

Christopher Lum  
lum@u.washington.edu

## **Beginner's Matlab Tutorial**

### **Introduction**

This document is designed to act as a tutorial for an individual who has had no prior experience with Matlab. For any questions or concerns, please contact

Christopher Lum  
lum@u.washington.edu

### **Starting the Program**

1. Start Matlab. After the program starts, you should see something similar to that shown in Figure 1 (the actual display may vary depending on the version of Matlab you have installed).

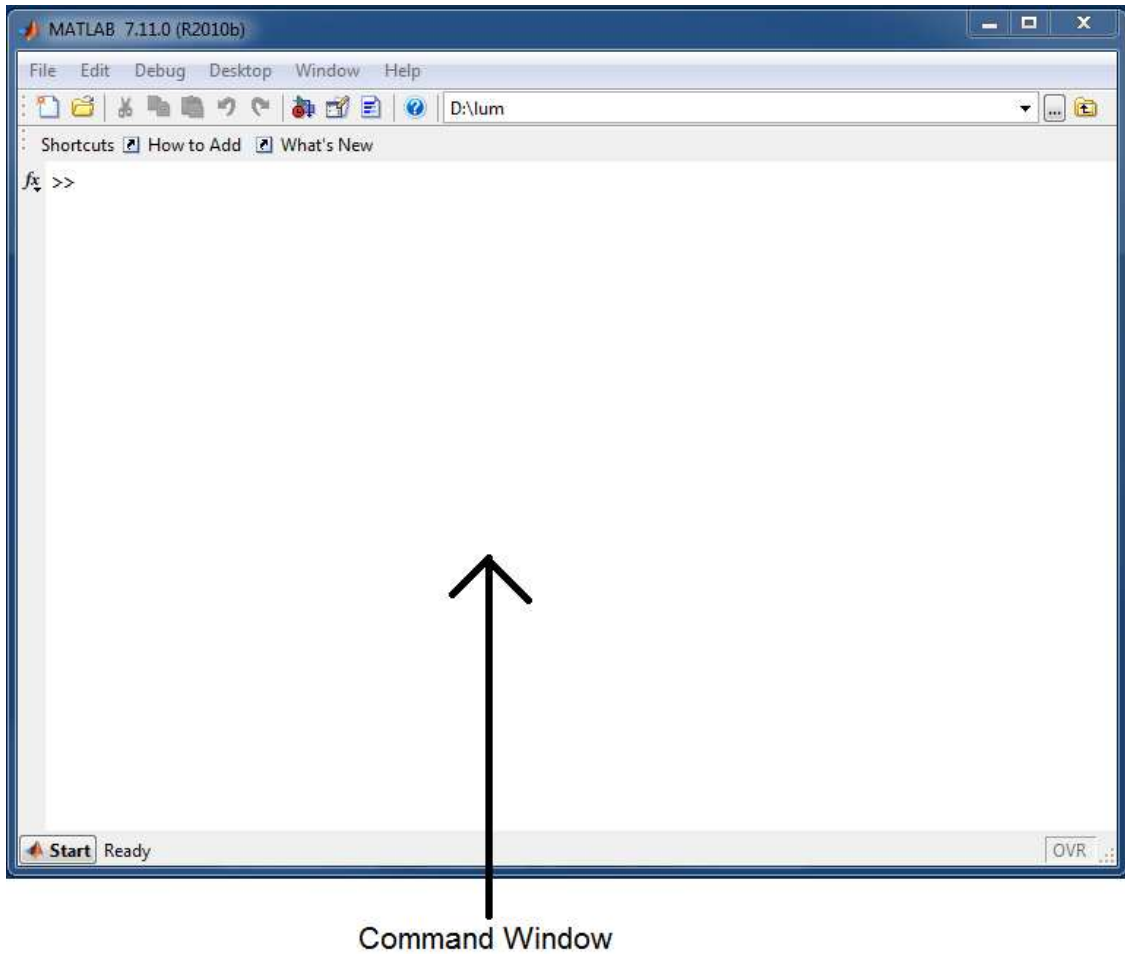


Figure 1: Basic Matlab interface showing only Command Window

2. If your window does not appear like this, it is possible that different windows are currently activated. Let us change the appearance and activate some useful windows. First, we'll start a new .m file. To do this use

File > New > Script

Or

File > New > M-File

3. This starts a new M-file which can be edited (more on this later). This probably opens the editor in a new window as shown below in Figure 2.

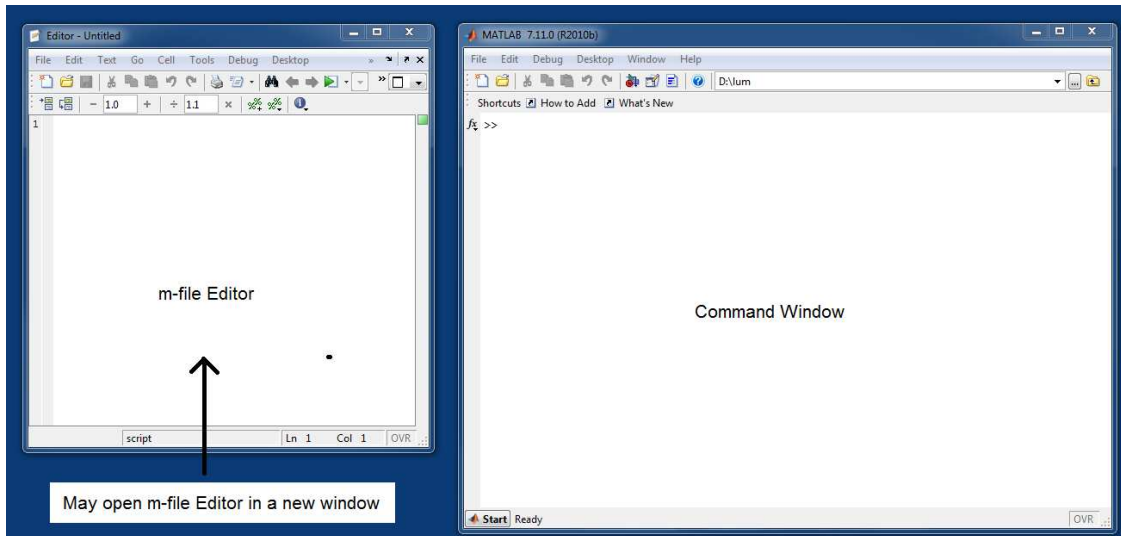


Figure 2: Screenshot of new m-file editor in new window

4. We would like to be able to see both the editor and the Command Window at the same time. Go back to the m-file editor and select

Desktop > Dock Editor

This will attach the m-file editor to the Command Window

5. We would also like to activate the Workspace window. To do this, go to the Command window and select

Desktop > Workspace

This will activate the Workspace window.

6. You can now drag around the 3 activated windows (Command Window, m-file editor, and Workspace) to arrange the views as you like. To drag a window, simply click on the window and then drag the blue bar (see Figure 3). The Matlab interface should now similar to Figure 3.

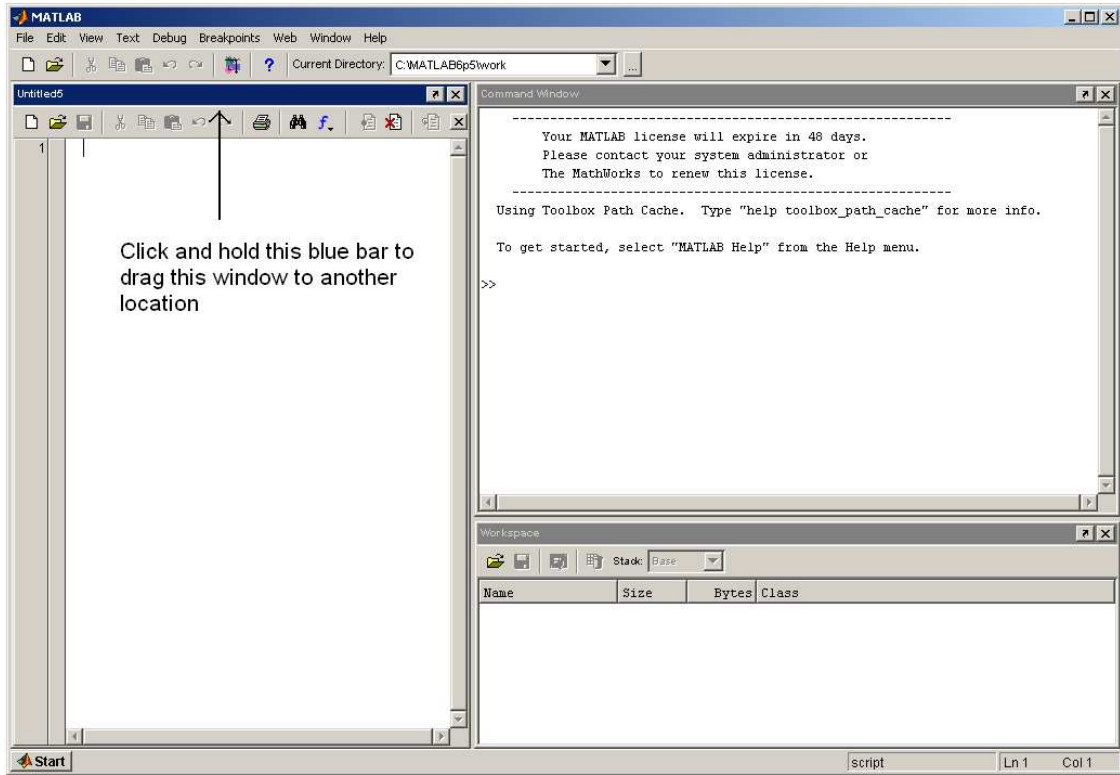


Figure 3: MATLAB interface

## Using Matlab

1. Matlab stores most of its numerical results as matrices. Unlike some languages (C, C++, C#), it dynamically allocates memory to store variables. Therefore, it is not necessary to declare variables before using them. Let's begin by simply adding two numbers. Click in the Command Window. You will see a flashing “|” symbols next to the “>>” symbol. Enter the following commands

1. Type in “x = 3” then hit “enter”
2. Type in “y = 2;” then hit “enter” (note the semicolon here!)
3. Type in “z = x + y” then hit “enter”

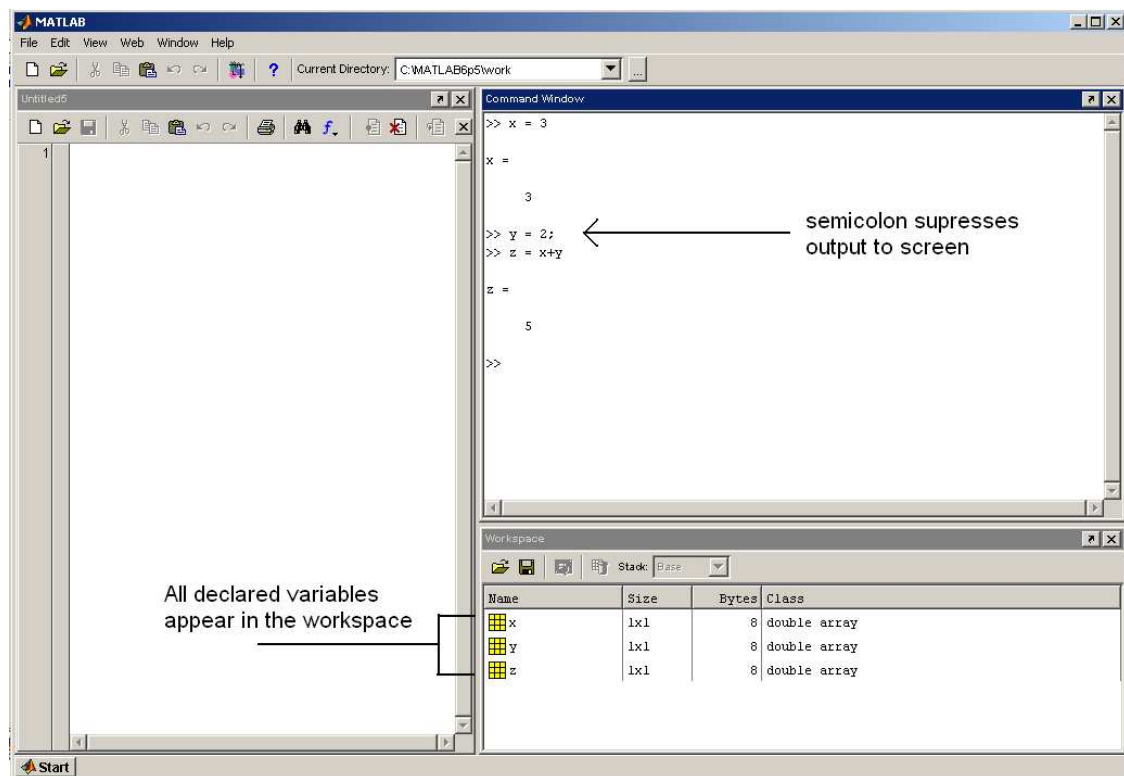


Figure 4: Entering in scalar values into **Matlab**

All declared variables appear in the workspace. Recall that these values are stored as matrices. The “size” column tells us the dimension of the matrix. As expected, all these variables are 1x1 scalar values. To double check on value stored in this matrix, simply double click any of the variables in the Workspace.

2. Now, let's assume that x and y are actually components of a 2D vector. Let's construct the vector  $\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix}$ . Note that we are making a column vector of size

2x1. We use the “[“ to denote the start of a matrix and “]” to denote the end of the matrix. The command to construct the vector is shown below

```
>> v = [x;y]

v =

     3
     2
```

Also notice that in the workspace, the variable “v” is of size 2x1 as expected.

3. In a similar fashion, if we want to create a horizontal vector, we use the space instead of the “;” to separate elements. For example

```
>> p = [12 -3]

p =

    12    -3
```

Once again notice that the variable “p” is created in the workspace and it is of size 1x2 as expected

4. We can create a 2D matrix in a similar fashion. You can use the “[“ to start the matrix, type in the first row with spaces in between elements, use a “;” to start the next row, and then repeat. Finally close the matrix with a “]“. For example, to create a 3x2 matrix, we can use syntax like

```
>> A = [1 2; 3 4; p]

A =

     1     2
     3     4
    12    -3
```

Notice that we used the variable ‘p’ as the last row. This is possible since the dimensions match.

5. Matlab treats most variables as matrices, and therefore operations like addition, multiplication, etc. must be done with matrices whose dimensions are consistent. For example, we could enter

```
>> q = A*v

q =

    8.5000
   15.5000
   24.0000
```

Note: Try multiplying v on the left of A (ie v\*A) and see what happens. This is not allowed because the dimensions are not consistent. This will become important later.

- Now, let's assume that we wanted to change the value of 'x'. This would be tedious to retype in all the commands again, so let's use the m-file to avoid retyping in all the commands each time we make a change.

The m-file is like a source file which will run all your commands in a top to bottom fashion. You can use the “%” sign to comment out lines. A sample m-file is shown below in Figure 5.

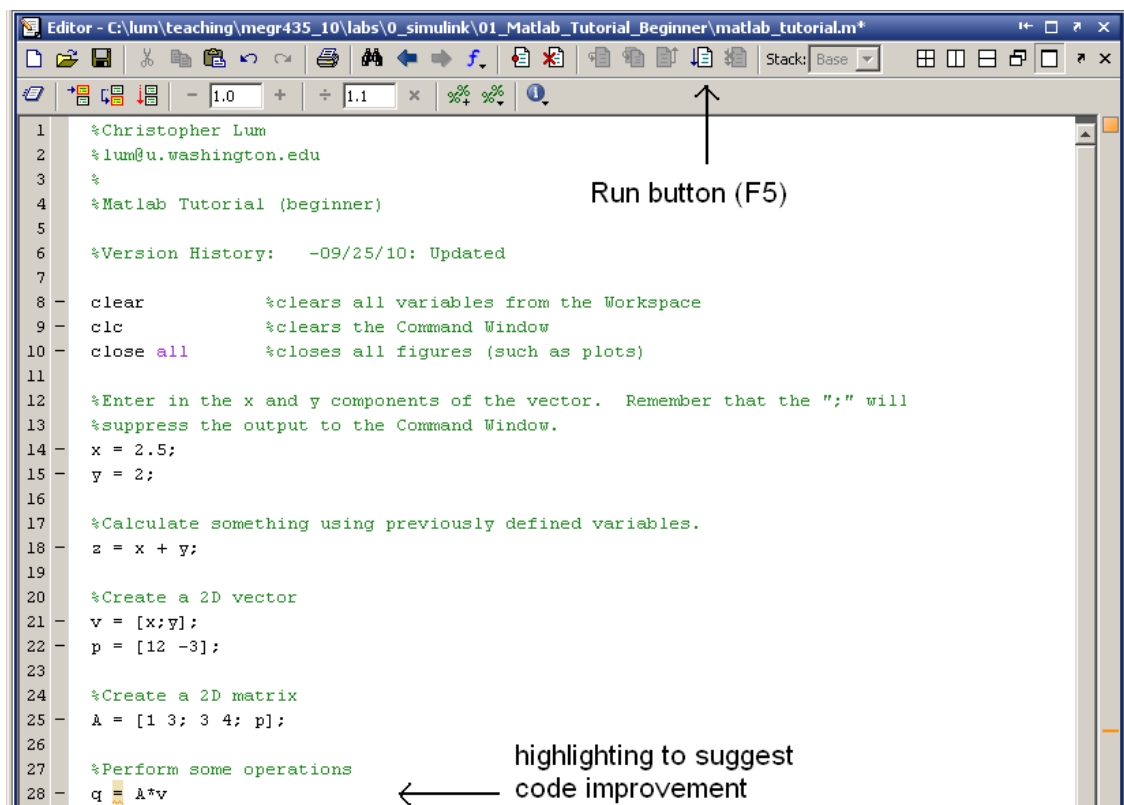


Figure 5: Example m-file for code so far

- You can now run this file by hitting the “Run” button or hitting F5. At this point, Matlab will force you to save you project. Navigate to your desired directory and

save the m-file. After you hit save, you will be prompted with a warning message as shown in Figure 6.

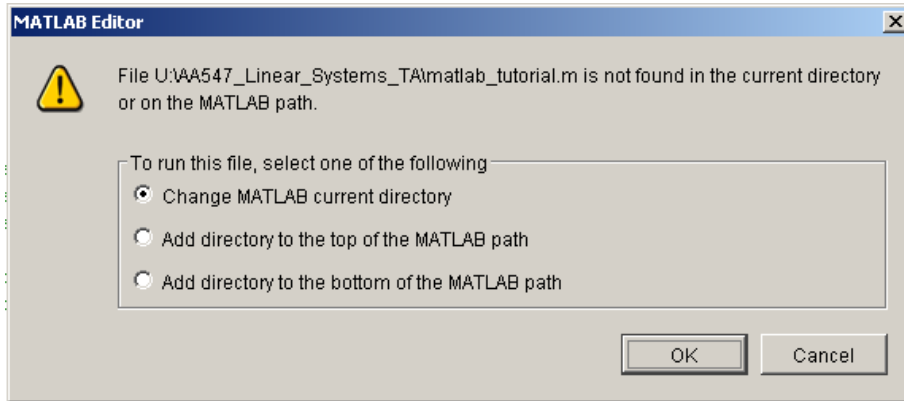


Figure 6: Current directory warning

This warning appears because the m-file that you are trying to run is not located in the current working directory. Therefore, you should select the first option (Change Matlab current directory) and hit OK.

## 2D Plotting with Matlab

Matlab has powerful plotting functions which make visualizing functions easy. Let's consider the function

$$f_1(x) = 3x + 4$$

1. If you were to plot this by hand on graph paper, you would probably follow a procedure such as
  - a. Choose a series of x values where you would like to evaluate the function.
  - b. Compute the corresponding y value at each of these x values.
  - c. Plot each x,y pair.
  - d. Connect the points with a straight line.



Table 1: Table of coordinates

$x$	$y = f(x)$
-1	$3*(-1) + 4 = 1$
0	$3*(0) + 4 = 4$
1	$3*(1) + 4 = 7$
2	$3*(2) + 4 = 10$
Etc.	Etc.

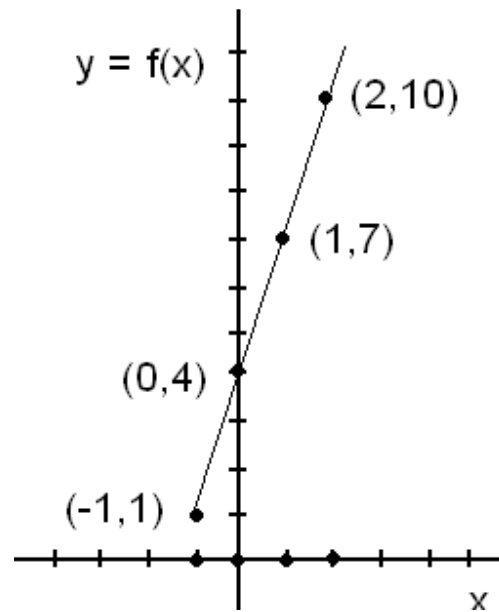


Figure 7: Plotting coordinates by hand

We can first create a vector which represents the  $x$  values where we would like to evaluate the function. Syntax in your m-file might look like

```
x = [-1 0 1 2];
```

2. We can now evaluate the function at these  $x$  values. Syntax in your m-file might look like

```
y = 3*x + 4;
```

3. Now that we have two vectors (one representing the  $x$  values and the other representing the  $y$  values), we can plot the function to see what it looks like. To do this we will use the “plot” function. To obtain help about any of Matlab’s functions, simply type in “help name\_of\_desired\_function” into the Command window. For example for help on the “plot” command, you would type “help plot” in the command window. Syntax in your m-file might look like

```
figure
plot(x, y)
```

The resulting plot is shown in Figure 8.

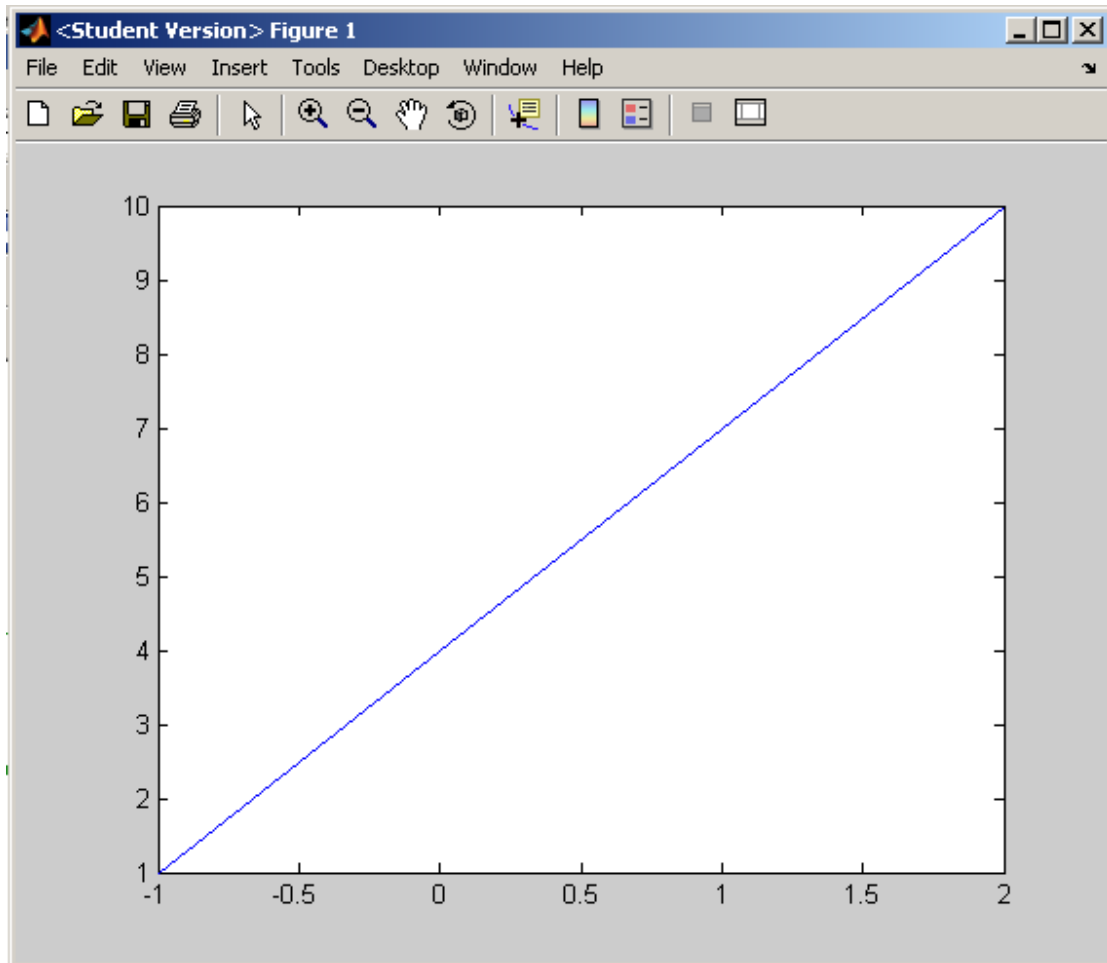


Figure 8: Plot of function

4. We can label the axis, add a title, turn on a grid, put a title and legend on the plot, and force the axis to a set range. Are more complete plotting syntax in your m-file might look like.

```

figure
plot(x,y)
xlabel('x')
ylabel('y = f(x)')
grid on
title('Plot of function f_1(x)')
legend('f_1(x)')
axis([-2 4 -2 12])

```

The resulting plot is shown in Figure 9

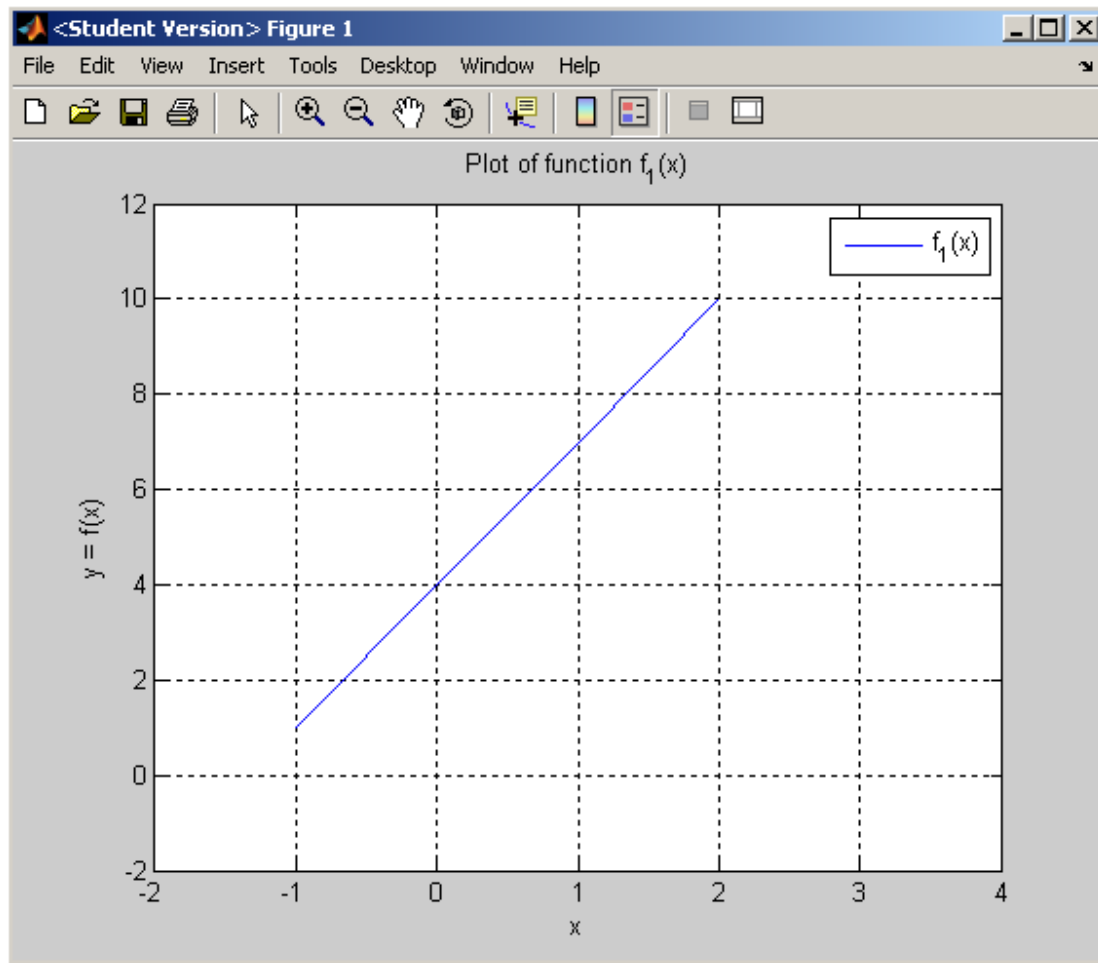


Figure 9: Figure with added options

5. You can export this figure so that it may be easily included into other documents. To do this, go to the figure and select

File > Save As (choose export file type the hit 'Save')

The resulting figure is shown in Figure 10.

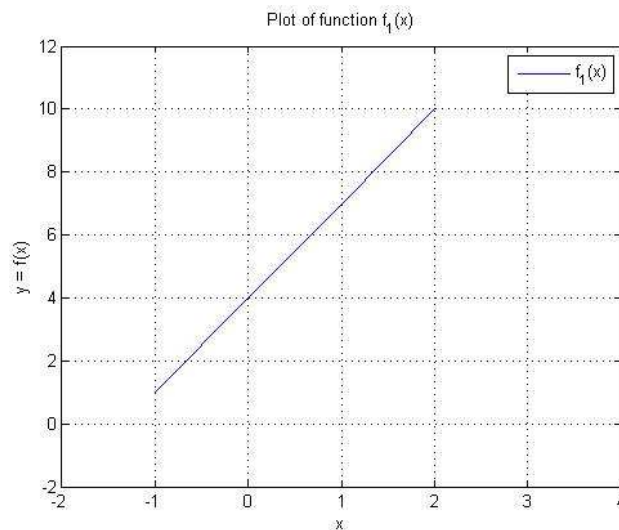


Figure 10: Plot of function f1 after being saved as a .jpg

Let's now consider a second function of the form

$$f_2(t) = g(t)h(t)$$

Where  $g(t) = 3t^2$

$$h(t) = \sin(4t + 4)$$

1. Once again, the first thing we need to do is create a list of  $t$  values where we would like to evaluate the function. Instead of manually typing in values, we can use the syntax as shown below

```
t = [0:1:5];
```

This says create a 1 dimensional matrix (a vector) starting from 0, ending at 5, with increments of 1. In other words, the variable  $t$  should look like (note that it is a 1 x 6 matrix)

$t$					
0	1	2	3	4	5

2. Let's now create a variable 'gt' which is  $g(t) = 3t^2$ . At first, you might be tempted to try syntax such as

```
gt = 3*t^2;
```

This will cause an error. To understand why, recall that Matlab treats most variables as matrices. Also, the command  $t^2$  is interpreted as  $t*t$ . If you recall,  $t$  is a 1 x 6 matrix. You cannot multiply a 1 x 6 matrix with a 1 x 6 matrix. This is what causes the error.

In reality, what you would like to do is

$$t^2$$

$0^2$	$1^2$	$2^2$	$3^2$	$4^2$	$5^2$
-------	-------	-------	-------	-------	-------

In other words, you would like to apply the square operation to each element of the matrix individually. This is known as an element-wise operation. The syntax to perform this is

```
gt = 3*t.^2;
```

The '.' in front of the operator denotes an element-wise operation

3. We can create  $h(t) = \sin(4t + 4)$  using the syntax

```
ht = sin(4*t + 4);
```

Note here that the term '4\*t' is a 1 x 6 matrix and the '4' is simply a 1 x 1 scalar. In this case, when adding a 1x1 to any other sized matrix, it will automatically apply an element-wise operation. This is why you do not need to use a '+' to perform the addition. The same goes for scalar multiplication (multiplying a 1 x 1 matrix into an m x n matrix).

4. We can finally evaluate  $f_2(t) = g(t)h(t)$  using syntax of

```
f2 = gt.*ht;
```

Notice that we need to use element-wise multiplication '.'\*' because  $gt$  is a 1x6 and  $ht$  is a 1x6 matrix.

5. Once again, we can plot this function. To illustrate how the plotting operates, let's plot the x,y points as large red 'x' marks. Syntax to do this is shown below

```
figure
plot(t,f2,'rx')
```

The output is shown below in **Figure 11**.

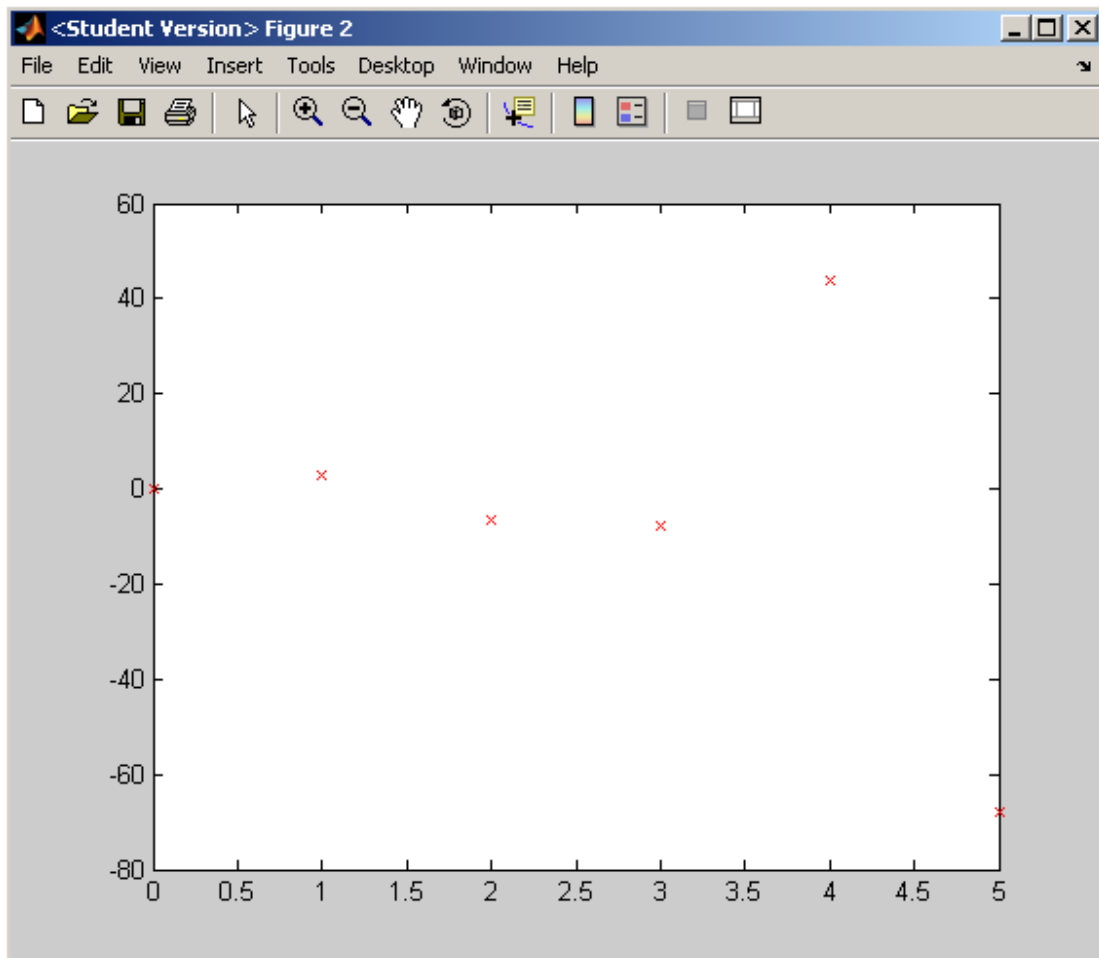


Figure 11: Plot of function  $f_2(t)$  using only 6 points

6. As can be seen in **Figure 11**, this is not a very fine resolution representation of the function. To fix this, we can go back to where we defined the variable  $t$  and choose a smaller step size. For example

```
t = [0:1:5];
```

↓ change to

```
t = [0:.01:5];
```

yields the plot shown below in **Figure 12**.

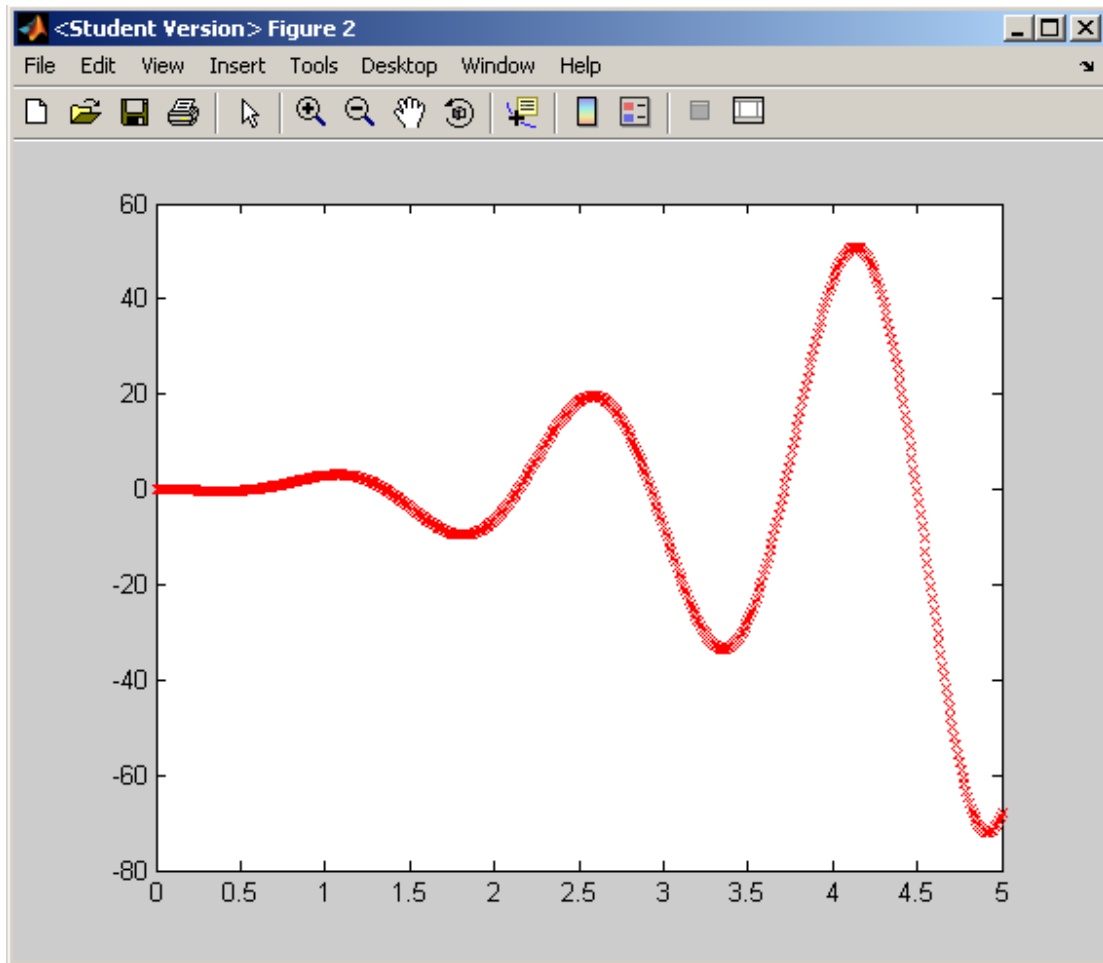


Figure 12: Function f2 after decreasing step size in  $t$  definition

A collection of m-file syntax for this section is shown in **Figure 13**.

```

%-----2D PLOTTING WITH MATLAB-----
%Function 1:    f1(x) = 3*x + 4
x = [-1 0 1 2];    %desired x values of function
y = 3*x + 4;      %function evaluated at x

figure            %start a new figure
plot(x,y)        %plot function
xlabel('x')      %add x label to figure
ylabel('y = f(x)') %add y label to figure
grid on         %turn on the grid on figure
title('Plot of function f_1(x)') %add title to figure
legend('f_1(x)') %add legend to figure
axis([-2 4 -2 12]) %force axis to be drawn in specified range

%Function 2:    f2(t) = g(t)*h(t)
%
%   where      g(t) = 3*t^2
%              h(t) = sin(4*t + 4)
t = [0:.01:5];

gt = 3*t.^2;
ht = sin(4*t + 4);

f2 = gt.*ht;

figure
plot(t,f2,'rx')

```

Figure 13: Complete m-file for 2D Plotting with Matlab section

---

Version History:	09/14/04: Created:	
	11/23/05: Updated:	Made this format match other to-do documents and removed references to AA547.
	12/01/05: Updated:	Changed headers to match how-to template
	12/09/05: Updated:	Made changes to layout and added footer.
	09/15/09: Updated:	Minor changes
	09/25/10: Updated:	Made major changes to make it more readable and understandable. Changed content significantly.
	04/12/12: Updated:	Updated some screenshots for new Matlab version.