# UNIT 1

# MATLAB Fundamentals

**1. Introduction to MATLAB** ........ **2**
**2. The MATLAB environment** ........ **3**
*2.1 Windows and desktop* ........ *4*
*2.2 Variables and workspace* ........ *6*
*2.3 Files and search path* ........ *9*
*2.4 Helps and demos* ........ *11*
*2.5 Relationship with Excel* ........ *12*
**3. Programming language** ........ **14**
*3.1 Basic symbols and commands* ........ *14*
*3.2 Mathematical functions and operators table* ........ *18*
*3.3 Logical operators* ........ *19*
*3.4 Functions created by the user* ........ *20*
**4. MATLAB graphics** ........ **21**
**5. Using toolboxes** ........ **22**
**6. Final comments and instructions for proposed exercises** ........ **24**

# 1. Introduction to MATLAB

*History:* MATLAB was originally written by Cleve Moler, founder of MathWorks Inc., with the aim to provide easy access to the matrix software developed in the UNIX projects EISPACK (Eigen system package) and LINPACK (linear system package).

*Versions and history:* The first version was written in the late 1970s in Fortran. The matrix, which did not need any *a priori* pre-sizing, was its unique data structure. The name "MATLAB" derives from MATrix LABoratory.

MATLAB was rewritten in C in the 1980s. Version 3 for MS-DOS appeared in the early 1990s. Version 4 for Windows 3.11 appeared in 1993 and included the first version of Simulink. The current version is the 8.x (the latest versions are identified with the name of the year: R2008a, R2008b, R2009a, etc.).

MATLAB is currently a *de facto* standard in engineering. Dedicated conferences are held at many universities and several companies sell their toolboxes as third parties. Many users also share their programs on the Internet with free access. For more information, refer to www.mathworks.com.

*Main features:*

- The MATLAB language is simple, yet powerful and fast. In a typical working session, it is not necessary to compile or to create executable files. Since M-files are text files, the memory requirements are small.

- A number of functions for mathematical operations and for signals and systems analysis are already implemented and available in commercial toolboxes. Users can access the corresponding m-files if they want to modify them. They can also create their own functions and specific toolboxes.

- MATLAB provides powerful tools for visualizing graphics in two and three dimensions, including effects and animations.

- MATLAB allows for the development of complex applications using graphics user interface (GUI) tools.

- MATLAB can communicate with other languages and environments and with hardware components such as sound cards, data acquisition cards and digital signal processors (DSPs).

*Constitutive parts:* There are three main parts.

- Environment (windows, variables and files).
- Graphic objects (see Unit 2).
- Programming language (see Unit 3).

This unit presents the MATLAB environment.

## 2. The MATLAB environment

The environment consists of a set of tools for working as a user or as a programmer. These tools can be used to import/export data, create/modify files, generate graphics and animation effects, and develop user applications.

*Windows:* There are several window types. The desktop includes the main windows that correspond to the MATLAB core. However, a number of secondary windows are opened and closed in a typical session in order to show figures, user interfaces, variable and file editors, SIMULINK models and libraries, etc. There are also specific help and demo windows.

*Variables:* They constitute the temporary objects (when the user exits MATLAB, all variables are deleted) and are stored in the workspace.

*Files:* Steady objects (not deleted when MATLAB is closed) that have their own text editor and folder structure. In addition to the basic files that constitute the basic core of MATLAB, there are user-created files and commercial files corresponding to the libraries, also called toolboxes. A toolbox is a set of files developed for specific applications, e.g., the *Curve Fitting Toolbox* is designed to find mathematical expressions that fit arbitrary curves.

There are also two “special” toolboxes whose hierarchy is above MATLAB (see Fig. 1): *Simulink*, for the numerical simulation of systems (dynamical, communication, etc.) and *Stateflow*, for the simulation of state machines.

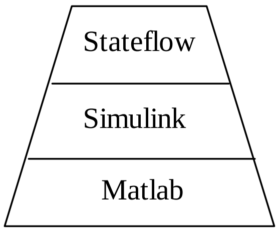
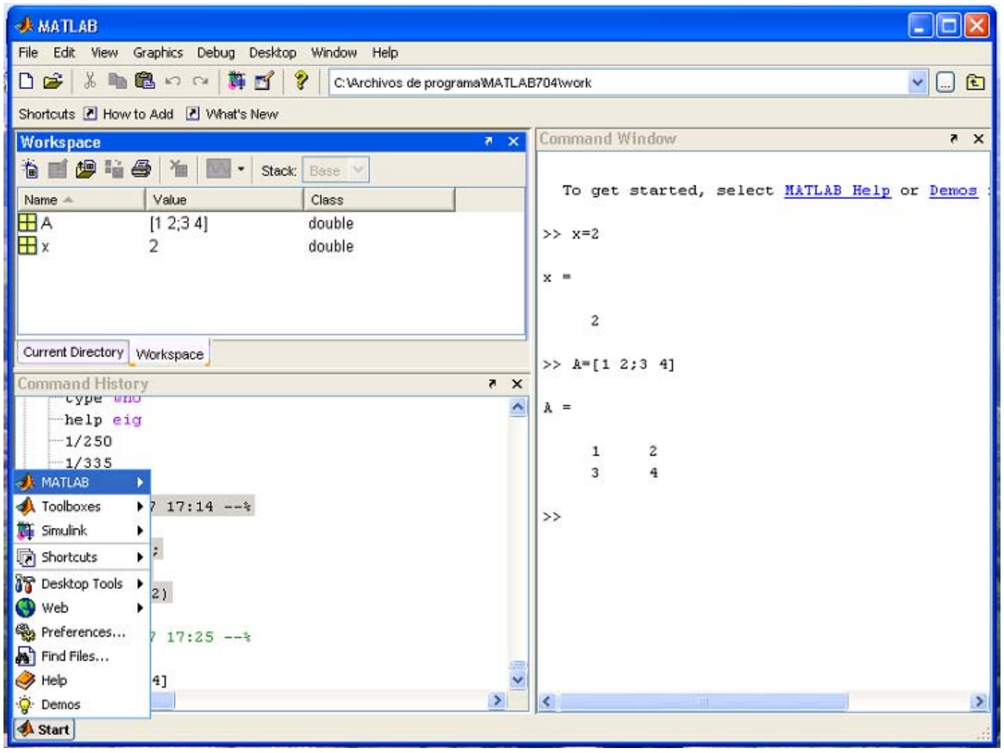


**Fig. 1.** MATLAB modules

## 2.1 Windows and desktop

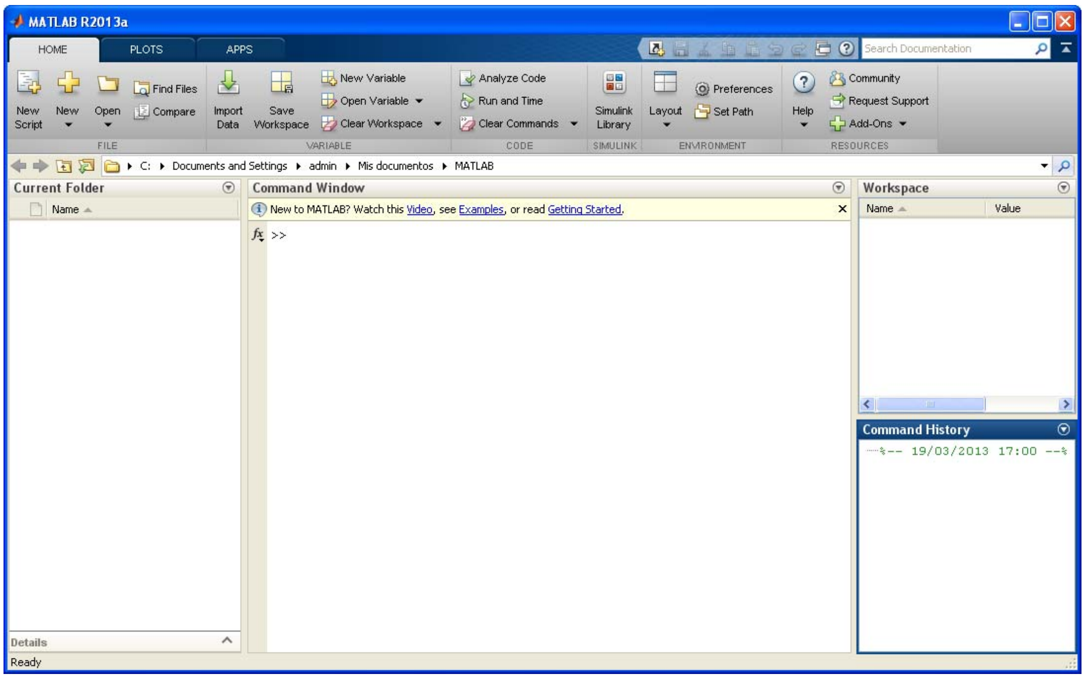
*Desktop:* When MATLAB is started, a desktop opens as shown in the following figure (version 7.0.4).



```
type who
help eig
1/250
1/335
...7 17:14 --%
...7 17:25 --%
```

```
  To get started, select MATLAB Help or Demos

>> x=2

x =

     2

>> A=[1 2;3 4]

A =

     1     2
     3     4

>>
```

**Fig. 2.** MATLAB desktop in version 7.0.4



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

```
fx >>
```

```
%-- 19/03/2013 17:00 --%
```

**Fig. 3.** MATLAB desktop in version 8.1.0.60

*Command window:* The main window. Commands and statements are written after the prompt >>. Operating system instructions can be executed from this window simply by typing the sign "`!`" after the prompt (example: >>`!dir`). The <↑> key can be used to recover previously typed statements.

*Other desktop windows:* The default desktop also includes the following windows.

- *Command History*: Contains the commands used in previous MATLAB sessions, indicating the date and time.
- *Current Directory*: Lists the files and folders in the current directory. By default, the selected current directory is <work>.
- *Workspace*: Shows the current variables as well as their type and value.

Other windows include:

- *Launch Pad:* A version 6 window that lists the toolboxes. This window does not appear in version 7 since the list of toolboxes is available from the Start button in this version.
- *Help:* Contains help documentation for the MATLAB itself and for different toolboxes.
- *Profiler*: Provides information about the resources used by the different functions in order to help in the code debugging.

Notice that the options available in the menu bar vary according to the window selected.

*Desktop layout:* It is possible to change the desktop layout. The available options can be explored in the menu bar and the *Start* button. Changes to the desktop can be saved (*Desktop* → *Save Layout*). The corresponding file is `startup.m`. If this file exists, it is one of the first ones loaded when MATLAB is launched. Another initialization file is `matlabrc.m`, which contains the default values concerning fonts, colours, dimensions, etc. of the different program elements. Preferences can be adjusted from *File* → *Preferences…*

*Secondary windows:* Apart from desktop windows, other windows open and close during the session as commands are executed (or controls are activated) to generate plots, execute demos, run SIMULINK models, etc.

*Other commands:* Other useful commands are

- `clc`: close command window
- `close [all] [all hidden]`: close (all) figure windows
- `exit, quit`: exit MATLAB
- `diary [on/off]`: export a session on a text file
- `ver`: list the installed toolboxes

For more information, type >>`help command_name`.

**Exercise 1. Desktop**

**1)** Start MATLAB and identify the following windows: *Command Window*, *Workspace*, *Current Directory*, *Command History*.

**2)** Add the *Help* and *Profiler* windows using the `Desktop` option in the main menu bar. Use the ↘ and ↗ buttons to dock/undock windows to/from the desktop. Click on × to close the windows.

**3)** See which other desktop configuration options are available in the `Desktop` option in the menu bar.

**4)** Switch to the default desktop: `Desktop` → `Desktop Layout` → `Default` (this is the recommended layout).

**5)** Select the command window by clicking inside it. Look at the different options in the main menu (`File`, `Edit`, `Debug`, `Desktop`, `Window`, `Help`). Identify their purpose.

**6)** Repeat the previous task with the toolbar buttons.

**7)** Repeat the previous task with the Start button. Notice that the same functions can be executed from the menu bar, the toolbar or the Start button.

(Note: This exercise is for practice purposes only. It is not necessary to submit it to Moodle.)

## 2.2 Variables and workspace

*Workspace:* Within a session, the variables generated by the commands are stored in the workspace. Workspace variables can be modified and/or used in other commands. This storage is temporary and only for the current session. Workspace variables are erased when you exit MATLAB.

*Array Editor:* To view the contents of a variable **`var1`**, simply type its name in the command window **`>>var1`**. You can also display its contents in the *Array Editor*. To open it, simply go to the *Workspace* window and double click on the icon of the variable in question ⊞var1. Another way to open the *Array Editor* is to type **`>>open var1`** in the command window.

*Importing and exporting variables:* The workspace variables can be saved in a data file (`save` command) and this file can then be loaded in a later session (`load` command). The data files in MATLAB have the extension `*.mat`. The default name for the data file is `matlab.mat`.

*Variable names:* If a name is not assigned to a variable, it receives the default name `ans` (for *answer*). The next unnamed variable will also be called `ans`, so it will substitute the first one.

MATLAB is case sensitive. Lowercase and uppercase names refer to different variables. Hence, `a` and `A` correspond to two different variables.

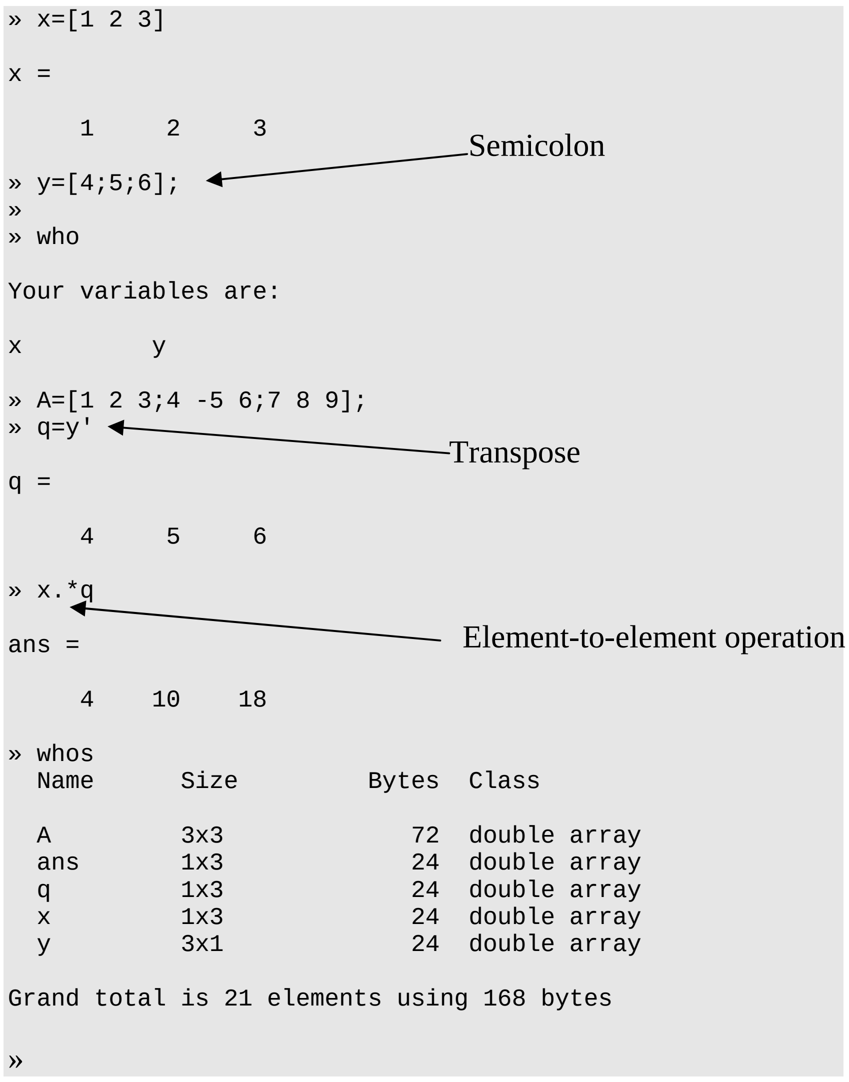
It is recommended not to assign a name to a variable that refers also to a function. A common error is to create a variable with the name `axis` and then to try to execute the function `axis`. In such a case, MATLAB gives an error message because it considers `axis` to be a variable, not a function. To solve this problem, type `>>clear axis`. To check if a name corresponds to a function, you can simply type: `>> help name`.

*Other commands:* Other useful commands include

- `who,whos`: to see the variables in the workspace
- `size`: to see a matrix dimension (`>>[nrow,ncol]=size(A)`)
- `length`: to see a vector length (`>>N=length(v)`)
- `clear [all]`: to erase variables (`>>clear` erases all of them, `>>clear var1` only erases the variable `var1`)
- `why`: to see the programmers' sense of humour, type `why` several times

For more information, type `>>help command_name.`

**Example 1. Command window and workspace**



```
» x=[1 2 3]

x =

     1     2     3

» y=[4;5;6];
»
» who

Your variables are:

x         y

» A=[1 2 3;4 -5 6;7 8 9];
» q=y'

q =

     4     5     6

» x.*q

ans =

     4    10    18

» whos
  Name       Size         Bytes  Class

  A          3x3             72  double array
  ans        1x3             24  double array
  q          1x3             24  double array
  x          1x3             24  double array
  y          3x1             24  double array

Grand total is 21 elements using 168 bytes

»
```

**Exercise 2. Variables and workspace (I). Entering and viewing data**

**1)** Variable generation: In the command window, enter a scalar, a matrix, a character string and a couple of commands. For instance:

```
>> x=2
>> A=[1 2;3 4;5 6];
>> A
>> s='hello'
>> a=2/0
>> 0/0
```

Notice that the typed commands are saved in the command history window.

Answer the following questions.

What is the semicolon for?
Does MATLAB distinguish between lowercase and uppercase?
If you type `>> y=40.5` and later on `>> y=102.3`, what happens to variable `y`?
What does `ans` mean?
What does `NaN` mean?
Now type `>>who` and `>>whos`. What do these two commands do?

**2)** Workspace window: Click on the workspace window to select it. Notice that it contains the created variables. Look at the options available in the menu bar (`File`, `Edit`, `View`, `Graphics`, `Debug`, `Desktop`, `Window`, `Help`) and the toolbar buttons.

**3)** Array editor window: Open the array editor window by double-clicking on a variable name or on the icon. See the value of the different variables. Take a general look.

(Note: This exercise is for practice purposes only. It is not necessary to submit it to Moodle.)

**Exercise 3. Variables and workspace (II). Save and load.**

**1)** Saving variables: To save all workspace variables in a `*.mat` file, click on . The default name is `matlab.mat`. Notice that the file appears in the Current Directory window.

**2)** Deleting variables: Select some variables and delete them by clicking on .

**3)** Loading data: Click on and select any data file (`*.mat`, `*.xls`, `*.txt`,…) to open the Import Wizard window, which enables a file or a part of one to be loaded to the workspace. Reload the previously deleted variables.

**4)** Text commands (`clear`, `save`, `load`). All previous operations can also be performed by using commands typed on the command window. For instance:

To save variables `a` and `A` to the file `bah.mat`:

```
>> save bah a A     or     >> save('bah','a','A')
```

Delete `s`

```
>> clear s
>> who
```

Delete all variables

```
>> clear, who
```

Load a data file `bah.mat`

```
>> load bah, who
```

For more details, type `>> help command_name`

(Note: This exercise is for practice purposes only. It is not necessary to submit it to Moodle.)

## 2.3 Files and search path

While variables are the temporary information of MATLAB, files constitute the permanent information that is not deleted when a session is closed.

*File types:* There are several types of files.

- Data files: Files with extension `*.mat`. Several formats are available (ASCII, binary, etc.) and they are created and loaded by executing the `save` and `load` commands, respectively, or using the options on the workspace window menu bar. The default name for data files is `matlab.mat`.

  MATLAB can also import data from other types of files (`*.txt`, `*.xls`, etc.). If the file in question is in the Current Directory, simply click on its icon to open the Import Wizard (see Exercise 6). Data can also be accessed from a file through the command window, for example, `>>ImportData('test.txt')`. Other related functions are `fopen`, `fread`, `fprintf`, ...

- M-files: Text files with the extension `*.m` contain commands as if they were written in the MATLAB command window. M-files can be created by the user and are stored by default in the working directory `<work>`. To edit the M-files, it is advisable to use MATLAB's M-file editor. However, any other text editor (for example, Windows Notepad) is also valid.

  Commercial M-files are organized in libraries named toolboxes (each toolbox is stored in a different folder) and the user can edit and modify them, too.

  In general, each command or MATLAB function corresponds to a file. For instance, the `roots` function corresponds to the `roots.m` file.

- Built-in files: These files cannot be modified by the user, their extensions are diverse (`*.dll`, `*.exe`, `*.mex`) and they correspond to the files of the MATLAB kernel (for example, the code of function `who` is not available to the user. The file `who.m` only contains the help of this function).

*Names for files:* MATLAB is case sensitive, e.g., see the difference between `>>who` and `>>WHO`. In general, functions should be typed in lowercase.

A name like `exercise1-2.m` is not valid because MATLAB will try to take `2` from the "variable" `exercise1`.

*Search path:* The user created files are saved by default in the <work> directory, but you can save files anywhere in the directory tree. To force MATLAB to consider the new location, we have to widen its path. To do so, select `File` → `Set Path...` in the main menu. Or click on the icon ... next to the Current Directory in the desktop toolbar.

You can also use the function `addpath` to make MATLAB take into account the functions of specific directories for the current session. For example:

```
addpath c:\matlab6p5\work\my_folder
addpath d:\projects
```

*Scripts and functions:* The power of MATLAB relies on the possibility of executing a large series of commands stored in a file. Such files are the so-called *M-files* since their name extension is *m*, *filename.m*. Apart from the commercial files provided in the toolboxes, users can create their own files. M-files can be script files or function files:

- Scripts declare neither input arguments nor output arguments. To execute script files, simply type their name (without an extension) in the command window or click on the (save and run) button in the M-file editor.

- Functions have input and output arguments declared. The function name is usually written in lowercase. Input arguments are inside parentheses ( ) while output arguments are inside brackets [ ].

*Other commands:* Other useful commands related to files are

| | |
|---|---|
| `what`: | lists the files that are in the current directory |
| `which`: | to see the complete path for a function (`>>which bode`) |
| `lookfor:` | to search for functions (see next example) |
| `type`: | to see the M-files code (`>>type roots`) |

For more information, type `>>help command_name.`

**Example 2.  Looking for functions**

If we want to compute the determinant of a matrix but we do not remember the particular MATLAB function, the function `lookfor` can be used (this function searches the required word in all of the names and help sentences of the functions).

```
>> lookfor determinant
DET    Determinant.
DET    Symbolic matrix determinant.
DRAMADAH Matrix of zeros and ones with large determinant or inverse.
>>
>> help det

 DET    Determinant.
    DET(X) is the determinant of the square matrix X.
    Use COND instead of DET to test for matrix singularity.

    See also COND.

 Overloaded methods
    help sym/det.m

>>

>> det([1 3;-2 4])

ans =

    10

>> type det
det is a built-in function.
>>
```

---

**Exercise 4. Toolboxes and search path**

**1)** Identify the default working folder (see Current Directory window or toolbar).

**2)** Identify which MATLAB version is installed and which commercial toolboxes are available. To do so, type >>`ver` in the command window.

**3)** Verify that toolboxes also appear in the Start button.

**4)** See the folder tree by using the menu options `File` → `Set Path…` See where the files that correspond to the different toolboxes are stored.

**5)** Add any folder to the MATLAB path.

**6)** Use `which` to determine in which folder the file that corresponds to function `fminsearch` is stored. Idem with function `roots`.

**7)** See what functions `fminsearch` and `roots` are for (function `help`) and see if their code can be edited by the user (function `type`).

(Note: This exercise is for practice purposes only. It is not necessary to submit it to Moodle.)

## 2.4 Helps and demos

MATLAB presents several help levels. All functions and installed toolboxes can be listed on the first level by using the command:

```
» help
```

On the second level, information about all of the functions of a particular toolbox can be retrieved by typing the command:

```
» help toolbox_name      (» help stats)
```

Finally, to obtain complete information about the usage and syntax of each function, use:

```
» help command_name      (» help bode)
```

In addition, the demonstration files (demos) are a set of scripts that offer a perspective of the toolboxes through the automatic execution of the most representative functions.

**Exercise 5. Helps and demos**

Choose any toolbox and use the Start button to enter the toolbox help.
Run a demo from the basic MATLAB and a demo of any toolbox of your interest.

(Note: This exercise is for practice purposes only. It is not necessary to submit it to Moodle.)

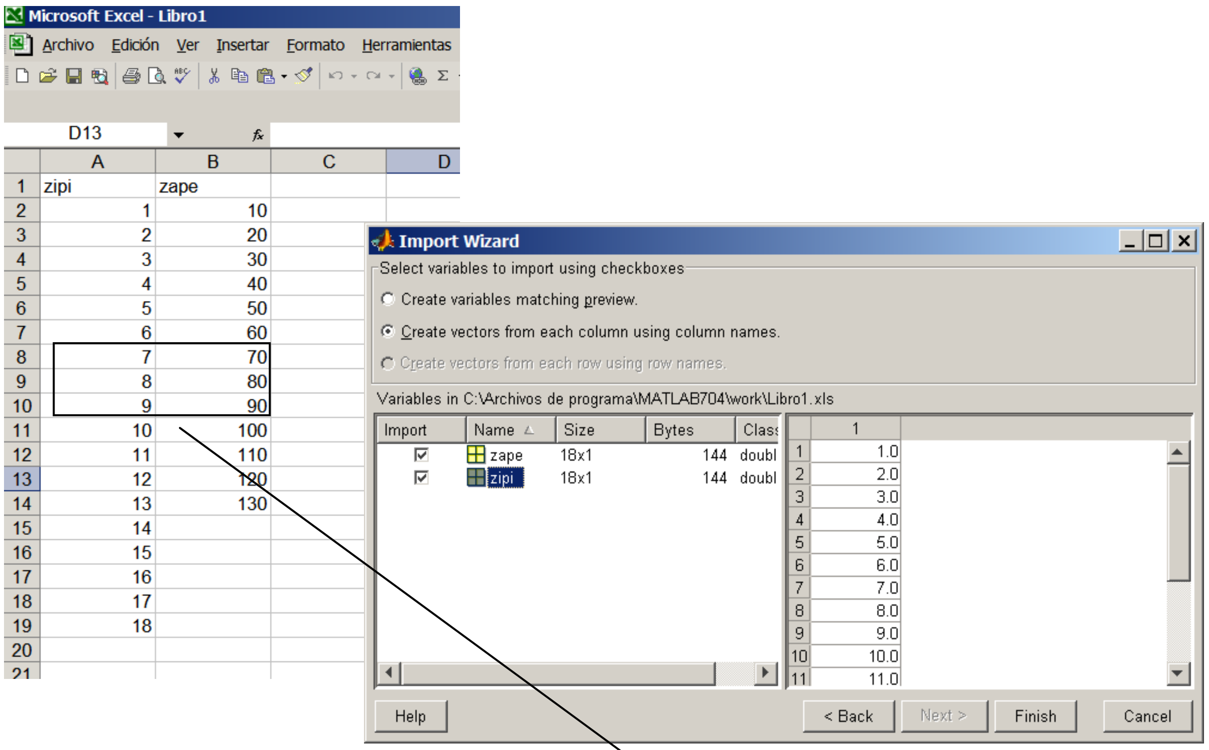
## 2.5 Relationship with Excel

MATLAB can exchange data with other Windows programs. The next exercise shows the relationship with Excel.

**Exercise 6. Variables and workspace (III). Relationship with Excel**

Create a simple Excel file like the following one (`Libro1.xls`) and save it in the MATLAB work directory `<work>`.

**1)** Load data from the workspace window: To import data from `Libro1.xls`, open the Import Wizard window by double-clicking on `Libro1.xls` in the Current Directory window or click on in the workspace window.



| | A | B | C | D |
|---|---|---|---|---|
| 1 | zipi | zape | | |
| 2 | 1 | 10 | | |
| 3 | 2 | 20 | | |
| 4 | 3 | 30 | | |
| 5 | 4 | 40 | | |
| 6 | 5 | 50 | | |
| 7 | 6 | 60 | | |
| 8 | 7 | 70 | | |
| 9 | 8 | 80 | | |
| 10 | 9 | 90 | | |
| 11 | 10 | 100 | | |
| 12 | 11 | 110 | | |
| 13 | 12 | 120 | | |
| 14 | 13 | 130 | | |
| 15 | 14 | | | |
| 16 | 15 | | | |
| 17 | 16 | | | |
| 18 | 17 | | | |
| 19 | 18 | | | |
| 20 | | | | |
| 21 | | | | |

Select variables to import using checkboxes
○ Create variables matching preview.
[✓ Create vectors from each column using column names.]
○ Create vectors from each row using row names.

Variables in C:\Archivos de programa\MATLAB704\work\Libro1.xls

| Import | Name | Size | Bytes | Class |
|---|---|---|---|---|
| ✓ | zape | 18x1 | 144 | doubl |
| ✓ | zipi | 18x1 | 144 | doubl |

| | 1 |
|---|---|
| 1 | 1.0 |
| 2 | 2.0 |
| 3 | 3.0 |
| 4 | 4.0 |
| 5 | 5.0 |
| 6 | 6.0 |
| 7 | 7.0 |
| 8 | 8.0 |
| 9 | 9.0 |
| 10 | 10.0 |
| 11 | 11.0 |

**2)** Load data using the command `xlsread`: To load a part of the data file named "chorizo" in the workspace, type

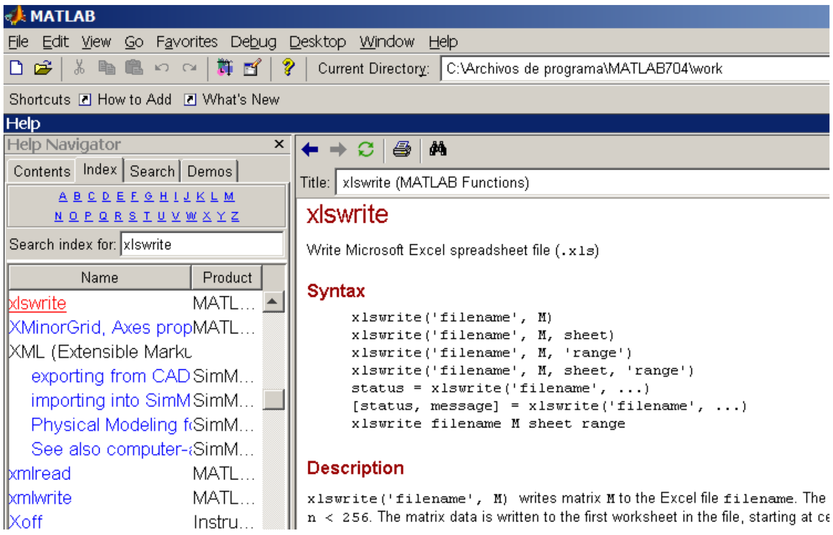
```
>> chorizo=xlsread('Libro1','A8:B10')
```

```
                    chorizo =
                         7    70
                         8    80
                         9    90
```

**3)** Save data using the command `xlswrite`: First of all, create a cell array variable (sorry about the names of the variable, I have not had breakfast today!).

```
>> starved={'chorizo','mortadela';1 2;3 4}
starved =
    'chorizo'    'mortadela'
    [      1]    [        2]
    [      3]    [        4]
```

Now use `xlswrite` to save `starved` in the file `Libro2.xls` on a sheet (not yet created) named `Embutidos` from the `B2` cell. Use the MATLAB help window to see the syntax for function `xlswrite`:


MATLAB
File Edit View Go Favorites Debug Desktop Window Help
Current Directory: C:\Archivos de programa\MATLAB704\work
Shortcuts How to Add What's New
Help
Help Navigator
Contents Index Search Demos
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
Search index for: xlswrite
Name Product
xlswrite MATL...
XMinorGrid, Axes prop MATL...
XML (Extensible Marku
exporting from CAD SimM...
importing into SimM SimM...
Physical Modeling f SimM...
See also computer-a SimM...
xmlread MATL...
xmlwrite MATL...
Xoff Instru...
Title: xlswrite (MATLAB Functions)

xlswrite

Write Microsoft Excel spreadsheet file (.xls)

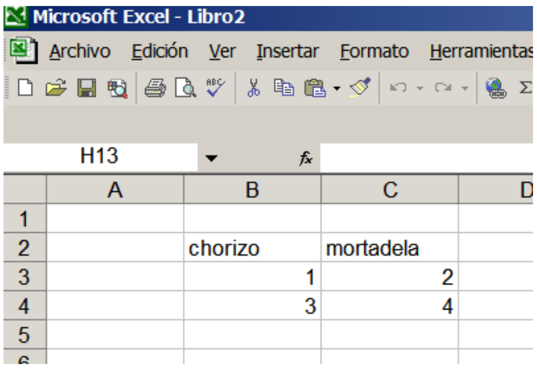
Syntax

```
xlswrite('filename', M)
xlswrite('filename', M, sheet)
xlswrite('filename', M, 'range')
xlswrite('filename', M, sheet, 'range')
status = xlswrite('filename', ...)
[status, message] = xlswrite('filename', ...)
xlswrite filename M sheet range
```

Description

`xlswrite('filename', M)` writes matrix `M` to the Excel file `filename`. The `n < 256`. The matrix data is written to the first worksheet in the file, starting at ce

```
>> xlswrite('Libro2',datos,'Embutidos','B2')
Warning: Added specified worksheet.
> In xlswrite>activate_sheet at 254
  In xlswrite at 212
```



| | A | B | C | D |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | chorizo | mortadela | |
| 3 | | 1 | 2 | |
| 4 | | 3 | 4 | |
| 5 | | | | |

(Note: This exercise is for practice purposes only. It is not necessary to submit it to Moodle.)

# 3. Programming language

## 3.1 Basic symbols and commands

*Matrix:* The basic computational unit in MATLAB is the rectangular matrix with real or complex elements. Square dimensional matrices, vectors and scalar variables are considered to be particular cases of this basic data structure.

**Matrix input:** Matrix elements are entered inside brackets `[ ]`. Elements from the same row can be separated with a space or a comma. A different row is specified by using semicolons or the Enter key ↵.

**Example:** Entering a matrix.

```
» A=[1 2;3 4]     (key ↵)
A =
      1   2
      3   4
```

**Semicolon:** It avoids the presentation of results in the command window.

```
» A=[1 2;3 4];    (key ↵)
»
```

*Submatrix reference:* Given a matrix **A**, if we want to pick out the sub-matrix **B** defined in rows 3 to 5 and columns 4 to 7 of matrix **A**, two possibilities are

```
»B=A(3:5,4:7)     or    »B=A([3,4,5],[4 5 6 7])
```

To obtain sub-matrix **C** defined in rows 1, 3 and 4 of **A**, the command is

```
»C=A([1 3 4],:)
```

The comma separates the specification of rows from the specification of columns. Notice that the colon symbol **:** after the comma means “all columns”.

*Special symbols:* There are several predefined variables. Imaginary number $\sqrt{-1}$ can be expressed either as **`i`** or **`j`**, **`Inf`** corresponds to $+\infty$ and **`pi`** refers to $\pi$. **`NaN`** (*Not-a-Number*) is obtained in non-definite operations, such as 0/0.

**Example:** Rectangular matrix with special symbols.

```
» A = [sin(pi/2) -4*j 9;sqrt(2) 0/0 log(0)]
Warning:    Divide by zero
Warning:    Log of zero
A =
      1.0000            0-4.0000i         9.0000
      1.4142         NaN                    -Inf
```

**Ans:** When no output variable is specified (the command is simply `»function_name(…)`), MATLAB assigns the result to the `ans` (for *answer*) variable.

```
» 12.4/6.9   (key ↵)
ans =
      1.7971
```

**Stop execution:** Key combination `<Control> <c>`.

*Command syntax:* MATLAB is interactive software in the sense that it establishes a “dialogue” with the user by means of a language based on commands or statements. In response to a statement, MATLAB executes it or gives the corresponding error message.

It is possible to enter several commands in the same row. These commands must be separated by commas or semicolons.

Commands can be
- Mathematical operations
- Script invocations (to execute a script, simply type the script name – no extension is needed)
- Function invocations (to execute a function, you must specify input and/or output arguments)

**Calling functions:** Input arguments are inside parentheses and output arguments (if there are two or more) are inside brackets.

```
» [output_arg]=nombre_función(input_arg)  (key ↵)
» nombre_función(input_arg)   (key ↵)
```

In a typical session, each command produces new variables that are stored in the workspace. These variables can be used later as input arguments for new commands introduced by the user. When the user exits MATLAB, all variables in the workspace are deleted.

**Example 3. Functions syntax**

Although the input and output variables can have any name, in order to interpret the result their order is important. For instance, function **`eig`** computes both the eigenvalues and eigenvectors of a given matrix.

Function **`eig`** with only one output argument (or with no output argument at all), computes the eigenvalues:

```
>> A=[-3 2;0 1];
>> eig(A)
ans =
    -3
     1
```

If you call **`eig`** with two output arguments, the function gives a first output variable containing the modal matrix (a matrix whose columns are the eigenvalues) and a second output variable containing the Jordan form of input matrix **A** (that is, a diagonal matrix with eigenvales in the main diagonal):

```
>> [eigenvect,diag]=eig(A)

eigenvect =
    1.0000    0.4472
         0    0.8944

diag =
    -3     0
     0     1

>>
```

In fact, most functions give different results depending on the number of input and/or output arguments. Therefore, it is advisable to check the help file before using a function for the first time, e.g. **`>>help eig`**.

---

*Special matrices generation:* It is possible to generate matrices (or vectors) whose elements are equal to "1" (function **`ones`**) or equal to "0" (function **`zeros`**). It is also possible to generate matrices of random elements: use **`randn`** to generate matrices of Gaussian-distributed elements or **`rand`** to generate uniform distributed numbers. Identity matrices can be generated with the **`eye`** function.

*Generation of vectors with equally spaced elements:* The two main functions are **`linspace`** and the colon **`:`**. Notice that the former can be used to specify the number of points, while the latter can be used to specify the step between samples.

```
>> x=linspace(0,1,5)   %generate 5 points between 0 and 1
x =
         0    0.2500    0.5000    0.7500    1.0000

>> x=linspace(0,1);   %100 points between 0 and 1  (default)
```

```
>> y=0:5
y =
     0     1     2     3     4     5

>> y=0:0.5:5
y =
  Columns 1 through 7
    0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000
  Columns 8 through 11
    3.5000    4.0000    4.5000    5.0000
```

*Polynomials and transfer functions:* Polynomials are entered as row vectors where the elements are the polynomial coefficients.

For instance, polynomial $s^4 + 5s^2 + 3s - 10$ is introduced as:

```
» polinomio=[1 0 5 3 -10];
```

Notice that the coefficient corresponding to $s^3$ is zero.

Transfer functions are introduced by separately typing the numerator and denominator polynomials. Thus the function

$$H(s) = \frac{s}{s^3 + 2s + 5}$$

will be defined by the instructions

```
»num = [1 0];
»den = [1 0 2 5];
```

It is also possible to combine the numerator and denominator into a single variable of class **tf** (transfer function).

```
»H = tf (num, den)
Transfer function:
     S
-------------
s ^ 3 + 2 s + 5
```

*Polynomial product:* To multiply polynomials, you can use the convolution product function, **conv**. It only allows two input arguments at a time but the functions can be nested. To get the product $(s+2)\cdot(s^2+4s)\cdot(3s^2+7s+2)$, type the following command:

```
>> conv([1 2],conv([1 4 0],[3 7 2]))
ans =
     3     25     68     68     16      0
```

That is, $3s^5 + 25s^4 + 68s^3 + 68s^2 + 16s$.

*Other data types:* If you want to store only one variable data of different types, you can use the *struct* or *cell array* variables.

For instance, to store the maximum temperature of several days along with the labels “day” and “max temp”, you can use a *cell* type variable. Type >>`help cell` for more information.

```
>> temp={'day','max temp';1,25.7;2,25.5;3,25.4}

temp =

    'day'     'max temp'
    [   1]    [ 25.7000]
    [   2]    [ 25.5000]
    [   3]    [ 25.4000]
```

To organize different information types using several fields, you can use *struct* type variables. Type `>>help struct` for more information. For instance, if you want to save the input and output records of an experiment along with the realization date, type:

```
>> x=0:10;
>> y=2*x;

>> experiment.input=x;
>> experiment.output=y;
>> experiment.date='1mar10';
>> experiment

experiment =
      input: [0 1 2 3 4 5 6 7 8 9 10]
     output: [0 2 4 6 8 10 12 14 16 18 20]
       date: '1mar10'
```

*Format:* The **`format`** command can be used to change the numerical format in which MATLAB presents the results. Several formats are

`short`: fixed comma with 4 decimals (default)
`long`: fixed comma with 15 decimals
`bank`: two decimals
`rational`: to show the rational numbers as the ratio of two integers.

Two useful instructions for submitting results are **`disp`** and **`sprintf`**:

```
>> disp 'hello'
hello
>> disp ('hello')
hello
>> disp(sprintf('\n\t\t\t Table 1'))

                   Table 1
```

Click `>>help command_name` for more information.

## 3.2 Mathematical functions and table of operators

Next tables show the basic operators and functions.

| Mathematical operators | |
|---|---|
| `+` | sum |
| `-` | subtract |
| `*` | product |
| `/` | right division |
| `\` | left division |
| `^` | power |
| `'` | conjugate transpose |

**Table 1.1**

| Basic functions | |
|---|---|
| `abs` | absolute value |
| `angle` | phase |
| `sqrt` | square root |
| `real` | real part |
| `imag` | imaginary part |
| `conj` | conjugate |
| `exp` | exponential basis *e* |
| `log` | natural logarithm |
| `log10` | basis 10 logarithm |

**Table 1.2**

| Trigonometric functions | |
|---|---|
| `sin` | sine |
| `cos` | cosine |
| `asin` | arcsin |
| `acos` | arcosine |
| `tan` | tangent |
| `atan` | arctangent |
| `sinh` | hyperbolic sine |
| `cosh` | hyperbolic cosine |
| `tanh` | hyperbolic tangent |

**Table 1.3**

*A dot before a mathematical operator:* A dot before an operator between two vectors or matrices indicates that the operation must be performed element-to-element. For instance, `A*B` is the matrix product of matrices **A** and **B**. If **A** and **B** have the same dimension, typing `C=A.*B` results in a **C** matrix whose $c_{ij}$ elements are computed as $c_{ij}=a_{ij}\times b_{ij}$.

## 3.3 Logical operators

The instructions in MATLAB may undergo a process of programming, such as running a particular set of functions only if it satisfies some Boolean condition, performing loops, etc.

*Boole algebra:* Relational operators are `==` (equality), `<`, `<=` (less than, less than or equal to) and `>`, `>=` (greater than, greater than or equal to). There are also commands for Boolean comparisons, e.g., `gt(i,1)` means i is greater than 1.

Boolean conditions are expressed inside parentheses, for instance, `(a==2)`. If this condition is true, the output value is “1”; if it is false, the output value is “0”.

Logical operators are `&` (*and*), `|` (*or*) and `~` (*not*); `xor` is not an operator but a function.

*Flux control:* Basic instructions for the flux control are `for...end`, `if... elseif... else...end`, `while...end`, `switch...case...otherwise...end`.

**Example:** We determine the sum of a vector’s components by using a loop.

```
»x=[1 2 3 4 5];
»add=0;
»for i=1:length(x)
          add=add+x(i);
end
»add
add =
      15
```

(Note: this loop was for illustration purposes only. To determine the sum of all the vector elements we can simply do: `sum([1 2 3 4 5])`)

*Parentheses and brackets:* Note that in the above example, we used parentheses to access the "i" element of the vector x. Parentheses ( ) are only used to index, to set a priority on mathematical operations and to contain the input parameters of functions. Brackets [ ] are used to define vectors and matrices and to contain the output variables in functions that give more than one.

*Helpful instructions:* Two instructions that are very useful are `size` (returns the size of a variable) and `length` (if the variable is a vector, it returns its length), since they are used to verify that the commands are running correctly. Also useful in programming is the `find` function, which returns the index of the elements within a vector or matrix that satisfy a given Boolean condition.

## 3.4 Functions created by the user

For the user, editing functions makes sense in situations when the same program structure will be used several times for different parameters that can be introduced externally.

To create the file containing the commands to be executed by our function or script, open a text editor and write the sequence of commands to be executed. In the case of a function, the file name has to coincide with the function name (e.g., `func1.m`). An *M-file* is structured in three parts:

*Calling:* Use the following syntax to call a function.

```
»[output_arguments] = function_name (input_arguments)
```

*Structure:* An M-file consists of three parts.

- Header (only in functions): `function` [output_args]=function_name(input_args)

```
function [sine,cosine,tangent]=func1(ang)
```

- Help comments (optional): Help comments appear in the command window when you type `>>help function_name`

```
% FUNC1 Test function
% [sine,cosine,tangent]=func1(ang) computes the sine,
% cosine and tangent of the angle 'ang'
```

- Commands collection:

```
sine=sin(ang);
cosine=cos(ang);
tangent=tan(ang);
```

**Exercise 7. Creation of functions by the user**
**1)** Open the M-file editor (click on 🗋) and write the instructions of the previous example.
**2)** Save the file using the same name as the function (`func1.m` in our case).
**3)** Type `>> help func1` in the command window.
**4)** Call the function `func1` for various values of `ang` and see the results.

(Note: This exercise is for practice purposes only. It is not necessary to submit it to Moodle.)

## 4. MATLAB graphics

To generate a simple *x-y* plot in MATLAB, follow these steps:

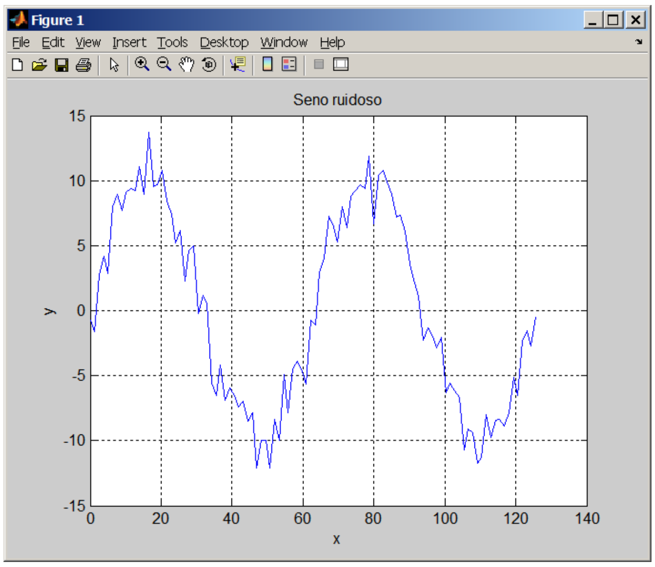
1. Generate (or load) a vector **`x`** containing the abscissas axis values.
2. Generate (or load) a vector **`y`** containing the y-axis values. The lengths of vectors **`x`** and **`y`** must be the same. You can also use names other than **`x`** and **`y`**.
3. Execute any plot command, e.g., **`>>plot(x,y)`**
4. Optionally, you can grid the plot **`(>>grid`**), label the axes (**`xlabel`**, **`ylabel`**) and insert a title (**`title`**).

Note: To draw two plots in the same figure, type **`>>plot(x1,y1,x2,y2).`**

**Example 4. Simple plot**

Next, we generate and plot two cycles of a sinusoid of frequency 0.1rad/s and amplitude 10 that is corrupted by additive white Gaussian noise with zero mean and intensity (variance) 3:

```
>> x=linspace(0,2*2*pi/0.1);
>> y=10*sin(0.1*x)+randn(size(x))*sqrt(3);
>> plot(x,y)
>> grid,xlabel('x'),ylabel('y'),title('Seno ruidoso')
```



To copy the figure in a document (Word, for example), simply select *Edit* → *Copy Figure* in the figure window menu bar.

## 5. Using toolboxes

To check which toolboxes are available in your MATLAB, simply type `>>ver`. Here we illustrate the use of some functions belonging to the *Signal Processing Toolbox*.

**Example 5. Signal spectrum. Power spectral density**

Let us obtain and plot the power spectral densities (PSDs) for the following signals:

**1)** Multisinusoid: $x(t) = 0.5\,\mathrm{sen}(2\pi 3t) - 0.2\,\mathrm{sen}(2\pi 9t)$.

**2)** Square signal with frequency 0.3Hz ($T$ = 3.33s) and pulse width 0.3s (duty cycle of 10%).

**Solution:**
First, we generate a time vector of length $N = 2^n$, $n$ integer. For instance,

```
t=0:0.01:10.23;
```

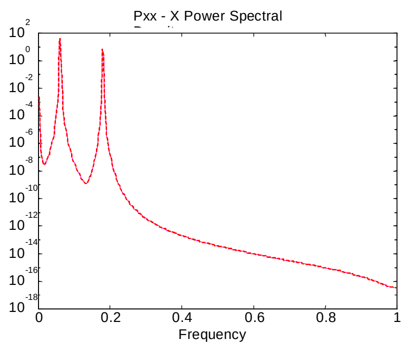
contains 1024 samples. Note that the sampling period is $T_s$ =0.01$s$ and hence the sampling frequency is $f_s = \dfrac{1}{T_s} = 100Hz$.

Second, we generate and plot the multisinusoid signal and the square signal:

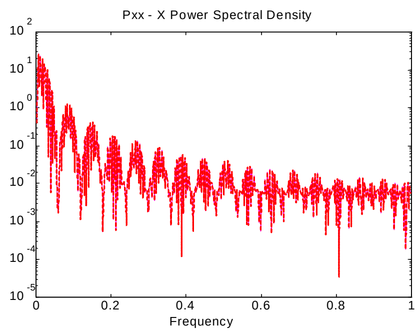
```
x1=0.5*sin(2*pi*3*t)-0.2*sin(2*pi*9*t);     plot(t,x1)
x2=0.5*square(2*pi*0.3*t,10)+0.5;           plot(t,x2)
```

To obtain the PSD, we use the function **`spectrum`**. To plot the results, we use **`specplot`**.

```
px1=spectrum(x1,1024);specplot(px1)
px2=spectrum(x2,1024);specplot(px2)
```



PSD for the multisinusoid.



PSD for the square signal.

Note that **`specplot`** plots the spectrum up to Nyquist frequency $f_s/2$. Moreover, it is normalized to 1. In the case of the multisinusoid signal, if we use the function **`ginput`**, we obtain the normalized frequencies $f_{1n}$ = 0.06 and $f_{2n}$ = 0.18. If we multiply them by $f_s/2$ = 50, the result is $f_1$ = 3 and $f_2$ = 9, as we might expect.

---

**Example 6. White noise autocorrelation**

Let us compute and plot the autocorrelation for a zero mean Gaussian distributed white noise with intensity 4.

**Solution:**
First, we generate a time vector of length $N = 2^n$, $n$ integer. For instance,

```
Ts=0.01;fs=1/Ts;fn=fs/2;
t=0:Ts:10.23;
```

contains 1024 samples. The sampling period is $T_s$ =0.01*s*. Hence, the sampling frequency is $f_s = \frac{1}{T_s} = 100Hz$ and the Nyquist frequency is $f_N = \frac{f_s}{2} = 50Hz$.

Second, we generate (and plot) the noise sequence,

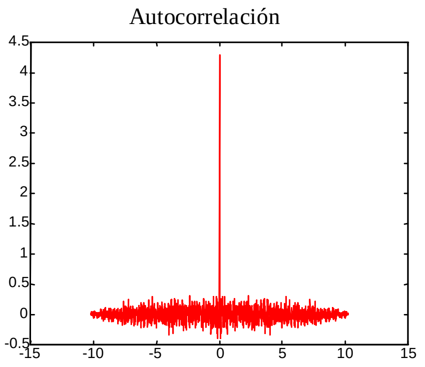
```
n=randn(size(t))*sqrt(4);plot(t,n)
```

and we check the mean and intensity values

```
mean(n),cov(n)
```

The autocorrelation sequence is obtained using the function **`xcorr`**. Its length is twice minus one the length of the original noise sequence. Use **`ginput`** or **`max`** to check that the central value coincides with the signal power.

```
rn=xcorr(n,'biased');
plot(-10.23:0.01:10.23,rn)
```

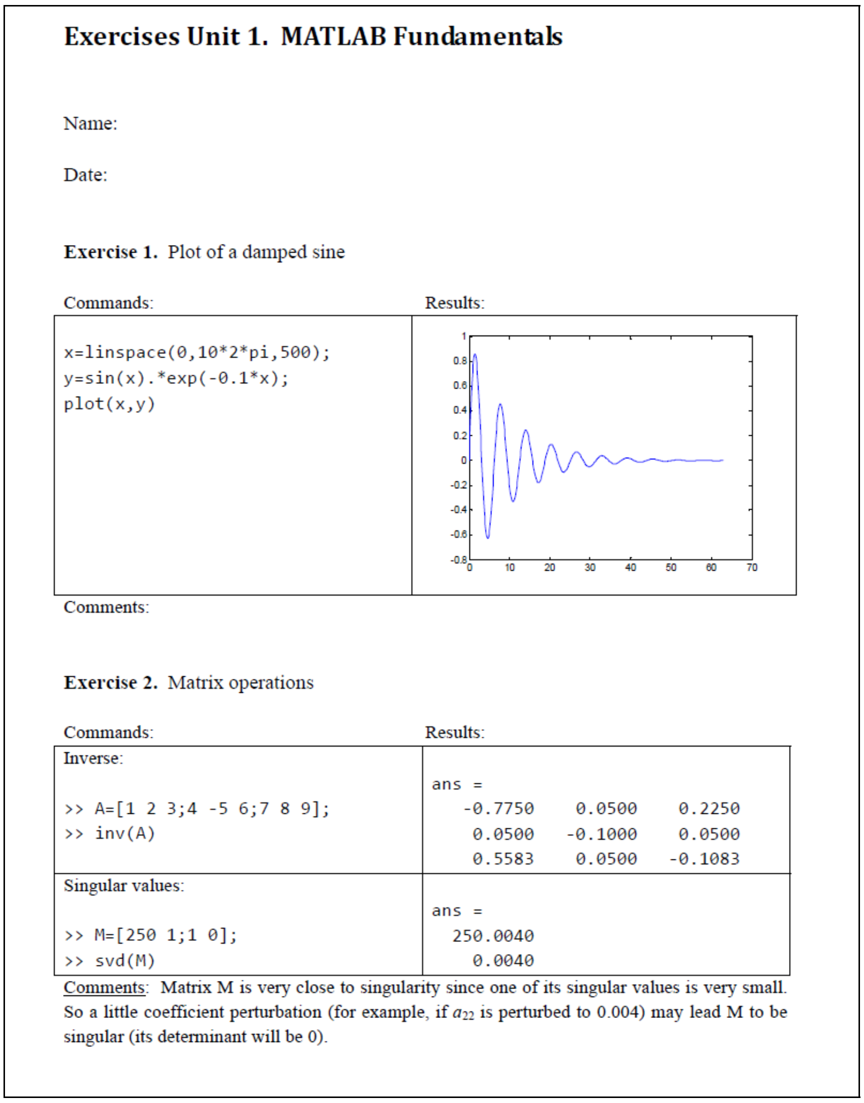


---

## 6. Final comments and instructions for proposed exercises

One of the main features of MATLAB is that there are different ways of doing the same things. In these notes, we have presented the most important and useful (text) commands. The action performed by many of these commands can also be executed from the menu bars and control buttons of recent versions but, since the windows layout is self-explanatory, it has not been considered necessary to describe what each submenu or button does. However, before beginning to work, it is advisable to take a look at all the options and controls in order to get an overview of all the possibilities of the program.

*How to present the proposed exercises:* In all units, you have to upload only one file to the virtual campus, for example, **`your_name_E1.pdf`**, containing the solution to the proposed exercises. The solution must include the MATLAB commands used and the commented results. Here is a suggestion for the results file:

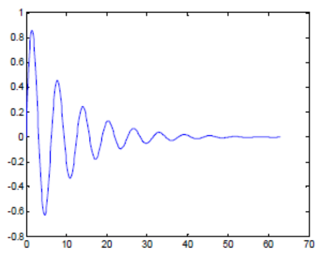


**Exercises Unit 1. MATLAB Fundamentals**

Name:

Date:

**Exercise 1.** Plot of a damped sine

| Commands: | Results: |
|---|---|
| `x=linspace(0,10*2*pi,500);`<br>`y=sin(x).*exp(-0.1*x);`<br>`plot(x,y)` |  |

Comments:

**Exercise 2.** Matrix operations

| Commands: | Results: |
|---|---|
| Inverse:<br><br>`>> A=[1 2 3;4 -5 6;7 8 9];`<br>`>> inv(A)` | `ans =`<br>`   -0.7750    0.0500    0.2250`<br>`    0.0500   -0.1000    0.0500`<br>`    0.5583    0.0500   -0.1083` |
| Singular values:<br><br>`>> M=[250 1;1 0];`<br>`>> svd(M)` | `ans =`<br>`  250.0040`<br>`    0.0040` |

Comments: Matrix M is very close to singularity since one of its singular values is very small. So a little coefficient perturbation (for example, if $a_{22}$ is perturbed to 0.004) may lead M to be singular (its determinant will be 0).