Ministry of Higher Education and Scientific Research

وزارة التعليم العالي و البحث العلمي

Amen-جامعة باجي مختار

BADJI MOKHTAR-ANNABA UNIVERSITY

Faculty of Technology-Department of Computer Science

COURSE: Programming tools for mathematics

Specialty: Computer Science

LMD 1st year

Presented

By

Pr^{r}HAFIDI Mohamed

Academic year 2024-2025

Foreword

Since the 1970s, the use of calculation software has become absolutely essential in the scientific field, for technicians, engineers, and researchers alike. Matlab was developed by Mathworks. It is one of the most popular scientific programming languages. This handout, entitled "Programming Tools for Mathematics," is intended for students in the S2 semester of the first year ofdegree (computer science department) from Badji Mokhtar University, Annaba. He was taught at the departmentfrom (2013). The format of this course is composed of 4 sessions of 4 hours of lectures associated with 4 sessions of 4 hours of practical work and is available at the address: https://sites.google.com/site/mhhafidi/.

Chapter 1: General Information and Getting Started

This chapter introduces the programming method in MATLAB. It includes all the basic concepts that the student needs to know. Topics covered include variables, numbers, main functions, main commands, and many other concepts.

1.1 Introduction

MATLAB (**MAST**rice**LAB**oratory) is an interactive programming environment for scientific computing, programming, and data visualization. It is widely used in engineering and scientific research, as well as in higher education institutions. Its popularity is due primarily to its strong and simple user interaction, but also to the following points:

- Its functional richness: With MATLAB, it is possible to perform complex mathematical manipulations by writing few instructions. It can evaluate expressions, draw graphs and execute classic programs. And above all, it allows the direct use of several thousand predefined functions.

- The possibility of using toolboxes: which encourages its use in several disciplines (simulation, signal processing, imaging, artificial intelligence, etc.).

- The simplicity of its programming language: a program written in MATLAB is easier to write and read compared to the same program written in C or PASCAL.

- Its way of handling everything as matrices, which frees the user from worrying about data typing and thus avoiding typecasting problems.

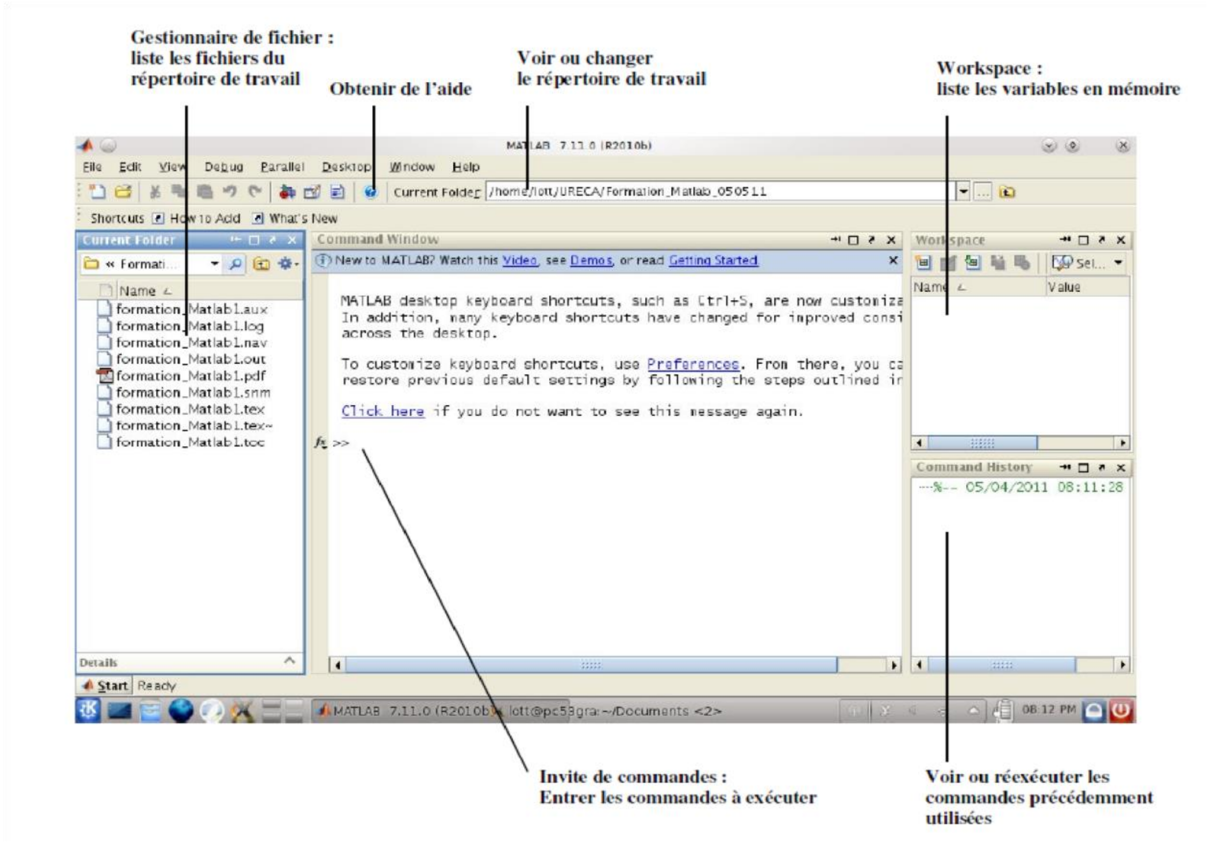
MATLAB was originally designed to do calculations mainly on vectors and matrices, hence its name '**Mast**rice**Lab**oratory', but later it was improved and expanded to be able to deal with many more areas.

MATLAB is not the only scientific computing environment available, as there are other competitors, the most prominent of which are MAPLE and MATHEMATICA. There are even free software clones of MATLAB, such as SCILAB and OCTAVE.

1.2. The Matlab interface

Currently MATLAB is at version**7.x**and at startup it displays several windows. Depending on the version we can find the following windows:

- **Current Folder**: indicates the current directory and existing files.
- **Workspace**: indicates all existing variables with their types and values.
- **Command History**: keeps track of all commands entered by the user.
- **Command Window**: we use to formulate our expressions and interact with MATLAB. Figure I.1 presents the window we use throughout this chapter.


Gestionnaire de fichier : liste les fichiers du répertoire de travail
Obtenir de l'aide
Voir ou changer le répertoire de travail
Workspace : liste les variables en mémoire
MATLAB 7.11.0 (R2010b)
File Edit View Debug Parallel Desktop Window Help
Current Folder: /home/lott/URECA/Formation_Matlab_050511
Shortcuts How to Add What's New
Current Folder
Command Window
Workspace
Name
Value
Command History
formation_Matlab1.aux
formation_Matlab1.log
formation_Matlab1.nav
formation_Matlab1.out
formation_Matlab1.pdf
formation_Matlab1.snm
formation_Matlab1.tex
formation_Matlab1.tex~
formation_Matlab1.toc
Details
Start Ready
Invite de commandes : Entrer les commandes à exécuter
Voir ou réexécuter les commandes précédemment utilisées

New to MATLAB? Watch this Video, see Demos, or read Getting Started

```
MATLAB desktop keyboard shortcuts, such as Ctrl+S, are now customiza
In addition, many keyboard shortcuts have changed for improved consi
across the desktop.

To customize keyboard shortcuts, use Preferences. From there, you ca
restore previous default settings by following the steps outlined ir

Click here if you do not want to see this message again.

>>
```

```
--%-- 05/04/2011 08:11:28
```

Figure I.1. The Matlab desktop

1.2.1 Basic tools

- The easiest way to use MATLAB is to write directly in the Command Window just after the cursor (prompt)>>

- If we want an expression to be calculated but without displaying the result, we add a semicolon '**;**' at the end of the expression

- To create a variable we use the simple structure: '**variable = definition**' without worrying about the type of the variable.

- It is possible to write multiple expressions in the same line by separating them with commas or semicolons.

- A variable name must only contain alphanumeric characters or the symbol '_' (underscore), and must start with an alphabet. We also need to be careful with capital letters because MATLAB is case sensitive (**HAS**And**has**are two different identifiers).

The basic operations in an expression are summarized in the following table:

The operation	**The meaning**
+	The bill
-	Subtraction
*	Multiplication
/	The division

\	The left division
^	The power
'	The transposed
(And)	Parentheses specify the order of evaluation

To see the list of variables used, either look at the window '**Workspace'**either we use the commands '**whos'**Or '**who'**.
whosgives a detailed description (the name of the variable, its type and its size), on the other hand**who**just gives the names of the variables.

1.2.2. Numbers

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 use the characters
'i' and 'j' (interchangeably) to designate the imaginary part.

MATLAB always uses real numbers (double precision) to perform calculations, which allows for calculation accuracy up to 16 significant digits. However, the following points should be noted:

- The result of a calculation operation is by default displayed with four digits after the decimal point.
- To display more numbers use the command**long format**(14 digits after the decimal point)**.**
- To return to the default display, use the command**short format**.
- To display only 02 digits after the decimal point, use the command**bank format**.
- To display numbers as a ratio, use the command**rat format**.

The function**vpa**can be used to force the calculation to present more significant decimals by specifying the number of decimals desired.

1.2.3. The main constants

MATLAB defines the following constants:

The constant	**Its value**
pi	□=3.1415...
exp(1)	e=2.7183...
i	$= \sqrt{-1}$
I	$= \sqrt{-1}$
Inf	∞
NaN	Not a Number
eps	$\varepsilon \approx 2 \times 10^{-16}$

1.2.4 Main functions

Among the frequently used functions, in the table below we can note the following:

The function	**Its meaning**
sin(x)	the sine of x (in radians)
cos(x)	the cosine of x (in radians)

tan(x)	the tangent of x (in radians)
donkey(s)	the arcsine of x (in radians)
acos(x)	the arc cosine of x (in radians)
atan(x)	the arctangent of x (in radians)
sqrt(x)	the square root of x
abs(x)	the absolute value of x
exp(x)	$= e^x$
log(x)	natural logarithm of x
log10(x)	logarithm to base 10 of x
image(s)	the imaginary part of the complex number x
real(x)	the real part of the complex number x
round(x)	round a number to the nearest whole number
floor(s)	round a number to the smallest integer
eye(s)	round a number to the nearest whole number
rem(m,n)	remainder of the integer division of m by n
lcm(m,n)	least common multiple of m and n
gcd(m,n)	greatest common divisor of m and n
factor(n)	prime factorization of n
conj(z)	conjugate of z
angle(z)	argument of z
abs(z)	Module of z

1.2.5 The main commands

MATLAB offers many commands for user interaction. We're sticking with a small set for now, and we'll introduce others as the course progresses.

The order	**Its meaning**
who	Displays the names of the variables used
whos	Displays information about the variables used
clear xy	Deletes the variables x and y
clear, clear all	Delete all variables
clc	Clears the command screen
exit, quit	Close the MATLAB environment
Long format	displays numbers with 14 digits after the decimal point
Short format	displays numbers with 04 digits after the decimal point
Bank format	displays numbers with 02 digits after the decimal point
Rat format	displays numbers as a ratio (a/b)

1.3 Training exercises:

1. Translate the following MATLAB instructions into mathematical expressions.

>>Y= 12*sqrt(2)*cos(2*pi*f*t-3*pi/4)	>>T= exp(2-sqrt(b^3-1/a))

>>Z= exp(sqrt(3*n^2+log(n/5))	>> S=abs(2*n^5-3)/sqrt(4*n^2+log(6*n))

2. Translate the following mathematical expressions into MATLAB instructions:

$$x = \frac{-b - \sqrt{b^2 - 4 * a * c}}{2 * a}$$
$$u = 12\sqrt{2} * cos(2 * \pi * f * t - \frac{3 * \pi}{4})$$
$$z = \exp(\sqrt{3 * n^2 + \ln(\frac{n}{5})})$$

Chapter 2: Vectors

MATLAB was originally designed to allow mathematicians, scientists, and engineers to easily use the mechanics of linear algebra. We first introduce vectors and then matrices, which are very intuitive and convenient in MATLAB.

2.1. Description

A vector is an ordered list of elements. If the elements are arranged horizontally, the vector is said to be a row vector; if the elements are arranged vertically, it is said to be a column vector.

To create a**line vector**you just have to write the list of its components in brackets**[]** and separate them with spaces or commas as follows:

```
>>v = [1 3.4 5  -6]
v =
1.0000 3.4000 5.0000 -6,0000
```

To create a**column vector**you just have to write the list of its components in brackets**[]** and separate them with semicolons (**;**) as follows:

```
>>U = [4 ;  -2; 1]
U =
4
-2
1
```

Or to calculate the transpose of a row vector:

```
>>U = [4  -21]'
U =
4
-2
1
```

If the components of a vector X are ordered with consecutive values with a step (increment/decrement) different from 1, we can specify the step with the following notation:

```
X = [value -initial: increment: value -final]
```

Pfor example:

```
>>x = [0 : pi / 10 : pi ]% [value -initial: increment: value -final]
x =
Columns 1 through 7
0 0.3142 0.6283 0.9425 1.2566 1.5708 1.8850
Columns 8 through 11
2.1991 2.5133 2.8274 3.1416
```

We can write more complex expressions like:

```
>> V = [1:2:5 , -2:2:1]
V =
1 3 5 -2 0
>> A = [1 2 3]
A =
1 2 3
>> B = [A, 4, 5, 6]
B =
1 2 3 4 5 6
```

2.2 Addressing and indexing

Accessing the elements of a vector is done using the following general syntax:**vector_name (positions),**Or**positions**: can be a simple number, or a list of numbers (a vector of positions).
Parentheses**(**And**)**are used for consultation.
The hooks**[**And**]**are used only during creation.

Examples:

Order	**Meaning**	**Example with:** **V = [5, -1, 13, -6, 7]**
>> **V(3)**	3rd position	years = 13
>> **V(2:4)**	From second position to fourth	years = -1 13 -6
>> **V(4:-2:1)**	From the 4th position to the 1st with step = -2	years = -6 -1
>> **V(3:end)**	From 3rd position to last	years = 13 -6 7
>> **V([1,3,4])**	1st, 3rd and 4th position only	years = 5 13 -6
>> **V(1) = 8**	Give the value 8 to the first element	V = 8 -1 13 -6 7
>> **V(6) = -3**	Add a sixth element with the value -3	V = 8 -1 13 -6 7 -3
>> **V(9) = 5**	Add a ninth element with the value 5	V = 8 -1 13 -6 7 -3 0 0 5
>> **V(2) = []**	Delete the second item	V = 8 13 -6 7 -3 0 0 5
>> **V(3:5) = []**	Delete from 3rd to 5th element	V = 8 13 0 0 5

2.3 Operations between vectors

With two vectors u and v, it is possible to perform the following operations:

The operation	**Meaning**	**Example with:** >> **u = [-2, 6, 1] ;** >> **v = [3, -1, 4] ;**

+	Addition of vectors	>> **u+2**years = 0 8 3>> **u+v**years = 1 5 5
-	Subtraction of vectors	>> **u-2**years = -4 4 -1>> **uv**years = -5 7 -3
.*	Element-by-element multiplication	>> **u*2**years = -4 12 2>> **u.*2**years = - 4 12 2
		>> **u.*v**years = -6 -6 4
./	Element-by-element division	>> **u/2** years = -1.0000 3.0000 0.5000 >> **u./2** years = -1.0000 3.0000 0.5000 >> **u./v** years = -0.6667 -6.0000 0.2500
.^	Power element by element	>> **u.**^**2**years = 4 36 1>> **u.**^**v** years = -8.0000 0.1667 1.0000

2.4 Special functions

Here are some of the most used functions related to vectors:

Order	**Meaning**
Linspace (start, end, number of elements)	The creation of a vector whose components are ordered by regular intervals and with a well-defined number of elements
Sum(V)	The sum of the elements of a vector V
prod (V)	The product of the elements of a vector V
mean (V)	The average of the elements of a vector V
fate (V)	Orders the elements of vector V in ascending order
length(V)	the size of the vector V
max(V)	largest element of vector V
min(V)	Smallest element of vector V
fliplr(V)	reverses the order of the elements of vector V

2.5 Training exercises

1. Given two vectors$x$= (x1, x2, . . . , xn) And$y$= (y1, y2, . . . , yn), write a program giving the vectorz= (z1, z2, . . . , zn) whose component number$i$is that of the two numbers*11,yi*which is the largest in absolute value.

2. Create a row vector where the first element is 1, the last element 33 with an increment of 2 between elements: (1, 3, 5,......,33).

3. Create a column vector in which the first element is 15; the elements decrease with an increment of -5, and the last element is -25. (Note: A column vector can be created by transposing a row vector)

Chapter 3: Matrices

The basic element for MATLAB is a matrix with complex elements. Thus, any real number is considered a one-row, one-column matrix whose only element has a zero imaginary part. A vector is simply a one-row or one-column matrix. Vectors are also used to represent polynomials and strings. Multidimensional arrays are matrices concatenated along certain directions.

3.1 Description

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

• Items must be enclosed in brackets**[**And**]**

• Spaces or commas are used to separate items in the same line

• A semicolon (or the key**enter**) is used to separate lines Examples:

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

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

Order	**Meaning**	**Result**
>> **x = 1:4**	creation of a vector x	x = 1 2 3 4
>> **y = 5:5:20**	creation of a vector y	y = 5 10 15 20
>> **z = 4:4:16**	creation of a z vector	z = 4 8 12 16
>> **A = [x ; y ; z]**	A is formed by the row vectors x, y and z	A = 1 2 3 4 5 10 15 20 4 8 12 16
>> **B = [x' y' z']**	B is formed by the column vectors x, y and z	B = 1 5 4 2 10 8 3 15 12 4 20 16
>>**C = [x ; x]**	It is formed by the same line vector x 2 times	C = 1 2 3 4 1 2 3 4

3.2 Addressing and indexing

It is useful to note the following possibilities:

- Access to a line element**i**and the column**I**is done by:**A(i,j)**
- Access to the entire line number**i**is done by:**A(i,:)**
- Access to the entire number column**I**is done by:**A(:,j)**

Examples:

Order	Meaning	Example with: A = [1,2,3,4;5,6,7,8; 9,10,11,12]
>> **A(2,3)**	the element on the 2nd row in the 3rd column	years = 7
>> **A(1,:)**	all elements of the 1st line	years = 1 2 3 4
>> **A(:,2)**	all elements of the 2nd column	years = 2 6 10
>> **A(2:3,:)**	all elements of the 2nd and 3rd row	years = 5 6 7 8 9 10 11 12
>> **A(1:2,3:4)**	The upper right sub-matrix of size 2x2	years = 3 4 7 8
>> **A([1,3],[2,4])**	the submatrix: rows (1,3) and columns (2,4)	years = 2 4 10 12
>> **A(:,3) = []**	Delete the third column	A = 1 2 4 5 6 8 9 10 12
>> **A(2,:) = []**	Delete the second line	A = 1 2 4 9 10 12
>> **A = [A, [0;0]]**	Add a new column with A(:,4)=[0;0]	A = 1 2 4 0 9 10 12 0
>> **A = [A ; [1,1,1,1]]**	Add a new line with A(3,:)=[1,1,1,1]	A = 1 2 4 0 9 10 12 0 1 1 1 1

The dimensions of a matrix can be acquired using the function**size**.

Example :

Order	Meaning	Result
>> **d = size(A)**	the variable**d**contains the dimensions of matrix A as a vector.	d = 3 4
>> **d1 = size(A, 1)**	d1 contains the number of lines (m)	d1 = 3
>> **d2 = size(A, 2)**	d2 contains the number of columns (n)	d2 = 4

3.3 Special functions

The following table shows some of the most used functions regarding matrices:

The function	Meaning
zeros(n)	Generates an n × n matrix with all elements = 0
zeros(m,n)	Generates an m × n matrix with all elements = 0
ones(n)	Generates an n × n matrix with all elements = 1
ones(m,n)	Generates an m × n matrix with all elements = 1
eye(n)	eye(n) Generates an n × n identity matrix
magic(n)	Generates a magic matrix of dimension n × n
rand(m,n)	Generates an m × n matrix of random values
det	Calculating the determinant of a matrix
inv	Calculates the inverse of a matrix
rank	Calculates the rank of a matrix
trace	Calculates the trace of a matrix
eig	Calculates the eigenvalues
dowry	Calculates the scalar product of 2 vectors
norm	Calculates the norm of a vector
cross	Calculates the vector product of 2 vectors
diag	Returns the diagonal of a matrix
diag(V)	Creates a matrix with vector V on the diagonal and 0 elsewhere.
tril	Returns the lower triangular part
trio	Returns the upper triangular part

3.4 Operations on matrices

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

Examples:

>> **A=ones(2,3)**	A = 1 1 1 1 1 1
>> **B=zeros(3,2)**	B = 0 0 0 0 0 0
>> **B=B+3**	B = 3 3 3 3 3 3
>> **A*B**	years = 9 9 9 9

>> **B=[B , [3 3 3]']**	B =

	3 3 3
	3 3 3
	3 3 3

>> **B=B(1:2,:)**	B = 3 3 3 3 3 3
>> **A=A*2**	A = 2 2 2 2 2 2
>> **A.*B**	years = 6 6 6 6 6 6
>> **A*eye(3)**	years = 2 2 2 2 2 2

3.5 Training exercises

1. Use the zeros, ones, and eye commands to create the following arrays:

$$a = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad b = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad c = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

2. Create the following matrix A:

$$A = \begin{bmatrix} 6 & 43 & 2 & 11 & 87 \\ 12 & 6 & 34 & 0 & 5 \\ 34 & 18 & 7 & 41 & 9 \end{bmatrix}$$

Use matrix A to:

- Create a column vector of 5 lines named **go** which contains the elements of the second line of A.
- Create a three-element column vector named**vb** which contains the elements of the fourth column of A.
- Create a 10-element column vector named**vc** which contains the elements of the first and second rows of A.
- Create a 6-element column vector named**vd** which contains the elements of the first and fifth columns of A.

Chapter 4: Programming in Matlab

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

4.1. General information

4.1.1 Comments

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

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

4.1.2 Reading data into a program

To read a value given by the user, it is possible to use the command**input**, which has the following syntax:
Variable = input('an indicative sentence')

When MATLAB executes such an instruction, the hint sentence will be displayed to the user while waiting for the user to enter a value. For example:

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

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

1 3 5 7 3 2 1 0
```

4.1.3 Writing Data to a Program

To display the value of a variable, we can use the function**available**, and which has the following syntax:

```
>>disp (object)
```

The object value can be a number, vector, matrix, string, or expression.

4.2.Logical and relational operators

The following operators are used to compare different variables:

The comparison operator	**meaning**
==	legality
~=	inequality
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
The logical operator	
&	THE**And**logic
|	THE**Or**logic
~	there**negation**logic

Comparing vectors and matrices is somewhat different from scalars, hence the usefulness of the two functions '**equal**' And '**isempty**' (which allow a concise answer to be given for comparison).

The function	**Description**
equal	tests whether two (or more) matrices are equal (having the same elements everywhere). It returns**1**if so, and**0**Otherwise.
isempty	tests whether an array is empty (contains no elements). It returns**1**if so, and**0**Otherwise.

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

Order	**meaning**	**execution**
>> **A = [5,2;- 1,3]**	Create matrix A	A = 5 2 1 3
>> **B = [5,1;0,3]**	Create matrix B	B = 5 1 0 3
>> **A==B**	Test if A=B?	years = 1 0 0 1
>> **equal(A,B)**	Test whether A and B are actually equal	years = 0
>> **C=[] ;**	Create the empty matrix C	
>> **isempty(C)**	Test if C is empty	years =1
>> **isempty(A)**	Test if A is empty	years = 0

4.3 Control structures

Control flow structures are instructions that define and manipulate the order of execution of tasks in a program. They provide the ability to perform different processing depending on the state of the program's data, or to create repetitive loops for a given process.

4.3.1 Selection - if . . . end and if . . . else . . . end

```
–       if(boolean expression)/script/ end
–       if(boolean expression)/script if true/ else /script if   fake /
end
```

The / symbol replaces one of the separator symbols: comma (,), semicolon (;), or line break. The use of the semicolon is strongly recommended to avoid often redundant displays.

Example :

```
>> m = -1;
>> if (m<0), a = -m, end
```

When there are more than two alternatives, the following structure can be used:

```
if (exp1)
script1 (evaluated if exp 1 is true)
elseif (exp2)
script2 (evaluated if exp 2 is true)
elseif (exp3)
. . .
else
script (evaluated if none of the expressions exp1, exp2, . . . is not true)
end
```

For example, 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
```

4.3.2. Switch construction. . . box

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

```
switch (selector)
case value 1, . . . / script 1 /
case Value 2, . . . / script 2 /
. . . otherwise / script end
```

As in the previous definitions, the / symbol replaces a separator: comma (,), semicolon (;) or line break.*selector*denotes an expression whose values can correspond to the values associated with the different case values. When the selector value corresponds to a case value, once the corresponding script is executed, execution continues immediately after the end, unlike what happens in some languages. This explains the absence of a break after each script.

Example :

```
        >> n = 17
        >> switch rem(n,3) % remainder of dividing n by 3
        case 0, disp('Multiple of 3')   case 1,
disp('1 modulo 3')      case 2, disp('2 modulo 3')
        otherwise, disp('Negative number')
        end
```

4.3.3 Conditional Iteration - while . . . end

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

```
while (boolean expression) / script / end
```

The symbol / represents, as in the previous definitions, a separator: comma (,), semicolon (;) or line break. On the other hand, the use of variables should be avoided*i*And*I*as indices since they are predefined variables whose value is sqrt(−1).

Example :

```
>>n = 1 ;
>>while (2^n  <= 100)
n = n + 1;
end
>>disp(n -1)
```

4.3.4. Repetition - for . . . end

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

```
for (k = list) / script / end
```

Example 1:

```
>>x  =[ ];
>>for (k = 1:5), x = [x, sqrt(k)], end
```

Example 2:

```
>>for (l = 1 :  3)
for ( k = 1 : 3)
A(l,k) = l^2 + k^2 ;
end
end
>>available)
```

It is possible to exit a for or while loop directly using the break command:

Example :

```
>> EPS = 1;
>> for (n = 1 : 100)
EPS = EPS / 2;
If ((EPS + 1) <= 1)
EPS = EPS*2 break  end      end      >>
n
```

The test (EPS + 1) <= 1 causes the for loop to exit at 52*th*iteration. In the following example, table A is the one from example 2.

```
>>for (l = 1 :  3)
for (k = 1 : 3)
if (A(l,k) == 10)
[l,k]
break
end
end
end
```

The double loop did not stop after the test A(l,k) == 10 was validated when l=1 and k=3. Indeed break**causes the nearest loop to exit**, here the inner for loop. A corrected version of the previous test could be the following with two breaks to be able to exit the two for loops:

```
>> output = 0;
>>for  (l=1:3)
if (exit)
break
end
for (k = 1:3)
if (A(l,k) == 10)
[l,k]
output = 1 ;
break
end
end
end
```

4.5. Scripts

A script is a sequence of expressions or commands. A script can extend over one or more lines. Individual expressions or commands must be separated by a comma, a semicolon, or the line break symbol consisting of three periods . . .
followed by<enter>(the role of the three points and inhibiting the evaluation mechanism during a line break). As with a single expression, typing<enter>triggers the evaluation process. Expressions are evaluated in the order in which they are written. Only the value of expressions followed by a comma or a line break is displayed; the value of expressions followed by a semicolon is not.

Example :

```
>> a = .5, 2*a, save a, b = pi; 2*b, c = a*b
>> years
```

Writing a script is quite tedious, so MATLAB allows you to save the text of a script as a text file called*m-files*, due to their extension.

4.5.1 Creating m-files

THE*m-files*allow scripts to be saved as text files and are used in particular to define new functions (a large part of MATLAB's predefined functions are stored as*m-files*in the matlab toolbox).
THE*m-files*can be created by any editor. In recent versions of MATLAB there is a small built-in editor that can be called from the file menu or from the command window menu bar.

In the editor window type the following lines:

```
% script - test . m
a = .5;          b = pi;
c = a * b
```

Save the file in the working directory as test.m.

4.5.2 Running an m-file

To run the script contained in a*m-file*and just type the name of this m file in the command window followed by< *enter* >

```
>>essay
```

4.6. Functions

We've seen a number of predefined functions. It's possible to define your own functions. The first method allows you to define simple functions on a command line. The second, much more general, method allows you to define very advanced functions by defining them in a file.

4.6.1 Inline functions Example
:

Let's say I want to define a new function that I call sincos defined mathematically by: sincos(x) = sin x − x cos x

We will write:

```
>> sincos = inline('sin(x)-x*cos(x)')

since =
Inline function:
sincos(x) = sin(x)      -x*cos(x)
```

We can now use this new function:

```
>> sincos(pi/12)      years =
0.0059
```

Now let's try to apply this function to an array of values:

```
>> sincos(0:pi/3:pi)

??? Error using ==> inline/subsref
Error in inline expression ==> sin(x)-x*cos(x)
??? Error using ==> *
Inner matrix dimensions must agree.
```

It doesn't work. MATLAB tries to multiply x row vector by cos(x) also row vector in the sense of matrix multiplication! Therefore you have to use a term-by-term multiplication.* in the function definition:

```
>>sincos = inline('sin(x )-x.*cos(x)')

since =
Inline function:
sincos(x) = sin(x)-x.*cos(x)

>>sincos(0:pi/3:pi )
years
00.34241.91323.1416
```

4.6.2 Functions defined in a file

This is the most general method, and it allows you to create functions with multiple outputs. Let's start by using the previous example (sincos). The order of operations is as follows:

1. Edit a new file called sincos.m

2. Type the following lines:

```
function s = sincos(x)
s = sin(x)-x.*cos(x);
```

3. Save

The result is the same as before:

```
>>sincos(pi/ 12)
years
0.0059
```

Now let's see how to define a function with multiple outputs. We want to create a function called cart2pol that converts Cartesian coordinates (x, y) (inputs of thefunction) in polar coordinates (r,□) (function outputs). Here is the content of the cart2pol.m file:

```
function [r, theta] = cart2pol (x, y)
r = sqrt(x.^2 + y.^2); theta
= atan(y./x);
end
```

Note that the two output variables are enclosed in brackets and separated by a comma.
To use this function, we would write for example:

```
>>[rr,tt] = cart2pol(1, 1)
rr =
1.4142
tt =
0.7854
```

We therefore simultaneously assign two variables rr and tt to the two outputs of the function, by putting these two variables in brackets, and separated by a comma. The brackets here do not have the same meaning as for arrays.
It is possible to retrieve only the first output of the function. MATLAB often uses this principle to define functions with a "main" output and "optional" outputs.
So, for our function, if only one output variable is specified, only the polar radius value is returned. If the function is called without an output variable, ans takes the polar radius value:

```
>>cart2pol(1, 1)
years
1.4142
```

Inside functions like the one we just wrote, you are allowed to manipulate three types of variables:
– The function's input variables. You cannot change their values. – The function's output variables. You must assign them a value.
– Local variables. These are temporary variables used to break down calculations, for example. They only have meaning within the function and will not be "seen" from the outside.
Now let's imagine you write a command file and a function (in two different files, of course), with the command file using the function. If you want to pass a value from the command file to the function, you normally have to pass it through an input to the function. However, sometimes you'd like the variable A in the command file to be usable directly by the function. To do this, you use the global directive.

4.7 Training exercises

1. Write a Matlab script program that creates a square matrix of order n such that each element on and above the main diagonal is equal to the product of the column number and the row number. Elements below the diagonal are equal to zero.
2. Given a matrix A of any dimensions, write a function which returns as output variables a vector consisting of the strictly positive elements of A as well as the number of non-zero elements of A.
3. You have a correlation giving the**CP** of a gas as a function of temperature:

$$\mathbf{Cp(T) = AT^3 + BT^2 + CT + D}$$

- Create a Matlab script that uses theCp function having 5 inputs: T, A, B, C, D.
- But it is more natural for this function to have only T as input.How to pass A, B, C, D which are constants?

Chapter 5: Graphical Representation under Matlab

MATLAB, in addition to allowing you to perform very high-level numerical calculations, can also produce impressive 2D or 3D graphics. To get a brief overview of MATLAB's graphical capabilities, you can access graphics demos.

```
>>      Matlab graphics demo
```

In this chapter, we will present the basic principles essential for drawing curves in MATLAB.

5.1 The plot command

The order**plot** allows you to plot a set of coordinate points (x_i,y_i),$i=1$, ..., N. The syntax is plot(x,y) where x is the vector containing the valuesx_i on the abscissa and y is the vector containing the valuesy_i in ordinate. Of course, the x and y vectors must be of the same dimension, but they can be row or column vectors. By default, the points (x_i,y_i) are connected to each other by straight line segments.

Here for example is another way to plot the graph of the function*h(x) = x sin(x)*between -2□and 2□.

```
>> x=[-2*pi:0.01:2*pi]; y = x.*sin(x);
>> plot(x,y)
```

Also try:

```
>> x=[-2*pi:1:2*pi]; y = x.*sin(x);
>>      plot(x,y)
```

In this example we have defined a vector x of equally distributed values between -2□and 2□(with a step of*0.01* in the first case and*1* in the second case) and the image was calculated by the function*h* of these values (vector y). We display the points with coordinates (x(i), y(i)).You can specify to MATLAB what the color of a curve should be, what the line style should be and/or what the symbol should be at each point (x_i,y_i). To do this, we give a third input parameter to the plot command which is a 3-character string of the form**'cst'** with**c** designating the color of the line,**s** the dot symbol and**t**the type of line. The possibilities are as follows:

Curve color		**Representation of points**		**Curve Style**	
the character	**its effect**	**the character**	**its effect**	**the character**	**its effect**
bOr**blue**	blue curve	**.**	a point	-	online full
gOr**green**	green curve	**o**	a circle	**:**	dotted
rOr**red**	curve in red	**x**	the symbol**x**	- **.**	in dot dash
cOr**cyan**	between green and blue	+	the symbol+	- -	in dash
mOr**magenta**	in red bright purplish	*****	a star*****		

yOr**yellow**	yellow curve	**s**	a square		
kOr**black**	black curve	**d**	a diamond		
		v	triangle		
			lower		
		^	upper triangle		
		<	left triangle		
		>	right triangle		
		p	pentagram		
		h	hexagram		

The default values are c = b, s = .and t = - which corresponds to a solid blue line connecting the points together. It is not mandatory to specify each of the three characters. You can specify just one or two. The others will be the default values. The command**grid** allows you to obtain a grid of the figure.

It is possible to plot multiple curves on the same figure by specifying multiple x1, y1, x2, y2, ... arrays as parameters to the plot command. If you want the curves to have a different appearance, you can use separate color and/or line style options after each pair of x, y vectors.

Example: We plot on the interval [-5, 5] the function*x² cos(x)*in solid blue line and the function*x cos(x)* in red dotted line.

```
>> x = [-5:0.01:5];
>> y = x.^2.*cos(x); z = x.*cos(x);    >> plot(x,y,'b-',x,z,'r:');
```

5.2 Improving the readability of a figure

5.2.1 Captioning a figure

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

It is also very interesting to be able to indicate significant locations or points in a figure by a comment indicating their importance. - To give a title for the horizontal x-axis, we use the function**xlabel**like this:

```
>> xlabel('This is the X-axis')
```

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

```
>> ylabel('This is the Y axis')
```

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

```
>> title('figure title')
```

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

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

- To place a text on a position manually chosen by the mouse, we use the function**gtext**, which has the following syntax:

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

- With these commands, it is possible to display a value contained in a variable in the middle of text. To do this, we construct a string table by converting the value contained in the variable into a string using the command**num2str**.

Example :

```
>>title(['Example number', num2str(numex)]).
```

The following example illustrates the use of the different commands for labeling a figure.

```
>> P = 5;
>> t = [0:.01:2];
>> c = 12*exp(-2*t) - 8*exp(-6*t);
>> plot(t,c); grid
>> xlabel('time in minutes')
>> ylabel('concentration in grams per
liter')
        >> title(['evolution of the concentration of the
product', num2str(P), ...
        'over time'])
>> gtext('maximum concentration')
```

5.2.2 Displaying multiple curves in the same window

It is possible to display several curves in the same graphics window using the command**hold on**. The results of all graphics instructions executed after calling the command**hold on** will be superimposed on the active graphics window. To restore the previous situation (the result of a new graphics instruction replaces the previous drawing in the graphics window) type**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')
```

There are therefore two ways of superimposing several curves on the same figure.

- We can either give several pairs of abscissa/ordinate vectors as arguments to the plot command,
- or use the hold on command. Depending on the context, one of these solutions will be preferred over the other.

5.2.3 Saving a figure

The print command allows you to save the figure of a graphics window to a file in various image formats. The syntax of the print command is:

```
print  -f<num>  -d<format>  <ficname>
```

where

- <num> denotes the number of the graphics window. If this parameter is not specified, the active window is taken into account.
- <ficname> is the name of the file in which the figure is saved. If no name extension is given, a default extension is added to the file name depending on the chosen format (.ps for PostScript, .jpg for jpeg, for example).
- <format> is the format in which the figure is saved. There are many such formats. You can get the complete list by typing help plot. The main ones are:

Format	**Meaning**
ps	Black and white PostScript
PSC	Color PostScript
eps	Black and white Encapsulated PostScript
EPSC	Color Encapsulated PostScript
jpeg	JPEG image format
tiff	TIFF image format

5.3 The fplot Command

The order**fplot** allows you to plot the graph of a function over a given interval. The syntax is: fplot('nomf', [x_{min}, x_{max}])

where: namef is either the name of a built-in MATLAB function, an expression defining a function of the variable x, or the name of a user function.

- [x_{min}, x_{max}] is the interval for which the graph of the function is plotted.

5.4 The subplot command

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

- m is the number of sub-windows vertically;
- n is the number of sub-windows horizontally;
- i is used to specify which sub-window the display should be in. The windows are numbered from left to right and from top to bottom.

The following example illustrates the use of the subplot command.

```
>> figure
>> subplot(2,3,1), fplot('cos',[0 4*pi]), title('cosine'), grid
>> subplot(2,3,2), fplot('sin',[0 4*pi]), title('sinus'), grid
>> subplot(2,3,3), fplot('tan',[-pi/3 pi/3]), title('tangent'), grid
>> subplot(2,3,4), fplot('acos',[-1 1]), title('arc-cosine'), grid
>> subplot(2,3,5), fplot('asin',[-1 1]), title('arc-sine'), grid
>> subplot(2,3,6), fplot('atan',[-sqrt(3) sqrt(3)]), title('arc-tangent'),        grid
```

5.5 The bar command

The MATLAB language not only allows the display of points to plot curves, but also offers the possibility of plotting bar graphs and histograms.
To plot a bar graph we use the function **bar** which has the same operating principle as the function **plot**.
Example :

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

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

We also note that MATLAB allows the use of a coordinate system other than the Cartesian system such as the polar coordinate system (for more details look for the functions **compass**, **polar** And **pink**).

5.6 Training exercises

1. Consider the following two functions: ☒

$$f(x) = sin(x-2)+4$$
$$g(x) = -2*x^3+x^2-3$$

- Give the MATLAB instructions necessary to plot the curve of the function f(x) with a variation of x from 0 to 2π, and a step size = π/12.
- Give the MATLAB instructions necessary to plot the curve of the function g(x) with a variation of x from -5 to 5, and a step size = 0.2.
- How to draw the curve of f(x) in green dotted line with diamond-shaped points? and How to draw the curve of g(x) in blue dashed line with square-shaped points?

2. Represent the graph of the function

$$f\colon [1,10] \to \mathbb{R}$$

$$x \mapsto \begin{cases} (\ln(x)+2)^2 & \text{si } \ln(x)-x+2 \geq 0 \\ x^2-4x & \text{si } \ln(x)-x+2 < 0. \end{cases}$$

Chapter 6: Polynomials

Polynomial calculus is the basis of many scientific fields, including digital and analog signal processing, process control, function approximation, and curve interpolation.

6.1 Description

Polynomials are treated as coefficient vectors in Matlab.
For example, the polynomial equation $p(x) = x^2 - 6x + 9$ will be represented by the following vector:

```
>>P = [1  -6 9]
P =
    1   -6    9
```

The number of elements in the vector is equal to the degree of the polynomial + 1.

The main problems with polynomials are:

- the search for roots;
- the assessment;
- adaptation to data.

6.1.1 Roots of a polynomial

The approximate calculation of polynomial roots is performed in Matlab with the roots command.

```
>> roots (P)
years =
          3.0000 + 0.0000i
3.0000 - 0.0000i
```

6.1.2 Evaluation of polynomials

To evaluate a polynomial at a point, we use the function polyval. Let's try to find the value of the polynomial P at 1.

```
>>Polyval(P,  1)
years
4
```

6.1.3 Determining a polynomial from these roots

The coefficients of a polynomial can be determined from its roots using the function **poly**.

For example, we are looking for the polynomial which has roots: 1, 2 and 3.

These can be defined as the elements of a vector r.

```
>>r = [1 2  3]
r =
1     2     3
```

The desired polynomial is then:

```
>>K = poly (r )
K =
1   -6   11   -6
```

Which corresponds to:*$K(x) = x^3 - 6x^2 + 11x - 6$.*
We check that the roots of the polynomial K are 1, 2 and 3.

```
>>roots = roots (K     )
roots =
3,0000
2,0000
1.0000
```

6.2 Graphical representation

To plot the graphical representation of the polynomial **K (x),** Let us define a domain for the variable x which contains the roots of **K**.

Domain of values of the variable**x** and evaluation of the polynomial **K**:

```
      >> x = 0:0.1:4;
      >>      y = polyval(K, x);
```

Plot the function y = K(x):

```
      >> plot(x,y)
>> grid on
      >> title ('plot of y = x^3 - 6x^2 + 11x -6')
>> xlabel('x')
>> ylabel('y')
```

6.3 Operations on polynomials

6.3.1 Multiplication

Multiplication of two polynomials can be done easily with MATLAB.Let two polynomials **P1** And **P2** defined by:

P(x) = x + 2

P2 (x) = $x^2 - 2x + 1$

```
>>P1 = [1  2]
P1 =
1     2
>>P 2 =[1  -2 1]
P2 =
1   -2     1
```

The result of multiplying **P1** by **P2** is the polynomial **P3** which is obtained with the function ***conv***.

```
>> P3 = conv(P1, P2)
P3 =
10-32
```

6.3.2 The division

The division of two polynomials is done by the function ***deconv***. The quotient **Q** and the rest **R** of the division can be obtained as an element of an array.

```
>>[Q, R] = deconv (P2, P 1)
Q =
1   -4
R =
0     0     9
```

```
>>[Q, R] = deconv (P3, P 1)
Q =
1   -2     1
R =
0     0     0     0
```

By dividing P3 by P1, we find the polynomial P2 (the remainder R is zero).

R is the zero polynomial if the division is exact.

6.3.3 Polynomial integration

The function **polyint** returns the coefficients q of the primitive Q(x) of the polynomial P(x):

```
      >> Q = [1 2 0 -3]
      Q =
      1 2 0 -3
  >> polyint(Q)
years =
  0.2500 0.6667 0 -3.0000 0
```

6.3.4 Polynomial derivation

The function **polyder** returns the coefficients q of the polynomial Q(x) derived from P(x):

```
>>Q = [1 2 0  -3]
Q =
12 0  -3
>>polyder(Q )
years
34  0
```

6.4 Polynomial interpolation:

MATLAB interpolates data by a polynomial.

Given vectors X = [-1 0 2] and Y = [0 -1 3], the above command determines the best polynomial interpolation of degree 2 at the points (-1,0), (0,-1) and (2,3).

```
>> p2 = polyfit(X,Y,2);
```

6.5 Training exercise:

1. Write a program that calculates the derivative and roots of a user-inputted polynomial and then draws their graphs on an interval containing all the roots. Hint: Do not use the polyder command.

2. Run the following program:

```
        x = 0:.1:3*pi ;
y = sin(x);      p =
polyfit(x, y, 5)
figure(1); clf
plot(x,y)          hold on
xx = 0:.001:3*pi ;
        plot(xx, polyval(p,xx), 'r-')
```

Chapter 7: Character Strings

Although MATLAB is a high-level, object-oriented scientific computing language, it also offers capabilities for processing strings, dates, and times. For more information on MATLAB string functions, you can consult the online help (help strfun).

7.1 Description

A **chain** A string is an ordered sequence of characters (text, for example).

In Matlab, the **chains** of characters are specified between 'simple quote', for example:

```
>> 'toto'
>> 'The result is:'
>> 'the apostrophe' %to put an apostrophe you must double the single quote
```

7.2 String Manipulation

7.2.1 Concatenation

The assembly of two or more **chains** of characters is called concatenation. One way to perform concatenation in Matlab is to use the brackets [ ]. For example,

```
>> ['The result of ' , 'multiplication' , ' is given below.']
>> debut_phrase = 'The result of ';
>> end_phrase = 'is given below.';
>> operation = 'multiplication';
>> complete_phrase = [start_sentence, operation,
end_sentence]
```

7.2.2 Extracting a piece of the character string

Elements of a can be accessed **chain** of characters in the same way as for a vector.

```
>> alphabet = 'abcdefghijklmnopqrstuwxyz';      >>
alphabet(1) % 1st letter of the alphabet
>>      alphabet (5:10) % from 5th to 10th
```

If the elements to be extracted are of indefinite size but separated by a particular character. We can divide the **chain** of characters using the regexp function. Example:

```
>> listCourse = 'tomatoes 300gr, radishes 200gr, butter 250gr';
```

We divide the **chain** at the comma level

```
>>splitList = regexp(CourseList, ',' , 'split' );
>>splittedList {1}
>>splittedList {2}
>>splittedList {3}
```

Multiple separators can also be specified between brackets (\s is the special character for space)

```
>> splitList2 = regexp(ListCourse, '[,\s]', 'split')
```

7.2.3 Tests on character strings

We will use the ischar function to check that a variable contains a **chain** of characters:

```
>> a = '1';
>> ischar(a) %true
>> a = 1;
>> ischar(a) %false
```

We will use the isempty function to check that a **chain** is not empty:

```
>> a = 'abcd';
>> ischar(a) %yes
>> isempty(a) %non
>> a = '';
>> ischar(a) %yes
>> isempty(a) %yes
>> a = [];
>> ischar(a) %non
>> isempty(a) %yes
>> a = [1,2];
>> ischar(a) %non
>> isempty(a) %non
```

The strcmp function allows you to compare two **chains** of characters. It returns 1 (true) if both **chain**s are identical and 0 (false) otherwise.

```
>> strcmp('toto', 'toto') %true
>> strcmp('toto', 'titi') %false
>> str1 = 'foo'; str2 = 'tutu';
>> strcmp(str1, str2) %false
>> strcmp(str1, 'toto') %true
>> strcmp(str2, 'toto') %false
```

7.3. Functions specific to character strings

Function	Meaning	Example
abs	Converting strings to numbers	>> abs('MATLAB') years =776584766566

double	Transforms a string into double-precision numeric format	>> String='MATLAB';
		>> Double=double(String)
		Double =776584766566

str2double	Converts the number to a string as a double-precision number	>> Ch1='123';>> str2double(Ch1) years =123
bin2dec	Converts a binary number in string form to its decimal value	>> bin2dec('0111') years =7
hex2num	This function provides a conversion of a number written in IEEE hexadecimal form (string) to a double float precision	>> hex2num('400921fb54442d18') years =3.1416
hex2dec	Converting a hexadecimal number (written as a string) into an integer.	>> hex2dec('F89') years =3977
tank	Converts an array of non-negative integers to characters (the first 127 are the ASCII codes for those characters).	>> char([776584])years = MAT
setstr	Conversely, knowing the ASCII codes of the characters in a string, we can obtain the character string corresponding by the setstr function	>> Code=abs('SIMULINK') Code =837377857673 7875 >> setstr(Code) years = SIMULINK

num2str	Transforms a number into a string.	>> ch1 = 'MATLAB';
		
		>> Version = 2009;

		>> String = [ch1, 'Version', 'R', num2str(2009),'a'] Chain = MATLAB Version R2009a
mat2str	Transforms the numeric matrix into a string.	>> MatString=mat2str([5 4 7; 8 4 2; 6 0 8]) MatString = [5 4 7;8 4 2;6 0 8]
str2num	Transforms a number seen as a string into a number.	>> Ch='1+3.14'>> str2num(Ch) years =4.1400
int2str	Converts an integer to a string.	>> int2str([23 56;21 9]) years =2356 219
dec2hex	Converting an integer to a hexadecimal number given as a string.	>> dec2hex(77)years = 4D
dec2bin	Converts the given integer argument to binary as a string.	>> dec2bin(3) years = 11
isspace	Returns 1 if the character is a space and 0 otherwise.	>> isspace('Mat lab') years =0001000
isstr	returns 1 if the argument is a string and 0 otherwise	>> isstr('MATLAB is a language of high-level programming') years =1

isletter	Returns an array of 0s and 1s; 1 for letters of the alphabet including letters with accents, and 0 for the rest of the characters	>> String=['è5f87';'çyéfg'] >> isletter(String) years = 10100 11111
strcat	Concatenates a sequence of strings given as arguments.	>> strcat('MATLAB','R2009a') years = MATLAB R2009a
strvcat	Vertical concatenation of character strings, independently of their lengths.	>> strvcat('MATLAB & SIMULINK','R2009a') years = MATLAB R2009a
strcmpi	Compares strings ignoring upper and lower case.	>> TestIdent=strcmpi('MATLAB','matlab') TestIdent =1

findstr	Research of a chain in another	>> String1='TLAB';
		>> String2='MATLAB';
		>> findstr(String1, String2)
		
		years =3

strfind	Looks for clues where one string is within another.	>> findstr(String2,'A')years =25
upper	Transforms respectively a string in uppercase letters	>> Shift=upper('MATlab') Shift = MATLAB
lower	Transforms respectively a lowercase string	>> Minus=lower('MATlab') Minus = matlab
Ch = strrep(Ch1,Ch2,Ch3)	replaces all occurrences of the string Ch2 in Ch1 with thechain Ch3.	>> Ch=strrep('This is an average level example','average','very good') Ch = This is an example of a very good level

strtrim	Removes spaces before the string.	>> String='MATLAB R2009 and SIMULINK' >> String2=strtrim(String) Chain2 = MATLAB R2009 and SIMULINK

strtok	Returns the first word of a string, plus the rest. Fields are separated by a delimiter, which defaults to a space.	>> String = 'MATLAB R2009a';
		>> [FirstWord, rest] = strtok(String)
		FirstWord = MATLAB
		
		remainder = R2009a

evaluation	Evaluating a string	>> x = [2*pi pi/3; pi/2 -1] >> String = 'exp(-det(sin(x)))'
	container of the orders MATLAB, operations and variable names	>> eval(String) years =2.3774
poly2str	Returns a polynomial as a character string	>> p=poly2str([1 -2 4],'x')p =x^2 - 2 x + 4

7.4 Training exercises

- A palindrome is a word or phrase that can be read from left to right or vice versa. We propose to write a Matlab script that determines whether a string is a palindrome. The solution would be to reverse the string and compare it with the original string.
- Write a Matlab script that extracts day, month, and year information from a date as a string with the formats 'dd-mm-yy' or 'dd/mm/yy'.