Chapter 3

Programing with MATLAB

Contents

3.1 Comparison operators and logical operators

The comparison operators in MATLAB are illustrated in Table 3.1.

Operator	Meaning
>	Greater than
<	smaller than
>=	Greater than or equal to
<=	less than or equal to
==	equals
~ =	different about

Table 3.1: Comparison operators.

These are *binary* operators, which return 0 (false) when the relationship is false and 1 (true) when the relationship is true.

Example:

```
» X= 4
»x>5
 Ans
= 0
```

The comparison can be made to an entire matrix, coefficient by coefficient. Example:

```
»A= [1 2 3; 4 5 6; 7 8 9];
»x>=6
ans=
0  0  0
0  0 1
1  1  1
```

The logical operators are presented in Table 3.2

Operator	meaning	equivalent function
~	ironed	not(A)
	and	and(A,B)
|	or	or(A,B)

Table 3.2: Logical operators.

3.2 Check-up instructions

3.2.1 The if statement

The **if statement** is the simplest and most widely used of the flow control structures. It makes it possible to orient the execution of the program according to the logical value of a condition. The general syntax is as follows:

```
if (Conditions)
Instructions
end
```

The *if else end* structure:

```
if (Conditions)
Instructions
else
Instructions
end
```

If it is necessary to check several conditions instead of only one, we can use **elseif** clauses for each new condition, and at the end we can put an **else** in the case where no condition has been evaluated as *true*. The general syntax is as follows:

```
if (Condition 1)
    Instructions
elseif (Condition 2)
    Instructions
    ..
    ..
elseif (Condition n)
    Instructions
else
    Instructions
end
```

3.2.2 The switch statement

The **switch** statement executes groups of statements according to the value of a variable or expression. Each group is associated with a **case** clause that defines whether this group should be executed or not according to the equality of the value of this case with the evaluation result of the switch expression.

If all the **boxes** have not been accepted, it is possible to add a clause **otherwise** that will be executed only if no box is executed.

The general structure of this instruction is:

```
switch (Expression)
    case value 1
        Instruction Group 1
    case  value 2
        Instruction Group 2
    . . .
    case  value N
        Instruction group N
    owerwise
        Instruction group if all checkboxes failed
end
```

Example: Here is an example of the **switch statement**

```
x = input ('Enter a number: ');
switch (Expression)
      Case (0)
            disp('x = 0 ')
      Case  10.
            disp('x = 10 ')
      Case 100
            disp('x = 100 ')
      owerwise
            disp('x n"is not 0 or 10 or 100 ')
end
```

The execution will give:

If the user entered for example a value of 30, then the message that will be displayed is: *x is not 0 or 10 or 100*.

3.2.3 while loop

The while loop is written as follows:

```
while (Conditions)
      Instruction 1
      Instruction 2
      .
      .
      .
      Instruction N
end
```

Example: Write a **while loop**, which will display the value of $\sqrt{n}$, for all even integers n from 0 to 20.

```
n=0
while (n < =20)
      sqrt(n)
      n = n + 2;
end
```

3.2.4 for loop

Instruction **for** repeats the execution of a group of instructions a determined number of times. It has the following general form:

```
for (Start:Step:End)
    Instruction 1
    Instruction 2
    .
    .
    .
    Instruction N
end
```

Example: Write a **for** loop, which will display the value of $\sqrt{n}$, for all even integers n from 0 to 20

```
n
for (n = 0 : 2 : 20)
    sqrt(n)
end
```

3.2.5 Wrap Up Exercise

Write MATLAB scripts to program the following predefined functions: **sum ()**, **prod()** and **mean ()** applied for the vectors using the for and while loop.

1. ***The sum(V) function***

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

2. ***The prod(V) function***

```
% The product of the elements of a vector V
n = length(V);  % n is the size of the vector V
Product = 1
for i = 1: n
    product=product*V(i);
end
```

3. ***The mean function (V)***

```
% The product of the elements of a vector V
n = length(V); % n is the size of vector V
mean = 0; i=1;
while i<=n
mean = mean+V(i);
end
mean = mean /n;
 disp(mean) % Show mean
```

3.3 Functions

Creating function files in MATLAB has two purposes:

1. Function files allow the user to define functions that are not among the built-in or predefined functions of MATLAB and to use them in the same way as the latter (these functions are called **user functions**).
2. Function files are also an important element in the programming of applications where functions play the role of the functions and procedures of the usual programming languages.

A function can have input arguments and output arguments as shown in Figure 3.1.

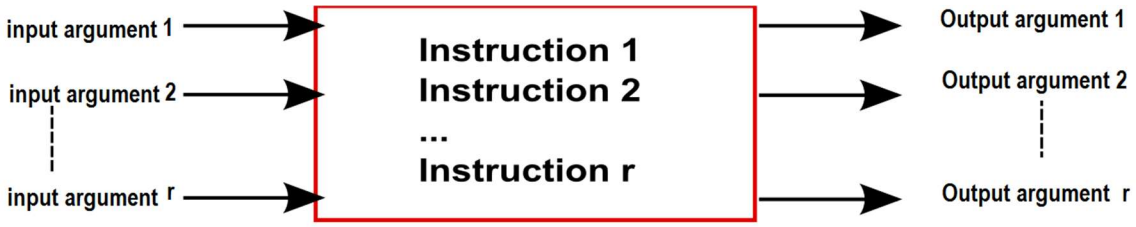

Figure 3.1: General diagram of a function.

3.3.1 Creating a function in an M-Files

It is possible to create our own functions by writing their source codes in M-Files (with the same function name) using the following syntax:

```
function [S1, S2, ..., Sm] = function name (E1, E2, ..., En)
% the body of the function
S1 = . . . % the returned value for S1 S2 =
. . . % the returned value for S2
...
Sm = . . . % the value returned for Sm
end % the end is optional
```

Where $S1, S2,...,Sm$ are the return values (results) and $E1, E2,...,En$.

Example: Write a function named ***Calculation*** that calculates the sum and product of the square roots of two positive integers a and b.

Figure 3.2 illustrates one solution.

Unlike a script, a function can be used in an expression for example: $4 * Calculation + 5$ (See Figure 3.3).

3.3.2 Comparison between a program is a function

To explain the difference between a MATLAB script and a function, Figure 3.4 shows the same example explained previously but using a script named *Calcul_1*.

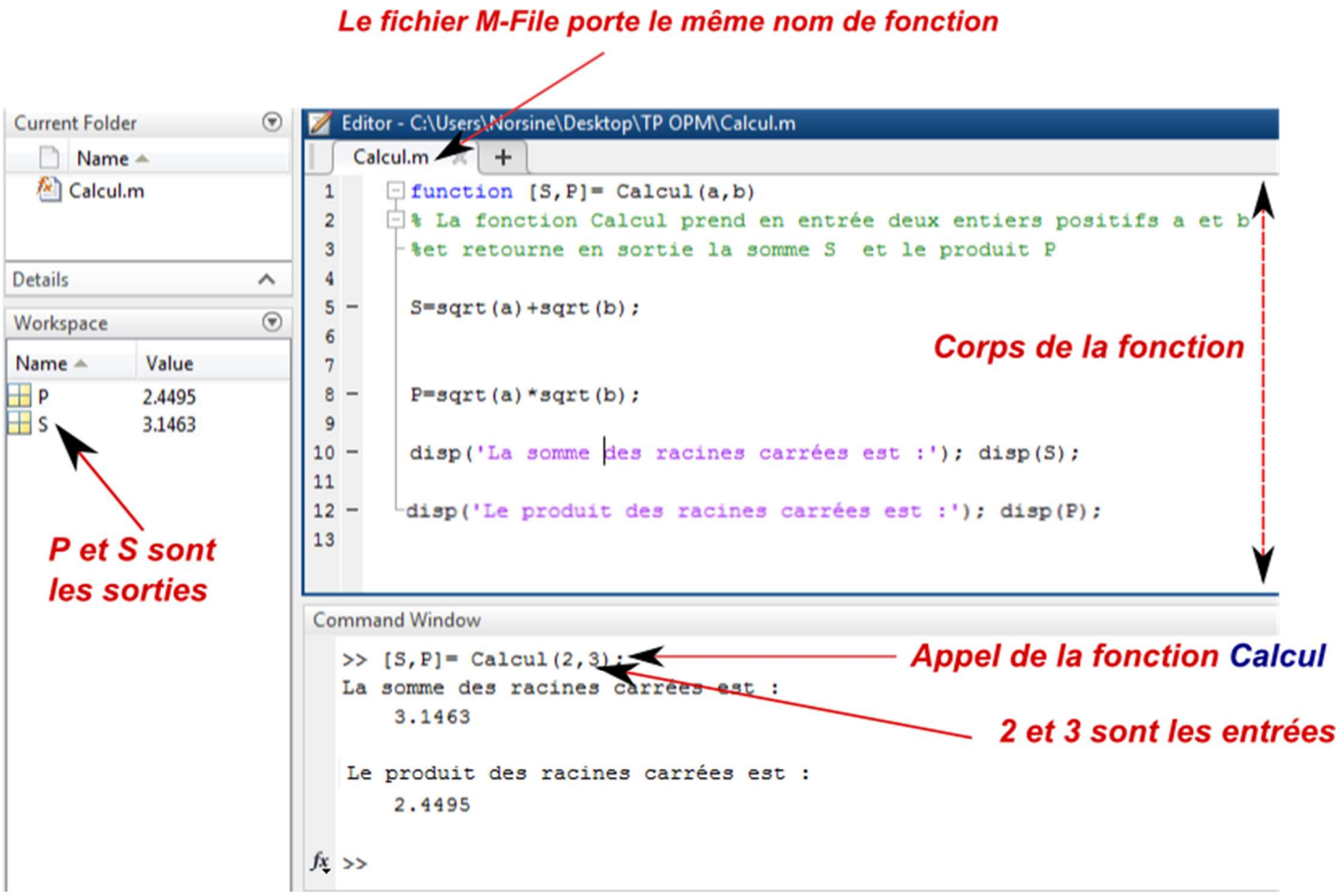

Figure 3.2: Demonstration of the creation and execution of the **Calculation** function.

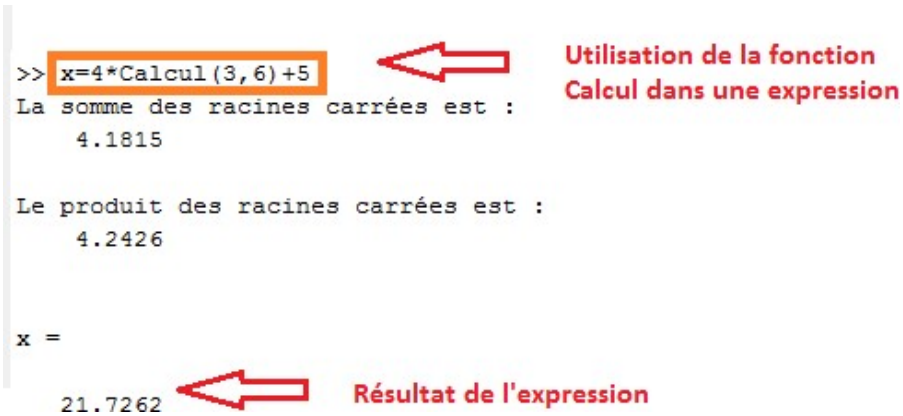

Figure 3.3: Demonstration of the creation and execution of the **Calculation** function.

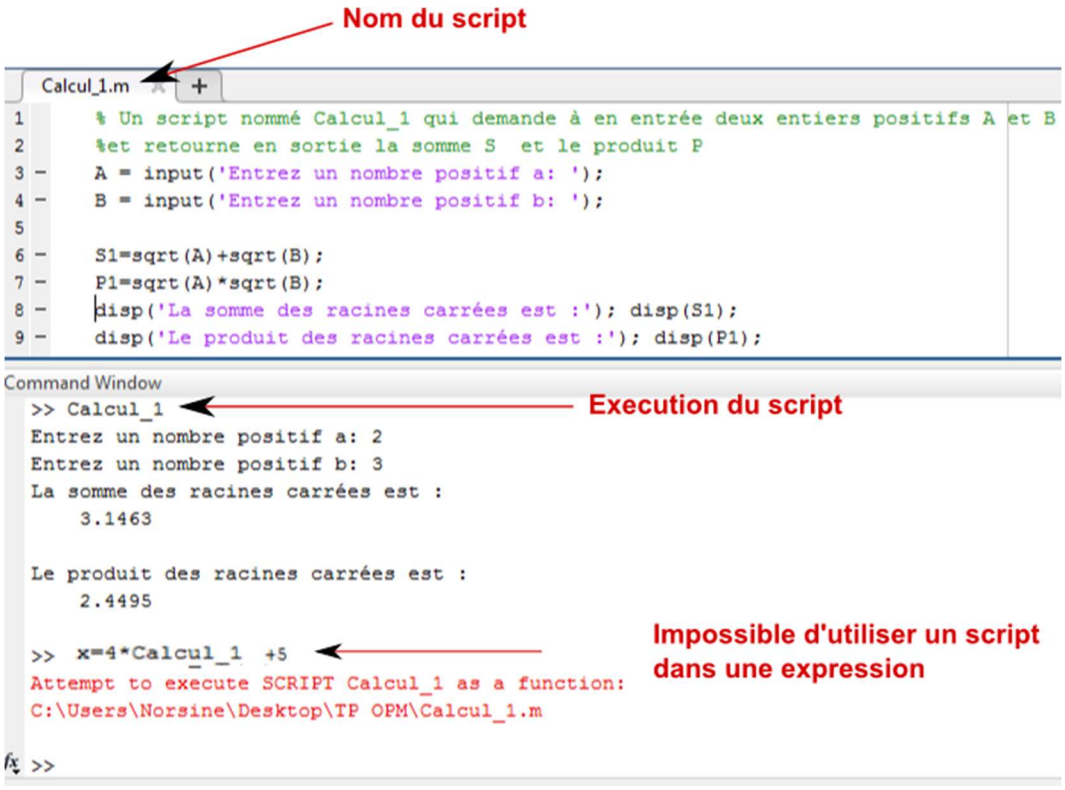

Figure 3.4: For demonstration purposes: Execution and use of a script in an expression.

3.4 Graphics and data visualization

MATLAB offers a powerful visualization system that enables the presentation and graphical display of data in a way that is both efficient and easy. In this section of the course, we will introduce the essential basics for drawing curves in MATLAB.

3.4.1 The plot function

MATLAB offers powerful features for creating graphics. The simplest display is done using the *plot* function.

To draw a curve $y = cos(\pi x)$ for example, where x=0 :4; just do:

```
»x= [0 :0.1 :4];
»y = cos(πx);
»plot(x,y)
```

Figure 3.5 illustrates the result of executing the *plot* function for displaying the curve of the function $y = cos(\pi x)$.

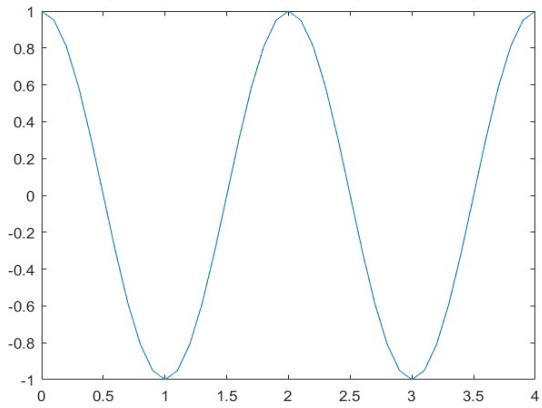

Figure 3.5: Graphical representation of the function $y = cos(\pi x)$, where x=0 :4.

3.4.2 The function appears

The management of the different figures is done using the *figure* function:

- To display several curves on the same figure, the graph is held using the *hold on* command. This hold is disabled using the *hold off* command.
- To save a figure, use the *print* function.
- The *close* function closes the active figure while the *close all* command closes all figures.

Charting and manipulating axes and scales is done by:

- **xlabel**('x-axis'): to give a title to the x-axis,
- **ylabel**('y-axis '): to give a title to the y-axis,
- **title**('speed evolution'): to give a title to the graph

```
»X = [0 :0.1 :4];
»y = cos(πx);
(figure 1).
»plot(X,Y)
```

The *figure(1)* command creates a new window under MATLAB named Figure 1. We will use it later if we want to keep this figure and display a second figure(2), etc.

3.4.3 Appearance of a curve

It is possible to manipulate the appearance of a curve by modifying:

- the color of the curve
- the shape of the coordinate points
- the type of line connecting the points.

To do this, we add a new argument (which can be called *a marker*) of the character string type to the plot function like this: `»plot (x, y, 'marker')`

The content of the marker is a combination of a set of special characters gathered in Table 3.3.

Example:

Modification of the appearance parameters of the curve $y = cos(\pi x)$ (see Figure 3.6):

```
»X = [0 :0.1 :4];
»y = cos(πx);
»plot (x, y, 'r: *')
```

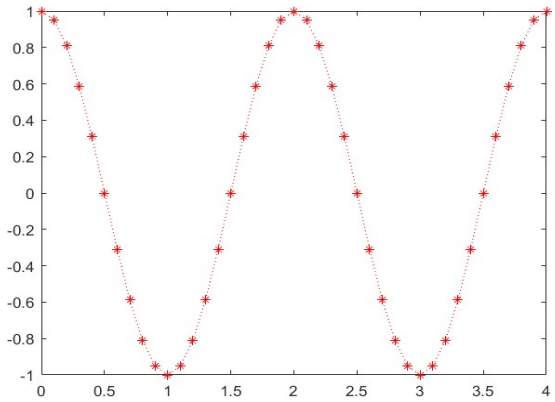

Figure 3.6: Modification of the appearance parameters of the function $y = cos(\pi x)$, where x=0 :4.

Curve Color	
Characters	Effect
b or blue	blue curve
g or green	green curve
r or red	red curve
y or yellow	yellow curve
k or black	blue curve
m or magenta	bright purplish curve
c or cyan	curve between green and blue
Plot Style	
Characters	Effect
-	online full
:	dashed
-.	in dashed dot
- -	in dashes
Points Representation	
Characters	Effect
.	a point.
◦	a circle •
×	the symbol ×
+	the + symbol
*	The * symbol
s	a square ■
d	a diamond ♦
v	lower triangle ▼
^	upper triangle ▲
<	left triangle ◀
>	right triangle ▶

Table 3.3: Parameters for manipulating the appearance of a curve.

3.4.4 Multi-function graph

By default in MATLAB, each new drawing with the plot command clears the previous one. To force a new curve to coexist with previous curves, there are two methods for drawing multiple curves in the same figure:

1. Use *holt on* function;
2. Use plot with multiple arguments.

Using the *holt on* function

If we want to place two curves on the same graph we use the *hold* on command (to use with the *hold off* command.)

For example, to draw the curve of the two functions $cos(x)$ and $sin(x)$ in the same figure, we can write (see Figure 3.7).

```
close all;
x=[0 :0.1 :4];
y1=cos(x);
y2=sin(x);
plot(x,y1,'b-o')
hold on
plot(x,y2,'r:s')
xlabel('Radians');
ylabel('Function Value');
title('Graph of cos(x) function and sin(x) function')
```

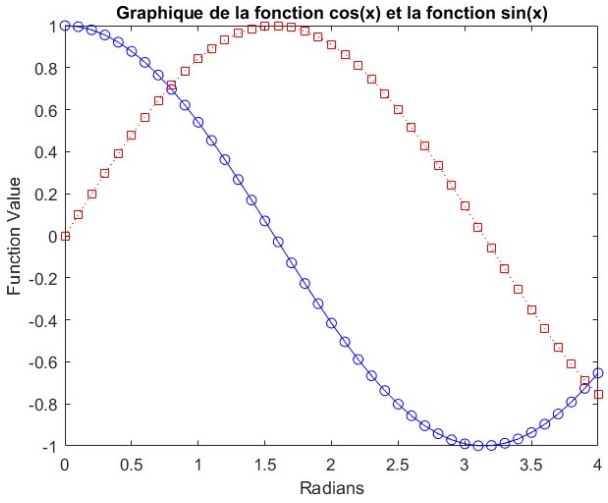

Figure 3.7: Graph of two functions $sin(x)$ and $cos(x)$ with *hold on*.

Using *plot* with multiple arguments

We can use *plot* with several pairs (x,y) or triples (x ,y, 'marker') as arguments.
For example, to draw the same previous functions we write (see Figure 3.8):

```
close all;
x=[0 :0.1 :4];
y1=cos(x);
y2=sin(x);
plot(x,y1,'g: * ',x,y2,'m – s ')
xlabel('Radians');
ylabel('Function Value');
title('Graph of cos(x) function and sin(x) function')
```

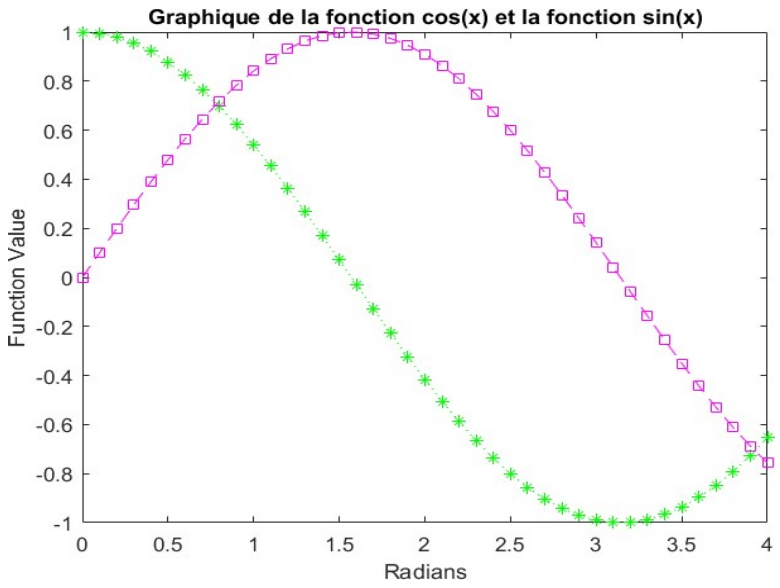

Figure 3.8: Graph of two functions *sin*(*x*) and *cos*(*x*) in *plot* use with several arguments .