Algerian Democratic and Popular Republic
Ministry of Higher Education and Scientific Research

University IBN KHALDOUN of Tiaret
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

Hizia Khelifa
Emails: khhizia@gmail.com
hizia.khelifa@univ-tiaret.dz

Year : 2022-2023

Contents

List of Figures

Syllabus

(Course outline)

Field/Sector: LMD

Year : 2^{nd} year mathematics LMD.

Semester : 3

Teaching unit : Methodological

Matter : Programming tools 2

Credit:2

Coefficient:1

Weekly Hourly Volume:3H

•Lecture (1 hour 30 minutes)

•Practical work (1 hour 30 minutes)

Teacher responsible for the subject: Hizia Khelifa

Rank:M.C.B

Academic year:2023/2024

E-mail:khhizia@gmail.com

:hizia.khelifa@univ-tiaret.dz

Teaching objectives :

This course is aimed at undergraduate students (2MI). It is part of the modules of the methodological teaching unit. This course serves to introduce the student to the basic concepts of the MATLAB programming language to enable them to quickly acquire autonomy in using the software. It allows you to:

(1) Acquire the knowledge that will allow you to use Matlab software effectively.

(2) Get a clear idea of the potential contribution of this software to solving problems arising from mathematics.

The course is divided into seven chapters:

Chapter 1: Getting Started

Chapter 2: Numbers in Matlab

Chapter 3: Vectors and Matrices

Chapter 4: Programming elements

Chapter 5: Polynomials

Chapter 6: Graphics in Matlab

Chapter 7: Symbolic calculation

Learning assessment method: The calculation of the average will be a combination of the final exam grade and the practical grade. The weighting of the scores is as follows:

Control	Weighting %
Final exam	60
TP	40
Total	100

Prerequisites Students wishing to follow this training must have prerequisites on:

1. Basic computing (installing software and editing text files).

2. The basic notions of algorithms: variables, data structures, control structures, loops, etc.;

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-to-learn programming.
- ⧫ Seamless handling of integer, real, and complex values.
- ⧫ Support for a wide range of numbers with high precision.
- ⧫ A 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.

MATLAB is a powerful tool widely used in various industries and fields for numerical computations, data analysis, modeling, and simulation. Some real applications of MATLAB include:

1. Engineering: MATLAB is extensively used in engineering for tasks such as signal processing, image processing, control system design, and computational fluid dynamics.

2. Finance: In finance, MATLAB is employed for risk management, portfolio optimization, financial modeling, and trading strategy development.

3. Biomedical Research: MATLAB is vital in biomedical research for tasks like medical image analysis, bioinformatics, and physiological modeling.

4. Aerospace: MATLAB is utilized in the aerospace industry for aircraft design, trajectory analysis, and satellite communication systems.

5. Education: MATLAB is commonly used in educational institutions for teaching mathematical concepts, numerical methods, and scientific computing.

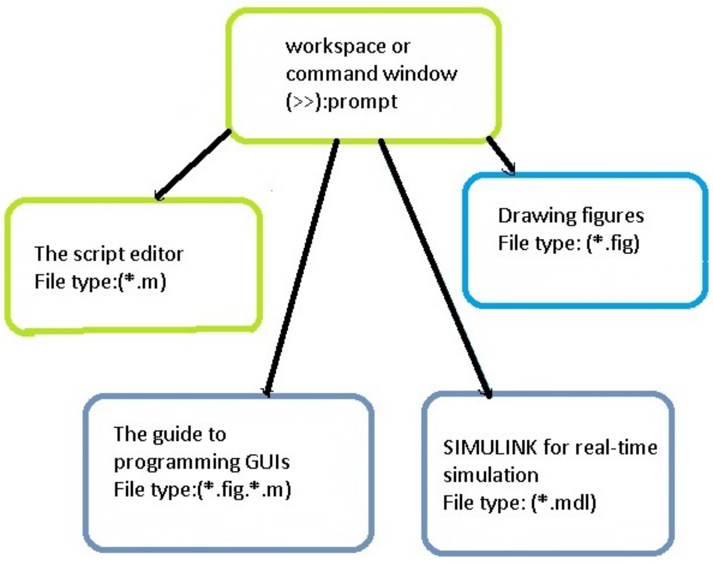
Chapter 1

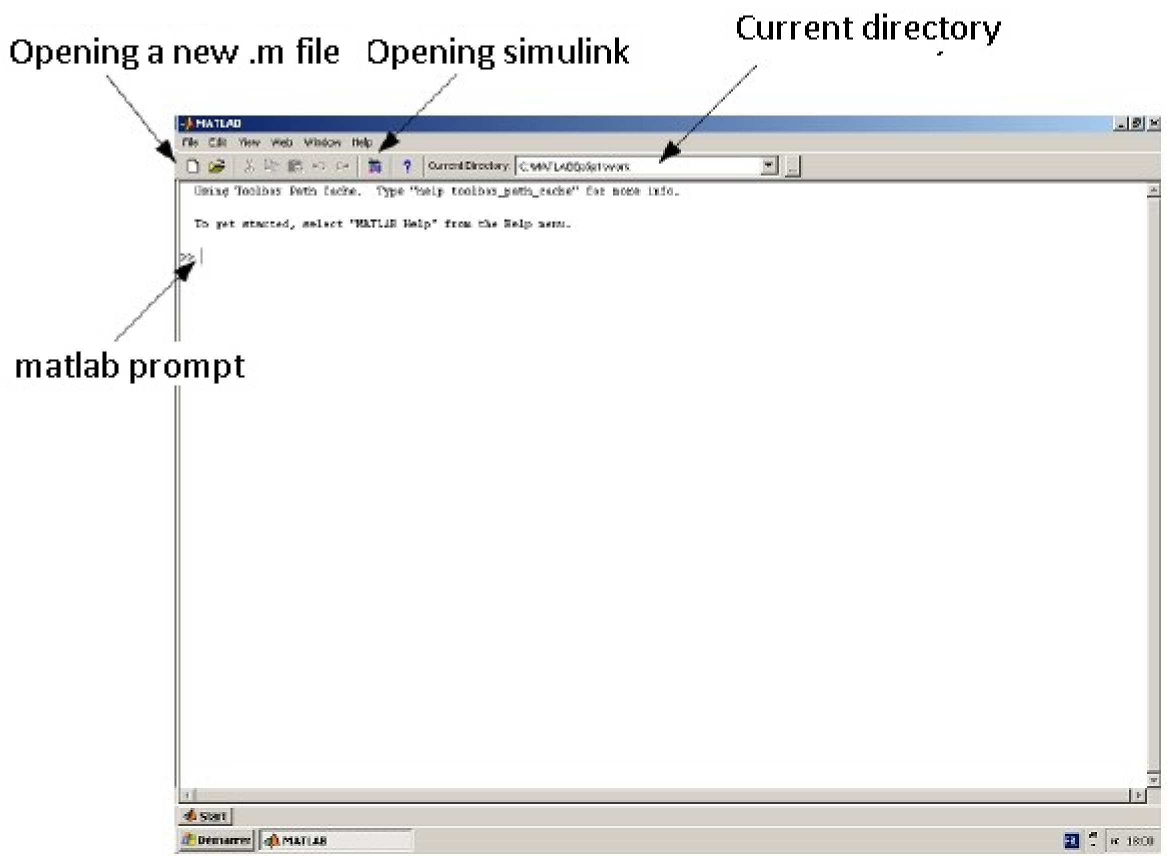
Getting Started

1.1 MATLAB environment

MATLABMATLAB stands for MATrix LABoratory, a platform designed for matrix manipulation. MATLAB is a scientific computing language based on the type of matrix variable (i.e. the type of basic data at the MATLAB level is the matrix), for numerical calculation and graphic visualization in 2D or 3D. It has a specific syntax with its specialized functions. It contains libraries specialized tools (toolbox) which meet specific needs: digital analysis, signal processing, image processing, automation, etc.

1.1.1 The MATLAB software structure





```
Using Toolbox Path Cache.  Type "help toolbox_path_cache" for more info.

To get started, select "MATLAB Help" from the Help menu.

>>
```

1.1.2 Mode of operation

There are two operating modes:

- **Interactive mode:** MATLAB executes instructions as they are given by the user. I.e. directly from the keyboard from the command window.
- **Executive mode:** MATLAB executes an M file (program in MATLAB language) line by line. i.e. in the form of sequences of expressions or scripts recorded in text files called m-files and executed from the command window.

1.2 Interface MATLAB

The MATLAB user interface may vary slightly depending on the MATLAB version and type of machine used. It typically consists of three main windows: The upper left window therefore displays Workspace. Below is the current directory and the Command History.

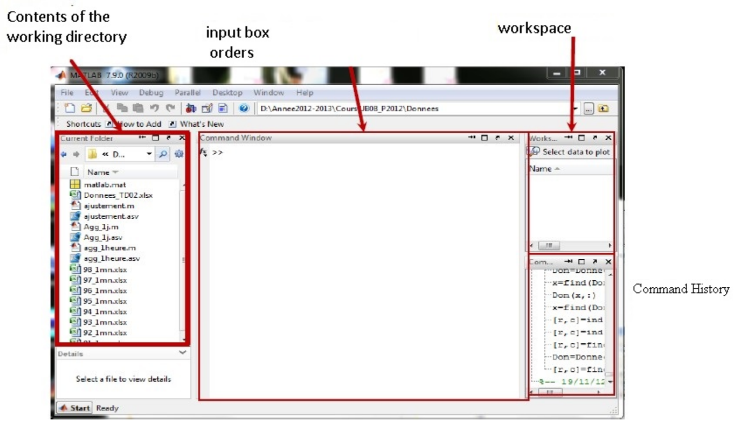


Figure 1.1: Interface MATLAB

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.

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.

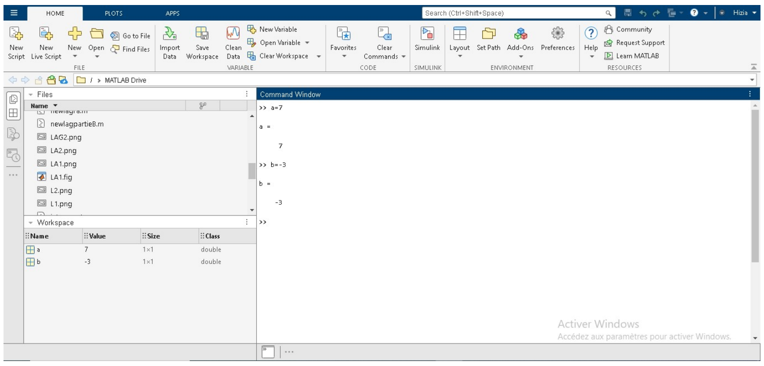
The status bar also contains buttons that allow you to easily add, modify or remove acknowledgment requests and comments.

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.

1.2.1 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



```
>> a=7

a =

     7

>> b=-3

b =

    -3

>>
```

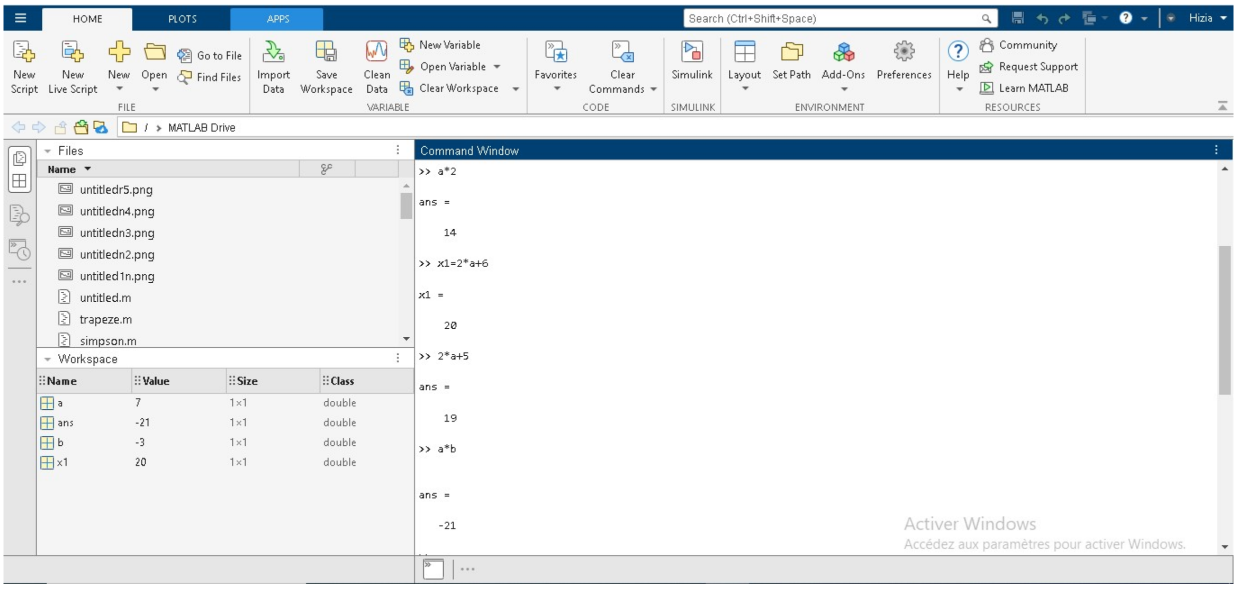
Name	Value	Size	Class
a	7	1×1	double
b	-3	1×1	double

Figure 1.2: Interface Command Window

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

ans =

    14

>> x1=2*a+6

x1 =

    20

>> 2*a+5

ans =

    19

>> a*b

ans =

   -21
```

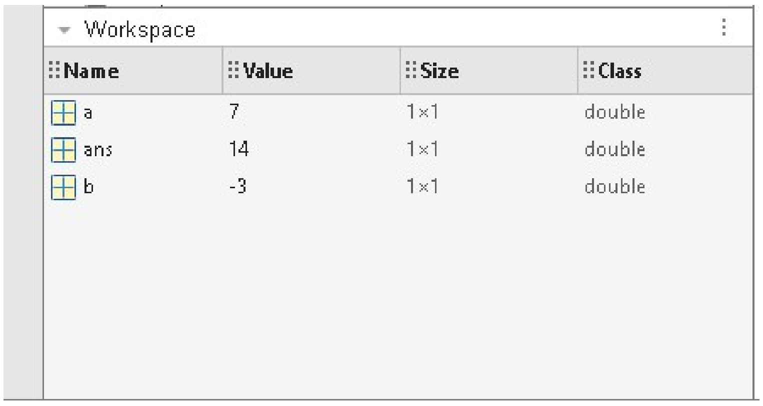
Name	Value	Size	Class
a	7	1×1	double
ans	-21	1×1	double
b	-3	1×1	double
x1	20	1×1	double

1.2.2 Boutton Start

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

1.2.3 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



Name	Value	Size	Class
a	7	1×1	double
ans	14	1×1	double
b	-3	1×1	double

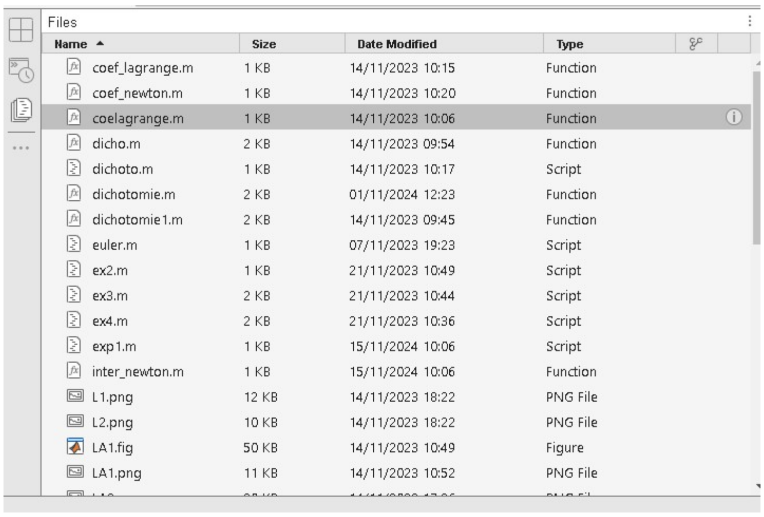
Figure 1.3: Interface Workspace

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.

1.2.4 Current Directory

The Current Directory is the common directory where files are recorded. 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.

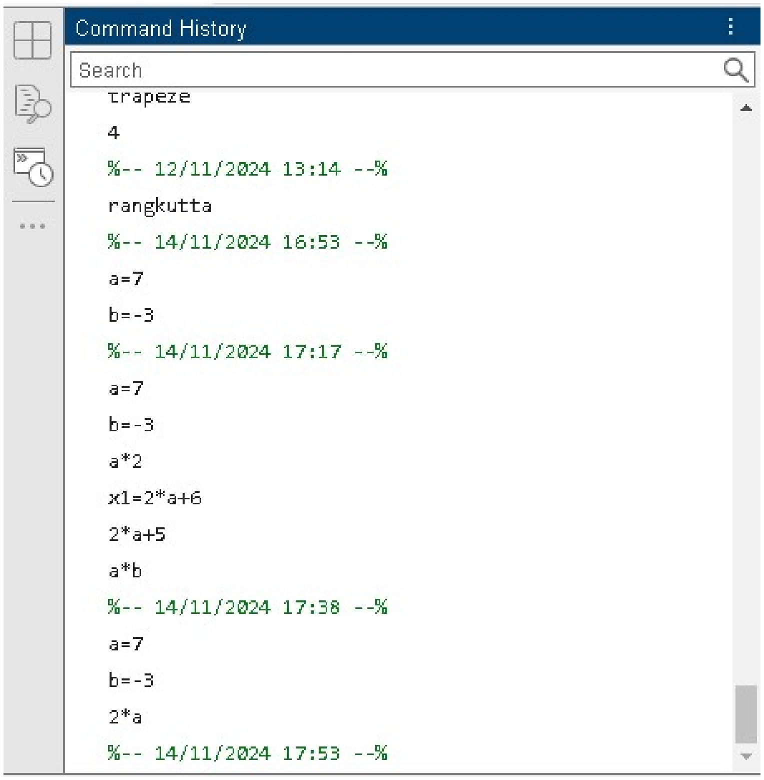


Name	Size	Date Modified	Type
coef_lagrange.m	1 KB	14/11/2023 10:15	Function
coef_newton.m	1 KB	14/11/2023 10:20	Function
coelagrange.m	1 KB	14/11/2023 10:06	Function
dicho.m	2 KB	14/11/2023 09:54	Function
dichoto.m	1 KB	14/11/2023 10:17	Script
dichotomie.m	2 KB	01/11/2024 12:23	Function
dichotomie1.m	2 KB	14/11/2023 09:45	Function
euler.m	1 KB	07/11/2023 19:23	Script
ex2.m	1 KB	21/11/2023 10:49	Script
ex3.m	2 KB	21/11/2023 10:44	Script
ex4.m	2 KB	21/11/2023 10:36	Script
exp1.m	1 KB	15/11/2024 10:06	Script
inter_newton.m	1 KB	15/11/2024 10:06	Function
L1.png	12 KB	14/11/2023 18:22	PNG File
L2.png	10 KB	14/11/2023 18:22	PNG File
LA1.fig	50 KB	14/11/2023 10:49	Figure
LA1.png	11 KB	14/11/2023 10:52	PNG File

Figure 1.4: Interface Current Directory

1.2.5 Command History

Command History inscribes orders as they are appelles 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.

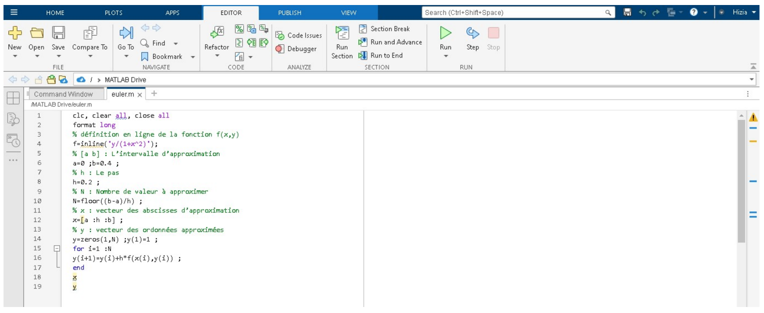


```
trapeze
4
%-- 12/11/2024 13:14 --%
rangkutta
%-- 14/11/2024 16:53 --%
a=7
b=-3
%-- 14/11/2024 17:17 --%
a=7
b=-3
a*2
x1=2*a+6
2*a+5
a*b
%-- 14/11/2024 17:38 --%
a=7
b=-3
2*a
%-- 14/11/2024 17:53 --%
```

Figure 1.5: Interface Command History

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.



```
clc, clear all, close all
format long
% définition en ligne de la fonction f(x,y)
f=inline('y/(1+x^2)');
% [a b] : L'intervalle d'approximation
a=0 ;b=0.4 ;
% h : Le pas
h=0.2 ;
% N : Nombre de valeur à approximer
N=floor((b-a)/h) ;
% x : vecteur des abscisses d'approximation
x=[a :h :b] ;
% y : vecteur des ordonnées approximées
y=zeros(1,N) ;y(1)=1 ;
for i=1 :N
y(i+1)=y(i)+h*f(x(i),y(i)) ;
end
x
y
```

Figure 1.6: Interface Editor/Debugger

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.

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:

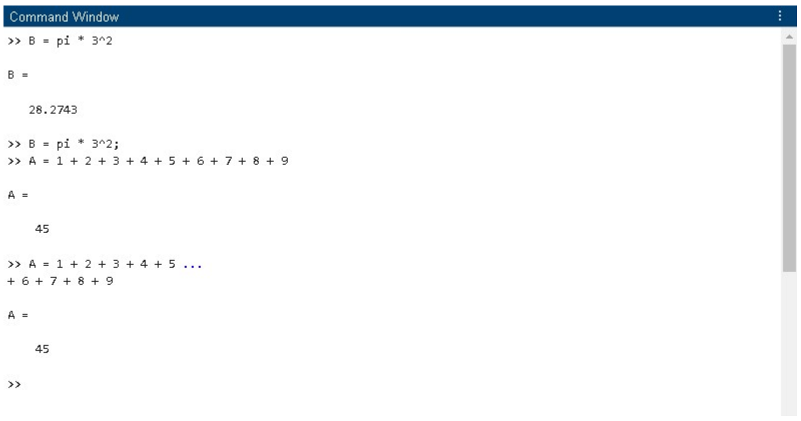
```
>>filename
```

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 procdure for your labs.

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 =

   28.2743

>> B = pi * 3^2;
>> A = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9

A =

    45

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

A =

    45

>>
```

1.2.6 Simulink

Simulink: This is the MATLAB graphics extension for working with block diagrams. SIMULINK is a tool for modeling, analysis and simulation of a wide variety of of physical and mathematical systems, including those with non-linear elements and those that use time continuous and discreet.

As an extension of MATLAB, SIMULINK adds many system-specific functions dynamic while retaining Matlab functionalities.

After starting MATLAB, there are three ways to start Simulink. You can click on the icon Simulink in the MATLAB toolbar or specify an existing Simulink file.

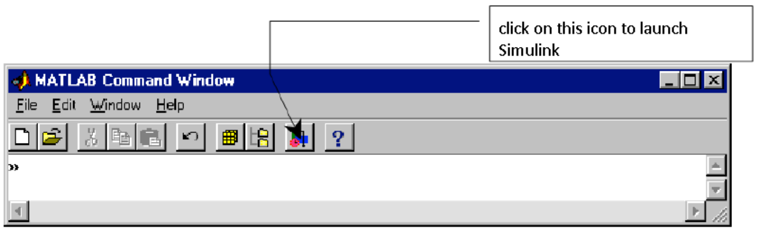


Figure 1.7: Icon to launch simulink

SIMULINK functions are grouped by type. The following window containing the libraries of simulink, appears as well as a working window.

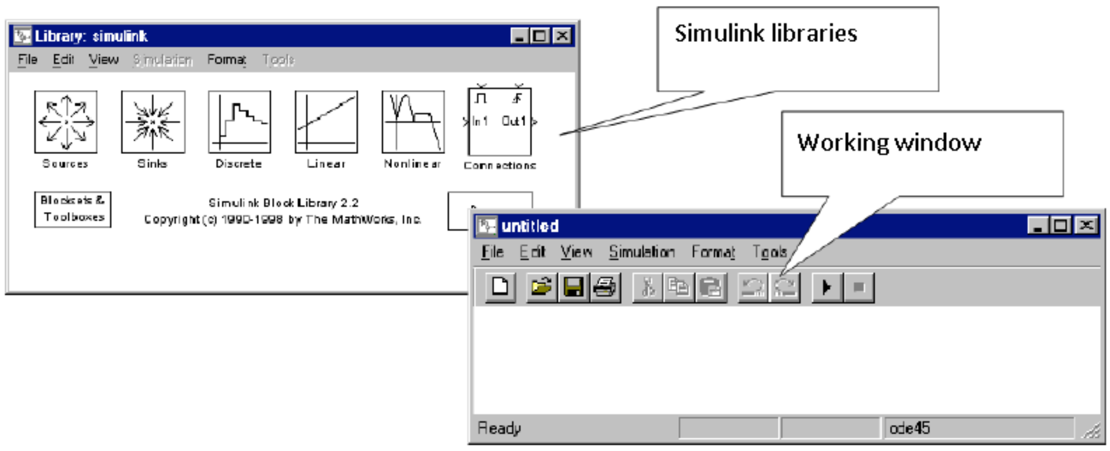

Simulink libraries
Working window
Library: simulink
File Edit View Simulation Format Tools
Sources
Sinks
Discrete
Linear
Nonlinear
Connections
Blocksets & Toolboxes
Simulink Block Library 2.2
Copyright (c) 1990-1998 by The MathWorks, Inc.
untitled
File Edit View Simulation Format Tools
Ready
ode45

Figure 1.8: Simulink libraries and Working window

Building a model in the working window:

Method of placing a component we select a Simulink library, then double-click to open it (for example, the 'Linear' library) on select a component (Sum example): we maintain pressing the left button the mouse is dragged the item in the window work, release the button. Exercise: build the environment described in the following figure, indicate above each element, the original library.

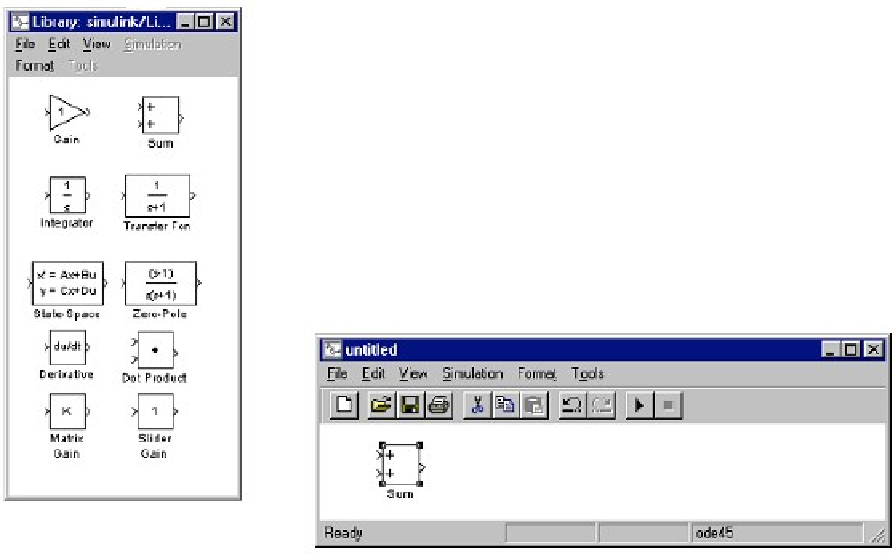


Figure 1.9: Building a model in the working window

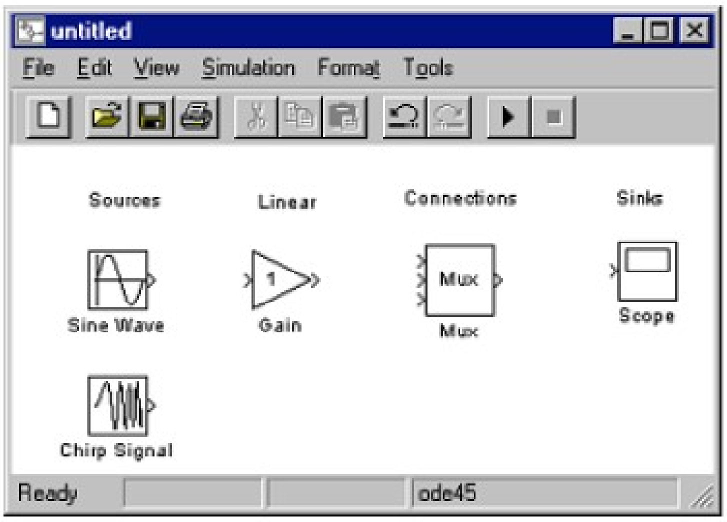


Figure 1.10: Building a model in the working window

Making connections

Method :

We select with the mouse, the symbol ¿ located on a component We keep the button pressed and pull the link to a symbol ¿ You can release the button to change the direction.

We check that the connection is correct by makes the arrow accentuated.

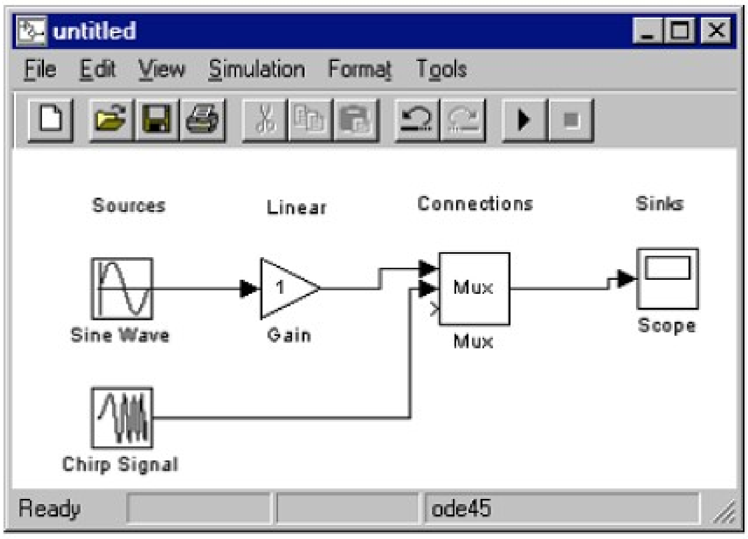
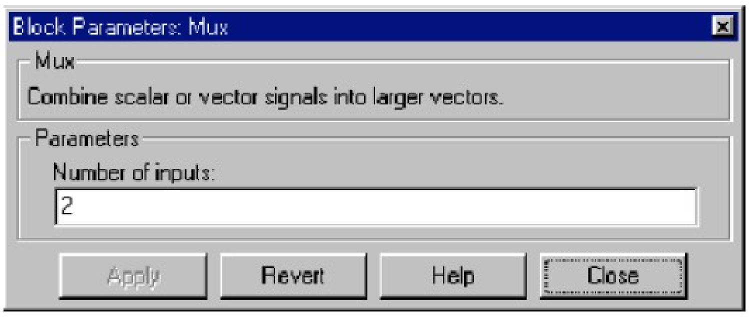


Figure 1.11: Making connections:Method 1

Component configuration:

Method :

Double-click on the 'Mux' component, window settings window opens, We type the desired values: here the value 2 to indicate 2 inputs, We close this window with Close, the new values are taken into account account.


Block Parameters: Mux

Mux

Combine scalar or vector signals into larger vectors.

Parameters

Number of inputs:

2

Figure 1.12: Making connections:Method 2

Designation of components

Each component has a default name example Gain, we can modify this name.

Method :

Click on the name

We type a new name

Signal return and recovery:

In order not to overload the drawing, we can use 2 components located in the connections library which enable wireless transition. These 2 components are called GOTO and FROM

Personalization from the working window:

It is possible to resize each component we select it, we enter a handle, we stretch or we decrease. In the Window Format menu we have other commands (you must first select a component).

- **Font:** allows you to choose the type of characters.
- **Flip name:** place the name above /below
- **Hide name:** to hide name

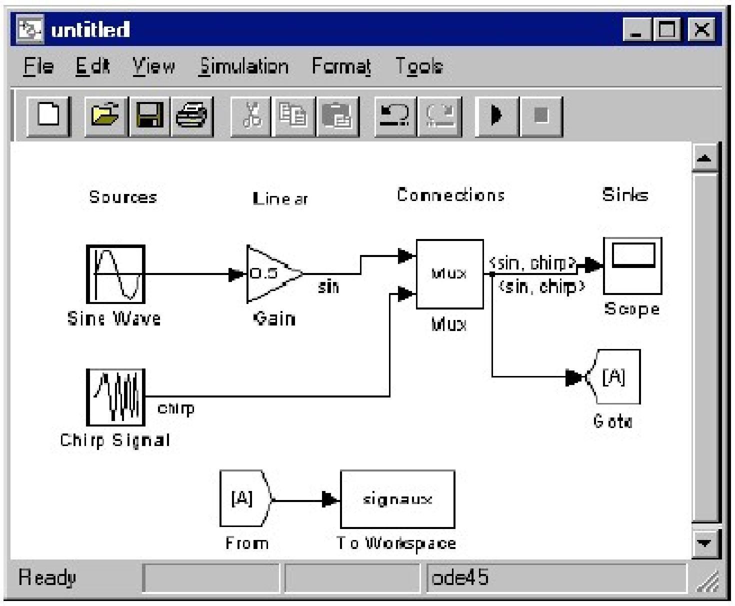


Figure 1.13: Signal return and recovery

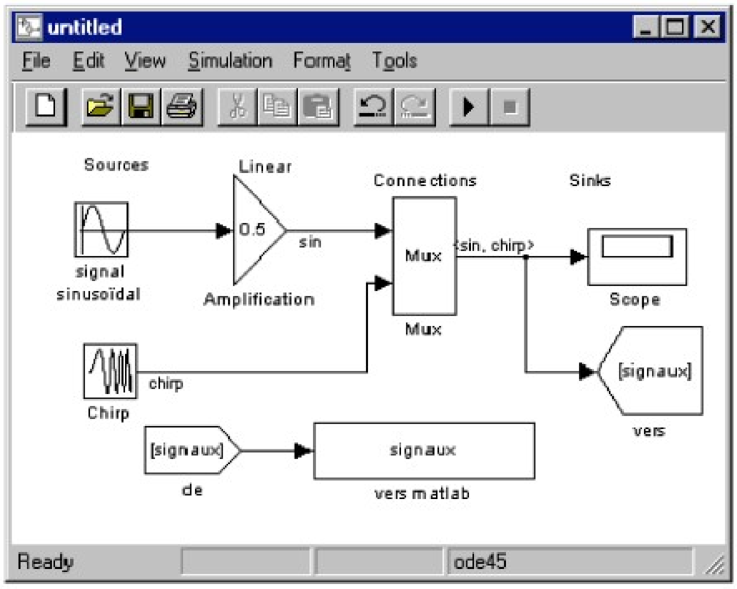


Figure 1.14: Customizing the working window

- **Flip block:** to flip the block
- **Rotate block:** to turn it 90°
- **Foreground color:** to select one color for text
- **Background color:** select a color for the block

You can also personalize the links or connections:

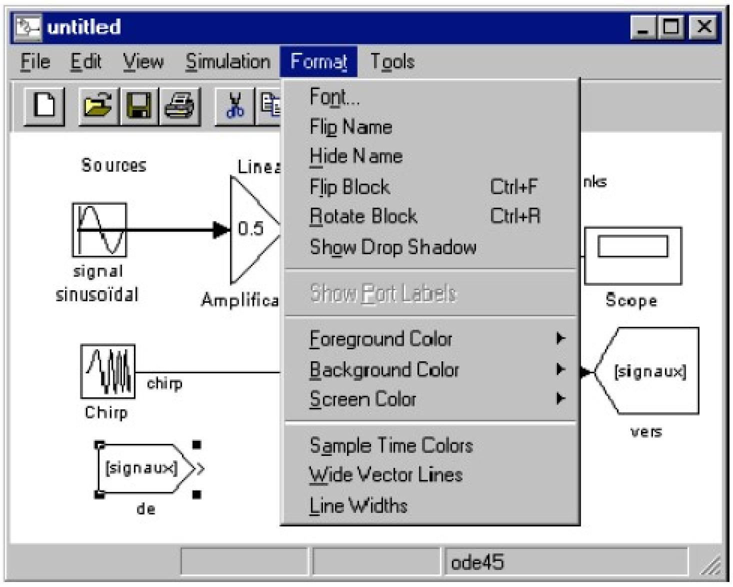
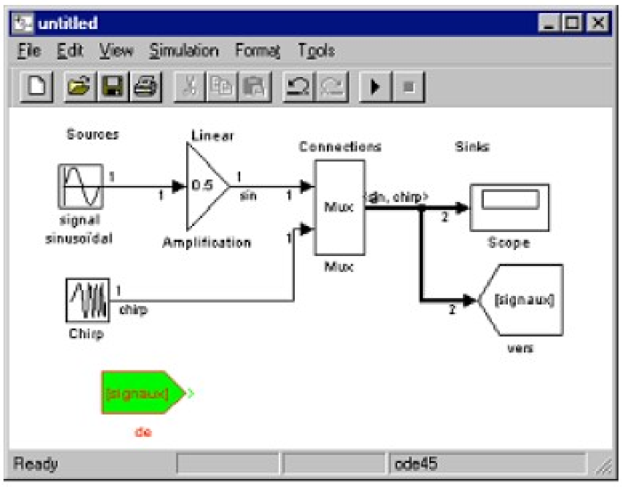


Figure 1.15: The Format menu

- **Wide Vector Lines** allows you to size the thickness of the links depending on the number of signals,

- **Line Width** allows you to obtain an indication of the number of signals on links Ctrl D allows you to update all this in case of modification.



Modifications :

Editing Components

We can :

√ Add a component at any time,

√ Delete a component by selecting it and key Delete,

- √ Modify the position of a component by selecting it and holding the left mouse button pressed and moved.
- √ Duplicate a component: select it, press the Ctrl key and drag the component.

You can go back from any operation using the **Undo** icon

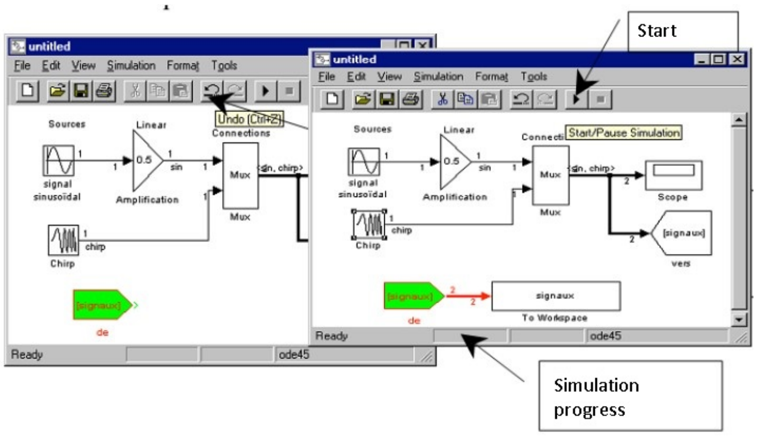
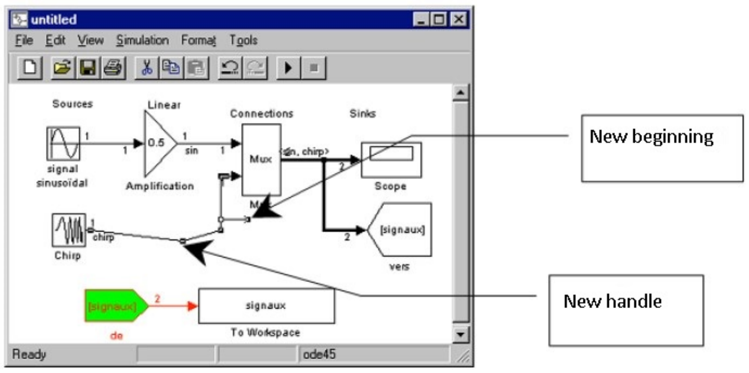


Figure 1.16: Simulation workflow

Editing links:

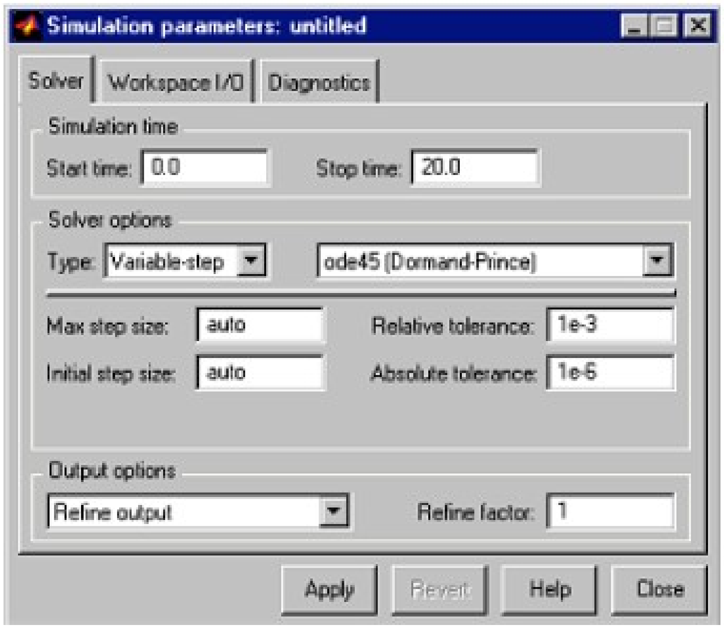
Using the handles located on the link (once selected these appear), pressing the right mouse button adds a new start, shift and left mouse button adds new ones direction change handles. The simulation uses a certain



number of parameters: simulation menu → parameters start time (0 by default) stop time (set 20s) We will study the other parameters later. Close which validates the modifications.

Launching the simulation

Simulation menu → start Or ctrl T Or icon > A ringtone indicates the end of the simulation. Exercise run the simulation after opening the scope You can zoom in /out With the first 3 icons Adjust the axes with 4° Save



Simulation time

Start time: 0.0 Stop time: 20.0

Solver options

Type: Variable-step ode45 (Dormand-Prince)

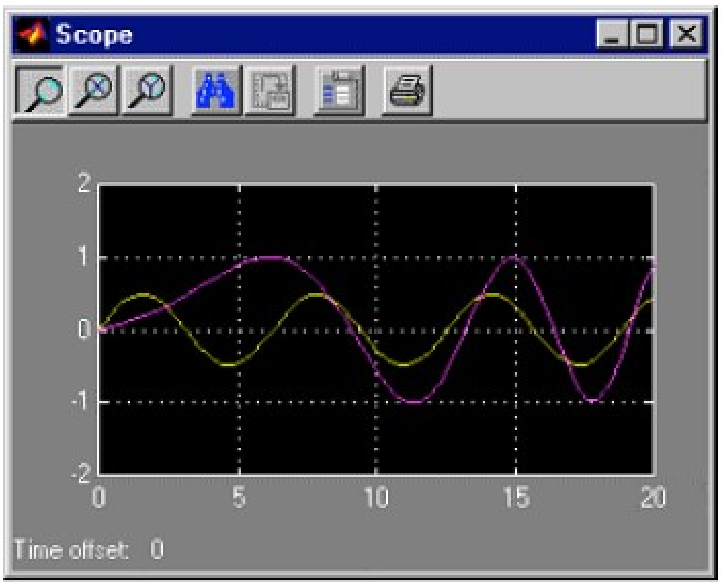
Max step size: auto Relative tolerance: 1e-3

Initial step size: auto Absolute tolerance: 1e-6

Output options

Refine output Refine factor: 1

data with 5° Adjust the scope with the 6° Print with latest



Link between simulink and MATLAB

For various operations, it is interesting to have the signals in the MATLAB or retrieve signals defined in MATLAB.

Sending signals to the MATLAB workspace

The **To Workspace** blocks from the Sinks library allow signals to be directed to the work environment. matlab in the example treated so far this is carried out with the block named "signals" on which arrives the tag of.

Recovery of signals from MATLAB

The **From Workspace** block of the Source library allows you to define signals in the Matlab environment and use them in the Simulink environment.

Chapter 2

Numbers in Matlab

2.1 Use of Matlab in the manner of a scientific calculator

MATLAB uses conventional decimal notation, with an optional decimal point '.' and the '+' or '-' sign for signed numbers. Scientific notation uses the letter 'e' to specify the scaling factor as a power of 10. Complex numbers uses the characters i and j (interchangeably) to designate the imaginary part. The following table gives a summary:

2.1.1 Variables and constants

Variable	Meaning
ans	the most recent answer
pi	number pi
eps	$\varepsilon \approx 210^{16}$
exp(1)	e=2.7183...
inf	plus infinity
-inf	minus infinity
NaN	Not-a-Number

2.1.2 Display controls

MATLAB always uses real numbers (double precision) to do calculations, which allows calculation precision of up to 16 significant digits.

the type	Examples
Integer	$5 \quad -83$
Real in decimal notation	$0.0205 \quad 3.1415926$
Real in scientific notation	$1.60210e-20 \quad 6.02252e23 \quad (1.60210 \times 10^{20} et 6.02252 \times 10^{23})$
Complex	$5+3i \quad -3.14159j$

But the following points should be noted:

- The result of a calculation operation is by default displayed with four digits after the comma.
- To display more numbers use the **format long** command (14 digits after the comma).
- To return to the default display, use the **format short** command.
- To display only 02 digits after the decimal point, use the **format bank** command.
- To display numbers as a ration, use the **format rat** command.
- The **vpa** function can be used to force the calculation to present more than significant decimals by specifying the number of decimals desired.

```
>> 8/3
ans =
2.6667
>> format long
>> 8/3
ans =
2.66666666666667
>> format bank
>> 8/3
ans =
2.67
>> format short
>> 8/3
ans =
2.6667
>> 7.2 * 3.1
```

```
ans =

22.3200

>> format rat

>> 7.2 * 3.1

ans =

558/25

>> 2.6667

ans =

26667/10000

>> sqrt(2)

ans =

1.4142

>> vpa(sqrt(2), 50)

ans =

1.4142135623730950488016887242096980785696718753769
```

MATLAB offers many commands for user interaction. We Let's content ourselves with a small set for the moment, and we will expose the others as we go along. the progress of the course.

- **who:** Shows the name of the variables used
- **whos:** Shows information about the variables used
- **clear x y:** Remove x and y variables
- **clear, clear all:** Remove all variables
- **clc:** Clears the controls screen
- **exit, quit:** Close the MATLAB environment

2.1.3 Mathematical operators

Operation	Meaning
+	addition
-	substraction
*	multiplication
/	division
^	power

2.1.4 The precedence of operations in an expression

Operation	Priority (1=max, 4=min)
Parentheses (and)	4
The power and the transpose ^ and '	3
Multiplication and division * and /	2
Addition and subtraction + and -	1

For example $5 + 2 * 3 = 11$ and $2 * 3\hat{}2 = 18$

Exercise : Create a variable n and give it the value 2, then write the following expressions:

•$3n^3 - 2n^2 + 4n$

• $\frac{e^{1+n}}{1 - \sqrt{2n}}$

• $|asin(2 * n)|$

• $\frac{ln(n)}{2n^3} - 1$

Solution : $>> n = 2;$

$>> 3 * n^3 - 2 * n^2 + 4 * n;$

$>> exp(1 + n)/(1 - sqrt(2 * n));$

$>> abs(asin(2 * n));$

$>> log(n)/(2 * n^3) - 1;$

2.1.5 Mathematics functions

Among the frequently used functions, the following can be noted:

The function	Meaning
sin(x)	sinus (in radians)
cos(x)	cosinus (in radians)
tan(x)	tangent (in radians)
asin(x)	sinus reverse (in radians)
acos(x)	cosinus reverse (in radians)
atan(x)	tangent reverse (in radians)
sqrt(x)	square root
abs(x)	absolute value
exp(x)	exponential
log(x)	Natural logarithm
log10(x)	decimal logarithm
round(x)	rounding a number to the nearest integer
floor(x)	rounding a number to the smallest integer
ceil(x)	rounding a number to the largest integer
gcd(x,y)	the GCD of x and y
lcm(x,y)	the PCCM of x and y
isprime(x)	Test if the number x is prime or not (the result is of logical type)
factor(x)	the divisors of x except 1 and itself

sign: will display the sign of a value.

y = sign(x)

0: if the corresponding element is equal to zero.

1: if the corresponding element is greater than zero.

-1: if the corresponding element is less than zero.

>> $l = 4;$

>> $m = -6;$

>> $n = 0;$

>> $sign(l)$

$ans = 1$

>> $sign(m)$

$ans = -1$

>> $sign(n)$

$ans = 0$

2.1.6 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.

Functions:

The function	Meaning
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)

Chapter 3

Vectors and Matrices

3.1 Vectors

MATLAB was originally designed to enable mathematicians, scientists and engineers to easily use the mechanisms of linear algebra. By Therefore, the use of vectors and matrices is very intuitive and convenient in MATLAB.

Definition 3.1.1: Vector

vector is an ordered list of elements. If the elements are arranged horizontally we say that the vector is a line vector, on the other hand if the elements are arranged vertically we say that it is a column vector.

To create a line vector, simply write the list of its components in square brackets [and] and separate them with spaces or commas as follows:

Example 3.1.1. $>> K = [5, 2, 13, -6]$ *% Creation of a line vector K*

$K =$

$5 \ \ 2 \ \ 13 \ -6$

$>> L = [4 \ \ -2 \ \ 1]$ *% Creation of a line vector L*

$L =$

$4 \ -2 \ \ 1$

To create a column vector it is possible to use one of the following methods:

- write the components of the vector in square brackets [and] and separate them by semicolons (;) as follows:

  $>> L = [4; -2; 1]$ % Creating a column vector L

$L =$

4

-2

1

- Write the vector vertically:

  $>> L = [$

  4

  -2

  1

  $]$

  $L =$

  4

  -2

  1

- Calculate the transpose of a line vector:

  $>> L = [4, -2, 1]'$

  $L =$

  4

  -2

  1

If the components of a vector X are ordered with consecutive values, we can denote it using the following notation:

$\boxed{\text{X = first-element : last-element}}$ (Brackets are optional in this case)

$>> M = 1 : 8$

$M =$

1 2 3 4 5 6 7 8

$>> M = [1:8]$

$M =$

1 2 3 4 5 6 7 8

If the components of a vector step (increment/decrement) different from 1, we can specify the step with the notation:

$\boxed{\text{X = first element: step: last element}}$ (Brackets are optional)

$>> N = [0:2:10]$

$N =$

0 2 4 6 8 10

$>> N = [-4:2:6]$

$N =$

-4 -2 0 2 4 6

$>> N = 0:0.2:1$

$N =$

0 0.2000 0.4000 0.6000 0.8000 1.0000

We can write more complex expressions like:

$>> E = [1:2:5, -2:2:1]$

$E =$

1 3 5 -2 0

$>> T = [1\ \ 2\ \ 3]$

$T =$

1 2 3

$>> R = [T, 4, 5, 6]$

$R =$

1 2 3 4 5 6

3.1.1 Referencing and accessing elements of a vector

Access to the elements of a vector is done using the following general syntax:

$$name\ vector(positions)$$

▼ Parentheses (and) are used for consultation. The square brackets [and] are used only during creation.

▼ positions: can be a simple number, or a list of number (a vector of positions)

$>> I = [5, -1, 13, -6, 7]$ % creation of the vector I which contains 5 elements

$I =$

$5 \quad -1 \quad 13 \quad -6 \quad 7$

$>> I(3)$ % 3^{rd} position

$ans =$

13

$>> I(2:4)$ % from second position to fourth

$ans =$

$-1 \quad 13 \quad -6$

$>> I(4:-2:1)$ % from the 4^{th} position to the 1^{st} with step = -2

$ans =$

$-6 \quad -1$

$>> I(3:end)$ % from the 3^{rd} position to the last

$ans =$

$13 \quad -6 \quad 7$

$>> I([1,3,4])$ the 1^{st}, the 3^{rd} and the 4^{th} position only

$ans =$

$5 \quad 13 \quad -6$

$>> I(1) = 8$ % give the value 8 to the first element

$I =$

$8 \quad -1 \quad 13 \quad -6 \quad 7$

$>> I(6) = -3$ % add a sixth element with value -3

$I =$

$8 \quad -1 \quad 13 \quad -6 \quad 7 \quad -3$

$>> I(9) = 5$ % add a ninth element with value 5

$I =$

$8 \quad -1 \quad 13 \quad -6 \quad 7 \quad -3 \quad 0 \quad 0 \quad 5$

$>> I(2) = [\ ]$ % Delete the second element

$I =$

$8 \quad 13 \quad -6 \quad 7 \quad -3 \quad 0 \quad 0 \quad 5$

$>> I(3:5) = [\ ]$ % Delete from 3^{rd} to 5^{th} element

$I =$

$8 \quad 13 \quad 0 \quad 0 \quad 5$

3.1.2 Element-by-element operations for vectors

With two vectors $\overrightarrow{p}$and $\overrightarrow{s}$, it is possible to perform element-by-element calculations in using the following operations:

★ Addition of vectors (+)

$>> p = [-2, 6, 1];$

$>> s = [3, -1, 4];$ $>> p + 2$

$ans =$

$0 \quad 8 \quad 3$

$>> p + s$

$ans =$

$1 \quad 5 \quad 5$

★ Subtraction of vectors $(-)$

$>> p - 2$

$ans =$

$-4 \quad 4 \quad -1$

```
>> p − s

ans =

− 5   7   − 3
```

★ Element-by-element multiplication (.∗)

```
>> p ∗ 2

ans =

− 4   12   2

>> p. ∗ 2

ans =

− 4   12   2

>> p. ∗ s

ans =

− 6   − 6   4
```

★ Division element by element (./)

```
>> p/2

ans =

− 1.0000   3.0000   0.5000

>> p./2

ans = −1.0000   3.0000   0.5000

>> p./s

ans =

− 0.6667   − 6.0000   0.2500
```

★ Element-by-element power(.∧)

```
>> p.^2

ans =

4   36   1
```

$>> p.\hat{}s$

$ans =$

$-8.0000 \quad 0.1667 \quad 1.0000$

3.1.3 The linspace function

The creation of a vector whose components are ordered by regular interval and with a well determined number of elements can be achieved with the function:

$$\boxed{\text{linspace (beginning, end, number of elements)}}$$

The increment step is automatically calculated by MATLAB according to the formula:

$$step = \frac{end - beginning}{number\ of\ elements - 1}$$

$>> G = linspace(1, 10, 4)$ % a vector of four elements from 1 to 10

$G =$

1 4 7 10

$>> z = linspace(13, 40, 4)$ % a vector of four elements from 13 to 40

$z =$

13 22 31 40

The size of a vector (the number of its components) can be obtained with the length function as follows:

$>> length(G)$ % the size of the vector

$ans =$

4

3.2 The matrices

A matrix is a rectangular array of (two-dimensional) elements. Vectors are matrices with a single row or a single column (one-dimensional). To insert a matrix, you must respect the following rules:

- Elements must be enclosed in square brackets [and]
- Spaces or commas are used to separate elements in the same line

- A semicolon (or the enter key) is used to separate lines

To illustrate this, considering the following matrix:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

Cette matrice pout tre crite en MATLAB avec une des syntaxes suivantes :

$>> D = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12];$

$>> D = [1 \quad 2 \quad 3 \quad 4; 5 \quad 6 \quad 7 \quad 8; 9 \quad 10 \quad 11 \quad 12];$

$>> D = [1, 2, 3, 4$

$5, 6, 7, 8$

$9, 10, 11, 12];$

$>> D = [[1; 5; 9], [2; 6; 10], [3; 7; 11], [4; 8; 12]];$

The number of elements in each row (number of columns) must be the same in all rows of the matrix, otherwise an error will be reported by MATLAB. For example :

$>> Q = [1 \;\; 2; 4 \;\; 5 \;\; 6]$

Error using vertcat

CAT arguments dimensions are not consistent.

A matrix can be generated by vectors as shown in the following examples:

$>> h = 1 : 4$ % creation of a vector h

$h =$

1 2 3 4

$>> f = 5 : 5 : 20$ % creation of a vector f

$f =$

5 10 15 20

$>> e = 4 : 4 : 16$ % creation of a vector e

$e =$

4 8 12 16

$>> R = [h; f; e]$ % R is formed by the line vectors h, f and e

$R =$

```
 1   2   3   4
 5  10  15  20
 4   8  12  16
```
$>> M = [h' \;\; f' \;\; e']$ % M is formed by the column vectors h, f and e

$M =$
```
 1   5   4
 2  10   8
 3  15  12
 4  20  16
```
$>> Q = [h; h]$ % Q is formed by the same line vector h 2 times

$Q =$

```
 1  2  3  4
 1  2  3  4
```

3.2.1 Referencing and accessing elements of a matrix

Access to the elements of a matrix is done using the following general syntax: Access to the elements of a matrix is done using the following general syntax:

$$\boxed{\text{matrix name (row positions, column positions)}}$$

• Positions: maybe a simple number, or a list of number (a vector of positions)

• Parentheses (and) are used for reference). The square brackets [and] are used only during creation.

It is useful to note the following possibilities:

- ▶ Access to an element of row i and column j is done by: W(i,j)
- ▶ Access to the entire line number i is done by: W(i,:)
- ▶ Access to the entire column number j is done by: W(:,j)

$>> W = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12]$ % creation of matrix W

$W =$
```
 1   2   3   4
 5   6   7   8
 9  10  11  12
```
$>> W(2, 3)$ % the element on the 2^{nd} row to the 3^{rd} column

$ans =$

7

$>> W(1, :)$ % all elements of the 1^{st} line

$ans =$

```
1   2   3   4

>> W(:,2)          % all elements of the 2nd column

ans =

2

6

10

>> W(2:3,:)          % all elements of the 2nd and 3nd line

ans =

 5    6    7    8
 9   10   11   12

>> W(1:2,3:4)          % The upper right submatrix of size 2 × 2

ans =

 3   4
 7   8

>> W([1,3],[2,4])          % the sub-matrix: rows (1,3) and columns (2,4)

ans =

  2    4
 10   12

>> W(:,3) = [ ]          % Remove third column

W =
 1    2    4
 5    6    8
 9   10   12
>> W(2,:) = [ ]          %Delete second line

W =

 1    2    4
 9   10   12

>> W = [W,[0;0]]          % Add a new column with W(:,4) = [0;0]

W =

 1    2    4   0
 9   10   12   0

>> W = [A;[1,1,1,1]]          % Add a new line with W(3,:) = [1,1,1,1]

W =
 1    2    4   0
 9   10   12   0
 1    1    1   1
```

The dimensions of a matrix can be acquired using the function **size**. However, with a matrix W of dimension $m \times n$ the result of this function is a vector of two components, one for m and the other for n.

```
>> i = size(W)
i =
3  4
```

Here, the variable d contains the dimensions of the matrix A in the form of a vector. To obtain the dimensions separately we can use the syntax:

```
>> s1 = size(W, 1)        % s1 contains the number of lines (m)
s1 =
3
>> s2 = size(W, 2)        %s2 contains the number of column (n)
s2 =
4
```

3.2.2 Automatic generation of matrices

In MATLAB, there are functions that allow you to automatically generate particular matrices. In the following table we present the most used ones:

The function	Meaning
zeros(n)	Generates an $n \times n$ matrix with all elements = 0
zeros(m,n)	Generates an $m \times n$ matrix with all elements = 0
ones(n)	Generates an $n \times n$ matrix with all elements = 1
ones(m,n)	Generates a matrix $m \times n$ with all elements = 1
eye(n)	Generates a dimension identity matrix $n \times n$
magic(n)	Generates a magic dimension matrix $n \times n$
rand(m,n)	Generates a matrix of dimension $m \times n$ of random values

3.2.3 Basic operations on matrices

The operation	Meaning
+	Addition of vectors
-	Subtraction of vectors
.*	Element-by-element multiplication
./	Division element by element
.	Reverse division element by element
.∧	Element-by-element power
$*$	Matrix multiplication
/	Matrix division $(A/B) = (A * B^{-1})$

The element-by-element operations on matrices are the same as those for vectors (the only condition necessary to do an element-by-element operation is that the two matrices have the same dimensions). On the other hand, the multiplication or division of matrices requires some constraints (consult a course on algebra matrix for more details).

$>> H = ones(2,3)$

$H =$

1 1 1

1 1 1

$>> G = zeros(3,2)$

$G =$

0 0

0 0

0 0

$>> G = G + 3$

$G =$

3 3

3 3

3 3

$>> H * G$

$ans =$

9 9

9 9

$>> G = [G, [3 \quad 3 \quad 3]'], \qquad or G(:,3) = [3 \quad 3 \quad 3]$

$G =$

3 3 3

3 3 3

3 3 3

$>> G = G(1:2,:), \qquad or\ G(3,:) = [\ ]$

```
G =

3 3 3

3 3 3

>> H = H * 2

H =

2 2 2

2 2 2

>> H. * G

ans =

6 6 6

6 6 6

>> H * eye(3)

ans =

2 2 2

2 2 2
```

3.2.4 Useful functions for processing matrices

Here are some of the most used functions regarding matrices:

The function	The usefulness
det	Calculating the determinant of a matrix
inv	Calculate the inverse of a matrix
rank	Calculate the rank of a matrix
trace	Calculate the trace of a matrix
eig	Calculate eigenvalues
dot	Calculate the dot product of 2 vectors
norm	Calculate the norm of a vector
cross	Calculate the cross product of 2 vectors
diag	Returns the diagonal of a matrix
diag(V)	Creates a matrix having the vector V in the diagonal and 0 elsewhere
tril	Returns the lower triangular part
triu	Returns the upper triangular part
sum	Sum of elements
prod	Produces elements
mean	Average value
fliplr	Flip from left to right
flipud	Flipping from top to bottom
rot90	Rotation at 90°
diff	Approximation of the derivative
[U,D]=eig(A)	eigenvectors and eigenvalues of A
eig(A)	eigenvalues of A
expm(A)	exponential of A
reshape(A,u,v)	creates a matrix of size [u,v], from A

$>> P = [1, 2; 3, 4];$

$>> det(P)$

$ans =$

-2

$>> inv(P)$

$ans =$

$-2.0000 \quad 1.0000$

$1.5000 \quad -0.5000$

$>> rank(P)$

$ans =$

2

$>> trace(P)$

$ans =$

5

$>> eig(P)$

```
ans =

− 0.3723

5.3723

>> s = [−1, 5, 3];

>> p = [2, −2, 1];

>> dot(s, p)

ans =

− 9

>> norm(s)

ans =

3

>> cross(s, p)

ans =

− 11    − 7    8

>> diag(P)

ans =

1

4

>> r = [−5, 1, 3];

>> diag(r)

ans =

− 5    0    0

0    1    0

0    0    3

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

K =

1    2    3
```

```
4   5   6

7   8   9

>> tril(K)

ans =

1   0   0

4   5   0

7   8   9

>> tril(K, -1)

ans =

0   0   0

4   0   0

7   8   0

>> tril(K, -2)

ans =

0   0   0

0   0   0

7   0   0

>> triu(K)

ans =

1   2   3

0   5   6

0   0   9

>> triu(K, -1)

ans =

1   2   3

4   5   6

0   8   9
```

```
>> triu(K,1)

ans =

0   2   3

0   0   6

0   0   0

>> T = [1 2 5; 4 7 8; 5 2 8];

>> sum(T)

ans =

10 11 21

>> prod(T)

ans =

20 28 320

>> mean(T)

ans =

3.3333 3.6667 7.0000

>> fliplr(T)

ans =

5   2   1

8   7   4

8   2   5

>> flipud(T)

ans =

5   2   8

4   7   8

1   2   5

>> rot90(T)

ans =
5  8  8

2  7  2

1  4  5
```

Chapter 4

Programming elements

So far we have seen how to use MATLAB to execute commands or to evaluate expressions by writing them in the line command (After the >> prompt), therefore the commands used are generally written in the form of a single instruction (possibly on a single line). However, there are problems for which the description of their solutions requires multiple instructions, which requires the use of multiple lines. For example, finding the roots of a second degree equation (with taking into account all possible cases).

A collection of well-structured instructions aimed at solving a problem data is called a program. In this part of the course, we will present the mechanisms for writing and executing programs in MATLAB.

4.1 Character strings

Character strings are defined between simple 'quotes': c = 'This is a text' will define the variable c as a character string that can be used to display the message between quotes. available(c) will display the string c on the screen.

Entering a string

```
>>ch='Introduction  à  matlab'
ch=
Introduction  à  matlab
```

The length of a string

```
>>length(ch)
ans=
21
```

The dimensions of a string

```
>>size(ch)

ans=

1  21
```

Comparison between 2 strings

```
>>strcmp('Matlab','Simulink')

ans=

0

>>strcmp('Matlab','Matlab')

ans=

1
```

Comparing 2 strings over a defined number of characters

```
>>strncmp('Matlab','Math',3)

ans=

1
```

The capital letter of a string

```
>>ch1='matlab';

>>upper(ch1)

ans=

MATLAB
```

The lowercase of a string

```
>>ch2='CONSTANTINE';

>>lower(ch2)
```

```
ans=

constantine
```

The ASCII code of a string

```
>>code=abs(ch2)

code=

67 79 78 83 84 65 78 84 73 78 69
```

Converting an ASCII code to characters

```
>>setstr(code)

code=

CONSTANTINE
```

Concatenating strings

• **Horizontal concatenation**

```
>>ch1='matlab';

>>ch2='2024'

>>strcat(ch1,ch2)

ans=

matlab 2024
```

• **Vertical concatenation**

```
>>ch3='math work';

>>strvcat(ch1,ch2,ch3)

ans=

matlab

2024

math work
```

Searching for a substring in another string

```
>>H='university IBN KHALDOUN'

>>findstr(H,'N')

ans=

13 21



>>findstr(H,'m')

ans=

[ ]
```

Type checking a variable

```
>>l='fmi tiaret'.

>>isstr(l)

ans=

1

>>m='2024'

>>isstr(m)

ans=

0
```

Decimal to hexadecimal

```
>>d=12

>>f=dec2hex(d)

f=

c
```

Hexadecimal to decimal

```
>>h='10'

>>p=hex2dec(h)
```

```
p=

16
```

Decimal to Binary

```
>>f=10

>>bin=dec2bin(f)

bin=

1010

>>bin=dec2bin(f,8)

bin=

00001010
```

Binary to decimal

```
>>b=00001111

>>j=bin2dec(b)

j=15
```

4.2 Generality

4.2.1 Variables, assignment

The assignment instruction name$\longleftarrow$value code: name=value. If there is no variable named name, this statement creates a variable called name and assigns value to it. Otherwise, its value will be modified (replaced) with the new value. When a variable is created, its name appears in the workspace an icon which indicates the nature of its content, and the description of its memory size.

Valid variable names start with a letter and are one word. MATLAB is case sensitive. There are five types of variables in MATLAB: integers, real numbers, complexes, character strings and the logical type.

4.2.2 Comments

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

>> $A = B + C;$ % A is the value of B+C

4.2.3 Writing long expressions

If the writing of a long expression cannot be enclosed in a single line, it is possible to divide it into several lines by putting at the end of each line at least three points.

>> $(sin(pi/3)\hat{}2/cos(pi/3)\hat{}2) - (1 - 2 * (5 + sqrt(x)\hat{}5/(-2 * x\hat{}3 - x\hat{}2)\hat{}1 + 3 * x));$

This expression can be rewritten as follows:

>> $(sin(pi/3)\hat{}2/cos(pi/3)\hat{}2) \cdots \hookleftarrow$

>> $(1 - 2 * (5 + sqrt(x)\hat{}5 \cdots \hookleftarrow$

>> $/(-2 * x\hat{}3 - x\hat{}2)\hat{}1 + 3 * x)); \hookleftarrow$

4.2.4 Inputs/Outputs

Reading data into a program (Inputs)

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

variable = input (An indicative sentence)

•**variable**:The value deposited by the user will be put in this variable

•**An indicative sentence**:A sentence helping the user to know what to enter

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

```
>>m=input ('Enter an integer: ')↩

Enter an integer:5↩

m=

5

>>m=input ('Enter an integer: ');↩
```

```
Enter an integer:5↩

>>n= input ('Enter a row vector: ')↩

Enter a row vector:[1:2:8,3:1:0]↩

n=

1  3  5  7  3  2  1  0
```

4.2.5 Writing data into a program (Outputs)

We have already seen that MATLAB can display the value of a variable by only typing the name of the latter.

```
>> m = 5;

>> m      % Ask Matlab to display the value of m

m=

5
```

With this method, MATLAB writes the name of the variable (A) then the sign (=) followed by the desired value. However, there are cases where we want to display only the value of the variable (without the name and without the = sign).

To do this, we can use the **disp** function, which has the following syntax: **disp (object)** The value of the object can be a number, a vector, a matrix, a string of characters or an expression.

Note that with an empty vector or matrix, **disp** displays nothing.

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

5

>> disp(m);     %The semicolon has no effect

5

>> n     %Display vector n using the classic method

n =

1  3  5  7  3  2  1  0

>> disp(n)     %Show vector n without 'n = '

1  3  5  7  3  2  1  0

>> J = 3 : 1 : 0     %Creating an empty J vector
```

```
J =

Empty matrix: 1-by-0

>> disp(C)      %disp displays nothing if the vector is empty

>>
```

4.3 Logical expressions

4.3.1 Logical operations

The comparison operation	Meaning
==	egality
=	inequality
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

The logical operation	Meaning
&	the and logical
|	the logical or
~	logical negation

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

- ♣ Any value equal to 0 will be considered false ($= 0 \Rightarrow$ False)
- ♣ Any value other than 0 will be considered true ($\neq 0 \Rightarrow$True).

The following table summarizes how logical operations work:

P	Q	$P\&Q$	$P\|Q$	$\sim Q$
1 (true)	1 (true)	1	1	0
1 (true)	0 (false)	0	1	0
0 (false)	1 (true)	0	1	1
0 (false)	0 (false)	0	0	1

```
>> P = 10;

>> Q = 20;

>> P < Q          %displays 1 (true)

ans =

1

>> P <= 10        %displays 1 (true)

ans =

1
```

```
>> P == Q            %displays0(false)

ans =

0

>> (0 < P)&(Q < 30)            %displays 1 (true)

ans =

1

>> (P > 10)|(Q > 100)            %displays 0 (false)

ans =

0

>>  (P > 10)            %displays 1 (true)

ans =

1

>> 10&1            % 10 is considered true therefore 1&1 = 1

ans =

1

>> 10&0            %1&0 = 1

ans =

0

>> Z = [1, 5, 8]

>> D = [7, −5, 5]

>> Z > D

ans =

0   1   1

>> Z < D

ans =

1   0   0

>> Z = D
```

```
ans =

0   0   0

>> Z  = D

ans =

1   1   1
```

4.3.2 Comparison of matrices

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

- **isequal**: tests if two (or more) matrices are equal (having the same elements everywhere). It returns 1 if this is the case, and 0 otherwise.
- **isempty**: tests if a matrix is empty (contains no elements).

  It returns 1 if this is the case, and 0 otherwise.

To better understand the impact of these functions, let's follow the following example:

```
>> T = [5, 2; -1, 3]          % Create matrix T

T =

5    2

-1   3

>> Y = [5, 1; 0, 3]          % Create matrix Y

Y =

5   1

0   3

>> T==Y          % Test if T=Y? (1 or 0 depending on position)

ans =

1   0

0   1

>> isequal(T,Y)          % Test if indeed T and Y are equal (the same)
```

```
ans =

0

>> I=[ ] ;              % Create the empty matrix I

>> isempty(I)           % Test if I is empty (shows true = 1)

ans =

1

>> isempty(T)           % Test if T is empty (displays false = 0)

ans =

0
```

4.4 Flux Control Structures

Flow control structures are instructions for defining and controlling manipulate the order of execution of tasks in a program. They offer the possibility of carry out different processing depending on the state of the program data, or perform repetitive loops for a given process.

MATLAB has eight flow control structures, namely:

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

The if statement

The if statement is the simplest and most widely used control flow structure. She allows you to guide the execution of the program according to the logical value

```
if (condition)                                  if (condition)

      instruction 1                               instruction set 1

      instruction 2                 or          else
 ⋮
                                                  instruction set 2

    instruction N                               end
```

If the condition evaluates to true, the statements between the if and the end will be executed, otherwise they will not be (or if an else exists the instructions between the else and the end will be executed). If it is necessary to check several conditions instead of just one, we can use elseif clauses for each new condition, and at the end we can put a else in the case where no condition has evaluated to true. So here is the general syntax:

```
if (expression 1)

      Instruction Set 1

else if (expression 2)

      Instruction Set 2

else if (expression n)

      Instruction set n

else

      Set of instructions if all expressions were false

end
```

the following program defines you according to your age:

```
>> age = input('Enter your age : ');
if (age < 2)
    disp('You are a baby')
elseif (age < 13)
    disp('You are a child')
```

```
elseif (age < 18)
    disp ('You are a teenager')
elseif (age < 60)
        disp ('You are an adult')
else
        disp ('You are an old man')
end
```

```
Enter 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 old man
```

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

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

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

All you have to do is write your program in this window, then save it with a name (for example: first Program.m). It is reported that The extension of M-Files files is always .m.

Now if we want to run our program, just go to the guest usual command (¿¿) then type the name of our file (without the .m) like this:

```
>> first Programme ↩
```

And the program execution will start immediately.

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

```
>> edit first Program
```

Let's create a program that finds the roots of a quadratic equation designated by: $ax^2 + bx + c = 0$.

Here is the M-File which contains the program, it is saved with the name: Equation2deg.m

```
% Resolution program the equation a*x^2 + b*x + c = 0
a = input ('Enter the value of a : '); % read a
b = input ('Enter the value of b : '); % read b
c = input ('Enter the value of c : '); % read c
delta = b^2 - 4*a*c; % Calculate delta
if delta < 0
  disp('No solution') % No solution
elseif delta== 0
  disp('Dual solution : ') % Dual solution
X = b/(2*a)
else
  disp('Two distinct solutions: ') % Two solutions
X1 = (-b + sqrt(delta))/(2*a)
X2 = (-b - sqrt(delta))/(2*a)
end
```

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

```
>>Equation2deg ↩
Entrez la valeur de a : -2 ↩
Entrez la valeur de b : 1 ↩
Entrez la valeur de c : 3 ↩
```

Deux solutions :

X1 =

-1

X2 =

1.5000

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

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

>> solve('$-2 * x^2 + x + 3 = 0$','x')

ans =

-1

3/2

The switch statement

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

So the general form of this instruction is:

```
switch (expression)     case value 1

          Instruction group 1

     case value 2

          Instruction group 2

          ⋮

     case value n
```

```
        Instruction group n

    otherwise

        Group of instructions if all boxes failed

end
```

Example :

```
x = input ('Enter a number : ') ;

switch(x)

    case 0

        disp('x = 0 ')

    case 10

        disp('x = 10 ')

    case 100

        disp('x = 100 ')

    otherwise

        disp('x is not 0 or 10 or 100 ')

end
```

The execution will give:

```
Enter a number: 50 ↩

x is not 0 or 10 or 100
```

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
```

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 vector

        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

a

end

Enter a number of your choice:

3

a =
1.0000   0.5000    0.3333

0        0         0

0        0         0

a =

1.0000    0.5000    0.3333

0.5000    0.3333    0.2500

0         0         0

a =

1.0000   0.5000    0.3333

0.5000   0.3333    0.2500

0.3333    0.2500    0.2000
```

Example 2

```
n = 4

x = [ ]

for i = 1:n

x = [x, i^2]

end

n =

4

x =

[ ]

x =

1

x =

1    4

x =

1    4    9
```

```
x =

1    4    9    16
```

Example 3:

```
m = 4

n = 3

for i = 1:m

for j = 1:n

H(i,j) = 1/(i+j-1) ;

end

end

H

>> m = 4

n = 3

H =

1.0000   0.5000   0.3333

0.5000   0.3333   0.2500

0.3333   0.2500   0.2000

0.2500   0.2000   0.1667
```

4.4.1 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 condition

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. Enter a number (0 to finish)

2

Enter a number (0 to finish)

0

>> **Example 2:**

eps = 1;

while (1+eps) > 1

eps = eps/2;

end

eps

eps =

1.1102e-16

Example 3:

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

n = 1;

while prod(1:n) < 1e100

n = n + 1;

end

n

n =

70

4.4.2 Continue and break outputs during instruction

In loop structures, a block of instructions is executed a certain number of times, and this number of times can even be infinite in the case of while loops.

Two instructions allow you to interrupt the execution of a block of instructions in a loop.

Continue: Interrupts execution of the currently executing block of instructions, and moves to the next iteration of the for or while loop.

Break: interrupts the execution of the block of instructions currently being executed, and completely exits the for or while loop, ignoring subsequent iterations

Example 1 (continue): the MATLAB program which allows you to display numbers divisible by 7 between 7 and 50. The continue instruction allows if a number not divisible by 7 to ignore its display with the disp instruction and move on to next number.

```
for n = 1:50
if mod(n,7)
continue
end
disp(['Divisible by 7: ' num2str(n)])
end
```

The execution gives:

```
Divisible by 7: 7
Divisible by 7: 14
Divisible by 7: 21
Divisible by 7: 28
Divisible by 7: 35
Divisible by 7: 42
Divisible by 7: 49
```

Example 2 (break):

```
c=input('give a number ')
n=0;
while n<=c
x=input('enter a number ');
```

```
if x==0

break

end

disp(['the number entered is',num2str(x)])

n=n+1;

end
```

The execution gives:

```
give a number 5

c =5

enter a number 3

the number entered is 3

enter a number 4

the number entered is 4

enter a number 0

>>
```

4.4.3 The functions

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

1−In computing, a function is a routine (a subroutine) which accepts arguments (parameters) and which returns a result.

2−In mathematics a function f is a relation which assigns to each value x at plus a single value f(x).

Creating a function in an M-Files

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

```
function [$r_1, r_2, \cdots, r_n$]=name function ($arg_1, arg_2, \cdots, arg_n$)

    % the body of the function

...

r1 = ··· % the value returned for r1

r2 =··· % the value returned for r2


r2 ···

rn = ··· % the value returned for rn

end % the end is optional
```

Or: $r_1 \cdots r_n$ are the returned values, and $arg1 \cdots arg_n$ are the arguments.

```
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
```

Chapter 5

Polynomials

5.1 Handling Polynomial Functions in MATLAB

5.1.1 Representation of polynomials

Let be the following polynomial:$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots a_2 x^2 + a_1 x^1 + a_0$ where n is the degree of the polynomial and $a_i (i = 1, 2, 3, \cdots n)$ are the coefficients of the polynomial. This polynomial can be written as:$P(x) = ((\cdots(\cdots(a_n x^n + a_{n-1})x^{n-1} + a_{n-2})x \cdots + a_1)x^1 + a_0)$.

After factorization, we have:

$P(x) = a_n(x - r_1)(x - r_2)(x - r_3)\cdots(x - r_n)$ where $r_0, r_1, r_2 \cdots r_n$ are the roots of the polynomial.

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

5.1.2 Arithmetic of polynomials

MATLAB defines functions for elementary or advanced processing of polynomials: root polynomial, evaluation and differentiation, curve approximation, etc.

The roots function

Calculate the roots of a polynomial. By convention, MATLAB saves roots in a vector column.

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 +0.0000i

0.4061 + 1.2472i

0.4061 - 1.2472i
```

The three roots of this polynomial are given as a column vector.

Determining the coefficients of a polynomial from its roots

Contrary to to the root function, **poly** returns the coefficients of a polynomial from its roots.

Example 1:

```
>>poly(r)

ans =

1.0000   5.0000  -3.0000  10.0000
```

Example 1:

```
clc; clear all

a = [ 3 -5 ]

poly(a)

a =

3 -5

ans =
```

1 2 -15 (that is to say: $x^2 + 2x - 15$).

```
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
```

Example 3:

```
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 4:

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

```
clc; clear all

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

r=roots(P)

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 5:

```
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** :

```
>> Valprop=eig(A)

Valprop =

6

1
```

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
```

It is possible to evaluate a polynomial for a matrix X via the MATLAB **polyvalm** function. The matrix X must be square.

Exemple :

>> $p = [\ 1\ 0\ \ -2\ \ -5]$.

```
>>X = [2  4  5; -1  0  3; 7  1  5];

>>Y = polyvalm(p,X)

Y =

377    179    439 111    81    136 490    253    639
```

Multiplication of polynomials

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

Example 1:

$A(x) = -2x^3 + 5x^2 - 3x + 1 \qquad B(x) = x^4 - 3x^2 - 5x - 8$

The Convolution product A(x). B(x) is given by:

$A = [-2 \ \ 5 - 3 \ \ 1];$

$B = [1 - 3 - 5 - 8];$

h=conv(A,B)

h =

-2 11 -8 1 -28 19 -8

Example 2:

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

p1=[3 -2];

p2=[2 -1];

conv(p1 , p2)

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 A (x) and B(x):

The division of A(x) by B(x):

clc

$A = [3 - 2 \ \ 3 \ \ 5];$

$B = [2 - 12];$

$[q, r]$=deconv(A,B)

q =

1.5000 -0.2500

r =

0 0 -0.2500 5.5000

5.1.3 Advanced treatment

The polyder function:

This function calculates the derivative of polynomials or their multiplication or even their division, depending on the syntax used.

Example: $>> p = [10 - 2 - 5]$;

$>> q = polyder(p)$

$q = 3 \ \ 0 - 2$

$>> a = [1 \ 3 \ 5]; b = [2 \ 4 \ 6];$

c = polyder(a,b)

c = 8 30 56 38

>> [q,d] = polyder(a,b)

q = -2 -8 -2

d = 4 16 40 48 36

The Polyfit function:

Calculate the coefficients of the polynomial that best approximates the data in a mark.

p = polyfit(x,y,n),such that x and y are the vectors of the coordinates of x and y and n and the degree of polynomial result. $>> x = [1 \ 2 \ 3 \ 4 \ 5]; y = [5.5 \ \ 43.1 \ \ 128 \ \ 290.7 \ \ 498.4];$

p = polyfit(x,y,3)

p =

-0.1917 31.5821 -60.3262 35.3400

The Polyint function:

This function performs the integral of the polynomial given in parameters, analytically. She returns it representation of the integral of the polynomial P as input and taking into account the integration constant if we point it out to him

$>> polyint(p, k).l$

The instruction >>polyint(p) assume that the integration constant k is 0.

Chapter 6

Graphics in Matlab

6.1 Graphs and data visualization in MATLAB

Based on the principle that a picture is worth a thousand words, MATLAB offers a powerful visualization system that allows presentation and graphical display data in a way that is both efficient and easy. In this part of the course, we will present the basic principles essential for draw curves in MATLAB.

6.1.1 The plot function

The plot function can be used with vectors or matrices. it draws lines by connecting points of coordinates defined in its arguments, and it to several shapes:

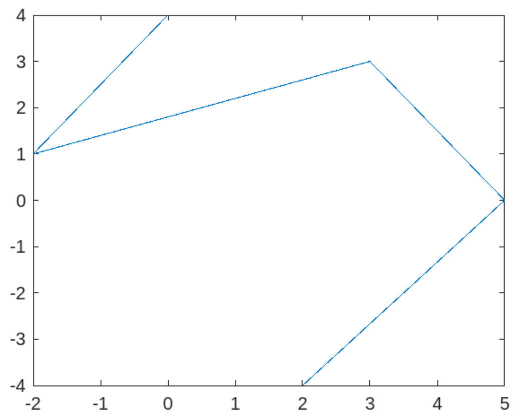
If it contains two vectors of the same size as arguments:

It considers the values of the first vector like the elements of the X axis (the abscissa), and the values of the second vector as the elements of the Y axis (the ordinates).

Example :

```
>> A = [2, 5, 3,-2,0]

A =

2   5   3   -2   0

>> B = [-4, 0, 3, 1,4]

B =

-4 0 3 1 4
```

```
>> plot(A,B)
```



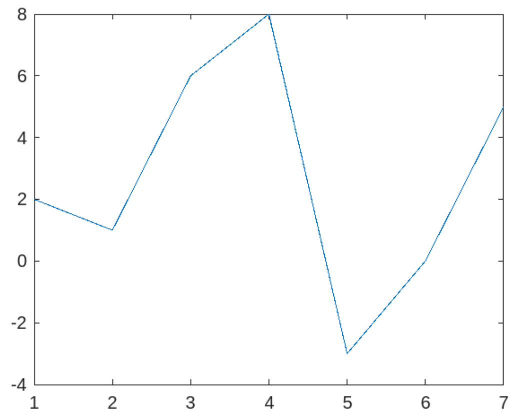
If it contains a single vector like argument :

she considers the values of the vector like the elements of the Y axis (the ordinates), and their positions relative we will define the X axis (the abscissa).

Example :

```
>> V = V = [2, 1, 6, 8,-3, 0,5]

V =

2   1   6   -8   -3   0   5
```

```
>> plot(V)
```

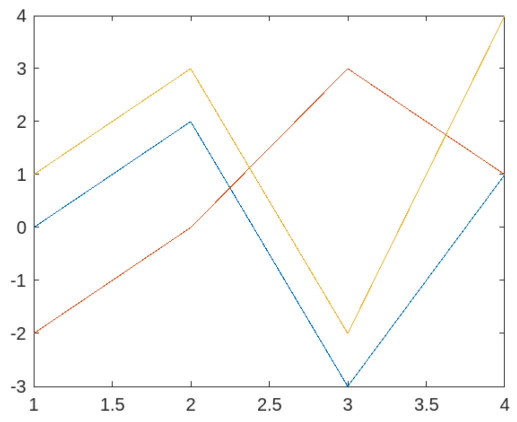


If it contains a single matrix as an argument:

it considers the values of each column as the elements of the Y axis, and their relative positions (the row number) as the values of the X axis. So, it will give several curves (one for each column).

Example :

```
>> M = [0 -2 1;2 0 3;-3 3 -2;1 1 4]

M =

0   -2   1

2   0   3

-3   3   -2

1   1   4

>> plot(M)
```



If it contains two matrices as arguments:

it considers the values of each column of the first matrix as the elements of the X axis, and the values of each column of the second matrix as the values of the Y axis.

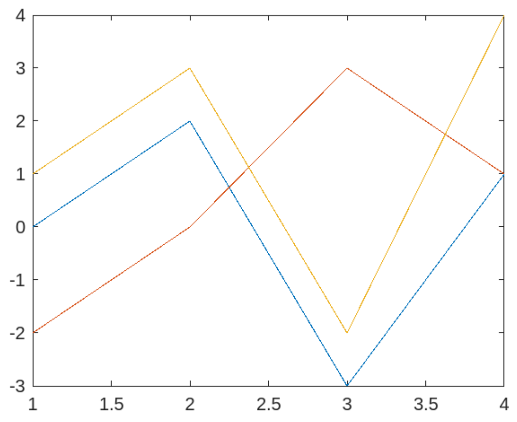
Example :

```
>> K = [1 1 1;2 2 2; 3 3 3;4 4 4]

K =

1   1   1

2   2   2

3   3   3

4   4   4

>> M = [0 -2 1;2 0 3;-3 3 -2;1 1 4]

M =

0   -2   1

2   0   3
```

-3 3 -2

1 1 4

>> plot(K,M)



It is obvious that the more the number of coordinates increases, the more precise the curve becomes. For example, to draw the curve of the function $y = sin(x) on [0, 2\pi]$ we can write:

The first figure

>> $x = 0 : pi/3 : 2 * pi;$

>> $y = sin(x);$

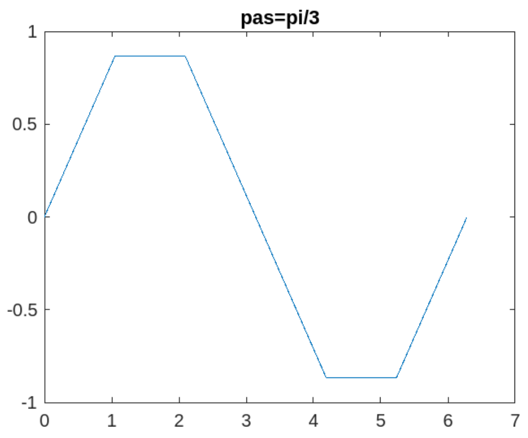
>> $plot(x, y)$

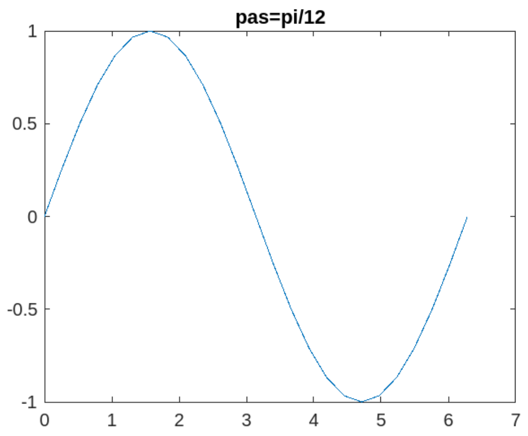
The second figure

>> $x = 0 : pi/12 : 2 * pi;$

>> $y = sin(x);$

>> $plot(x, y)$





6.1.2 Change the 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 and the type of line connecting the points. To do this, we add a new argument (which we can call a marker) of type string character to the **plot** function like this:

plot (x, y, 'marker')

The content of the marker is a combination of a set of special characters collected in the following tables:

Table of usual colors:

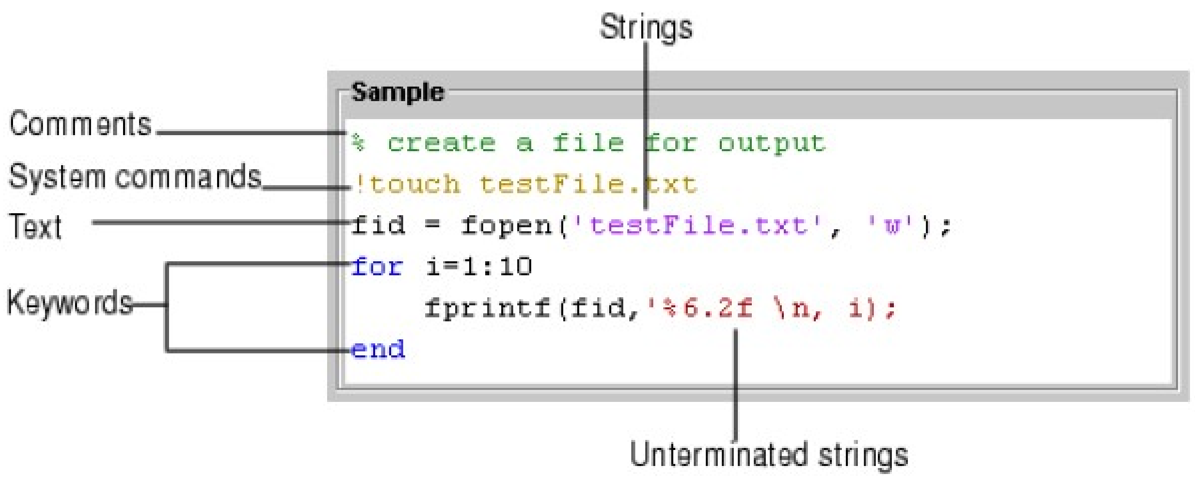
Short name	long name	its effect
y	yellow	curve in yellow
m	magenta	curve in bright purplish red
c	cyan	curve in between green and blue
r	red	curve in red
g	green	curve in green
b	blue	curve in blue
w	white	curve in white
k	black	curve in black

Marker table:

the character	its effect
$\cdot$	a point.
$\circ$	a circle ●
$\times$	the symbol $\times$
$+$	the $+$ symbol
$*$	star $*$
s	a square ■
d	a diamond ⧫
$\vee$	lower triangle ▼
$\wedge$	upper triangle ▲
$<$	left triangle ◀
$>$	right triangle ▶
p	pentagram $\star$
h	hexagram $*$

Curve Style

the character	its effect
–	online full ___
:	dotted ······
–.	dash dot –. – . – .
––	dashed – – – – –



```
% create a file for output
!touch testFile.txt
fid = fopen('testFile.txt', 'w');
for i=1:10
    fprintf(fid,'%6.2f \n, i);
end
```

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.

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

★ **Errors** Error text, including any hyperlinks, is colored red.

List of properties:

LineWidth: specify the width (at points) of the line.

MarkerEdgeColor: specifies the marker color or edge color for filled guides (circle, square, triangles, etc.).

MarkerFaceColor: specify the color of the face of the filled marks.

MarkerSize: specifies the size of the marker in point units.

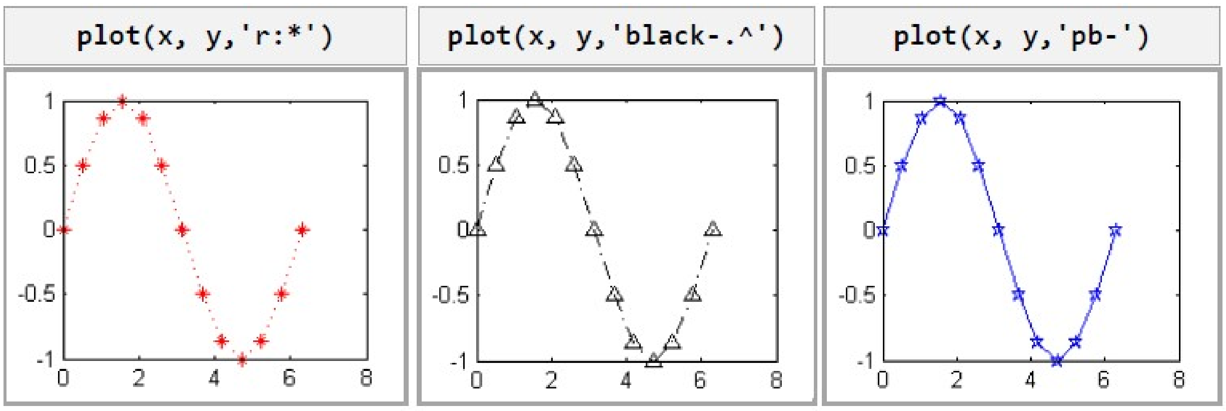
Example :

Let's try to draw the function $y = sin(x)$ for $x = [0...2\pi]$ with step $= \pi/6$.

$>> x = 0 : pi/6 : 2 * pi;$

$>> y = sin(x);$

By changing the marker we obtain different results, and here are some examples:



6.1.3 Annotating a figure

In a figure, it is preferable to put a textual description helping the user to understand the meaning of the axes and know the purpose or interest of the visualization concerned.

It is also very interesting to be able to report locations or points significant in a figure by a comment indicating their importance.

√ To give a title to a figure containing a curve we use the function **title** like this:

>> title('figure title')

√ To give a title for the vertical y axis, we use the function **ylabel** like this:

>> ylabel('This is the axis of ordered Y')

√ To give a title for the horizontal x-axis, we use the function **xlabel** like this:

>> xlabel('This is the axis of abscissa X')

√ To write text (a message) on the graphics window at a specified position by the x and y coordinates, we use the **text** function like this:

>> text(x, y, 'This point is important')

√ To put text on a position chosen manually by the mouse, we let's use the **gtext** function, which has the following syntax:

>> gtext('This point is chosen manually')

√ To set a grid, use the **grid** (or **grid on**) command. For remove it reuse the same **grid** (or **grid off**) command.

Example :

Let's draw the function: $y = -2x^3 + x^2 - 2x + 4$ for x varies from -4 to 4, with a step = 0.5.

>> x = 4:0.5:4;

>> y = $2 * x.^3 + x.^2 - 2 * x + 4$;

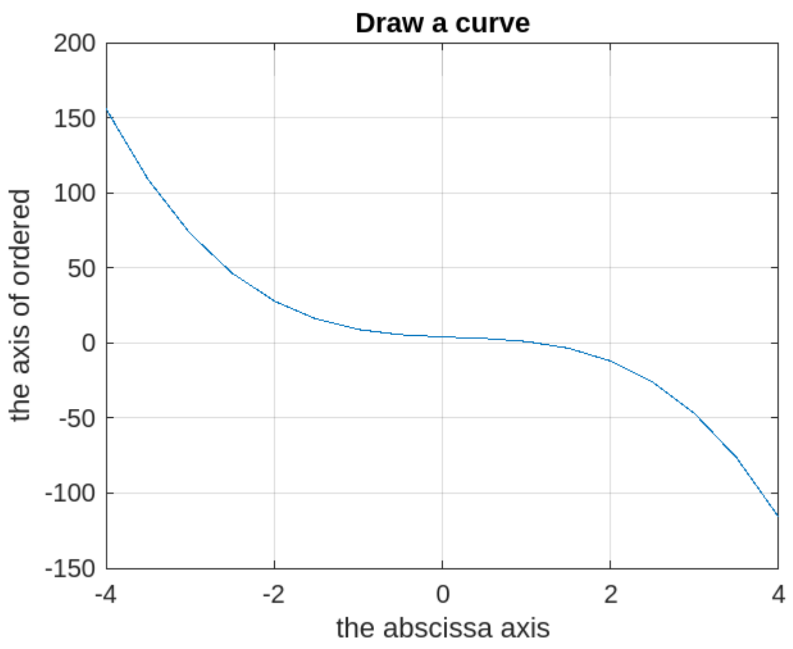
>> plot(x,y)

>> grid

>> title('Draw a curve')

>> xlabel('the abscissa axis')

```
>> ylabel('the axis of ordered')
```



6.1.4 Draw multiple curves in the same figure

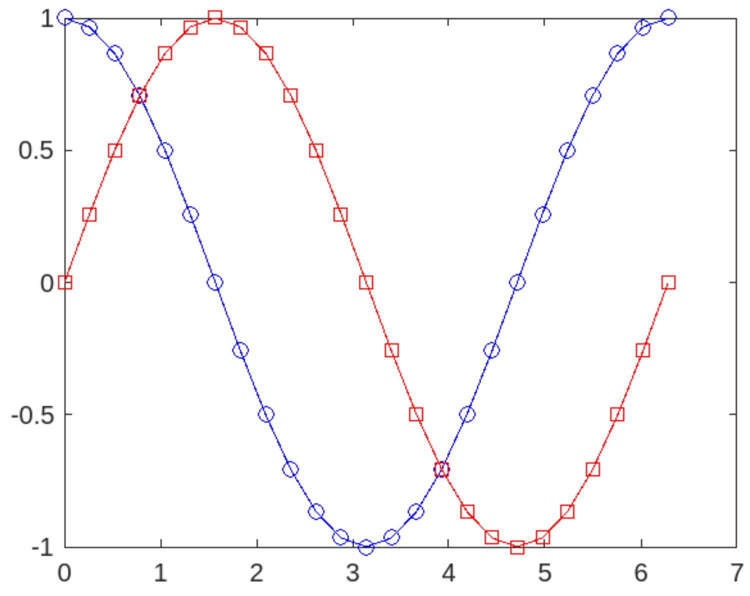
By default in MATLAB, each new drawing with the **plot** command erases the previous. To force a new curve to coexist with previous curves, There are at least three methods:

The hold command

The **hold** (or **hold on**) command activates the preserve old curves mode, which allows the display of several curves in the same figure. To cancel its effect he just rewrite **hold** (or **hold off**).

For example to draw the curve of the two functions cos(x) and sin(x) in the same figure, we can write:

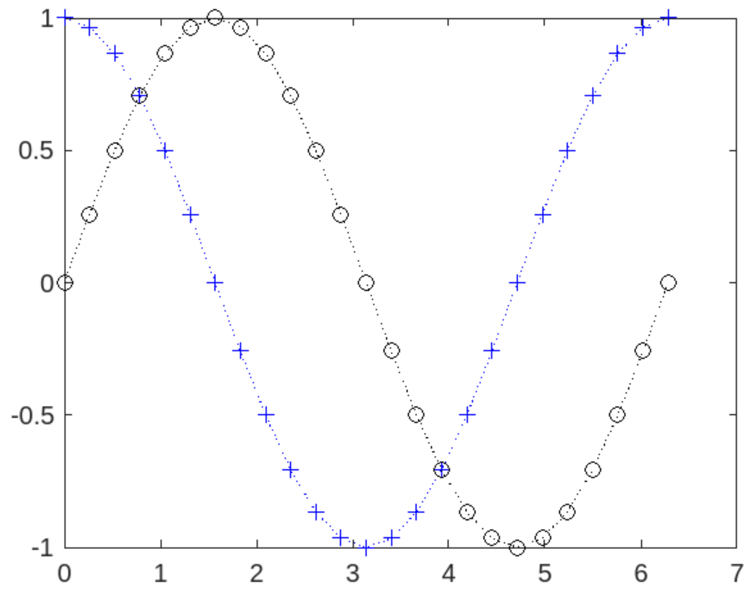
```
>> x=0:pi/12:2*pi;
>> y1=cos(x);
>> y2=sin(x);
>> plot(x,y1,'b-o')
>> hold on
>> plot(x,y2,'r-s')
```



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:

```
>> x=0:pi/12:2*pi;

>> y1=cos(x);

>> y2=sin(x);

>> plot(x,y1,'b:+',x,y2,'k:o')
```

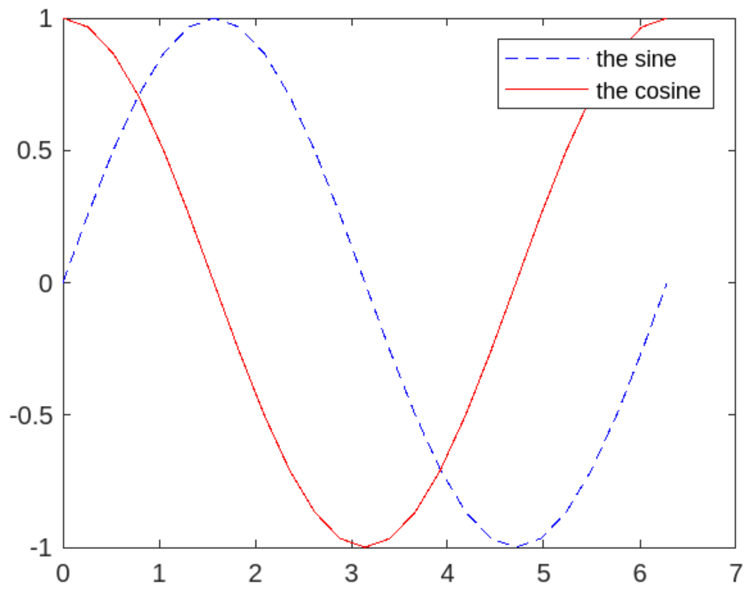


Use matrices as argument to plot function

In this case we obtain several curves automatically for each column (or sometimes the rows) of the matrix. This case has already been presented before.

After being able to put several curves in the same figure, it is possible to distinguish by putting a legend indicating the names of these curves. To do this, we use the legend function, as illustrated in the following example which draws the curves of the two functions sin(x) and cos(x):

```
>> x=0:pi/12:2*pi;
>> y1=sin(x);
>> y2=cos(x);
>> plot(x,y1,'b–',x,y2,'-r')
>> legend('the sine','the cosine')
```



It is possible to move the **legend** (which is located by default in the upper right corner) by using the mouse with drag and drop.

6.1.5 Manipulating the axes of a figure

MATLAB calculates the limits (minimum and maximum) of the X and Y axes by default and automatically choose the appropriate partitioning. But it is possible to control the appearance of the axes via the **axis** command.

To define the limits of the axes it is possible to use this command with the following syntax:

axis ([xmin xmax ymin ymax]) Or axis ([xmin,xmax,ymin,ymax])

- ♣ **xmin** and **xmax** define the minimum and maximum for the x-axis.
- ♣ **ymin** and **ymax** define the minimum and maximum for the y axis.

To return to the default display mode, we write the command: axis auto

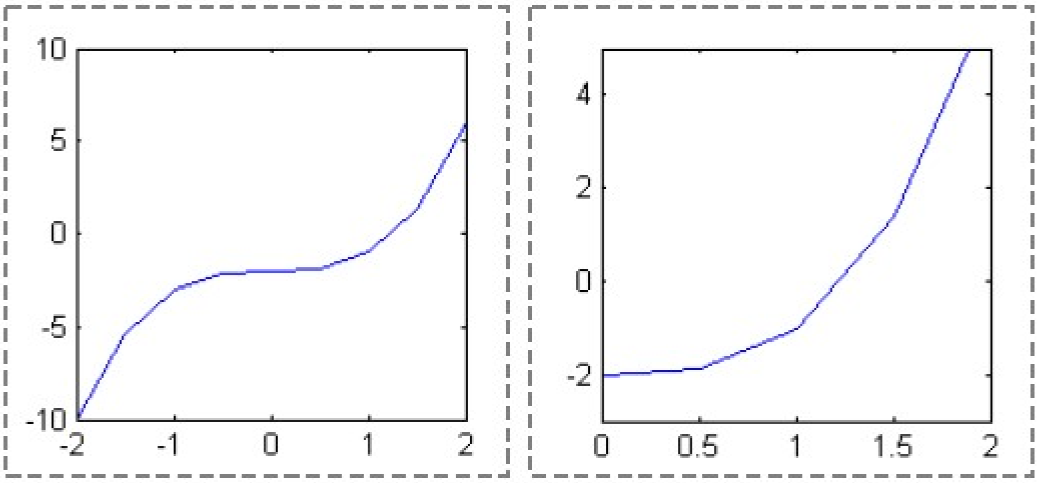
Example :

$f(x) = x^3 - 2$

>> x = -2:0.5:2;

>> y = $x.^3 - 2$;

>> plot(x,y)



```
>> axis auto
```

```
>> axis([0,2,-3,5])
```

Among the other options of the axis command, we present the following:

- ► To make the size of both axes the same (the size not the partitioning, we use the **axis square** command.

  It is so named because it gives the appearance of axes like a square.

- ► To make the partitioning of the two axes identical we use the command **axis equal**.

- ► To return to the default display and cancel the changes we use the **auto axis** command.

- To make the axes invisible we use the **axis off** command. To make them visible again we use the **axis on** command.

Exercise :

• Draw the graph of the function $2x^2 * sin(-2x) + x$ in $[-pi \;\; pi]$

function y=foc(x)

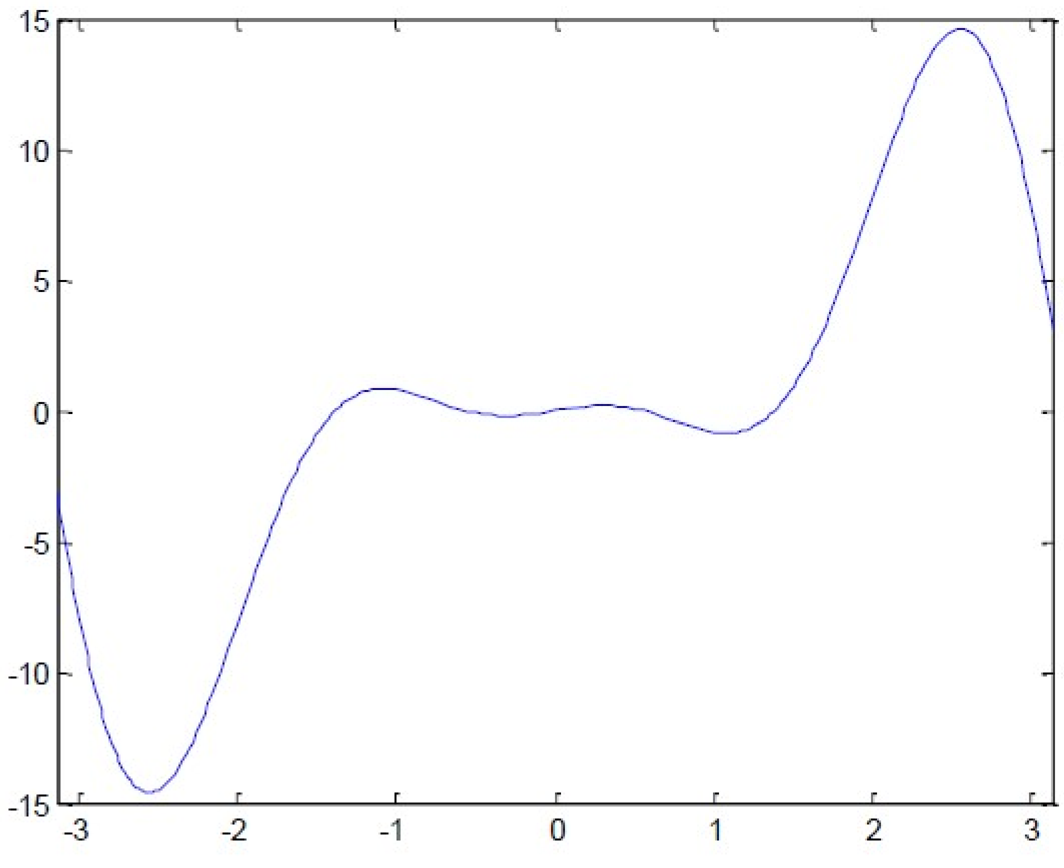
$y =' 2 * x\hat{}2 * sin(-2 * x) + x';$

end

ans =

$2 * x\hat{}2 * sin(-2 * x) + x$

$>> fplot('2 * x\hat{}2 * sin(-2 * x) + x', [-pi \;\; pi])$

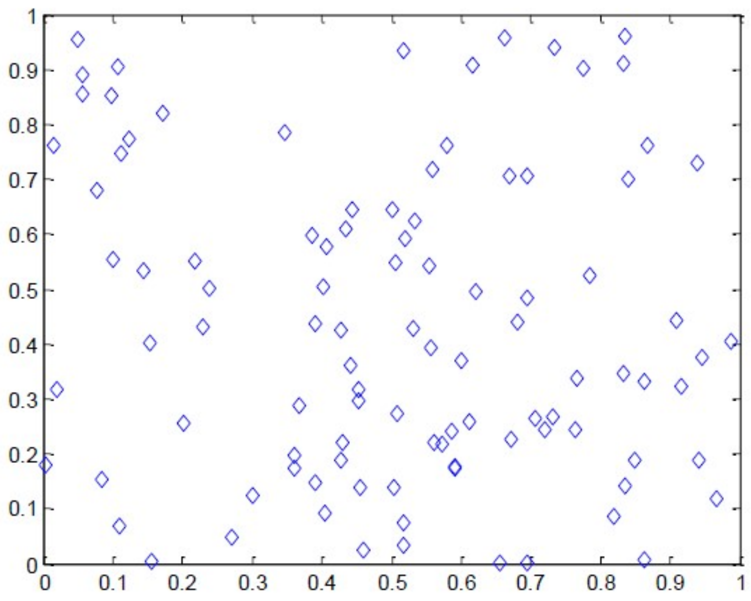


• N=100;

x = rand(1,N);

y = rand(1,N);

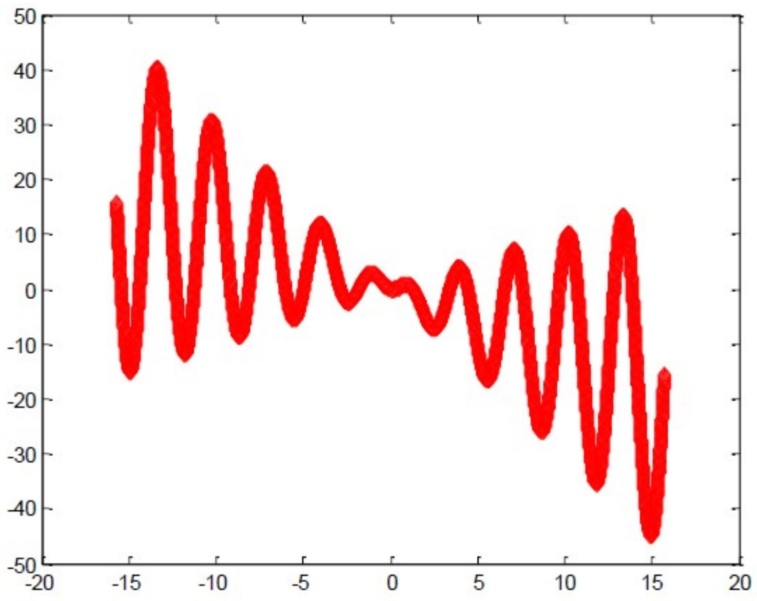
plot(x,y,'bd')



• $X = -10 : 0.5 : 10;$

$Y = x.\hat{}4 - x.\hat{}2;$

$plot(x, y,' dr')$

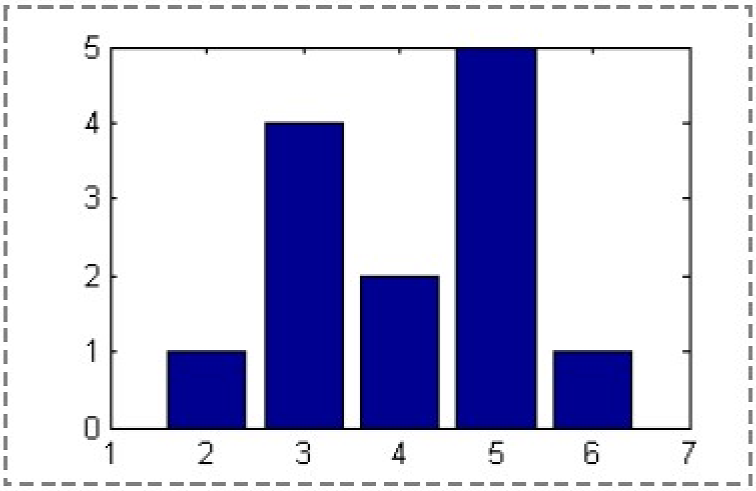


6.2 Other types of charts

The MATLAB language does not only allow the display of points to plot curves, but it also offers the possibility of drawing bar graphs and histograms. To draw a bar graph we use the **bar** function which has the same principle of operation than the **plot** function.

Example :

```
>> X=[2,3,5,4,6];
>> Y=[1,4,5,2,1];
>> bar(X,Y)
```



It is possible to modify the appearance of the sticks, and there is the **barh** function which draws the sticks horizontally, and the **bar3** function which adds a 3D effect. Among the very interesting drawing functions not presented (due to lack of space) we can find: **hist, stairs, stem, pie, pie3, ...**etc. (which we encourage you to to study).

We also point out that MATLAB allows the use of a coordinate system other than the Cartesian system like the polar coordinate system (for more detail look for the **compass, polar** and **rose** functions).

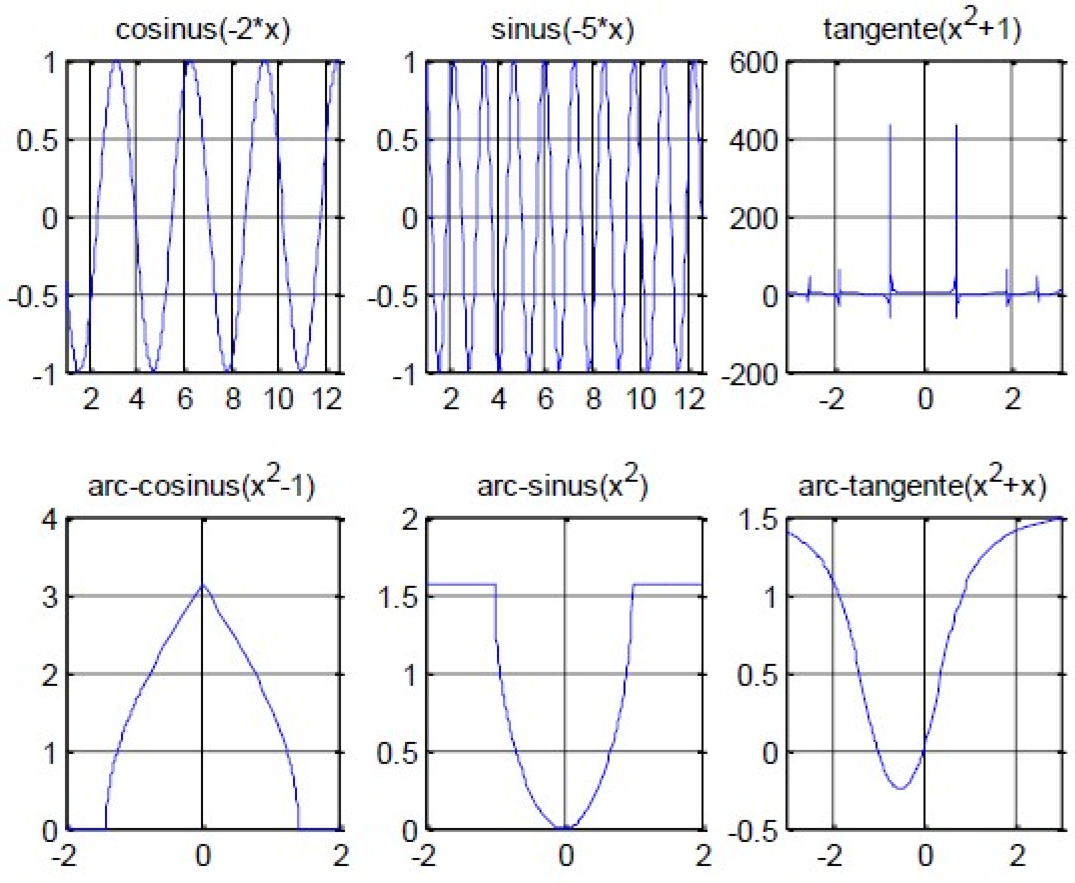
The subplot command

Example

```
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



6.3 3D graphic

6.3.1 Draw the level lines of a function of 2 variables

Function

the function	meaning
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
plot3	trace point by point a 3D graph
contour	(see example)
meshz	(see example)
meshgrid	(see example)
mesh	(see example)
meshc	(see example)

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 xmin ¡ xmax and ymin ¡ ya ¡ ymax we proceed as follows:

creation of a mesh, with a mesh of length h, of the domain [xmin , xmax] x [ymin , ymax] using the command **meshgrid**,

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

- 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).

Example 1

Thus to draw the contour lines of the function $f(x, y) = yxe^{(x^2y^2)}$ on the domain $[-1, 1]x[-1, 1]$ 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 = $Y * X. * exp(-X.\hat{}2 - Y.\hat{}2)$;

contour(X,Y,Z) , grid on

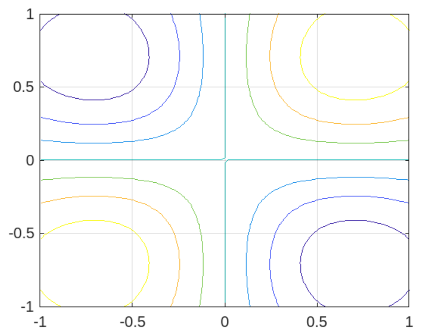
You can also write a user function g.m,

```
function x3 = g(x1,x2)

x3 = x2* x1.*exp(-x1.^2-x2.^2);

end
```

and execute

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

>> Z = g(X,Y);

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

grid on



Example 2

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')

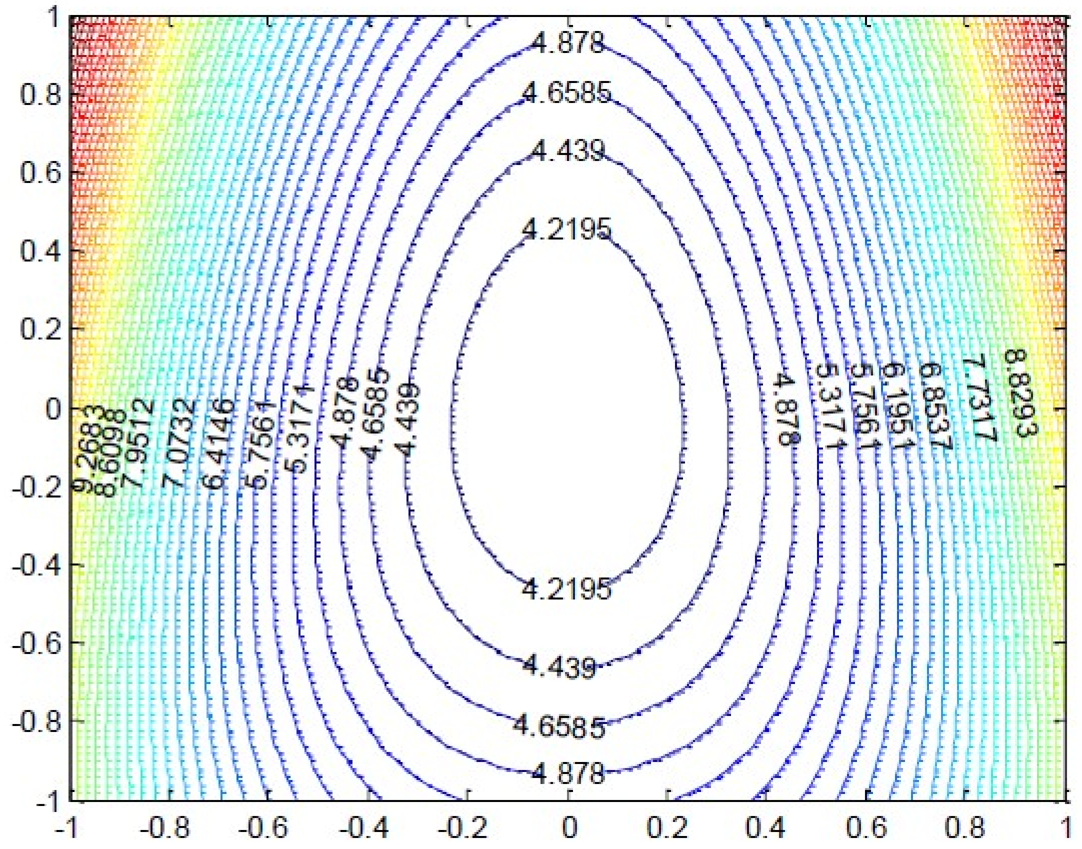
You can then use the mouse to select the level lines for which you want to display the value. So to draw 40 lines of level of function $z = (x^2 + 2)^2 + (x^2 + y)^2$on the domain $[-1, 1]x[-1, 1]$.

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

$Z = (X.\hat{}2 + 2).\hat{}2 + (X.\hat{}2 + Y).\hat{}2;$

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

clabel(C,h,'manual')



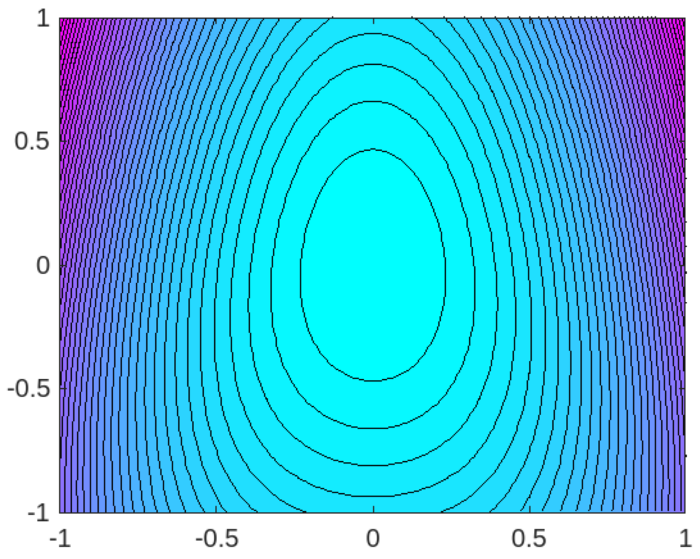
Example 3:

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.

$[X, Y] = meshgrid(-1 : 0.02 : 1, -1 : 0.02 : 1);$ $Z = (X.\hat{}2 + 2).\hat{}2 + (X.\hat{}2 + Y).\hat{}2;$ contourf(X,Y,Z,40);

colormap(cool);



6.3.2 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 xmin< x < xmax and ymin< y <ymax we proceed as follows:

- Creation of a mesh, mesh of length H, in the field [xmin, xmax] x [ymin, ymax] thanks to the Meshgrid command, $[X, Y] = meshgrid(x_{min} : h : x_{max}, y_{min} : h : y_{max})$.

- 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,mesh(X,Y,Z).

Example

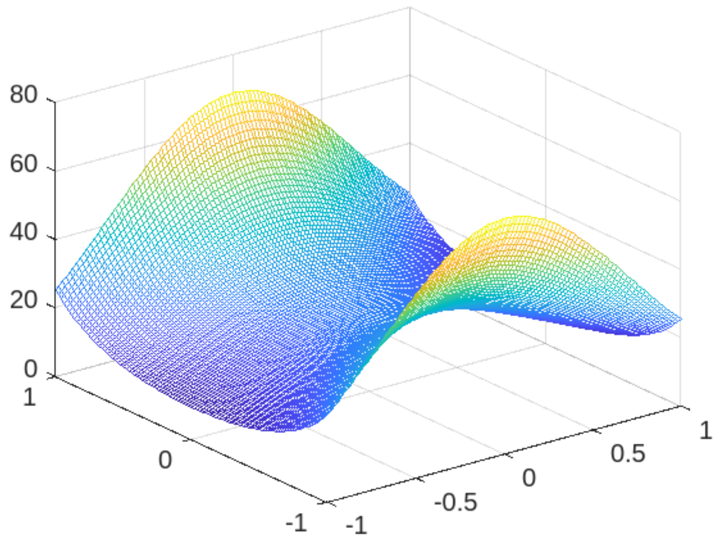
$z = 2xe^{(x^2+y^2)}$ on the domain $[-1, 1] \times [-1, 1]$ with a mesh of mesh length h=0.02, we execute:

clc; clear all

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

$Z = 2 * X * Y. * exp(-X.\hat{}2 + Y.\hat{}2 - 1)$;

mesh(X,Y,Z)



6.3.3 Represent a parameterized surface

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

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

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$umin < u < umax$and $vmin < v < vmax$ we proceed as follows:

- creating a mesh, h-length mesh, domain $[xmin, xmax]x[ymin, ymax]$ using the command meshgrid, $[U, V] = meshgrid(umin : h : umax, vmin : h : vmax)$. 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:

$$\begin{cases} x &= vcos(u) \quad , \\ y &= v(u) \quad , \\ z &= 2(v) \quad , \end{cases}$$

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*$U.\hat{}2 + V.\hat{}2$;

surf(X,Y,Z)

If the function is defined as a user function in the G1.m file,

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

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

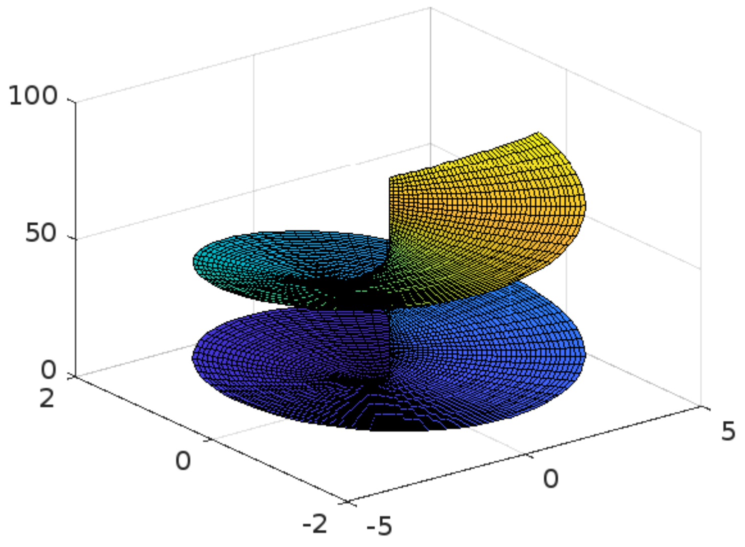
x2 = v.*sin(2*u);

x3 = 2*u;

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

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

>> surf(X,Y,Z)



Example 2:

$$Either\ parametric\ equation : \begin{cases} x & = cos(t) \quad , \\ y & = 2tsin(t) \quad , \\ z & = 5(t) + 1 \quad , \end{cases}$$

clc; clear all

t = 0 : pi/200 : 10*pi;

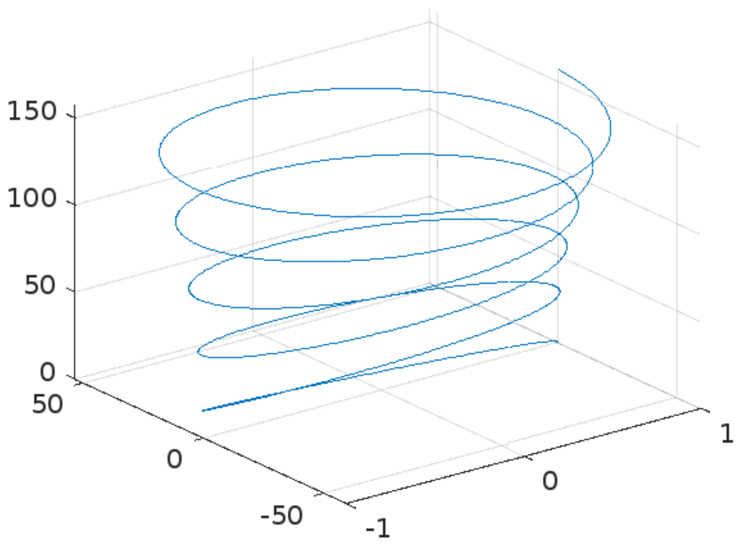
x = t.*cos(t);

y = 2*t.*sin(t);

z = 5*t+1;

plot3(cos(t) , 2*t.*sin(t) , 5*t),

grid on



Example 3:

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

clc; clear all

x = -80 : 80

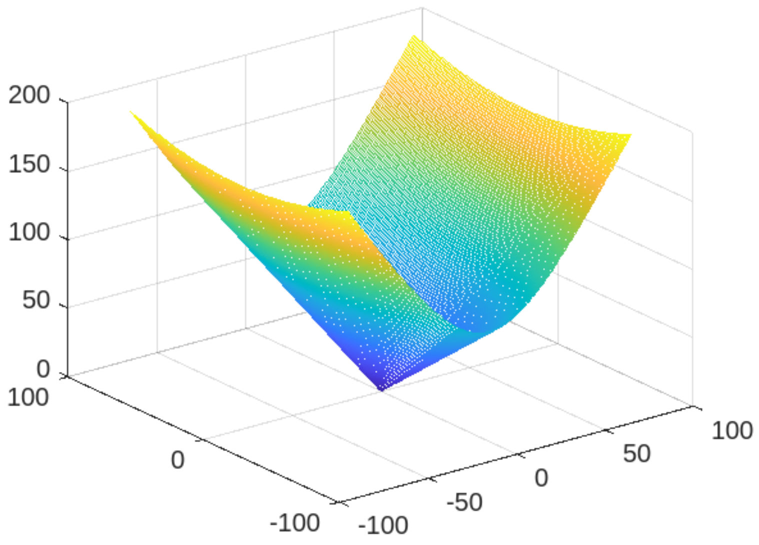
y = -80 : 80

$[X, Y]$ = meshgrid(x , y);

Z = sqrt (5.*X.^2 + Y.^2);

mesh (X , Y , Z)

grid on



Exercise 1:

Type the following code. Deduce the purpose of each function in the example.

$x = -pi : 0.1 : 3 * pi; y = x. * sin(x);$

$plot(x, y)$

clf

$plot(x, y, x, 2 * y)$

$plot(x, y,' r-', x, 2 * y,' g+')$

$xlabel('x')$

$ylabel('y')$

title(['the graph of the function sin(x) over the interval num2str(x(1)) ', num2str(x(end))])

Exercise 2:

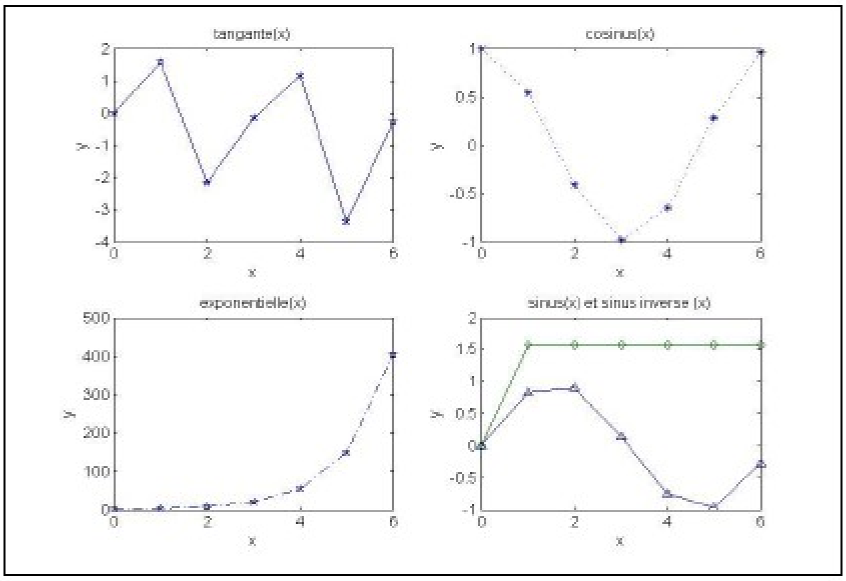
Consider the functions: $f(x) = sin(x2) + 4$ et $P(x) = 2x^3 + x^2 3$

1)Plot the curve of the function $f(x)$ while varying x from 0 to 2π with an increment of $\pi/12$. Use a dashed green line style with diamond-shaped points.

2)Plot the curve of the function P(x) while varying x from -5 to 5 with an increment of 0.2. Use a blue dashed line with square-shaped points.

Exercise 3:

Write the set of MATLAB instructions to plot the following curves in the same Figure:



Chapter 7

Symbolic calculation

7.1 Calling the symbolic toolbox

Symbolic Math Toolbox provides functions for solving, plotting, and manipulating symbolic math equations. You can create, run, and share symbolic math code. In the MATLAB Live Editor, you can get next-step suggestions for symbolic workflows. The toolbox provides functions in common mathematical areas such as calculus, linear algebra, algebraic and differential equations, equation simplification, and equation manipulation.

Symbolic Math Toolbox lets you analytically perform differentiation, integration, simplification, transforms, and equation solving. You can perform dimensional computations and convert between units. Your computations can be performed either analytically or using variable-precision arithmetic, with the results displayed in mathematical typeset.

You can share your symbolic work with other MATLAB users as live scripts or convert them to HTML, Word, LaTeX, or PDF documents. You can generate MATLAB functions, Simulink function blocks, and Simscape equations directly from symbolic expressions.

7.1.1 Familiarization and handling of formal computing with Matlab

- Computer algebra software is software that facilitates symbolic calculation whose main part is based on the manipulation of mathematical expressions under their symbolic forms.

- Computer algebra is the field of mathematics and computer science that focuses on to algorithms operating on objects of a mathematical nature through finite and exact representations.

- ▶ Concretely solving problems from various applications often requires both symbolic and digital treatments.
- ▶ Matlab offers a range of procedures and processing for digital data, but also for symbolic expressions.
- ▶ Matlab has very advanced computer algebra functions, in addition to its environment offers symbolic calculation tools and at the same time provides powerful graphics and calculation.
- ▶ Matlab can be used to carry out calculations focused on the manipulation of expressions formal and obtain mathematical expressions as results.
- ▶ Calculation approaches with symbolic variables to solve equations algebraic or differential,...

7.2 Development and implementation of an expression

7.2.1 Symbolic expressions

⧫ The "syms" function should be used to create symbolic objects in Matlab

Declaration of variables

Exemple:

syms a b c x;

Exemple:

sym a;

sym b;

sym c;

sym d;

Declaration of variables with constraints

Exemple:

syms x theta real ;

It is equivalent to: x = sym('x','real');

theta = sym('theta','real');

7.2.2 Development of expressions

collect command

Example:

>> syms x y;

>> $E = (x-2)\hat{}2 + (y+3)\hat{}2;$

>> $collect(E)$

$ans =$

$x\hat{}2 - 4*x + (y+3)\hat{}2 + 4$

>> $collect(E, y)$

$ans =$

$y\hat{}2 + 6*y + (x-2)\hat{}2 + 9$

Expand command

La fonction expand dveloppe une expression

Example 1:

>> $expand(E)$

$ans =$

$x\hat{}2 - 4*x + y\hat{}2 + 6*y + 13$

>> $expand((x+y)\hat{}3)$

$ans =$

$x\hat{}3 + 3*x\hat{}2*y + 3*x*y\hat{}2 + y\hat{}3$ >> $symx;$

>> $E2 = (x+3)*(x-7)\hat{}2;$

>> $expand(E2)$

$ans =$

$x\hat{}3 - 11*x\hat{}2 + 7*x + 147$

>> $expand(sin(x+y))$

$ans =$

$cos(x)*sin(y) + cos(y)*sin(x)$

```
>> expand(sin(x - y))

ans =

cos(y) * sin(x) - cos(x) * sin(y)

>> expand(cos(2 * x))

ans =

cos(x)^2 - sin(x)^2

>> expand(sin(2 * x))

ans =

2 * cos(x) * sin(x)

>> expand(tan(2 * x))

ans =

- (2 * tan(x))/(tan(x)^2 - 1)
```

Example 2:

```
>> expand(3 * cos(3 * x) - sin(2 * x))

ans =

3 * cos(x)^3 - 9 * cos(x) * sin(x)^2 - 2 * cos(x) * sin(x)

>> expand((2 * x - 7) * (x - 1) * (x + 4))

ans =

2 * x^3 - x^2 - 29 * x + 28

>> expand((x - 1) * (x + 2) * (x - 3) * (x - 5))

ans =

x^4 - 7 * x^3 + 5 * x^2 + 31 * x - 30
```

Example 3:

```
>> syms a;

>> syms b;

>> expand(cos(a + b))

ans =
```

$cos(a) * cos(b) - sin(a) * sin(b)$

$>> expand(cos(a - b))$

$ans =$

$sin(a) * sin(b) + cos(a) * cos(b)$

$>> expand(sin(a - b))$

$ans =$

$cos(b) * sin(a) - cos(a) * sin(b)$

$>> expand(sin(a + b))$

$ans =$

$cos(a) * sin(b) + cos(b) * sin(a)$

Factor commands

Integers can be charged, although the form of the result depends on whether the entry is numeric or symbolic.

For a numeric (integer) input, the result is a list of factors, repeated according to multiplicity.

For a symbolic input, the output is a symbolic expression. An important command for working with symbolic expressions is simplify, which tries to reduce an expression to a simpler expression equal to the original.

Example 1:

$>> factor(x\hat{}2 - 9)$

$ans =$

$(x - 3) * (x + 3)$

$>> factor(2 * x\hat{}2 + 4 * x)$

$ans =$

$2 * x * (x + 2)$

$>> factor(x\hat{}3 - 4 * x\hat{}2 - 28 * x - 32)$

$ans =$

$(x - 8) * (x + 2)\hat{}2$

$>> factor((cos(x))\hat{}3 - cos(x) * sin(x)\hat{}2)$

$ans =$

$cos(x)*(cos(x)-sin(x))*(cos(x)+sin(x))$

Example 2:

$>> factor(x\hat{}6-27*x\hat{}5-152*x\hat{}4+5328*x\hat{}3-15392*x\hat{}2-83088*x+262080)$

$ans =$

$(x-3)*(x-26)*(x-6)*(x+4)*(x+14)*(x-10)$

Simplify command

Example 1:

$>> factor((cos(x))\hat{}3-cos(x)*sin(x)\hat{}2)$

$ans =$

$cos(x)*(cos(x)-sin(x))*(cos(x)+sin(x))$

$>> simplify(sin(x)/(1+cos(x))+(1+cos(x))/sin(x))$

$ans =$

$2/sin(x)$

$>> simplify(-cos(x)\hat{}2*sin(x)+sin(x)\hat{}3+sin(x))$

$ans =$

$2*sin(x)\hat{}3$

Addition, Subtraction, Multiplication, Division

Example 1:

$>> syms\ xy;$

$>> E1 = x\hat{}6+5*x\hat{}2;$

$>> E2 = y\hat{}3-2;$

$>> E3 = -3*x\hat{}4-2*x\hat{}2-1;$

$>> S1 = E1+E3$

$S1 =$

$x\hat{}6-3*x\hat{}4+3*x\hat{}2-1$

$>> expand(S1)$

$ans =$

```
x^6 - 3*x^4 + 3*x^2 - 1

>> simplify(S1)

ans =

(x^2 - 1)^3

>> S2 = E3/E1

S2 =

- (3*x^4 + 2*x^2 + 1)/(x^6 + 5*x^2)

>> S3 = E1 - E3

S3 =

x^6 + 3*x^4 + 7*x^2 + 1

>> S4 = E2 - y^3 + y^2 - 2*y + 3

S4 =

y^2 - 2*y + 1

>> simplify(S4)

ans =

(y - 1)^2

>> simplify(S1*(x + 1)^2)

ans =

(x - 1)^3*(x + 1)^5
```

7.2.3 Solving an equation

solve command

Example 1:

```
syms x;

Equation = x^2 + 2*x - 8;

Solution = solve(Equation, x);

S1 = Solution(1);

S2 = Solution(2);
```

disp('the solutions of this equation are the two roots given below: ');

disp(S1);

disp(S2);

% After execution

the solutions of this equation are the two roots given below:

−4

2

Example 2:

$syms\ x\ real;$

$S1 = solve(4 * cos(x)\hat{}4 - 2 * sin(x)\hat{}2, x)$

$S1 =$

$pi/4$

$>> syms\ x\ real;$

$>>$

$solve('cos(x)\hat{}2 - 3 * sin(3 * x) - 3 * cos(4 * x)', x)ans =$

$pi/2$

$syms\ a\ b\ c\ x;$

$S2 = solve('a * x\hat{}2 + b * x + c')$

$S2 =$

$-(b + (b\hat{}2 - 4 * a * c)\hat{}(1/2))/(2 * a)$

$-(b - (b\hat{}2 - 4 * a * c)\hat{}(1/2))/(2 * a)$

Example 3:

$>> solve('x > 3', x)$

$ans =$

$(3, Inf)$

$>> solve('x >= 3', x)$

```
ans =

[3, Inf)

>> solve('x^2 > 4', x)

ans =

Dom : Interval(2, Inf)

Dom : Interval(-Inf, -2)

>> solve('x^2 - x - 6 > 0', x)

ans =

Dom : Interval(-Inf, -2)

Dom : Interval(3, Inf)
```

Example 4:

```
>> solve('x^2 - 36 >= 0', x)

ans =

Dom : Interval([6, Inf)

Dom : Interval(-Inf, [-6])

>> solve('2 * exp(2 * x) - 5 * exp(x) = 63')

ans =

log(7)
```

dsolve command

Example 1:

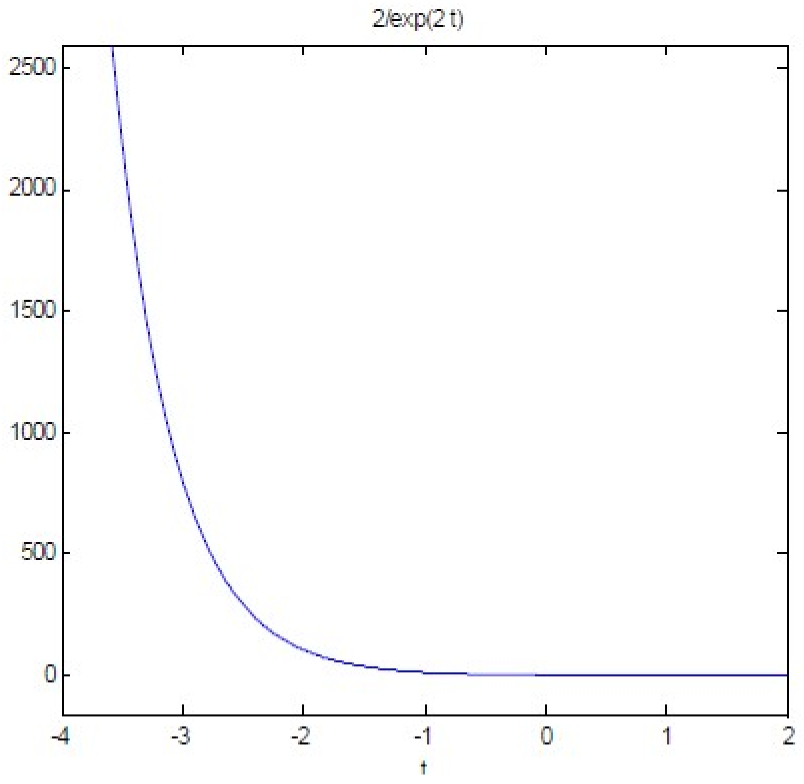
```
>> syms y;

>> dsolve('Dy + 2 * y = 0')

ans =

C2/exp(2 * t)

Solution1 = dsolve('Dy + 2 * y = 0', 'y(0) = 2')

ans =

2/exp(2 * t)
```

$ezplot(Solution1, [-4\ 2])$

$xlabel('t')$

$ylabel('y(t)')$

$gridon$



Example 2:

$>> syms\ y\ t;$

$>> dsolve('2 * Dy - y = -t\hat{}2 + 5 * t')$

$ans =$

$t\hat{}2 - t + C2 * exp(t/2)2$

$Solution2 = dsolve('2 * Dy - y = -t\hat{}2 + 5 * t','\ y(-1) = 5')$

$Solution2 =$

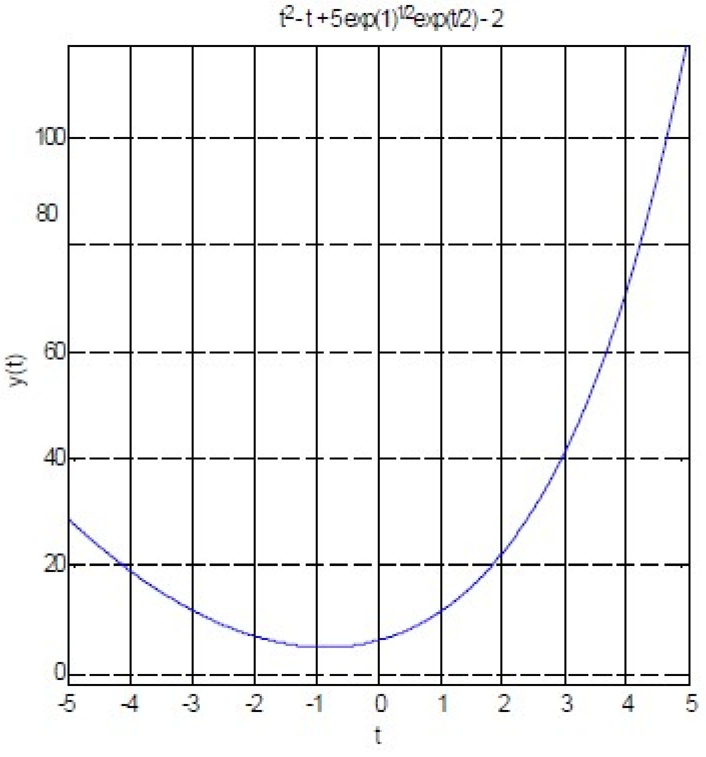
$t\hat{}2 - t + 5 * exp(1)\hat{}(1/2) * exp(t/2) - 2$

$ezplot(Solution2, [-5\ 5])$

$xlabel('t')$

$ylabel('y(t)')$

gridon



Example 3:

$syms\ y;$

$Solution3 = dsolve('D2y + 0.5 * y = 0',' y(0) = 1',' Dy(0) = 0')$

$Solution3 =$

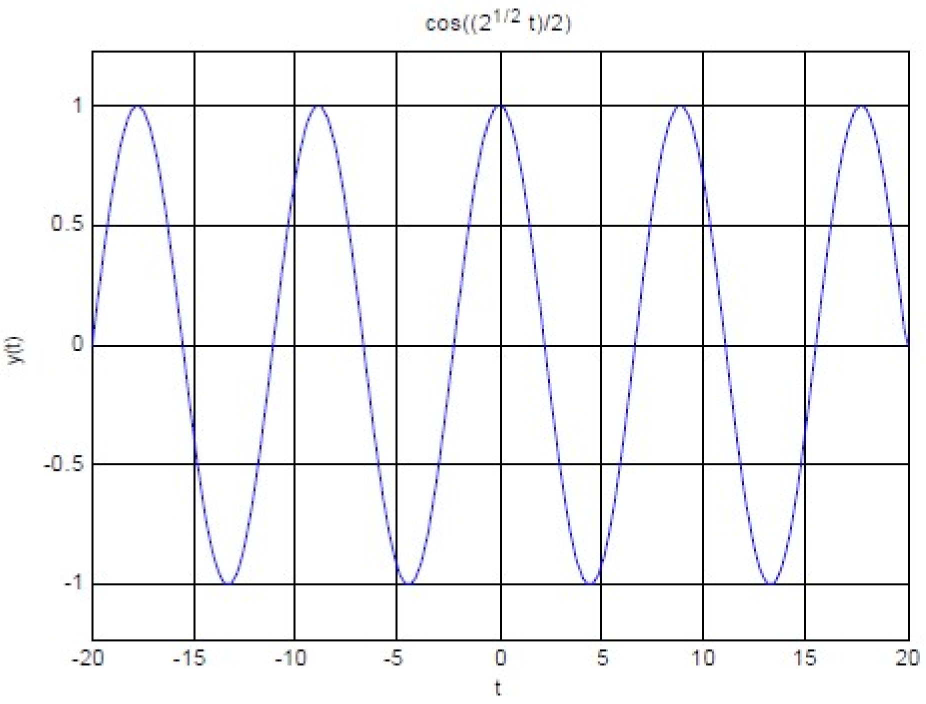
$cos((2\hat{}(1/2) * t)/2)$

$ezplot(Solution3, [-20\ 20])$

$xlabel('t')$

$ylabel('y(t)')$

gridon



7.2.4 Evaluating symbolic expressions

subs & class & double commands

Example:

$>> syms\ x;$

$>> E1 = x\char`^2 - 2 * x\char`^2 + 7;$

$>> V1 = subs(E1, x, 1)$

$V1 =$

6

$>> V2 = subs(E1, x, -3)$

$V2 =$

-2

$>> class(V1)$

$ans =$

double

$>> H = double(V1)$

$H =$

6

7.2.5 Polynomial conversion

poly2sym & sym2poly commands

Example:

$>> syms\ x;$

$>> poly2sym([1 - 5\ 7\ 13])$

$ans =$

$x^3 - 5 * x\hat{}2 + 7 * x + 13$

$>> poly2sym([2\ 5 - 1])$

$ans =$

$2 * x\hat{}2 + 5 * x - 1$

$>> sym2poly(x\hat{}3 - 3 * x\hat{}2 + 11 * x - 9)$

$ans =$

$1 - 3\ 11 - 9$

$>> sym2poly(-8 * x\hat{}3 + 2 * x - 5)$

$ans =$

$-8\ 0\ 2 - 5$

7.3 Derivative and primitive of a function

7.3.1 Derivatives of a function

diff command

We can calculate higher derivatives of an expression by the function **diff** **Example 1:**

$>> syms\ x;$

$>> diff(2 * x\hat{}4 + 3 * x\hat{}3 - 5 * x\hat{}2 + 21, 1)\%First\ derivative$

$ans =$

$8 * x^3 + 9 * x\hat{}2 - 10 * x$

$>> diff(2 * x\hat{}4 + 3 * x\hat{}3 - 5 * x\hat{}2 + 21, 2)\%Second\ derivative$

$ans =$

$24 * x\hat{}2 + 18 * x10$

$>> syms\ x\ t;$

$>> f = sin(2 * t) + t\hat{}4;$

$>> df = diff(f, t)$

$df =$

$2 * cos(2 * t) + 4 * t\hat{}3$

$>> d6f = diff(f, t, 6)$

$d6f =$

$(-64) * sin(2 * t)$

Example 2:

$>> syms\ x\ \ y;$

$diff(cos(2 * x * y), x)$

$ans =$

$(-2) * y * sin(2 * x * y)$

$>> diff(cos(2 * x * y), y)$

$ans =$

$(-2) * x * sin(2 * x * y)$

$>> diff(x\hat{}3 - 2 * y + 5, x, 1)$

$ans =$

$3 * x\hat{}2$

$>> diff(x\hat{}3 - 2 * y + 5, y, 1)$

$ans =$

-2

$>> diff(x\hat{}3 - 2 * y + 5, x, 3)$

$ans =$

6

7.3.2 Intgration

int command

MATLAB can calculate indefinite and definite integrals. The command to calculate an integral indefinite (antiderivative) is exactly analogous to that of calculating a derivative

To calculate a definite integral, we need to specify the limits of the integration.

Remark 1:

MATLAB does not add the "integration constant". It simply returns an antiderivative (if possible). If the integrand is too complicated, MATLAB just returns the unevaluated integral, and prints a warning message

Remark 2:

MATLAB has commands for estimating the value of a definite integral that cannot be calculated analytically.

Example 1:

```
>> syms x;
>> int(sin(x)^2,0,pi/2)
ans =
pi/4
>> int(3/8 - 1/2*cos(2*x) + 1/8*cos(4*x),0,pi/2)
ans =
(3*pi)/16
>> int(1/(1 + x^2),0,1)
ans =
pi/4
>> int(1/(1 + x^2)^2,0,1)
ans =
pi/8 + 1/4
```

Example 2:

$>> syms\ x;$

$>> int(sin(x) * cos(x)/(1 + sin(x)))$

$ans =$

$sin(x) - log(sin(x) + 1)$

$>> int(sin(x) * cos(x)/(1 + sin(x)), 0, pi/2)$

$ans =$

$1 - log(2)$

$>> int(x\hat{\ }2 - 6 * x + 3)$

$ans = (x * (x\hat{\ }2 - 9 * x + 9))/3$

$>> int(x\hat{\ }2 - 6 * x + 3, 1, 3)$

$ans =$

$-28/3$

7.4 Calculation of the limited development of a function

7.5 Limited development

7.5.1 Taylor order

Example:

$>> syms\ x;$

$>> taylor(cos(x), 7) \quad \%Development\ at\ order 7$

$ans =$

$-x\hat{\ }6/720 + x\hat{\ }4/24 - x\hat{\ }2/2 + 1$

$>> taylor(sin(x)/log(1 + x), 8)$

$ans =$

$(1027 * x\hat{\ }7)/120960 - (31 * x\hat{\ }6)/2880 + (23 * x\hat{\ }5)/1440 - x\hat{\ }4/240 - x\hat{\ }3/24 - x\hat{\ }2/4 + x/2 + 1$

$>> taylor((1/(1 - x)) - tan(x), 6)$

$ans =$

$(13 * x\hat{\ }5)/15 + x\hat{\ }4 + (2 * x\hat{\ }3)/3 + x\hat{\ }2 + 1$

$>> taylor(log(1 + x\hat{}2) * sqrt(1 - x), 7)$

$ans =$

$(137 * x\hat{}6)/384 + (3 * x\hat{}5)/16 - (5 * x\hat{}4)/8 - x\hat{}3/2 + x^2$

7.6 Limit calculation

7.6.1 limit command

$>> syms\ x;$

$>> limit((x - 7)/(x\hat{}2 - 49), 7)$

$ans =$

$1/14$

$>> limit((x - 7)/(x\hat{}2 - 49), +inf)$

$ans =$

0

$>> limit(sin(3 * x)/tan(7 * x), 0)$

$ans =$

$3/7$

$>> limit((sin(x) - 1) * tan(x)\hat{}2, pi/2)$

$ans =$

$-1/2$

Exercise 1:

Verify if a multivariable function is the solution to a PDE, $w(x, y) = sin(x) + sin(y)$ a solution of the differential equation

$$\frac{\partial^2 u}{\partial^2 x} + \frac{\partial^2 u}{\partial^2 x} = 0$$

solution:

$>> syms\ x\ y$

$>> w = sin(pi * x) + sin(pi * y)$

$>> diff(w,x,2)+diff(w,y,2)$

$>> simplify(ans)$

Exercise 2:

Let the following BVP be:

$$\begin{cases} -\dfrac{\partial^2 u}{\partial^2 x} & = x+1 \quad 0<x<1 \quad , \\ u(0) & = 2 \qquad u(1)=0 \qquad , \end{cases}$$

solution:

MATLAB does not add an integration constant, so I do it explicitly:

clear $syms\ x\ C1\ C2$

$r1 = int(-(1+x),x)+C1$

$r2 = int(r1,x)+C2$

Exercise 3:

solving the system of equations in multiple variables

$$\begin{cases} x+y & =1 \quad , \\ 2x\times y & =1 \quad , \end{cases}$$

solution:

$>> syms\ x\ y$

$>> s = solve(x+y-1, 2*x-y-1, x, y)$

$>> s.x$

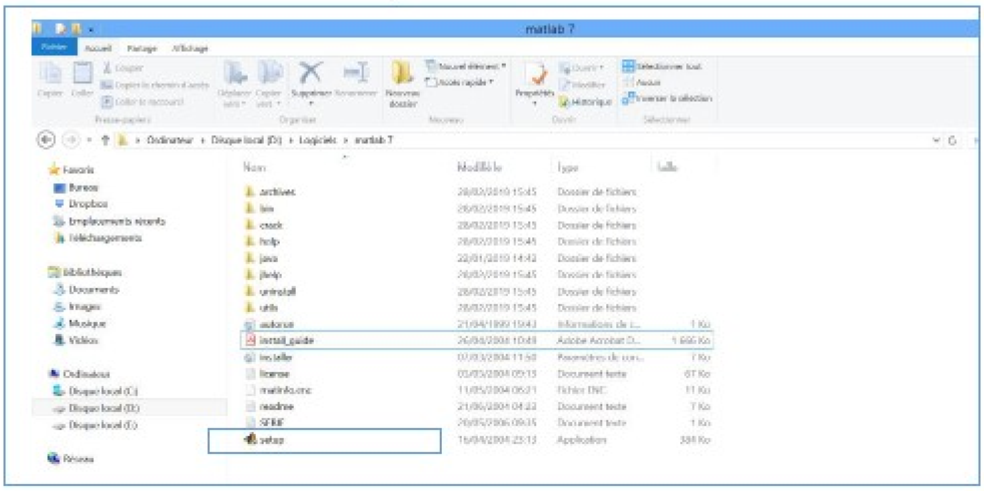
$>> s.y$

Chapter 8

Appendix:MATLAB Installation Guide

Through the following guide, we will learn how to install version 7 of MALAB. For a successful installation, follow the following steps:

1) A first step consists of launching the installation by double-clicking on the setup file (the file installation of MATLAB).

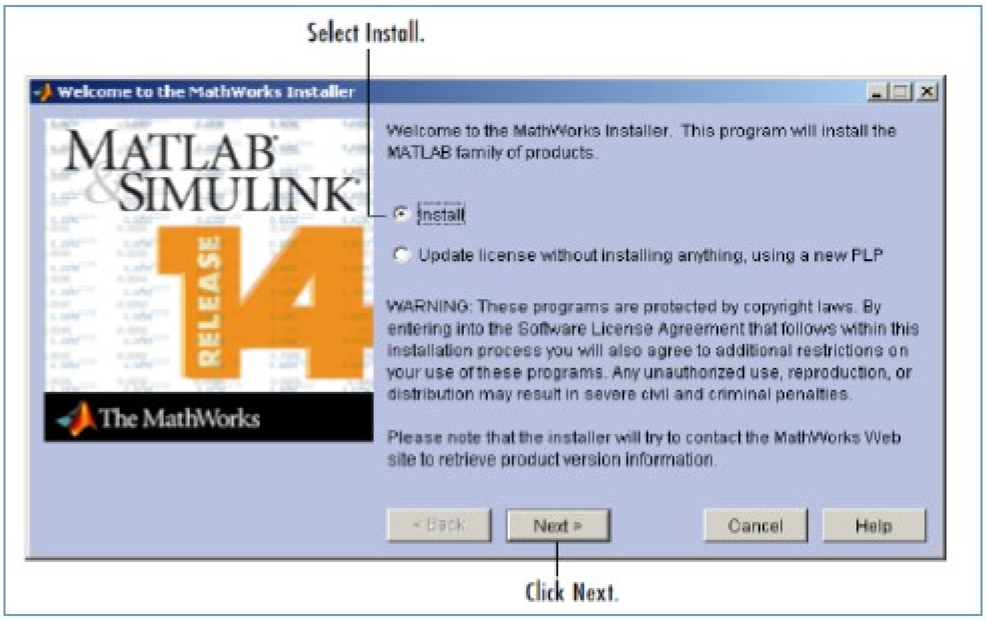


matlab 7

Ordinateur › Disque local (D:) › Logiciels › matlab 7

Nom	Modifié le	Type	Taille
archives	28/02/2019 15:45	Dossier de fichiers	
bin	28/02/2019 15:45	Dossier de fichiers	
crack	28/02/2019 15:45	Dossier de fichiers	
help	28/02/2019 15:45	Dossier de fichiers	
java	22/01/2019 14:42	Dossier de fichiers	
jhelp	28/02/2019 15:45	Dossier de fichiers	
uninstall	28/02/2019 15:45	Dossier de fichiers	
utils	28/02/2019 15:45	Dossier de fichiers	
autorun	21/04/1999 19:43	Informations de i...	1 Ko
install_guide	26/04/2004 10:48	Adobe Acrobat D...	1 666 Ko
installer	07/03/2004 11:50	Paramètres de con...	7 Ko
license	03/03/2004 05:13	Document texte	67 Ko
matlab.cnc	11/05/2004 06:21	Fichier CNC	11 Ko
readme	21/06/2004 04:22	Document texte	7 Ko
SERIE	20/05/2006 09:35	Document texte	1 Ko
setup	16/04/2004 23:13	Application	388 Ko

2) Check the Install button radio and click on the Next button.



Welcome to the MathWorks Installer. This program will install the MATLAB family of products.

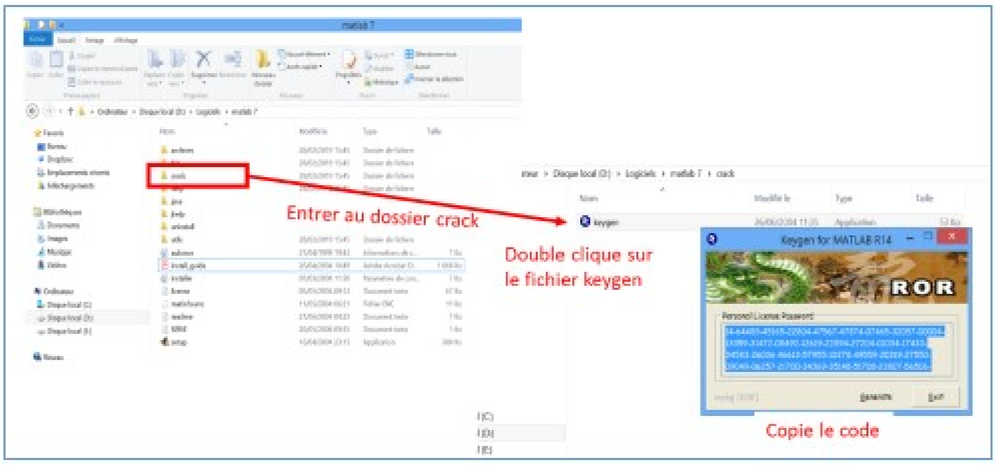
(● Install)

○ Update license without installing anything, using a new PLP

WARNING: These programs are protected by copyright laws. By entering into the Software License Agreement that follows within this installation process you will also agree to additional restrictions on your use of these programs. Any unauthorized use, reproduction, or distribution may result in severe civil and criminal penalties.

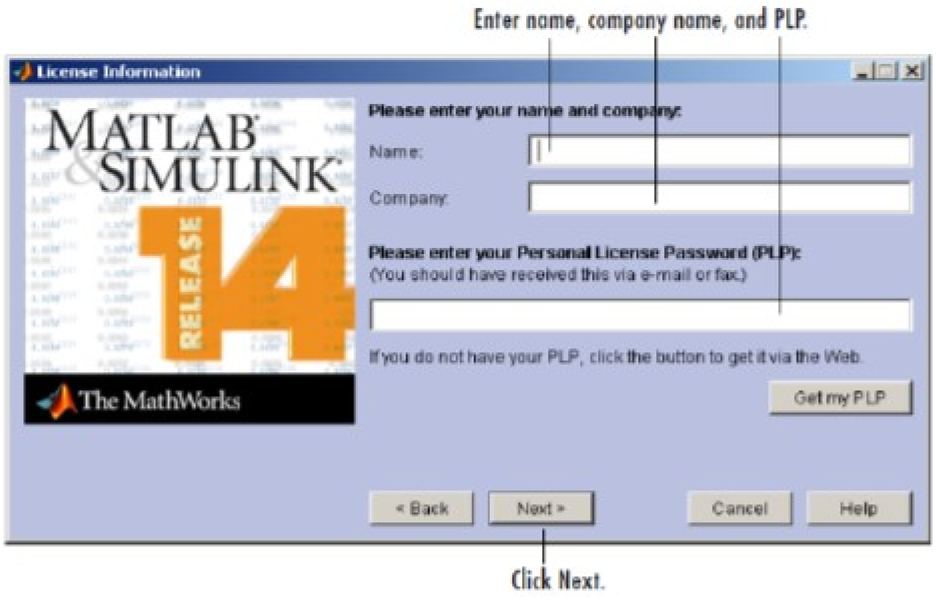
Please note that the installer will try to contact the MathWorks Web site to retrieve product version information.

3) Copy the activation code.



4) Fill the Name and company fields with characters. Then, copy the activation code from the crack file (see

previous slide) and paste it into the PLP box;



Please enter your name and company:

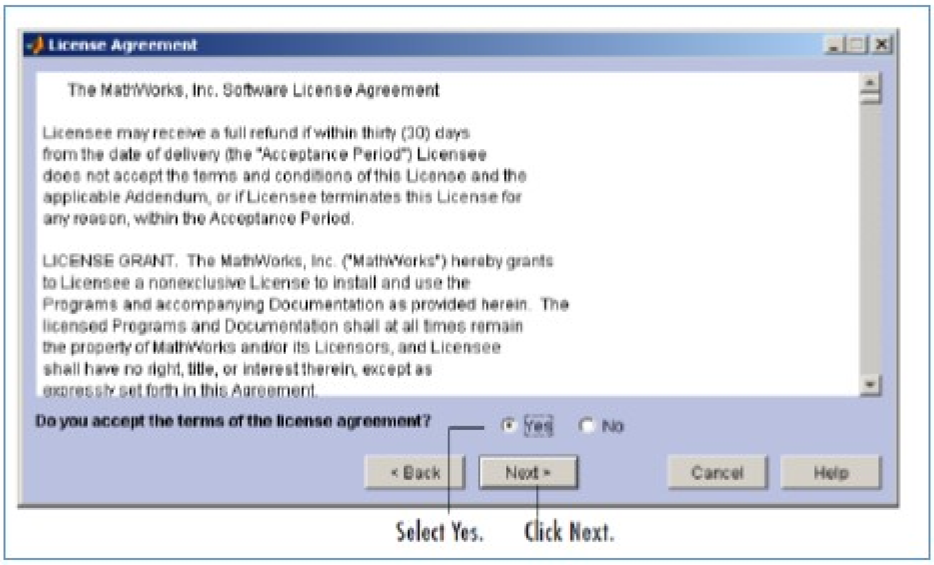
Name:

Company:

Please enter your Personal License Password (PLP):
(You should have received this via e-mail or fax.)

If you do not have your PLP, click the button to get it via the Web.

5) Check the Yes button and click Next.



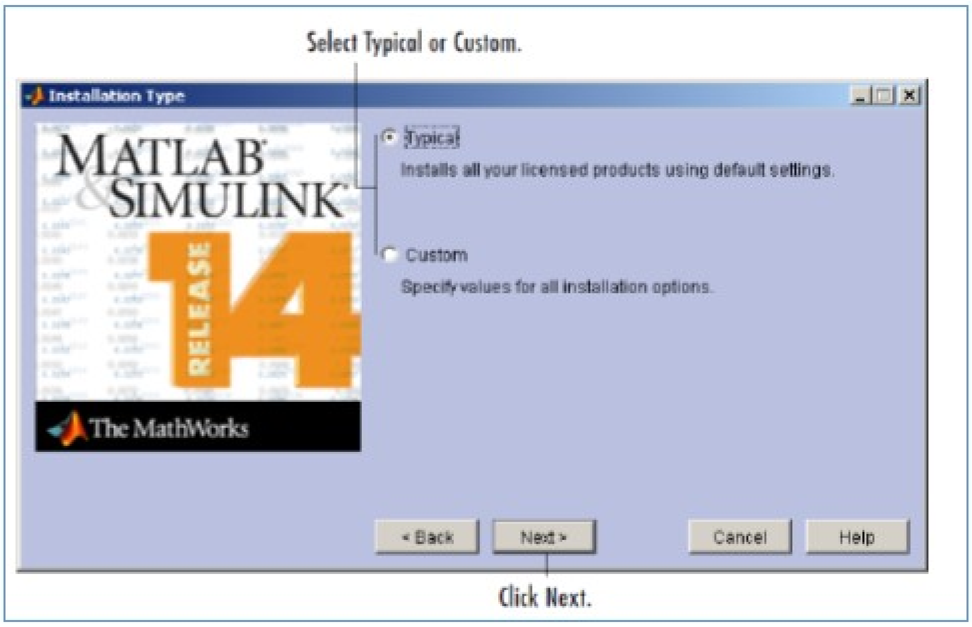
The MathWorks, Inc. Software License Agreement

Licensee may receive a full refund if within thirty (30) days from the date of delivery (the "Acceptance Period") Licensee does not accept the terms and conditions of this License and the applicable Addendum, or if Licensee terminates this License for any reason, within the Acceptance Period.

LICENSE GRANT. The MathWorks, Inc. ("MathWorks") hereby grants to Licensee a nonexclusive License to install and use the Programs and accompanying Documentation as provided herein. The licensed Programs and Documentation shall at all times remain the property of MathWorks and/or its Licensors, and Licensee shall have no right, title, or interest therein, except as expressly set forth in this Agreement.

Do you accept the terms of the license agreement? [✓ Yes] No

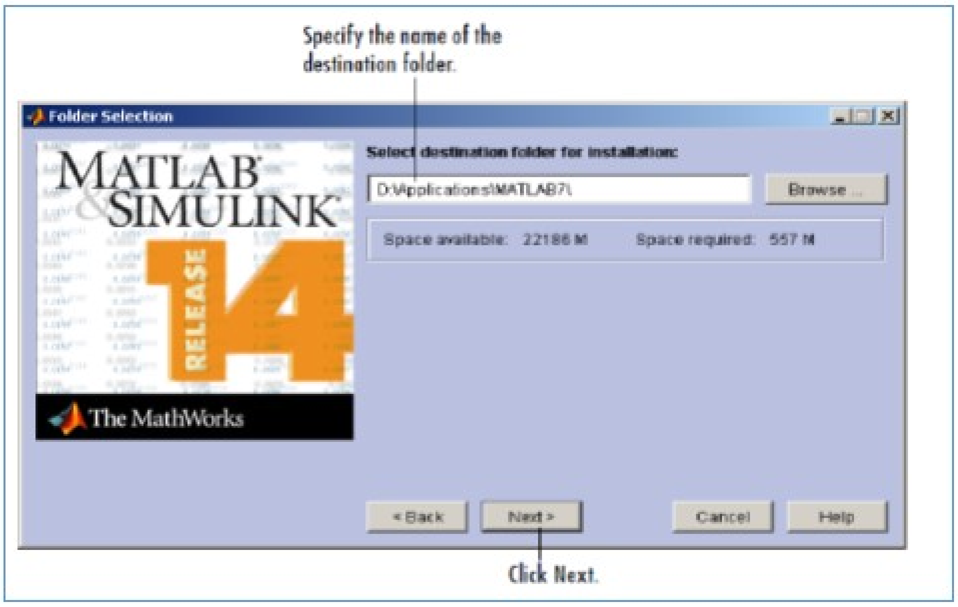
6) Check the Typical installation type and click Next.



(●) Typical
Installs all your licensed products using default settings.

( ) Custom
Specify values for all installation options.

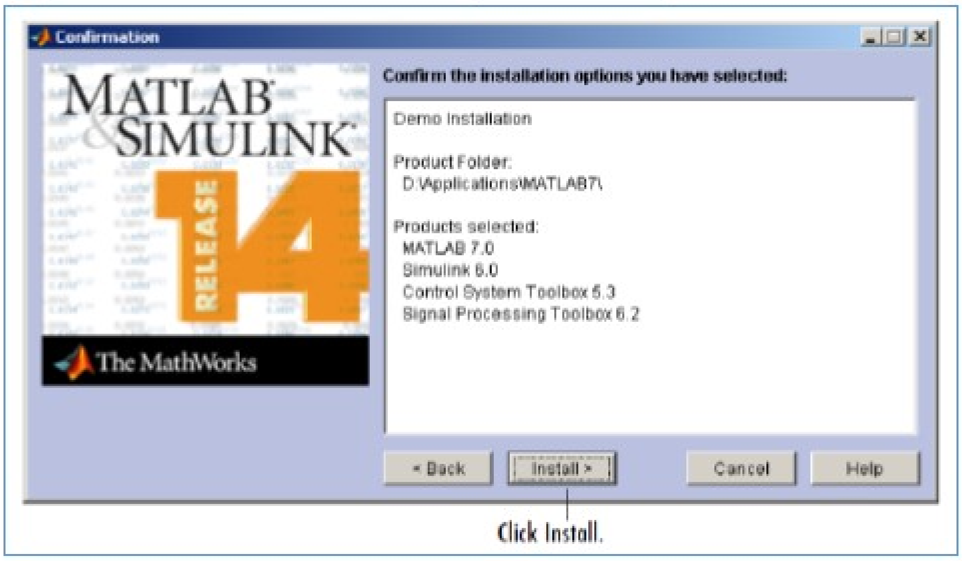
7) Specify the installation file then Click on Next;



Select destination folder for installation:
D:\Applications\MATLAB7\

Space available: 22186 M   Space required: 557 M

8) Confirm the installation and Click Next;



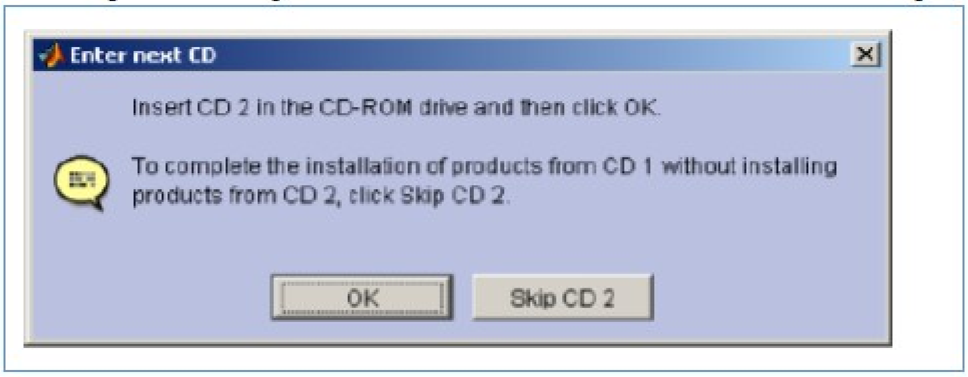
Confirm the installation options you have selected:

Demo Installation

Product Folder:
D:\Applications\MATLAB7\

Products selected:
MATLAB 7.0
Simulink 6.0
Control System Toolbox 6.3
Signal Processing Toolbox 6.2

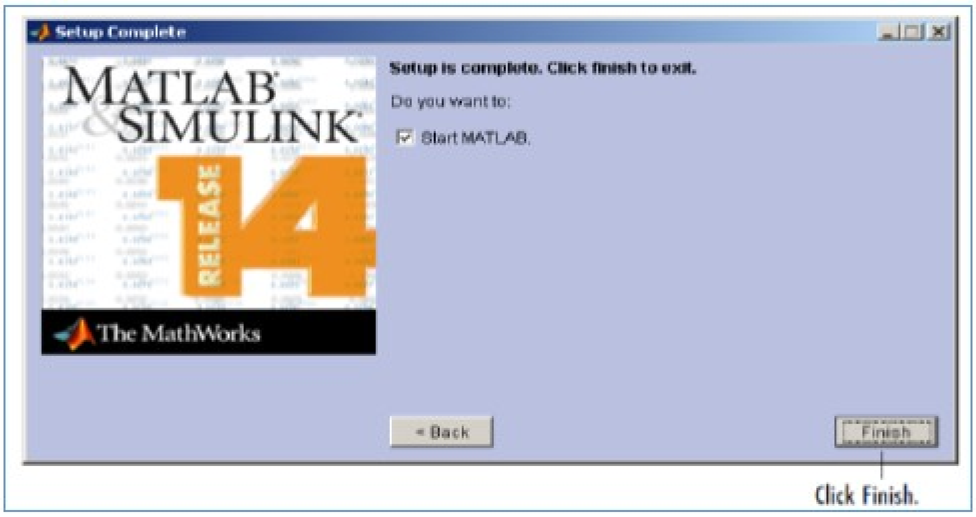
9) Click on Skip CD 2. Then on Next on the new window that appears;



Insert CD 2 in the CD-ROM drive and then click OK.

To complete the installation of products from CD 1 without installing products from CD 2, click Skip CD 2.

10) Complete the installation by Clicking on Finish;

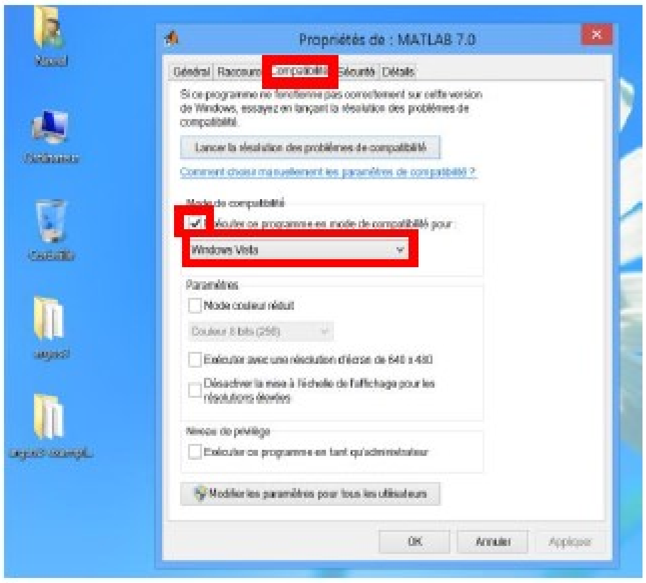


Setup is complete. Click finish to exit.

Do you want to:

☑ Start MATLAB.

11) After installation, if MATLAB is displayed correctly it is good, otherwise, go to the MATLAB icon in the desktop and click on the right you will have the menu in the figure, then click on properties. Go to the compatibility tab, then choose Windows Vista as compatibility mode;


Propriétés de : MATLAB 7.0
Général
Raccourci
Compatibilité
Sécurité
Détails
Lancer la résolution des problèmes de compatibilité
Comment choisir manuellement les paramètres de compatibilité ?
Modifier les paramètres pour tous les utilisateurs
OK
Annuler
Appliquer

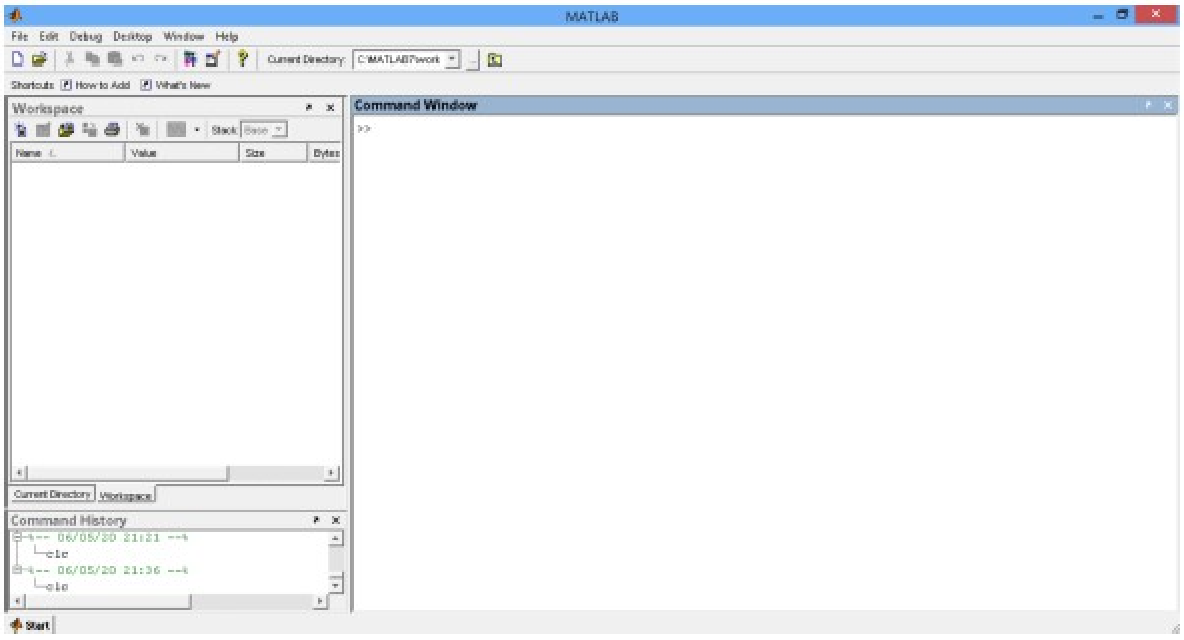
Si ce programme ne fonctionne pas correctement sur cette version de Windows, essayez en lançant la résolution des problèmes de compatibilité.

Mode de compatibilité
[✓] Exécuter ce programme en mode de compatibilité pour :
Windows Vista

Paramètres
[ ] Mode couleur réduit
Couleur 8 bits (256)
[ ] Exécuter avec une résolution d'écran de 640 x 480
[ ] Désactiver la mise à l'échelle de l'affichage pour les résolutions élevées

Niveau de privilège
[ ] Exécuter ce programme en tant qu'administrateur

12) Double click on the MATLAB icon to launch, and there you have MATLAB installed on your machine


MATLAB
File Edit Debug Desktop Window Help
Current Directory: C:\MATLAB7\work
Shortcuts How to Add What's New
Workspace
Stack: Base
Name Value Size Bytes
Current Directory Workspace
Command History
Command Window
Start

```
>>
```

```
%-- 06/05/20 21:21 --%
clc
%-- 06/05/20 21:36 --%
clc
```

Bibliography

[1] Attaway, S. (2013). Matlab: a practical introduction to programming and problem solving. Butterworth-Heinemann.

[2] Bastien, J., & Martin, J. N. (2003). Introduction l'analyse numrique: applications sous Matlab: cours et exercices corrigs. Dunod.

[3] Biran, A., & Breiner, M. (2004). MATLAB pour l'ingenieur. Pearson Education.

[4] Chaturvedi, D. K. (2017). Modeling and simulation of systems using MATLAB and Simulink. CRC press.

[5] Grenier, J. P. (2007). Débuter en algorithmique avec Matlab et Scilab. Ellipses.

[6] Higham, D. J., & Higham, N. J. (2016). MATLAB guide. Society for Industrial and Applied Mathematics.

[7] Koko, J. (2009). Calcul scientifique avec MATLAB: outils MATLAB spcifiques, quations aux drives partielles. Ellipses.

[8] Merrien, J. L. (2007). Exercices et problmes d'Analyse numrique avec Matlab: Rappels de cours, corrigs dtaills, mthodes. Dunod.

[9] Moore, H. (2014). MATLAB for Engineers. Prentice Hall Press.

[10] Pianosi, F., Sarrazin, F., & Wagener, T. (2015). A Matlab toolbox for global sensitivity analysis. Environmental Modelling & Software, 70, 80-85.

[11] Pietruszka, W. D. (2012). MATLAB und Simulink in der Ingenieurpraxis. Wiesbaden: Vieweg+ Teubner Verlag.

[12] Postel, M. (2004). Introduction au logiciel Matlab. en ligne]. Disponible sur: https://www. ljll. math. upmc. fr/ postel/matlab/(consult le 27/02/2020).

[13] Salehi, P., & Behzadfar, N. (2021). Investigation and simulation of different medical image processing algorithms to improve image quality using simulink matlab. Signal Processing and Renewable Energy, 5(4), 15-28.

[14] Sizemore, J., & Mueller, J. P. (2014). MATLAB for Dummies. John Wiley & Sons.