Full Name: _____

EEL 3135 (Spring 2021) – Lab #01 Due: Jan. 26 - Feb. 1, 2021 (Section Dependent)

Acknowledgement: The material in this and all forthcoming labs are in part based on the labs developed jointly by EEL3135 faculty (namely, Professor Joel Harley and Professor Tan Wong) and peer instructors and, in part, based on labs developed by the authors of the textbooks and the Peer Instructors in the past semesters.

Lab 01 Information:

- The material in this section is informational. Please read through the section as it helps you work on the lab exercises in the next section. There may be code examples in this informational section. You are welcome to copy-and-paste them to MATLAB to run the code, but no submission is needed on any test run.

Why Labs?: This course provides a theoretical and practical basis for the study of Signals & Systems. In the theoretical part of the course, you will be introduced to concepts ranging from complex numbers and Euler's formulas, to frequency-domain and complex-domain analysis of signals and the systems that process them. You'll be introduced to digital filtering, the sampling theorem, the Fourier series, and discrete and fast Fourier transforms.

As you'll see this week, the theoretical part of this course introduces concepts on a mathematical level. In the real world, you as a budding engineer will be asked to process real-world signals. To that end, the practical "lab" portion of this course aims to train you to apply your knowledge using the computational tools that are at your disposal. We ask you to apply your theoretical knowledge to solve problems; first computer-generated ones, then real-world ones. Labs exercises may include create sounds, arranging sounds to make music, observing aliasing, blur and de-blur images, removing noise from audio signals, and using the Fast Fourier Transform to find the heart rate from a real-world electrocardiogram. By the end of the semester, we hope that you'll understand the theory behind all of these things, and be able to apply them in the real world. These labs can be a lot of work, but can also be a lot of fun. Be sure to start early!

Why MATLAB?: These laboratory exercises will be done in the scientific analysis platform from Mathworks called MATLAB. While all the labs could be completed using other popular platforms, such as Python, we decide to teach you how to solve Signals & Systems problems using MATLAB, because MATLAB is a, if not the most, widely adopted scientific analysis partform in the industry.

All MATLAB versions within recent years starting from version R2017a will work for the labs. Some earlier versions may also work. MATLAB can be accessed for free through the ECE cluster `https://view.ece.ufl.edu` or UF Apps `https://apps.ufl.edu`. However, we recommend that you purchase the standalone student version and install on your own computer to use because the standalone version is more convenient, and in case access to the internet is not available.

Purchasing MATLAB: Mathworks offers student version of MATLAB that can be purchased at their website. There are different toolboxes that contain application specific optimized APIs for development. **In our case, the only toolbox needed is the Signal Processing Toolbox.** (This toolbox is already installed in the ECE cluster and UF Apps if you decide to use the free versions of MATLAB there.) There are 2 purchasing options:

- If you only want the Signal Processing Toolbox, the MATLAB core would cost 50 dollars and the Signal Processing Toolbox would cost 10 dollars, a total of 60 dollars.
- Mathworks offers the MATLAB core along with many available toolboxes for 100 dollars.
- We suggest the second option because MATLAB and other toolboxes may be used in other courses. It is more cost effective to purchase them all now.

A Note on Lab Grading: By the time you finish this course, we expect you to have a beginner's working background knowledge for signal processing in the real world. Students that have completed this course have found the experience gained in it to be invaluable in their work as signal processing engineers for real-world companies. As such, our grading is oriented toward two goals:

- We look to see that you have completed your tasks.
- We want to train you to reach a systems-level objective using many small theoretical steps.

At the beginning of this course, we will ask you to follow specific steps en route to a solution. As the course goes on, we will expect you to come up with more and more of the intermediate steps by yourself. TAs will be present and available to assist you, but by the end of this course we want you to be comfortable breaking down real-world signal processing problems and solving them. To that end, in your submissions, we need to see what you did. We want to see that you got the correct result, and that you came up with and thought about the intermediate steps on the way to your result. Therefore, we require you to submit all your code with comments and results when answering the lab exercises. We will describe the MATLAB publishing tool that can help you to do so later.

Basic MATLAB Objects: Before we jump into our full labs, let's briefly look at MATLAB, matrices, and vectors. In MATLAB, almost everything is considered a matrix (hence its name: MATrix LABoratory). For those of you familiar with C, C++, Java, or other languages, a vector is (essentially) a 1-D array and a matrix is (essentially) a 2-D array. I say "essentially" because vectors and matrices can be used in many operations that are not inherent to all 1-D and 2-D arrays (e.g., matrix multiplication). The size of a matrix is written as (# of rows $\times$ # of columns). Consider the following examples:

A 1-D Array of size 1x3. This is called a row vector since it has only 1 row:

$$\begin{bmatrix} a & b & c \end{bmatrix}$$

A 1-D Array of size 3x1. This is called a column vector since it has only 1 column:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Matrix/2-D Array of size 3x3:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Now, let's try to create vectors and matrices in MATLAB. When you open MATLAB and you will see a prompt like this: `>>` in the command window. Since MATLAB is an interactive computing environment, you can start typing at the prompt and you see things happening right away.

Let's start by typing `a=2` and press the Enter key:

```
>> a=2
a =
         2
```

This statement assigns the variable `a` the value `2`. You do not have to declare variables in MATLAB. The basic data type in MATLAB is a matrix of double precision numbers. The variable `a` is automatically set to be a 1x1 matrix by the command above.

We can also define vectors. Below are example row (`1 x 3`) and column (`3 x 1`) vectors.

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

>> c=[2;3;5]
c =
         2
         3
         5
```

We can validate the size by using the `size()` or `length()` function. For example,

```
>> size(b)
ans =
        1  3

>> size(c)
ans =
        3  1

>> length(b)
ans =
        3

>> length(c)
ans =
        3
```

We can also define matrix by combining the above notation. Below is example (`2 x 3`) matrix:

```
>> A=[1 2 3;4 5 6]
A =
        1  2  3
    4  5  6
```

We can validate the size by using the `size()` function. For example,

```
>> size(A)
ans =
        2  3
```

MATLAB Commenting: In MATLAB, commenting is done with the `%` symbol. Anything on a single line after a `%` symbol is considered as comment and is ignored. For example,

```
>> a=1  % assign the value 1 to a
```

ignores everything after the `%` symbol.

MATLAB Help: One of the greatest benefits of MATLAB is its well-documented help system, which can be invoked using the `help` and `doc` commands. For example, typing in the command window:

```
>> help zeros
zeros  Zeros array.
zeros(N) is an N-by-N matrix of zeros.

zeros(M,N) or zeros([M,N]) is an M-by-N matrix of zeros.
```

```
zeros(M,N,P,...) or zeros([M N P ...]) is an M-by-N-by-P-by-...
   array of
zeros.

zeros(SIZE(A)) is the same size as A and all zeros.

zeros with no arguments is the scalar 0.

zeros(..., CLASSNAME) is an array of zeros of class specified by the
string CLASSNAME.

zeros(..., 'like', Y) is an array of zeros with the same data type,
   sparsity,
and complexity (real or complex) as the numeric variable Y.

Note: The size inputs M, N, and P... should be nonnegative integers.
Negative integers are treated as 0.

Example:
x = zeros(2,3,'int8');

See also eye, ones.

Reference page for zeros
Other functions named zeros
```

tells you how to use the `zeros` function and it tells you about **related functions** (`eye` and `ones`, in this case). Using help to find similar functions to those you already know can be extremely valuable in some circumstances. Whenever you see a MATLAB function you do not know, use `help`. If you click on the link to the full documentation, you can usually learn even more about the function. You can also experiment with the `doc` command to get a better documentation interface and sometimes more detailed explanation.

If you don't know what MATLAB functions/commands should be use to perform a task, it may also be helpful to do a descriptive Google search. For example, if you don't know *a priori* how to generate a matrix of all zeros elements in MATLAB, a good descriptive search phase to use in Google could be "generate zero matrix MATLAB." Typically, a sequence of Google searches and uses of the `help` or `doc` command will get you the information needed to perform a task in MATLAB. That's how I program in MATLAB. Of course, you can always get help from the TAs and the instructors.

Writing Scripts in the Editor Window: You will be writing and submitting MATLAB scripts throughout this class. Start a new script by clicking on "New Script" in the upper-left in the "HOME" toolbar. This should produce an editor window, which is effectively an IDE for MATLAB. You may also start the editor window by typing `edit` in the command window of MATLAB. You

can then enter your MATLAB commands and/or code in the editor window. Remember to click on the "Save" button to save your script. You can then sequentially run the script by clicking the "Run" button in the "EDITOR" toolbar. You may also run the script by typing its name (without the suffix) in the command window after the script has been saved to the "Current Folder" in MATLAB. You can also use many standard software development tools, such as setting up break points and stepping through your code, in the editor window.

For example, type

```
a=2
b=[1 2 3]
c=[2;3;5]
```

into the editor window and press "Run." The same output should appear in the command window as in the last section. We typically want to suppress output to the command window. To do that, end each line with a semicolon `;`. Now instead type into the editor window

```
a=2;
b=[1 2 3];
c=[2;3;5];
disp(b*c);
```

to suppress command window displaying the results, except for specifically displaying `b*c`. Try saving the script as `abc.m` (the suffix `.m` indicates that the file is a MATLAB script) using the "Save" button and run the script again by typing the script's name `abc` in the command window:

```
>> abc
     23
```

Do you know why the output of the line `disp(b*c);` is `23`?

MATLAB Functions: In addition to working interactively with MATLAB by typing commands after the prompt in the command window, you can write functions in the MATLAB editor. Here is a very simple function:

```
function [y,Fs] = timescale(inputFile,outputFile,scaleFactor)
  [y, Fs] = audioread(inputFile);
  Fs = Fs*scaleFactor;
  audiowrite(outputFile,y,Fs);
end
```

This function takes as input the filename of an audio file with a '.wav' extension, the filename to save the output, and the scale factor to scale the sampling rate of the audio signal. The function outputs the audio signal along with its properties. Type `help audioread` to know more about `Fs`. Time-scaling an audio signal makes the signal play faster or slower depending on the scale factor. Write this code in the MATLAB editor window, and save it as `timescale.m`. Use the `chirp.wav` file distributed with this lab. You play the saved .wav file by clicking on it or by using `soundsc` in MATLAB.

To call this function, type:

```
>> [y,Fs] = timeScale('chirp.wav','chirpFaster.wav',2);
>> soundsc(y,Fs);
>> [y,Fs] = timeScale('chirp.wav','chirpSlower.wav',0.5);
>> soundsc(y,Fs);
```

If MATLAB complains it cannot find this function, you have to make sure the function is under the MATLAB path, or the files `timeScale.m` and `chirp.wav` appear under "Current Folder." To edit the path, go to File→Set Path and specify the folder you want to include in the path. The order of the paths is relevant, that is, if two functions from two different folders are both under the path, then the function from the first path will be called.

MATLAB Plotting: Plotting with MATLAB is easy. Execute the following code one by one in the command window (or using a script)

```
>> stem([1:5],[3 6 2 4 1])
>> plot([1:5],[3 6 2 4 1])
>> plot([1:5],[3 6 2 4 1],'r+')
```

If you feel uncomfortable with the display, try the axis command as follows:

```
>> axis([-1 10 0 7])
```

What does this command do? Note that after you plot a figure (with `stem`, `plot`, etc.), you can export and save it in different formats. Check out the available formats under File→Save As.

MATLAB Loops: MATLAB has both 'for' loops and 'while' loops. We will generally use 'for' loops in this course. To write a 'for' loop, execute the following code.

```
>> for n = 1:10
>> disp(['Number ' num2str(n)])
>> end
```

The `n = 1:10` indicates the range of numbers (from `1` to `10`) that the loop executes over. To write a 'while' loop that does the same operations, execute the following code.

```
>> n = 0;
>> while n <= 10
>> disp(['Number ' num2str(n)])
>> n = n + 1;
>> end
```

MATLAB If-Else statements: By using 'if' and 'if-else' statements, you can conditionally execute specific commands. Try the following in a script and run it a few times:

```
rnumber = rand;
if rnumber > 0.5
disp('Large Number')
else
disp('Small Number')
end
```

What do these commands do? The 'if-else' statement is useful when your current analysis depends on previous results.

Publishing MATLAB Scripts: We require all lab reports be submitted in PDFs generated using the MATLAB Publisher. An explanation on how to use the tool can be found here.

Generatelly, the publisher also uses commenting (i.e., the `%` symbol) to process the script to be published, as described before. For the labs, you may often use `%%` to separate the code into sections in your MATLAB script and `disp()` to provide any answers to the questions that are asked in the lab exercises. You can also insert an image into your report by using `% <<FILENAME.PNG>>`. This may come in handy when you want to insert handwritten solutions into your published report. If you know LaTeX, you can insert nicely typeset equations using `% $$LaTeX code$$`.

Try entering the following script using the MATLAB editor:

```
%% Matrix multiplication example:
b=[1 2 3]
c=[2;3;5]
disp('b*c = ');
disp(b*c);
%% Plot a curve:
plot([1:5],[3 6 2 4 1]);
```

Click publish and be sure to save as a PDF. In the main MATLAB window, there should be a "PUBLISH" toolbar. Use the right drop-down menu under "Publish" (right-most button) and click on the "Edit Publishing Option." Change the "Output file format" under "Output settings" to "pdf." Then save this as the new User Default. This will automatically publish your code as a PDF to submit as your lab report.

Lab 01 Questions

- Your laboratory solutions should be submitted on Canvas as a single PDF. Use the skeleton code provide with the lab to answer each question.
- Use the provided skeleton code as the basis for your solutions (easier for you and the graders).

Question #1: Decide on a Team Name! In Slack, change your private channel called `<Day>-<Time>:Team-<Num>` to `<Day><Time>:Team-<Name>`

Question #2: Provide (1) the names of every other member of your Lab Team and (2) one thing each person wants to learn about in this course.

Question #3: Comment every line of the following MATLAB script (also in the skeleton code). Say what each line is doing in your comment. Explain each MATLAB line by using no more than one comment line, as done in the first line.

```
b=ones(3,2)
c=size(a);
abs([-5.2 , 3])
floor(3.6)
d=[1:-3.5:-9];
f=d(2); g=sin(pi/2);
K=[1.4, 2.3; 5.1, 7.8];
m=K(1,2);
n=K(:,2);
comp = 3+4i;
real(comp)
imag(comp)
abs(comp)
angle(comp)
disp('haha, MATLAB is fun');
3^2
4==4
[2==8 3~=5]
x=[1:2:8];
y=[5 7 6 8];

q = zeros(10,1);
for ii = 1:10
    q(ii) = ii^2;
end
figure(1021);
stem(x,y)
hold on;
plot(x,y, 'k', 'linewidth', 2)
plot(x,y,'+r', 'markersize', 20);
hold off;
xlabel('Horizontal Axis')
ylabel('Vertical Axis')
```

Question #4: With MATLAB, plot each of the following functions.

(a) Use MALTAB to generate the following discrete-time signals:

```
vect1 = [0 pi/4 2*pi/4 3*pi/4 4*pi/4 5*pi/4 6*pi/4 7*pi/4];
vect2 = cos(vect1);
```

Plot `vect2`. Use the `stem` function. The horizontal axis should be in radians. Label the vertical axis "Amplitude" and label the horizontal axis "Angle (Radians)".

(b) Use MALTAB to generate the following discrete-time signals:

```
theta = 0:pi/20:3*pi;
y = cos(theta);
```

Plot `y`. Use the `plot` function. The horizontal axis should be in radians. Label the vertical axis "Amplitude" and label the horizontal axis "Angle (Radians)".

(c) Plot $\cos(20\pi t)$ for values of t in the range $-3 \leq t \leq 3$. Choose vector for t that creates a smooth sinusoid in the plot. Use the `plot` function. Label the vertical axis "Amplitude" and label the horizontal axis "Time (Seconds)".

Question #5: This exercise shows you how to write a MATLAB function that calculates all the complex-valued solutions to the following polynomial:

$$z^n = a$$

where a is a complex number and n is a positive integer. The solutions are called the nth roots of the complex number a. For the special case of $a = 1$, the solutions are called the *nth roots of unity* (see Appendix A in the textbook). **Hint:** Let $a = Ae^{j\phi}$ be the polar form of a. Then there are exactly n solutions/roots, for $k = 0, 1, \ldots, n-1$, given by

$$z = A^{\frac{1}{n}} e^{j\frac{\phi+2\pi k}{n}} = \underbrace{\left(A^{\frac{1}{n}} e^{j\frac{\phi}{n}}\right)}_{r_1} \cdot \underbrace{\left(e^{j\frac{2\pi k}{n}}\right)}_{r_2},$$

(a) Write a MATLAB function `r = myroots(n, a)` to find the nth roots of any complex number by completing the skeleton function provided in `myroots.m`. The function should take a and n above and return all of the roots `r`. Remember to comment your code.[1]

(b) After the generation of `myroots.m`, type in the command window,

```
>> help myroots
```

Explain: What purpose does this command serve?

(c) Use your function to calculate the 9th roots of 2 and the 23rd roots of $-j$. Check your results!

[1] It is a good practice to print out intermediate results in your function code for debugging when you are developing your function. However, please turn off printing of all intermediate results (do you know how?) in the submitted version of your code to avoid generating extraneously long published report.