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

In this lab, you will work through a series of exercises to get you started in Matlab ("Matrix Laboratory"). You will learn how to use Matlab variables, perform basic operations, and write your own script files. **Note: All lab exercises and the lab report should be completed as pairs.**

What Is Expected From You In Lab 1 Part One

- Completion of 2 in-lab check offs with TA highlighted in a blue box (5 points)

PRE-LAB: Install Remote Desktop Connection

If you have not done so for a previous class, install Remote Desktop Connection (RDC) on your laptop so you can install Matlab. Follow the instructions on the EE website here: https://www2.ee.washington.edu/computing/faq/labs/remote_computing.html

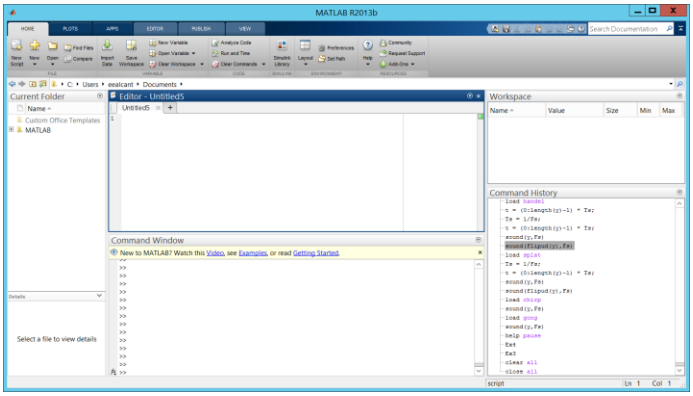
EXERCISE #1: Matlab Variables and Basic Operations

In this exercise, we will learn about variables and basic operations in Matlab. In the end, you should have an understanding of the following concepts and functions:

New Matlab Concepts/Functions	
• Creating Matlab variables • Suppressing output in command window • Accessing element(s) in vector • Accessing element(s) in matrix	• Getting length of vector • Getting size of a matrix or vector • Getting number of rows or columns in matrix or vector

1) ***Lab Practice***:
   a) Log in to your EE account and then load the current version of **Matlab** on your system. You may need to search for Matlab using the Search charm on Windows.

   b) Once Matlab is running, adjust the window locations until it looks like the one below using Home -> Layout. To get the EDITOR window to show up, click Home Tab -> New Script. Ask the Lab TA if you have trouble configuring Matlab this way.



   c) There are 5 important windows:
      i) **Current Folder**: This shows the current folder you are working in. This directory is where Matlab expects to find all your files (M-files, audio, etc).

ii) **Editor**: This is where you will type and save your M-files (more on M-files later).

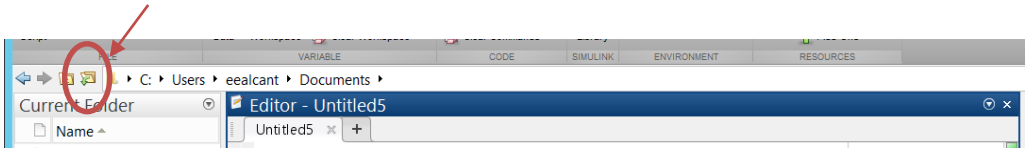
iii) **Command Window**: This is where you type commands. Press ENTER to run a command**.**

iv) **Workspace**: This displays information about all active variables. You should adjust the workspace window displays Name, Value, Size, Min, and Max. If any of these are omitted, add them by right-clicking on of the column names and selecting the missing columns.

v) **Command History**: This keeps track of all the commands you have entered.

d) ***Setting up Current Folder***. We need to change your current folder so that all your work is saved in a folder dedicated to Lab 1. To do this, follow these set of instructions

i) Click the button "Browse For Folder"



ii) If you are not already in this folder, find the MATLAB folder under Documents.

iii) Under folder MATLAB, create a new folder by right-clicking in CURRENT FOLDER window and selecting New Folder. Call this folder **Lab 1**

iv) Double-click the folder **Lab 1** to go into that folder, and then press "Select Folder"

e) ***Practice with Matlab Variables***.

i) In Matlab, all variables are matrices and can be classified as one of the following:

Variable Type	In Math	In Matlab as a Matrix
Scalar	$x = 1$	x is a 1x1 matrix
Column Vector	$y1 = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$	y1 is a 3x1 matrix
Row Vector	$y2 = [1 \quad 3 \quad 5]$	y2 is a 1x3 matrix
Matrix	$z = \begin{bmatrix} 3 & 6 & 9 \\ 1 & 1 & 1 \end{bmatrix}$	z is a 2x3 matrix

ii) Like many other programming languages, variables can be defined in Matlab by typing: **<VariableName> = <InitialValue>**

iii) Let us now practice creating the Matlab variables above on the COMMAND window. As you work on the COMMAND window, notice how the WORKSPACE WINDOW changes:

- Create the scalar **x** by typing and then pressing ENTER:
  >> x = 1
- Create the column vector **y1** by typing and then pressing ENTER:
  >> y1 = [2; 4; 6]
- Create the row vector **y2** by typing and then pressing ENTER:
  >> y2 = [1, 3, 5]
  Note: you can also create a row vector by using spaces instead of commas:
  >> y2 = [1 3 5]
- Create the matrix **z** by typing and then pressing ENTER:
  >> z = [3 6 9; 1 1 1]

f) ***Suppressing Output in Command Window***. Notice that output is displayed on the COMMAND window after each line of code you type in. You can suppress the output by ending your statement with a semicolon. Note the difference between the two lines of code. On the COMMAND window, type the following and press ENTER:

- Suppress the output on the command window: >> x = 1;
- Display the output on the command window: >> x = 1

g) ***Practice with Basic Operations***.

i) *To access an element in a vector*: We will use the ( ) operator and specify the index we want to access. Unlike other programming languages, Matlab is 1-indexed, which means the first element in a vector or matrix is at index 1 and not at index 0. Let us access the first element in vector **y1**. On the COMMAND window, type:
>> y1(1)

ii) *To access (or extract) a range of elements in a vector*: We can use the colon operator and specify the start index and end index we want to access. Let us extract the first two elements in **y1** and change the values to -1 by typing the following:
>> y1(1:2) = -1

iii) *To access an element in a matrix*: You must specify two indices (row, column). Let us extract the element in the 1st row and 2nd column of matrix **z**, and then change the first two columns in the 2nd row to -2 by typing the following:
>> z(1,2)
>> z(2, 1:2) = -2

iv) *To get the length of a vector*: You can use the **length** function. Let us get the length of column vector **y1** by typing the following:
>> length(y1)

v) *To get the overall size of a matrix or vector*: You can use the **size** function. Let us get the size of matrix **z** by typing the following:
>> size(z)

vi) *To get the number of rows or columns in a matrix or vector*: You can use the **size** function and specify an additional parameter 1 or 2, where 1 gives the number of rows and 2 gives the number of columns
- Let us get the number of rows in matrix **z**: >> size(z, 1)
- Let us get the number of columns in matrix **z**: >> size(z, 2)

2) ***Lab Exercise***: Let us now practice what you have learned with a short exercise. To aid in the programming process, we will write code using M-files which have the extension *.m. M-files can be used as scripts that contain a sequence of commands that are executed exactly as if they were manually typed into the Matlab COMMAND window.

a) Open up a new script by doing the following: Go to Home tab -> New Script

b) Save your script: Go to Editor tab -> Save. Then, save file as **Ex1.m**

c) On the Matlab script window on the upper center window, first copy or type in the following lines of code and comments (which are done in Matlab using the % symbol):

```
% FILE: Ex1.m
% NAME: [FILL IN NAME HERE]
% DESCRIPTION: Matlab Variables and Basic Operations

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

% PART A

% PART B

% PART C

% PART D
```

d) Under PART A:
i) Write the line of code to create a 3 x 1 column vector called **y1** with values 4, 6, and 2. Suppress the output using a semicolon at the end of the line of code.

ii) Write the line of code to a 3 x 3 matrix called **z**, where the first row has all ones, the second row has values 3, 6, and 9, and the third row has all zeros. Again, suppress the output by using a semicolon at the end of the line of code.

iii) Test your current script by pressing <F5> or by going to Editor -> Run. You should see your two variables **y1** and **z** on your WORKSPACE window.

e) Under PART B:
   i) Write the line of code to extract the value 6 in vector **y1**, and store that value in variable **c**. That is, write the following line of code:

      c = FILL IN REST OF CODE

      Display the output so we can see the value of **c** on the COMMAND window. Therefore, omit the semicolon at the end of the line of code.

   ii) Using the colon operator, write the line of code to extract the last two elements in vector **y1** and store them in variable **c1**. Display the output by omitting the semicolon.

   iii) Test your script so far by pressing <F5> or by going to Editor -> Run. You should see the variables **y1**, **z**, **c,** and **c1** on your WORKPSACE window. You should also see the values of **c** and **c1** displayed on the COMMAND window.

f) Under PART C:
   i) Write the line of code to access the value 9 in matrix **z**, and store that value in variable **d**. Display the output on the COMMAND window.

   ii) Using the colon operator, change the values of matrix **z** so that the last two columns in the 3rd row are -1. Display the output on the COMMAND window.

   iii) Test your script so far by pressing <F5> or by going to Editor -> Run. You should see the variables **y1**, **z**, **c**, **c1**, and **d** on your workspace window. You should also see the values of **c**, **c1**, **d**, and **z** displayed on the COMMAND window.

g) Under PART D:
   i) Write the line of code to get the number of columns in vector **y1**, and store that value in variable **e**. For this part, display the output on the COMMAND window.

   ii) Test your script so far by pressing <F5> or by going to Editor -> Run.

3) ***Lab Check-Off***: Demonstrate your script for **Ex1.m** with your Lab TA**.**

EXERCISE #2: More Matlab Variables and Basic Operations

In this lab exercise, we will practice performing other basic Matlab operations. In the end, you should have an understanding of the following concepts and functions:

New Matlab Concepts/Functions	
• Matrix addition or subtraction • Transposing a matrix • Matrix scalar multiplication	• Squaring elements of a matrix • Creating time samples vector • Accessing signal at particular time(s)

1) ***Lab Practice***:

a) ***Addition or Subtraction Operator***. In Matlab, we have the ability to add, subtract, multiply, and divide matrices. Let us start with matrix addition and subtraction. If you have taken a course in linear algebra or another math course where matrices were used, you know that matrices can only be added or subtracted under certain conditions.

i) We can only add or subtract matrices with the same dimensions. To illustrate this, type the following on the COMMAND window:

- Create a 3 x 2 matrix called **a**: >> a = [2 2; 4 4; 6 6];
- Create another 3 x 2 matrix called **b**: >> b = [2 2; 2 2; 2 2];
- Add them together and display out the output: >> a + b

Notice we get an output from Matlab without an error.

ii) We cannot add or subtract matrices of different dimensions. To illustrate this, type the following on the COMMAND window:

- Create a 2 x 3 matrix called **c**: >> c = [1 1 1; 1 1 1];
- Subtract **c** from **a**, and display the output: >> a – c

Notice we get an error from Matlab: *Matrix dimensions must agree.*

b) ***Transpose Operator***. In Matlab, you can turn a row vector into a column vector, and vice versa. You can also restructure a matrix so that the row elements now become column elements. This operation is called transposing a matrix or vector.

i) We can transpose a matrix by using the ' operator:

- Create and display 3 x 2 matrix called **d**: >> d = [1 1; 2 2; 3 3]
- Transpose matrix **d**, store in **e**, and display: >> e = d'

Notice how the row elements of **d** have now become the column elements in **e**.

ii) Transposing is useful when operating on a combination of row and column vectors with the same length. Using the same vectors **a** and **c** from (b):

- Subtract **c** from **a** without error: >> a – c'

c) ***Multiplication Operator***. In Matlab, there are 3 type of multiplication operations: scalar multiplication, matrix multiplication, and array multiplication. For this lab, we will only focus on scalar multiplication. Matrix multiplication and array multiplication will be introduced in future labs.

i) In scalar multiplication, you take a number (a scalar) and multiply every element in the matrix by this scalar value. To illustrate this,

- Create a 2 x 2 matrix called **A** and display output: >> A = [1 2; 3 4]
- Perform scalar operation **2A** and display output: >> 2 * A

Notice how every element in A is now scaled by 2.

ii) We can square each element of a matrix using the .^ or .* operator:

- Perform $\mathbf{A^2}$ using first operator and display output: >> A.^2
- Perform $\mathbf{A^2}$ using second operator and display output: >> A.*A

d) ***Creating a Signal that is a Function of Time***. In EE 235, we deal with continuous-time signal processing. However, in Matlab, there are two issues. First, Matlab can only handle finite-range signals. This means defining a signal over the range $\mathbf{-\infty \leq t \leq \infty}$ is impossible; we can only define a signal over a range like $\mathbf{-2 \leq t \leq 10}$. Second, Matlab only deals with digitized representations of a signal. This means we have to specify the samples of time we want to use and with what particular granularity or sampling period.

i) To illustrate this, let us practice by defining a signal $\mathbf{x(t) = 2t}$ over the range $\mathbf{0 \leq t \leq 5}$ with a sampling period of 1. This sampling period means you will compute **x(t)** at t = 0, 1, 2, 3, 4, and 5.

- Define time samples vector **t**: >> t = [0 1 2 3 4 5];
- Compute signal **x(t) = 2t**: >> x = 2 * t;
- Display values of **x**: >> x

ii) Let us now increase the time range from -2 to 10 with the same sampling period of 1. Instead of defining every time sample in the vector, we can define the samples by using the colon operator. The general form of the colon operator is **j:k**, which will create a row vector with values [j, j+1, … k].

- Define time samples vector **t**: >> t = -2:10;
- Compute signal **x(t) = 2t**: >> x = 2 * t;
- Display values of **x**: >> x

iii) Let us use the time range from 0 to 2 with a sampling period of 0.5. We can use the second form of the colon operator **j:i:k**, which creates a row vector with values [j, j + i, j + 2i, … k].

- Define and display time samples vector **t**: >> t = 0:0.5:2
- Compute signal **x(t) = 2t**: >> x = 2 * t;
- Display value of **x**: >> x

iv) In many examples, you will the hear the language "use a sampling frequency of 2" to define the time samples. A sampling frequency Fs = 2 means we want to have 2 samples per second. In sampling period language, this means we want the period Ts between samples to be Ts = 1 / Fs = 0.5. Hence, we can also define the time samples as such instead:

- Define sampling frequency: >> Fs = 2;
- Define time samples vector **t**: >> t = 0:1/Fs:2

e) ***Accessing Value(s) of Signal at Defined Time(s)***. In future labs, we will be interested in accessing or extracting values of a signal at particular time(s). To do this, we need to understand the relationship between time and Matlab indices.

i) We can find this relationship by considering the example where we define a time samples vector starting at t = 0 with a sampling frequency of 10. Again, Fs = 10 means we want the period between samples to be 1 / Fs = 0.1.

Index, i	1	2	3	4
Time, t	0.0	0.1	0.2	0.3

Using the sampling frequency Fs = 10, we can define a relationship between each index and time sample as follows:

Index, i	1	2	3	4
Time, t	0.0	0.1	0.2	0.3
Finding i from t	1 = 0.0*(10) + 1	2 = 0.1(10) + 1	3 = 0.2(10) + 1	4 = 0.3(10) + 1

As you can see, the general formula to determine the index is: $\boxed{i = t \times F_S + 1}$

ii) Matlab index **i** must be an integer. It is possible for **t x Fs** to be a floating-point value, depending on your selection for **t** and **Fs**. We will learn how to handle this special case in a future lab.

iii) Let us now consider the following example. Define a signal **x(t) = -3t** over the range **0 ≤ t ≤ 5** with a sampling frequency of 5 (which means a sampling period of 0.2). We will extract **x(3)** and then extract **x(t)** over the range **1 ≤ t ≤ 2**

- Define sampling frequency: >> Fs = 5;
- Define time samples vector: >> t = 0:1/Fs:5;
- Define function **x(t)**: >> x = -3 * t;
- Compute index for t = 3: >> t_index = 3 * Fs + 1;
- Extract **x(t)** at t = 3 and display: >> x(t_index)
- Extract from t = 1 to t = 2: >> t_start = 1 * Fs + 1;
  >> t_end = 2 * Fs + 1;
  >> x(t_start:t_end)

2) ***Lab Exercise***: Let us now practice what you have learned with a short exercise.

a) Open up a new script and save it as **Ex2.m**

b) Copy or type in the following for **Ex2.m**:

```
% FILE: Ex2.m
% NAME: [FILL IN NAME HERE]
% DESCRIPTION: More Matlab Variables and Basic Operations

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

% PART A

% PART B

% PART C

% PART D
```

c) For PART A: Write the lines of code to define the following two signals **x(t) = 0.5t** and **y(t) = t².** Both signals will be defined over the range **0 ≤ t ≤ 3** with a sampling frequency of 2.

d) For PART B: Write the line of code to define a new signal **z(t)**, where **z(t) = x(t) – 2y(t)**.

e) For PART C: Write the line of code to access **z(t)** at t = 2, and store the result in variable **w1**. Display the output on the COMMAND window.

f) For PART D: Write the lines of code to extract values of **z(t)** from **0 ≤ t ≤ 1.5**, and store the values in variable **w2**. Display the output on the COMMAND window.

g) Run and test your script.

3) ***Lab Check-Off***: Demonstrate your script for **Ex2.m** with your Lab TA**.**