abs(x)`** returns the absolute value of x real (or the module if x is complex).
- **`int(x)`** returns the truncation of x real (the integer before the decimal).
- **`floor(x)`** returns the integer part of x real (the integer n for which $n \leq x < n+1$).
- **`ceil(x)`** for x real returns the integer n for which $n-1 < x \leq n$.

Probability and statistics

- **`factorial(n)`** returns the factorial of n with n positive or zero integer.
- **`grand(1,p,"uin",m,n)`** returns a vector of p random integer sequences taken between m and n with p positive integer, m and n integers and $m \leq n$.
- **`grand(1,p,"unf",a,b)`** returns a vector of p random real sequences taken between a and b with p positive integer, a and b real and $a \leq b$.
- **`sum(n)`** returns the sum of the values of vector n (used to calculate a total).
- **`cumsum(n)`** returns the vector of increasing cumulative values of vector n (used to calculate the increasing cumulative numbers).
- **`length(v)`** returns the number of coordinates of vector v.
- **`gsort(v)`** returns the vector of numbers or strings v sorted in descending order.
- **`gsort(v,"g","i")`** returns the vector of numbers or strings v sorted in ascending order.
- **`mean(v)`** returns the average of the vector of numbers v.
- **`stdev(v)`** returns the standard deviation of numbers v vector.
- **`bar(v,n,couleur)`** draws the bar graph with v as X-coordinate, n as Y-coordinate, v and n being same size line vectors. By default, **`bar(n)`** draws the bar graph of n in blue with 1,2,3… as X-coordinates.
- **`bar(v,[n1',n2'])`** draws a double bar graph with v as X-coordinate, n1 as Y-coordinate in blue and n2 as Y-coordinate in green, with v, n1 and n2 being same size line vectors.
- **`rand(n,p)`** with n and p positive integers, returns a matrix $n \times p$ of numbers randomly taken between 0 and 1.
- **`rand()`** returns a real number randomly taken between 0 and 1.
- **`floor(x)`** returns the integer part of x real number. In particular, if p is real between 0 and 1, **`floor(rand()+p)`** will be 1 with p probability and 0 with $1-p$ probability.

Display and plot

- **`clf`** means " clear figure " and clears the current figure on the graphics window.
- **`plot`** allows to draw curves and scatter plots in 2 dimensions.
- **`linspace(a,b,n)`**, with a and b real and n integer, defines a vector of n values regularly spaced between a and b.
- **`scf`** allows to open or to select other graphics windows than the current one.
- **`surf`** allows 3-D surface plots.
- **`bar(X,Y)`** in which X and Y are vectors, draws the bar graph of the series of values for X which has for numbers the values of Y.
- **`plot(X,Y,"*")`** draws the scatter plot of coordinates (X(i),Y(i)) as stars. The color can be specified.
- **`plot(Y,"+")`** draws the scatter plot of coordinates (i,Y(i)) as cross.
- **`disp("Sentence")`** displays what is written in double quotes.
- **`disp(A)`** in which A is a matrix of numbers, displays the table of the values of A.
- **`disp("Sentence"+string(x))`** displays the sentence and the value of number x.
- **`xclick`** returns the coordinates of the point clicked in a graphics window.

Utilities

- **`unique(v)`** returns the vector v with a unique occurrence of its repeated components.
- **`sum(v)`** returns the sum of all the elements of the vector or the matrix v.
- **`prod(v)`** returns the product of all the elements of the vector or the matrix v.
- **`find(<test about v>)`** returns the indices of the elements of vector v satisfying the test.
- **`disp(x,y,…)`** displays the values of its arguments in the console.
- **`string(x)`** converts number x to a string.
- **`format(n)`** in which n is an integer greater than or equal to 2, sets the display to n characters, including the sign and the decimal dot.
- **`zeros(n,p)`** defines a $n \times p$ matrix that only contains zeros.
- **`feval(x,y,f)`** in which x and y are respectively vectors of size m and n, defines the matrix $m \times n$ whose element (i, j) is $f(x(i), y(j))$.
- **`help`** function opens the help browser on the right function page.
- **`tic`** starts a clock.
- **`toc`** stops the clock.