



University mohamed boudiaf of m'sila
Faculty of Mathematics and Computer Science
Department of Mathematics



Domain: Mathematics and Computer Science

Field: Mathematics

Level: math license second year

Polycopied course for the module
Programming Tools 2 with MATLAB

Mihoubi Hamza

E-mail: hamza.mihoubi@univ-msila.dz

Year 2021-2022

Table of Contents

Introduction	1
1. First part: basic elements	
1.1 Interface MATLAB.....	3
1.1.1 Command Window.....	4
1.1.2 Boutton Start.....	5
1.1.3 Workspace.....	5
1.1.4 Current Directory.....	5
1.1.5 Command History.....	6
1.1.6 Editor/Debugger.....	6
1.1.7 The M-files.....	7
1.2 Use of MATLAB in the manner of a scientific calculator.....	10
1.2.1 Special variables and constants.....	10
1.2.2 Mathematical operators.....	10
1.2.3 Math functions.....	11
1.2.4 Calculation on complex numbers.....	12
1.2.5 The Problem with clc; clear; close all.....	13
1.3 Calculation on matrix.....	16
1.3.1 Definition of a vector.....	16
1.3.2 Some useful functions.....	17
1.3.3 Definition of a matrix.....	18
1.3.4 Particular matrices.....	18
1.3.5 Extraction of sub-arrays.....	19
1.3.6 Matrix construction by blocks.....	19
1.3.7 Operations on tables.....	20
1.3.8 Multiplication, division and power in the matrix sense.....	21
1.3.9 Distinction between term operations and matrix operations....	22
1.3.10 Relational or logical operations on arrays.....	26
1.3.11 Creation of the .m file of a function.....	29
1.4 Graph in two dimensions and management of the graphic windows.....	35
1.4.1 Draw the graph of a function.....	35
1.4.2 The Plot command.....	38
1.4.3 The loglog command.....	44
1.4.4 Legend of a figure.....	45
1.4.5 Display multiple curves in a single window.....	46
1.5 3D Graphic	58
1.5.1 Draw the level lines of a function 2 variables.....	58
1.5.2 Represent an equation surface $z=g(x,y)$	61
1.5.3 Represent a parameterized surface.....	62
1.6 Calculation on polynomials	69
1.6.1 Operation polynomials in MATLAB.....	69
1.6.2 Handling Polynomial Functions in MATLAB.....	70
1.6.3 Evaluation of a Polynomial.....	72

1.6.4	Determining the coefficients of a polynomial from its roots ...	73
1.6.5	Graphic Representation.....	73

2. Part two: MATLAB programming language and use of scripts

2.1	Script files	78
2.2	Functions.....	79
2.2.1	Creating a function in an M-Files.....	79
2.3	Structures of control of flux.....	80
2.3.1	Conditional Statements if, elseif, else.....	80
2.3.2	Conditional Statements switch ,case.....	82
2.3.3	Conditional loop while.....	83
2.3.4	Iterative loop for.....	84
2.4	Examples of applications.....	87
2.4.1	Digital integration.....	87
2.5.2	Laplace transformation.....	89

Reference

First part: Basic elements

Introduction

This document is an introduction to MATLAB, scientific computing software. Its objective is to prepare the student for practical work in Automatic Control, Mechanics and Numerical Analysis in which this tool is intensively used for the application and simulation of the theoretical principles presented in class. In addition, this manual offers the opportunity for the student to train in widely used professional software.

Cleve Moler, then a professor of computer science at the University of New Mexico, created MATLAB in the 1970s to help his students. It was engineer Jack little who identified the commercial potential of MATLAB in 1983. C. Moler, J. little, and Steve Bangart created MathWorks in 1984 and rewrote MATLAB in C.

MATLAB allows interactive work either in command mode or in programming mode; While still having the possibility of making graphic visualizations. Considered one of the best programming languages (C or FORTRAN), MATLAB has the particularities following with respect to these languages:

- Easy programming,
- Continuity among integer, real and complex values,
- The wide range of numbers and their precision,
- The very comprehensive mathematical library,
- The graphical tool which includes graphical interface functions and utilities,
- The possibility of linking with other classic programming languages (C or FORTRAN).

The best way to learn how to use this software is to use it yourself, by experimenting, making mistakes and trying to understand the error messages that will be returned to you. These messages are in English! This document is intended to help you for some first steps with MATLAB.

First part: Basic elements

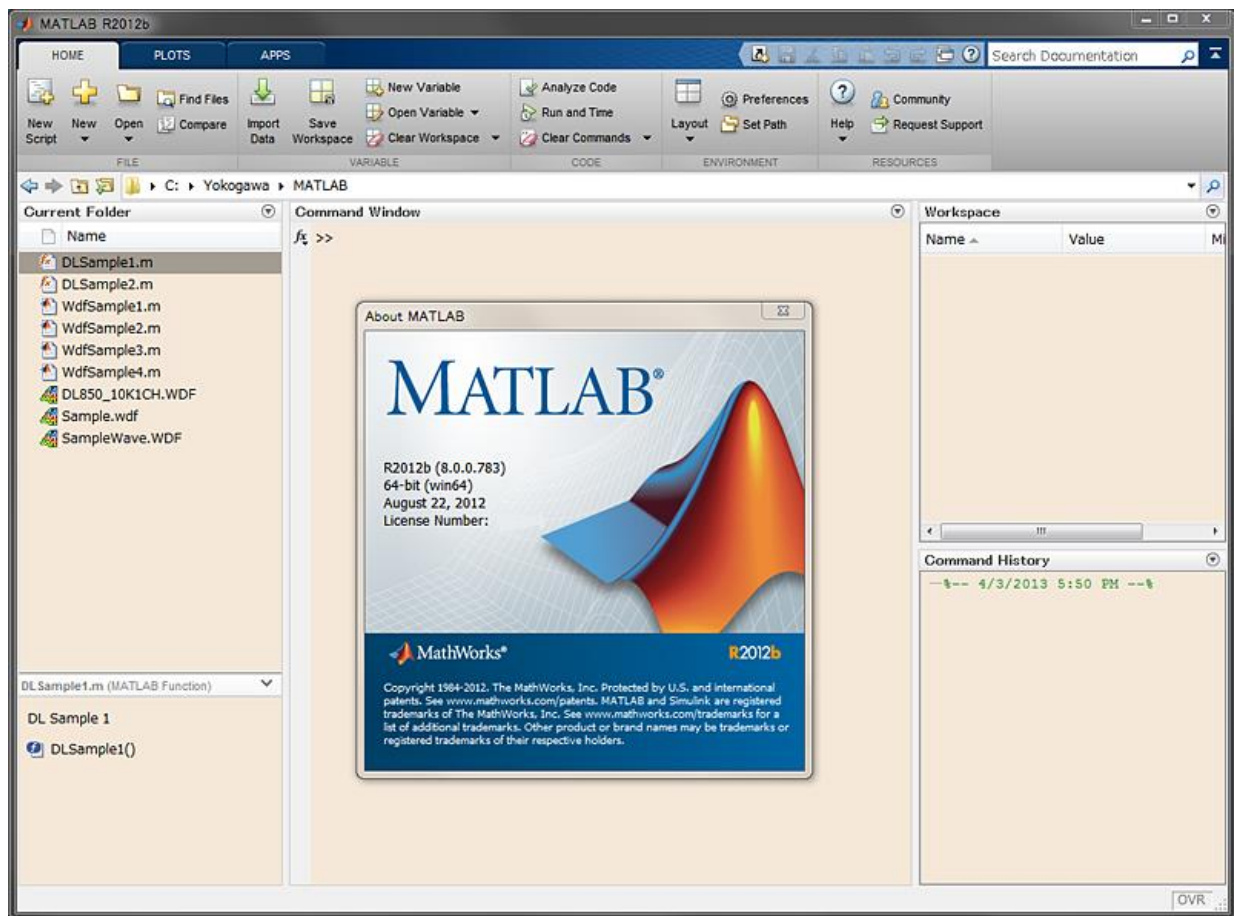
1.1 Interface MATLAB

- 1.1.1 Command Window**
- 1.1.2 Boutton Start**
- 1.1.3 Workspace**
- 1.1.4 Current Directory**
- 1.1.5 Command History**
- 1.1.6 Editor/Debugger**
- 1.1.7 The M-files**

First part: Basic elements

1.1 Interface MATLAB

When MATLAB is launched, the following interface appears:



There are three windows. The upper left window therefore displays Workspace. Below is the current directory and the Command History.

Finally, on the right, there is the Command Window. To "close" an interface window, simply click on the button representing an arrow in the upper right corner of each window.

The interface also has menu bars [1].

The commands available in the menus are:

Edit! Clear Command Window Clear instructions or results can be seen in the Command Window.

Edit! Clear Command History Cancel previously recorded commands.

Edit! Clear Command Workspace Erases stored variables from memory.

View Controls the visual aspects of different windows.

The status bar also contains buttons that allow you to easily add, modify or remove acknowledgment requests and comments.

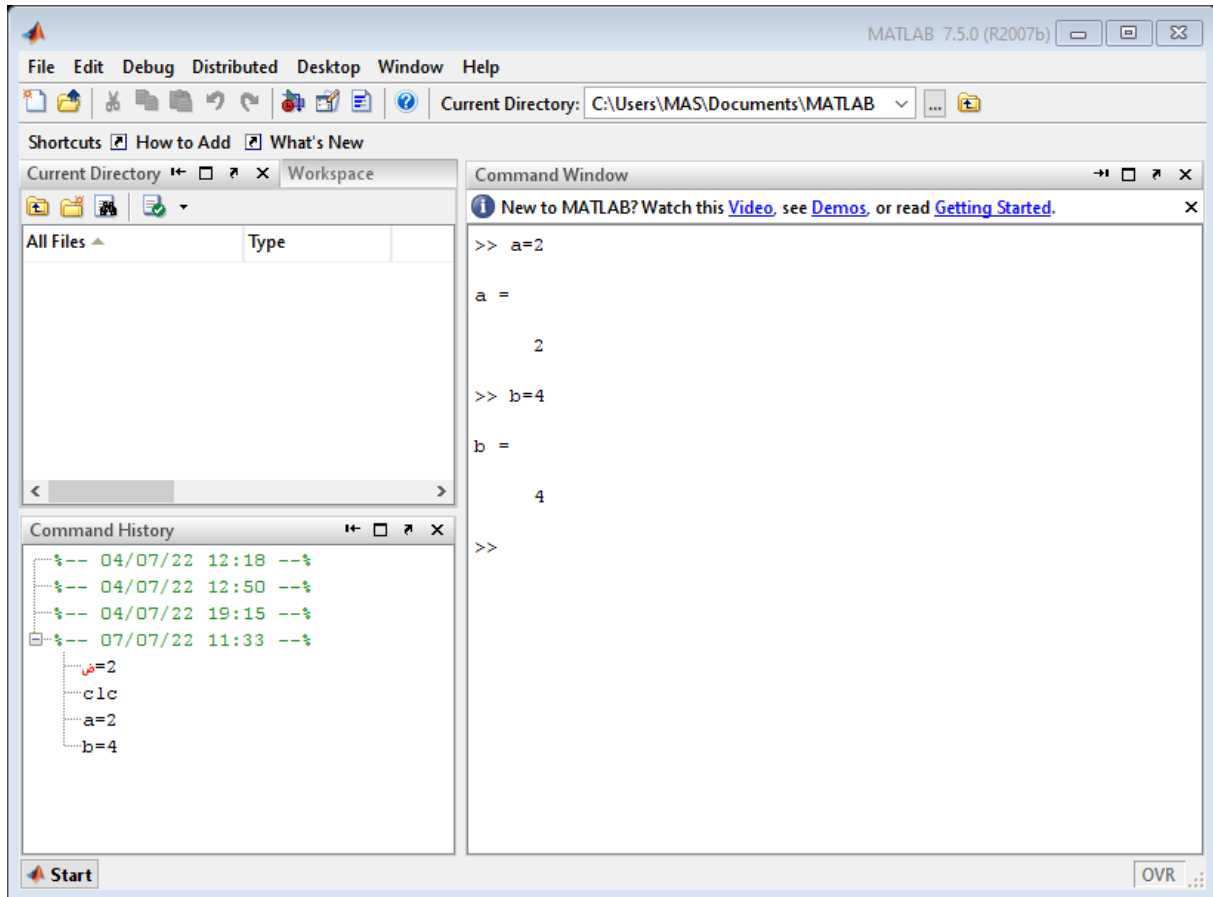
The Current Directory can be determined by this bar without having to go through the window of the same name.

By clicking slipping the horizontal and vertical bars separating the different windows of interface, it is possible to extend or to make look smaller these windows.

First part: Basic elements

1.1.1 Command Window

One of the most important windows of MATLAB, Command Window treats given instructions. It is after invitation ('prompt ') » that it is necessary to enter asked instructions. Results will be displayed from the return of line.



You can avoid retaking an instruction by hitting the up arrow ("), which will display the previous instruction. Continue to weigh until the desired instruction.

It is easy to observe on the image the instructions given as well as the answers obtained:

Display Instructions

```
>> a=2
a =
2
>> 2*a
ans =
4
>> x1=2*a+6
x1 =
10
>> a*b
ans =
8
```

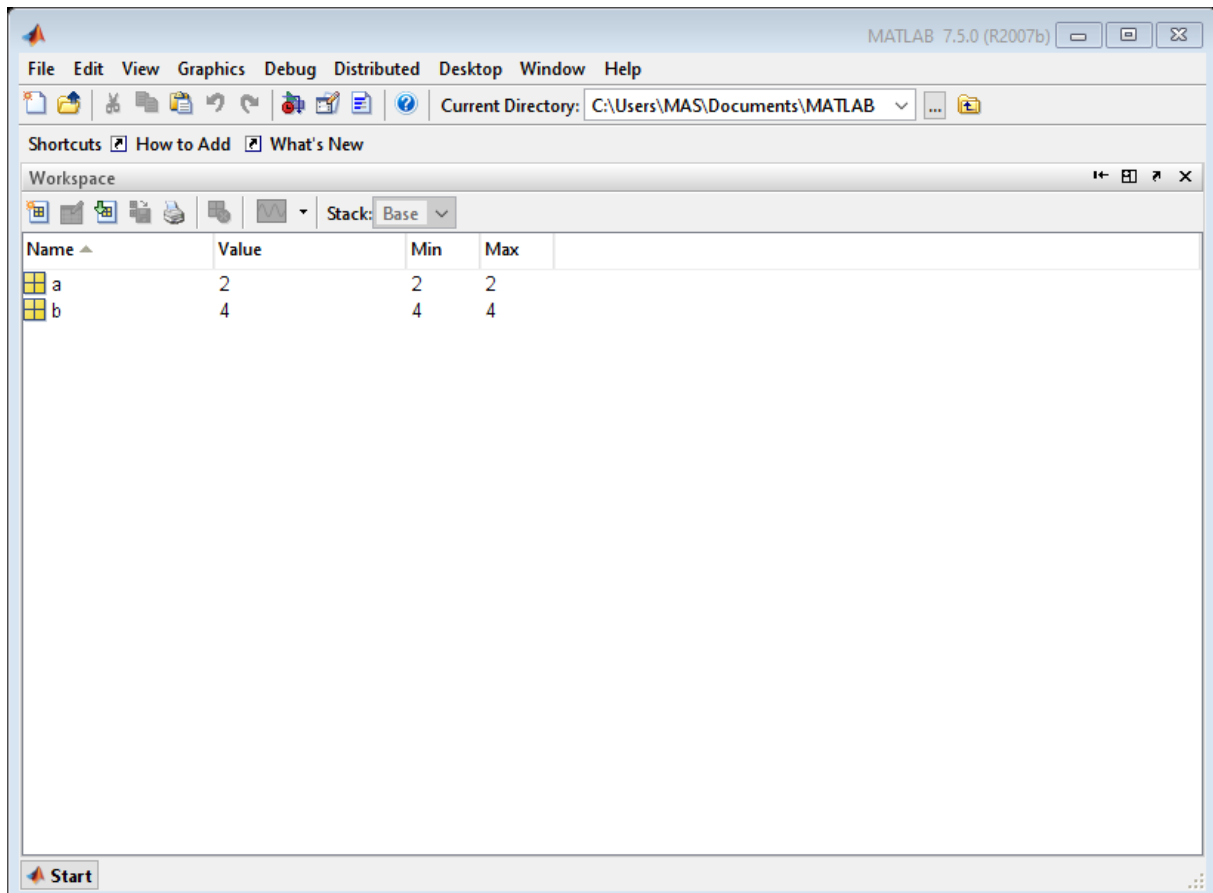
First part: Basic elements

1.1.2 Boutton Start

The Start button is a tool to quickly open certain MATLAB functions. For more information, see its topic in MATLAB Help.

1.1.3 Workspace

The Workspace window allows you to view stored variables. It contains their name, dimensions and the type of variable. Since Matlab is based on matrices, all the variables are made up of several dimensions: a scalar is a 1 1 matrix and a vector is a 1 n or n 1 matrix, etc. It is possible to delete some variables as well as to edit them. To clear them all, use the Clear Workspace command in the Edit menu.



By double-clicking on a variable, the Array Editor window appears. This window contains the values of the variables and allows you to modify them. In the following example, the initial variable is a 1 1 matrix. By adding values in the adjacent boxes, the matrix was transformed to 5 3 dimensions.

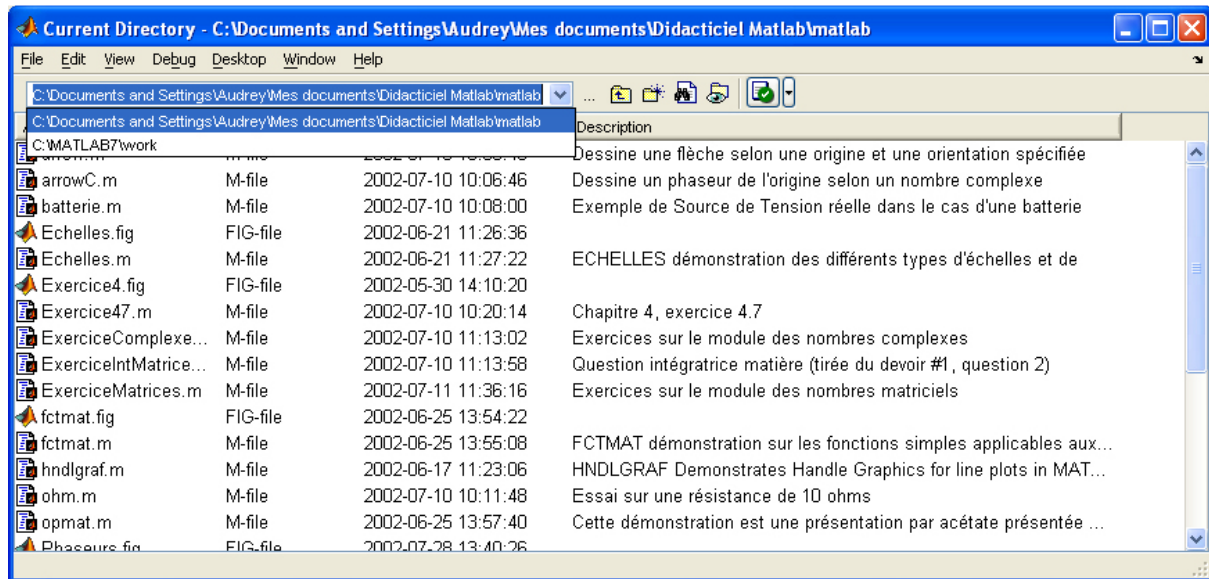
1.1.4 Current Directory

Current Directory is the common directory where are recorded files-Mr. It is hard recommended to create a directory other than that provides by Matlab to manage better files contained indoors. Although you will learn it in the course of educational software program, some general principles in comparison with Current Directory are necessary:

To compile an M-file, it must be saved in the current directory.

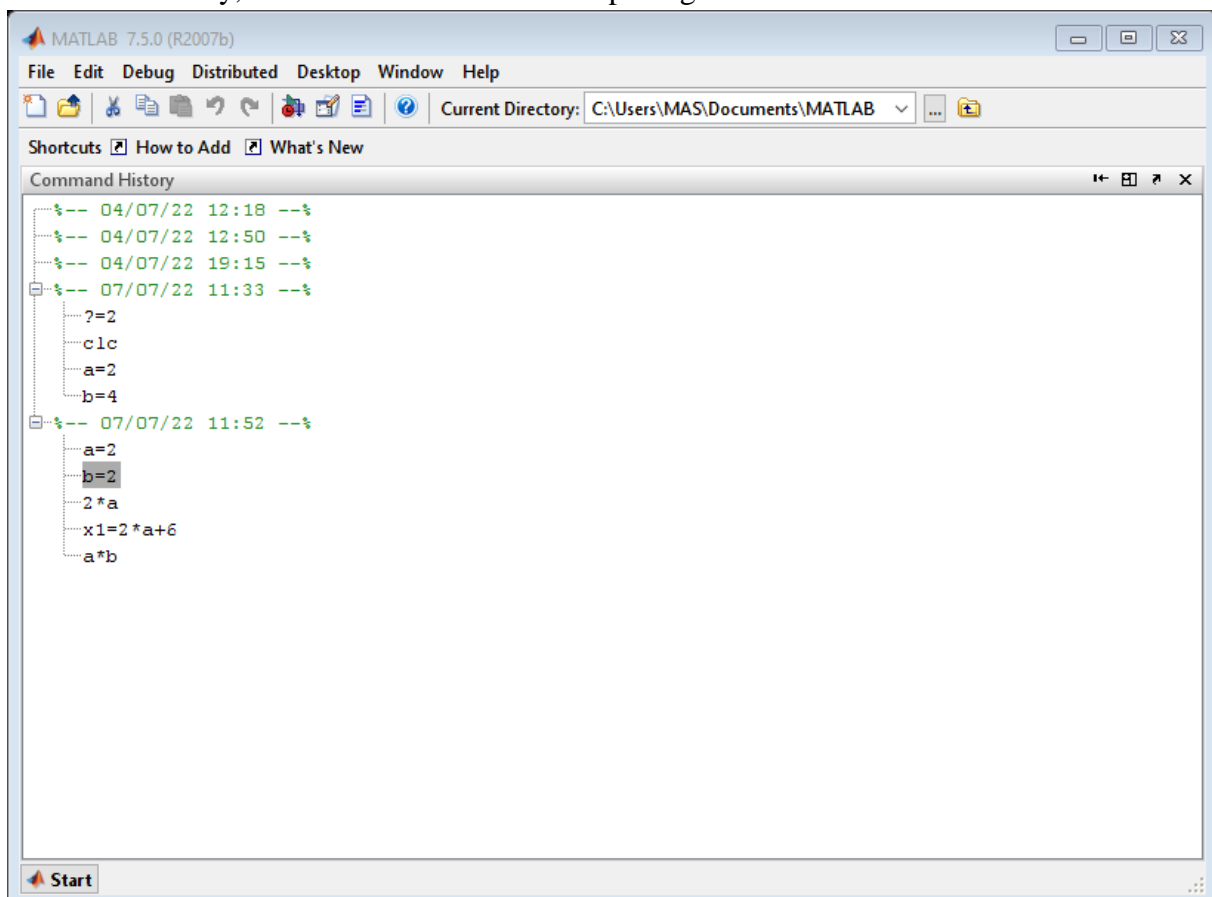
If an M-file calls a function other than a Matlab function (i.e. a function created by the user), the M-file calling the function and the M-file defining the function must be in the same current directory.

First part: Basic elements



1.1.5 Command History

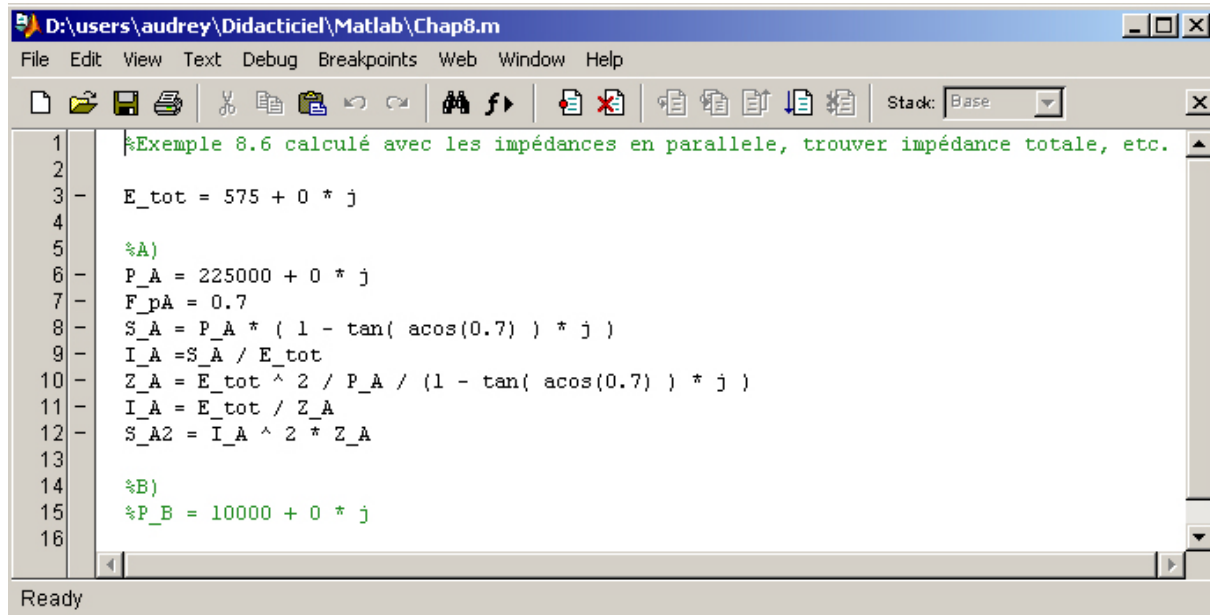
Command History inscribes orders as they are appellées in Command Window. It keeps these orders in memory, as well as the date and the opening time of each of the sessions of Matlab.



1.1.6 Editor/Debugger

To process M-files, you must use the Matlab editor/debugger. This window is not part of the basic Matlab interface and opens when you open an M-file or when you create a new M-file.

First part: Basic elements



The screenshot shows the MATLAB editor window titled 'D:\users\audrey\Didacticiel\Matlab\Chap8.m'. The menu bar includes File, Edit, View, Text, Debug, Breakpoints, Web, Window, and Help. The toolbar contains various icons for file operations, editing, and running. The script content is as follows:

```
1 %Exemple 8.6 calculé avec les impédances en parallele, trouver impédance totale, etc.
2
3 E_tot = 575 + 0 * j
4
5 %A)
6 P_A = 225000 + 0 * j
7 F_pA = 0.7
8 S_A = P_A * ( 1 - tan( acos(0.7) ) * j )
9 I_A = S_A / E_tot
10 Z_A = E_tot ^ 2 / P_A / (1 - tan( acos(0.7) ) * j )
11 I_A = E_tot / Z_A
12 S_A2 = I_A ^ 2 * Z_A
13
14 %B)
15 %P_B = 10000 + 0 * j
16
```

The status bar at the bottom indicates 'Ready'.

On the bar of buttons, a button is essential: the button RUN compiles program, i.e. carry out the orders of program. He can also be carried out with F5.

1.1.7 The M-files

In order to avoid having to retype a series of commands, it is possible to create a MATLAB program, known as an “M-file” (“M-file”), the name coming from the ending “.m” of these files. Using the MATLAB editor, create a text file that contains a series of MATLAB commands. To create an M-file, go to the File New M-File menu or click the blank button. Recording is normally done in the current directory. Once the file has been saved (under the filename.m for example), it is necessary to call it in MATLAB using the command:

```
>> filename
```

The commands stored there will then be executed and the results displayed in the Command Window. If you need to make a change to your series of commands, just edit the line of the M-file in question and run the M-file by entering the file name in MATLAB again.

Other ways to run the M-file is to click the Run button, or go to the Debug Run menu in the Editor/Debugger, or press F5.

M-files save you from having to retype a series of repeated commands and allow you to preserve your instructions, commands and calculations thanks to the recording. This is the recommended procedure for your labs.

Example

Creating an M-File - Procedure

Create a file by the button or menu

Write some instructions. For example, write the following instructions:

```
A=90
B=100
C=A+B
```

– Save the m-file in the current directory. If you don't, Matlab will ask you to save it before compiling it.

– Return to Matlab: the responses should have appeared in the command window. These answers should be :

First part: Basic elements

```
>> A =  
90  
B =  
100  
C =  
190
```

- **The ";" and the "..."**

The semicolon at the end of a line tells MATLAB not to return the result of the operation to the screen. A common practice is to put ";" at the end of all the lines and remove some of them when something goes wrong in our program, in order to see what happens.

In the editor/debugger as in the command window, the use of «. .» is useful to continue on the line below the current instruction.

```
>> B = pi * 3^2  
B =  
28.274  
>> B = pi * 3^2;  
>>
```

The following two instructions give the same result

```
>> A = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9  
A =  
45  
>> A = 1 + 2 + 3 + 4 + 5...  
+ 6 + 7 + 8 + 9  
A =  
45
```

1.2. Use of Matlab in the manner of a scientific calculator

- 1.2.1 Special variables and constants**
- 1.2.2 Mathematical operators**
- 1.2.3 Math functions**
- 1.2.4 Calculation on complex numbers**
- 1.2.5 The Problem with clc; clear; close all**

First part: Basic elements

1.2 Use of MATLAB in the manner of a scientific calculator

1.2.1 Special variables and constants

ans	the most recent answer
pi	number pi
inf	plus infinity
-inf	minus infinity
NaN	Not-a-Number

1.2.2 Mathematical operators

+	addition
-	substraction
*	multiplication
/	division
^	power

```
>> (2 + 5.2)*10 / (5^3)
ans =
0.5760
>> -2.52e3
ans =
-2520
>> 2*pi
ans =
6.2832
    long format e displays 16 digits:
>> format long e
>> 2*pi
ans =
6.283185307179586e+000
    format short (format par défaut) affiche 5 chiffres :
>> format short
>> 2*pi
ans =
6.2832
>> 1 / 0
Warning: Divide by zero
ans =
Inf
>> -1 / 0
Warning: Divide by zero
```

First part: Basic elements

```
ans =  
-Inf  
>> 0 / 0  
Warning: Divide by zero  
ans =  
NaN
```

1.2.3 Math functions

sin(X)	sinus
asin(X)	sinus reverse
cos(X)	cosinus
acos(X)	cosinus reverse
tan(X)	tangent
atan(X)	tangent reverse

exp(X)	exponential
log(X)	Natural logarithm
log10(X)	decimal logarithm
sqrt(X)	square root
abs(X)	absolute value

```
>> sin(2)  
ans =  
0.9093  
sinus (45°) :  
>> sin(45*pi/180)  
ans =  
0.7071  
>> 1 + exp(2.5)  
ans =  
13.1825  
Utilisation de variables  
>> 5*3  
ans =  
15  
>> ans+4  
ans =  
19  
>> a= 2 + log(15)  
a =
```

First part: Basic elements

```
4.7081
>> b = - 45
b =
-45
>> a * b
ans =
-211.8623
>> c = a - sqrt(abs(b))
c =
-2.0002
```

1.2.4 Calculation on complex numbers

In MATLAB, i and j represent the basic imaginary unit. You can use them to create complex numbers such as $6i+9$. You can also determine the real and imaginary parts of complex numbers and compute other common values such as phase and angle [2].

- **Functions:**

i	pure imagination
j	pure imagination
conj(X)	conjugate of the complex number X
real(X)	real part
imag(X)	imaginary part
abs(X)	module
angle(X)	argument (in radians)

Example

```
>> (4 - 2.5i)*(-2 + i)/(1 + i)
ans =
1.7500 + 7.2500i
>> a = 1 + i
a =
1.0000 + 1.0000i
>> b = -2 + 3.5j
b =
-2.0000 + 3.5000i
>> a + b
ans =
```

First part: Basic elements

```
-1.0000 + 4.5000i
>> a * b
ans =
-5.5000 + 1.5000i
>> a / b
ans =
0.0923 - 0.3385i
>> conj(a)
ans =
1.0000 - 1.0000i
>> a * conj(a)
ans =
2
>> real(a)
ans =
1
>> imag(conj(a))
ans =
-1
>> abs(a)
ans =
1.4142
>> angle(a)
ans =
0.7854
    sqrt : fonction racine carrée
>> c = 2 - sqrt(3)*i
c =
2.0000 - 1.7321i
>> abs(c)
ans =
2.6458
>> angle(c)
ans =
-0.7137
    Argument en degrés :
>> angle(c)*180/pi
ans =
-40.8934
```

1.2.5 The Problem with clc; clear; close all

clc: cleans up the command window and now one can work without getting confused with the commands for previous runs

First part: Basic elements

clear: erases the variables from previous runs this will reduce chances of error in subsequent runs and the programmer does not have to worry about unnecessary trash variables.

close all: closes all currently open figures. This can be very helpful during subsequent runs of the same script. If the figure from the previous run has not been closed then the subsequent run will plot the data on the already open figure. Which of course is a total waste.

Example

```
clc; close all; clear all; %clears command window
```

```
A = []; %variable A matrix
```

```
for i=1:6
```

```
for j=1:5
```

```
A(i,j) = (i - j) / j^2;
```

```
end
```

```
end
```

```
A=A'
```

```
A =
```

0	1.0000	2.0000	3.0000	4.0000	5.0000
-0.2500	0	0.2500	0.5000	0.7500	1.0000
-0.2222	-0.1111	0	0.1111	0.2222	0.3333
-0.1875	-0.1250	-0.0625	0	0.0625	0.1250
-0.1600	-0.1200	-0.0800	-0.0400	0	0.0400

1.3 Calculation on matrix

- 1.3.1 Definition of a vector**
- 1.3.2 Some useful functions**
- 1.3.3 Definition of a matrix**
- 1.3.4 Particular matrices**
- 1.3.5 Extraction of sub-arrays**
- 1.3.6 Matrix construction by blocks**
- 1.3.7 Operations on tables**
- 1.3.8 Multiplication, division and power in the matrix sense**
- 1.3.9 Distinction between term operations and matrix operations**
- 1.3.10 Relational or logical operations on arrays**
- 1.3.11 Creation of the .m file of a function**

First part: Basic elements

1.3 calculation on matrix

With Matlab, we essentially work with one type of object: matrices. A scalar variable is a matrix of dimension 1 x 1 and a vector is a matrix of dimension 1 x n or n x 1. It is essential to be comfortable with these notions to better understand the philosophy of Matlab and to exploit it effectively.

1.3.1 Definition of a vector

A vector is nothing else than a picture of stocks. There are several manners of creating a vector and the simplest of them is to write it expressly.

```
>> v = [1 3 3 4]
v =
1 3 3 4
```

All components are given in brackets and values are separated by a space (or a comma « , »). Here we have defined a line vector. A column vector is created using a semicolon « ; » as delimiter.

```
>> v = [1 ; 3 ; 3 ; 4]
v =
1
3
3
4
```

Although simple, this method is not practical for defining large vectors. A second method uses the two-point operator « : » to discretize an interval with a constant step.

```
>> v = 0:0.2:1
v =
0 0.2 0.4 0.6 0.8 1
```

This statement creates a vector containing values ranging from 0 to 1 with a step of 0.2. The syntax is as follows: vector = initial_value: increment: final_value. By default, the step is equal to 1.

```
>> v = 0:6
v =
0 1 2 3 4 5 6
```

Finally, predefined functions make it possible to automatically generate vectors.

```
>> v = linspace(0,10,1000);
>> v = logspace(-1,2,1000);
```

The first function creates a vector of 1000 points with stocks going of 0 - 10 also spacing out. The second creates a vector of 1000 points on space from 10⁻¹ to 10² with a logarithmic spacing out.

```
>> v = [9 4 -1 3 11 0.3];
>> v(4)
ans =
3
>> v (2:4)
ans =
4 -1 3
```

First part: Basic elements

`v(3)` returns the 3rd element of vector `v`. Argument `2:4` selects a block of elements (here from the second to the fourth).

1.3.2 Some useful functions

We present in this paragraph a set of usual functions related to the use of tables.

<code>+</code>	addition of matrices
<code>-</code>	subtraction of matrices
<code>*</code>	die product
<code>^</code>	power
<code>eye (n)</code>	unit matrix (identity matrix) of size $n \times n$
<code>inv (X)</code>	inverse of square matrix <code>X</code>
<code>rank (X)</code>	<code>X</code> matrix rank (number of independent columns or rows)
<code>det (X)</code>	determining the square matrix <code>X</code>
<code>X '</code>	transposed from matrix <code>X</code>
<code>/</code>	right division: <code>A / B</code> is equivalent to <code>A * inv(B)</code>
<code>\</code>	left division: <code>A B</code> is equivalent to <code>inv(A) * B</code>
<code>length(v)</code>	returns the size of the table.
<code>max(v)</code>	returns the maximum value of the array.
<code>min(v)</code>	returns the minimum value of the table.
<code>mean(v)</code>	returns the average value of the array elements.
<code>sum(v)</code>	calculate the sum of the elements in the table.
<code>prod(v)</code>	calculating the product of the elements in the table.
<code>sort(v)</code>	Ranks the elements in the table in ascending order.

All mathematical functions are applicable to vector-type variables.

In this case, the function is performed on each element of the vector.

```
>> v = [0 pi/4 pi/2 pi 2*pi]
```

```
v =
```

```
0 0.7854 1.5708 3.1416 6.2832
```

```
>> cos(v)
```

```
ans =
```

First part: Basic elements

1.0 0.7071 0.0000 -1.0000 1.0000

1.3.3 Definition of a matrix

The definition of a matrix is delimited by square brackets «[]». The different elements of a line are separated by a space and the different lines are separated by semicolons «; ». Thus to define

a matrix variable $M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$.

We will write:

```
>> M = [1 2 3; 4 5 6; 7 8 9];
```

ou

```
>> M = [1, 2, 3; 4, 5, 6; 7, 8, 9];
```

Access to an element of a matrix is done by specifying indices in parentheses following its name.

The element located $i^{\text{ième}}$ line and the $j^{\text{jème}}$ column is obtained by the command $M(i,j)$. For example, the value is retrieved by typing M_{23}

```
>> M(2,3)
```

ans =

6

You can also modify one of the elements directly by assigning it a new value.

```
>> M(2,3) = 11;
```

```
>> M
```

M =

1 2 3

4 5 11

7 8 9

1.3.4 Particular matrices

Some particular matrices, and very used, are more easily defined through functions. These functions take in argument the dimensions of the matrix which they like to construct. The first indicates the number of lines and second numbers it of columns.

The no matrix:

```
>> Z = zeros(2,4)
```

Z =

0 0 0 0

0 0 0 0

A matrix full of 1:

```
>> U = ones(3,3)
```

U =

1 1 1

1 1 1

1 1 1

The matrix identity:

```
>> I = eye(3)
```

I =

1 0 0

0 1 0

First part: Basic elements

```
0 0 1
A random matrix (elements between 0 and 1):
1) :>> R = rand (2, 2)
R =
0.9575 0.1576
0.9649 0.9706
A diagonal matrix:
>> D = diag([42,33,0,71])
D =
42 0 0 0
0 33 0 0
0 0 0 0
0 0 0 71
```

Unlike the previous ones, this last function takes as an argument a vector. The size of the diagonal matrix is therefore determined by the size of the vector.

1.3.5 Extraction of sub-arrays

It is often useful to extract blocks from an existing table. For this we use the operator « : ». To do this, it is necessary to specify for each index the start value and the end value. The general syntax is therefore as follows (for a two-dimensional array): array (start:end, start:end).

Thus to extract the block $\begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$ from the matrix M, we will type:

```
>> M(1:2, 2:3)
ans =
2 3
5 6
```

The character « : » alone, means the entire length is extracted. In this way, one can isolate a complete row, or column.

Example

```
>> M(1:2, :)
ans =
1 2 3
4 5 6
>> M(1, :)
ans =
1 2 3
>> M(:, 2)
ans =
2
5
8
```

1.3.6 Matrix construction by blocks

You know this principle in mathematics. For example, to start the previously defined matrices and vectors, we can define the matrix

First part: Basic elements

$$N = \left[\begin{array}{c|c} M & V \\ \hline U & 0 \end{array} \right]$$

Which is a 4x4 matrix. To do ,can under Matlab, we do as if the blocks were scalars, and we simply write:

```
>> N= [M V U 0]
```

N =

```
1    2    3    11
11   12   13   12
21   32   23   13
1    2    3     0
```

Or by using the character;

```
>> N= [M V; U 0]
```

This syntax is very used to lengthen vectors or matrices, for example if I want to add a column A M, made up by V:

```
>> M= [M V]
```

M =

```
1    2    3    11
11   12   13   12
21   32   23   13
```

If I want to add a line to him, made up of U:

```
>> M = [M; U]
```

M =

```
1    2    3
11   12   13
21   32   23
1    2    3
```

1.3.7 Operations on tables

We saw in the preamble that Matlab did not make a strong distinction between tables and matrices. In fact, for Matlab, everything is a picture, and a matrix is only a table which has a particular mathematical meaning.

So, if historically MATLAB offered functions in most cases for counting implicating matrices, today the functional ghost broadly stretched, and MATLAB offers functions for all counting on numerical data, tabulées in picture, or matrices in the mathematical sense of term. In this chapter, we review the arithmetic operations that we can perform with tabulated data or matrices [3].

First part: Basic elements

- **Addition and subtraction**

Both operators are the same as for scalars, namely + and -. From the moment the two tables concerned have the same size, the resulting table is obtained by adding or subtracting the terms from each table.

1.3.8 Multiplication, division and power term to term

These operators are noted $*$, $/$ And $^$ (Be careful not to forget the point).

They are planned to carry out term operations on two tables of the same size.

These operations are fundamental when we want to trace curves, and we will always see it again in this case of use, or more generally when we want to carry out these arithmetic operations on a set of tabulated data.

- **Multiplication**

Since you can manipulate matrices, Matlab also offers these matrix operations. The multiplication is noted simply $*$ and should not be confused with the term term multiplication (which by definition does not give the same result).

It goes without saying that if we write $a*b$, the number of columns of A must be equal to the number of B lines for the multiplication to work.

- **Division**

The matrix division is defined as the multiplication by the reverse of the matrix. Thus A/B represents the matrix has multiplied (in the matrix sense) by the reverse matrix of B.

- **Complement**

There is also a division on the left which is noted \backslash . Thus the syntax $A \backslash B$ means the opposite of A multiplied by B. This symbol can also be used to resolve linear systems: if V is a vector, $a \backslash v$ represents mathematically $a^{(-1)}$ to say the solution of the linear system $AX = V$.

- **Power**

The umpteenth power of a matrix represents this matrix multiplied, in the matrix sense, n times by itself.

First part: Basic elements

1.3.9 Distinction between term operations and matrix operations

It is fundamental to clearly distinguish the multiplication operation (and by consequence of division and power) defined term term of that defined for matrices.

In many cases (and especially if the matrices are square and of the same dimensions), the two types of operations can be carried out, without producing a syntax error for the Matlab language, but will produce completely different digital results.

Example

To show the difference between operators. * And *, let's take a trivial example involving the Multipliée Identity matrix at the matrix (I*A). Here is the multiplication in the sense of matrices:

```
>> [1 0; 0 1] * [1 2; 3 4]
ans =
     1     2
     3     4
```

In this case, we find the matrix $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, by definition of what the identity matrix is.

And now let's look at the term-term multiplication:

```
>> [1 0; 0 1] .* [1 2; 3 4]
ans =
     1     0
     0     4
```

- **Fundamental**

If you handle tabulated data, the operations you are going to carry out are necessarily term term; the operators *, / and ^ will then be preceded by a point.

If you perform matrix calculations, the operators have no point.

- **Complement**

For these three operations, for tables or matrices, there are also functions that can replace operators:

<i>Operators and equivalent functions</i>			
Term term operations		matrix operations	
A.*B	times(A,B)	A*B	mtimes(A,B)
A./B	rdivide(A,B)	A/B	mrdivide(A,B)
A.^B	power(A,B)	A^B	mpower(A,B)

First part: Basic elements

- **Transposition**

The transposition operator is the character and is often used to transform line vectors into column vectors and vice versa.

- **Summary**

The following table summarizes the various operators applicable to matrices.

The following table summarizes the different operators applicable to matrices or tables.

<i>The different operators applicable to matrices or tables</i>		
Matlab operator	Mathematical writing	General term
A	A	A_{ij}
B	B	B_{ij}
A+B	A+B	$A_{ij}+B_{ij}$
A-B	A-B	$A_{ij}-B_{ij}$
A.*B		$A_{ij}B_{ij}$
A./B		A_{ij}/B_{ij}
A.^B		$A_{ij}^{B_{ij}}$
A.^s		A_{ij}^s
A*B	AB	$\sum_k A_{ik}B_{kj}$
A/B	AB^{-1}	
A\B	$A^{-1}B$	
A^n	A^n	
A'	A^T	A_{ji}

Example

Entering a square matrix of size 3 x 3:

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

```
A =
```

```

     2     4     6
     1     9     7
    -3     1     1
```

```
>> A (2, 3)
```

```
ans =
```

```
7
```

```
>> A (2, 3) = 7
```

```
A =
```

```

     2     4     6
     1     9     7
    -3     1     1
```

```
>> A'
```

First part: Basic elements

```
ans =
```

```
      2      1      -3
      4      9      1
      6      7      1
```

```
>> inv(A)
```

```
ans =
```

```
      0.0238      0.0238     -0.3095
     -0.2619      0.2381     -0.0952
      0.3333     -0.1667      0.1667
```

```
>> D = A * inv(A)
```

```
D =
```

```
      1.0000          0          0
      0.0000      1.0000      0.0000
          0     -0.0000      1.0000
```

```
>> rank(A)
```

```
ans =
```

```
3
```

```
>> det(A)
```

```
ans =
```

```
84
```

```
>> eye(7)
```

```
ans =
```

```
ans =
```

```
      1      0      0      0      0      0      0
      0      1      0      0      0      0      0
      0      0      1      0      0      0      0
      0      0      0      1      0      0      0
      0      0      0      0      1      0      0
      0      0      0      0      0      1      0
      0      0      0      0      0      0      1
```

```
>> B = [1 9 0 ; 1 3 1 ; 0 -1 1]
```

```
B =
```

```
      1      9      0
      1      3      1
      0     -1      1
```

```
>> A + B
```

```
ans =
```

First part: Basic elements

```
      3      13      6
      2      12      8
     -3       0      2
>> 11 + A

ans =

      13      15      17
      12      20      18
       8      12      12
>> 11 * A
ans =

      22      44      66
      11      99      77
     -33      11      11
>> A * B
ans =

       6      24      10
      10      29      16
      -2     -25       2
>> B * A
ans =

      11      85      69
       2      32      28
      -4      -8      -6
>> A*A*A*A
ans =

      -768      4008      2880
     -1428      7812      5772
       180     -564     -420
>> A^6
ans =

     -55440      324576      240480
    -110016      635760      470304
       7200     -42048     -30384
Entering a matrix with complex coefficients of size 2 x 3:
>> C = [2- i 0 0; 1 - i 2*i 2]
C =

      2.0000 - 1.0000i      0      0
      1.0000 - 1.0000i      0 + 2.0000i      2.0000
>> C * A
```

First part: Basic elements

ans =

```
4.0000 - 2.0000i    8.0000 - 4.0000i    12.0000 - 6.0000i  
-4.0000           6.0000 +14.0000i    8.0000 + 8.0000i
```

1.3.10 Relational or logical operations on tables

As with arithmetic operations, logical operations that exist for scalar numbers can also be applied to digital tables. Relational operators make it possible to make logical comparisons between digital values.

The logical value 'True' is called True, and 'False' is called False; These values are of the logical type.

All relational or logical operations refer a result equal to logical value 0 or 1.

- **Relational operators**

Relational operators allow the comparison of two values between them. The following table synthesizes the syntaxes of the different relational operators available:

<i>Syntax of relational operators</i>		
Relational operation	Syntaxe MATLAB	
	symbol	function
A equal to B	A == B	eq(A,B)
A different from B	A ~= B	ne(A,B)
A greater than B	A > B	gt(A,B)
A greater than or equal to B	A >= B	ge(A,B)
A less than B	A < B	lt(A,B)
A less than or equal to B	A <= B	le(A,B)

Example

Let's take some examples of relational operations on any numeric values:

```
>> 20 == 20  
ans =  
    1  
>> 1.4 > 6.2  
ans =  
    0
```

- **Comparison of a table and a scalar**

First part: Basic elements

Example

```
>> A=[2 4; 5 2]
```

```
A =
```

```
    2    4
    5    2
```

```
>> A > 3
```

```
ans =
```

```
    0    1
    1    0
```

Les termes de A supérieurs à 2 donnent 1 (true), les autres 0 (faux). La possibilité d'appliquer une opération relationnelle sur un tableau sera exploitée dans la suite pour calculer les valeurs d'une fonction définie par morceaux.

- **Comparison of two tables**

Example

Now let's take any two tables of the same dimension, and compare them:

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

```
A =
```

```
   -1    6
    3   -4
```

```
>> B=[-9 2.5 ; 1 -2]
```

```
B =
```

```
 -9.0000    2.5000
  1.0000   -2.0000
```

```
>> T = A > B
```

```
T =
```

```
    1    1
    1    0
```

- **Complement**

It may be useful to know if the Table T Result of a test contains that non-zero values (or logical values true) or if there is at least one non-null value (or logical value True).

To do this, you can use the All (T) and Any (T) functions, respectively. By default, if T is not a vector, these functions test the presence of non -zero values according to the columns of T.

Example

Let's resume the previous example:

```
>> all(T)
```

First part: Basic elements

```
ans =  
  
    1    0  
>> any(T)  
ans =  
    1    1  
>> all(all(T))  
ans =  
    0  
>> any(any(T))  
ans =  
    1
```

- **Logical operators**

Logical operators are operators that apply exclusively to logical type values. They allow the combination of logical conditions. The following table gives the syntax of logical operators available in Matlab:

Syntax of logical operators		
Relational operation	Syntaxe MATLAB	
	symbol	function
A et B	A & B	and(A,B)
A ou B	A B	or(A,B)
A or exclusive B		xor(A,B)
Negation of A	~A	not(A)

While this table recalls the result of these logical operations:

Logic table					
A	B	and(A,B)	or(A,B)	xor(A,B)	not(A)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

A and B can be logical scalar values or logical values tables of the same dimensions. For tables, these operations apply to term.

First part: Basic elements

- **Short-circuit logic operators**

We call logical operators short-circuit operators `**` and `||`. Unlike operators `*` and `|`, short-circuit logical operators do not assess the second operand, if the value of the first already allows you to know the overall result.

- **Table lengths**

The `size` function applied to a matrix returns a table of two integers: the first is the number of lines, the second the number of columns. The command also works on the vectors and returns 1 for the number of lines (resp. Columns) of a line vector. For vectors, the `Length` command is more practical and returns the number of components of the vector, whether line or column.

1.3.11 Creation of the .m file of a function

To define a new function in Matlab, we write the definition of the function in a file with an extension `.m` (M-File function). The name of the file must be the name of the first defined function (the visible only one). It is performed by typing the name of the function with the list of arguments in parentheses.

The first line of the file of function must follow the following syntax:

Function arguments of exit = name (arguments of entrance)

They must therefore declare stocks calculated by function in arguments of exit, and parameters of function in arguments of entrance.

Example 1

Recursive function computing by dichotomy of the root of a function if $f(x)=x^3-3x-7$ and x on $[a, b]=[1, 4]$.

```
function x=racinefun(a,b)%[a,b]=[1,4]
% calcul la racine de f(x) définie ci-après sur [a,b]
fa=f(a); fb=f(b); x=(a+b)/2;
if (fa*fb>0) x=-Inf; return; end;
while (b-a)>eps*x
    x=(a+b)/2; fx=f(x);
    if(sign(fx)==sign(fa))
        a=x; fa=fx;
    else
        b=x; fb=fx;
    end;
end;
% definition de f(x) (fonction locale)
function y=f(x)
y=x^3-3*x-7;
```


First part: Basic elements

The function root ci over is written in a racinefun.m file. To carry it out, they knock simply

```
>> racinefun(1,4)
```

```
ans =
```

```
2.4260
```

Example 2

we want to create a function which calculates the cube of a number X then returns the result y.

The program will be:

```
function y = cub3(x)
```

```
% cette fonction calcule le cube d'un nombre x
```

```
    y = x^3 ;
```

```
end
```

```
>> cub3(5)
```

```
ans =
```

```
125
```

The function will be recorded under the name "cub3.m". A function must always end with an end delimator ("end").

Example 3

```
function y = foc(x)
```

```
% cette fonction calcule le cube d'un nombre x
```

```
    y =cos( x.^2)+x.^3+10./(x-1) ;
```

```
end
```

Let be the function:

a) Start to open a text editor: In the Matlab command window:

File -> New -> M-file

With version 6.5. the default text editor is the 'M-File Editor' application.

b) Give this function a name (in this example foc) and enter its mathematical expression:

c) Safeguard the file in your working directory (for instance c:USERS)

Name:foc

Extension: .m

d) Add the way of the directory where is your foc.m file

```
>> path(path,'c:\USERS')
```

File -> Set Path -> Add Folder

-> Save -> Close

- **Valuation of a function**

Calculation of $y(x=0)$:

```
>> foc(0)
```

First part: Basic elements

```
ans =  
    -9  
Calculation of y ( x = 10 ) :  
>> foc(10)  
ans =  
    1.0020e+003  
>> foc(1)  
Warning: Divide by zero  
ans =  
Inf  
With a vector in argument, the function returns a vector:  
  
>> foc([0 1 2 3 4 5])  
ans =  
    -9.0000         Inf    17.3464    31.0889    66.3757    128.4912>> >>  
x=0:6  
x =  
     0     1     2     3     4     5     6  
>> y = foc(x)  
y =  
    -9.0000 Inf    17.3464    31.0889    66.3757    128.4912    217.8720  
With a matrix in argument, the function returns a matrix:  
>> foc( [ 0 1 2 3 ; 5 6 7 8] )  
ans =  
    -9.0000         Inf    17.3464    31.0889  
    128.4912 217.8720    344.9673    513.8204
```

Exercise:

Be the matrix $A = \begin{pmatrix} 1 & 4 & 1 & 1 \\ 1 & 7 & 1 & 2 \\ 1 & 4 & 1 & 2 \\ 3 & 10 & 2 & 5 \end{pmatrix}$

a/ Create matrix A using MATLAB

b/ Extract the following blocks from the matrix A:

$$b1 = \begin{pmatrix} 1 & 2 \\ 1 & 2 \\ 2 & 5 \end{pmatrix}, b2 = \begin{pmatrix} 1 & 7 & 1 \\ 1 & 4 & 1 \\ 3 & 10 & 2 \end{pmatrix}.$$

c/ Give the values of $A(3,2)$, $A(3:4,3)$, $A(3,:)$, $\text{tril}(A)$

d/ Write with MATLAB the matrix D defined by:

$D = Id - A.*A^t$ where Id denotes the identity matrix and A^t the transpose matrix of A.

First part: Basic elements

e/ Define matrix $\mathbf{B} = [0.5*\text{ones}(4,2) - 2*\text{ones}(4,2)]$ and give acquired result. Is the product of A and B possible ? Justify your answer. If yes which is the MATLAB order which allows to make this product?

Solutions

A =

```
1     4     1     1
1     7     1     2
1     4     1     2
3    10     2     5
```

b1 =

```
1     2
1     2
2     5
```

b2 =

```
1     7     1
1     4     1
3    10     2
```

ans =

```
4
```

ans =

```
1
2
```

ans =

```
1     4     1     2
```

ans =

```
1     0     0     0
1     7     0     0
1     4     1     0
3    10     2     5
```

D =

```
0     -4     -1     -3
-4    -48     -4    -20
-1     -4     0     -4
-3    -20     -4    -24
```

B =

```
-1.5000    -1.5000
-1.5000    -1.5000
-1.5000    -1.5000
-1.5000    -1.5000
```

ans =

```
4     4
```

ans =

```
4     2
```

ans =

First part: Basic elements

-10.5000	-10.5000
-16.5000	-16.5000
-12.0000	-12.0000
-30.0000	-30.0000

1.4 Graph in two dimensions and management of the graphic windows

- 1.4.1 Draw the graph of a function**
- 1.4.2 The Plot command**
- 1.4.3 The loglog command**
- 1.4.4 Caption a figure**
- 1.4.5 Display multiple curves in a single window**

First part: Basic elements

1.4 Graph in two dimensions and management of the graphic windows

1.4.1 Draw the graph of a function

- **Functions**

fplot	trace point by point the graph of a function
grid	adds a grid
xlabel	adds a legend for the x-axis
ylabel	adds a legend for the y-axis
title	adds a title
axis	modifies the scales of the axes
zoom	zooms in
gtext	places a legend with the mouse
hold	adds a graph in the current window
figure	creates a new window

The fplot command draws the graph of a function over a given interval.

Syntax is: `fplot('nomf', [xmin , xmax])` où

- nomf is either the name of an incorporated matlab function, or an expression defining a function of the variable X, or the name of a user function.
- [xmin , xmax] is the interval for which the function graph is drawn.

Let us illustrate by examples the three ways to use the command fplot.

Example 1

Draw the function graph `'x*cos(2*x)+x^2'` in `[-2*pi 2*pi]`

```
>> fplot('x*cos(2*x)+x^2', [-2*pi 2*pi])
```

First part: Basic elements

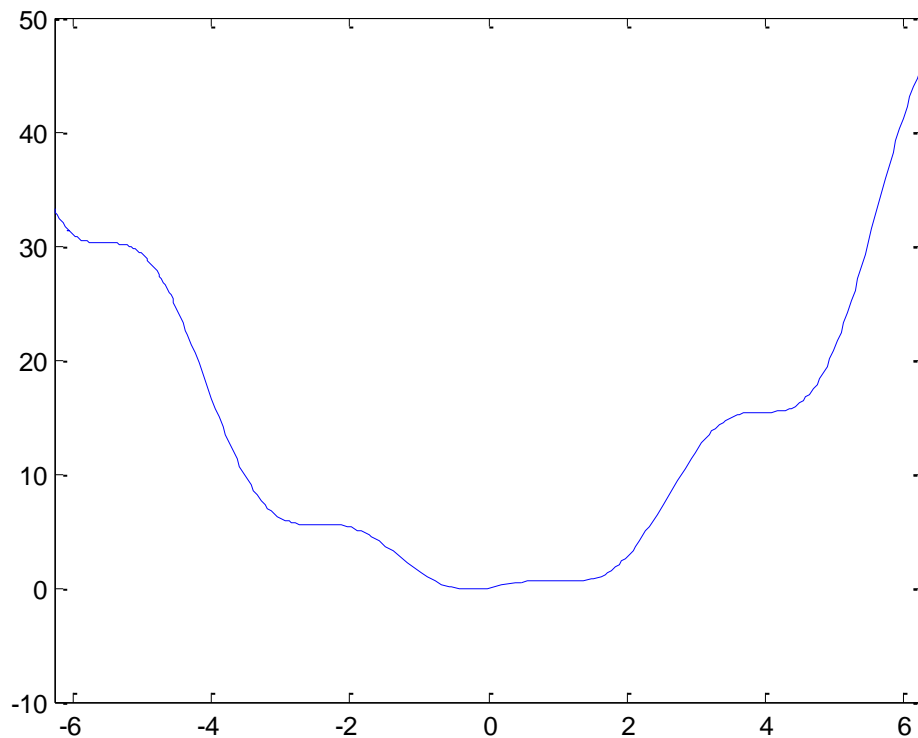


Fig. 1 – Function graph ' $x \cos(2x) + x^2$ ' in $[-2\pi, 2\pi]$.

To plot the graph of the function $h(x)$, we can define the user function h in the file $h.m$ as follows:

```
function y=h(x)
    y=... ;
>>fplot('h', [-x x]).
```

To plot the graph of the function $h(x)$, we can define the user function h in the file $h.m$ as follows:

```
fplot('h', [-x x]).
```

Example 2

plot the graph of the function $2x^2 \sin(-2x) + x$ in $[-\pi, \pi]$

```
function y=foc(x)
y='2*x^2*sin(-2*x)+x';
end
ans =

2*x^2*sin(-2*x)+x
>> fplot('2*x^2*sin(-2*x)+x', [-pi pi])
```

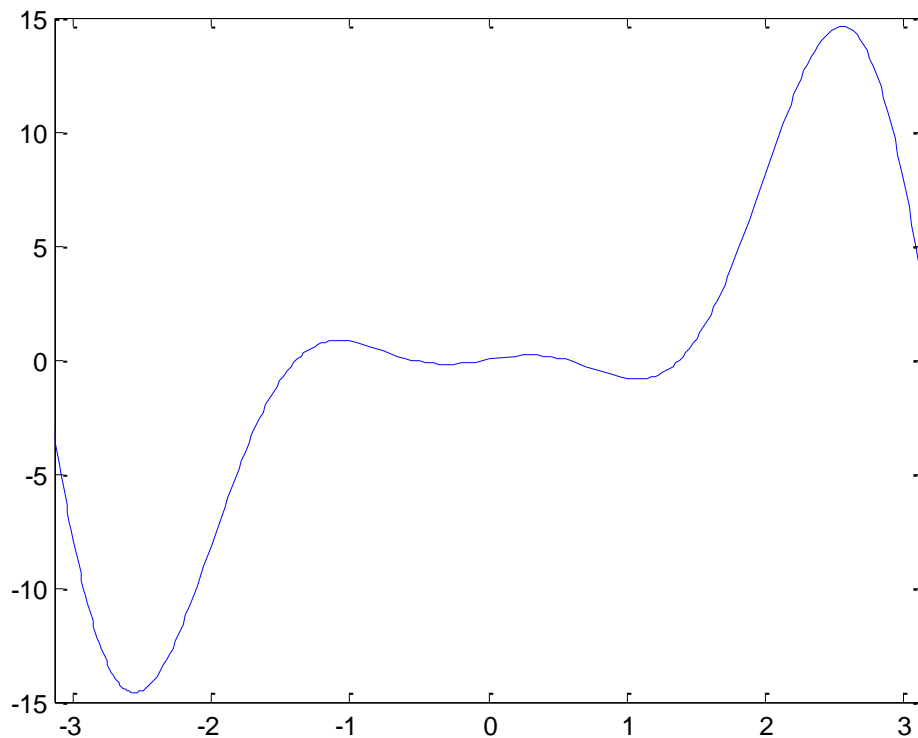


Fig. 2 – Function graph $2*x^2*\sin(-2*x)+x$ in $[-\pi, \pi]$.

It is possible to draw several functions on the same figure. It is necessary to use for this the `fplot` command as follows: `fplot('nom_f1 , nom_f2 , nom_f3', [x_min , x_max])` where `nomf_f1`, `nom_f2`, `nom_f3` is either the name of an embedded matlab function, or an expression defining a function of variable `x`, or the name of a user function.

It is also possible to manage the bounds of the values in ordinates. To limit the graph to the ordinates between the values `y_min` and `y_max` we will pass as the second argument of the `fplot` command the array `[x_min , x_max , y_min , y_max]`, [4].

Example 3

plot the graph of the function $-\sin(-x)$, $x*\cos(-x)$ in $[-3, 2, -2, 1]$
`fplot('[-sin(-x) , x*cos(-x)]', [-3 ,2, -2 ,1])`

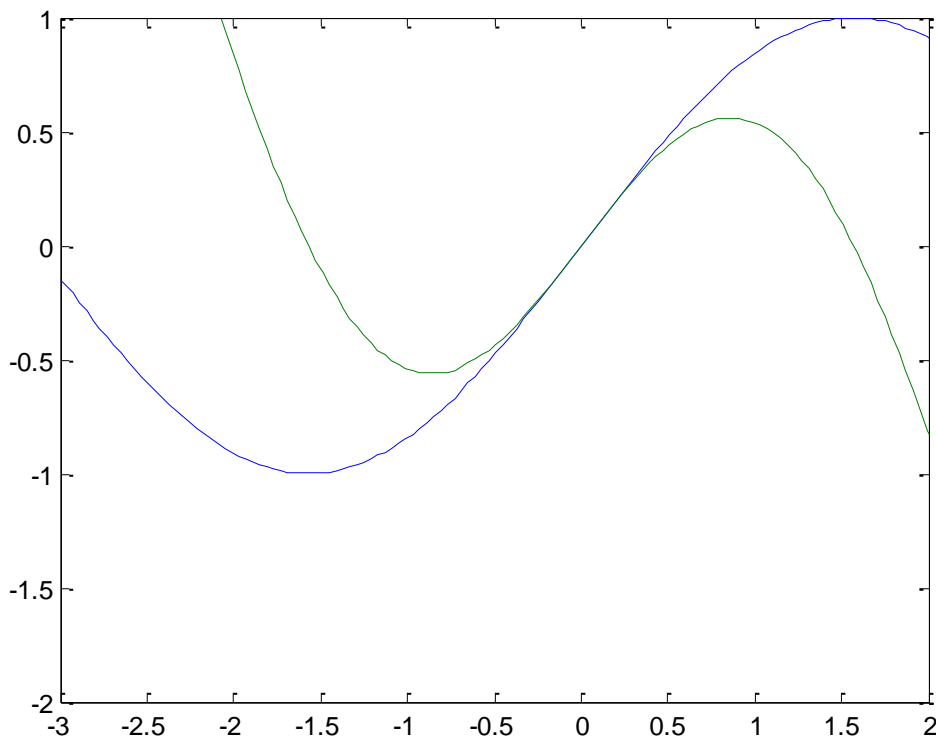


Fig. 3 – Function graph ($-\sin(-x)$, $x\cos(-x)$) in $[-3, 2, -2, 1]$.

1.4.2 The Plot command

The plot command allows you to draw a set of coordinate points (x_i, y_i)

$i = 1, \dots, N$. The syntax is `plot(x,y)` where x is the vector containing the x values on the abscissa and y is the vector containing the y_i values on the ordinate. Of course the vectors x and y must be of the same dimension but they can be row or column vectors.

By default, the points (x_i, y_i) are connected to each other by straight segments.

Example 1

plot the graph of the function $h(x) = 2x \sin(2x) - 1$ in $[-\pi, \pi]$

```
x=[-5*pi:0.001:5*pi];  
y = 2*x.*sin(2*x)-x;  
plot(x,y)
```

First part: Basic elements

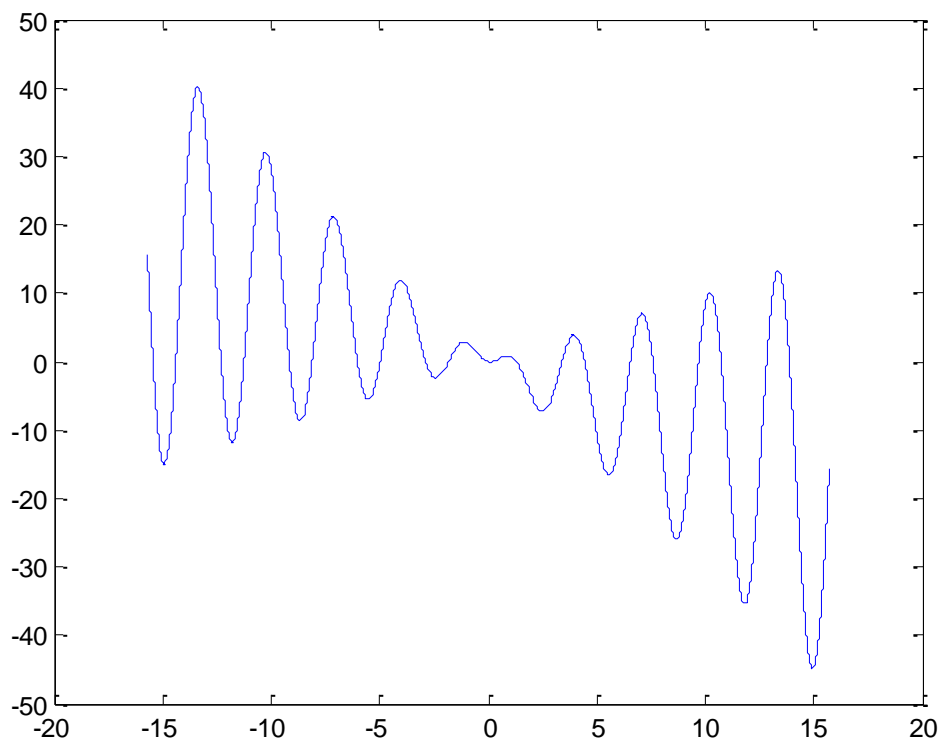


Fig. 4– Function graph $2x \cdot \sin(2x) - x$ in $[-5\pi, 5\pi]$.

We can specify to matlab what should be the color of a curve, what should be the line style and / or what should be the symbol at each point (xi, yi). For this we give a third input parameter to the plot command which is a string of 3 characters of the form 'cst' with c denoting the color of the line, s the symbol of the point and t the line style. The possibilities are as follows:

Color Name	Short Name
'red'	'r'
'green'	'g'
'blue'	'b'
'cyan'	'c'
'magenta'	'm'
'yellow'	'y'
'black'	'k'
'white'	'w'

: Point

o: Circle

First part: Basic elements

x: X-Mark

+: Plus

s: Square

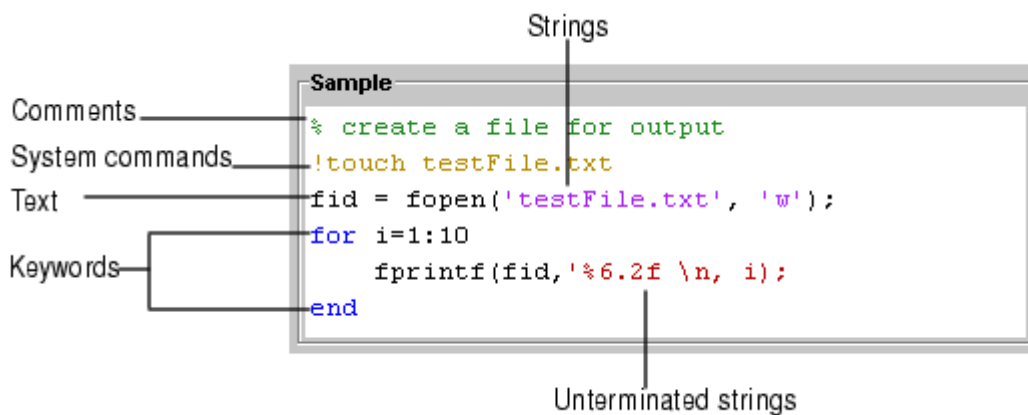
*: Star

d: Diamond

v: Triangle (down)

^: Triangle (up)

<: Triangle (left)



The default colors are listed here:

Keywords--Flow control functions, such as `for` and `if`, as well as the continuation ellipsis (`...`), are colored blue.

Comments--All lines beginning with a `%`, designating the lines as comments in MATLAB, are colored green. Similarly, the block comment symbols, `%{` and `%}`, as well as the code in between, appear in green. Text following the continuation ellipsis on a line is also green because it is a comment.

Strings--Type a string and it is colored maroon. When you complete the string with the closing quotation mark (`'`), it becomes purple. Note that for functions you enter using command syntax instead of function syntax, the arguments are highlighted as strings. This is to alert you that in command notation, variables are passed as literal strings rather than as their values. For more information, see MATLAB Command Syntax in the MATLAB Programming documentation.

Unterminated strings--A single quote without a matching single quote, and whatever follows the quote, are colored maroon. This might alert you to a possible error.

First part: Basic elements

System commands--Commands such as the ! (shell escape) are colored gold.

Errors--Error text, including any hyperlinks, is colored red.

Example 2

```
X = -10 : 0.05 : 10;  
Y = x.^4 - x.^2;  
plot (x, y, 'r')
```

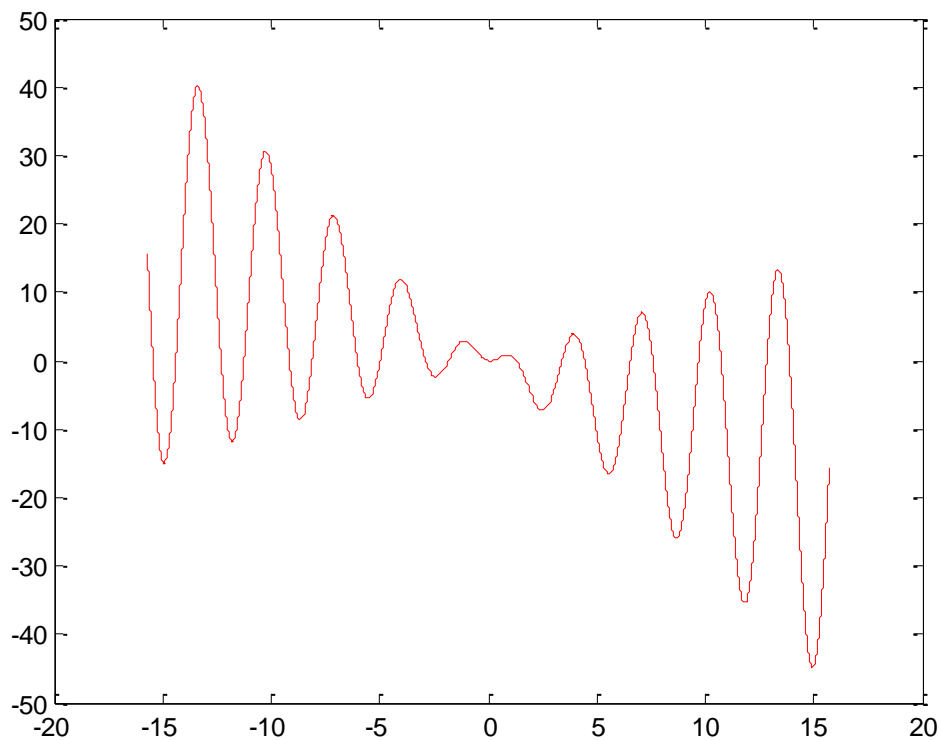


Fig. 5-- Function graph $Y = x.^4 - x.^2$; in $[-10 \ 10]$.

```
X = -10 : 0.5 : 10;  
Y = x.^4 - x.^2;  
plot (x, y, 'dr')
```

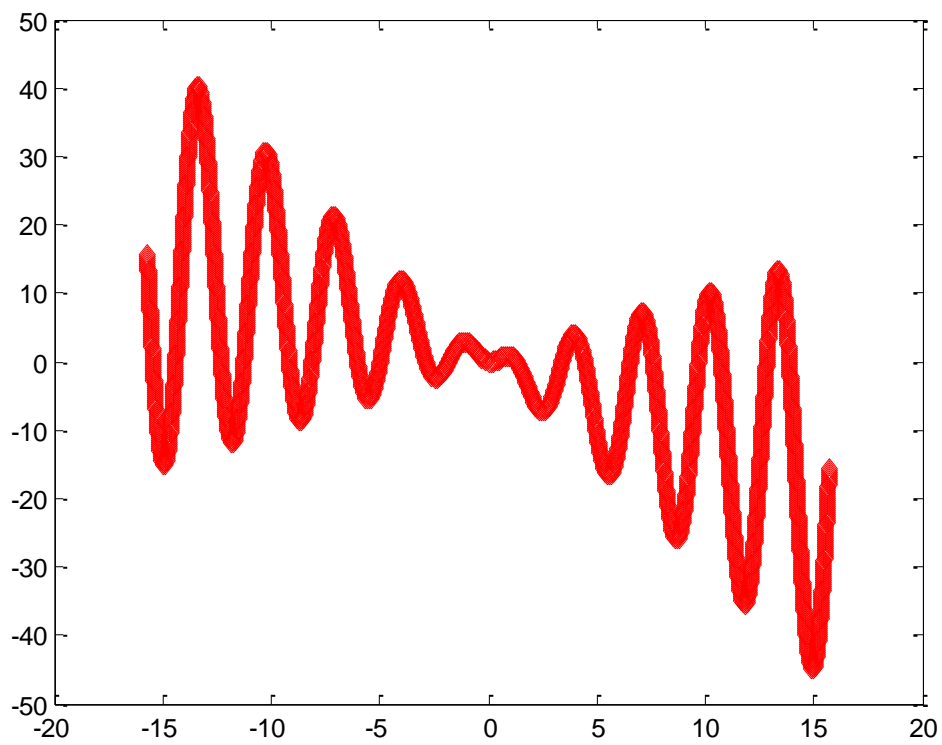


Fig. 6– Function graph $Y = x.^4 - x.^2$; in $[-10 \ 10]$.

Example 3

```
x = [-4:0.01:4];  
y = x.^3.*sin(3*x)-x; z = 2*x.*sin(2*x)-x;  
plot(x,y,'b-',x,z,'r:');
```

First part: Basic elements

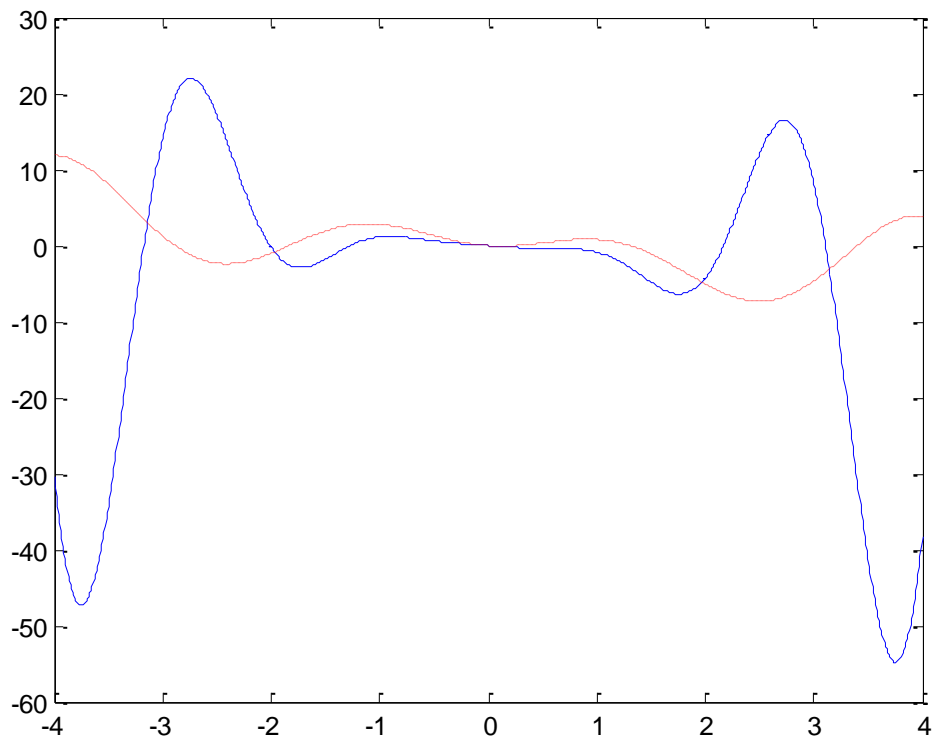


Fig. 7– Function graph y and z.

Example 4

```
N=100;  
x = rand(1,N); y = rand(1,N);  
plot(x,y,'bd')
```

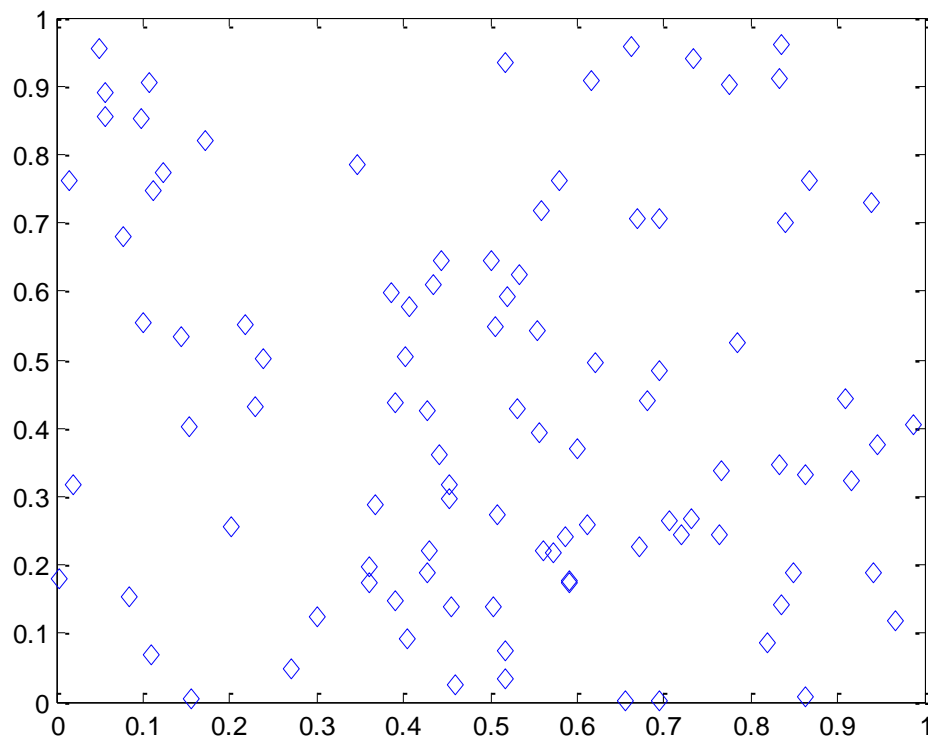


Fig. 8 – Function graph x and y .

1.4.3 The command loglog

If x and y are two vectors of the same dimension, the `loglog(x,y)` command displays the $\log(x)$ vector against the $\log(y)$ vector. The `loglog` command is used in the same way as the `plot` command

Example

```
x = [1:40:3000];  
y = x.^3;  
loglog(x,y)
```

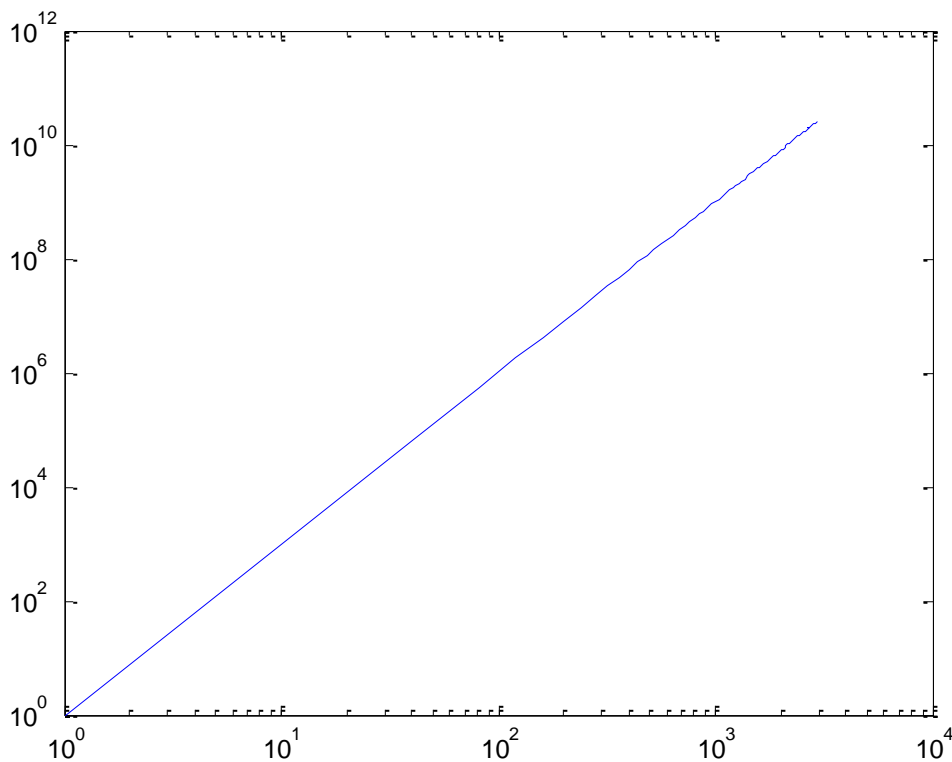


Fig. 9 – Result of the order $\log\log(x,y)$.

1.4.4 Legend of a figure

It is recommended to put a legend to a figure. The `xlabel` command is used to caption text below the x-axis. The syntax is `xlabel('legend')` to get the word legend in legend. The `ylabel` command does the same for the y-label axis. The `title` command is used to give a title to the figure. The syntax is `title('the title')` to get the title as the title.

You can also write a given text at a specific position on the figure thanks to the `text` command. The syntax is `text(posx, posy, 'a text')` where `posx` and `posy` are the coordinates of the point where a text is to begin writing.

The `gtext` command allows you to place the text at a chosen position on the figure using the mouse. The syntax is `gtext('a text')`. A sight, which is moved using the mouse, appears. All it takes is a "mouse click" for the text to appear at the selected position.

It is possible with these commands to display a value contained in a variable in the middle of the text. To do this, we build an array of type character string by converting the value contained in the variable into a character string using the `num2str` command.

For example, suppose the `numex` variable contains the number of the sample being processed, say 5. The title of the figure Example number 5 is obtained by the statement: `title(['Exemple numero ', num2str(numex)])`.

Example

```
t = [0:0.001:8];  
h = 15*exp(-t) - 8*exp(-5*t);
```


First part: Basic elements

```
plot(t,h);  
grid  
xlabel('temps en minutes')  
ylabel('concentration en gramme par litre')  
title(['evolution de la concentration du produit ,   num2str(P),  
... au cours du temps'])  
gtext('concentration maximale')
```

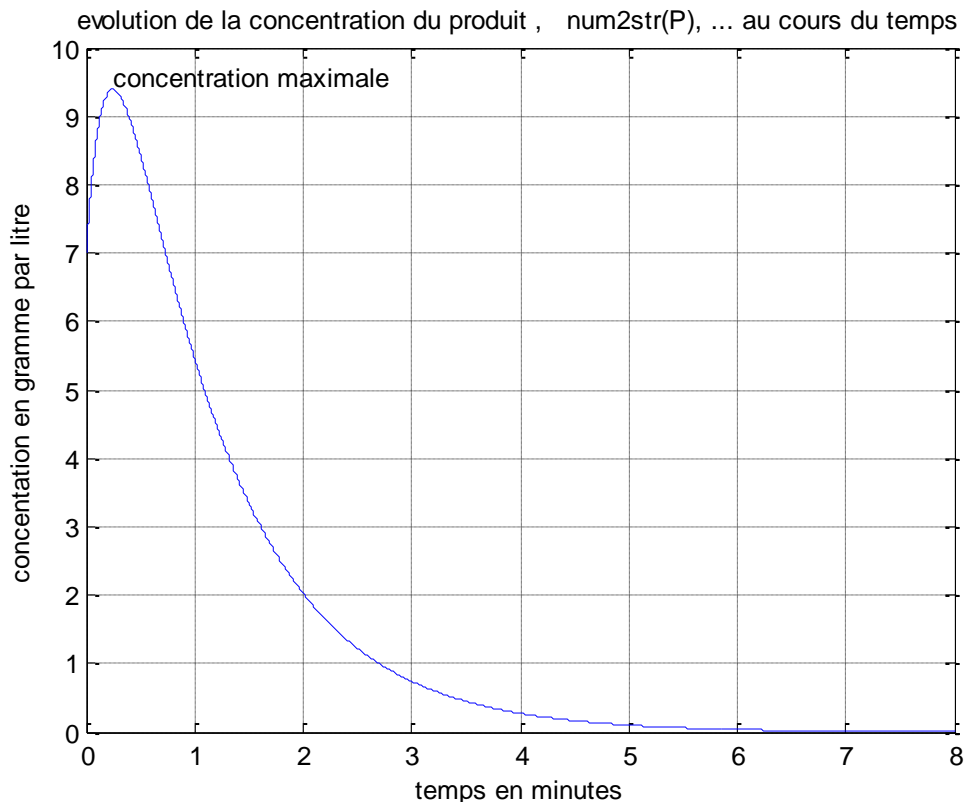


Fig. 10 – Graph of the function h.

1.4.5 Display multiple curves in a single window

It is possible to display several curves in the same graphics window thanks to the hold on command. The results of all graphics statements executed after calling the hold on command will be overlaid on the active graphics window. To restore the previous situation (the result of a new graphic instruction replaces the previous drawing in the graphics window) we will type hold off.

Example 1

```
clc  
figure  
hold on  
fplot('-2*exp(x)', [-1 1], 'k')  
fplot('log(x)', [1/3 4], 'g')  
plot([-1:0.01:3], [-1:0.01:3], 'r')  
grid on  
hold off
```

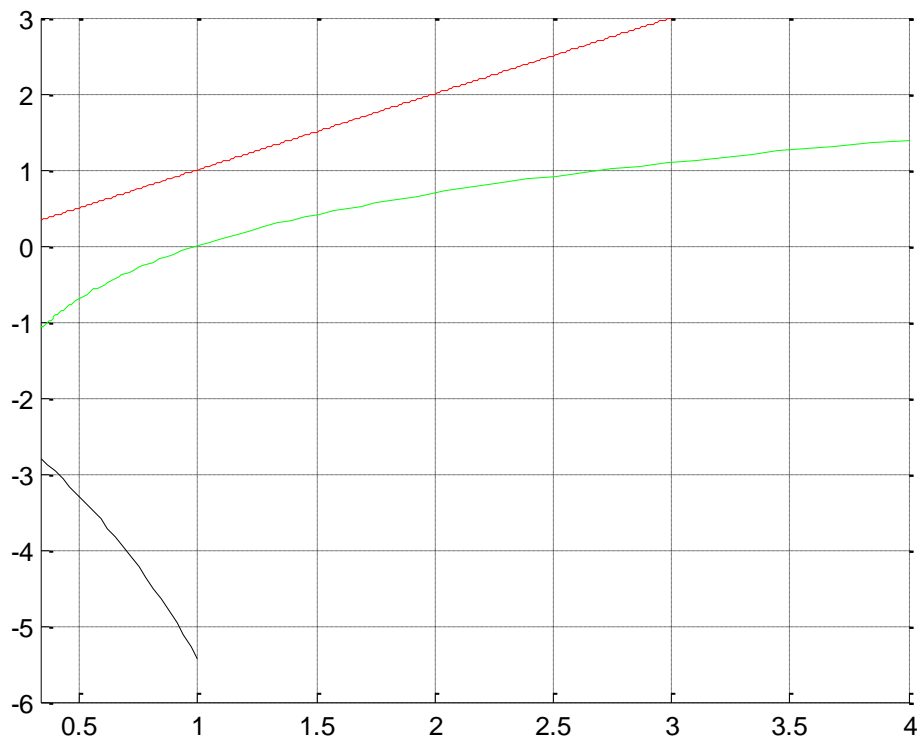


Fig. 11 – Multiple curves in the same graphics window.

It is possible to break down a window into panes and display a different figure on each of these subwindows using the subplot command. The syntax is subplot(m,n,i) where

- m is the number of sub-windows vertically;
- n is the number of subwindows horizontally;
- i is used to specify in which pane the display should be made.

Example 2

```
clc
figure
subplot(2,3,1), fplot('cos(-2*x)',[1 4*pi]), title('cosinus(-2*x)'), grid
subplot(2,3,2), fplot('sin(-5*x)',[1 4*pi]), title('sinus(-5*x)'), grid
subplot(2,3,3), fplot('tan(x^2+1)',[-pi pi]), title('tangente(x^2+1)'), grid
subplot(2,3,4), fplot('acos(x^2-1)',[-2 2]), title('arc-cosinus(x^2-1)'), grid
subplot(2,3,5), fplot('asin(x^2)',[-2 2]), title('arc-sinus(x^2)'), grid
subplot(2,3,6), fplot('atan(x^2+x)',[-sqrt(9) sqrt(9)]), title('arc-tangente(x^2+x)'), grid
```

First part: Basic elements

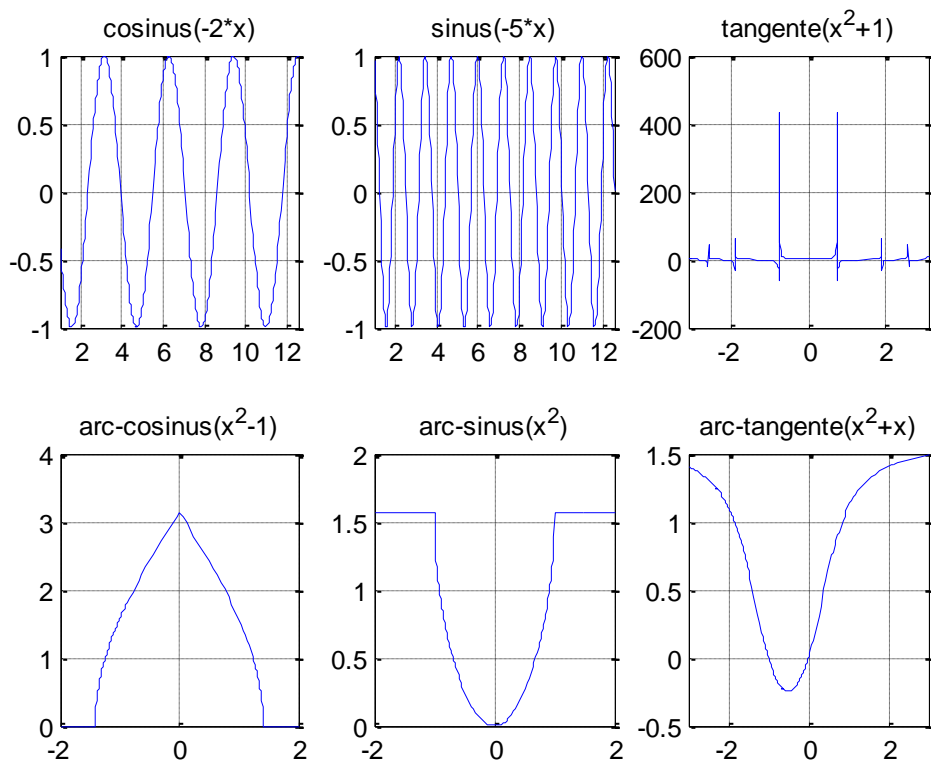


Fig. 12 – Graphic window broken down into sub-windows.

Example 3

Time (hours)	0	2	4	6	8	10	12	14	16
Temperature (°C)	20	23	30	33	32	37	34	39	36

```

clc
Time = [0 2 4 6 8 10 12 14 16];
Temperature = [20 23 30 33 32 37 34 39 36];
plot(Time , Temperature)
grid on
xlabel ( ' Time ( hours)' )
ylabel ( ' Temperature  ( °C)' )
title ( ' Monitoring of temperature ')
axis ( [ 0 18 10 40 ] )

```

First part: Basic elements

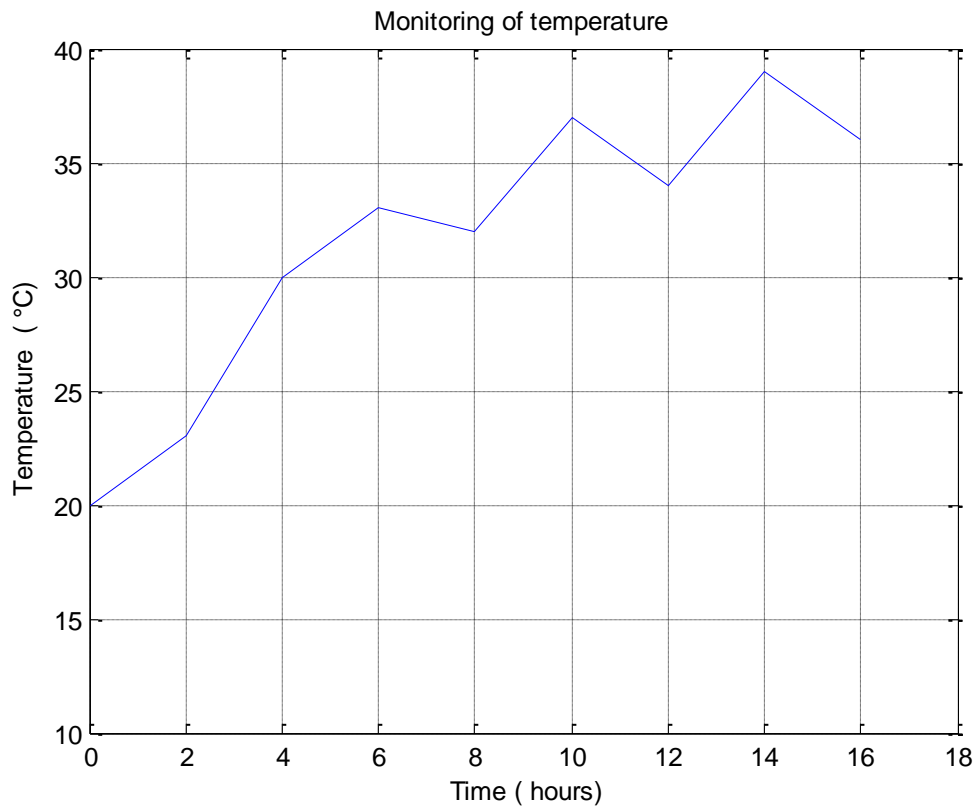


Fig. 13 – Graph of the Monitoring of temperature.

Example 4

```

$$y = \frac{t(t^2+1)}{10}$$
  
t = 0 : 0.001 : 2  
y = t.*(1 + t.^3)./5  
plot ( t , y ) ; grid on
```

First part: Basic elements

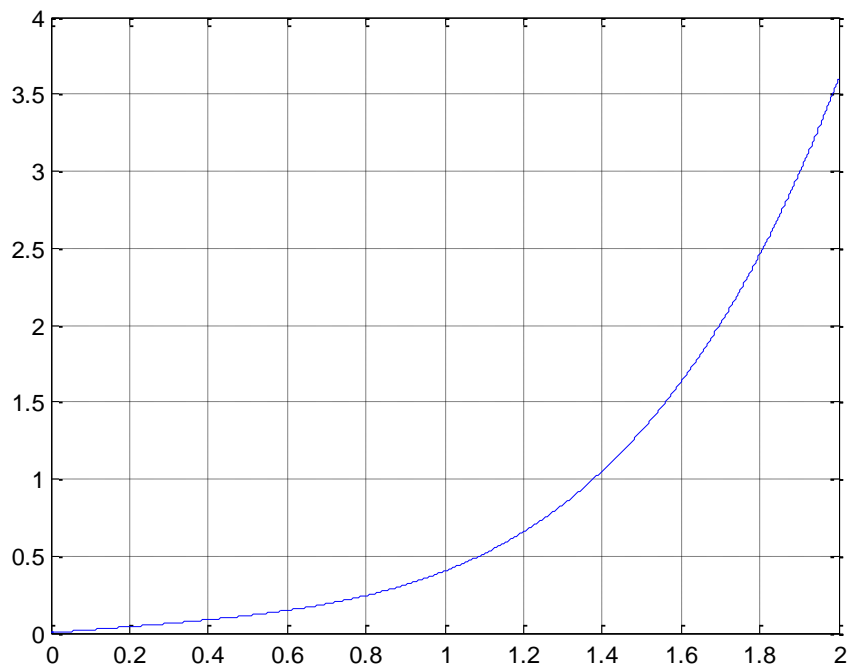


Fig. 14 – Graph of the function y .

Example 5

```
x = 4*cos(2*t), y = sin(3*t)
```

```
t = 0 : pi/200 : 2*pi
```

```
x = 4*cos(2*t)
```

```
y = sin(3*t)
```

```
plot ( x , y )
```

```
grid on
```

First part: Basic elements

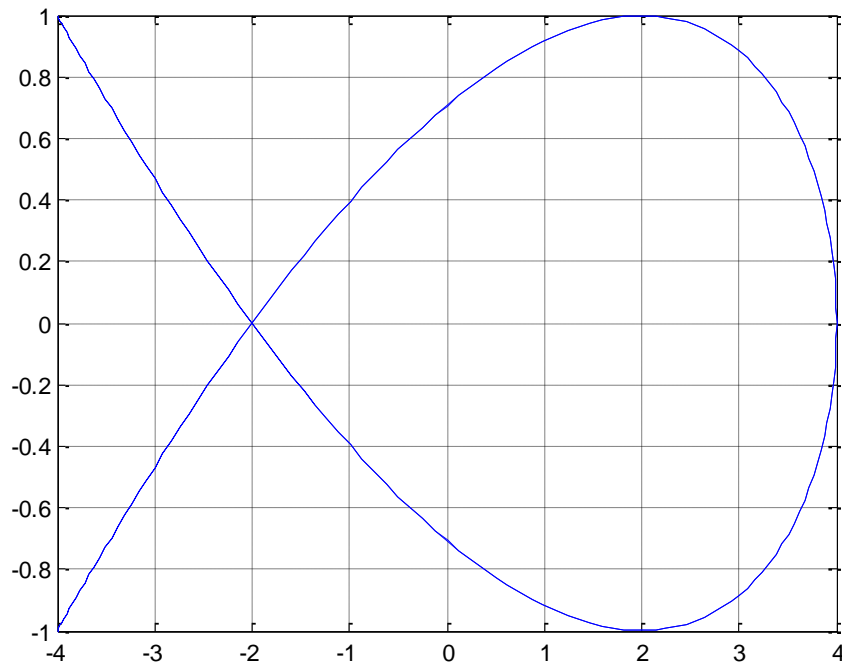


Fig. 15 – Graph of the function x and y.

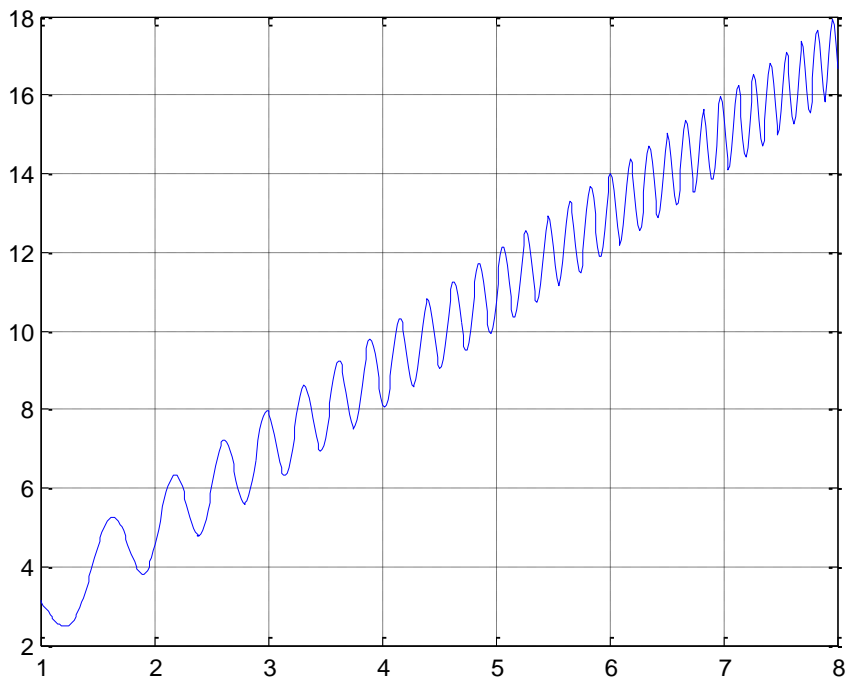
Example 6

$$y = f(x) = 1 + 2x + \sin(3x^2)$$

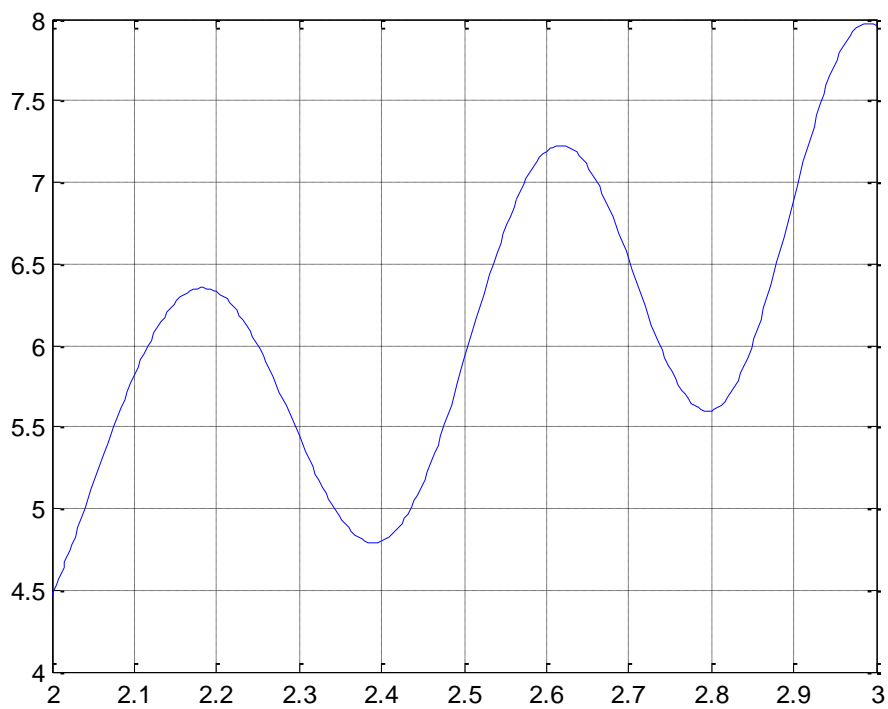
• First method:

```
fplot('1+ 2*x + sin(3*x*x)', [ 1 8 ]),grid on
```

First part: Basic elements



```
fplot('1+ 2*x + sin(3*x*x)', [ 2 3 4 9 11]),grid on
```



```
grid off  
xlabel('axis of abscissa ')  
ylabel('axis of ordinates')  
title('y=f(x) ')
```

First part: Basic elements

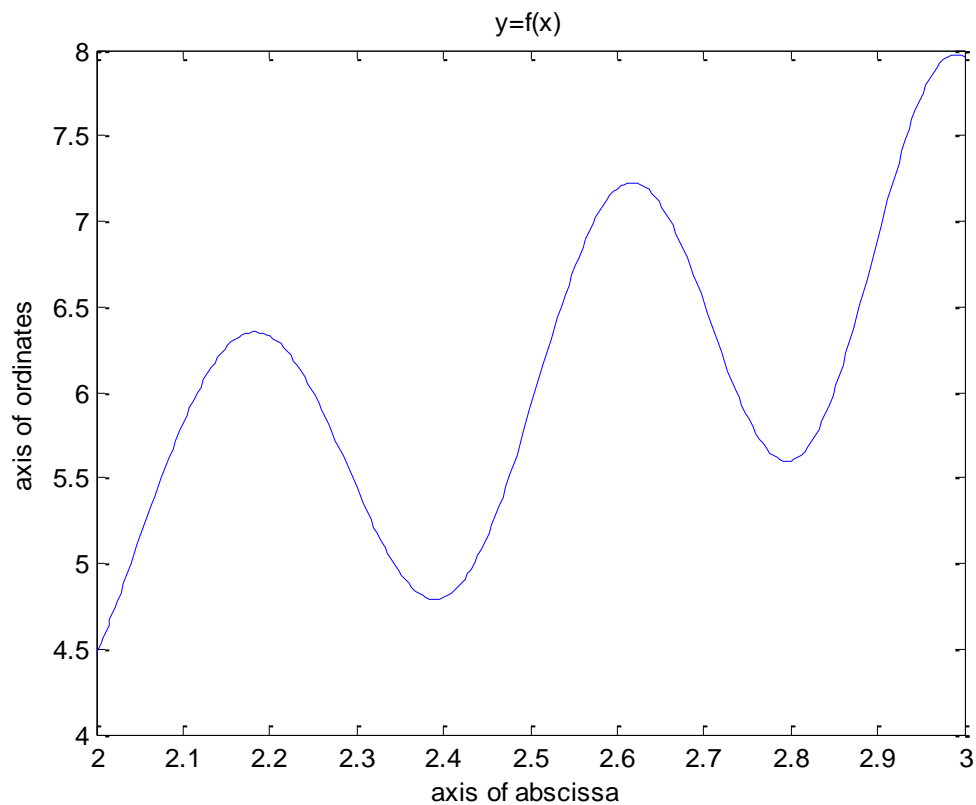


Fig. 16 – Graph of the function $f(x)$.

zoom on

right click: zoom out

left click: zoom in

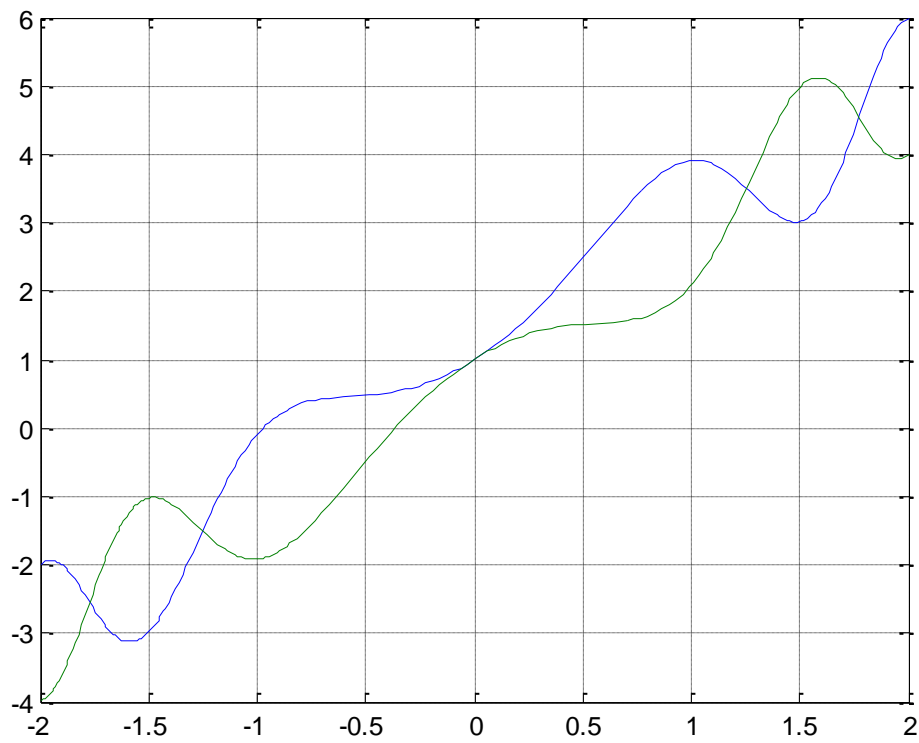
left click and drag: zoom of an area

zoom off

To draw several graphs in the same window:

```
fplot('1+ 2*x + sin(2*x*x) , 1+ 2*x - sin(2*x*x) ', [ -2 2 6  
]),grid on
```


First part: Basic elements



```
gtext('fonction 1')  
gtext('fonction 2')
```

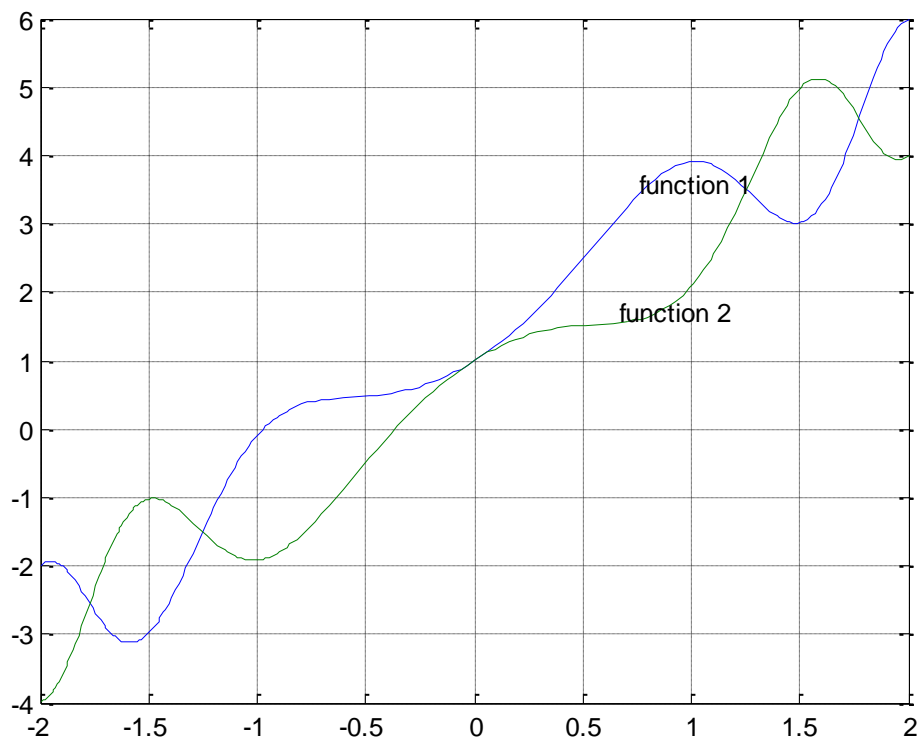


Fig. 16 – Graph of the function $f_1(x)$ end $f_2(x)$.

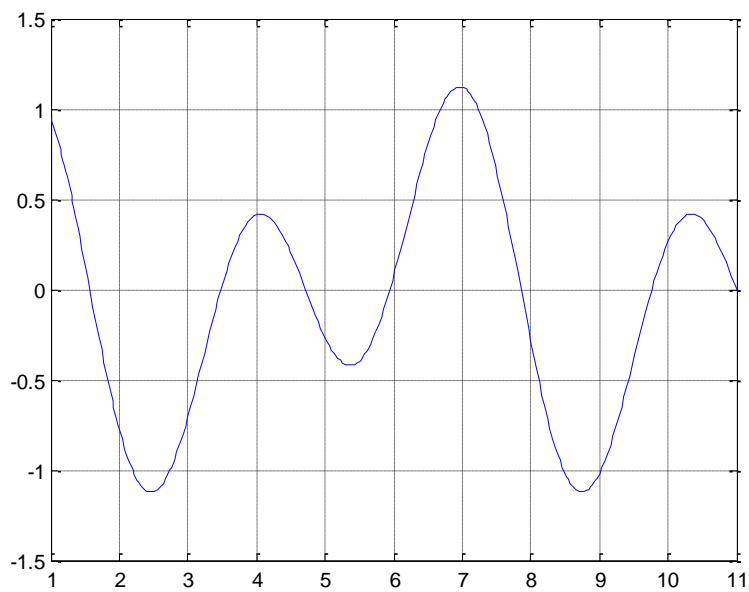
First part: Basic elements

• Second method

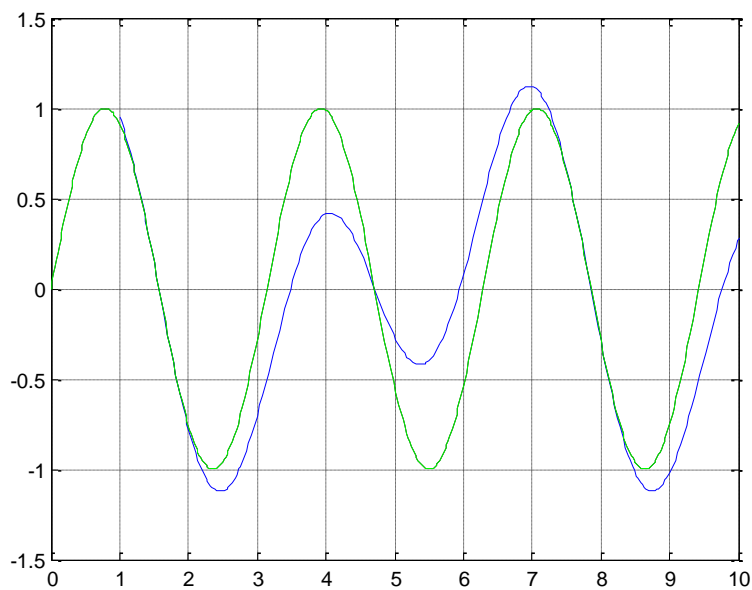
$$y = f(x) = \sin(2x) + \frac{\cos(x)}{2} - \frac{\sin(x)}{4}$$

We are going to create the function's .m file:

```
function y=f2(x)
y=sin(2*x)+0.5.*cos(x)-0.25.*sin(2*x);
>> fplot('f2', [ 1 11]),grid on
```



```
>> hold on
>> fplot('sin(2*x)', [ 0 10 ], 'g')
```



First part: Basic elements

```
>> [X Y] = fplot ( 'sin(2*x) ' , [ 1 11 ] )
```

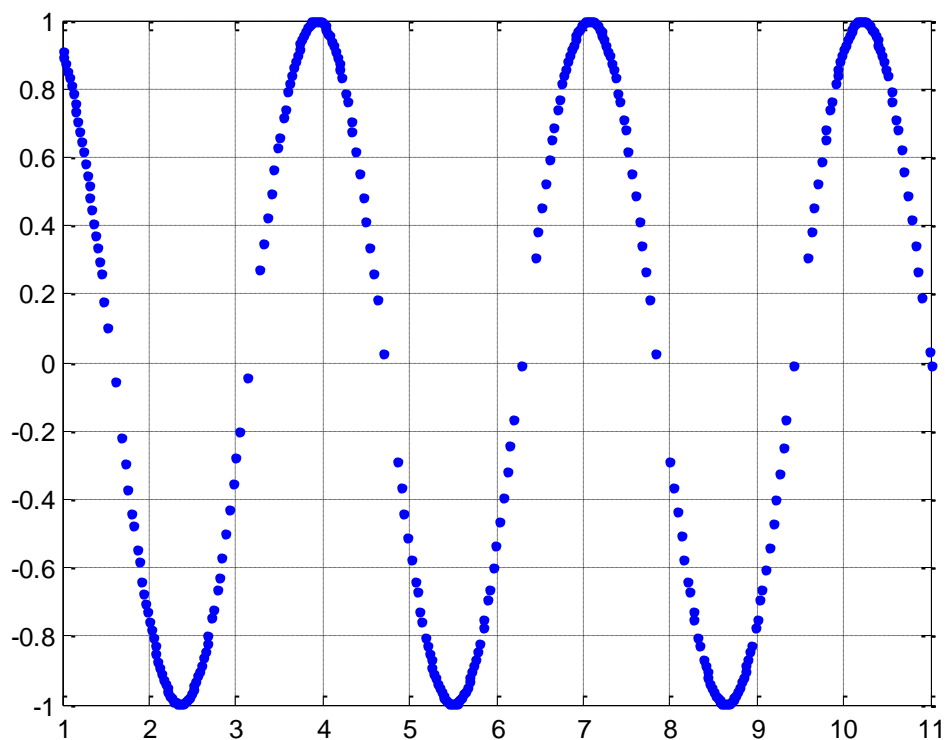
```
X =
```

```
1.0000  
1.0200  
1.0400  
1.0600  
.....  
10.9000  
10.9800  
11.0000
```

```
Y =
```

```
0.9093  
0.8919  
0.8731  
.....  
0.1900  
0.0311  
-0.0089
```

```
>> fplot ( 'sin(2*x) ' , [ 1 11 ] , '.' ),grid on
```



First part: Basic elements

1.5 3D graphic

1.4.2 Draw the level lines of a function of 2 variables

1.4.3 Represent an equation surface $z=g(x,y)$

1.4.4 Represent a parameterized surface

First part: Basic elements

1.5 3D graphic

1.5.1 Draw the level lines of a function of 2 variables

- **Function**

meshgrid	(see example)
mesh	(see example)
meshc	(see example)
meshz	(see example)
contour	(see example)
view	Adjust the viewing angle
grid	Add a grid
xlabel	Add a legend for the abscissa axis
ylabel	Add a legend for the ordinate axis
zlabel	Add a legend for the Z axis
title	Add a title
hold	Add a graph to the current window
figure	Create a new window
plot3	trace point by point a 3D graph
view	Adjust the viewing angle

The outline control allows you to draw the level lines of a function of 2 real variables. This function can be defined by a Matlab expression, or be defined as a user function. To trace the level lines of the $G(X, Y)$ function for $x_{\min} < x_{\max}$ and $y_{\min} < y_{\max}$ we proceed as follows:

creation of a mesh, with a mesh of length h , of the domain $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ using the command **meshgrid**,

$[X,Y] = \text{meshgrid}(x_{\min}:h:x_{\max}, y_{\min}:h:y_{\max}).$

- Evaluation of the function at the nodes of this mesh, either by calling the user function defining the function, $Z = g(X,Y)$ or directly by defining the function with a MATLAB expression.
- Display of the level lines thanks to the command `contour, contour(X,Y,Z)`.

First part: Basic elements

Example 1

Thus to draw the contour lines of the function $f(x,y) = yxe^{(x^2-y^2)}$ on the domain

$[-1, 1] \times [-1, 1]$ taking a mesh of mesh of length $h=0.02$:

```
clc; clear all
[X,Y] = meshgrid(-1:0.02:1, -1:0.02:1);
Z = Y*X.*exp(-X.^2-Y.^2);
contour(X,Y,Z) , grid on
You can also write a user function g.m,
```

```
function x3 = g(x1,x2)
x3 = x2* x1.*exp(-x1.^2-x2.^2);
and execute
```

```
>> [X,Y] = meshgrid(-1:0.02:1, -1:0.02:1);
>> Z = g(X,Y);
>> contour(X,Y,Z) , grid on
```

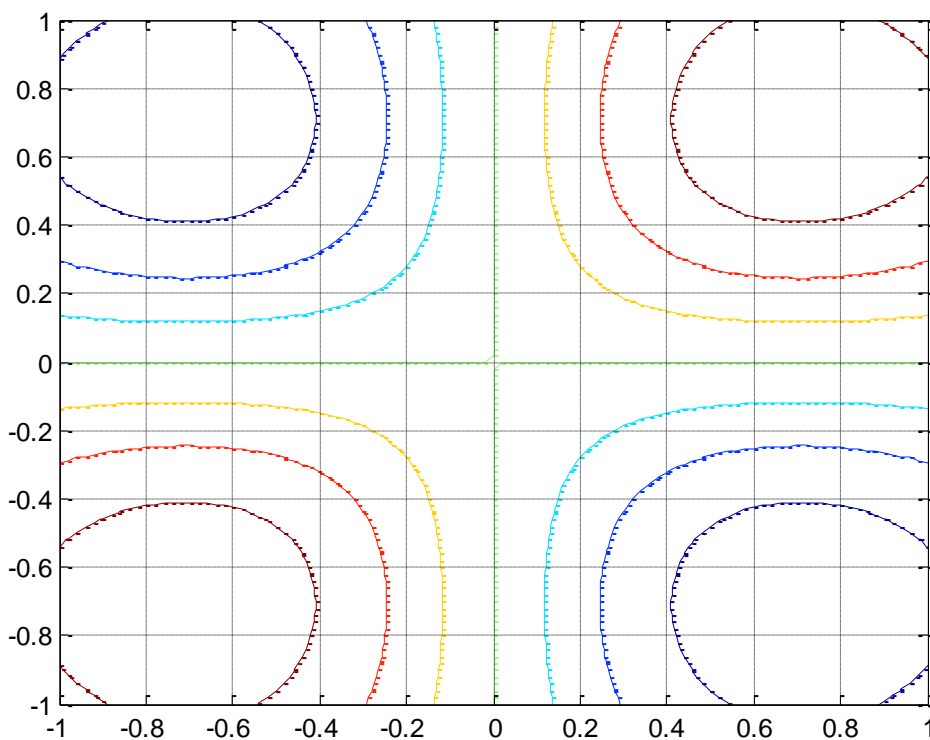


Fig. 1 – Graph of the function $f(x,y)$.

Example 2

First part: Basic elements

Visualization of level lines using the contour command. The number of level lines is determined automatically from the extreme values taken by the function on the domain considered. To impose the number n of level lines to be displayed, simply call the contour function with the value n as the fourth parameter, `contour(X,Y,Z,n)`. There are two ways to display the contour line values in the figure. If we want to display the values for all level lines, we use the `clabel` command in the following way:

```
>> [C,h] = contour(X,Y,Z,n)
>> clabel(C,h)
```

If we want to display only the values of a few level lines, we use the `clabel` command in the following way :

```
>> [C,h] = contour(X,Y,Z,n)
>> clabel(C,h, 'manual')
```

You can then use the mouse to select the level lines for which you want to display the value.

So to draw 40 lines of level of function $z = (x^2+2)^2 + (x^2+y)^2$ on the domain $[-1, 1] \times [-1, 1]$.

```
[X,Y] = meshgrid(-1:0.02:1, -1:0.02:1);
```

```
Z = (X.^2+2).^2 + (X.^2+Y).^2;
```

```
[C,h] = contour(X,Y,Z,40);
```

```
clabel(C,h, 'manual')
```

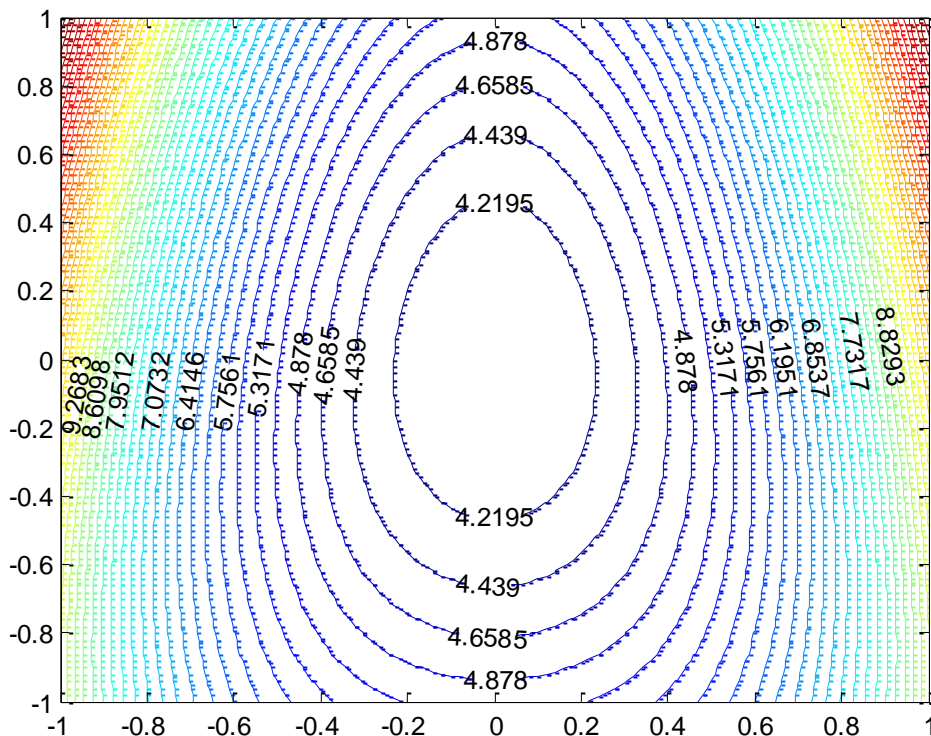


Fig. 2 – Graph of the function $z(x,y)$.

Example 3

First part: Basic elements

Visualization of level lines using the contour command. It is possible to modify the color palette using the **colormap** command. Typing `help graph3d` in the MATLAB control window will get you all available color palettes.

The **contourf** command is used in the same way as the contour command. It is used to display, in addition to the level lines, a continuous gradient of colors which varies according to the values taken by the function.

```
[X,Y] = meshgrid(-1:0.02:1, -1:0.02:1);  
Z = (X.^2+2).^2 + (X.^2+Y).^2;  
  
contourf(X,Y,Z,40);  
  
colormap(cool);
```

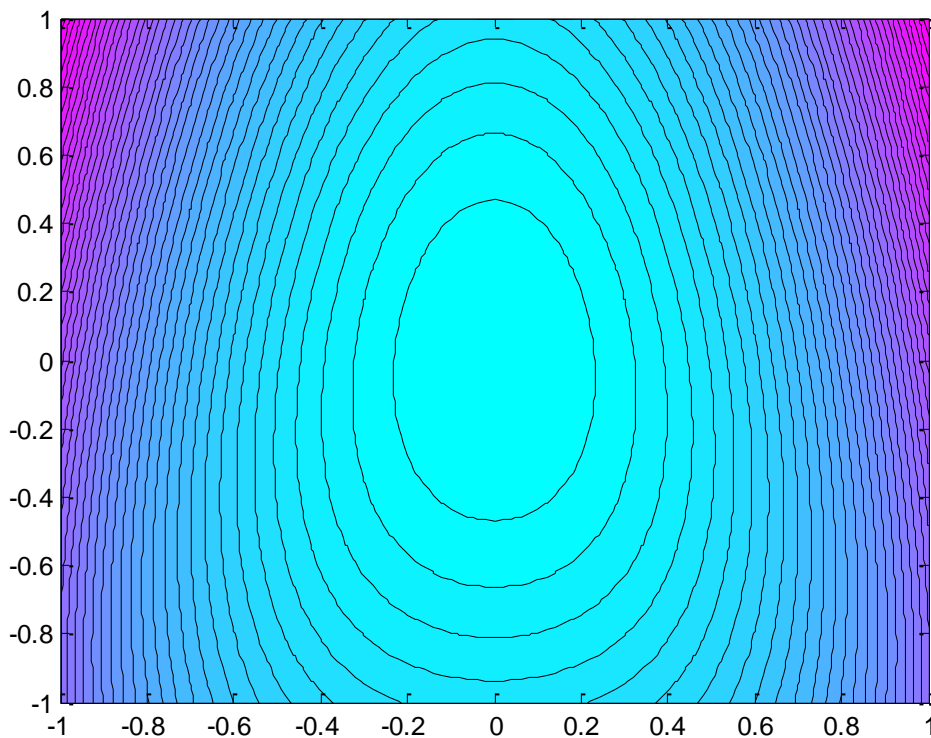


Fig. 3 – Graph of the function $z(x,y)$.

1.5.2 Represent an equation surface $z=g(x,y)$

The mesh command allows you to draw a surface with the equation $z=g(x,y)$. The function g can be defined directly by a MATLAB expression or be defined as a user function. To plot the surface with equation $z=g(x,y)$ for $x_{\min} < x < x_{\max}$ and $y_{\min} < y < y_{\max}$ we proceed as follows:

First part: Basic elements

- Creation of a mesh, mesh of length H, in the field $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ thanks to the **Meshgrid** command, $[X,Y] = \text{meshgrid}(x_{\min}:h:x_{\max}, y_{\min}:h:y_{\max})$.
- Evaluation of the function to the nodes of this network, either by calling for the user function defining the function, $Z = g(X,Y)$ either directly by defining the function by an expression MATLAB.
- Surface display thanks to the mesh command, $\text{mesh}(X,Y,Z)$.

Example

$z = 2xe^{(-x^2+y^2)}$ on the domain $[-1, 1] \times [-1, 1]$ with a mesh of mesh length $h=0.02$, we execute:

```
clc; clear all
```

```
[X,Y] = meshgrid(-1:0.02:1, -1:0.02:1);
```

```
Z = 2*X*Y.*exp(-X.^2+Y.^2-1);
```

```
mesh(X,Y,Z)
```

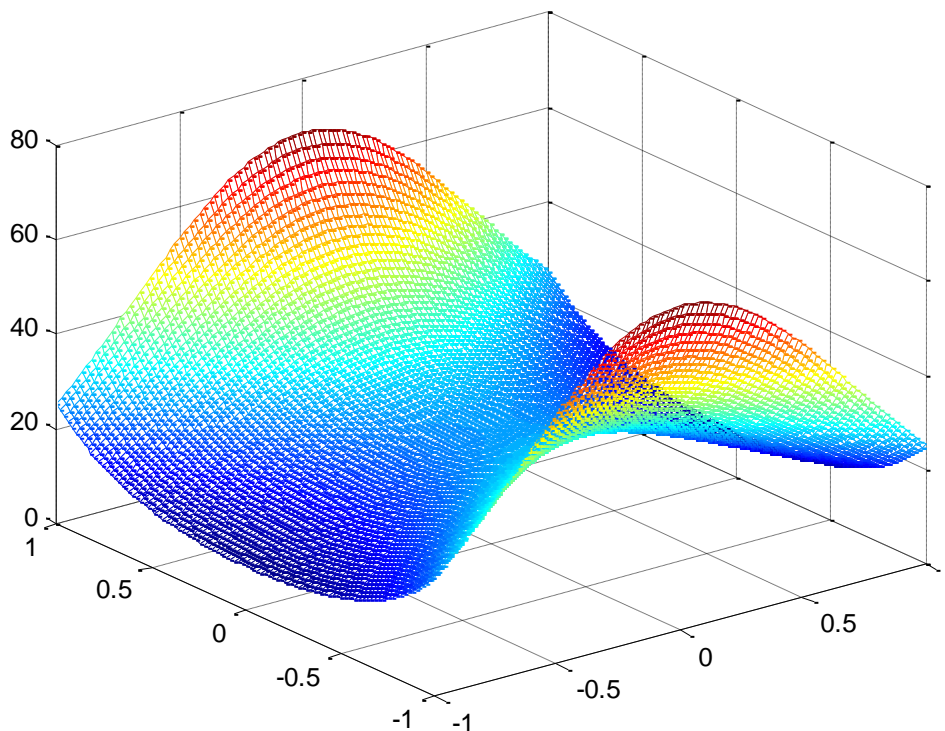


Fig. 4 – Graph of the function $z(x,y)$.

1.5.3 Represent a parameterized surface

The **surf** control allows you to draw a parameterized surface of equations

First part: Basic elements

$$\begin{cases} x = f_1(u, v) \\ y = f_2(u, v) \\ z = f_3(u, v) \end{cases}$$

The f= function (f1, f2, f3) can be defined directly by a MATLAB expression or be defined as a user function. To plot the parameterized area of equation (E) for $u_{\min} < u < u_{\max}$ and $v_{\min} < v < v_{\max}$ we proceed as follows:

- creating a mesh, h-length mesh, domain $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ using the command **meshgrid**, $[U, V] = \text{meshgrid}(u_{\min}:h:u_{\max}, v_{\min}:h:v_{\max})$. Evaluation of the function to the nodes of this network, either by calling for the user function defining the function, $[x, y, z] = f(u, v)$ either directly by defining the function by a Matlab expression.
- Surface display thanks to the surf control, **surf(X,Y,Z)**.

Example 1

Thus to draw the configured surface of equations:
$$\begin{cases} x = v \cos(u) \\ y = v \sin(u) \\ z = 2v \end{cases}$$

on the domain $[0, 2\pi] \times [0, 2]$ with a mesh mesh of length $h=0.05$, we execute:

```
clc; clear all
```

```
[U,V] = meshgrid(0:0.05:2*pi, 0:0.05:2);
```

```
X = 2*V.*cos(2*U);
```

```
Y = V.*sin(2*U);
```

```
Z = 2*U.^2+V.^2;
```

```
surf(X,Y,Z)
```

Si la fonction est définie comme une fonction utilisateur dans le fichier G1.m,

```
function [x1, x2, x3] = G1(u,v)
```

```
x1 = 2*v.*cos(2*u);
```

```
x2 = v.*sin(2*u);
```

```
x3 = 2*u;
```

On exécute:

```
>> [U,V] = meshgrid(0:0.05:2*pi, 0:0.05:2);
```

```
>> [X,Y,Z] = G1(U,V);
```

```
>> surf(X,Y,Z)
```

First part: Basic elements

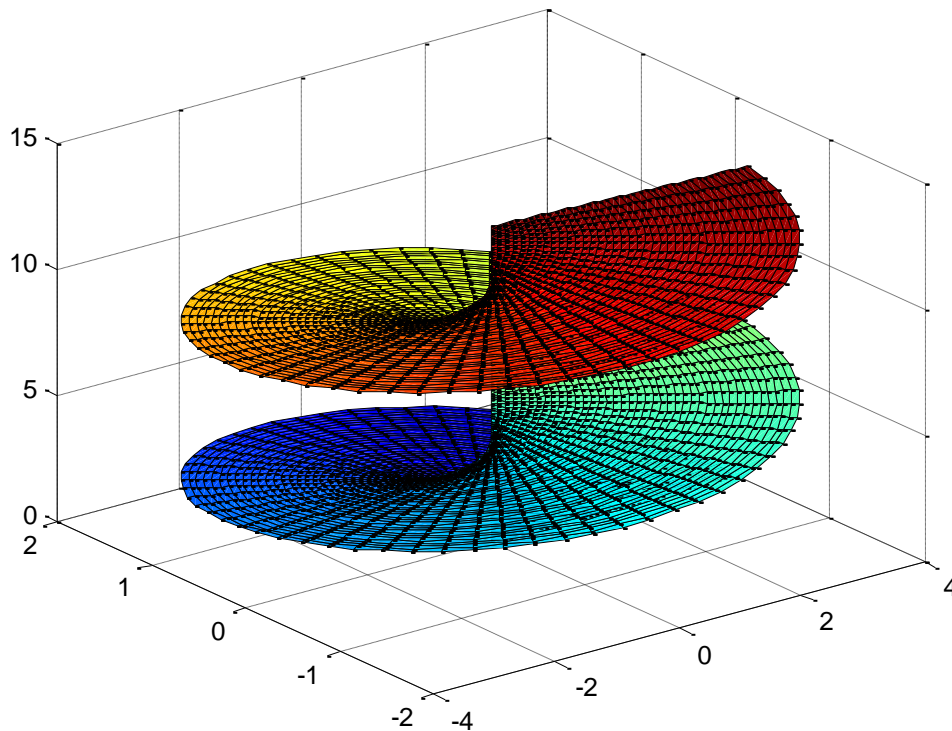


Fig. 5 – surface d'équations.

Example 2

Either parametric equation:
$$\begin{cases} x = \cos(t) \\ y = 2t\sin(t) \\ z = 5t \end{cases}$$

```
clc; clear all
t = 0 : pi/200 : 10*pi;
x = t.*cos(t);
y = 2*t.*sin(t);
z = 5*t+1;
plot3( cos(t) , 2*t.*sin(t) , 5*t),grid on
```

First part: Basic elements

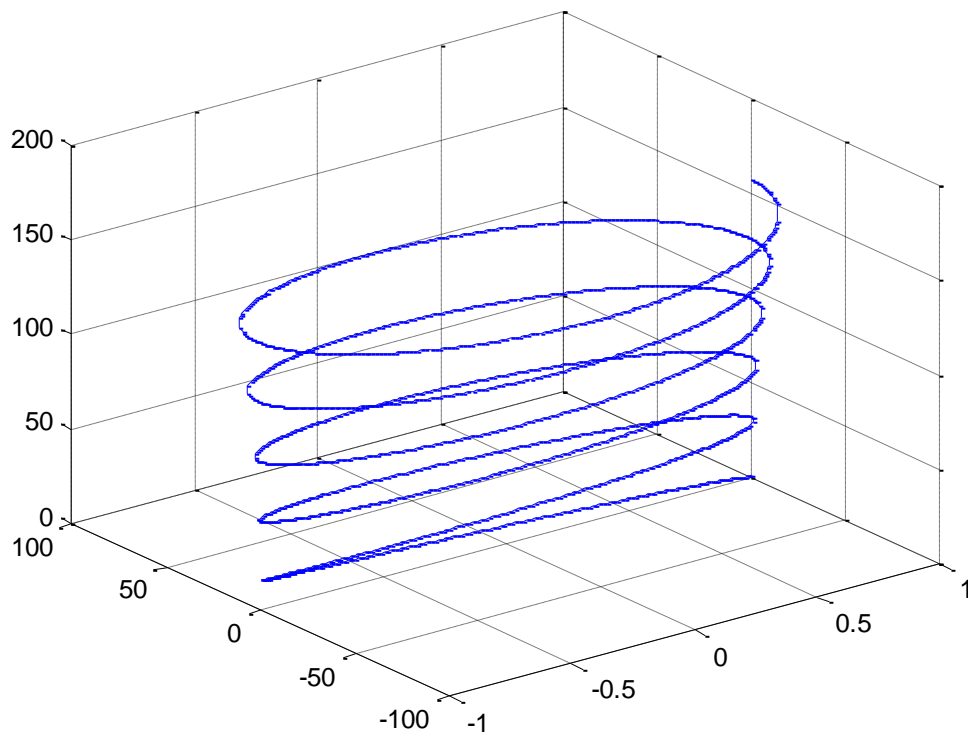


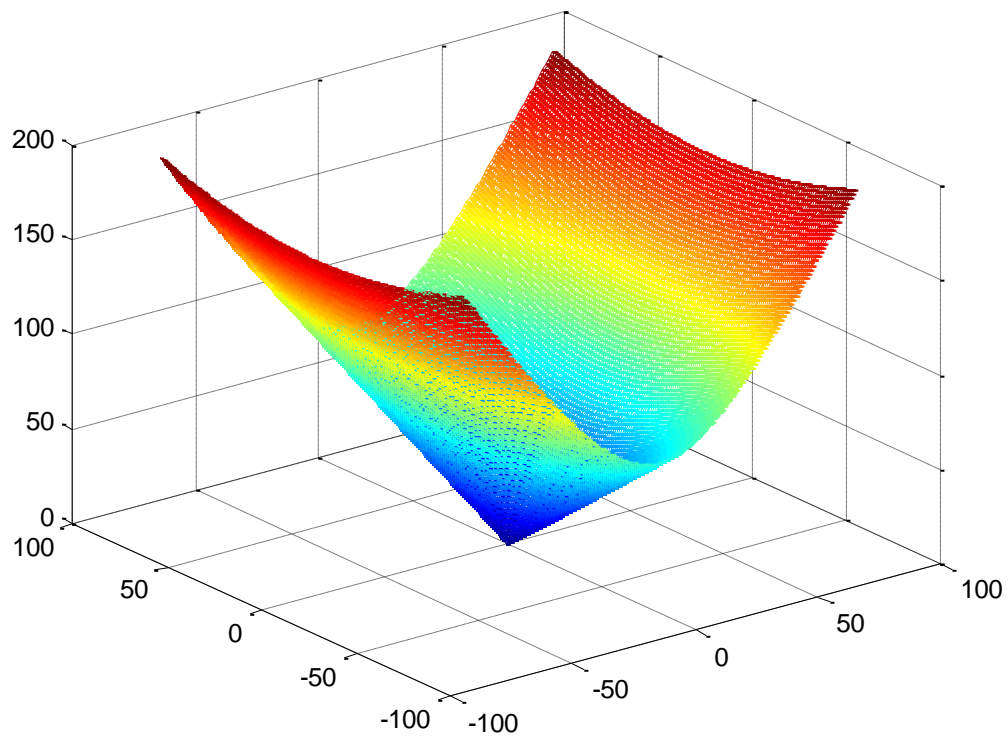
Fig. 6 – surface d'équations.

Example 3

$$z = f(x, y) = \sqrt{5x^2 + y^2}$$

```
clc; clear all
x = -80 : 80
y = -80 : 80
[X , Y] = meshgrid(x , y);
Z = sqrt (5.*X.^2 + Y.^2);
mesh (X , Y , Z)
grid on
```

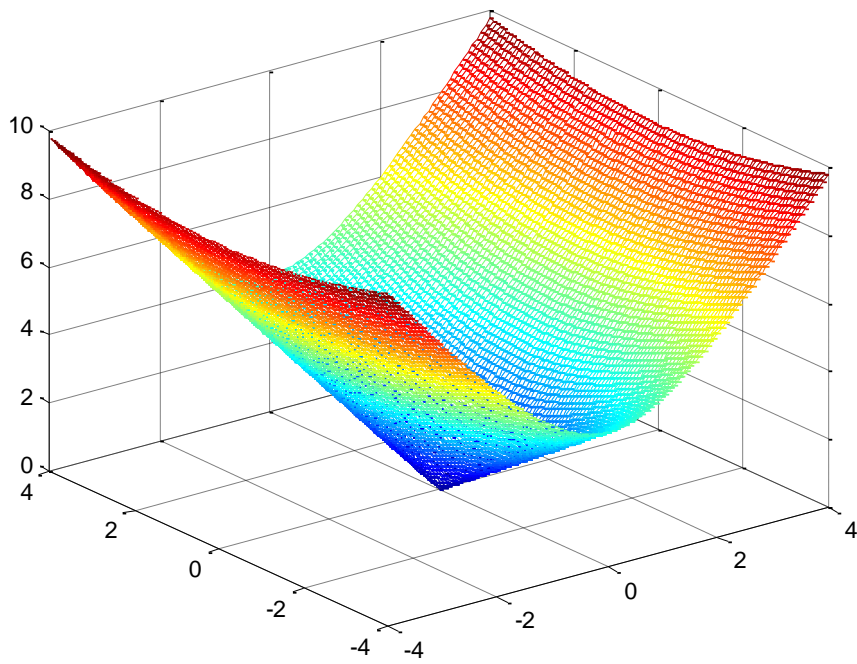
First part: Basic elements



For better resolution:

```
x = -4 :0.08: 4  
y = -4 :0.08: 4  
[X , Y] = meshgrid(x , y)  
Z = sqrt (5.*X.^2 + Y.^2)  
mesh (X , Y , Z)  
grid on
```

First part: Basic elements



1.6 Calculation on polynomials

1.6.1 Operations on polynomials in MATLAB

1.6.2 Handling Polynomial Functions in MATLAB

1.6.3 Evaluation of a Polynomial

1.6.4 Determining the coefficients of a polynomial from its roots

1.6.5 Graphic Representation

First part: Basic elements

1.6 Calculation on polynomials

1.6.1 Operations on polynomials in MATLAB

- **Functions**

conv	polynomial product
residue	decomposition into simple elements
roots	Find the roots of a polynomial
poly	Find the polynomial from its roots
polyval	Evaluates the polynomial

In Matlab, polynomials are represented in the form of line vectors whose components are given in order of decreasing powers. A degree polynomial is represented by a size vector (N+1).

Example

the polynomial $f(x) = 8x^5 + 2x^3 - 3x^2 + 4x - 2$ is represented by :

`f=[8 0 2 -3 4 -2]`

`f =8 0 2 -3 4 -2`

Other functions in MATLAB such as : '*conv*', '*deconv*', '*roots*', etc. can be used in addition to vector-specific operations.

- **Multiplication of polynomials**

The 'conv' function gives the product of convolution of two polynomials.

Example 1

$$\begin{cases} f(x) = -2x^3 + 5x^2 - 3x + 1 \\ g(x) = x^4 - 3x^2 - 5x - 8 \end{cases}$$

The Convolution product $g(x) = f(x) \cdot g(x)$: is given by:

`f=[-2 5 -3 1];`

`g=[1 -3 -5 -8];`

`h=conv(f,g)`

`h =`

`-2 11 -8 1 -28 19 -8`

Example 2

$(3x - 2)(2x - 1) = ?$

`p1=[3 -2];`

`p2=[2 -1];`

`conv(p1, p2)`

First part: Basic elements

ans =

6 -7 2

In other words : $(3x - 2)(2x - 1) = 6x^2 - 7x + 2$.

• Polynomial Division

The 'deconv' function gives the convolution ratio of two polynomials. The following example shows the use of this function. Let the same previous functions f(x) and g(x):

La division de f(x) par g(x) :

```
clc
```

```
f=[ 3 -2 3 5 ];
```

```
g=[ 2 -1 2 ];
```

```
h=deconv(f,g)
```

h =

1.5000 -0.2500

And the polynomial obtained is: $h(x) = 1.5x - 0.25$

1.6.2 Handling Polynomial Functions in MATLAB

Let be the following polynomial: $P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x^1 + a_0$ where n is the degree of the polynomial and $a_i (i = 0, 1, 2, \dots, n)$ are the coefficients of the polynomial.

This polynomial can be written as: $P(x) = ((\dots ((a_n x^n + a_{n-1})x^{n-1} + a_{n-2})x \dots + a_1)x^1 + a_0)$.

After factorization, we have: $P(x) = a_n (x - r_1)(x - r_2)(x - r_3) \dots (x - r_n)$ where $r_0, r_1, r_2 \dots r_n$ are the roots of the polynomial.

Example 1

The polynomial $P(x) = 2x^3 + 10x^2 - 6x + 20$, that is represented in MATLAB by:

```
P=[1 5 -3 10];
```

To find these r_i , roots, one must perform the function 'roots'.

```
r=roots(P);
```

and the result given is:

```
clc; clear all
```

```
P=[2 10 -6 20];
```

```
r=roots(P)
```

r =

-5.8122

0.4061 + 1.2472i

0.4061 - 1.2472i

The three roots of this polynomial are given as a column vector. When the racines r_i are known, the coefficients can be recalculated by the order 'poly'.

```
>>poly(r)
```

First part: Basic elements

ans =

1.0000 5.0000 -3.0000 10.0000

Example 2

```
clc; clear all
```

```
P = [ 2 -4 -8 4 -14 16 ]
```

```
r=roots(P)
```

```
poly(r)
```

```
format long e
```

```
roots(P)
```

```
P =
```

```
2      -4      -8      4      -14      16
```

```
r =
```

```
3.1942
```

```
-1.9745
```

```
-0.0496 + 1.2000i
```

```
-0.0496 - 1.2000i
```

```
0.8794
```

```
ans =
```

```
1.0000      -2.0000      -4.0000      2.0000      -7.0000      8.0000>>
```

```
format long e
```

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

```
ans =
```

```
3.194190220624354e+000
```

```
-1.974500417245445e+000
```

```
-4.955755139294071e-002 +1.199959128943957e+000i
```

```
-4.955755139294071e-002 -1.199959128943957e+000i
```

```
8.794252994069751e-001
```

Example 3

Complex coefficient polynomial: $(1-i)x^2 + (2-5i)x + 8 = 0$

```
clc; clear all
```

```
P = [ 1-i 2-5i 8]
```

```
r=roots(P)
```

First part: Basic elements

```
format short e
```

```
P =
```

```
1.0000e+000 -1.0000e+000i 2.0000e+000 -5.0000e+000i  
8.0000e+000
```

```
r =
```

```
-3.3768e+000 +2.7863e+000i  
-1.2324e-001 -1.2863e+000i
```

Example 4

```
clc; clear all
```

```
A=[4 6;1 3]
```

```
p=poly(A)
```

```
A =
```

```
4 6
```

```
1 3
```

```
p =
```

```
1 -7 6
```

Thus, the characteristic polynomial of the matrix A is: $P(x) = x^2 - 7x + 6$. The roots of this polynomial are the eigenvalues of the matrix A. these roots can be obtained by the function '*eig*' :

```
>> Val_prop=eig(A)
```

```
Val_prop =
```

```
6
```

```
1
```

1.6.3 Evaluation of a Polynomial

To evaluate the polynomial $p(x)$ at a given point, we must use the 'polyval' function. We evaluate this polynomial for $x=1$, for example:

```
clc; clear all
```

```
P = [ 2 -7 10 ]
```

```
polyval(P,1)
```

```
P=
```

```
2 -7 10
```

```
ans =
```

```
5
```

First part: Basic elements

1.6.4 Determining the coefficients of a polynomial from its roots

Example

```
clc; clear all
a = [ 3 -5 ]
poly(a)
a =
     3     -5

ans =
     1     2    -15      (that is to say: x2 +2x -15).
clc; clear all
a = [ 1+i 2-i 5]
poly(a)

a =
 1.0000e+000 +1.0000e+000i 2.0000e+000 -1.0000e+000i
5.0000e+000
ans =
 1.0000e+000 -8.0000e+000 1.8000e+001
+1.0000e+000i -1.5000e+001 -5.0000e+000i
Verification
ans =
 1.0000e+000 -8.0000e+000 1.8000e+001
+1.0000e+000i -1.5000e+001 -5.0000e+000i

p =

 1.0000e+000 -8.0000e+000 1.8000e+001
+1.0000e+000i -1.5000e+001 -5.0000e+000i
ans =
 5.0000e+000 +8.8818e-016i
 2.0000e+000 -1.0000e+000i
 1.0000e+000 +1.0000e+000i
```

1.6.5 Graphic Representation

Example

$$y = f(x) = x^2 - 3x + 1$$

- **Using the function plot**

```
clc; clear all
p = [ 6 -4 1 ]
x = 0 : 0.01 : 2;
y = polyval( p , x)
```

First part: Basic elements

```
plot (x , y)  
grid on
```

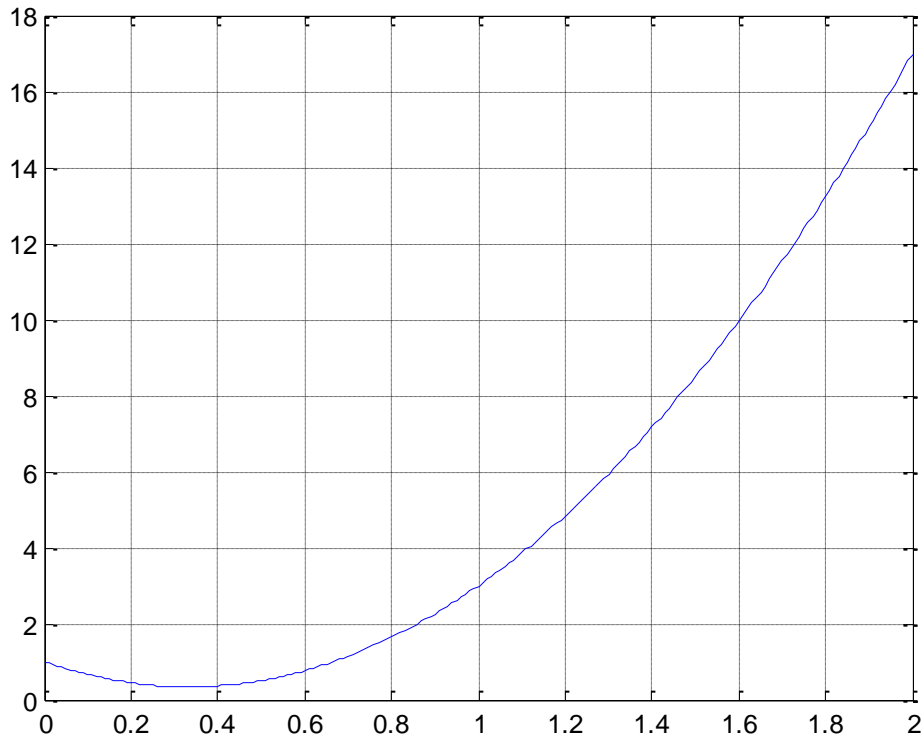
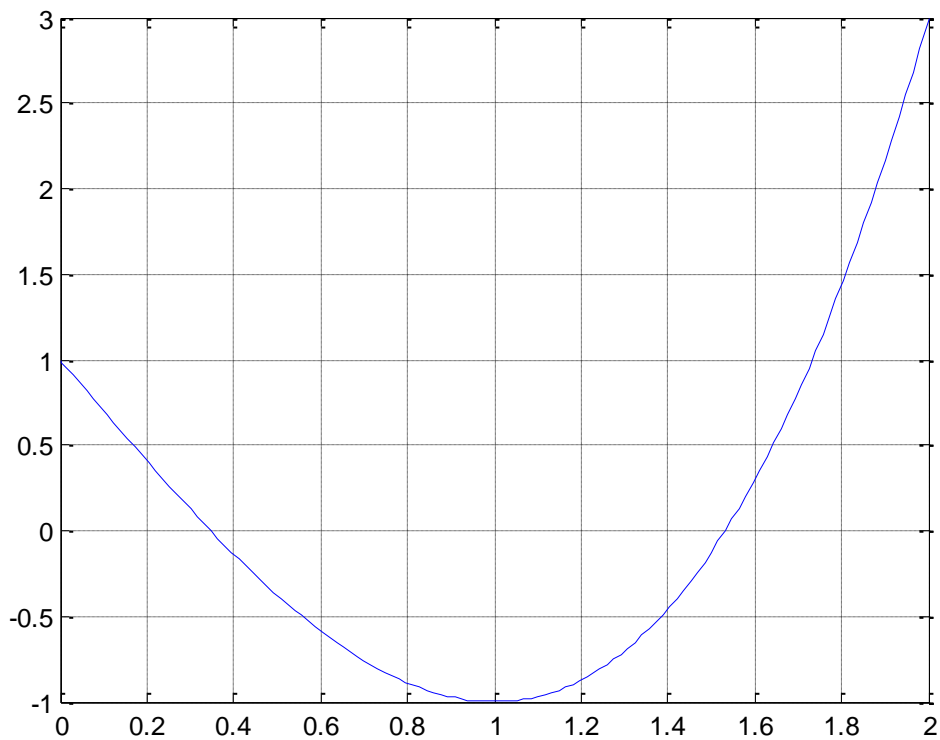


Fig. 1 – Graph of the polynomial $p(x)$.

- **Using the function fplot**

```
clc; clear all  
fplot ( 'x^3 - 3*x + 1', [ 0 2 ] )  
grid on
```

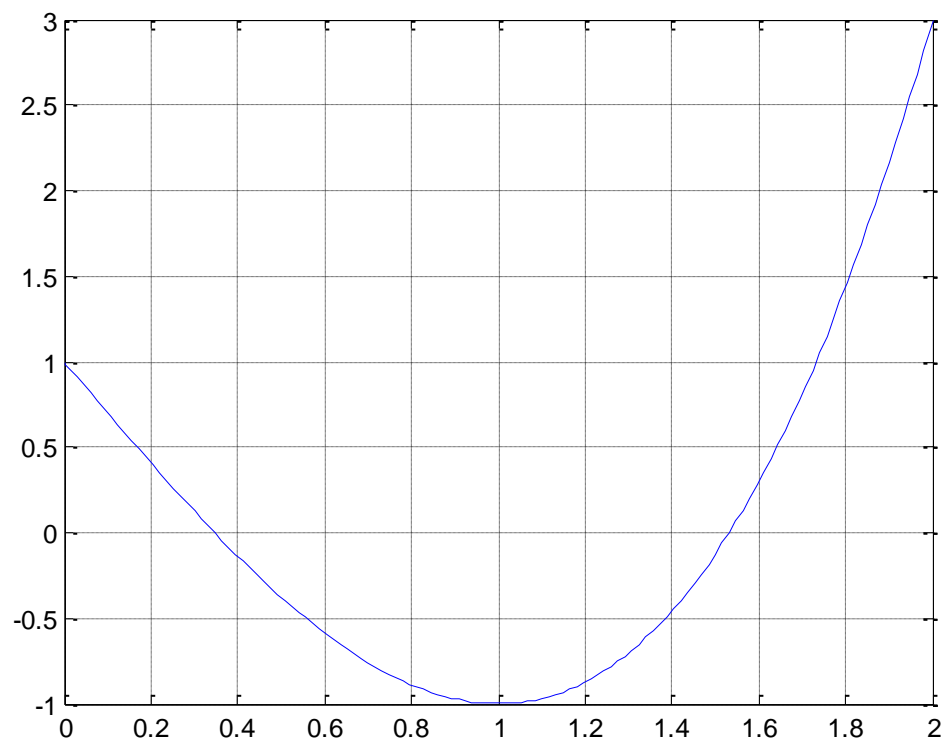
First part: Basic elements



You must create the .m file of the function:

```
function y=h33(x)
P=[1 0 -3 1];
y=polyval(P,x);
>> h3(5)
ans =
    111
fplot('h33',[0 2]),grid on
```

First part: Basic elements



Part Two: The Matlab Programming Language and Using Scripts

2. Part Two: The Matlab Programming Language and Using Scripts

2.1 Script files

2.2 Functions

2.2.1 Creating a function in an M-Files

2.3 Structures of control of flux

2.3.1 Conditional Statements if, elseif, else

2.3.2 Conditional Statements switch ,case

2.3.3 Conditional loop while

2.3.4 Iterative loop for

2.4 Examples of applications

2.4.1 Digital integration

2.5.2 Laplace transformation

2.1 Script files

A script file is used to group series of Matlab commands. This avoids having to enter several times long sequences of instructions. When launching, the instructions it contains sequentially executed as if they were launched from the command prompt. A script stores its variables in the workspace, which is shared by all the scripts. Thus, all the variables created in the scripts are visible from the Window and Vice Versa command. When Matlab detects an error, the program stops and an error message is displayed on the screen (with the number of the line where the error is detected).

Let us edit our script tp1f.m.

Example 1

```
x = cos(10);
y = sin(10);
results (1) = x + y ;
results (2) = x * y ;
results (3) = sqrt(x^2+y^2);
results
```

Our script can then be executed, either by typing its name (without the extension) at the prompt of order, either by clicking on the editor's "RUN" button (icon with a green triangle).

```
>> tp1f
results=

    -1.3831    0.4565    1.0000
```

After running tp1f, the x and y variables are still available in Matlab memory. To be convinced, let's type

```
>> who
```

Your variables are:

```
ans    results x      y
```

```
>> whos
```

Name	Size	Bytes	Class	Attributes
ans	1x3	24	double	
results	1x3	24	double	
x	1x1	8	double	
y	1x1	8	double	

Example 2

Let's evoke the Matlab editor with the instruction :

```
>> edit tp2f
```

This will open the editor and create the file tp2f.m

Enter the following instructions into the tp2f.m file :

```
clc
```

```
x = 1:2*pi/N:4*pi;
```

```
y = sin(w*2*x.^2);
```

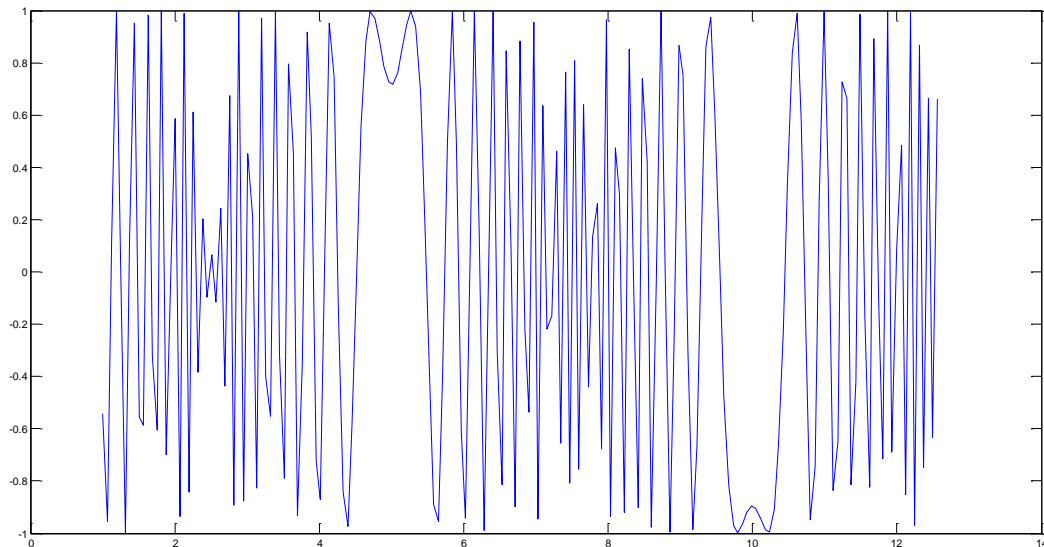
```
plot(x,y)
```

Next, the command sequence

```
>> N=100;w=5;
```

```
>> tp2f
```

produces the figure 1.



2.2 Functions

There is a conceptual difference between functions in computer science and mathematics:

1. In computer science, a function is a routine (a sub -program) which accepts arguments (parameters) and which returns a result
2. In mathematics, a function f is a relation that assigns to each value x at most a single value $f(x)$

2.2.1. Creating a function in an M-Files

MATLAB contains a big number of prédéfinies functions as `sin`, `cos`, `sqrt`, `sum`, ... etc. and it is possible to create our own functions by writing their source codes in files **M-Files** (carrying the same name of function) by respecting the following syntax:

```
function [y1,...,yN] = myfun(x1,...,xM)
```

`function [y1,...,yN] = myfun(x1,...,xM)` declares a function named `myfun` that accepts inputs `x1,...,xM` and returns outputs `y1,...,yN`. This declaration statement must be the first executable line of the function. Valid function names begin with an alphabetic character, and can contain letters, numbers, or underscores. You can save your function:

In a function, file which contains only function definitions. The name of the file must match the name of the first function in the file. In a script file which contains commands and function definitions.

Functions must be at the end of the file. Script files cannot have the same name as a function in the file. Files can include multiple local functions or nested functions. For readability, use the `end` keyword to indicate the end of each function in a file. The `end` keyword is required when: Any function in the file contains a nested function. The function is a local function within a function file,

and any local function in the file uses the end keyword. The function is a local function within a script file.

Example 1

Write a function that calculates the square root of a number by Newton's method

```
function r = tpracine(number)
    r = number/2;
    precision = 6;
    for i = 1:precision
        r = (r + number ./ r) / 2;
    end[80]>> x = tpracine(10)
x =
    3.1623
>> x = tpracine(101)
x =
    10.0499
>> x = tpracine([15 52 166 3])
x =
    3.8730    7.2111   12.8841    1.7321
```

2.3 Structures of control of flux

The flow control structures are instructions to define and manipulate the order of execution of tasks in a program. They offer the possibility of carrying out different treatments depending on the statement of the program data, or making repetitive loops for a given process.

2.3.1 Conditional Statements if, elseif, else

This structure makes it possible to execute an instructions block as a function of the logical value of an expression. Its syntax is:

```
if    expression

    instructions ...

end
```

All instructions `instructions` is executed only if `expression` is true. Several exclusive tests can be combined.

```
if expression1

instructions1 ...

elseif expression2
```

```
instructions2 ...  
  
else  
  
instructions3 ...  
  
end
```

Several `elseif` can be concatenated. Their block is carried out if corresponding expression is true and if all previous conditions were not satisfied. The linked block `instruction3` in `else` is as for him carried out if none of the previous conditions was accomplished.

Example 1

We initialize a matrix `A` as a function of the value of a Nomex variable (example number) as follows:

```
x=input('Enter a value');  
y=3;  
if x<5  
y=0;  
elseif x>10  
y=2;  
end
```

Example 2

```
age = input('Enter your age : ');  
  
    if (age < 3)  
  
        disp('You are a baby')  
  
        elseif (age < 14)  
  
            disp('you are a child')  
  
            elseif (age < 20)  
  
                disp ('You are a teenager')  
  
                elseif (age < 65)  
  
                    disp ('you are an adult')  
  
                    else  
  
                        disp ('You are an oldster')  
  
                    endEnter your age : 2
```

You are a baby

Enter your age : 10

you are a child

Enter your age : 19

You are a teenager

Enter your age : 59.5

you are an adult

Enter your age : 80

You are an oldster

2.3.2 Conditional Statements switch, case

In this structure, a numeric expression is successively compared to different values. As soon as there is identity, the corresponding block of instructions is executed. Its syntax is:

```
switch expression
case valeur1,
    instructions1 ...
case valeur2,
    instructions2 ...
case valeur3,
    instructions3 ...
otherwise
    instructions ...
end
```

The tested expression, an expression, must be a scalar or character string. Once an instructioni block is executed, the execution flow exits the structure and resumes after the end. If no check box checks the tie, the block that follows otherwise is executed.

Example 1

Part Two: The Matlab Programming Language and Using Scripts

This example does a very simple job. The core idea is to pass through a switch statement and print message based on some condition. We create a basic logic of matching the number and providing an output based on the number.

```
N = input('Enter a number of your choice: ');
switch N
case -2
disp('negative one selected')
case 0
disp('zero selected')
case 2
disp('positive one selected')
otherwise
disp('Some other value')
end
```

Output: At the command prompt, enter the number -2. negative two Repeat the code and enter the number 5. Some other value.

Example 2

In this example of Switch Statement in Matlab, based on the grade obtained, we classify the distinction.

```
Enter_grade = 'A';
switch(enter_grade)
case 'A'
fprintf('Excellent performance!\n' );
case 'B'
fprintf('Well done performance\n' );
case 'C'
fprintf('Very Good performance\n' );
case 'D'
fprintf('You passed.. Congratulations\n' );
case 'F'
fprintf('Better luck next time\n' );
otherwise
fprintf('Invalid grade. Please enter correct value\n' );
end
```

2.3.3 Conditional loop while

This mechanism makes it possible to repeat a series of instructions as long as a condition is verified. Its syntax is:

```
While expression
```

```
Instructions
```

```
end
```

Example 1

```
a=1 ;

while (a~=0)

a = input ('Enter a number (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.

Example 2

```
eps = 1;

while (1+eps) > 1

eps = eps/2;

end
```

Example 3

Find the first number whose factorial uses more than 100 digits:

```
n = 1;
while prod(1:n) < 1e100
n = n + 1;
end
```

Example 4

2.3.4 Iterative loop for

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

```
for variable = expression

statement

...

statement

end
```

Example 1

```
k = input('Enter a number of your choice: ');  
  
a=zeros(k,k) % Preallocate matrix  
  
for m = 1:k  
  
    for n = 1:k  
  
        a(m,n) = 1/(m+n -1);  
  
    end  
  
end
```

Example 2

```
n = 4  
  
x = []  
  
for i = 1:n  
  
    x = [x, i^2]  
  
end
```

Example 3

```
clc  
  
m = 4  
  
n = 3  
  
for i = 1:m  
  
    for j = 1:n  
  
        H(i,j) = 1/(i+j-1) ;  
  
    end  
  
end
```


Part Two: The Matlab Programming Language and Using Scripts

H

```
>> m =
```

4

```
n =
```

3

```
H =
```

1.0000	0.5000	0.3333	0.2500
0.5000	0.3333	0.2500	0.2000
0.3333	0.2500	0.2000	0.1667
0.2500	0.2000	0.1667	0.1429
0.2000	0.1667	0.1429	0.1250
0.1667	0.1429	0.1250	0.1111

Example 4

Determine the type of plot to create based on the plot type value. If the plot type is "main" or "pie5", create a 3D pie chart. Use an array of cells to hold the two values.

```
clc
```

```
x = [15 62 24 8];
```

```
plottype = 'pie5';
```

```
switch plottype
```

```
case 'bar'
```

```
    bar(x)
```

```
    title('Bar Graph')
```

```
case {'pie','pie5'}
```

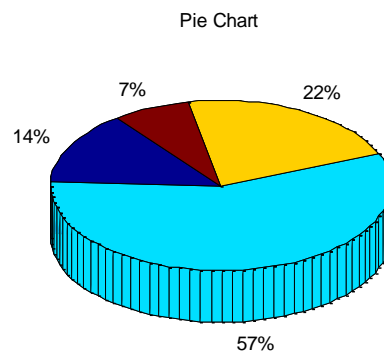
```
    pie3(x)
```

```
    title('Pie Chart')
```

```
otherwise
```

```
    warning('Unexpected plot type. No plot created.')
```

```
end
```



2.4 Examples of Applications

2.4.1 Digital integration

Numerical integration is used when an integral is impossible or difficult to solve analytically.

- **Trapeze method**

The approach is to subdivide the interval on which we want to integrate and approximate the integral of the function with a certain number of trapeziums. Each subdivision is of equal width h , where $h = (b-a)/n$ where N is the number of subdivisions.

$$\int_a^b f(x) \approx \frac{h}{2} [f(a) + 2f(a+h) + 2f(a+2h) + 2f(a+3h) + \dots + 2f(a+(n-1)h) + f(b)]$$

- **MATLAB Program**

Calculate the integral using the trapezium method $I = \int_1^6 f(x) = e^{-x^2}$.

```
% trapex.m test program for numerical integration using the
composite
% trapezoidal rule to solve the integral of exp(-x^2) between
% a and b
clear; help trapex;
format long; % configures MATLAB to report numbers with more
decimal places
a=input('input a (starting value)->');
b=input('input b (end value) ->');
n=input('input number of intervals (n) ->');
h=(b-a)/n;
fa=exp(-a^2); % f(a)
fb=exp(-b^2); %f(b)
ff=0;
for i=2:n
ff=ff+(2*exp(-(a+(h*(i-1)))^2)); % sum of 2f(a + i(h)) where i = 1
to n-1
end
result=(h/2)*(fa+fb+ff)
% result = f(a) + f(b) + sum of 2f(a + i(h)) where i = 1to n-1
>> input a (starting value)->1
input b (end value) ->6
input number of intervals (n) ->50

result =

0.140016129321009
```

- **Method of Simpson**

The Simpson integration method is based on the interval division $[x_i, x_{i+1}]$ in three, so obtain an equal step $h/3$.

The Simpson integration formula can be given by:

$$\int_a^b f(x) = \frac{h}{3} \left[f(x_1) + f(x_{n+1}) + 4 \sum_{j=2(j \text{ paire})}^n f(x_j) + 2 \sum_{j=3(j \text{ impaire})}^{n-1} f(x_j) \right]$$

- **MATLAB Program**

Calculate the integral using the simpson method $I = \int_1^6 f(x) = e^{-x^2}$.

```
% Simpsonex.m - A program for composite Simpsons 1/3 rule
% to numerically integrate a function between a and b with
% n subintervals.
% The example program integrates exp(-x^2)
format long % sets MATLAB to report more decimal places
clear; help simpsonex;
a=input('input a (starting value)->');
b=input('input b (end value) ->');
n=input('input number of intervals (n) ->');
h=(b-a)/n; % interval with
fa=exp(-(a^2)); % f(a)
fb=exp(-(b^2)); % f(b)
ff=0;
for i=2:2:n; % all 4*f(a+nh) terms to f(b) h=(1,3,5,7,...,n-1)
x = (a+(i-1)*h);
fx = exp(-x^2);
ff = ff + 4*fx;
end
for i=3:2:n; % all 2*f(a+nh) terms to f(b) h=(2,3,4,6,...,n-2)
x = (a+(i-1)*h);
fx = exp(-x^2);
ff = ff + 2*fx;
end
result=(h/3)*(fa+fb+ff) % integral result
% with approximation to area under curve

>>input a (starting value)->1

input b (end value) ->6
```

```
input number of intervals (n) ->50
```

```
result =
```

```
0.139401977198315
```

2.5.2 Laplace transformation

The Laplace transform of a time function $f(t)$ is given by the following integral:

$$\mathcal{L}\{f(t)\} = \int_0^{\infty} f(t) \cdot e^{-st} dt$$

The Laplace transform is also called transformed from $f(t)$ into $F(s)$. You can see that this process of transformation or integration converts $f(t)$, a function of the symbolic variable t , in another function $F(s)$, with another variable s .

The transformation of Laplace transforms the differential equations into algebraic equations. To calculate a laplace transform of a function $f(t)$, write:

```
laplace (f(t))
```

Example 1

In this example, we will calculate the Laplace transform of certain commonly used functions. Create a script file and type the following code:

```
clc
syms s t a b w
laplace(a)
laplace(t)
laplace(t^3)
laplace(exp(b*t))
laplace(sin(2*w*t))
laplace(cos(2*w*t))
ans =
1/s^2
ans =
1/s^2
ans =
6/s^4
ans =
1/(s-b)
ans =
2*w/(s^2+4*w^2)
ans =
s/(s^2+4*w^2)
```

- **The reverse Laplace transform**

Matlab allows us to calculate the reverse Laplace transform using the `ilaplace` command.

Example 1

```
clc
syms s t a b w
ilaplace(1/s^4)
ilaplace(3/(w-s))
ilaplace(1/(s^3+4))
ilaplace(exp(-b*t))
ilaplace(2*w/(s^2 - w^2))
ilaplace(2*s/(s^2 + w^2))
ans =
1/6*t^3
ans =
-3*exp(w*t)
ans =
1/12*exp(-2^(2/3)*t)*2^(2/3)+1/12*exp(1/2*2^(2/3)*t)*(-
cos(1/2*3^(1/2)*2^(2/3)*t)+3^(1/2)*sin(1/2*3^(1/2)*2^(2/3)*t))*2^(2
/3)
ans =
ilaplace(exp(-b*t),t,x)
ans =
2*sinh(w*t)
ans =
2*cos(w*t)
```

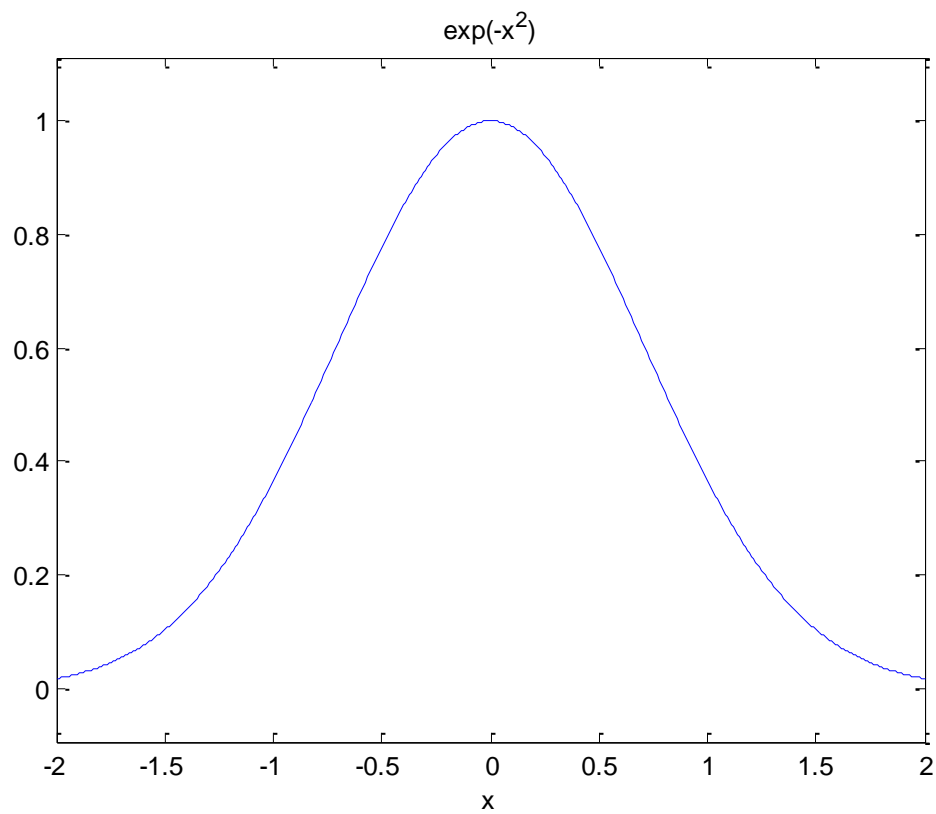
- **The Fourier Transforms**

Fourier's transforms commonly transform a mathematical function of time, $f(t)$, into a new function, sometimes noted as F , whose argument is the frequency with cycles/s (hertz) or radians units per second. The new function is then called Fourier transform and/or frequency spectrum of the function f .

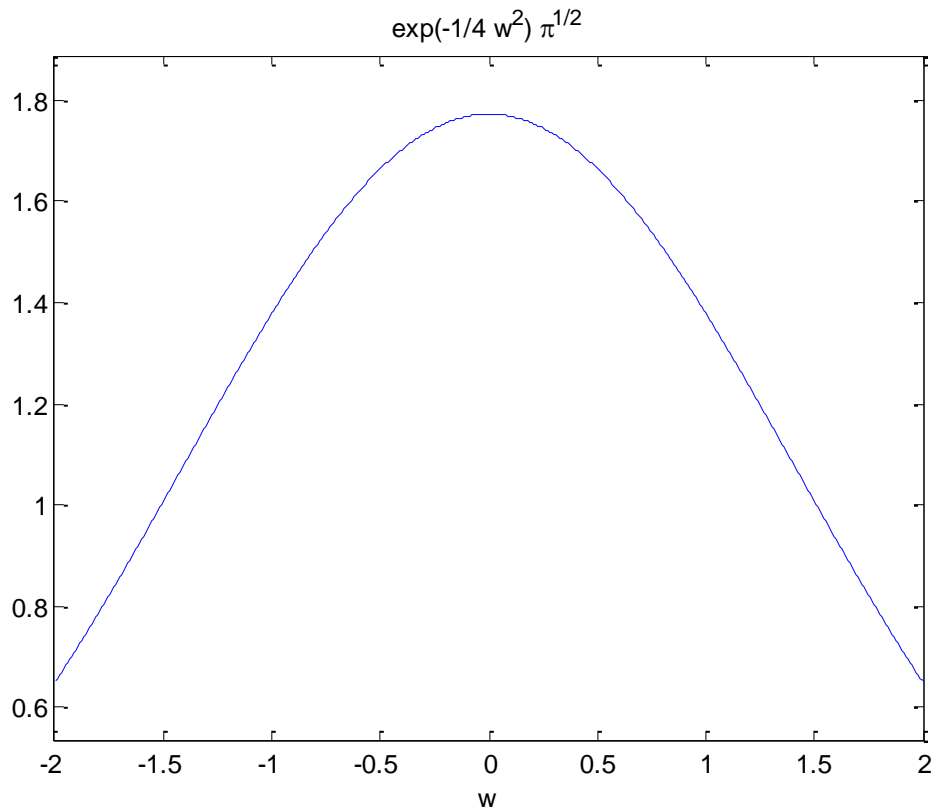
Example 1

Create a script file and type the following code:

```
clc
syms x
f = exp(-x^2);           %our function
ezplot(f,[-2,2])         % plot of our function
FT = fourier(f)           % Fourier transform
```



```
FT =  
exp(-1/4*w^2)*pi^(1/2)  
Tracé de la transformée de Fourier comme :  
ezplot(FT, [-2,2])
```



- **Reverse Fourier transformed**

Matlab provides the Fourier command to calculate the reverse Fourier transform of a function.

Example 1

Find the Inverse Fourier Transform of $\exp(-w^2/4)$

```
% MATLAB code specify the variable
```

```
% w and t as symbolic ones
```

```
syms w t
```

```
% define Frequency domain function X(w)
```

```
X=exp(-w^2/4);
```

```
% ifourier command to transform into
```

```
% time domain function x(t)
```

```
% using 1st syntax, where by default
```

```
% independent variable = w
```

```
% and transformation variable is x .
```

```
x1 = ifourier(X);
```

```
% using 2nd syntax, where transformation variable = t
```

```
x2 = ifourier(X,t);
```

```
% using 3rd syntax, where independent variable
```

Part Two: The Matlab Programming Language and Using Scripts

```
% = w (as there is no other
% variable in function) and transformation
% variable = t
x3 = ifourier(X,w,t);

% Display the output value
disp('1. Inverse Fourier Transform of  $\exp(-w^2/4)$  using ifourier(X)
:')
disp(x1);

disp('2. Inverse Fourier Transform of  $\exp(-w^2/4)$  using
ifourier(X,t) :')
disp(x2);

disp('3. Inverse Fourier Transform of  $\exp(-w^2/4)$  using
ifourier(X,w,t) :')
disp(x3);
x1=exp(-x^2)/pi^(1/2)
x2=exp(-t^2)/pi(1/2)
x3=ex(-t^2)/pi^1/2).
```


Reference

- [1] Matlab Guide, D. J. Higham, N. J. Higham, SIAM, 2000.
- [2] Introduction to Scientific Computing, A Matrix-Vector Approach Using MATLAB, C.F. Van Loan, MATLAB curriculum Series, 1997.
- [3] Apprendre et Maîtriser Matlab, versions 4 et 5 et SIMULINK_r, M. Mokhtari, A. Mesbah, Springer, 1997.
- [4] The MATHWORKS, Inc., Using MATLAB Graphics, The Mathworks, Inc., Natick, MA, 1996.

Some internet sites

- le site officiel de Matlab <https://www.mathworks.com/>
- https://www.tutorialspoint.com/matlab/matlab_transforms.htm
- <http://engineering.nyu.edu/mechatronics/vkapila/matlabtutor.html>
- <http://matlab.izmiran.ru/help/toolbox/webserver>