Laboratory 2: MATLAB FUNDAMENTALS

OBJETIVES

- Familiarize the students with Matlab.
- Learn to use basic commands for data manipulation, definition, numerical calculations and visualization.

DESCRIPTION

What is MATLAB?

MATLAB is a high level programming language, with a direct focus towards scientific computation. More explicitly, MATLAB is a program used for numerical calculation with a vast number of pre-programmed commands formulated to solve scientific problems. From this point of view, MATLAB can be considered as a powerful programmable scientific calculator.

The name MATLAB comes from the words *MATtrix LABoratory*, because many of the computations are performed in the form of vectors or matrix, although it can also deal with scalars and complex numbers. Some of MATLAB applications are:

- Computation and mathematics
- Development of algorithms.
- Modeling and simulation of systems.
- Exploration, visualization and data analysis.
- Creation of scientific plots.

These MATLAB applications make this program a great working tool widely used by engineers.

Working in MATLAB environment

MATLAB can be accessed as any other **Windows** application, by clicking twice over the "Desktop" or from the **Start** menu. In both cases, a window similar to the one shown in Figure 1 is opened. The most important components of the MATLAB environment are the **Command Window, Workspace browser** and the **Command History**.

Command Window. It's the window where all instructions (or commands) are executed. In this window appears the MATLAB **prompt** (>>). This means that the program is ready to receive the next command.
Workspace Browser. This window lists all the data variables that have been created and which are allocated in the programs memory.
Command History. This window lists all the commands previously entered.

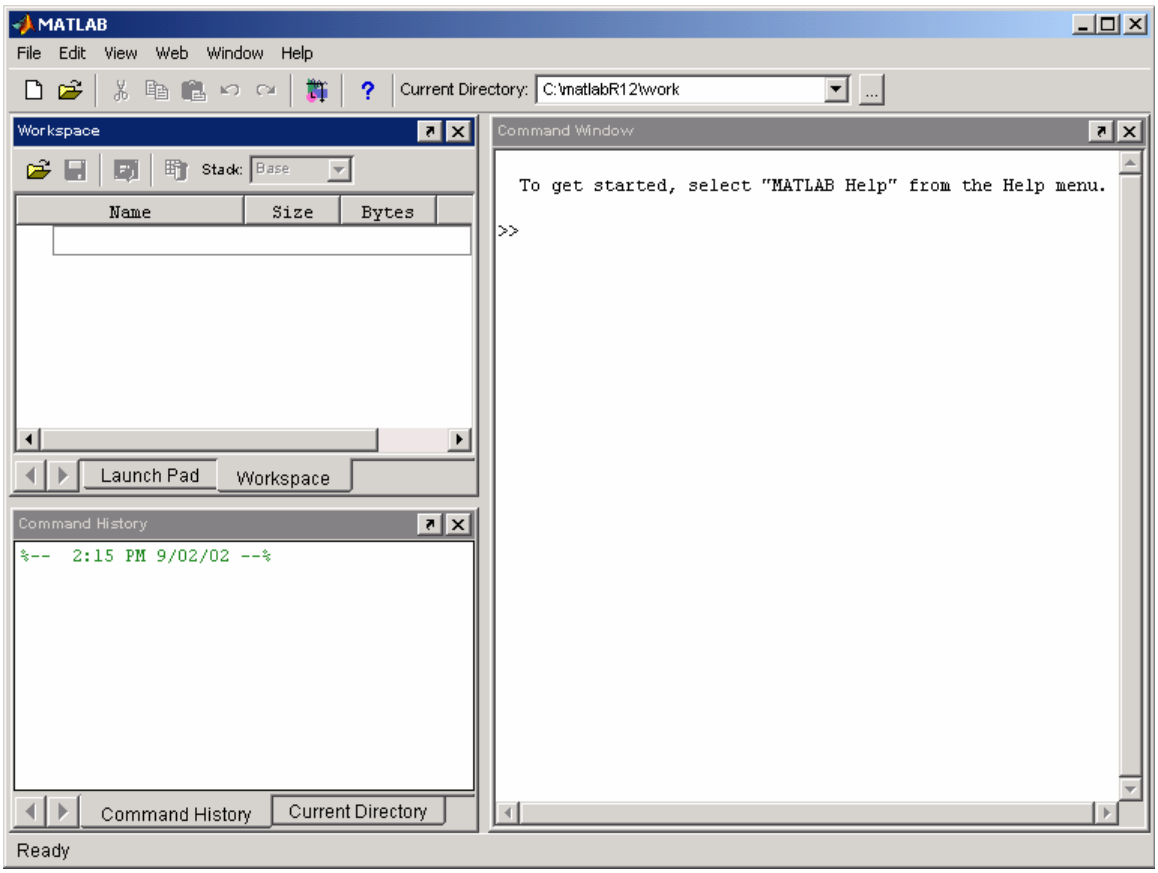

Figure 1. Matlab *Command*, *History* and *Workspace* windows

Numerical Computations

Before studying some MATLAB numerical applications, it is convenient to be familiarized with the types of expressions (numbers, variables and operators) that are used.

As we mentioned before, in the command window appears the **prompt**, >>, which indicates we can enter any instruction or command. For instance, we could enter a numerical expression. MATLAB will produce an answer `` **ans="** followed by the result of the operation. For example

```
>> 3+4
ans =
     7
>> 2*(5+3*(1+4))
ans =
    40
```

Basics operations between numbers are written with the following notation: addition (+), subtraction (-), multiplication (*), division (/) and to the power (^). For example,

```
>> 2+3.5^(1.7*2)
ans =
```

```
72.7671
```

MATLAB has pre-defined functions such as the logarithm, roots, trigonometric functions, inverse, etc., such as those found in a scientific calculators. See Table 1.

Table 1.

Basics Functions		Basic Trigonometric Functions	
sqrt	Square root	sin	Sine
exp	Exponential Function	con	Cosine
log	Natural Logarithm, e	tan	Tangent
log10	Logarithm Base 10	asin	Arcsine
round	Round to closest integer	acos	Arccosine
abs	absolute Value	atan	Arctangent

For example,

```
>> sqrt(4)
ans =
     2
>> log10(100)
ans =
     2
```

Exercises
Compute the following quantities and write your answer (Note: The operations are written in mathematical notation, In order to compute it, you would need to use Matlab functions).
ln(2): _____
log(5): _____
sin(π/4):_____
$\sqrt[3]{-1}$:_____
cos(arctg($1/\sqrt{2}$)):_____

Definition of Variables
A variable is simply a label that we temporary assign to a number. For instance, >>
```
a=1;
>> b=2;
```

From now on, **a** and **b** represent the quantities 1 and 2 respectively. Now we can perform operations with these variables:

```
>> c=a+b
c =
    3
```

If we simply want to know the value of one of the variables, we type the name of that variable. Example;

```
>> a
a =
     1
```

The name of the variables can be formed by letters, numbers and the underscore character ``_", up to a maximum of 19 characters and the first character should always be a letter.

When we want to recall all the variables that has been declared and used at any moment, or the instructions that have been used, we can look at the **Workspace & Command History** windows, as shown in Figure 2.

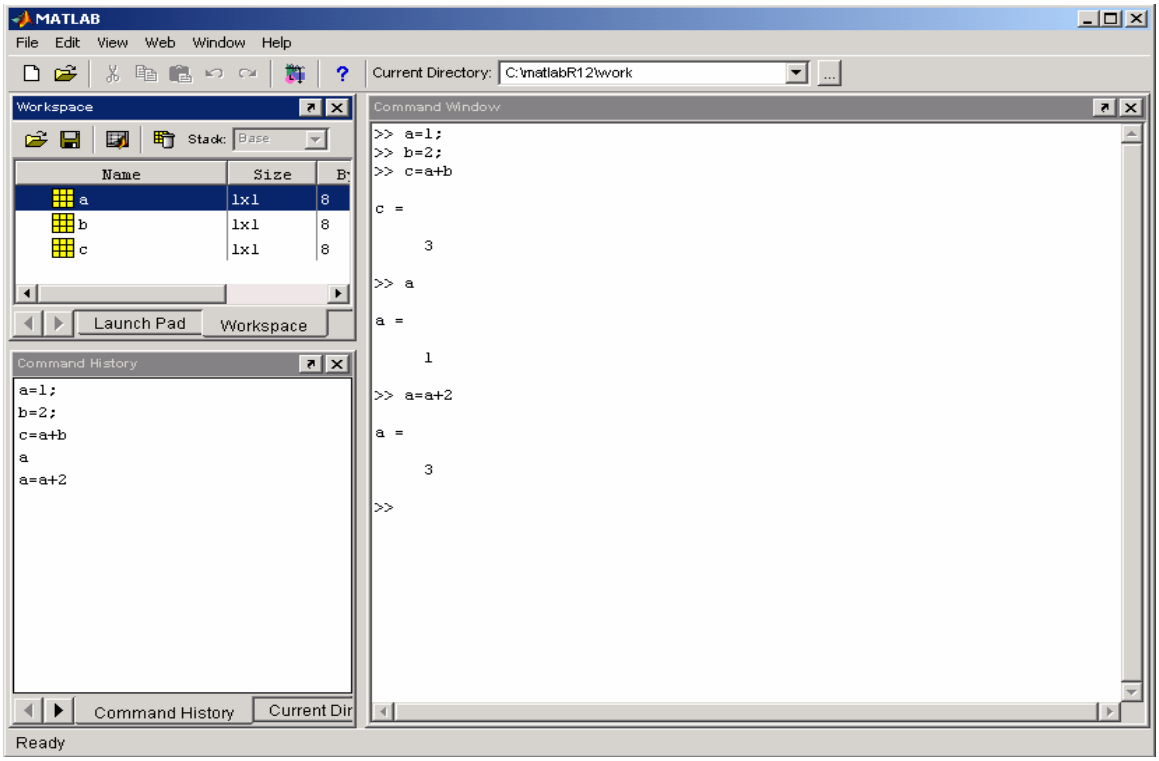

Figure 2

Complex numbers

MATLAB can also work with complex numbers. To define a complex number in MATLAB, let's write the real part plus the imaginary part with the letter "i" (imaginary unit).

```
>> 0.5+2i
ans =
   0.5000 + 2.0000i
```

Exercises

Compute the following quantities and write your answer. Due *a=2-3i* and *b=1+2i*

a+b: _____

a/b: _____

a*b: _____

Special Variables

There are some special variables in MATLAB that are pre-defined when the program starts. For instance:

ans	Stores the result of the last expression to which a variable has not been assigned.
pi	Stores the constant π
i or j	Represents the imaginary unit
Inf or inf	Represents infinity
NaN or nan	Represents some undetermined value, means "Not a Number". Such as 1/0

There also exist commands used to round real numbers to the nearest integer. The following commands are common examples:

Table 3

floor(X)	Rounds the elements of X to the nearest integers towards minus infinity.
ceil(X)	Rounds the elements of X to the nearest integers towards infinity.
round(X)	Rounds the elements of X to the nearest integers.
fix(X)	Rounds the elements of X to the nearest integers towards zero.

Examples:

```
>>floor(pi)
ans =
          3

>>ceil(pi)
ans =
          4

>>  y=fix(pi)+5
y =
          8
```

Common Logic Conditions

Table 4

<	Less than	= =	Equal to
>	Greater than	~=	Not equal to
>=	Greater than or equal to	&	AND,(requires 2 or more conditions be true)
<=	Less than or equal to	|	OR (requires for at least 1 condition to be true)

These operations are those that compare the values among variables. These conditions can be applied to any matrix or scalar and the result will be fixed in a series of 0's and 1's. The zeros indicate that the condition is false, while the ones indicate that the condition is true.

Examples:

The condition x == 4 is true when the variable x is equal to 4.

```
>>x = 0:4              x = [ 0     1     2     3     4 ]
>>y=(x==4)             y = [ 0     0     0     0     1 ]
```
(The condition is true for x = 4, but it is false for x= 0,1,2, and 3.

Exercise:

If R is between 5 and 10 (`R =5:10`) and S is equal to R, less or equal to 7; What are the values that make S true? Complete the following Matlab procedure:

```
R = 5:10                    R= [   ,   ,   ,   ,   ,   ]
S = (             )         S= [   ,   ,   ,   ,   ,   ]
```
The values that make S true are: _____.

Vectors & Matrices

The variables not only can be numbers, they can also represent matrices or vectors. In fact, for MATLAB, numbers are simply a 1x1 matrix. Therefore, a single number is the smallest matrix.

To define a matrix, open a bracket,"**[**" , enter the matrix elements one by one and then close with another bracket "**]**". To separate the elements in a row use a space or a comma, and to start a new row, use the semi colon or the Enter key. See the next two examples:

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

a =
     1     2     3
     4     5     6
     7     8     9
```

will yield the same result as

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

A vector is defined as a row,

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

or column.

```
>> d=[1;2;3]
d= 1
   2
   3
```

To extract the value of an element in a vector or matrix, simply indicate its position as shown below.

```
>> c(1)
ans =
       1

>> a(3,1)
ans =
     7
```

Matrix Operations

Two matrixes can be multiplied and added given that the matrix have the appropriate dimensions. They can also be multiplied by a scalar and can be raised to some power if they are squared. For instance, let's add the matrices defined before.

```
>> a+b
ans =
    2    4    6
    8   10   12
   14   16   18
```

```
Where a and b were both 3x3
matrices given by
     1     2     3
     4     5     6
     7     8     9
```

Exercise
Multiply and subtract two matrices of same dimensions (such as **a** and **b**)
a*b= a-b=

Other operations that can be performed on a matrix are **inv** (inverse of the matrix), **conj** (conjugate), **'** (transpose).

Some other functions to operate vectors and matrix are

Functions for vectors		Functions for matrixes	
length	Returns the maximum value in between the number of rows and columns.	**size**	Size of the matrix
Max	maximum value	**det**	Determinant de the matrix
Min	minimum value	**sum**	Sum of elements
Mean	average value	**expm**	Matrix Exponential

Exercise
Given *a* = ***[3 2 1 ; 6 5 4 ; 9 8 7]*** and ***b = 1./a;*** Determine the following information of matrix **b**:
minimum value :_____
maximum value :_____
average values:_____
size:_____.

Graphs

One of the fundamental utilities offered by MATLAB is the graphic representation of data and functions. For example, we can create bar graphs, step graphs, graphs in polar coordinates, two-dimensional, and tri-dimensional; very similar to those obtained with **Microsoft Excel**

The more common instruction used to graph is **plot.** If **x** is a vector, the command **plot(x)** will plot the values of the components stored in the vector with respect to its position. For example, when performing the following command, one obtains the graph represented in Figure 3.

```
>> x=[2 3 6 -1 0 5 5 4 -1];
>> plot(x)
```

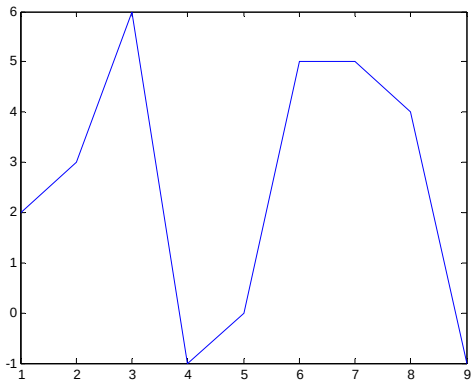

Figure 3

If **x** and **y** are two vectors of same lenght then **plot(x,y)** represents the vector **y** as a function of **x**.

```
>> x=[1;1.5;2;2.5;3;3.5;4;4.5;5];
>> y=[1;2.25;4;6.25;9;12.25;16;20.25;25];
>> plot(x,y)
```

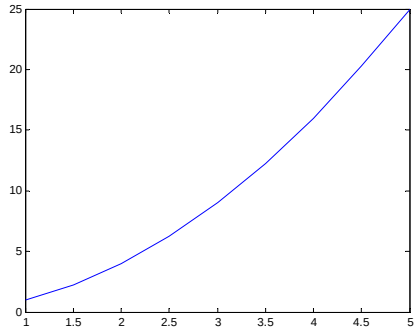

Figure 4

We can also plot functions, for example

```
>> x=0:0.1:2*pi;  %x varies from 0 to 2pi
>> y=cos(x);      %compute the y values
>> plot(x,y);     %plot y versus x
```

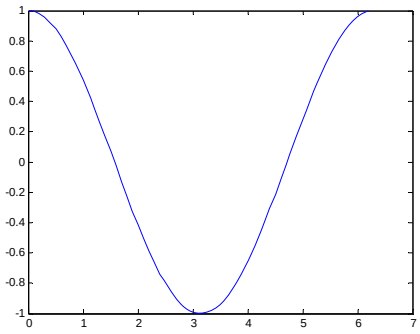

Figure 5

To place labels or titles to the graphs we can use the following functions:

```
>> xlabel('variable x')          % Place a label to the horizontal axis
>> ylabel('variable y')          % Place a label to the vertical axis
>> title('graph de y vs x')      % Place a title to the graph
>> gtext('cos(x)')               % Place text on the graph
```

If it is desired to apply colors or other styles to the graph, we can use some of the following commands:

y yellow	g green	* stars
m magenta	b blue	. dots
c cyan	w white	-- dashed
r red	k black	- solid

These commands are executed in the following manner:
plot(x,y,'r')

Exercise
Represent the followings functions in the given intervals.
$y1=x1^2$ x1 varies in the range [-1,2], with increase of 0.1.
$y2 = e^{-x2^2}$ x2 varies in the range [-2,2], with increase of 0.1.

Matlab Applications

Digital Image Processing

This example shows how to improve the state of an image that has been polluted with noise using MATLAB. The image is obtained from the camera located on your computer display using the program **Logitech.** Launch **Quickcam** found in the **Start** menu, and save the picture with the name "**mifoto.jpg**' in the folder **MatlabR7.0>work**. If you desire to save the photo under another name, then you must change the name in the MATLAB sub-routine. For this, write **edit modulo** in the **Command Window**, then change the name as **original=imread('preferred name')** and save. A program designed in MATLAB is saved with the file extension "**.m**", thus the common name **M-file**, which can be seen in the **Command Window** by typing "`ls`".

```
original=imread('mifoto.jpg');                  %import image
bwimage=rgb2gray(original);
dirt=imnoise(bwimage,'salt & pepper',0.5);   %apply noise (variable called "dirt")

clean = medfilt2(dirt);                         %filter image
%[color,map]= gray2ind(clean,64);              %convert grayscale to indexed color image

imshow(original)                                %Display Images
pause
figure,imshow(bwimage)
```

```
pause
figure,imshow(dirt)
pause
figure, imshow(clean)
%pause
%figure, imshow(color)
```

Filtering Audio signals

The following code makes reference to a Butterworth low-pass filter designed for digital filtering of audio signals.

```
% Declare the specifications of the filter
wp=0.05;                          % passband edge frequency
ws=0.07;                          % stopband edge frequency
ap=1;                             % peak passband ripple
as=20;                            % minimum stopband attenuation

[x,ft]=wavread('señal.wav');      % read the input signal
wavplay(x)

[n1,wo]=buttord(wp,ws,ap,as);     % Determine the order of the filter
[b1,a1]=butter(n1,wo);            % design the digital low-pass filter
tf(b1,a1,1/ft,'variable','z^-1')  % transfer function
pause

y=filter(b1,a1,x);                % filter the input signal.
wavplay(y)
```