EGR 111
Vectors and Plotting Data

This lab introduces script files, 1D arrays, and plotting. This lab also describes the Current Folder, the comment symbol ("%"), and MATLAB file names.

New MATLAB Commands:

```
clear, plot, title, xlabel, ylabel, title, legend,
figure, comments (%)
```

1. MATLAB's Current Folder

Check to see if there is a folder in your P: drive called MATLAB. If not, create folder called MATLAB in your P: drive. We will use this folder to store all the files for this course.

Start MATLAB. Near the top of the MATLAB window, you will see the name of the Current Folder (see Figure 1 below). The Current Folder is the default folder that MATLAB uses when saving files and looking for files. For this course, you should always begin your MATLAB session by changing this folder to P:\MATLAB by clicking on the "Browse for Folder" button. If you don't set it to P:\MATLAB, then MATLAB won't be able to find the files in this folder. You can also check your Current Folder by typing the following MATLAB command in the command window: `pwd`

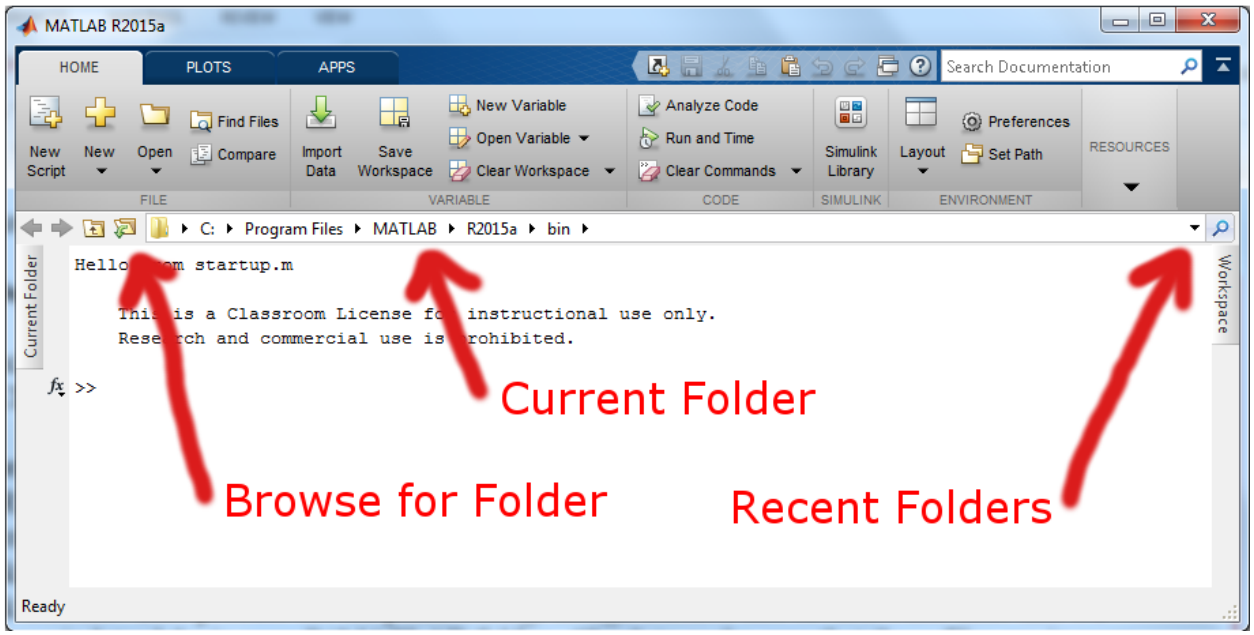


Figure 1: Current Folder

2. MATLAB Script Files

Typing commands into MATLAB's command window is convenient for performing short computations in MATLAB. However, in many cases you will want to save your commands so that you can re-use them without re-typing them. A MATLAB script file allows you to save a group of MATLAB commands in a file, to edit the commands, and to run the commands.

To open a new script file, click on File, New, Script.

Type the following commands into the editor window that opens up:

```
x = 1
y = 2
```

Click File, Save, select the folder P:\MATLAB, and give the file a name that adheres to the following rules:

- starts with a letter
- contains only letters, digits, and underscores (the "_" symbol)
- does not contain spaces or operators like -, +, *, /, !, etc.
- ends in ".m"

It is recommended to start the filename with a capitol letter in order to prevent accidentally using the name of a built-in MATLAB function.

Running the script file has the same effect as typing (or copying/pasting) the commands from the file into the command window. There are two ways to run the script file. The first way is to click on the green button in the editor window that says "Run" or "Save and Run" when the mouse hovers over it:

The second way to run the file is to save the file, and then in the command window, type the filename without the ".m". So if you have saved the file with the filename Scriptfile.m, then to run the file you would type: `Scriptfile`, or whatever name you have given to the file (but without the .m).

Common Error: If you get an error that says something like "??? Undefined function or variable 'Scriptfile'", then check to make sure that the folder where you saved the file is the same as the Current Folder which is shown at the top of the MATLAB window (see step 1 above). You can always change the Current Folder by using the Browse for Folder button near the top of the window.

3. Vectors

In MATLAB we can use the assignment operator to assign a value to a variable. For example:
```
a = 4
```

If a program needs a large number of related values, it is often very inconvenient to define a large number of variables to store the values. Fortunately, in addition to storing a single value, MATLAB variables can also store a whole list of values. Type the following command:
```
b = [2,4,6,8,10]
```

The square brackets tell MATLAB that you are defining a vector (also called a 1D array). The variable b now stores the values 2, 4, 6, 8, and 10. Each value stored in the 1D array is called an element. So, for example, the first element of b is 2, the second element is 4, etc.

If the values in square brackets are separated by commas , the values are stored in a row and the vector is called a row vector. Although MATLAB also allows you to separate the values in a row vector with spaces instead of commas, it is good practice to use commas (as shown above).

One of MATLAB's strengths is that it can easily store and process arrays of numbers, which can represent text strings, audio signals, and other sets of one-dimensional data.

4. Indexing

To access a particular element in a vector, type the variable name followed by the element number in parentheses. For example, assuming that you defined the vector b as shown above, you can access the third element of vector b by typing b(3), which is the value 6:
```
b(3)
```

MATLAB will respond with "`ans = 6`".

You can replace the value of a given element, without affecting the other elements, by using the variable name and element number in an assignment. To replace the current value of the third element with the value 100, type the following:
```
b(3) = 100
```

The command above overwrites the third element of the vector b with the new value of 100. Note that the old value (which was 6) is discarded and lost.

5. Colon Operator

MATLAB has a way to easily create vectors called the colon operator. First, let's experiment with creating vectors. Type the following:
```
x = 1:10
```
What is the first number? The increment between numbers? The last number?
```
x = 1:2:10
```

What is the first number? The increment between numbers? The last number?

```
x = 10:-1:1
```

What is the first number? The increment between numbers? The last number?

```
t = 0:0.1:1
```

What is the first number? The increment between numbers? The last number?

Note that the notation start:inc:end generates a vector of equally-spaced values from start to end incrementing by the value of inc. In some cases, such as 1:2:10, the last value is just before the value of end. If the increment is not given (start:end), MATLAB uses a default increment of 1.

The numeric operators have higher precedence than the colon operator, so the expression 1:2*3 will result in 2*3 being evaluated first, resulting in 1:6 or [1, 2, 3, 4, 5, 6].

If we want to use a generated vector in an expression, we can enclose the colon expression in square brackets. For example, type the following to yield 2 times the row vector [0:0.1:1]:

```
t = 2 * [0:0.1:1]
```

6. Linspace Function

The command `linspace` can also be used to generate vectors of equally-spaced values. It is more convenient than the colon operator when you would rather specify the number of values to generate instead of the increment between the values.

For example, to generate a vector that starts at 0, ends at 2, and has 10 elements, type the following: `x = linspace(0,2,10)`

If you only give `linspace` the start and end values, it will default to 100 elements.
So, `x = linspace(0,2)` would return a vector with 100 elements between 0 and 2.

7. Using the Colon Operator in Indexing

In order to access a single element in a vector, we type the variable name followed by the element number in parentheses as follows:

```
a = [10 20 30 40 50]
a(3) = 100
```

In addition to accessing a single element, MATLAB also allows several elements of a vector to be selected using the colon operator. For example, let's replace the first three elements of the vector b by the value 100 as follows:

```
b = [10 20 30 40 50]
b(1:3) = 100
```

In the command above, MATLAB first expands 1:3 to the vector [1 2 3], and then it accesses those elements and sets their value to 100 (without changing the remaining elements).

If we want to access several elements at the end of a vector, we can use the end statement:

```
c = [10 20 30 40 50]
c(3:end) = 100
```

In the command above, the statement "`end`" is replaced by the number of elements in the variable c. So `c(3:end)` is equivalent to `c(3:end)`, which is [30 40 50].

We can access every other element of a vector as follows:

```
d = [10 20 30 40 50]
d(1:2:end) = 100
```

In the command above, the statement "`1:2:end`" is replaced by "`1:2:5`", which is "`[1 3 5]`". So the command accesses the first, third, and fifth elements and sets them to 100.

The colon operator allows us to easily access any desired part of a vector.

8. How to Delete Elements

If you want to delete elements in a vector, you set them equal to the empty matrix, which is represented as square brackets with nothing between them. In the example below, the second element of the vector x is deleted:

```
x = [10 20 30 40 50]
x(2) = []
```

9. Column Vectors

A list of values can also be stored in a column vector by separating the elements by semicolons instead of commas (or spaces) as follows:

```
e = [10; 20; 30; 40; 50]
```

MATLAB prints the output:

```
e =
    10
    20
    30
    40
    50
```

In addition, a row vector can be converted to a column vector (or vice versa) using the transpose operator as follows:

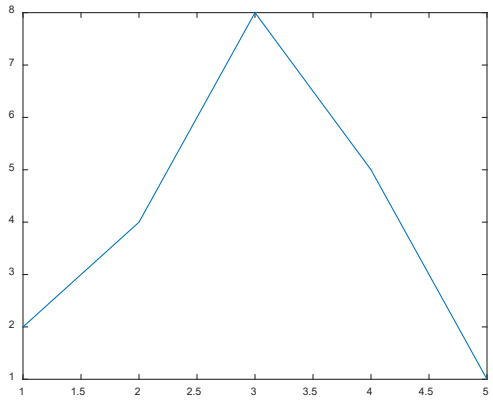
```
rowVector = [10, 20, 30, 40, 50]
colVector = rowVector'
```

Exercise 1: Write the commands that will generate the vector x1 = [0.1 0.2 0.3 … 1.0] and use indexing to replace the <u>even</u> indexed elements (those at positions 2, 4, 6, etc.) with the value 100.

10. Plotting

MATLAB makes it easy to plot data. For example, suppose we would like to plot the values 2, 4, 8, 5, and 1. Open a new script file and type the following commands.

```
clear
data = [2, 4, 8, 5, 1]
plot(data)
```



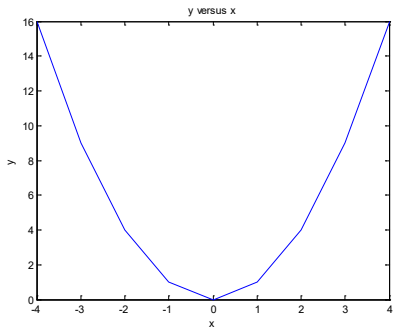
When you run the script file, a plot should appear in a new figure window.

It is good practice to put the `clear` command as the first command in a script file to clear the variables in the workspace. This practice prevents your script file from relying on variables that are currently in the workspace, but will not be there if you close MATLAB and restart it.

The command `plot(data)` makes a plot of the values in vector x. Note that it connects the values by straight lines. If you only give the plot command one vector, it uses the values 1, 2, 3, … for the x-values.

Next let's plot the function $y = x^2$. Open a new script file, type the following commands into the file, and run the file.

```
clear
x = -4:4
y = x.*x
plot(x,y)
xlabel('x')
ylabel('y')
title('y versus x')
```



The command "`x = -4:4`" computes the values of x where we want to plot a function using the colon operator. Recall that -4:4 expands to [-4 -3 -2 -1 0 1 2 3 4].

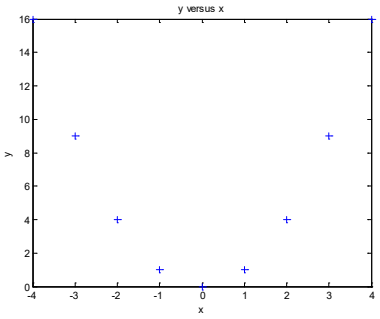
The command `y = x.*x` computes the square of each of those values. Why the period before the asterisk (.*) on the second line? Well, MATLAB, which is short for MATrix LABoratory, is set up to perform matrix operations by default, which is not what we want in this case. Here we want to multiply the first element of x by the first element of x, then multiply the second element of x by the second element of x, and so on. We call this operation "element-by-element multiplication" or "array multiplication". In order to tell MATLAB that we want element-by-element multiplication instead of matric multiplication, we put the period before the asterisk (.*). We also use the dot in front of other operations such as / and ^ to indicate that we want element-by-element operations: "`./`" and "`.^`". Note that if we do not include the dot, the command `y = x*x` would cause an error.

The third line plots the data in an x-y graph. The values from the first argument (x) is plotted on the horizontal scale, and the second argument (y) is plotted on the vertical scale. By default the data points are connected by straight lines. Also notice that the xlabel, ylabel and title commands cause strings to be placed on the x-axis, y-axis, and title. Always make sure to label the axes on your graphs!

If you wanted to use this graph in a report, you could click on Edit, Copy Figure, and then paste the graph into a word processor or other program.

MATLAB has lots of useful options for graphs. For example, if you don't want the data points to be connected by straight lines, you can have MATLAB mark the data points with plus signs (or other symbols) instead. Modify the `plot` command as follows and run the script file:

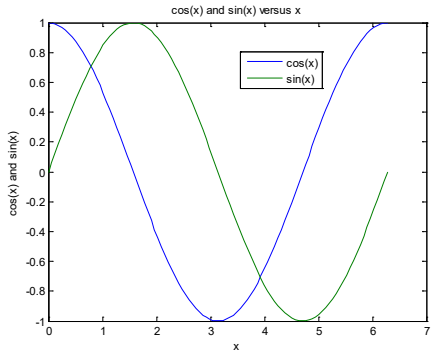
```
clear
x = -4:4
y = x.*x
plot(x,y,'+')
xlabel('x')
ylabel('y')
title('y versus x')
```



Notice that the new plot overwrites the old one.

Suppose we want to compare two functions, such as cosine and sine by placing both lines on the same graph. Open a new script file, type the following commands into the file, save it, and run it.

```
clear
figure(2)
x = linspace(0,2*pi);
y = cos(x);
z = sin(x);
plot(x,y,x,z)
xlabel('x')
ylabel('cos(x) and sin(x)')
title('cos(x) and sin(x) versus x')
legend('cos(x)','sin(x)')
```



The `figure(2)` command opens a second figure window and causes the following commands to operate in that window. The linspace command generates a vector from 0 to 2*pi with a 100 evenly-spaced points. The value of the variable `pi` is pre-defined to 3.141592653589793 when you start up MATLAB. The `plot(x,y,x,z)` command specifies the x and y values for two lines, so both are plotted on the same plot. You can add an arbitrary number of lines to a plot by specifying more pairs of data.

The `legend` command in used with plots with more than one line so we can tell which line is which. We give the `legend` command a string that describes each of the lines. Also, if you do not like where MATLAB has placed the legend in the figure window, you can click on the legend and drag it to a new location.

We can also specify the line color and line type (solid, dashed, etc.). To see all of the options, type: `help plot`

Exercise 2: Write a script file that plots the function $y = x^3$ from x = -10 to x = 10. Label the axes and title the plot. Please name the file VectorsEx2.m.

Exercise 3: Write a script file that plots the function $y1 = x^3$ and the function $y2 = 10*x^2$ from x = -10 to x = 10 on the same graph. Add a legend, label the axes, and title the plot. Please name the file VectorsEx3.m.

Checkpoint 1: Show your instructor your script files and the plots from Exercise 2 and 3.