University of Washington
Department of Electrical Engineering
EE 235 Lab 1 Part Two:
Introduction to Matlab

In this lab, you will work through another series of exercises to finish off your introduction to Matlab ("Matrix Laboratory"). **Note: All lab exercises and the accompanying report for this lab should be completed as pairs.**

What Is Expected from You in Lab 1 Part Two

- Completion of 3 more in-lab check offs with TA (5 points)
- Completion of a lab report summarizing all 5 exercises (10 points)

EXERCISE #3: Matlab Plotting Basics

Thus far, we have used Matlab to perform matrix-based calculations. Matlab is also very useful for data visualization. In this exercise, we will learn the basics of plotting functions in Matlab. In the end, you should have an understanding of the following concepts and functions:

New Matlab Concepts/Functions	
• Loading a new figure window • Plotting a function • Labeling your plots	• Turning on the figure grid • Adjusting axes limits • Plotting multiple plots using subplot

1) ***Lab Practice***:
   a) ***Plotting Basics***.
      i) To plot signals, we will make use of the following basic commands and functions:

figure	Load a new figure window for plotting
plot( )	Plot a signal
title( )	Add title to a figure
xlabel( )	Add label for x-axis
ylabel( )	Add label for y-axis
grid on	Turn on grid lines
xlim( )	Adjust limits on x-axis
ylim( )	Adjust limits on y-axis

      ii) Let us practice by plotting **x(t) = 2t**, **0 ≤ t ≤ 5**, with a sampling period of 1. We will adjust the x-axis so that the limits are from -2 to 7. In addition, we will adjust the y-axis so that it covers the range from -2 to 12.

- Define time samples: >> t = 0:5;
- Compute signal **x(t)**: >> x = 2*t;
- Load figure window: >> figure;
- Plot the signal **x(t)** vs. **t**: >> plot(t, x);
- Add a title: >> title('x(t) = 2t');
- Label x axis: >> xlabel('t');
- Label y axis: >> ylabel('x(t)');

- Turn on grid: >> grid on;
- Adjust x-axis: >> xlim([-2 7]);
- Adjust y-axis: >> ylim([-2 12]);

b) ***Multiple Plots on One Window***.

i) To graph multiple plots on one window, you can use the **subplot** function. This function takes in three arguments as follows: **subplot(m, n, p)**. The first two arguments are used to break the window into an **m by n** grid, while the third argument selects where to place the following plot on the window.

ii) To practice, suppose we want to create a 2 x 2 figure window and hence 4 plots total. Therefore, the argument **p** to **subplot(m, n, p)** can be one of the following p = 1, 2, 3, or 4. The window will be filled in starting with the top left, then proceeding to the right on the first row before going to the next row, as shown below.

p = 1	p = 2
p = 3	p = 4

iii) Let us illustrate this with an example:

- Define time samples: >> t = 0:0.1:1;
- Define the 4 signals: >> x1 = t;
  >> x2 = -2*t;
  >> x3 = 3*t.^2;
  >> x4 = -4*t.^2;
- Load figure window: >> figure;
- Access 1st subplot: >> subplot(2, 2, 1);
- Plot x1 vs. t: >> plot(t, x1);
- Access 2nd subplot: >> subplot(2, 2, 2);
- Plot x2 vs. t: >> plot(t, x2);
- Access 3rd subplot: >> subplot(2, 2, 3);
- Plot x3 vs. t: >> plot(t, x3);
- Access 4th subplot: >> subplot(2, 2, 4);
- Plot x4 vs. t: >> plot(t, x4);

iv) When using subplots, the plots must be labeled individually. To individually label a plot, you will need to access a specific plot using the **subplot** function. Then, you can proceed to add the labels or make any other changes. Let us title, label, and adjust the y-axis only for the 2nd subplot:

- Access 2nd subplot: >> subplot(2, 2, 2);
- Add title: >> title('x2(t)');
- Label x-axis: >> ylabel('x2(t)');
- Adjust limits: >> ylim([-4 4]);

c) ***Importance of Sampling Frequency***. In the last exercise, we defined the term sampling frequency **Fs**. We will use this practice section to visualize the importance of the sampling frequency selection.

i) Let us first plot **x(t) = cos(2πt)**, **0 ≤ t ≤ 5**, with a sampling frequency Fs = 10

- Define sampling frequency: >> Fs = 10;
- Define time samples using colon operator: >> t = 0:1/Fs:5;
- Generate signal **x(t)**: >> x = cos(2*pi*t);
- Open figure window: >> figure;
- Create 2 x 1 subplot window and plot as 1st plot: >> subplot(2, 1, 1);
- Plot function: >> plot(t, x);

ii) Let us now decrease the sampling frequency to Fs = 1:

- Define sampling frequency: >> Fs = 1;
- Define time samples: >> t = 0:1/Fs:5
- Generate signal **x(t)**: >> x = cos(2*pi*t);
- Plot on current figure window as 2nd subplot: >> subplot(2, 1, 2);
- Plot signal: >> plot(t, x);

You should see that we do not get an accurate plot of x(t) for the case when Fs = 1. This is because the number of samples used per second is not high enough. In other words, the period between samples is too long. In this case, we are sampling the signal at times when the cosine signal is always 1! Therefore, the selection of the sampling frequency is very important, which will be discussed in a future lab.

2) ***Lab Exercise***: Open a new script and call it **Ex3.m** with the outline below:

```
% FILE: Ex3.m
% NAME: [FILL IN NAME HERE]
% DESCRIPTION: Matlab Plotting Basics

% Clear all variables and close all windows
clearvars;
close all;

% PART A


% PART B
```

a) For PART A: Define a signal **x(t) = 2 – t** over the range **-2 ≤ t ≤ 4** with a sampling frequency of Fs = 5. Plot **x(t) vs. t** on a 1 x 2 figure window as the 1st subplot. Adjust range of the x-axis to be between -5 and 5, as well as range of the y-axis to be between -5 and 5. Turn on the grid and be sure to label the plot and axes appropriately.

b) For PART B: Using the same time samples from (a), define and plot $\mathbf{y(t) = -0.5t^2}$ on the 2nd subplot. Adjust range of x-axis to be between -5 and 5, as well as range of y-axis to be between -5 and 5. Turn on the grid and be sure to label the plot and axes appropriately.

3) ***Lab Check-Off***: Demonstrate your script for **Ex3.m** with your Lab TA**.**

EXERCISE #4: Sound Files

In this exercise, we will learn how to play and plot sound files. We will use sound files that are already built into Matlab. In the end, you should have an understanding of the following concepts and functions:

New Matlab Concepts/Functions	
• Loading a Matlab data file • Playing a sound file	• Plotting a sound file • Computing time samples vector

1) ***Lab Practice***:
   a) ***Loading and Playing Sound Files***
      i) The built-in sound files we will use are stored in a Matlab data file with extension ***.mat**. These data files will carry two variables:

         - **y**: This is the audio samples vector
         - **Fs**: This is the sampling frequency (or rate) of the vector **y**

      ii) To load the variables in the sound file into your workspace, we will use the command **load**. As an example, let us load the data file **train.mat**:

```
>> load train.mat;
```

      iii) We will play sound files using the function **sound**. The function **sound** takes in two arguments, the audio samples vector **y** and the sampling rate **Fs**, and returns no value. It would play the audio samples in vector **y**. It has the following header: **sound(y, Fs)**. Since the train file is now loaded, let us practice playing the train sound file:

```
>> sound(y, Fs)
```

   b) ***Plotting Sound Files***
      i) To plot a sound file versus time, we need its audio samples vector and its corresponding time samples vector. We can compute the time samples vector by using the sampling rate **Fs** as follows:

         - Suppose the number of elements in a vector **y** is 12,880. If we start counting at 0, this means the counts will range from 0 to 12,879, or more generally, 0 to **length(y) - 1**

         - Suppose the sampling rate is **Fs = 8192**, which means there are 8,192 samples per second. In terms of the sampling period, **Ts** = 1 / 8192 = 0.000122 seconds per sample, which means there is a time increment of 122 μsec between sound samples.

         - To convert our counts 0 to 12,879 to numbers in seconds, we just need to multiply the counts by the sampling period:

```
>> t = (0:length(y) – 1) * (1/Fs);
```

      ii) As an example, let us plot and play the sound signal from **splat.mat**:

- Load sound Matlab data file: >> load splat.mat
- Compute time vector: >> t = (0:length(y)-1) * (1/Fs);
- Load figure: >> figure;
- Plot sound file: >> plot(t, y);
- Play sound file: >> sound(y, Fs);

2) ***Lab Exercise***: Open a new script and call it **Ex4.m** with the outline below:

```
% FILE: Ex4.m
% NAME: [FILL IN NAME HERE]
% DESCRIPTION: Sound Files

% Clear all variables and close all windows
clearvars;
close all;

% PART A

% PART B

% PART C

% PART D
```

a) For PART A:
   i) Load sound file **chirp.mat**. Store **y** in variable **chirpSound** and **Fs** in **chirpFs**
   ii) Load sound file **gong.mat**. Store **y** in variable **gongSound** and **Fs** in **gongFs**

b) In PART B:
   i) Using **chirpFs**, compute the time samples for **chirp** and call that vector **t_chirp**
   ii) Using **gongFs**, compute the time samples for **gong** and call that vector **t_gong**

c) In PART C: Using the sound and time sample vectors from (a) and (b), plot the two sound signals on one figure window using a 2 x 1 subplot window. No need to adjust the axes or turn on the grid, but do label the plot and axes appropriately.

d) In PART D: Using **chirpSound** and **gongSound**, along with their corresponding sampling frequency variables, Play the two sound files in succession with a 4-second pause in between. You can use the function **pause** to help you do this. Type **help pause** on the COMMAND window for assistance on the appropriate function syntax.

3) ***Lab Check-Off***: Demonstrate your script for **Ex4.m** with your Lab TA**.**

EXERCISE #5: Functions and Wrap-Up

Up to this point, you have written Matlab scripts, which are simply a sequence of commands that Matlab will run. Matlab functions, on the other hand, can take in parameters and output return values just like other programming languages. In this last exercise, we will learn how to write a simple one-argument function with no return value, which should set you up well for Lab #2. In the end, you should have an understanding of the following concepts:

New Matlab Concepts/Functions	
• Writing a Matlab function	• Calling a Matlab function

1) ***Lab Practice***:
   a) ***Writing Functions***
      i) Like Matlab scripts, Matlab functions are stored in a file with extension *.m and are referred to as M-files.

      ii) The syntax to declare a function (or define a function header) is similar to other programming languages. One of the main differences is the inclusion of the keyword ***function*** in the function header. An example one-argument function declaration with no return value is given below:

         function myexample(s)

      iii) Functions must be saved in an M-file with the same name as the actual function. As an example, let us create a function called *myexample* that will play the chirp sound 3 consecutive times. A pause in between will be determined by the incoming parameter **s**

- Open a new script and save it as **myexample.m**
- Type in or copy the following lines of code

```
function myexample(s)

        load chirp

        sound(y, Fs);
        pause(s);
        sound(y, Fs);
        pause(s);
        sound(y, Fs);
end
```

- Include the following function header and in-line comments that begin with a percent % symbol

```
% MYEXAMPLE Play chirp sound 3 times
% USAGE: myexample(2) plays chirp sound 3 times with a 2 sec pause
% AUTHOR: Eldridge Alcantara
function myexample(s)

        % Load chirp sound
        load chirp

        % Play sound vector
        sound(y, Fs);
        pause(s);
        sound(y, Fs);
        pause(s);
        sound(y, Fs);

end
```

- Run your function from the COMMAND window
  - First try with a pause of 4 seconds: >> myexample(4)
  - Now try with a pause of 1 second: >> myexample(1)

2) ***Lab Exercise***: You will now write your own function that will amplitude scale the train sound based on a user input parameter. The original sound and the scaled sound file will also be plotted and played. To start, open a new script and call it **ExAmpScale.m** with the outline below:

```
% EXAMPSCALE [FILL IN YOUR OWN DESCRIPTION]
% USAGE: [FILL IN EXAMPLE USAGE]
% AUTHOR: [FILL IN NAMES HERE]
function ExAmpScale(A)

        % PART A

        % PART B

        % PART C

        % PART D

end
```

a) For PART A:
   i) Load sound file **train.mat**. Note the MAT file will provide two variables: **y** and **Fs**
   ii) Using variable **y**, compute the corresponding time samples and call it **t_y**

b) For PART B:
   i) Using the parameter **A**, scale the values of **y** and store it in a new vector **yScaled**

c) For PART C:
   i) Close all previous plot windows.

ii) Using a 1 x 2 subplot window, plot the original sound vector **y** on the 1st plot and the scaled sound vector **yScaled** on the 2nd plot. Adjust the y-axis to -3 to 3 on both plots. No need to adjust the x-axis or turn on the grid, but do label the plot and axes appropriately.

d) For PART D:
   i) Play the original sound file **y** followed by the scaled sound file **yScaled** using a 4 second pause in between

e) Test your function from the COMMAND two times. In the first round, use an amplitude scale factor of 3. In the second round, use an amplitude scale factor of 0.25.

3) ***Lab Check-Off***: Demonstrate your function calls to **ExAmpScale.m** with your Lab TA**.**

FOR NEXT WEEK: LAB REPORT DUE

1) Use the lab report format on the class website
2) Turn in a PDF of your lab report (one per team) online via link on class website
3) Report is due prior to the start of your next lab section
4) Lab turn-in times will be checked, so no late lab reports will be accepted