Tecnológico de Monterrey **Campus Ciudad de México** **School of Engineering and Sciences** **Department of Mechatronics**	**TE2019 Digital Signal Processing Laboratory** Dr Luis A. Montesinos Silva **Lab 1 – Introduction to MATLAB**

INTRODUCTION

MATLAB, which stands for MATrix LABoratory, is a computing language based on vectorial computations. MATLAB provides a convenient environment for numeric and symbolic computations in signal processing, control, communications and many other related areas [1]. The purpose of this lab activity is to familiarize yourself with the essential MATLAB functions used to perform calculations, signal generation and loading and writing audio data.

OBJECTIVES

1. To identify the principal elements and functionalities of the MATLAB software package
2. To use essential MATLAB functions to perform calculations, signal generation, loading, and writing data.
3. To appreciate MATLAB as a development, simulation and testing environment for digital signal processing.

PROCEDURE

0. Getting started with MATLAB [1]

a) There are two types of programs in MATLAB: one is the *script* which consists of a list of commands using MATLAB functions or your functions, and the other is the *function* which is a program that can be called from scripts; it has specific inputs and provides corresponding outputs.

b) Create a directory where you will put your work, and from where you will start MATLAB. This is important because when executing a script, MATLAB will search for it in the current directory and if it is not present in there, MATLAB gives an error indicating that it cannot find the desired script.

c) Once you start MATLAB, you will see three windows: the *command window*, where you will type commands, the *current folder* which shows the contents of the working directory, and the *workspace*, where the variables used are kept (Figure 1).

d) Your first command on the command window should be to change to the directory where you keep your work. You can do this by using the command **cd** (change directory) followed by the name of the desired directory. It is also essential to use the commands **clear all** and **close all** to clear all previous variables in memory and close all figures. If at any time you want to find in which directory you are, use the command **pwd** which will print the working directory which you are in.

e) Help is available in several forms in MATLAB. Just type **helpwin**, **helpdesk** or **demo** to get started. If you know the name of the function, help followed by the name of the function will give you the necessary information on that particular function, and it will also indicate a list of related functions. Typing 'help' on the command window will give all the HELP topics. Use help to find more about the functions listed in Table 0.1 for numeric MATLAB and in Table 0.2 for symbolic MATLAB.

f) To type your scripts or functions you can use the editor provided by MATLAB, simply type **edit** and the name of your file (Figure 1). You can also use any text editor to create scripts or functions, which need to be saved with the .m extension.

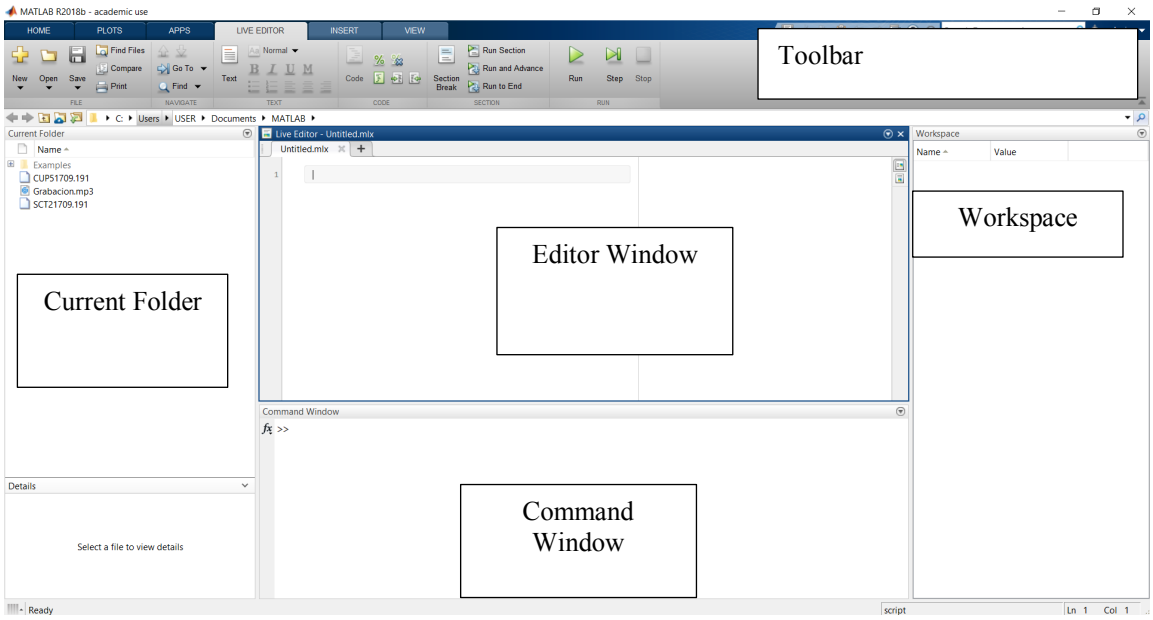

Figure 1. MATLAB's graphical user interface

1. Worked examples [1]

The following examples of scripts and functions are intended to illustrate different features of numerical computations—numerical data in and numerical data out—using MATLAB. Please consult Table 1 for a basic set of variables and functions, and use help to provide details.

1.1. MATLAB as a Calculator

In many respects MATLAB can be considered a sophisticated calculator. To illustrate this, consider the following script where computations with complex numbers, conversion of complex numbers from one form into another, and plotting of complex numbers are done. It is started with the commands **clear all** and **close all** in order to clear all the existing variables, and all the figures. Copy the script to the Editor and run it.

```
%%
%% Example 1.1 - Computation with complex numbers
%%
clear all;
close all;
z = 8 + j*3
v = 9 - j*2
a = real(z) + imag(v)       % real and imaginary
b = abs(z+conj(v))          % absolute value, conjugate
c = abs(z*v)
d = angle(z)+angle(v)       % angle
d1 = d*180/pi % conversion to degrees
e = abs(v/z)
f = log(j*imag(z+v))        % natural log
```

Notice that commands are not followed by a semicolon (;) and as such when we execute this script MATLAB displays the corresponding answer—if we want to avoid this the semicolon suppresses the answer from MATLAB. Also, some statements are followed by a comment initiated after the % symbol—text in these comments are ignored

by MATLAB. This simple script illustrates a large number of functions available to perform different calculations, some of them having names similar to the actual operations: real and imag find the real and imaginary parts while abs and angle compute the magnitude and the phase of a complex number. As expected, the phase is given in radians which can be transformed into degrees. The values of $j = \sqrt{-1}$ (i in MATLAB) and π do not need to be predefined.

1.2. MATLAB as a Signal generator

Numerically, the generation of a signal in MATLAB consists in creating either a vector, if the signal is one-dimensional, or a matrix if the signal is two-dimensional, or a sequence of matrices if three dimensions are needed. These vectors and matrices would correspond to sampled signals depending on time, to signals that depend on two space variables—like images—or to a video signal which is represented by a sequence of matrices that change with time.

Example 1.2a. MATLAB provides data files for experimentation that you can load and work with using the function load. For instance, the file train.mat is the recording of a train whistle, sampled at the rate of Fs samples/s (which accompanies the sampled signal y(n)). To be able to work with this file you can just type the following commands to **load** the data file (with extension .mat) into the work space:

```
%%
%% Example 1.2a - Loading of test signal train
%%
clear all
close all
load train
whos
```

Information as regards the file is given by the function **whos**. The sampling frequency is Fs = 8192 samples/s, and the sampled train sequence is a column vector with 12,880 samples.

Example 1.2b. One could then listen to this signal using the function **sound** with two input parameters, the sampled signal y and Fs , and then **plot** it:

```
%%
%% Example 1.2b - Listening to/plotting test signal train
%%
sound(y,Fs)
t = 0:1/Fs:(length(y)-1)/Fs;
figure
plot(t, y')
grid on
xlabel('n')
ylabel('y[n]')
```

Example 1.2c. To plot the signal, MATLAB provides the functions **plot** and **stem**. The function plot gives an interpolated version of the sequence, making the plot look continuous. The function stem plots the sequence of samples. To plot 200 samples of the train signal with stem we use the following script. Notice the way MATLAB gets samples 100 to 299 from the signal.

```
%%
%% Example 1.2c - Using stem to plot 200 samples of train
%%
figure
```

```
n = 100:299;
stem(n, y(100:299))
xlabel('n')
ylabel('y[n]')
title('Segment of train signal')
axis([100 299 -0.5 0.5])
```

Example 1.2d. It is possible to generate your signals by using many of the functions available in MATLAB. The following script illustrates how to generate a wave audio file and then to read it back and listening to it.

```
%%
%% Example 1.2d - Creating a WAV file from scratch and reading it back
%%
clear all
close all
Fs = 5000;                                          % sampling rate
t = 0:1/Fs:5;                                       % time vector
y = 0.1*cos(2*pi*2000*t) - 0.8*cos(2*pi*2000*t.^2); % sinusoid and chirp
%% writing chirp.wav file
audiowrite('chirp.wav', y, Fs)
%% reading chirp.wav back into MATLAB as y1 and listening to it
[y1, Fs] = audioread('chirp.wav');
sound(y1, Fs)                                       % sound generated by y1
figure
plot(t(1:1000), y1(1:1000))
```

The signal we create with this script is a sinusoid added to a chirp (a sinusoid with varying frequency). The signal has a duration of 5 s, and it is sampled at a rate of 5000 samples per second, so the signal has a total of 25,001 samples. The function **wavwrite** converts the signal generated by MATLAB into a wav audio file called 'chirp.wav'. This file is then read back by **wavread** and the function sound plays the sound file. The figure generated shows a segment of the generated sound file.

2. Problem-solving

In this activity, the fundamental frequencies of a 'virtual' recorder (i.e. *flauta dulce* in Spanish) will be generated following the next steps:

Step 2.1 To generate each fundamental frequency (Table 2), a vector containing a sinusoidal waveform of amplitude A must be defined. The general expression for a sinusoidal function is:

$$y(t) = A \sin(2\pi f t + \varphi) + \beta$$

where A is the amplitude, f is the frequency of the sinusoidal, t is the time vector, φ is the phase, and β is the offset. The frequency for each musical note, f, can be calculated as:

$$f = \nu / \lambda \ \ [\text{Hz}]$$

where ν is the speed of sound in air (~343 m/s), and λ is the wavelength produced for each musical note (Table 2).

As a guidance, a pseudocode for your scripts and suggested constant values are provided below:

```
Amplitude = define_value;                % A value of 20 is recommended
Duration = define_value;          % Duration of the tone [s]. A value of no longer than 3 is suggested
Sampling_frequency = define_value;     % The standard for audio is 44,100 samples/second
Sampling_period = 1/Sampling_frequency;
Time_vector = 0:Sampling_period:Duration;
f_note1= define_value             % Fundamental frequency of the musical note [Hz]
y_note1 = Amplitude * sin(2 * pi * f_note1 * t);
```

Step 2.2 Once defined, each musical tone can be played using MATLAB function **sound**:

```
sound(y)
sound(y, Fs)
sound(y, Fs, nBits)
```

Reproduce the sequence of the 12 musical tones; you may insert a pause between tones by using MATLAB function **pause** to make tones distinguishable from each other. Then plot the 12 musical notes.

Step 2.3 Generate the following sequence of notes:

mi mi mi do# mi sol#

do sol mi la si la sol, mi sol la fa sol mi do re si
do sol mi la si si la# la sol, mi sol la fa sol mi do re si
sol fa# fa re# mi sol la do la do re
sol fa# fa re# mi do do do

3. Reporting

Create and submit a report that contains your program lists (source code), table with calculated fundamental frequencies and plots. Use the template provided for this course, which can be downloaded from Blackboard.

REFERENCES

[1] L. Chaparro, *Signals and systems using MATLAB*. Waltham, MA: Elsevier, 2019.

Table 1. Basic numeric MATLAB functions. Reproduced from [1]

Special variables	ans	default name for result
	pi	π value
	inf, NaN	infinity, not-a-number error, e.g., 0/0
	i, j	$i = j = \sqrt{-1}$
	FUNCTION(S)	OPERATION
Mathematical	abs, angle	magnitude, angle of complex number
	acos, asine, atan	inverse cosine, sine, tangent
	acosh, asinh, atanh	inverse cosh, sinh, tanh
	cos, sin, tan	cosine, sine, tangent
	cosh, sinh, tanh	hyperbolic cosine, sine, tangent
	conj, imag, real	complex conjugate, imaginary, real parts
	exp, log, log10	exponential, natural and base 10 logarithms
Special operations	ceil, floor	round up, round down to integer
	fix, round	round toward zero, to nearest integer
	.*, ./	entry-by-entry multiplication, division
	.^	entry-by-entry power
	x', A'	transpose of vector x, matrix A
Array operations	*x=first:increment:last*	row vector x from *first* to *last* by *increment*
	x=linspace(*first,last,n*)	row vector x with n elements from *first* to *last*
	A=[x1;x2]	A matrix with rows x1, x2
	ones(N,M), zeros(N,M)	$N \times M$ ones and zeros arrays
	A(i,j)	(i,j) entry of matrix A
	A(i,:), A(:,j)	i-row (j-column) and all columns (rows) of A
	whos	display variables in workspace
	size(A)	(number rows, number of colums) of A
	length(x)	number rows (colums) of vector x
Control flow	for, elseif	for loop, else-if-loop
	while	while loop
	pause, pause(n)	pause and pause n seconds
Plotting	plot, stem	continuous, discrete plots
	figure	figure for plotting
	subplot	subplots
	hold on, hold off	hold plot on or off
	axis, grid	axis, grid of plots
	xlabel, ylabel, title, legend	labeling of axes, plots, and subplots
Saving and loading	save, load	saving and loading data
Information and managing	help	help
	clear, clf	clear variables from memory, clear figures
Operating system	cd, pwd	change directory, present working directory

Table 2. Wavelengths for musical tones in a recorder

Musical note	Wavelength [cm]	Frequency [Hz]
Do	32.79	
Do#	30.95	
Re	29.21	
Re#	27.57	
Mi	26.03	
Fa	24.57	
Fa#	23.19	
Sol	21.89	
Sol#	20.66	
La	19.50	
La#	18.40	
Si	17.37	