

Practice 0 of Algebra

Introduction to Scilab

Contents

1	Introduction	2
2	Introduction of data in Scilab	2
3	Help	5
4	How to save and recover sessions	5
5	Operations with scalars	6
6	Operations with matrices	7
6.1	Basic operations	7
6.2	Other operations with matrices	8
6.3	Element-wise operations	8
6.4	Special types of matrices	9
7	Internal manipulation of matrices	9
8	Polynomials	11
9	Decision and loop structures with Scilab	12
9.1	The conditional if	12
9.2	The loop for	13
9.3	The loop while	14
10	Funtions with Scilab	14
11	End of session	16

1 Introduction

The programme Scilab works fundamentally with matrices with real, complex, or boolean coefficients. In this practice we will describe the basic operation of the programme and all the basic aspects about Scilab that the students should know in order to carry out the practices of Discrete Mathematics and Linear Algebra.

Scilab is a free programme. In particular, it can be used and distributed freely and the source code can be consulted and modified under the license conditions. The programme can be downloaded from <http://www.scilab.org>. These practices, with slight modifications, can be carried out with other programmes of similar syntax, for example free systems, like Octave, or commercial systems like Matlab.

We will begin by indicating the basic operation of the programme, by showing the different ways of introducing matrices in Scilab and how to manipulate them. Next we will describe how to carry out with Scilab the main mathematical operations between matrices. We will finish the practice by showing some of the most usual decision and loop structures of Scilab.

The Scilab programme is available for many platforms. Among them we can emphasise GNU/Linux, Microsoft Windows, and MacOSX.

Among the different practices we will be introducing more operators, commands, and basic functions as soon as we will need them and we will describe its operation with more detail. What we intend to do here is to give the main basic operators, commands, and functions in order to look for them later more easily.

The examples we will show are developed with the version 4.1.2 of Scilab under the operating system GNU/Linux with an Intel® Core™2 Duo E8400 processor at 3 GHz. The execution of Scilab in other platforms or operating systems can give slightly different results. Hence we must pay attention to this matter.

In these bulletins we will use the following typographical conventions:

- The input from the keyboard will appear in a bold typewriter font (for example, **entry**).
- The results or the output from Scilab will appear in a medium weight typewriter font (for example output).
- The generic expressions in the description of the Scilab commands will appear with a slanted typewriter font (for example, in *name=expression*, we will understand that *name* and *expression* must be replaced by suitable strings in Scilab).

2 Introduction of data in Scilab

The program Scilab can be run by writing in the command line

scilab

or, in a window graphic system, with a double click (or a single click) on the corresponding icon, or by looking for it in the suitable menu. A window will appear with the following text:

scilab-4.1.2

Copyright (c) 1989-2007
Consortium Scilab (INRIA, ENPC)

Startup execution:
loading initial environment

-->


If the programme Scilab is invoked with the option `-nw`, the text appears on the terminal without any new window. This option is suitable if no graphical capacities are needed or only a text terminal is available.

The prompt `-->` tells us that we can introduce new commands.

Vectors are introduced by writing in order their components separated by a comma (,) or by a blank and by enclosing the whole expression between brackets ([]). *Scalars* are introduced directly, without necessity of brackets. In the case of decimal numbers, the integer part is separated from the decimal one by a point (.).

For example, the scalar $a = 2$ is introduced by typing

a=2

and when pressing the key  it appears

-->**a=2**

a =

2.

-->

The vector $\vec{v} = (1, -1, 3)$ is introduced as

-->**v=[1 -1 3]**

v =

1. - 1. 3.

or as

-->**v=[1,-1,3]**

v =

1. - 1. 3.

Note: In some versions of this programme there appear some exclamation signs around the matrices.

A matrix of size $m \times n$ is a collection of $m \times n$ numbers organised in rows and n columns. If A is a matrix, $a_{i,j}$ denotes the term occupying the i th row and the j th column and the matrix and we can write that $A = (a_{i,j})$.

In Scilab, the elements of a matrix are introduced between brackets [], by writing in order the elements of each row separated by commas or blanks and by separating the rows by a semicolon (;) or the character \leftarrow (carriage return). There is no need to declare previously the size of the matrices.

For example, the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

can be introduced in Scilab as

A=[1 2 3;4 5 6;7 8 9]

or as

**A=[1 2 3
4 5 6
7 8 9]**

or also as

**A=[1 2,3;4 5 6
7 8 9]**

When introducing some of these expressions and pressing \leftarrow we obtain

```
A  =  
  
1.      2.      3.  
4.      5.      6.  
7.      8.      9.
```

Example 1. Introduce the matrix A of the previous example. The screen of Scilab should be more or less like the one shown on Figure 1.


Figure 1: Introduction of a matrix in Scilab

The commands of Scilab have the form

variable=expression

or simply

expression

The variable names must begin by a letter, followed by more letters or digits, up to a maximum of 25 characters. Scilab does not carry out the sentences until  is pressed. If a sentence ends with a semicolon (;), the result is not shown on the screen, but the result is stored in the memory. Scilab distinguishes between uppercase and lowercase letters. Of course, no other commands of Scilab can be used as variable names.

It is possible to include several commands in the same command line by separating them by a comma (,), to show the results, or by a semicolon (;), to hide the results. In the same way, if a sentence is too long and it does not fit in one line, it is possible to separate it in several lines by writing at the end of each line three or more dots (...).

It is possible to recall previous commands with the cursor (arrow) keys.

If an expression is assigned to a variable, this one remains stored as a working variable during all the session (or until it is assigned another value or cleared with the command **clear**, which erases all variables, or **clear** followed by the name of one or more variables to erase them). In this way, to recover this expression in any moment, it will be enough to write the name of the variable. If we use the second form of the commands, the result is stored into the variable **ans**. This allows us to recall the result of the immediately previous operation if we have not assigned it to another variable.

The commands **who** and **whos** tell us which are the variables of the workspace.

Comments are introduced preceded by the characters // (two slashes) out of a string. Everything after this mark is not executed.

3 Help

We can find help about any of the commands or functions of Scilab by typing **help** or, if Scilab is used in a graphical environment, by clicking with the mouse over the button **Help**. We can also find help about a specific command by typing

help *command*

4 How to save and recover sessions

It is possible to save all variables of the workspace with the command **save**. For example

```
-->save('file.dat')
```

saves all variables in the file `file.dat`, while

```
-->save('file.dat',A,b,c)
```

saves the variables `A`, `b` and `c` in the file. In order to recover the contents of the files, the command **load** is used with the syntax

```
-->load('file.dat','A','b','c')
```

In order to save the content of a work session, the command **diary** can be used:

```
-->diary('file.txt')
```

starts recording all work session in the text file `file.txt`. To finish the recording we will use

```
-->diary(0)
```

If the file exists previously, it will be erased. This is the reason we must pay a lot of attention when writing the name of the file. A good strategy is to include the date (and, if needed, the time) in the file name. For instance,

```
-->diary('md221010.txt');
```

5 Operations with scalars

Scilab is able to work with real and complex scalars. In order to work with decimal numbers, the integer part and the decimal part are separated with a *point* (.). The operations addition, subtraction, and multiplication are obtained with the operators **+**, **-** and *****, respectively. The power of a scalar a to a scalar p is written as a^p . Square roots can be also obtained with the operator **sqrt**. Roots of other indices can be represented as powers with a fraction as an exponent, as for instance $2^{(1/3)}$ for $\sqrt[3]{2}$. For the division, there are two operators, **/** and ****. For example, a/b means a divided by b , while $a\b b$ means b divided by a .

The hierarchy of the mentioned operators is as usual: from highest (the last in execution) to lowest (the first in execution), it is the following:

+ **-** ***** **/** **** **^**

In order to change the execution order, we can use parentheses.

Example 2. Let us compute with Scilab the value of the expression $2x^3y + \sqrt{y^5z}$, with $x = 1.5$, $y = 3.7$, $z = 9$.

First we introduce the variables:

```
-->x=1.5; y=3.7; z=9;
```

Next, we input the expression:

```
-->2*x^3*y+sqrt(y^5*z)
ans =
```

103.97472

The result is 103.97472.

6 Operations with matrices

Next we will show how Scilab executes the usual matrix operations, as well as other functions which will be useful for some practices. Since vectors are just matrices with a unique row, the operations we will describe will be also valid for vectors.

6.1 Basic operations

Addition and subtraction of matrices: To sum two matrices we will use the operator $+$ and to subtract two matrices, the operator $-$. Both matrices must have the same size (that is, the same number of rows and columns). If we want to add or subtract two matrices which do not have the same size, Scilab will signal an error.

Product of a scalar by a matrix: It is carried out with the operator $*$.

Product of matrices: It is also carried out with the operator $*$. The number of columns of the first matrix must coincide with the number of rows of the second matrix. If we want to multiply two matrices which do not satisfy this condition, an error message is obtained.

Power of a matrix: To compute the product of a matrix A by itself n times, we will use the operator $^$. For example A^n . If the matrix is not square, Scilab will give an error message.

Example 3. Now we will show some matrix calculations.

```
-->X=[1 2;3 -1]
X =

    1.    2.
    3.   -1.

-->Y=[0 1]
Y =

    0.    1.

-->X*Y
!--error 10
inconsistent multiplication

-->X+Y
!--error 8
inconsistent addition
```

```
-->X-Y
      !--error 9
inconsistent subtraction
```

```
-->Y*X
ans  =

      3.  - 1.
```

6.2 Other operations with matrices

Inverse of a matrix: Scilab calculates the inverse of a square matrix (if it exists) with the function `inv()`. If the introduced matrix does not have an inverse, an error message will appear. In the practices of Linear Algebra we will study with detail how this function works.

Transposed of a matrix: The transposed matrix of a given matrix is the matrix obtained by interchanging rows by columns. Scilab computes it with the operator apostrophe or single quote mark (`'`), for example, `A'`.

6.3 Element-wise operations

In some occasions it could be useful to carry out element-wise operations with matrices which do not correspond to any mathematical operation between matrices. We will next show some of these operations.

Addition of one scalar to all elements of a matrix: It is carried out by putting the operator `+` between the matrix and the scalar. For example, `A+1` is the matrix obtained when we add 1 to each element of the matrix `A`.

Element-wise product and quotient: To multiply element by element the operator `.*` is used. For example, `A.*B`. Both matrices must have the same size to carry out this operation. Analogously we can compute element-wise quotients of matrices with the operator `./`.¹

Element-wise power: To raise to a power all elements of a matrix, we can use the operator `.^`. The matrix can have any size and the exponent can be any number.

¹This last operator can be also used, for example, to divide a number by all elements of a matrix. In this case, if the first number is an integer, it could be needed to separated from the operator with a blank (for example, `1 ./A`) or by writing the decimal point (`1./A`) because `1./A` would be interpreted as the quotient between the real number `1.` and the matrix `A`.

6.4 Special types of matrices

In Scilab some special matrices have been implemented. For example, **zeros**(m, n) is the null matrix of m rows and n columns, **ones**(m, n) is the matrix of m rows and n columns composed only by ones, and **eye**(m, n) is the matrix of m rows and n columns with ones in the diagonal and zeros in the other entries (when $m=n$, it is the identity matrix of order n , I_n). Analogously, if mat is a matrix, **zeros**(mat), **ones**(mat) and **eye**(mat) give the matrices of the same dimensions as mat composed, respectively, by zeros, ones, and ones in the diagonal and zeros in the other positions.

7 Internal manipulation of matrices

Suppose that we have introduced a matrix A in our Scilab session and we want to change the element of the row i and the column j by the value *expression*. Then we will use the order

$$A(i, j) = \text{expression}$$

This action modifies the matrix A. Therefore, if we do not want to lose the original matrix A original, we can copy it to another one by typing, for example,

```
-->B=A
```

and carrying out later the modification to the matrix B , for instance,

```
-->B(2,3)=10
```

If the indices are greater than the matrix size, Scilab adds enough rows or columns to allocate the new elements. It is worth taking into account that Scilab identifies the matrices 1×1 with scalars.

Scilab allows the introduction of matrices defined *by blocks*. Suppose that A and B are matrices with the same number of rows. We can construct the matrix

$$C = (A \ B)$$

obtained by putting the columns of B on the right of the ones of A . This matrix is introduced in Scilab by writing

```
-->[A,B]
```

or

```
-->[A B]
```

Analogously, if A and B are two matrices with the same number of columns, we can construct the matrix

$$D = \begin{pmatrix} A \\ B \end{pmatrix}$$

by putting the rows of B below the rows of A . This matrix could be introduced in the form

```
-->[A;B]
```

or as

```
-->[A
--> B]
```

In the same way, we can add to an $m \times n$ matrix A a new row by typing

```
-->[A; [a1 a2 ... an]]
```

and a new column by typing

```
-->[A,[a1;a2; ...; am]]
```

This way of working by blocks is valid for any number of matrices whenever the number of rows and columns are compatible.

Example 4. Let us consider the matrices

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \quad B = \begin{pmatrix} 7 & 8 & 2 \\ 4 & 1 & 6 \\ 0 & 0 & 1 \end{pmatrix}.$$

```
-->A=[1 2 3
--> 4 5 6
--> 7 8 9]
A =
```

```
1.    2.    3.
4.    5.    6.
7.    8.    9.
```

```
-->B=[7 8 2
--> 4 1 6
--> 0 0 1]
B =
```

```
7.    8.    2.
4.    1.    6.
0.    0.    1.
```

```
-->C=[A,B]
C =
```

```
1.    2.    3.    7.    8.    2.
4.    5.    6.    4.    1.    6.
7.    8.    9.    0.    0.    1.
```

4.	5.	6.	4.	1.	6.
7.	8.	9.	0.	0.	1.

-->**D=[A;B]**
D =

1.	2.	3.
4.	5.	6.
7.	8.	9.
7.	8.	2.
4.	1.	6.
0.	0.	1.

Once introduced a matrix *matrix*, it is possible to work with submatrices extracted from it, by using expressions as the next ones:

- *matrix(i,j)* for the element occupying the position corresponding to the row *i* and the column *j* of *matrix*,
- *matrix(:,j)* for the column *j* of *matrix*,
- *matrix(i,:)* for the row *i* of *matrix*,
- *matrix(r:s,:)* for the submatrix formed by the rows between the *r* and the *s* of *matrix*,
- *matrix(:,r:s)* for the submatrix formed by the columns between the *r* and *s* of *matrix*,
- *matrix([r s],:)* for the submatrix formed by the rows *r* and *s* of *matrix*, and
- *matrix(:, [r s])* for the submatrix formed by the columns *r* and *s* of *matrix*.

Let us note that these expressions allow us also to modify a matrix which has been introduced. Hence, for example, if we want to modify the row *i* of *matrix*, it is enough to type *matrix(i,:)=new row* and the program will return the matrix *matrix* modified. We must take into account that the dimensions of the submatrix and the new matrix must be compatible. Besides, we can assign to all elements of a submatrix the same element with the same syntax.

As above, we must take care if we do not want to lose the initial matrix.

8 Polynomials

To define a polynomial variable in the indeterminate **s**, we can define the variable **s** in the form

```
-->s=poly(0,"s")
```

```
s =
```

```
s
```

```
-->p=3+4*s+s^2
```

```
p =
```

```
          2  
3 + 4s + s
```

The roots of a polynomial are calculated with the function **roots**. For the polynomial **p** we have introduced we would have

```
-->roots(p)
```

```
ans =
```

```
- 1.
```

```
- 3.
```

In order to evaluate a polynomial in a particular value we will use the function **horner**. For instance

```
-->horner(p,2)
```

```
ans =
```

```
15.
```

In the practices of Linear Algebra we will work with polynomials.

9 Decision and loop structures with Scilab

Scilab is a programming language in which it is possible to use conditional commands and to execute different loops. For future reference we describe some of these commands. All these commands end with the reserved word **end**.

9.1 The conditional **if**

The command **if** is used to evaluate a logical expression and execute a series of commands if the expression is true.

The syntax is:

```

if condition then commands
elseif condition then commands
...
else commands
end

```

where the parts **elseif** and **else** are optional, and there may be more than one **elseif** clause. The word **then** can be replaced by a carriage return or a comma and *must be located always in the same line as the corresponding if or elseif command*.

If the condition of the line with **if** is true, the following commands are executed. Otherwise, it is executed the first of the series of **elseif** commands for which the condition is true, and if none of them is true, the commands of the **else** clause are executed.

In a future practice we will study with detail the comparison operators and the logical operators. We present an easy example.

```

-->x=-4; if x<0 then r=-x,elseif x==0 then r=0,else r=x,end
r =

4.

```

```

-->x=5; if x<0 then r=-x,elseif x==0 then r=0,else r=x,end
r =

5.

```

Scilab also has a command **select/case** which can be a useful alternative when an expression has only a small number of interesting values. For more information, type **help select** in Scilab.

9.2 The loop for

The loop **for** is used to execute some commands for all columns of a matrix. Its syntax is

```

for variable=expression do command, ... command,end

```

Here *expression* is usually a matrix and the orders are executed for all columns of the matrix. The usual formats for this expression are

```

start:step:end

```

or

```

start:end

```

Example 5. With the following example we can write the squares of all odd numbers between 1 and 9:

```
-->for x=1:2:9 do [x,x^2],end
ans =

    1.    1.
ans =

    3.    9.
ans =

    5.   25.
ans =

    7.   49.
ans =

    9.   81.
```

With the following command we compute the sum of the squares of the 10 first natural numbers. The usage of semicolon instead of comma hides the output of all intermediate values.

```
-->sum=0;for x=1:10 do sum=sum+x^2;end;sum
sum =

    385.
```

9.3 The loop **while**

This loop is used to execute some commands while a certain boolean expression is true. Its syntax is

```
while condition do commands,...[,else commands], end
```

The word **do** can be replaced by **then**, by a comma, or a carriage return. The word **do** or **then** must be in the same line as **while**. The **else** clause is executed when *condition* becomes false.

10 Functions with Scilab

We have seen some functions like **zeros** or **ones**, which allow us to construct Scilab objects from its parameters or arguments. The syntax of a Scilab function is

```
function(arg1, arg2...)
```

to give a result, or

`[var1, var2...]=function(arg1, arg2...)`

for functions returning several results, assigned to the variables *var1*, *var2*...

In order to define a function in Scilab, we can use the keywords **function** and **endfunction**. Their syntax is

```
function  argsoutput=nameoffunction(argsinput)
           commands
endfunction
```

where *argsinput* is a list of variable identifiers which will be assigned in order to the parameters and *argsoutput* is an identifier or a list of identifiers of variables separated by commas and enclosed between brackets. The list of commands must contain some assignments of these variables to compute the result.

For example the following function computes the square of a given number:

```
-->function y=squared(x)
-->y=x*x
-->endfunction
```

```
-->z=squared(3)
z  =
```

9.

As we see, the output argument is **y** and the input argument is **x**. The value of the function is the value of **y** when the definition of the function ends after substituting the parameter **x** by the used argument.

The following function admits two parameters, **x** and **y**, and returns two results, which correspond to the sum and the difference of its arguments:

```
-->function [sum, dif]=sumdif(x,y)
-->    sum=x+y
-->    dif=x-y
-->endfunction
```

```
-->[m,n]=sumdif(5,3)
n  =
```

2.

m =

8.

We observe that the variable **m** remains assigned to the sum of both numbers and the variable **n** remains assigned to the difference between both numbers.

It is interesting to store one or several functions in a text file, in order to use them in several sessions. To execute all commands (not only the definitions of functions) of a text file *file.sci*, we can use the command **exec**:

```
-->exec('file.sci')
```

The graphical version of the programme Scilab includes a function editor, accessible from the menu, which allows the execution of the selected code.

11 End of session

To finish the current Scilab session, we type the command **exit**. If we are working in a window environment, we can also exit by closing the Scilab window.