Project 1: Introduction to MATLAB for Signal Processing

Lab Report: You will write a formal lab report in double-column IEEE format for each lab. Plots are to be included in the document in the middle of the text. Plots need a caption. You are asked to **label** the axes of your plots and include a title for every plot.
Forgeries and plagiarism are a violation of the honor code and will immediately have penalties (a 0 for the project) and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students and you are allowed to consult old lab reports but the submitted work should be original and it should be your own work.

1 MATLAB concepts

1.1 Overview

Matlab will be used extensively in all the labs. The primary goal of this lab is to familiarize yourself with using Matlab.

Here are three specific goals for this lab:

1. Learn basic Matlab commands and syntax, including the `help` system.
2. Write and edit your own script files in Matlab, and run them as commands.
3. Learn a little about advanced programming techniques for Matlab, i.e., vectorization.

1.2 Getting Started

After logging in, you can start Matlab by double-clicking on a Matlab icon, typing `matlab` in a terminal window, or by selecting Matlab from a menu. The following steps will introduce you to Matlab.

(a) Run the Matlab help desk by typing `helpdesk`. The help desk provides a hypertext interface to the Matlab documentation. Two links of interest are **Getting Help**, and **Getting Started** with MATLAB.

(b) One of the tabs in the Help Desk is called Demos, where you can see information on getting started and some other basics of MATLAB. Run the Matlab demos: type `demo` and explore a variety of basic Matlab commands and plots.

(c) Explore the Matlab help capability available at the command line. Try the following:

```
help
help edit
help plot
help colon        %<--- a VERY IMPORTANT notation
help ops
help zeros
help ones
lookfor filter      %<--- keyword search
```

NOTE: it is possible to force MATLAB to display only one screen-full of information at once by issuing the command `more on`).

(d) **control-C** will stop the execution of any MATLAB command. For example, using **ctl-C** while `lookfor` is running will force it to print out all results found so far.

1.3 Calculate and Plot

(a) Use MATLAB as a calculator. Try the following:

```
pi*pi - 10
sin(pi/4)
ans ^ 2       %<--- "ans" holds the last result
```

(b) Do variable name assignment in MATLAB. Try the following:

```
x = sin( pi/5 );
cos( pi/5 )       %<--- assigned to what?
y = sqrt( 1 - x*x )
ans                 %<--- what value is contained in ans?
```

(c) Complex numbers are natural in MATLAB. The basic operations are supported. Try the following:

```
z = 3 + 4i, w = -3 + 4j
real(z), imag(z)
abs([z,w])      %<-- Vector (or Matrix) constructor
conj(z+w)
angle(z)
exp( j*pi )
exp(j*[ pi/4, 0, -pi/4 ])
```

(d) Use **help** arith to learn how the operation xx.*xx works when xx is a vector; compare array multiplication (dot-star) to matrix multiplication. When unsure about a command, use **help**.

1.4 Matlab Array Indexing

(a) Make sure that you understand the **colon** notation. In particular, explain in words what the following MATLAB code will produce

```
jkl = 0 :  6
jkl = 2 :  4 :  17
jkl = 99 :  -1 :  88
ttt = 2 :  (1/9) :  4
tpi = pi * [ 0:0.1:2 ];
```

(b) Extracting and/or inserting numbers in a vector is very easy to do. Consider the following definition of xx:

```
xx = [ zeros(1,3), linspace(0,1,5), ones(1,4) ]
xx(4:6)
size(xx)
length(xx)
xx(2:2:length(xx))
```

Explain the results echoed from each of the last four lines of the above code.

(c) Observe the result of the following two assignments:

```
yy = xx; yy(4:6) = exp(1)*(1:3)
```

Now write a statement that will take the vector xx defined in part (b) and replace the even indexed elements (i.e., xx(2), xx(4), etc) with the constant π^e. *Use a vector replacement, not a loop.*

1.5 Matlab Script Files

MATLAB Array Operations: There are two kinds of multiplication in MATLAB: matrix multiplication and array multiplication. Many other operations such as division and exponentiation also have this dual character. MATLAB uses a period to change the behavior from a matrix operator to an array operator, e.g., dot-star (.?) instead star (?) for multiplication.

(a) Experiment with vectors in MATLAB. Think of the vector as a set of numbers. Try the following:

```
xk = cos( pi*(0:11)/4 )    %<---comment: compute cosines
```

Explain how the different values of cosine are stored in the vector xk. What is xk(1)? Is xk(0) defined?
NOTES: the semicolon at the end of a statement will suppress the echo to the screen. The text following the % is a comment; it may be omitted.

(b) (A taste of vectorization) Loops can be written in MATLAB, but they are NOT the most efficient way to get things done. It's better to **always avoid loops** and use the colon notation instead. The following code has a loop that computes values of the cosine function. (The index of yy() must start at 1.) Rewrite this computation without using the loop (follow the style in the previous part).

```
yy = [ ];   %<--- initialize the yy vector to be empty
for k=-5:5
   yy(k+6) = cos( k*pi/5 )
end
yy
```

Explain why it is necessary to write yy(k+6). What happens if you use yy(k) instead?

(c) Plotting is easy in MATLAB for both real and complex numbers. The basic plot command will plot a vector y versus a vector x. Try the following:

```
x = [-3 -1 0 1 3 ];
y = x.*x - 3*x;
plot( x, y )
z = x + y*sqrt(-1)
plot( z )      %<---- complex values: plot imag vs. real
```

Use `help arith` to learn how the operation `xx.*xx` works when `xx` is a vector; compare array multiplication (`dot-star`) to matrix multiplication.

When unsure about a command, use `help`.

(d) Use the built-in MATLAB editor to create a script file called `mylab1.m` containing the following lines:

```
tt = -1 :  0.01 :  1;
xx = cos( 5*pi*tt );
zz = 1.4*exp(j*pi/2)*exp(j*5*pi*tt);
plot( tt, xx, 'b-', tt, real(zz), 'r--' ), grid on      %<---
plot a sinusoid
title('TEST PLOT of a SINUSOID')
xlabel('TIME (sec)')
```

Note: *Do not save* this file or any of your MATLAB files to the local hard disk. Your computer account contains a private networked directory where you can store your own files. Use the MATLAB command `addpath()` to allow MATLAB to "see" your personal directory (usually the `Z:` drive).

Explain why the plot of `real(zz)` is a sinusoid. What is its phase and amplitude ? Make a calculation of the phase from a time-shift measured on the plot.

(e) Run your script from MATLAB. To run the file `mylab1` that you created previously, try

```
mylab1            %<---will run the commands in the file
type mylab1       %<---will type out the contents of
                  %    mylab1.m to the screen
```

2 Manipulating Sinusoids with Matlab

Write a MATLAB script file to do steps (a) through (d) below. Include a listing of the script file with your report.

- Generate a time vector `(tt)` to cover a range of t that will exhibit approximately two cycles of the 20-Hz sinusoids defined in the next part, part (b). Use a definition for `tt` similar to part 1.5(d). If we use T to denote the period of the sinusoids, define the starting time of the vector `tt` to be equal to $-T$, and the ending time as $+T$. Then the two cycles will include $t = 0$. **Finally, make sure that you have at least 30 samples per period of the sinusoidal wave.** In other words, when you use the colon operator to define the time vector, make the increment small enough to generate 30 samples per period.

- Generate two 20-Hz sinusoids with arbitrary amplitude and time-shift.

$$x_1(t) = A_1 \cos(2\pi(20)(t - t_{m_1})) \qquad x_2(t) = A_2 \cos(2\pi(20)(t - t_{m_2}))$$

Select the value of the amplitudes and time-shifts as follows: Let A_1 be equal to your age and set $A_2 = 0.8A_1$. For the time-shifts, set $t_{m_1} = 2T/3$ and $t_{m_2} = -T/(M + D)$ where D and M are the day and month of your birthday, and T is the period.

Make a plot of both signals over the range of $-T \le t \le T$. For your final printed output in part (d) below, use `subplot(3,1,1)` and `subplot(3,1,2)` to make a three-panel subplot that puts both of these plots in the same figure window. See `help subplot`.

Note: it is possible to force all three plots to have the same vertical extent by using MATLAB's `axis` command. For example, `axis([-50,50,-3,3])` would make the x-axis run from -50 to $+50$, and the y-axis from -3 to $+3$. Consult `help axis` to obtain more information.

- Create a third sinusoid as the sum: $x_3(t) = x_1(t) + x_2(t)$. In MATLAB this amounts to summing the vectors that hold the values of each sinusoid. Make a plot of $x_3(t)$ over the same range of time as used in the plots of part (b). Include this as the third panel in the plot by using `subplot(3,1,3)`.
- Before printing the three plots, put a title on each subplot, and include your name in one of the titles. See `help title`, `help print` and `help orient`, especially `orient tall`.

2.1 Theoretical Calculations

Remember that the phase of a sinusoid can be calculated after measuring the time location of a positive peak,[1] if we know the frequency.

- Make measurements of the "time-location of a positive peak" and the amplitude from the plots of $x_1(t)$ and $x_2(t)$, and write those values for A_i and t_{m_i} directly on the plots. Then calculate (by hand) the phases of the two signals, $x_1(t)$ and $x_2(t)$, by converting each time-shift t_{m_i} to phase. Write the calculated phases ϕ_i directly on the plots.
  Note: when doing computations, express phase angles in radians, not degrees!
- Measure the amplitude and time-shift of $x_3(t)$ directly from the plot and then calculate the phase (ϕ_3) by hand. Write these values directly on the plot to show how the amplitude and time-shift were measured, and how the phase was calculated.
- Now use the phasor addition theorem. Carry out a phasor addition of complex amplitudes for $x_1(t)$ and $x_2(t)$ to determine the complex amplitude for $x_3(t)$. Use the complex amplitude for $x_3(t)$ to verify that your previous calculations of A_3 and ϕ_3 were correct.

2.2 Complex Amplitude

Write one line of MATLAB code that will generate values of the sinusoid $x_1(t)$ above by using the complex-amplitude representation:

$$x_1(t) = \Re e\{Xe^{j\omega t}\}$$

Use constants for X and ω.

[1]Usually we say time-delay or time-shift instead of the "time location of a positive peak."

3 Aquire and Display signals in MATLAB

The objective is to learn to ?acquire? signals into MATLAB and to display them for visualization. Waveform visualization is an important tool as well as a skill in signal analysis. Signals we learn to process in this course may be generated or recorder internally as a data array within MATLAB, or acquired from external sources such as a wav file. Furthermore, with a proper format converter, you may also record your own voice or music with your own computer and then load the recorded results, after converting them to the supported formats, into MATLAB for processing to produce interesting effects.

Using simple examples, we learned to assign data values to a variable such as: x=10; y=[12345]; yy=[123;456]; A slightly more sophisticated example is:

```
xx = -0.2:0.01:0.3 ;
```

which will create a row vector of dimension 51, with values that span between 0.2 & 0.3 at an increment of 0.01. The indices of a vector zz can be manipulated to perform simple operations such as reversal, e.g.,

```
zz = exp(-0.1*(0:20)); zzReversed = zz(length(zz):-1:1)
```

In this lab, you will learn to use these expressions to generate some interesting signals. Once a signal is generated, an immediate tool we use very often is a plot for visualization and inspection. You will learn to use common plot routines to show, in a number of ways, the signal you have generated or acquired (see below). Often times, we need to process a signal that is obtained by some other means, instead of being generated ?internally? with MATLAB commands. For example, you may have digital music on your computer, stored as files. Or, you may use your own computer to record a sound or your own speech, which can then be stored as an electronic file. In this lab, you will also learn to read or load external data into MATLAB for processing, and to write the processed data or signal into a file for future consumption.

The following list of commands will be used:
plot; subplot; figure; hold; input; audioread; audiowrite; audiorecorder; record; play; getaudiodata
In case you have an early version of MATLAB that does not support audiorecorder and related commands, search MATLAB and/or the web for the appropriate command.

3.1 Generating Sinusoids and Decaying Sinusoids

Generating a sinusoid creating a script file called mySinusoid.m containing the following lines for future use:

```
function xs = mySinusoid(amp, freq, pha, fs, tsta, tend)
% amp = amplitude
% freq = frequency in cycle per second
% pha = phase, time offset for the first peak
% fs = number of sample values per second
% tsta = starting time in sec
% tend = ending time in sec
tt = tsta :  1/fs :  tend; % time indices for all the values
xs = amp * cos( freq*2*pi*tt + pha );
end
```

This function mySinusoid can be called once the argument values are specified. For example, amp

= 7;

```
freq = 70;
pha = -pi/3;
fs = 8000;
tsta = 0;
tend = 2; %a 2-sec long signal xs = mySinusoid(amp, freq, pha,
fs, tsta, tend); %<--- plot first three cycles of the generated
sinusoid ts = tsta:1/fs:tsta+3/freq;
Lt = length(ts);
plot( ts, xs(1:Lt), ?b-?, ts, 2*xs(1:Lt), ?r--?  ), grid on
title(?TEST PLOT of TWO SINUSOIDS (scaling by 2)?)
xlabel(?TIME (sec)?)
```

You may want to try other numbers to appreciate the use of the function. In lecture, the three parameters of a sinusoid have been studied, along with many properties of sinusoids. Here we are most interested in the plotting tool which has many options (so you may want to read its documentation carefully). Now, extend the previous example to make a decaying sinusoid, i.e.,

$$x(t) = Ae^{-bt}\cos(\omega t + \phi)$$

Write a function called myDecayingSinusoid(A, b, omega, phi, fs, tStart, tEnd)
Then pick the correct numbers to generate and plot a decaying sinusoid xDecay that is 3 seconds long, with a frequency of 60 Hz, an amplitude of 6, a phase of ?/3 rads, and a decay parameter b = 0.9. What happens to the plot if we increase the decaying rate to b = 4.

3.2 Reading WAV File into MATLAB and Playing an Array

Many sound files are stored in .wav format; see http://en.wikipedia.org/wiki/WAV for more information about this particular format. Other formats are available, such as mp3; we will focus on .wav in this lab. The MATLAB command audioread is the one to use to read data in wav files into MATLAB data arrays. For example:

```
xx = audioread(‘ece2026Lab01voice.wav’);
```

will load the data in the file ece2026Lab01voice.wav into the array xx. Another command length(xx) will tell you how many values have been read into xx. In many applications, you may need to know what is called the sampling rate, which is the number of sample values per second at which the data was acquired into the file. These parameters can be retrieved as part of the audioread result:

```
[xx, fs] = audioread(?ece2026Lab01voice.wav?);
```

For this exercise, you need to access some wav files. If you have your own inventory of wav files, you are welcome to use them here. (Make sure the file is long enough for the exercise; see below). If not, a file called ece2026Lab01voice.wav can be downloaded from Website for your use. Learn to use the audioread command to read the data into an array, say xx. You can find out how long the signal is in seconds by reading the length of the array via length(xx) which gives you the total number of samples and then divide it by fs, the sampling rate in samples per second. Then, plot the sound wave xx, from t = 0.25 to t = 0.5. You need to know how to translate time into the index of the array; an example has already been shown above.

In MATLAB, an array of data can be *played*to produce audible sound, using:

```
soundsc(xx, fs);
% what is xx and fs???  Make sure you know what these are
```

Try the command yourself and listen to the data you just read in from the file ece2026lab01voice.wav.

3.3 Processing the Data and Writing the Result into a wav File: Time reversal

Following Section 2.2, you have an array xx that contains a signal or a sound. There are many simple oper- ations that can be performed on xx. For example, scaling where we reduce the signal?s amplitude by a half (called attenuation),

```
xh = xx * 0.5;
```

A more complicated operation would be to reverse the time axis of the signal and then plot it.

```
Lx = length(xDecay);
xDecayReversed = xDecay(??:??:??); % figure out how to fill in
those ??s
plot( ...  xDecayReversed ...  ???  )
```

In the plot of xDecayReversed, it should be easy to see the time reversal. If the signal were a sound file, then the output of the time-reversal could be written out to a new wav file:

```
Lxh = length(xh);
xhReversed = xh(??:??:??); % figure out how to fill in those ??s
audiowrite( ?ECE2026lab01outRev.wav?  , xhReversed , fs );
```

If we listen to xhReversed, the sound will be played backwards. Note that the argument fs can be copied from the audioread result. But you can experiment with your own number, such as reducing it to half or doubling it to twice the original value. Using soundsc, play the output wav file ECE2026lab01outRev.wav. You can even combine two operations to have the sound file be played backwards with every other samples skipped.