I. Introduction to MATLAB

MATLAB is the essential tool for digital signal processing that you'll be using during your studies at ENSEA. It's a simple programming language, with an abundant collection of useful functions.

I.1. Start-up and basic operations

MATLAB is a widely-used software package, and documentation is plentiful: many books on using MATLAB are available from the library, many webinars for beginners are available on the MathWorks website www.mathworks.fr/webinars and many engineering schools and universities offer an online course of basic functions. The following is a brief summary of what you'll need for the next 2 sessions.

I.1.1. Launching a Linux work session

MATLAB is launched from a terminal by typing *matlab&* and opens a window divided into several sub-windows. The "**Current Directory**" sub-window shows your **working directory, i.e.** the location on disk where your personal files will be stored. It goes without saying that you must be working in a directory in your session and not in the default directory.

Always work *in a personal directory, to avoid interaction with other students' files.* ***Always*** *save your files on a USB stick or in your session at the end of the session. Otherwise, you may not find them intact.*

The "**Command Window**" sub-window displays a command prompt: **>>**

A MATLAB work session consists of a series of commands entered from the keyboard in this sub-window, which are executed immediately. These commands manipulate variables, using operators and functions.

The result of each instruction is displayed in the command window, unless terminated by a semicolon ';'.

I.1.2. Variables

MATLAB knows only one type of variable: **matrices of complex numbers**. A scalar (real or complex) is treated as a 1x1 matrix. A vector is a 1xN or Nx1 matrix (MATLAB distinguishes between row and column vectors and refuses operations between matrices of incompatible dimensions).

Variables are designated by names. As with C, and unlike VHDL, MATLAB is a case-sensitive language, so variable names are case-sensitive.

MATLAB is a semi-interpreted language, which means that, unlike C, variables do not need to be declared in advance. MATLAB creates them (or adapts their dimensions) as commands are executed.

- The symbols '[ ]' and ';' are used to define variables explicitly. For example :

  `X = [ 1 2 ; 3 4 ]` (or equivalent: `X = [ 1 , 2 ; 3 , 4]`)

  The comma (or space) separates elements on the same line, while the semicolon indicates a jump to the next line.

  ATTENTION: as the space is an authorized separator, its use can create ambiguities.

  `X1 = [1 + 2 3]` is different from `X2 = [1 +2 3]`.

- The syntax **α:β:γ** is used to define a line vector whose elements go from **α** to **γ** with a step **β**.
  `Y = 3 : 0.3 : 5, Z = 5 : -0.3 : 3`

  When **β** = 1, it can be omitted: `N = 0 : 9`

- Many functions can be used to define particular matrices. For example :

A = zeros (2,3)	2x3 matrix with zero elements
B = ones (3,2)	3x2 matrix whose elements are 1
C = eye (2,3)	matrix 2x3 whose elements are zero except on the main diagonal where they are 1
D = randn (3,2)	matrix 3x3 whose elements are random variables (normal distribution)

Predefined variables

Several variables are predefined when MATLAB is started.

pi	3.14159265358979
eps	relative precision equal to 2^{-52}
i	imaginary unit ($\sqrt{-1}$)
j	imaginary unit ($\sqrt{-1}$)
ans	variable containing the result of the last command

WARNING: these variables can be redefined without constraint: **pi = 2** is a legal command.
To return to the initial value, use **clear pi**.

The workspace

The workspace is made up of all the variables allocated in RAM. The **who** and **whos** commands are used to identify existing variables and their dimensions. The **clear A command** destroys variable **A** and frees the associated memory space. The **close (2) command** destroys the second graphics window. The **all** parameter can be used on the 2 previous commands to reset the workspace.

Variable types and names

The readability of a code is strongly correlated to the names of the variables used, which must be as explicit as possible. In order to clarify the situation for everyone, you should endeavor to respect the following convention:

- indices, scalars: a lowercase letter,
- global constant (equivalent to a macro in C): name in uppercase,
- time line vector: name beginning with a capital letter,
- matrices (operator or concatenation of vectors): name with at least 2 capital letters.

MATLAB is a non-compiled, case-sensitive language, so check the syntax of your variables to avoid errors at runtime:

```
temp = 37.2 ; Threshold_temp = Temp +[-1,1] ;
```

Finally, as in C, structured programming is facilitated by the **struct** command.

I.1.3. Handling matrices and elements

Matrices can be manipulated globally:

```
M1 = ones (4,6); M2 = 2 * M1
```

or by elements, using parentheses

```
M1(3,4) = M2 (1,2)
```

Symbol**:** used to designate an entire row (or column).

```
MRand = randn (3,3) , Vy = MRand ( : , 2) , Vx = MRand (3, : )
```

When a matrix is a vector (row or column), it is sufficient to use a single index: **Vy(2)**

The **:** symbol can also be used to transform a matrix into a column vector: **Vx(:)**

Unlike C, **array indices start at 1**. MATLAB returns an error if an index is negative or zero. Error messages during execution are self-explanatory, so don't hesitate to click on them to quickly identify the error.

Wherever we use a scalar, we can use a matrix, provided the dimensions are correct. This feature makes it possible to write down matrix element manipulations in concise form.

```
Abs = 10 : 20 , Ord = Abs(11:-1:1) , Ind = 1:2:8 , Ord (Ind) = 0
MRand = randn (4,4) , Ma = MRand ( : , [3,1,2])
C1 = [ Abs , Ord] , C2 = [ Abs ; Ord]
```

The `size` command returns a 1x2 matrix indicating the size of a matrix. The `length(Abs)` command returns the result of `max(size(x))`, i.e. the number of components if **x** is a row or column vector:

```
s = size (Ma)
l = length (Vx)
```

The `end` symbol can be used to designate the last element of an array.

```
X = 10 : 20 ; [X (length(X) - 1) , X (end - 1)]
```

I.1.4. Arithmetic operators

MATLAB provides the following basic operators:

`+, - , *, /, \` addition, subtraction, multiplication, division

`^`	rise to power
`'`	complex transposition (conjugate transpose)
`. '`	transposition

USED ON THEIR OWN, THESE OPERATORS OBEY THE LAWS OF MATRIX OPERATIONS. PRECEDED BY A DOT, THEY OPERATE ON MATRIX ELEMENTS.

In the case of transposition, '.'' performs a transposition without conjugation.

Exception: to simplify writing, MATLAB converts scalars into matrices of the required dimension.

`X = diag([3,5,4]) , Y = X + 3` (strictly speaking, you should have written `Y = X + 3 * ones(3,3)`)

I.1.5. Specific commands for signal tracing

The `stem` command plots a vector as a sampled signal:

```
N = 0 : 49 ; nu0 = 0.05 ; Y = sin (2*pi*nu0*N); figure (3); stem(Y);
```

The plot is unreadable when the number of samples is large, so we prefer the `plot`.

The `plot` command is used to plot 1D :

- `plot (Y)` plots vector **Y** against implicit index vector `1:length(y)`
- `plot (N, Y)` plots the vector **Y** as a function of the vector **N** (**they must be the same size**!)
  If **Y** is a complex vector, `plot (y)` is equivalent to `plot (real(y), imag (y))`
- `plot (N, Y, 'r', N, Y, 'g--', N, Y, 'bd')` plots **Y** versus vector **N** in 3 different ways (for other patterns and colors, see Help).

Commands (`grid`, `title`, `xlabel`, `ylabel`, `legend`) add grids, titles and labels to axes and curves respectively, once they have been drawn.

There are several ways to plot multiple curves in the same graphical window, including the following 2:

- superimposed, with a single call to the `plot` function. Each curve must be defined by an abscissa vector and an ordinate vector of the **same dimensions** (amazing, isn't it?).
  ```
  N = 0 : 99;
  hold on; plot (N, sin(2*pi*0.02*N), 'r', N, sin(2*pi*0.04*N), 'g'); step(N, sin(2*pi*0.06*N), 'm'); hold off;
  ```
- side by side, by subdividing the graphics window using the `subplot` command (see Help)
  ```
  subplot (211); plot (N, sin(2*pi*0.02*N), 'r');
  subplot (212); plot (N, sin (2*pi*0.04*N), 'g');
  ```

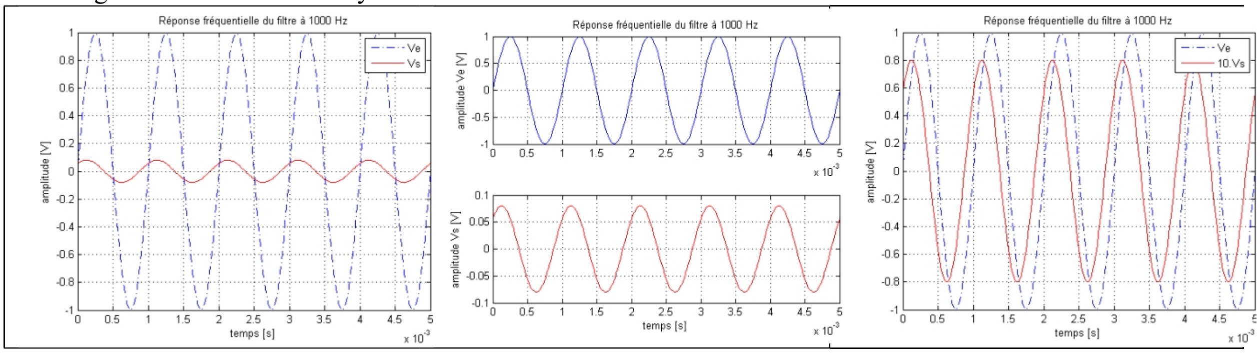
Multiple graphics windows can be opened for separate plotting.

```
close all;

figure(4); plot (N, sin(2*pi*0.02*N), 'r'); figure; plot(N, sin(2*pi*0.04*N), 'g');
```

I.1.6. Plot layout

A plot should be as explicit as possible to highlight the features indicated in the figure's title. It is therefore advisable to distort and zoom in on the axes, even if this means making several plots: one at full scale and others detailing a particular feature. It goes without saying that if more than one plot is required, they should all appear on the same figure to make it easier for the reader to understand. Often, simple signal shaping is all that's needed, as demonstrated by the following tests on a completely dummy example where only the third plot deserves to appear in a report. At a frequency of 1000 Hz, the complex gain of the filter is clearly $0.8e^{j.\pi/4}$ ...



I.1.7. Scripts and functions

In practice, MATLAB is used to simulate processes that require you to write programs containing numerous instructions. These are never typed directly into the command window, but into text files with the suffix '.m'.

MATLAB considers these files as executable programs and distinguishes between two types: **scripts** and **functions**. A large number of functions are supplied with MATLAB and are installed in a TOOLBOX subdirectory of the MATLAB installation directory. All functions in this subdirectory can be accessed from any session. However, if you create files in your session's default directory that are the same name as those in the TOOLBOX directory, they will obviously be inaccessible.

Scripts

A script contains a sequence of MATLAB instructions, as they might be entered from the keyboard. To execute it, simply enter the file name as a command in the main window (without the '.m' extension). The effect is exactly the same as if the commands had been entered successively in the main command window. In particular, variables created or modified in the script appear in the workspace, and are therefore observable once the script has been completed.

The advantage of a script is, of course, that you can replay all commands without retyping them on the keyboard, and keep a record of the commands you execute, even after MATLAB has been shut down.

For each practical part of the ESD course, you'll be asked to write a script that can include functions.

The functions

Like a script, a function is a file with the extension '.m' that contains a sequence of command instructions. But it must begin with the function command, which defines the equivalent of the prototype in C and thus specifies the incoming and outgoing parameters (you read that right, it's a plural).

Unlike scripting, variables created or modified in a function are local (with the exception of variables declared as output variables) and do not appear in the workspace once the function has finished.

It is therefore not advisable to start the development of an algorithm as a function, as during debugging no local variables are observable after an error stop, forcing you to use the MATLAB debugger in step-by-step mode or with breakpoints. It's easier to start development as a script, then transform it into a function if necessary once debugging is complete.

Two functions are provided for TP1, whose code you can consult in Moodle.

Development of scripts and functions

MATLAB code can be broken down into a script (main function) which can call other scripts (e.g. initialization of all variables) and/or function(s). To enhance readability and facilitate development, a file (script or function) can be broken down into blocks. A block is delimited by a comment beginning with "**%% blablabla**" (space is essential). The different design phases of a Matlab code will therefore be :

1. Succession of instructions typed into the main script in the text editor (which must be *"Desktop/Dock Editor* in MATLAB's main window),
2. Execution of each instruction in step-by-step mode,
3. Instructions are encapsulated in a block as soon as they are validated,
4. Analysis of the script produced to identify groups of instructions that are repeated, in order to extract them from the script and create functions. These are generally display instructions, which make the code more cumbersome.

A development criterion that significantly improves the readability of code is not to go beyond a page in the editor. The underlying idea is not to force the user to use the vertical scroll bar. Of course, this criterion does not take into account the size of the file header, where the following details must be provided

- type, size, unit of incoming and outgoing parameters,
- the algorithm used,
- an example of a function launch with expected results.

Development is facilitated by the debugger (Debug from the main menu xor corresponding icons xor F1x keyboard shortcuts), which allows you to execute your scripts and/or functions in step-by-step mode, block-by-block, function-by-function, right up to the breakpoint, and so on. An alternative to Debug mode is to insert the **keyboard command in** execution mode: this gives the user the opportunity to check the contents of variables at a critical moment, for example. To restart execution or not, **return** or **dbquit**.

I.1.8. Online help

MATLAB features an extensive online help system. It includes examples and a comprehensive - and therefore lengthy - tutorial.

MATLAB online help is available :

- from the command window, with the **help** xor **doc** xxx commands: **help** displays help on command xxx directly in the command window, while **doc** opens the general html help window in color.
- with the "MATLAB Help" menu opens the general html help window, which is more convenient for navigating through all the help and searching for unfamiliar commands.

The same command may appear in different TOOLBOXes (e.g. **doc step**). Only the command prototype can be used to identify the TOOLBOX used.

***ATTENTION** Mnemonics are indicated in uppercase (e.g. PI) in help, whereas they must be used in lowercase (pi) in programs.*