The University of Calgary
Department of Electrical & Computer Engineering

ENEL 471 Introduction to Communications Systems and Networks
Self-Learning Tutorial – Familiarization with MATLAB

1. Objectives

The objectives of this tutorial are to:

- teach you the basics of using MATLAB (MATrix LABoratory) Tool
- become familiar with the MATLAB package by working through several examples and practice problems

2. Overview

MATLAB® is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation [1]. Typical uses include

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

In this lab you will learn the basics of the MATLAB software. You will learn how to write MATLAB script and function programs, and make plots. You will practice using the software by running through some exercises that serve as pre-work for the Labs and Assignments in ENEL 471.

At the end of this lab, you will be able to use the basic arithmetic operations in MATLAB and become familiar with the built-in functions in MATLAB. You will be able to plot graphs, and generate m-file script and function programs. In essence, after completing this Lab, you should have the necessary MATLAB background to do the assignments and Labs in ENEL 471.

3. Background Readings and Preparation

MATLAB's *Introduction* Tutorial (see References [1], [2] and [3] in Section 5.0)

4. MATLAB Tutorial

Step 0: Log in to your ENEL account.

Step 1: Starting MATLAB and MATLAB Windows.

Start MATLAB by either double-clicking the **MATLAB** icon if it's on the desktop *or* Left Click on **Start → Programs → MATLAB → R2017a → MATLABR2017a**.

The default MATLAB Desktop appears as shown below:

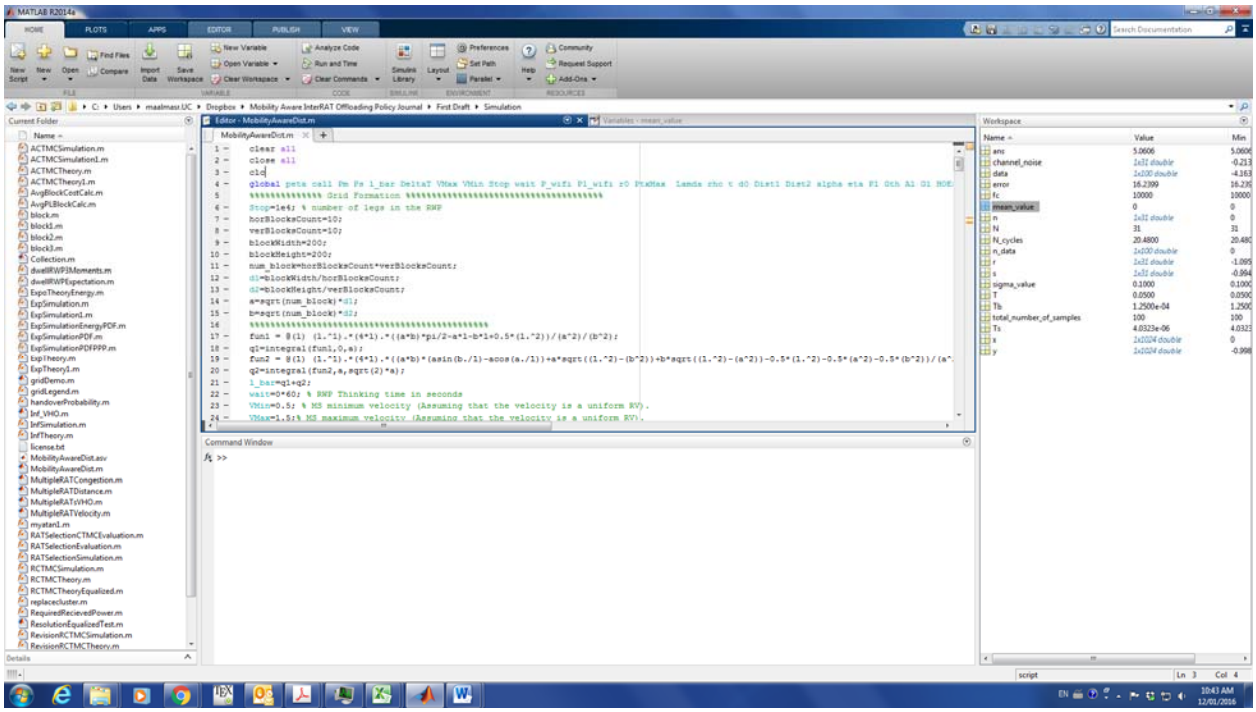

Figure 1 MATLAB layout.

The MATLAB Desktop consists of 10 components, four of which are only visible on the desktop shown above. The main functionality of each component is given as follows:

i) **Command Window**: Main interface used to execute MATLAB commands and run MATLAB scripts.

ii) **Workspace** Browser: Graphical User Interface (GUI) to the Workspace, which contains user-generated variables. This GUI can be used to view and make changes to these variables.

iii) **Variable Editor**: GUI interface that displays variable contents in table format and permits editing of variable values.

iv) **Command History** Browser: GUI interface that can be used to view and execute the previously executed commands.
v) **Current Directory** Browser: GUI interface that can be used to browse directories, selectively display directory contents, and open the files in these directories.
vi) **Help** Browser: Main GUI interface to the extensive help documents available in MATLAB.
vii) **Start Button:** GUI interface that permits one-stop access to all of the GUI tools and more help documentation, including various demos.
viii) **Editor:** GUI interface used to create, edit, and debug user-defined MATLAB command programs, called m-files.
ix) **GUIDE:** GUI Layout Editor used to generate user-defined GUI's.
x) **Profiler:** GUI interface used to assess the performance of m-file programs.

The default MATLAB desktop also has 6 pull-down menus. The menus **File**, **Edit**, **Window**, and **Help** are standard in most Microsoft Windows products. The additional **Debug** and **Desktop** menus are used to debug MATLAB programs and manipulate the Desktop, respectively.

Step 2: Working in the Command Window

At the >> prompt of the Command Window, you are now ready to start entering commands at the prompt or issue commands for opening the other windows or exit from MATLAB. For example, to exit MATLAB, at the prompt type **exit** (or **quit**) and press the **Enter** key.

Step 3: Performing Basic Arithmetic Operations

In MATLAB, the basic arithmetic operations and their respective symbols are "addition" (+), "subtraction" (-), "multiplication" (*), "division" (/), and "exponentiation" (^). The order of Precedence for the MATLAB basic operations is as follows: ^ has the highest priority, followed by * and /, each having the same priority. The lowest priority operations are + and -. In each of the three priority groups, MATLAB operates left to right. Of course, a lower priority operation, if enclosed in parentheses, can be done before a higher priority operation. For nested parentheses, the innermost are executed first.

→ Enter the following MATLAB statements at the flashing vertical line cursor just after the >> prompt and then press the keyboard key **Enter** to execute the command. (If the cursor is not flashing, click the mouse in the Command Window).

3.1 Addition:

Example 3.1.1 At the prompt type y = 1+2+3 and press **Enter** key. The output y = 6 is displayed.

```
>> y = 1+2+3 Enter
>> y =6
```

Note: The letter "y" is a MATLAB variable (user-selected) that is used to store the result of the addition operation. In MATLAB, there are 3 main rules for naming variables: 1) variable names must begin with a letter and they can contain any combinations of letters, numbers, and underscores; 2) MATLAB variable letters are case-sensitive, i.e., m_1 and M_1 are different variables; and 3) a variable name can be of any length, but MATLAB uses only the first 63 characters of the name.

Example 3.1.2 At the prompt type 1+2+3 and press **Enter** key. The output ans = 6 is displayed.

>> 1+2+3 **Enter**
>> ans =6

Note: If no assignment variable is specified in a statement, MATLAB automatically creates the default variable "**ans**" to store the results.

Example 3.1.3 At the prompt type y_Ex3p1p3 = 1+2+3; and press **Enter** key. The output is NOT displayed.

>> y_Ex3p1p3 = 1+2+3; **Enter**
>>

Note: If a statement is followed by a semi-colon (;), MATLAB performs the calculation but suppresses the output. To see the result of the calculation, you can type the variable name that stores the result followed by the **Enter** key, as shown in Example 3.1.4.

Example 3.1.4 At the prompt type y_Ex3p1p3 and press **Enter** key. The output is displayed.

>> y_Ex3p1p3 **Enter**
>> y_Ex3p1p3 =6

3.2 Subtraction

Example 3.2.1 At the prompt type y = 15-5-3 and press **Enter** key. The output y = 7 is displayed.

>> y = 15-5-3 **Enter**
>> y =7

Note: MATLAB does the calculation from left to right

Example 3.2.2 At the prompt type y = 15-(5-3) and press **Enter** key. The output y = 13 is displayed.

>> y = 15-(5-3) **Enter**
>> y =13

Note: Parentheses are used to control the order of operation.

3.3 Multiplication

Example 3.3.1 At the prompt type y = 15*5*3 and press **Enter** key. The output y = 225 is displayed.

>> y = 15*5*3 **Enter**
>> y =225

3.4 Division

Example 3.4.1 At the prompt type y = 15/5/3 and press **Enter** key. The output y = 1 is displayed.

>> y = 15/5/3 **Enter**
>> y =1

Note: MATLAB does the calculation from left to right.

Example 3.4.2 At the prompt type y = 15/(5/3) and press **Enter** key. The output y = 9 is displayed.

>> y = 15/(5/3) **Enter**
>> y =9

Note: Parentheses are used to control the order of operation.

3.5 Exponentiation

Example 3.5.1 At the prompt type y = 4^3^2 and press **Enter** key. The output y = 4096 is displayed.

>> y = 4^3^2 **Enter**

>> y =4096

Note: MATLAB does the calculation from left to right.

Example 3.5.2 At the prompt type y = 4^(3^2) and press **Enter** key. The output y = 262144 is displayed.

>> y = 4^(3^2) **Enter**

>> y =262144

Note: Parentheses are used to control the order of operation.

3.6 Mixed Operations

Example 3.6.1 At the prompt type y = 20/4^2*3+5-6 and press **Enter** key. The output y = 2.7500 is displayed.

>> y = 20/4^2*3+5-6 **Enter**

>> y =2.7500

Note 1: MATLAB does the calculation in the following order: 4^2 first (=16); then 20/16 (=1.25); then 1.25*3 (=3.75); then 3.75+5 (=8.75), then 8.75-6 (=2.75).

Note 2: MATLAB displays output number on the screen using the default 4 decimal digits (called **short** format). MATLAB has several other formats for displaying numbers. Details of these formats can be found by typing **help format** and pressing the **Enter** key in the Command Window. Note that the format in which numbers are displayed does not affect how MATLAB computes and saves numbers.

Example 3.6.2 At the prompt type y = 20/4^(2*3)+(5-6) and press **Enter** key. The output y = -0.0051 is displayed.

>> y = 20/4^(2*3)+(5-6) **Enter**

>> y =-0.9951

Note 1: MATLAB does the calculation in the following order: (2*3) first (=6); then (5-6) (= -1); then 4^6 (=4096); then 20/4096 (=0.0049), then 0.0049+(-1) (= -0.9951).

Step 4: Familiarity with MATLAB built-in mathematical functions.

MATLAB has several built-in mathematical functions. A function has a name and an argument in parentheses with syntax *func_name*(*arg*). The "*func_name*" is a standard MATLAB name of the function and "*arg*" is the argument which can be a number, a variable that has been assigned a numerical value prior to its use in the function, or a computable expression that can be made up of numbers and/or variables.

4.1 Built-in Elementary Mathematical Functions: Suppose x is a defined argument, built-in elementary mathematical functions in MATLAB include: square root of x (sqrt(x)), exponential of x (exp(x)), absolute value of x (abs(x)), natural (i.e., base e) logarithm of x (log(x), base 10 logarithm of x (log10(x)), and factorial of x (factorial(x)). *Note:* To use the factorial function, x must be non-negative.

Example 4.1.1 At the prompt type sqrt(45) and press **Enter** key. The output ans = 6.7082 is displayed.

>> sqrt(45) **Enter**
>> ans =6.7082

Example 4.1.2 At the prompt type exp(4) and press **Enter** key. The output ans = 54.5982 is displayed.

>> exp(4) **Enter**
>> ans =54.5982

Example 4.1.3 At the prompt type abs(-45) and press **Enter** key. The output ans = 45 is displayed.

>> abs(-45) **Enter**
>> ans =45

Example 4.1.4 At the prompt type log(245) and press **Enter** key. The output ans = 5.5013 is displayed.

>> log(245) **Enter**
>> ans =5.5013

Example 4.1.5 At the prompt type log10(245) and press **Enter** key. The output ans = 2.3892 is displayed.

>> log10(245) **Enter**
>> ans =2.3892

Example 4.1.6 At the prompt type factorial(5) and press **Enter** key. The output ans = 120 is displayed.

>> factorial(5) **Enter**
>> ans =120

4.2 Built-in Trigonometric Mathematical Functions: Suppose angle θ is a defined argument **expressed in radians**, built-in trigonometric mathematical functions in MATLAB include: sine of θ (sin(θ)), cosine of θ (cos(θ)), tangent of θ (tan(θ)), cotangent of θ (cotan(θ)).

Note 1: Trigonometric functions expect radian angles. As such, angle θ given in degrees (i.e., θ^o) must first be converted to angle θ in radians: $\theta = \theta^o$*pi/180.

Note 2: In MATLAB, the constant π is represented as **pi**. At the command prompt, type pi and press the Enter key, an output ans = 3.1416 (using the default output format) is displayed.

Note 3: The inverse trigonometric functions of x in MATLAB are asin(x), acos(x), atan(x) and acot(x) for arcsin(x), arccos(x), arctan(x) and arccot(x), respectively. The output of each inverse trig function is angle θ in radians. To convert to angle θ in degrees, use: $\theta^o = \theta$*180/pi.

Example 4.2.1 Calculate the sine of 30^o. At the prompt type sin(30*pi/180) and press **Enter** key. The output ans = 0.5000 is displayed.

>> sin(30*pi/180) **Enter**
>> ans =0.5000

Example 4.2.2 Calculate the cosine of 30^o. At the prompt type cos(30*pi/180) and press **Enter** key. The output ans = 0.8660 is displayed.

>> cos(30*pi/180) **Enter**
>> ans =0.8660

Example 4.2.3 Calculate the tangent of 30^o. At the prompt type tan(30*pi/180) and press **Enter** key. The output ans = 0.5774 is displayed.

>> tan(30*pi/180) **Enter**
>> ans =0.5774

Example 4.2.4 Calculate the cotangent of 30^o. At the prompt type cot(30*pi/180) and press **Enter** key. The output ans = 1.7321 is displayed.

>> cot(30*pi/180) **Enter**
>> ans =1.7321

Example 4.2.5 Calculate the inverse sine of 0.5 and express the output angle in degrees. At the prompt type asin(0.5)*180/pi and press **Enter** key. The output ans = 30.0000 is displayed.

>> asin(0.5)*180/pi **Enter**
>> ans =30.0000

Step 5: Creation of vectors (one-dimensional arrays).

5.1 Creating a vector from a known list of numbers: The vector is created by typing the elements (numbers) inside square brackets [ ]. The syntax is: vector_name = [type vector elements].

→ To create a row vector, type the left square bracket [and then type the elements with a space or comma between the elements. Type the right square bracket] after the last element and press the **Enter** key.

→ To create a column vector, type the left square bracket [and then enter the elements with a semicolon between them (or press the **Enter** key after each element). Type the right square bracket] after the last element and press the **Enter** key.

Example 5.1.1 Generate row vector A having elements 2, 4, 6, 8 and 10. At the prompt type A = [2 4 6 8 10] or A = [2, 4, 6, 8, 10] and press **Enter** key. The output A = 2 4 6 8 10 is displayed.

>> A = [2 4 6 8 10] **Enter**
>> A =2 4 6 8 10

Example 5.1.2 Generate column vector B having elements 1, 3, 5, 7 and 9. Method 1: At the prompt type B = [1 **Enter** 3 **Enter** 5 **Enter** 7 **Enter** 9] and press **Enter** key. The output B =

1
3
5
7
9

is displayed.

Method 2: At the prompt type B = [1; 3; 5; 7; 9] and press **Enter** key

```
>> B = [1; 3; 5; 7; 9] Enter
>> B =
        1
        3
        5
        7
        9
```

5.2 Creating a vector with constant spacing by specifying the first term, the spacing, and the last term, separated by colons. The syntax is: vector_name = [first_term : spacing : last_term] or vector_name = first_term : spacing : last_term.

→ The three terms may or may not be enclosed by square brackets.

→ If the spacing is omitted (i.e., only the first and last terms specified), the default spacing is 1.

Example 5.2.1 Generate a vector A having first term of 1, spacing of 2 and last term of 14. At the prompt type A = [1 : 2 : 14] or A = 1 : 2 : 14 and press **Enter** key. The output A = 1 3 5 7 9 11 13 is displayed.

```
>> A = [1 : 2 : 14] Enter
>> A =
        1 3 5 7 9 11 13
```

Example 5.2.2 Generate a vector B having first term of -3, spacing of 1 and last term of 3. At the prompt type B = [-3 : 3] or B = -3 : 3 and press **Enter** key. The output B = -3 -2 -1 0 1 2 3 is displayed.

```
>> B = -3 : 3 Enter
>> B =-3 -2 -1 0 1 2 3
```

5.3 Creating a vector with constant spacing by specifying the first and last terms, and the number of terms separated by commas. The syntax is: vector_name = **linspace**(first_term, last_term, number of elements).

→ The MATLAB built-in function **linspace**() is used.

→ If the number of elements is omitted, the default value of 100 is used.

Example 5.3.1 Generate a vector A having first term of 1, last term of 13 and 7 terms. At the prompt type A = linspace(1, 13, 7) and press **Enter** key. The output A = 1 3 5 7 9 11 13 (same as that of Ex. 5.2.1) is displayed.

```
>> A = linspace(1, 13, 7) Enter
>> A =
        1 3 5 7 9 11 13
```

5.4 Vector Addressing: The address of an element in a vector is its position in the row (or column). For a vector named A, A(k) refers to the element in position k, k = 1, 2, 3, ….

→ In MATLAB, the indices for vectors (and matrices) are NOT allowed to begin at zero. The first index position is 1. (For a time series beginning at time zero (i.e., t = 0), always use the index 1 for t = 0).

Example 5.4.1 Generate a vector A having first term of 1, last term of 13 and 7 terms. At the prompt type A = linspace(1, 13, 7); and press **Enter** key. Notice that the output is suppressed. At the prompt type A(1) **Enter** and the output ans = 1 is displayed. Verify A(2) to A(7). What is A(0)?

```
>> A = linspace(1, 13, 7); Enter
>> A(1) Enter
ans =
        1
```

→ A single vector element, e.g., A(k) can be used just as a variable. For example, it is possible to change the value of only one element of a vector by reassigning a new value to a specific address. A single element can also be used as a variable in a mathematical expression.

Example 5.4.2 For the vector A generated in Ex. 5.4.1, assign a new value of 8 to the element in position 2 and display vector A again. At the prompt type A(2) = 8; A and press **Enter** key. The output A = 1 8 5 7 9 11 13 is displayed.

```
>> A = linspace(1, 13, 7); Enter
>> A(2) = 8; A Enter
>> A =
        1 8 5 7 9 11 13
```

Note: Multiple statements can be written on the same line prior to pressing the Enter key. The multiple statements are separated either by comma (output displayed) or semicolon (output suppressed)

Example 5.4.3 For the vector A generated in Ex. 5.4.2, add elements A(6) and A(7) together and assign the result to variable y. At the prompt type y = A(6) + A(7) and press **Enter** key. The output y = 24 is displayed.

```
>> y = A(6) + A(7) Enter
>> y =
         24
```

5.5 Vector Operations: The following operations can be performed with vectors:

5.5.1 Addition, subtraction, multiplication, and division of a vector by a constant

Example 5.5.1.1 Generate vector A as in Ex. 5.4.1, and add constant 5 to A. Call the new vector Z1. At the prompt type Z1 = A + 5 and press **Enter** key. The output Z1 = 6 8 10 12 14 16 18 is displayed.

```
>> Z1 = A + 5 Enter
>> Z1 =
        6 8 10 12 14 16 18
```

Example 5.5.1.2 Generate vector A as in Ex. 5.4.1, and subtract constant 6 from A. Call the new vector Z2. At the prompt type Z2 = A - 6 and press **Enter** key. The output Z2 = -5 -3 -1 1 3 5 7 is displayed.

```
>> Z2 = A - 6 Enter
>> Z2 =
        -5 -3 -1 1 3 5 7
```

Example 5.5.1.3 Generate vector A as in Ex. 5.4.1, and multiply it by constant 2. Call the new vector Z3. At the prompt type Z3 = A * 2 and press **Enter** key. The output Z3 = 2 6 10 14 18 22 26 is displayed.

```
>> Z3 = A * 2 Enter
>> Z3 =
        2 6 10 14 18 22 26
```

Example 5.5.1.4 Generate vector A as in Ex. 5.4.1, and divide it by constant 2. Call the new vector Z4. At the prompt type Z4 = A / 2 and press **Enter** key. The output Z4 = 0.5000 1.5000 2.5000 3.5000 4.5000 5.5000 6.5000 is displayed.

```
>> Z4 = A / 2 Enter
>> Z4 =
        0.5000 1.5000 2.5000 3.5000 4.5000 5.5000 6.5000
```

5.5.2 Addition (or subtraction) of vectors. Two vectors <u>of the same size</u> can be added (or subtracted).

Example 5.5.2.1 Given X = [1 2] and Y = [3 4], find Z = X+Y. At the prompt type X = [1 2]; Y = [3, 4]; and press **Enter** key. Type Z = X+Y and press **Enter** key. The output Z = 4 6 is displayed.

```
>> X = [1 2]; Y = [3, 4]; Enter
>> Z = X+Y Enter
>> Z =
        4 6
```

Example 5.5.2.2 Given Z = [10, 5] and Y = [6 8], find X = Z-Y. At the prompt type Z = [10 5]; Y = [6 8]; and press **Enter** key. Type X = Z-Y and press **Enter** key. The output X = 4 -3 is displayed.

```
>> Z = [10, 5]; Y = [6 8]; Enter
>> X = Z-Y Enter
>> X =
        4  -3
```

5.5.3 Element by element operations on vectors: The element-by-element operators are used on 2 vectors having the same size. The operators include element-by-element multiplication (.*), element-by-element division (./), element-by-element exponentiation (.^), and element-by- element exponentiation of a vector by a constant.

5.5.3.1 Element-by-element Multiplication: Given 2 vectors X and Y both having the same size, the array operator ".*" multiplies the two vectors element by element.

Example 5.5.3.1 Given X = [1 2] and Y = [3 4], find Z = X.*Y. At the prompt type X = [1 2]; Y= [3, 4]; and press **Enter** key. Type Z = X.*Y and press **Enter** key. The output Z = 3 8 (i.e., [1*3 2*4] is displayed.

```
>> X = [1 2]; Y = [3, 4]; Enter
>> Z = X.*Y Enter
>> Z =
        3  8
```

Note: The operation Z = X * Y cannot be done. (I.e., 2 vectors X and Y, though having the same size, cannot be multiplied together because the number of columns in X is not equal to the number of rows in Y, according to linear algebra rules)

5.5.3.2 Element-by-element Division: Given 2 vectors X and Y both having the same size, the array operator "./" divides the two vectors element by element.
Example 5.5.3.2 Given X = [1 2] and Y = [3 4], find Z = X./Y. At the prompt type X = [1 2]; Y = [3, 4]; and press **Enter** key. Type Z = X./Y and press **Enter** key. The output Z = 0.3333 0.2500 (i.e., [1/3 2/4] is displayed.

```
>> X = [1 2]; Y = [3, 4]; Enter
>> Z = X./Y Enter
>> Z =
        0.3333  0.2500
```

5.5.3.3 Element-by-element Exponentiation of one vector by another: Given 2 vectors X and Y both having the same size, the array operator ".^" raises each element of X by a corresponding element of Y.
Example 5.5.3.3 Given X = [1 2] and Y = [3 4], find Z = X.^Y. At the prompt type X = [1 2]; Y= [3, 4]; and press **Enter** key. Type Z = X.^Y and press **Enter** key. The output Z = 1 16 (i.e., $[(1)^3$ $(2)^4]$ is displayed.

```
>> X = [1 2]; Y = [3, 4]; Enter
>> Z = X.^Y Enter
>> Z =
        1  16
```

Note: The operation Z = X ^ Y cannot be done. (I.e., 1 vector cannot be raised to the power of another vector!)

5.5.3.4 Exponentiation of a vector by a scalar: Given a vector X and a scalar, the array operator ".^" raises each element of A by the given scalar.
Example 5.5.3.4 Given X = [2 3] and a scalar = 2, find Z = X.^2. At the prompt type X = [2 3]; and press **Enter** key. Type Z = X.^2 and press **Enter** key. The output Z = 4 9 (i.e., $[(2)^2$ $(3)^2]$ is displayed.

```
>> X = [2  3]; Enter
>> Z = X.^2 Enter
>> Z =
        4  9
```

5.5.4 Using Vectors (Arrays) in MATLAB Built-In Mathematical Functions: MATLAB has many built-in functions for analyzing vectors (arrays). A defined vector (i.e., input vector) is passed into a built-in function and the function operates on each element of the vector (i.e., element-by-element application of the function). The result (output) from such an operation is an output vector in which each element is the result of applying the built-in function to the corresponding element of the input vector.

Example 5.5.4 Given X = [0 : π/4 : π], find Y = cos(X). At the prompt type X = [0 : pi/4 : pi] and press **Enter** key. The output X = 0 0.7854 1.5708 2.3562 3.1416 is displayed (i.e., the elements of vector X). Type Y = cos(X) and press **Enter** key. The output Y = 1.0000 0.7071 0.0000 -0.7071 -1.0000 is displayed.

```
>> X = [0 : pi/4 : pi] Enter
>> X =
        0  0.7854  1.5708  2.3562  3.1416
>> Y = cos(X) Enter
>> Y =
        1.0000  0.7071  0.0000  -0.7071  -1.0000
```

Note 1: The feature of MATLAB, in which vectors (arrays) can be used as arguments in functions, is called vectorization. *Make every effort to use vectorization while doing calculations in MATLAB.* For example, to generate 10,000 samples of a sine wave sine($2\pi t/10$), where t = 1, 2, …. 10,000:

Inefficient Method:

```
for t = 1:10000        % looping over t (a scalar) from 1 to 10000
    y(t) = sin(2 * pi * t / 10);  % element-by-element calculation of vector y end
end
```

Efficient Method:

```
t = 1 : 10000          % define t as a vector having an initial value of 1 and
                       % final value of 10000 with spacing of 1 (see Step 5.2)
y = sin(2 * pi * t / 10);  % pass vector t as an argument to the sine function
                           %the output is vector y
```

Note 2: Some common MATLAB built-in functions for analyzing vectors (arrays) are given in Table 1. It is assumed that X is a vector.

Table 1. MATLAB built-in vector functions

Function Name	Description	Example
length(X)	Returns the number of elements in a vector X	>> X = [1 2 3 4 5]; **Enter** >> Y = length(X) **Enter** >> Y = 5
max(X)	Returns the largest element in X	>> X = [2 1 3 5 4]; **Enter** >> Y = max(X) **Enter** >> Y = 5
[maxval, psn] = max(X)	Returns the largest value (maxval) and the position (psn) of the largest value in the vector	>> [maxval, psn] = max(X) **Enter** maxval = 5 psn = 4
min(X)	Returns the smallest element in X	>> X = [2 1 3 5 4]; **Enter** >> Y = min(X) **Enter** >> Y = 1
[minval, psn] = min(X)	Returns the smallest value (minval) and the position (psn) of the largest value in the vector	>> [minval, psn] = min(X) **Enter** minval = 1 psn = 2

sum(X)	Returns the sum of the elements of vector X	>> X = [2 1 3 5 4]; **Enter** >> Y = sum(X) **Enter** >> Y = 15
sort(X)	Arranges the elements of vector X in ascending order	>> X = [2 1 3 5 4]; **Enter** >> Y = sort(X) **Enter** >> Y = 1 2 3 4 5
mean(X)	Returns the mean value of the elements of vector X	>> X = [2 1 3 5 4]; **Enter** >> Y = mean(X) **Enter** >> Y = 3
median(X)	Returns the median value of the elements of vector X	>> X = [2 1 3 5 4]; **Enter** >> Y = median(X) **Enter** >> Y = 3
var(X)	Returns the variance of the elements of vector X	>> X = [2 1 3 5 4]; **Enter** >> Y = var(X) **Enter** >> Y = 2.5000
std(X)	Returns the standard deviation of the elements of vector X	>> X = [2 1 3 5 4]; **Enter** >> Y = std(X) **Enter** >> Y = 1.5811

5.5.5 Transpose of a Vector: The transpose operator, denoted by "'", when applied to a vector, switches a row (column) vector to a column (row) vector.

Example 5.5.5 Given X = [2 3] (row vector) find the transpose of X, X^T. At the prompt type X = [2 3]; and press **Enter** key. Type Z = X.' and press **Enter** key. The output Z = 2 3 is displayed.

```
>> X = [2  3]; Enter
>> Z = X.' Enter
>> Z =
        2
        3
```

5.5.6 Scalar (or *dot*) product of 2 vectors: Two vectors can multiply each other only if both have the same length *n* (i.e., same number of elements) and one is a row vector (1 x *n*) and the other is a column vector (*n* x 1). The multiplication of a row vector by a column vector (both having the same length) gives a 1 x 1 matrix, which is a scalar. This is the dot product of two vectors.

Note: MATLAB also has a built-in function, named dot(X, Y), that computes the dot product of 2 vectors X and Y.

Example 5.5.6 Given X = [2 3] (row vector) and Y = [4 6]. Let Y = W^T. Find the dot product of vectors X and Y. At the prompt type X = [2 3]; Y = [4 6]; and press **Enter** key. Type Z = X*Y.' and press **Enter** key. The output Z = 26 is displayed. Verify the same output using the MATLAB built-in *dot()* function. Type dot(X, Y) and press **Enter** key. The output ans = 26 is displayed.

```
>> X = [2  3]; Y = [4 6]; Z = X*Y.' Enter
>> Z =
          26
>> dot(X,Y)
>> ans =
           26
```

5.6 Creating, Editing, Saving and Running an M-file Script Program: An alternative way of executing commands with MATLAB is first to create a file with a list of commands (known as script file, also called a program), save it and then run the file.

5.6.1. Creating a Script File: In MATLAB, script files are created and edited in the MATLAB Editor window, which can be opened in one of two ways:

→ At the prompt of the Command window type **edit** and press the **Enter** key, OR
→ Left click **File → New → M-file**.

5.6.2 Editing a Script File: Once the window is open, type the commands of the script file line by line. MATLAB automatically numbers a new line every time the Enter key is pressed. It is useful to begin each line of the program with several comment lines providing the filename, purpose of the file, author(s) name(s) and date of coding. In MATLAB, each comment line is started by the % character. An example of a short program written in the script file "Pre_Lab_Exercises.m" is shown in the Figure below:

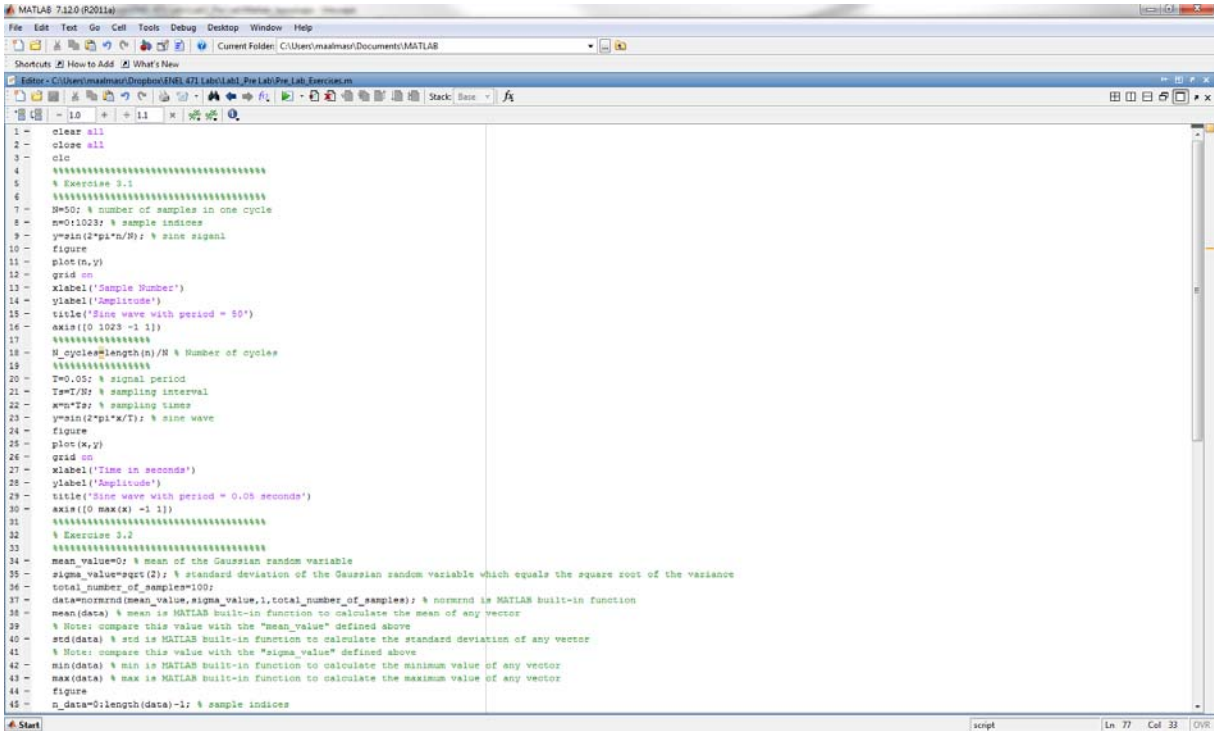

Figure 2 m-file example.

5.6.3 Saving a Script File: After entering the commands, the file must first be saved before it can be executed. To save a file, on the MATLAB editor window, left click on **File → Save As**… → then navigate to the folder where the file is to be saved, enter the file name in the **File name** field and then click the **Save** button. The rules for naming a script file are exactly the same as those for naming a variable (i.e., must begin with a letter, can include digits and underscore, and be up to 63 characters long. Note: The names of user-defined variables, predefined variables, MATLAB commands or functions should not be used to name script files. All the MATLAB script files end with the extension ".m").

5.6.4 Running a Script File:
Preambles: Before running a script file, first ensure that the **Current Directory** field of the command window contains the path to the folder in which the script file is stored. To do this:
→ click on the browser icon (this is the button represented by a box that contains 3 periods ("…") besides the pull down menu of the "Current Directory" field. A "Browse for Folder" box appears
→ navigate to the folder that contains the file to be executed and click the "OK" button.
Running the file: To run a script file, either
→ At the prompt, type the filename and press the **Enter** key, or
→ Open the file name by typing at the prompt: edit filename **Enter** and then left click on the **Run** icon

5.7 Creating, Editing, Saving and Using a "User-defined" Function Program: A user-defined function is a MATLAB program that is created by the user, saved as a function file, and then can be used like a built-in function. The main feature of a function file is that it has an input and an output. The input and the output can be one or several variables, and each can be a scalar, vector, or an array of any size.

5.7.1. Creating a Function File: In MATLAB, function files are created and edited in the MATLAB Editor window, which can be opened as described in Step 5.6.1.

5.7.2 Editing a Script File: Once the editor window is open, type the first line called the Function Definition Line (FDL). The FDL defines the file as a function file, defines the name of the function and specifies the number and order of the input and output arguments. The form of the function definition line is: function [output arguments] = function_name(input arguments)
Note 1: The word function, *typed in lower case letters*, must be the first word in the FDL
Note 2: List the output arguments inside *square* brackets. If there is more than one output argument, they are separated by commas. If there is only one output argument, the square brackets are not required. If there are no output arguments, both the square brackets and the equal sign must be omitted. *For the function file to work, the output arguments must be assigned values in the computer program that is in the function body*.
Note 3: Specify the name of the function after the equal sign. Select a meaningful name for the function that reflects the function's purpose. Adopt the same rules for naming variables and script files.
Note 4: List the input arguments inside *parentheses*. If there is more than one input argument, they are separated by commas. If there is only one input argument, the parentheses are still required. If there are no input arguments, the parentheses must be given.

For the function file to work, the input arguments must be assigned values in the calling program before the function is called.

After entering the FDL, enter the commands of the function file line by line. MATLAB automatically numbers a new line every time the Enter key is pressed. It is useful to begin each line of the program with several comment lines providing the filename, purpose of the function file, author(s) name(s) and date of coding. In MATLAB, each comment line is started by the % character.

Note 5: Write the computer code that performs the calculation within the function file in terms of the input arguments – same name(s) and syntax (scalars, vectors and arrays).

5.7.3 Saving a Function File: After entering the commands, the file must first be saved before it can be executed. To save a file, on the MATLAB editor window, left click on **File → Save As**…

→ then navigate to the folder where the file is to be saved, enter the file name in the **File name** field and then click the **Save** button. It is highly recommended that the file is saved with a name that is identical to the function name in the FDL. In this way the function is called (used) by using the function name. (If a function file is saved with a different name, the name it is saved under must be used when the function is called). A user-defined function filename must end with the extension ".m".

5.7.4 Using a Function File:

Preambles: Before using a function file, first ensure that the **Current Directory** field of the command window contains the path to the folder in which the function file is stored. To do this:

→ click on the browser icon (this is the button represented by a box that contains 3 periods ("…") besides the pull down menu of the "Current Directory" field. A "Browse for Folder" box appears

→ navigate to the folder that contains the function file to be called and click the "OK" button.

Calling the function file: To call a function file, either

→ At the Command Window prompt, type the function calling and press the Enter key. The function calling syntax is the FDL excluding the word "function". That is: [output arguments] = function_name(input arguments) OR

→ Include the function calling syntax in the script file or another function that can call the function.

Note: The names of the input and output arguments used when a function is called need not be the same as those used in the function file. However, they must have corresponding formats (scalar, vector or array). For example, if an input (output) argument is a scalar in the function file, a scalar variable must be used when the function is called.

5.8 Making Two-Dimensional Plots in MATLAB

5.8.1 Basic Linear Plot: Assume vectors Y and X are previously defined. The elements of Y can be plotted against the elements of X. At the command prompt, type the following:

```
>>X = [1:10]; Y = X.^2;
>> plot(X,Y) Enter
```

→ A MATLAB output showing a graph of Y versus X appears, as shown below:

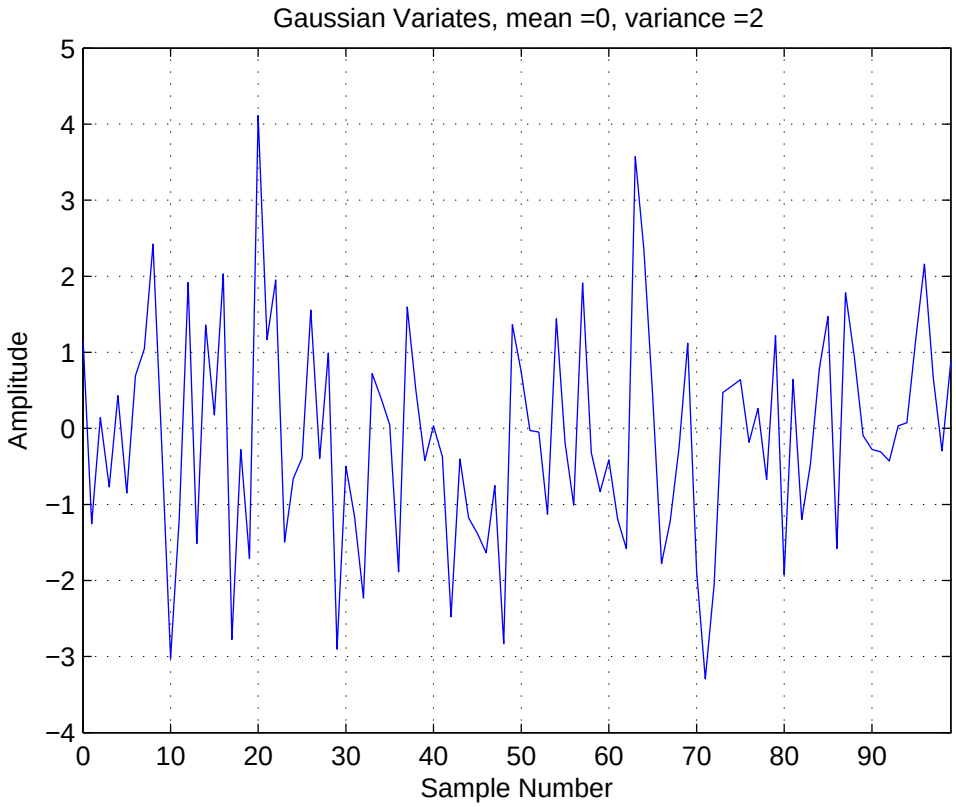

Figure 3 Plot output example.

Note 1: The number of elements of vector Y must be the same as that of vector X.
Note 2: Assume the vector pairs (X1, Y1), (X2, Y2), … are defined. To make multiple plots on the same figure using these vectors type

>> plot(X_1, Y_1, X_2, Y_2, …) **Enter**

→ plots Y_1 vs. X_1, Y_2 vs. X_2, etc., overlaid on the same plot. Each curve is plotted using a different color.

Note 3: The multiple plots can also be made using the **hold on** command. In this case, each vector pair is plotted separately, as illustrated below:

>>plot(X_1, Y_1) **Enter**
>>hold on **Enter**
>> plot(X_2, Y_2) **Enter**
>> hold off **Enter**

Note 4: For easy visual comparison of multiple graphs on the same plot, it is better to keep the range of X_1, X_2, … equal.

5.8.2 Linear Plot with Options: Three options can be specified: line style (L), marker type (M) and color (C). The three options must be enclosed in single quotes (i.e., the options are typed inside the plot command as strings) and can be used in any combination. The commands in step 5.8.1 with options included then become:

>> plot(X,Y, 'LMC') **Enter**
>> plot(X_1, Y_1, 'LMC' X_2, Y_2, 'LMC' …) **Enter**

Note 1: The options are specified for each curve.
Note 2: The available LMC options are listed in the Table shown below:

Line Style, L	Marker Type, M	Color, C
- = solid line (default)	+ = plus sign	r = red
-- = dashed line	o = circle	g = green
: = dotted line	* = asterik	b = blue
-. = dash-dot line	. = dot	c = cyan
	- = triangle (up)	m = magenta
	v = triangle (down)	y = yellow
	< = triangle (left)	k = black
	> = triangle (right)	w = white
	p = pentagram	
	h = hexagram	
	s = square	
	d = diamond	

5.8.3 semilogx, semilogy, loglog Plots: are logarithmic plots that are used when the data to be plotted varies over a wide range of values. They all have the same argument format as the plot command described in Steps 5.8.1 and 5.8.2.
→ semilogx(…) has a base 10 logarithmic scale for the x-axis.
→ semilogy(…) has a base 10 logarithmic scale for the y-axis.
→ loglog(…) has a base 10 logarithmic scale for both the x-axis and y-axis.
Note: Logarithmic plotting function cannot plot negative value (because logarithm of a negative number is undefined). Also, the logarithm function will not plot a 0 value, since the logarithm of 0 is $-\infty$.

5.8.4 Annotating Plots: MATLAB provides a variety of ways to format plots. Examples include adding title, x-axis and y-axis labels, legend and axis scaling. The Table below describes the general usage of commands for annotating plots in MATLAB.

Label Description	Command	Example
Plot- title placed at the top of the current figure	title('string')	title('Performance Plot')
x-axis label placed below the x-axis	xlabel('string')	xlabel('x-axis label')
y-axis label placed to the left of the y-axis	ylabel('string')	ylabel('y-axis label')
Legend placed on the figure. The legend labels are assigned in the same sequence as the plots were generated.	legend('string1', 'string2', …)	legend('Theory', 'Experiment', …)
Grid lines for a plot	grid on: adds grid lines to the plot grid off: removes grid lines from the plot	grid on grid off

Axis scaling changes the current minimum and maximum values to the specified limits, xmin and xmax (for the x-axis) and ymin and ymax (for the y-axis).	axis([xmin xmax ymin ymax])	axis([2.0 10.0 5.0 20.0])

5.8.5 Generation of Random Numbers: Many physical processes and engineering applications are random and can be characterized using a random number or a set of random numbers. MATLAB has two built-in functions **rand()** and **randn()** that can be used to generate random numbers.

5.8.5.1 The rand function: generates uniformly distributed numbers with values between 0 and 1. The Table below describes example usage of the rand function.

Command	Description	Example
rand	Generates a single random number between 0 and 1	>> val = rand **Enter** >> val = 0.5743
rand(1, n)	Generates an n-element row vector of random numbers between 0 and 1	>> Y = rand(1, 3) **Enter** >> Y = 0.5743 0.3124 0.7845
rand(n)	Generates an n x n matrix with random numbers between 0 and 1	>> B = rand(2) **Enter** >> B = 0.4309 0.8934 0.4310 0.1986
rand(m, n)	Generates an m x n matrix with random numbers between 0 and 1	>> X = rand(2, 3) >> X = 0.1459 0.3789 0.5706 0.4781 0.3330 0.9842 0.2478 0.5892 0.7644

Note 1: Scalar val, vector Y, and matrices B and X in the above Table assume an internal variable 'seed', which keeps changing as random numbers are generated. To generate a particular random sequence that can be reused in experiments, MATLAB allows the initialization of the seed value that must be set before calling the rand function. The syntax is as follows:

```
rand('seed', initial_value)   %initial_value is user specified, non-negative integer rand(1,n)
                              %generates a row vector of n elements uniform over 0 and 1
```

Note 2: Random numbers that are uniformly distributed in a range (a, b) can be obtained by multiplying rand by (b-a) and adding the product to a: (b-a)*rand + a

5.8.5.2 The randn function: generates normally distributed numbers with mean 0 and standard deviation of 1. The randn command can be used to generate a single number, a vector or a matrix in the same way as the rand command, as illustrated in the Table on the previous page.

Note 1: A particular random sequence whose elements are normally distributed can be generated by initializing the seed value.
Note 2: The mean and standard deviation can be changed. To generate a normally distributed random variable with mean m_val and standard deviation std_val, multiply the number generated by the randn function by the desired standard deviation and add the product to the desired mean: (std_val*randn) + m_val.

5.9 Comment Statement:
The % symbol is used to define a comment statement in MATLAB. All text on a line that follows a % symbol is interpreted as a comment statement.

5. References

[1] Introduction to MATLAB. Online documentation accessible from MATLAB Command window: Help menu → Product Help → Getting Started (under "Documentation Set" section)
[2] A. Gilat, MATLAB – An Introduction with Applications. John Wiley & Sons, Inc. 2005.
[3] B.L.F. Daku, MATLAB Tutor CD: Learning MATLAB Superfast. John Wiley & Sons, Inc.2006.