Scilab
A Hands on Introduction

by
Satish Annigeri Ph.D.
Professor of Civil Engineering
B.V. Bhoomaraddi College of Engineering & Technology, Hubli
satish@bvb.edu

Department of Civil Engineering
B.V. Bhoomaraddi College of Engineering & Technology, Hubli
17 & 18 April, 2004

Table of Contents

Preface

Scilab is a software for numerical mathematics and scientific visualization. It is capable of interactive calculations as well as automation of computations through programming. It provides all basic operations on matrices through built-in functions so that the trouble of developing and testing code for basic operations are completely avoided. Its ability to plot 2D and 3D graphs helps in visualizing the data we work with. All these make Scilab an excellent tool for teaching, especially those subjects that involve matrix operations. Further, the numerous toolboxes that are available for various specialized applications make it an important tool for research. Being compatible with Matlab®, all available Matlab M-files can be directly used in Scilab. Scicos, a hybrid dynamic systems modeler and simulator for Scilab, simplifies simulations. The greatest features of Scilab are that it is multi-platform and is free. It is available for many operating systems including Windows, Linux and MacOS X. More information about the features of Scilab are given in the Introduction.

Scilab can help a student understand all intermediate steps in solving even complicated problems, as easily as using a calculator. In fact, it is a calculator that is capable of matrix algebra computations. Once the student is sure of having mastered the steps, they can be converted into functions and whole problems can be solved by simply calling a few functions. Scilab is an invaluable tool as solved problems need not be restricted to simple examples to suit hand calculations.

Scilab is the outcome of years of development and continues to be improved and developed. Having a rich set of features and being in wide use, its developers could very well have chosen to commercialize it. But they have chosen to make it a 'free' software. Free, as in 'free of cost' as well as in 'freedom', because the source code is also available for those who wish to modify and improve it. You can visit the following websites to see some definitions of software freedom and licensing issues:

http://www.fsf.org/licenses/licenses.html and

Open Source Initiative (**http://www.opensource.org/licenses**

You can also read the Scilab software license at the following website:

http://scilabsoft.inria.fr/license.txt

The Scilab license is included in the Appendix at the end of this document.

When its developers have been so generous, we as users must contribute to this movement by learning to use it and applying it to solve problems. This tutorial is an attempt to introduce students to the basics of Scilab. The next part of the tutorial is aimed at teaching students of Civil Engineering to the basics of Scilab by applying it to the problem of matrix analysis of plane frames. I hope this motivates students to learn and apply Scilab to solve a wider range of problems.

This is the first version of this document and will certainly contain errors, typographical as well as factual. You can help improve this document by reporting all errors you find and by suggesting modifications and additions. Your views are always welcome. I can be reached at the email address given on the cover page.

Acknowledgments

It goes without saying that my first indebtedness is to the developers of Scilab and the consortium that continues to develop it. I must also thank Dr. A.B. Raju, E&EE Department, BVBCET who first introduced me to Scilab and forever freed me from using Matlab.

April 2004 *Satish Annigeri*

Introduction

Scilab is a scientific software package for numerical computations providing a powerful open computing environment for engineering and scientific applications. Developed since 1990 by researchers from INRIA (French National Institute for Research in Computer Science and Control, **http://www.inria.fr/index.en.html**) and ENPC (National School of Bridges and Roads, **http://www.enpc.fr/english/int_index.htm**), it is now maintained and developed by Scilab Consortium (**http://scilabsoft.inria.fr/consortium/consortium.html**) since its creation in May 2003.

Distributed freely and open source through 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 designed to be an open system where the user can define new data types and operations on these data types by using overloading.

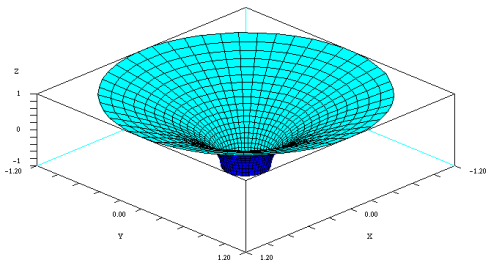
A number of toolboxes are available with the system:

- 2-D and 3-D graphics, animation
- Linear algebra, sparse matrices
- Polynomials and rational functions
- Simulation: ODE solver and DAE solver
- Scicos: a hybrid dynamic systems modeler and simulator
- Classic and robust control, LMI optimization
- Differentiable and non-differentiable optimization
- Signal processing
- Metanet: graphs and networks
- Parallel Scilab using PVM
- Statistics
- Interface with Computer Algebra (Maple, MuPAD)
- Interface with Tcl/Tk
- And a large number of contributions for various domains.

Scilab works on most Unix systems including GNU/Linux and on Windows 9X/NT/2000/XP. It comes with source code, on-line help and English user manuals. Binary versions are available.

Some of its features are listed below:

- Basic data type is a matrix, and all matrix operations are available as built-in operations.
- Has a built-in interpreted high-level programming language.
- Graphics such as 2D and 3D graphs can be generated and exported to various formats so that they can be included into documents.



To the left is a 3D graph generated in Scilab and exported to GIF format and included in the document for presentation. Scilab can export to Postscript and GIF formats as well as to Xfig (popular free software for drawing figures) and LaTeX (free scientific document preparation system) file formats.

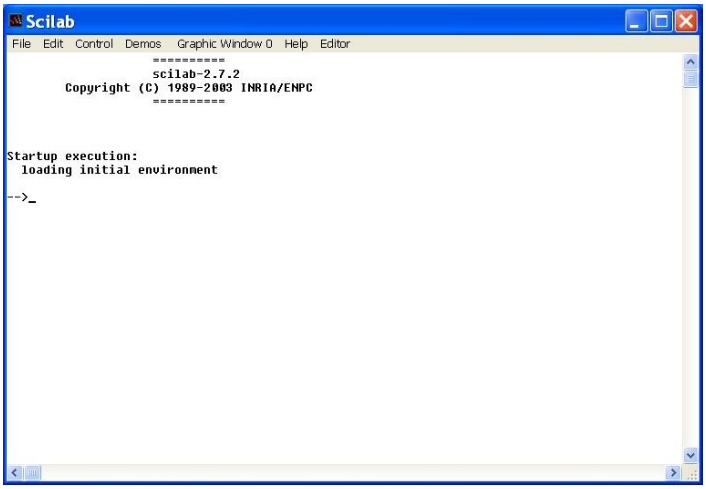
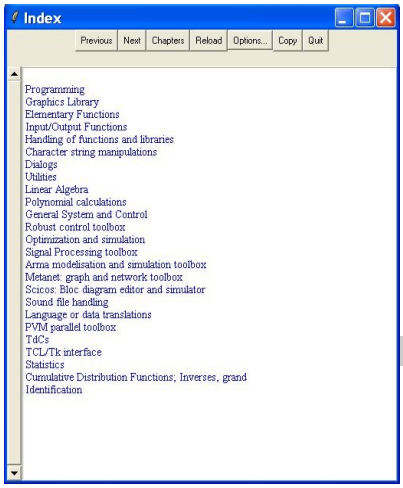


Fig. 1 Scilab environment

When you startup Scilab, you see a window as shown in Fig. 1 above. The user enters Scilab commands at the prompt (**-->**). But many of the commands are also available through the menu at the top. The most important menu for a beginner is the "Help" menu. Clicking on the "Help" menu opens a help window with a list of topics on which help is available. Clicking on the relevant topic takes you into a hypertext linked documents similar to web pages.



Help on specific commands can also be accessed directly from the command line instead of having to navigate through a series of links. Thus, to get help on the Scilab command "inv", simply type

```
-->help inv
```

on the command line.

Scilab can be used as a simple calculator to perform numerical calculations. It also has the ability to define variables and store values in them so that they can be used later. This is demonstrated in the following examples:

```
-->2+3
ans =
      5.
-->2/3
ans =
      .6666667
-->2^3
ans =
      8.
```

```
-->a=2
a = 2.
-->b=
      3.
b =
      3.
-->c=a+b
c =
      5.
```

```
-->pi=atan(1.0)*4
pi =
      3.1415927
-->sin(pi/4)
ans =
      0.7071068
-->exp(0.1)
ans =
      1.1051709
```

Note that Scilab creates a variable named "`ans`" to store results of calculations whenever the user does not supply a variable for the purpose. You could enter more than one command on the same line by separating the commands by semicolons(;). The semicolon suppresses echoing of intermediate results. Try the command `-->a=5;` and you will notice that the prompt reappears immediately without echoing `a=5`.

Tutorial 2 – The Workspace and Working Directory

While the Scilab environment is the visible face of Scilab, there is another that is not visible. It is the memory space where all variables and functions are stored, and is called the **Workspace**. Many a times it is necessary to inspect the workspace to check whether a variable or a function has been defined or not. The following commands help the user in inspecting the memory space: **`who`** , **`whos a`**nd **`who_user()`**. Use the online help to learn more about these commands.

The **`who`** command lists the names of variables in the Scilab workspace. Note the variable names preceded by the "%" symbol. These are special variables that are used often and therefore predefined by Scilab. It includes %pi (π), %e (e), %i ($\sqrt{-1}$), %inf (∞), %nan (NaN) and others.

The **`whos`** command lists the variables along with the amount of memory they take up in the workspace. The variables to be listed can be selected based on either their type or name. Some examples are:

`-->whos()`	Lists entire contents of the workspace, including functions, libraries, constants
`-->whos -type constants`	Lists only variables that can store real or complex constants. Other types are boolean, string, function, library, polynomial etc. For a complete list use the command **`-->help typeof`**.
`-->whos -name nam`	Lists all variables whose name begins with the letters nam

To understand how Scilab deals with numbers, try out the following commands and use the **`whos`** command as follows:

`-->a1=5;`	Defines a real number variable with name '**`a1`**'
`-->a2=sqrt(-4)`	Defines a complex number variable with name '**`a2`**'
`-->a3=[1, 2; 3, 4]`	Defines a 2x2 matrix with name '**`a3`**'
`-->whos -name a`	Lists all variables with name starting with the letter '**`a`**'

```
Name     Type          Size        Bytes

a3       constant      2 by 2      48
a2       constant      1 by 1      32
a1       constant      1 by 1      24
```

Now try the following commands:

`-->a1=sqrt(-9)`	Converts '**`a1`**' to a complex number
`-->whos -name a`	Note that '**`a`**' is now a complex number
`-->a1=a3`	Converts '**`a1`**' to a matrix
`-->whos -name a`	Note that '**`a`**' is now a matrix
`-->save('ex01.dat')`	Saves all variables in the workspace to a disk file **`ex01.dat`**
`-->load('ex01.dat')`	Loads all variables from a disk file **`ex01.dat`** to workspace

Note the following points:

- Scilab treats a scalar number as a matrix of size 1x1 (and not as a simple number) because the basic data type in Scilab is a matrix.
- Scilab automatically converts the type of the variable as the situation demands. There is no need to specifically define the type for the variable.

Tutorial 3 – Matrix Operations

Matrix operations that are built-in into Scilab are addition, subtraction, multiplication, transpose, inversion, determinant, trigonometric, logarithmic, exponential functions and many others. Study the following examples:

`-->a=[1 2 3; 4 5 6; 7 8 9];`	Define a 3x3 matrix
`-->b=a';`	Transpose **a** and store it in **b**.
`-->c=a+b`	Add **a** to **b** and store the result in **c**. **a** and **b** must be of the same size.
`-->d=a-b`	Subtract **b** from **a** and store the result in **d**.
`-->e=a*b`	Multiply **a** with **b** and store the result in **e**. **a** and **b** must be comptible for matrix multiplication.
`-->f=[3 1 2; 1 5 3; 2 3 6];`	Define a 3x3 matrix with name **f**.
`-->g=inv(f)`	Invert matrix **f** and store the result in **g**. **f** must be square and positive definite. A warning will be displayed if it is ill conditioned.
`-->f*g`	The answer must be an identity matrix
`-->det(f)`	Determinant of **f**.
`-->log(a)`	Matrix of log of each element of **a**.
`-->a .* b`	Element by element multiplication.
`-->a^2`	Same as **a*a**.
`-->a .^2`	Element by element square.

There are some handy functions to generate commonly used matrices, such as zero matrices, identity matrices etc.

`-->a=zeros(5,8)`	Creates a 5x8 matrix with all elements zero.
`-->b=ones(4,6)`	Creates a 4x6 matrix with all elements 1
`-->c=eye(3,3)`	Creates a 3x3 identity matrix
`-->d=eye(3,3)*10`	Creates a 3x3 diagonal matrix

It is possible to generate a range of numbers to form a vector. Study the following command:

`-->a=[1:5]`	Creates a vector with 5 elements as follows [1, 2, 3, 4, 5]
`-->b=[0:0.5:5]`	Creates a vector with 11 elements as follows [0, 0.5, 1.0, 1.5, ... 4.5, 5.0]

A range requires a start value, an increment and an ending value, separated by colons (:). If only two values are given (separated by only one colon), they are taken to be the start and end values and incremented is assumed to be 1.

You can create an empty matrix with the command **a=[]**.

Tutorial 4 – Sub-matrices

A sub-matrix can be identified by the row and column numbers at which it starts and ends. Let us first create a matrix of size 5x8.

`-->a=rand(5,8)*100`	Generates a 5x8 matrix whose elements are generated as random numbers.

Since the elements are random numbers, each person will get a different matrix. Let us assume we wish to identify a 2x4 sub-matrix of '**a**' demarcated by rows 3 to 4 and columns 2 to 5. This is obtained as **a(3:4, 2:5)**. The range of rows and columns is represented by the range commands 3:4 and 2:5 respectively. Thus 3:4 defines the range 3, 4 while 2:5 defines the range 2, 3, 4, 5. However, matrix '**a**' remains unaffected.

`-->b=a(3:4, 2:5)`	This command copies the sub-matrix of into '**b**'.

A sub-matrix can be overwritten just as easily as it can be copied. To make all elements of the sub-matrix between the above range equal to zero, use the following command:

`-->a(3:4, 2:5)=zeros(2,4)`	This command creates a 2x4 matrix of zeros and puts it into the sub-matrix of '**a**' between rows 3:4 and columns 2:5.

Note that the sub-matrix on the left hand side and the matrix on the right side (a zero matrix in the above example) must be of the same size.

While using range to demarcate rows and/or columns, it is permitted to leave out the start (or end) value in the range, in which case it is assumed to be 1 (or the number of the last row or column). To indicate all rows (or columns) it is enough to use only the colon (:). Thus, the sub-matrix consisting of all the rows and columns 2 and 3 of **a**, the command is **a(:, 2:3)**. Naturally **a(:, :)** represents the whole matrix, which of course could be represented simply as **a**.

Tutorial 5 – Statistics

Scilab can perform all basic statistical calculations. The data is assumed to be contained in a matrix and calculations can be performed treating rows (or columns) as the observations and the columns (or rows) as the parameters. To choose rows as the observations, the indicator is '**r**' or 1. To choose columns as the observations, the indicator is '**c**' or 2. If no indicator is furnished, the operation is applied to the entire matrix element by element. The available statistical functions are **sum()**, **mean()**, **stdev()**, **st_deviation()**, **median()**.

Let us first generate a matrix of 5 observations on 3 parameters. Let the elements be random numbers. This is done using the following command:

`-->a=rand(5,3)`	Creates a 5x3 matrix of random numbers .

Assuming rows to be observations and columns to be parameters, the sum, mean and standard deviation are calculated as follows:

`-->s=sum(a, 'r')`	Sum of columns of **a**.
`-->m=mean(a,1)`	Mean value of each column of **a**.
`-->sd=stdev(a, 1)`	Standard deviation of **a**.
`-->sd2=st_deviation(a, 'r')`	Standard deviation of **a**. Sample size std.
`-->mdn=median(a,'r')`	Median of columns of **a**.

The same operations can be performed treating columns as observations by replacing the '**r**' or 1 with '**c**' or 2.

When neither '**r**' (or 1) nor '**c**' (or 2) is supplied, the operations are carried out treating the entire matrix as a set of observations on a single parameter.

The maximum and minimum values in a column, row or matrix can be obtained with the **max()** and **min()** functions respectively in the same way as the above statistical functions, except that you must use '**r**' or '**c**' but not 1 or 2.

Tutorial 6 – Plotting Graphs

Let us learn to plot simple graphs. We first have to generate the data to be used for the graph. Let us assume we want to draw the graph of $\cos(x)$ and $\sin(x)$ for one full cycle (2π radians). Let us first generate the values for the x-axis with the following command:

```
-->x=[0:%pi/16:2*%pi];
```

In the above command, note that **%pi** is a predefined constant representing the value of π. The command to create a range of values, **0:%pi/16:2*%pi**, requires a starting value, an increment and an ending value. In the above example, they are 0, $\pi/16$ and 2π respectively. The increment is optional and when not given, it is taken to be 1. Thus, '**x**' is a vector with 33 elements.

Next, let us create the values for the y-axis, first column representing cosine and the second sine. They are created by the following commands:

```
-->y=[cos(x) sin(x)]
```

Note that **cos(x)** and **sin(x)** are the two columns of a new matrix which is first created and then stored in y. We can now plot the graph with the command:

```
-->plot2d(x,y)
```

The graph generated by this command is shown below. The graph can be enhanced and annotated. You can add grid lines, labels for x- and y-axes, legend for the different lines etc.

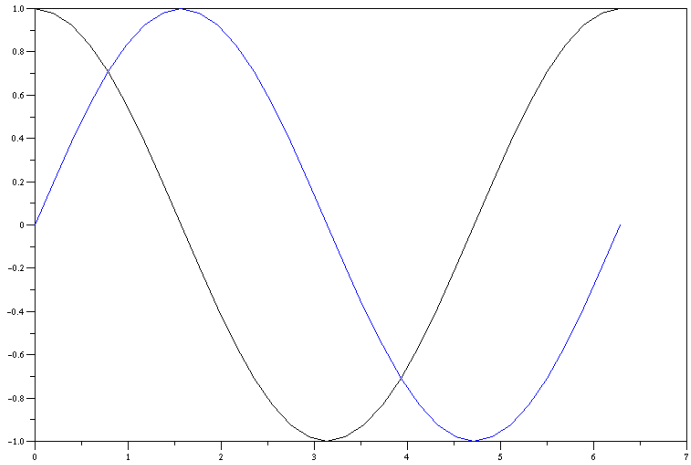


Fig. 2 Graph of sin(x) and cos(x) using function ***plot2d()***

You can learn more about the **plot2d** and other related functions from the online help.

Tutorial 7 – Scilab Programming Language

Scilab has a built-in interpreted programming language so that a series of commands can be automated. The programming language offers most of the features of any high level language, such as looping (for, while), conditional execution (if-then-else, select) and functions. The greatest advantage is that the statements can be any valid Scilab commands.

To loop over an index variable '**i**' from 1 to 10 and display its value each time, the following loop you can try the following commands at the prompt:

```
-->for i=1:10
-->disp(i)
-->end
```

The **for** loop is closed by the corresponding **end** statement. Once the loop is closed, the block of statements enclosed within the loop will be executed. The **disp(i)** command displays the value of **i**.

Conditional execution is performed using the **if-then-elseif-else** construct. Try the following statements on the command line:

```
-->x=10;
-->if x<0 then disp('Negative')
-->elseif x==0 then disp('Zero')
-->else disp('Positive')
-->end
 Positive
```

This will display the word **Positive**.

A list of all the Scilab programming language primitives and commands can be displayed by the command **what()**. The command produces the following list:

```
if      else    for      while    end      select   case    quit
exit    return  help     what     who      pause    clear   resume
then    do      apropos  abort    break    elseif
```

You can learn about each command using the help command. Thus **help while** will give complete information about **while** along with examples and other commands that are related to it.

Tutorial 8 – Functions in Scilab

Functions serve the same purpose in Scilab as in other programming languages. They are independent blocks of code, with their own input and output parameters that can be associated with variables at the time of calling the function. They modularize a program and encapsulate a series of statements and associate them with the name of the function.

Scilab provides a built in editor, called as SciPad within Scilab wherein the user can type the code for functions and compile and load it into the workspace. SciPad can be invoked by clicking on 'Editor' choice on the main menu at the top of the Scilab work environment.

Let us write a simple function to calculate the length of a line in the x-y plane, given the coordinates of its two ends (x1, y1) and (x2, y2). The length of such a line is given by the expression $l=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$. Before defining the function in SciPad, let us try it out in Scilab.

```
-->x1=1; y1=4; x2=10; y2=6;
-->dx=x2-x1; dy=y2-y1;
-->l=sqrt(dx^2 + dy^2)
l =
      9.2195445
```

Studying the above statements to be put into the functions, notice that **x1**, **y1**, **x2** and **y2** are input values and the length '**l**' is the output parameter. Define the code as described below:

1. Open SciPad by clicking on Editor on the main menu.
2. Type the lines of code to define the function.
3. Save the contents of the file to a disk file by clicking File > Save in SciPad and choosing a name for the file.
4. Load the function into Scilab by clicking on 'Load into Scilab' on the SciPad main menu.

type the following code into SciPad:

```
function [l] = len(x1, y1, x2, y2)
dx=x2-x1; dy=y2-y1;
l=sqrt(dx^2 + dy^2);
endfunction
```

In the above code, **function** and **endfunction** are Scilab keywords and must be typed exactly as shown. They signify the start and end of a function definition.

The variable enclosed between square brackets is an output parameter, and will be returned to Scilab workspace (or to the parent function from where it was invoked). The name of the function has been chosen as '**len**' (short for length, as the name length is already used by Scilab for another purpose). The input parameters **x1**, **y1**, **x2** and **y2** are the variables bringing in input from the Scilab workspace (or the parent function from where it is called). This function will be invoked as **ll = len(xx1,yy1,xx2,yy2)** where **ll** is the output variable and **xx1**, **yy1**, **xx2** and **yy2** are the input variables. Note that their names need not match the corresponding names in the function definition.

Note the semicolons (;) used at the end of the executable statements which suppress the echo of intermediate results. You can remove them if you need to debug the function when the results are not matching the expected results. Alternately, use the function **disp()** to print out intermediate results within a function to debug it and remove it once the bug is eliminated.

To try out the function, load it into Scilab by first saving the contents to a disk file (File > Save) and then clicking on 'Load into Scilab'. If there are no syntax errors in your function definition, it will be loaded into Scilab workspace and this can be verified with the command **whos -type function** and searching for the name of your function, namely, **len**. To use the function enter the following command: **l=len(1,4,10,6)**.

However, if there are syntax errors, the line on which the error occurs is displayed and you will have to remove the syntax errors and repeat the above steps.

Tutorial 9 – Miscellaneous Commands

Some commands related to operations on disks are important and they are listed below. They are important when you want to change the directory in which your function files are stored or from where they are they are to be read.

pwd	Prints the name of the current working directory
getcwd()	Same as **pwd**.
chdir('dir')	Changes the working directory to a different disk location named **dir**.

It is possible to save all the variables in the Scilab Workspace to a disk file so that you can quickly reload all the variables and functions from a previous session and continue from where you left off. The commands used for this purpose are:

save('pf.bin')	Saves entire contents of the Scilab workspace (variables and functions) in the file **pf.bin** in the current working directory.
load('pf.bin')	Restores contents of the Scilab workspace from the file **pf.bin** in the current working directory.
save('pf.bin', xy)	Saves only the variable **xy** of the Scilab workspace in the file **pf.bin** in the current working directory.

It is possible to determine the size of variables in the Scilab workspace with the following command:

size(xy)	Returns a 1x2 matrix containing the number of rows and columns in the variable **xy**.
length(xy)	Returns the number of elements in **xy** (rows multiplied by columns).

Scilab can open files on disk and a number of functions are available for opening, closing reading and writing disk files. The following lines of code illustrate how this can be accomplished:

```
-->n=10; x=25.5; xy=[100 75;0 75; 200, 0];
-->fd=file("ex01.dat", "w"); // Opens a file ex01.dat for writing
-->mfprintf(fd, "n=%d, x=%f\n", n, x);
-->mfprintf(fd, "%12.4f\t%12,4f\n", xy);
-->mclose(fd);
```

You can now open the file **ex01.dat** in a text editor, such as notepad and see its contents. You will notice that the commands are similar to the corresponding commands in C, namely, **fopen()**, **fprintf()** and **fclose()**. Note that in the second command, the format string must be sufficient to print one full row of the matrix **xy**. The sequence of operations must always be, open the file and obtain a file descriptor, write to the file using the file descriptor and close the file using the file descriptor.

SCILAB License
****

1- Preface
****
The aim of this license is to lay down the conditions enabling you to use, modify and circulate the SOFTWARE. However, INRIA and ENPC remain the authors of the SOFTWARE and so retain property rights and the use of all ancillary rights.

2- Definitions
****
The SOFTWARE is defined as all successive versions of SCILAB software and their documentation that have been developed by INRIA and ENPC.

SCILAB DERIVED SOFTWARE is defined as all or part of the SOFTWARE that you have modified and/or translated and/or adapted.

SCILAB COMPOSITE SOFTWARE is defined as all or a part of the SOFTWARE that you have interfaced with a software, an application package or a toolbox of which you are owner or entitled beneficiary.

3- Object and conditions of the SOFTWARE license
****
a) INRIA and ENPC authorize you free of charge, to reproduce the SOFTWARE source and/or object code on any present and future support, without restriction, providing the following reference appears in all the copies: Scilab (c)INRIA-ENPC.

b) INRIA and ENPC authorize you free of charge to correct any bugs, carry out any modifications required for the porting of the SOFTWARE and to carry out any usual functional modification or correction, providing you insert a patch file or you indicate by any other equivalent means the nature and date of the modification or the correction, on the corresponding file(s) of the SOFTWARE.

c) INRIA and ENPC authorize you free of charge to use the SOFTWARE source and/or object code, without restriction, providing the following reference appears in all the copies: Scilab (c)INRIA-ENPC.

d) INRIA and ENPC authorize you free of charge to circulate and distribute, free of charge or for a fee, the SOFTWARE source and/or object code, including the SOFTWARE modified in accordance with above-mentioned article 3 b), on any present and future support, providing:

   * the following reference appears in all the copies: Scilab (c)INRIA-ENPC.

   * the SOFTWARE is circulated or distributed under the present license.

   * patch files or files containing equivalent means indicating the nature and the date of the modification or the correction to the SOFTWARE file(s) concerned are freely circulated.

4- Object and conditions of the DERIVED SOFTWARE license
****
a) INRIA and ENPC authorize you free of charge to reproduce and modify and/or translate and/or adapt all or part of the source and/or the object code of the SOFTWARE, providing a patch file indicating the date and the nature of the modification and/or the translation and/or the adaptation and the name of their author in the SOFTWARE file(s) concerned is inserted. The SOFTWARE thus modified is defined as DERIVED SOFTWARE. The INRIA authorizes you free of charge to use the source and/or object code of the SOFTWARE, without restriction, providing the following reference appears in all the copies: Scilab (c)INRIA-ENPC.

b) INRIA and ENPC authorize you free of charge to use the SOFTWARE source and/or object code modified according to article 4-a) above, without restriction, providing the following reference appears in all the copies: "Scilab inside (c)INRIA-ENPC".

c) The INRIA and the ENPC authorize you free of charge to circulate and distribute for no charge, for non-commercial purposes the source and/or object code of DERIVED SOFTWARE on any present and future support, providing:

   * the reference " Scilab inside (c)INRIA-ENPC " is prominently mentioned;

   * the DERIVED SOFTWARE is distributed under the present license;

   * the recipients of the distribution can access the SOFTWARE code source;

* the DERIVED SOFTWARE is distributed under a name other than SCILAB.

d) Any commercial use or circulation of the DERIVED SOFTWARE shall have been previously authorized by INRIA and ENPC.

5- Object and conditions of the license concerning COMPOSITE SOFTWARE
****

a) INRIA and ENPC authorize you to reproduce and interface all or part of the SOFTWARE with all or part of other software, application packages or toolboxes of which you are owner or entitled beneficiary in order to obtain COMPOSITE SOFTWARE.

b) INRIA and ENPC authorize you free, of charge, to use the SOFTWARE source and/or object code included in the COMPOSITE SOFTWARE, without restriction, providing the following statement appears in all the copies: "composite software using Scilab (c)INRIA-ENPC functionality".

c) INRIA and ENPC authorize you, free of charge, to circulate and distribute for no charge, for purposes other than commercial, the source and/or object code of COMPOSITE SOFTWARE on any present and future support, providing:

* the following reference is prominently mentioned: "composite software using Scilab (c)INRIA-ENPC functionality ";

* the SOFTWARE included in COMPOSITE SOFTWARE is distributed under the present license ;

* recipients of the distribution have access to the SOFTWARE source code;

* the COMPOSITE SOFTWARE is distributed under a name other than SCILAB.

e) Any commercial use or distribution of COMPOSITE SOFTWARE shall have been previously authorized by INRIA and ENPC.

6- Limitation of the warranty
****

Except when mentioned otherwise in writing, the SOFTWARE is supplied as is, with no explicit or implicit warranty, including warranties of commercialization or adaptation. You assume all risks concerning the quality or the effects of the SOFTWARE and its use. If the SOFTWARE is defective, you will bear the costs of all required services, corrections or repairs.

7- Consent
****

When you access and use the SOFTWARE, you are presumed to be aware of and to have accepted all the rights and obligations of the present license.

8- Binding effect
****

This license has the binding value of a contract. You are not responsible for respect of the license by a third party.

9- Applicable law
****

The present license and its effects are subject to French law and the competent French courts.