MATLAB

Chapter III : Introduction to programming with Matlab

We have seen so far how to use Matlab to perform commands or to evaluate expressions by writing them in the command line (After prompt >>), so the commands used are usually written as a single statement (possibly on a single line).
However, there are problems whose description of their solutions requires several instructions, which require the use of several lines. For example, searching for the roots of a second-degree equation (taking into account all possible cases).
A collection of well-structured instructions for solving a given problem is called a program. In this part of the course, we will present the mechanisms of writing and executing programs in Matlab.

1. General :

1.1 Comments :

Comments are explanatory sentences ignored by Matlab and intended for the user to help him understand the part of the commented code.
In Matlab a comment starts with the % symbol and occupies the rest of the line.
For example :

```
>> A=B+C ;            % Give A the value of B+C
```

1.2 Writing long expressions :

If a long expression cannot be written in a single line, it can be divided into several lines by putting at the end of each line at least three points.
Example :

```
>> (sin(pi/3)^2/cos(pi/3)^2)-(1-2*(5+sqrt(x)^5/(-2*x^3-x^2)^1+3*x)) ;
```

This expression can be rewritten as follows :

```
>> (sin(pi/3)^2/cos(pi/3)^2)- ...  ↵
>> (1-2*(5+sqrt(x)^5 .....              ↵
>> /(-2*x^3-x^2)^1+3*x)) ;              ↵
```

1.3 Reading data in a program (Inputs) :

To read a value given by the user, it is possible to use the **input** command, which has the following syntax :

```
variable = input ('an indicative sentence')
```

- The value deposited by the user will be put in this variable
- A sentence to help the user know what to enter

When Matlab executes such an instruction, the indicative phrase will be displayed to the user waiting for the latter to enter a value.

for example :

```
>> A = input ('Enter a whole number : ')        ↵
   Enter a whole number : 5                     ↵
   A =
      5
>>

>> A = input (' Enter a whole number : ');      ↵
   Enter a whole number : 5                     ↵
>>

>> B = input ('Enter a vector line : ')         ↵
   Enter a vector line : [1:2:8,3:-1:0]         ↵
   B =
      1     3     5     7     3     2     1     0
```

1.4 Writing data in a program (Outputs) :

We have already seen that Matlab can display the value of a variable by typing only the name of this last. For example :

```
>> A = 5 ;
>> A                  % Ask Matlab to display the value of A
   A =
       5
```

With this method, Matlab writes the name of the variable (A) then the sign (=) followed by the desired value. However, there are cases where only the value of the variable is displayed (without the name and without the sign =).
To do this, we can use the **disp** function, which has the following syntax:**disp(object)**
The value of the object can be a number, a vector, a matrix, a string or an expression.
It is reported that with an empty vector or matrix, **disp** displays nothing.
Example :

```
>> disp(A)                    % Display the value of A without 'A = '
```

```
     5
>> disp(A);                  % Semicolon has no effect
     5
>> B                         % Display vector B by the classical method
   B =
      1     3     5     7     3     2     1     0
>> disp(B)                   % Display the vector B without 'B = '
      1     3     5     7     3     2     1     0
>> C = 3 :1 :0               % Creating an empty C vector
   C =
   Empty matrix: 1-by-0
>> disp(C)                   % disp displays nothing if vector is empty
>>
```

2. Logical expressions :

2.1 Logical operations :

the comparison operation	its meaning
==	equality
~=	inequality
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
logical operations	**its meaning**
&	the bitwise and
|	the logical OR
~	the logical negation

In Matlab a logical variable can take the values 1(true) or 0(false) with a small rule that assumes that :

1) Any value equal to 0 will be considered false **(= 0 ⇒ false)**
2) Any value other than 0 will be considered true **(≠ 0 ⇒ true)**.

The following table summarizes the operation of logical operations :

a	*b*	*a & b*	*a | b*	*~a*
1 *(*true*)*	1*(*true*)*	1	1	0
1 *(*true*)*	0 *(*false*)*	0	1	0
0*(*false*)*	1 *(*true*)*	0	1	1
0 *(*false*)*	0 *(*false*)*	0	0	1

For example :

```
>> x=10;
```

```
>> y=20;
>> x < y            % displays 1 (true)
      ans =
            1
>> x <= 10          % displays 1 (true)
      ans =
            1
>> x == y           % displays 0 (false)
      ans =
            0
>> (0 < x) & (y < 30)           % displays 1 (true)
      ans =
            1
>> (x > 10) | (y > 100)          % displays 0 (false)
      ans =
            0
>> ~(x > 10)        % displays 1 (true)
      ans =
            1
>> 10 & 1                       % 10 is considered true therefore 1 & 1 = 1
      ans =
            1
>> 10 & 0                       % 1 & 0 = 1
      ans =
            0
```

2.2 Matrix comparison :

The comparison of vectors and matrices differs somewhat from scalars, hence the usefulness of the two functions 'isequal' and 'isempty' (which allow to give a concise answer for comparison).

Function	Description
isequal	tests whether two (or more) matrices are equal (having the same elements everywhere). Returns 1 if so, and 0 otherwise.
isempty	tests if a matrix is empty (contains no elements). Returns 1 if it is, and 0 otherwise.

To better perceive the impact of these functions follow the following example :

```
>> A=[5,2;-1,3]          % create the matrix A
      A =
            5     2
            -1     3
>> B=[5,1;0,3]           % create the matrix B
      B =
            5     1
            0     3
>> A==B                  % test whether A=B ? (1 or 0 depending on the position)
```

```
      ans =
            1     0
            0     1
>> isequal(A,B)          % Test if A and B are equal (the same)
      ans =
            0
>> C=[] ;                % Create the empty matrix C
>> isempty(C)            % Test if C is empty (true = 1)
      ans =
            1
>> isempty(A)            % Test if A is empty (displays false = 0)
      ans =
            0
```

3. Flow control structures

Flow control structures are instructions for defining and manipulating the order of execution of tasks in a program. They offer the possibility to perform different treatments depending on the state of the program data, or to perform repetitive loops for a given process. Matlab has eight flow control structures, namely :

- **if**
- **switch**
- **for**
- **while**
- continue
- break
- try - catch
- return

We expose the first four : (**if, switch, for** and **while**)

3.1 the if statement :

The if statement is the simplest and most widely used flow control structure. It guides the execution of the program according to the logical value of a condition.Its general syntax is as follows :

```
if(condition)
      instruction_1
      instruction_2
      . . .
      Instruction_N
end
```

oubien

```
if (condition)
      ensemble d'instructions 1
else
      ensemble d'instructions 2
end
```

If the condition is evaluated at vari, the instructions between the **if** and the end will be executed, sinonelles will not (or **if** an else exists the instructions between the **else** and the end will be executed). If it is necessary to check several conditions instead of only one, **elseif** clauses can be used for each new condition, and in the end an **else** can be set in case no condition has been evaluated to true.Here is the general syntax:

```
if (expression _1)
       set of instructions 1
elseif (expression_2)
       set of instructions 2
              ....
elseif (expression_n)
       set of instructions n
else
       set of instructions if all expressions were false
end
```

For example, the following program defines you according to your age :

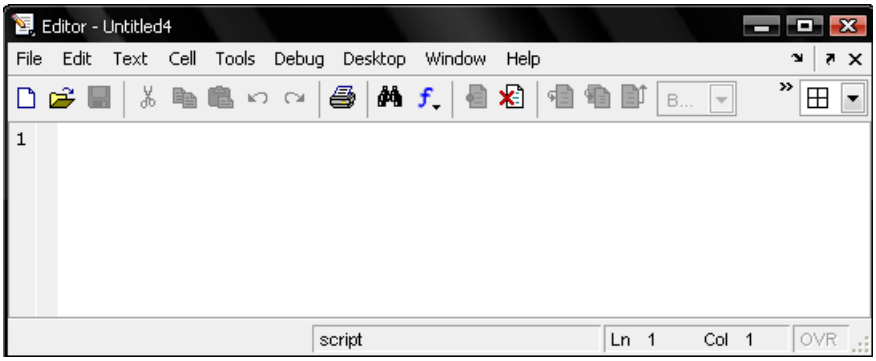
```
>> age = input('Enter your age : '); ...
 if (age <2)
      disp('You are a fool')
   elseif (age <13)
      disp('You are a child')
   elseif (age < 18)
      disp ('You are an adolescent')
   elseif (age <60)
      disp ('You are unadulterated)
   else
      disp ('You are an old man)
   end
```

As you can see, writing a Matlab program directly after the command prompt (the prompt >>) is a bit unpleasant and annoying.

A more convenient method is to write the program to a separate file, and call that program (if necessary) by typing the file name in the command prompt.
This approach is defined in Matlab by M-Files, which are files that can contain data, programs (scripts) or functions that we develop.

To create an M-Files simply type the command edit, or simply go to the menu: File New M-Files (or click on the icon ).

In any case an editing window like this will appear :



All you have to do is write your program in this window and save it with a name (for example: 'First_Program.m'). It is reported that the extension of the M-Files files is always '.m'.
Now, if we want to run our program, just go to the usual command prompt (>>) and then type the name of our file (without the '.m') like this:

```
>> First_Program          ↵
```

And the program will start running immediately.

To return to the editing window (after closing it) simply enter the command :

```
>> edit First_Program          ↵
```

Example :

Let's create a program that finds the roots of a second-degree equation designated by :
$ax^2+bx+c=0$. Here is the M-File that contains the program (it is saved with the name 'Equation2deg.m')

```
% Programme de résolution de l'équation a*x^2+b*x+c=0

a = input ('Entrez la valeur de a : ');            % lire a
b = input ('Entrez la valeur de b : ');            % lire b
c = input ('Entrez la valeur de c : ');            % lire c

delta = b^2-4*a*c ;                          % Calculer delta
if delta<0
disp('Pas de solution')                      % Pas de solution
elseif delta==0
   disp('Solution double : ')                % Solution double
   x=-b/(2*a)
else
   disp('Deux solutions distinctes: ')       % Deux solutions
x1=(-b+sqrt(delta))/(2*a)
   x2=(-b-sqrt(delta))/(2*a)
end
```

If we want to run the program, just type the name of the program:

```
>> Equation 2 of g                    ↵
       Enter the value of a : -2 ↵
       Enter the value of b : 1        ↵
       Enter the value of c : 3        ↵
       Two solutions:
       x1 =
              -1
       x2 =
              1.5000
```

Thus, the program will be executed following the instructions written in its M-File. If an instruction is terminated by a semicolon, then the value of the variable concerned will not be displayed, but if it ends with a comma or a line break, then the results will be displayed.

Note : Il et Note: There is the predefined **solve** function in Matlab to find the roots of an equation (and much more). If we want to apply it to our example, just write:

```
>>solve('-2*x^2+x+3=0','x')
 years =
 -1
        3/2
```

3.2 The switch statement :

The **switch** statement executes groups of statements based on the value of a variable or expression. Each group is associated with a **case** clause that defines whether or not this group should be executed according to the equality of the value of this box with the evaluation result of the **switch** expression.If not all **cases** have been accepted, it is possible to add an **otherwise** clause that will be executed only if no box is executed.

Therefore, the general form of this instruction is:

```
switch (expression)
       case value_1
              Instruction group 1
       case value_2
              Instruction group 2
              . . .
       case value_n
              Instruction group n
       otherwise
              Package instructionswhere the boxes have failed
end
```

Example :

```
 x = input ('Enter un number: ') ;
 switch(x)
```

```
        case 0
              par ('x = 0 ')
        case 10
              par('x = 10 ')
        case 100
              par('x = 100 ')
        otherwise
              par('x n'' is not s 0 or 10 or 100')
end
```

The execution will give:

```
Enter a number : 50      ↵
x is not 0 or 10 or 100
```

3.3 The for statement :

The **for** statement repeats the execution of a group of instructions a specified number of times. It has the following general form:

```
for variable = expression_vector
      instruction group
end
```

vecteur The expression_vector corresponds to the definition of a vector: *start: not: end or start: end*

The variable will go through all the elements of the vector defined by the expression, and for each it will execute the group of instructions.

Example :

In the following table, we know three forms of the for statement with the Matlab result:

The instruction for	`for i = 1 : 4`
	`  j=i*2 ;`
	`  disp(j)`
	`end`
		`for i = 1 : 2 : 4`
		`  j=i*2 ;`
		`  disp(j)`
		`end`
			`for i = [1,4,7]`
			`  j=i*2 ;`
			`  disp(j)`
			`end`

the résultat of the exécution	2 4 6 8	2 6	2 8 14

3.4 The while statement :

The **while** statement repeats the execution of a group of statements an indeterminate number of times depending on the value of a logical condition. It has the following general form:

```
while (condition)
      set of instructions
end
```

As long as the expression of **whil**e is evaluated to true, the instructionsset will run in a loop.

Example :

```
a=1 ;
while (a~=0)
      a = input ('Enter unnombre (0 to finish) : ') ;
end
```

This program asks the user to enter a number. If this number is not equal to 0 then the loop repeats, otherwise (if the given value is 0) then the program stops.

4. Recap exercise

There are predefined functions in Matlab given in the table below. Let's try to program them (for a given vector V).

function	*Description*	*The program that simulates it*

sum (V)	The sum of the elements of a vector V	`n = length(V);`
		`sum = 0 ;`
		`for i = 1 : n`
		`sum=sum+V(i) ;`
		`end`
		`disp(sum)`

prod (V)	The product of elements of a vector V	`n = length(V);`
		`product = 1 ;`
		`for i = 1 : n`
		`product=product*V(i) ;`
		`end`
		`disp(product)`

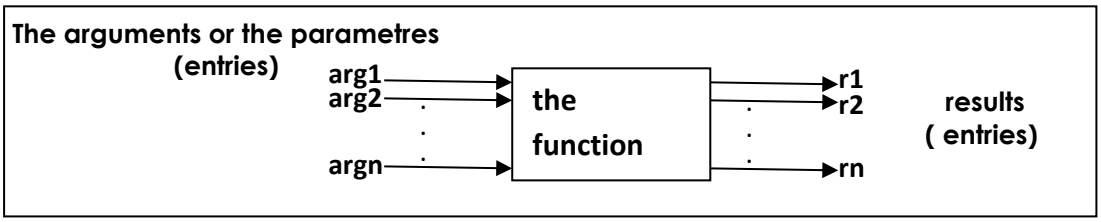
mean (V)	The average of the elements of a vector V	`n = length(V);`
		`moyenne = 0 ;`
		`for i = 1 : n`
		`moyenne = moyenne+V(i) ;`
		`end`
		`moyenne = moyenne / n`

diag (V)	Create a matrix with vector V in the diagonal, and 0	`n = length(V);`
		`A = zeros(n) ;`
		`for i = 1 : n`
		`  A(i,i)=V(i) ;`
		`end`
		`disp(A)`

sort(V)	Order elements of vector V in ascending order	`n = length(V);`
		`for i = 1 : n-1`
		`  for j = i+1 : n`
		`  if V(i) > V(j)`
		`  tmp = V(i) ;`
		`  V(i) = V(j) ;`
		`  V(j) = tmp ;`
		`  end`
		`  end`
		`end`
		`disp(V)`

5. The functions

There is a difference in concept between functions in computer science or mathematics:
1. In computer science, a function is a routine (a sub-program) that accepts arguments (parameters) and returns a result.



1. In mathematics a function **f** is a relationship that assigns to each value x no more than one value **f(x).**

5.1 Creating a function in an M-Files:

Matlab contains a large number of predefined functions such as **sin, cos, sqrt, sum**, … etc. And it is possible to create our own functions by writing their source codes in M-Files (with the same function name) respecting the following syntax:

```
function [r1, r2, …, rn] = nom_fonction (arg1, arg2, …, argn)

      % The body of the function
      . . .
      r1 = . . .  % the value returned for r1
      r2 = . . .  % The value returned for r2
      . . .
      rn = . . .  % the value returned for rn
end               % The end is optional
```

Or: $r_1 \ldots r_n$ are the values returned, and $arg_1 \ldots arg_n$ are the arguments.

Example : Write a function that calculates the square root of a number by the Newton method (view in the TD).

Solution :

```
>> edit
```

The root file. m

```
function r =racine(numbre)
r = numbre/2;
precision = 6;
for i = 1:precision
        r = (r + numbre ./ r) / 2;
end
```

Execution :

```
>> x = root (9)
      x =
           3
>> x = root (196)
      x =
         14.0000
>> x = root ([16,144,9,5])
      x =
           4.0000   12.0000    3.0000    2.2361
```

Remark :

Unlike a program (a script), a function can be used in an expression for example :

2* root (9)-1.

Comparison between a program is a function

program	*fonction*

`a = input('Enter a positive number: ');`
`x = a/2;`
`Precision = 6;`
`for i = 1:precision`
`  x = (x + a ./ x) / 2;`
`end`
`disp(x)`
	`function r =root(number)`
	`r = number /2;`
	`Precision = 6;`
	`for i = 1: Precision`
	`  r = (r + number ./ r) / 2;`
	`end`

execution :
`>> root` ↵
`Enter a positive number: 16`↵
`4`
	execution :
	`>> root (16)`
	`ans =`
	`4`

one cannot write expressions such as : **>> 2* root + 4**	you can write phrases like : **>> 2* root (x) + 4**

6. Polynomials

MATLAB represents a polynomial as a row vector containing the coefficients arranged in decreasing order of powers. For example, the polynomial P given by $P(x) = x^2 - 6x + 9$ is represented as:
Let me know if you need further adjustments:

```
>> P = [1 −6 9]
```

The following table shows some MATLAB commands for manipulating polynomials:

```
>> y=polyval(p,x)

>> z=roots(p)

>> p=conv(p1,p2)

>> [q,r]=deconv(p1,p2)

>> y=polyder(p)

>> y=polyint(p)
```