Introduction to MATLAB

Introduction:

MATLAB is a powerful high level scripting language that is optimized for mathematical analysis, simulation, and visualization. You can interactively solve problems by command line entries, or write extensive scripts and functions as part of a larger program structure for more challenging problems. The language syntax by default assumes all data arguments are matrix or vector based, so it is well suited for problems of high dimensionality and manipulating large data sets (like the sampled signals we will study!)

MATLAB has a huge library of built-in functions which implement a variety of complex operations, including many which are ideally suited for application in signals and systems, linear algebra, statistics, digital communications, control systems, digital signal processing, system analysis, and many other engineering and science disciplines.

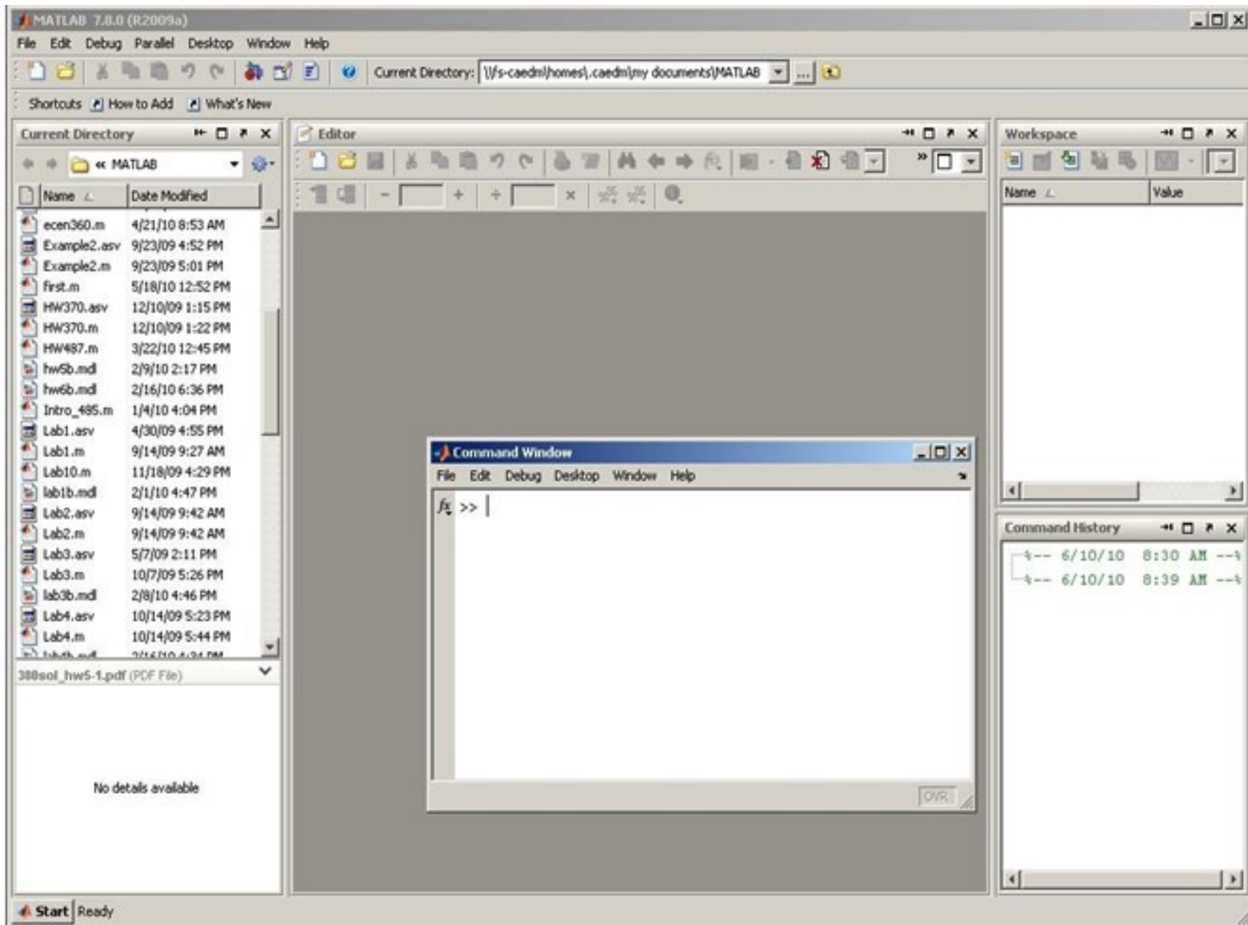
To become proficient in MATLAB requires that you develop familiarity with its already programmed-in capabilities, or that you learn how to search the excellent set of documentation and help tools to find built in functions that fit your needs. It also has a complete program development suite of tools including a real-time debugger to assist you in efficient code development. MATLAB is a tool you will use in several classes and for much of your engineering career. You may also later encounter languages (which are typically free shareware) such as Python, Octave, and Ruby. These are similar in concept to MATLAB but have significant syntax differences.

Task 1. Introduction to MATLAB basics

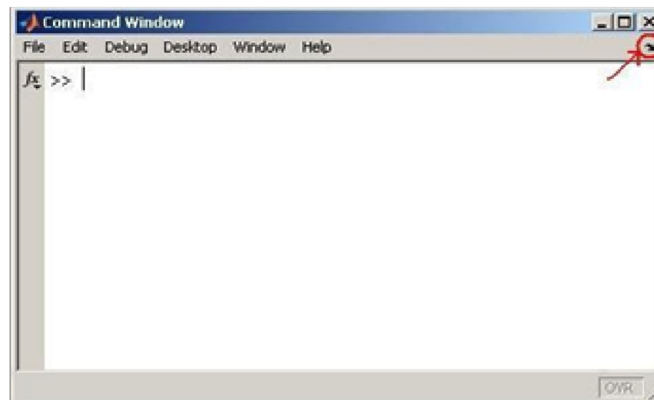
Starting MATLAB

To open MATLAB on a Windows computer, click Start >> All programs >> Math Programs >> MATLAB 2013/2014/2015

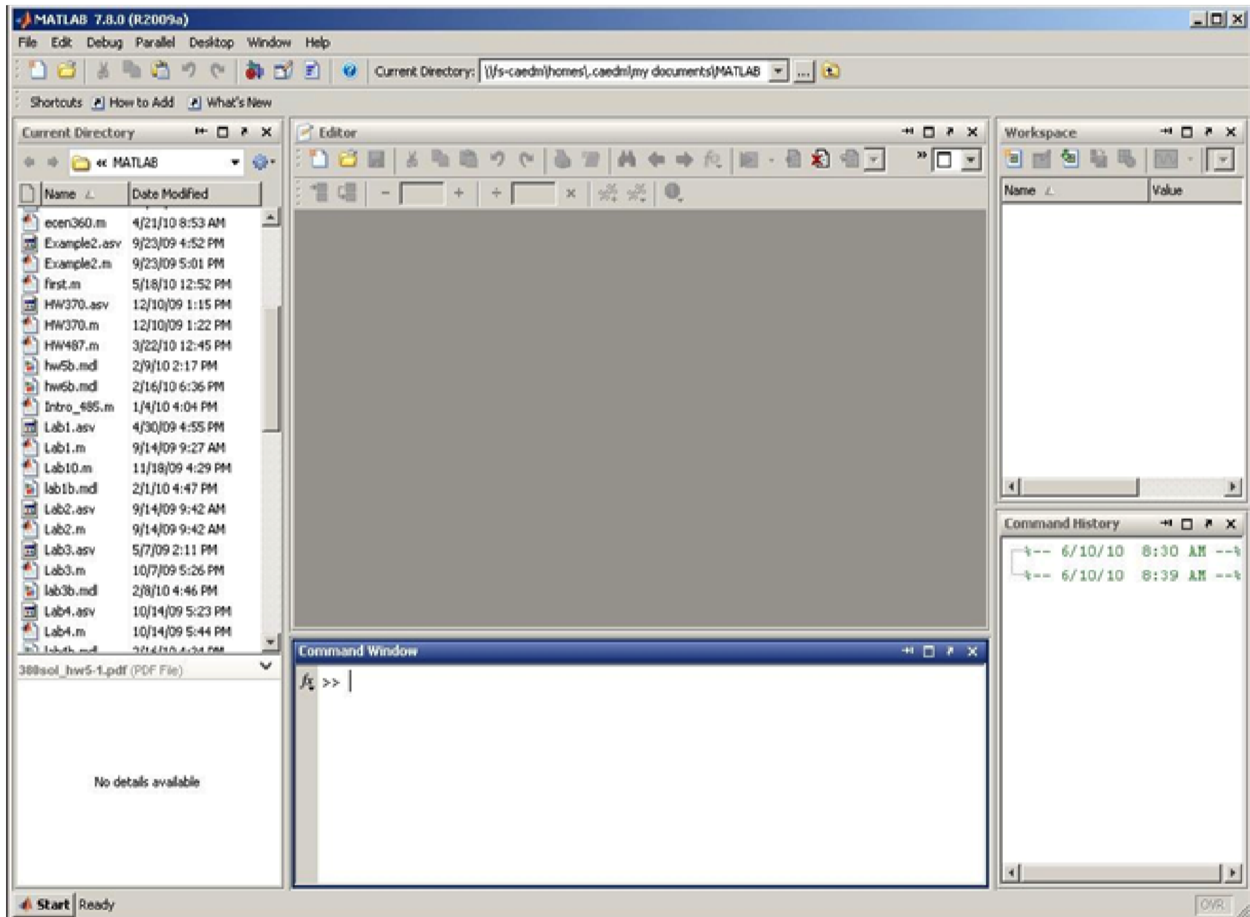
When you first open MATLAB, it should look something like this:



Notice that there is a main program and also a smaller, separate command window. You will be using both to program. It is probably a good idea to dock the smaller console onto the main program so you don't have to keep switching back and forth. This can be done by clicking on the Desktop menu option of the smaller command window and then clicking the only option: Dock Command Window or clicking the arrow on the command window shown below.



Your screen should now look something like this:

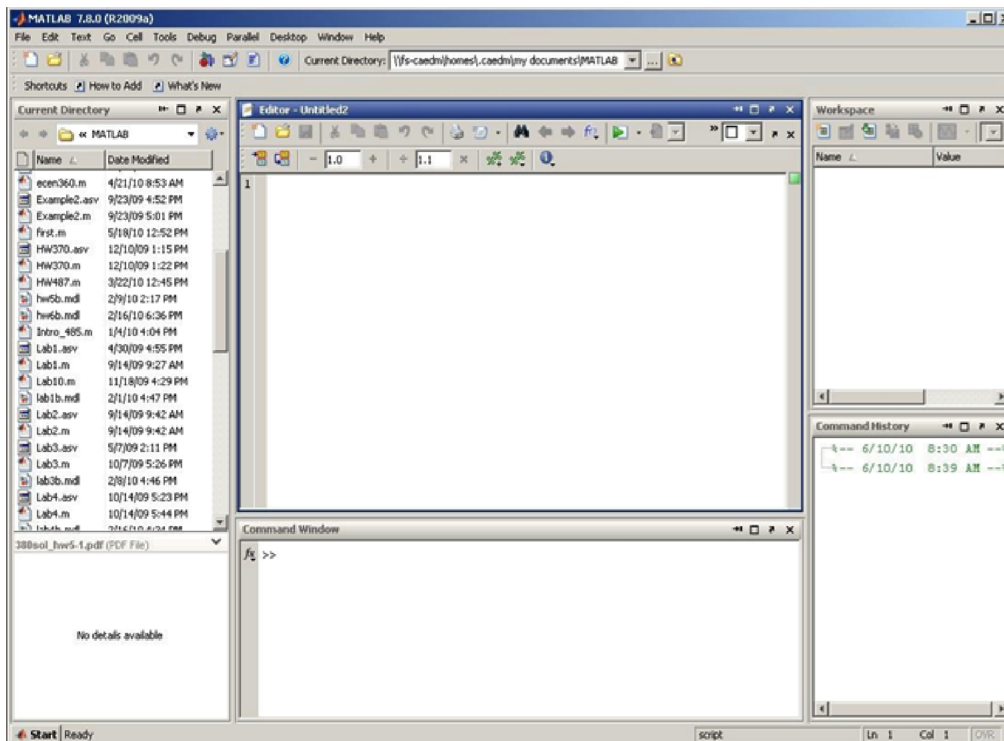


Notice on the left side of the screen, MATLAB shows the directory you are in and all the files and folders that are saved there. This is where you will keep all the programs you write or import. You can organize or create information there directly. On the right side of the screen in the upper quadrant is where all your variables will display. The right, lower quadrant will display your recent commands from the command window.

Creating an M-file

You can calculate problems create and use variables, etc. from the command window, but it is generally better practice to make your own file from which to run your program. M-files are scripts of MATLAB commands that are stored in a text file. These scripts constitute your programming source code. M-files allow the user to edit code without reentering it into the command line. To create an M-file, select File >> New >> Script or select the New script button, circled in red in the picture below.

Your screen should now look like:



When you start writing your program, you will need to save it as ".m" file in the MATLAB directory before you can run it.

MATLAB variables

In the following discussion, whenever you see indented lines with this font, like the line below, type this code into MATLAB at the ">>" prompt.

MATLAB allows you to directly calculate and display computational results to the screen without creating storage in memory. For example, try typing

```
2*1.5
```

• But even more useful, to calculate a value and store it with a name (a variable), enter the following code shown below:

```
x = 3*sqrt(2);
x
```

Note that ending the line with a semicolon ";" suppresses printing the value to the screen, but typing "x" allows you to print out that variable's value. Now try

```
x = 3*sqrt(2)
```

You see that screen output is not suppressed. In many of the statements in this lab, the semicolon is deliberately omitted so that you can see the results. However, most of the time you use MATLAB, it is probably a good idea to include the semicolon.

Now enter these same lines of code into an M-file and save it into your current working directory with filename "first." From the command line, enter "first" and see what happens.

• You can create a vector-valued output as follows:

`vec = [3 2 5.6]` and

perform operations on it:

`vec3 = 3*vec` which

multiplies each element by 3.

Note: You can use the Workspace window to view the values of variables and vectors you created.

- An incremental vector (that is, a vector that contains a series of numbers each separated by a fixed increment) can be created as [start:increment:end]. The notation [start:end] assumes an increment of 1. Let's first create a vector that runs from 1 to 100, incrementing by 1.

```
index1 = 1:100
```

Remember, to suppress the output when creating the vector you can simply add a semicolon to the end of the command:

```
index1 = 1:100;
```

Try creating a vector index2 from 0 to 5 that increments by 0.5:

```
index2 = 0:0.5:5
```

*How about an index that runs from 0 to 2*pi with 10 elements?*

```
index3 = 0:2*pi/10:2*pi
```

Does index3 have 10 elements? To see the number of elements in a vector, type:

```
length(index3)
```

Can you explain why index3 has 11 elements?

To transpose our index3 vector, type the following:

```
index3'
```

Or we could store a transposed version of index3 in a new vector index4:

```
index4 = index3';
```

- A 2-D array (or matrix) can be formed as follows:

```
array1 = [ 3 2 5.6 7; 1 4 5 9; 1 2 3 4]
```

where spaces (or optionally commas) separate columns, and semicolons separate rows. To reference a single element of an array, put the index values in parentheses. In the previous example, to reference the 3rd row and 2nd column, type:

```
q1 = array1(3,2)
```

You can also refer to a range of elements:

```
q2 = array1(1:3,4)
```

The 1:2 in this case selects the 1st through 3rd rows in the array, and the 4 selects the 4th column. You can even reference ranges of arrays on multiple dimensions. Try:

```
q3 = array1(2:3,2:4)
```

To select all elements of an array on a given dimension, a colon alone can be used for that dimension:

```
q4 = array1([1 3],:)
```

Note that the [1 3] selects just the 1st and 3rd rows of the matrix, while the : selects all columns.

You can also use indexing ranges on the assignment side of the equation. Try:

```
array2 = array1   array2(2:3,2:3) =
[0.5 0.5; 0.7 0.7]
```

What happened?

Spend some time to understand why you get the output you get. Effective indexing into vectors and matrices is a powerful skill in MATLAB. Try some new matrices and indexing selections and assignment variations of your own.

- MATLAB conveniently allows you to perform the same operation on every element of an input vector or array:

```
z1 = sqrt(index2);
```

or:

```
z2 = sqrt(array1);
```

This calculates the square root of every element of index2 and array1 and stores the results in z1 and z2 respectively. Note that we've used the semicolon at the end of the each line to suppress the output. To view the contents of a variable, vector, or array, just type its name at the MATLAB prompt and hit enter:

```
z1
```

Matrix operations and element-by-element operations

MATLAB can perform operations such as multiplication on vectors and matrices in different ways. Let's create a couple of simple 2-dimensional matrices:

```
 m1 = [1 4 3 ; 2 3 1 ; 5 4 3 ]
m2 = [1 1 1 ; 0 0 1 ; 0 2 0 ]
```

Now try the following:

```
m1*m2
```

This performs a matrix multiplication. If we wish instead to multiply each element in m1 and m2 by each other on an element-by-element basis, we can type:

```
m1.*m2
```

What is the difference between the following two operations?

```
m1^2 m1.^2
```

Basic plotting

MATLAB has a number of powerful plotting capabilities. We will look at a few of them here.

- plot(X,Y) plots the vector Y versus the vector X.

Remember our vector index3, which contains values running from 0 to 2*pi. Let's plot it:

```
 plot(index3)
```

Now let's plot its sin of these values:

```
plot(sin(index3))
```

Note that MATLAB plots each element of index3 or sin(index3) as connected lines. The horizontal axis begins with 1 and counts forward by default. Often this is not very useful. If you want to use your own index and add labels, type the following:

```
plot(index3, sin(index3));
xlabel('input');
 ylabel('output');
```

Suppose you want to compare sin(index3) to cos(index3) on the same plot with different line styles and also add a legend:

```
y1 = sin(index3); y2
= cos(index3);
plot(index3,y1,'-',index3,y2,':') legend('sine','cosine')
```

See “help plot” to learn about changing line colors and other line styles.

- Suppose you just want to plot y1 as a discrete sequence, type the following:

```
stem(index3,y1) legend('sine','cosine')

stem(index3,y1,'+')
legend('sine','cosine')
```

- A bar graph requires putting the input into a different format. Combine the two row vectors y1 and y2 into a single matrix by transforming each row into a column and combining:

```
newmatrix = [y1' y2']
```

Then bar plot:

```
bar(index3,newmatrix)
```

If you don't like the fact that MATLAB creates extra space on the margins of the plot, try:

```
axis tight
```

- 2-D plots of different sorts are also possible:

```
xind = [0:0.2:20]; yind
= [0:0.2:10];
[xx,yy] = meshgrid(xind,yind); % create matrices of x values and y
sinxy = sin(0.1*pi*xx - 0.2*pi*yy); mesh(xind,yind,sinxy)
```

You can create a new figure with

```
figure
```

or activate an existing figure (for example, #1) with

```
figure(1)
```

You can create a new figure with

- Create some 1-D and 2-D functions of your own and plot them.

Special arrays

MATLAB can create a number of different special arrays with a single command. Here are some of the more important ones.

zeros (surprise!) creates an array of zeros:

```
z0 = zeros(5,2)
```

ones (surprise again!) creates an array of ones:

```
o1 = ones(6)
```

Some random but useful MATLAB tips

Once you have assigned a value to a variable in MATLAB (like o1 in the example above), that variable stays active for the remainder of your MATLAB session. In order to view all of the variables that are currently defined for your session, try typing:

```
who
```

As you see, this returns a complete list of defined variables. If you want more information about your current variables, you can type:

```
whos
```

This command gives you additional information about each variable, such as its size and type. To clear a variable and remove it from the variable space, you can use the clear command. If you use the clear command without any arguments, it clears the entire variable space. Don't try this right now, as we'd like to keep our variable space intact for the moment. But, try the following:

```
z1 = zeros(10);
whos z1 z1 clear
z1
z1
```

A common mistake for scientists and engineers using MATLAB is to accidentally redefine the variables i and j. These lower case variables are both pre-defined in MATLAB as the square root of negative one. MATLAB allows you to redefine them, e.g.:

```
j = 20
```

Since these variables are commonly used in programming as indices or counters, it is common for us to get careless and use them as indices or counters in MATLAB. I STRONGLY suggest getting in the habit of using the symbol names 1i and 1j in MATLAB when you want –1. These are also pre-defined, but you can't muck them up because you are not allowed to redefine them.

While at the MATLAB prompt, you can see the current directory (or folder) that you are working in using the command:

```
Pwd
```

You can change directories using the cd command and view files in the current directory using the ls command. Your current variable space can be saved to a .mat file (a file with the .mat extension) using the save command, and then reload that variable space using the load command. Try the following:

```
save myvars.mat ls
clear whos load
myvars whos
```

As you'll see, this series of commands (1) creates a file in the current directory called "myvars.mat" which contains your current variable space, (2) clears out the variable space, and (3) reloads your variable space.

Conclusion

As you continue the course you will learn other MATLAB tools and commands. Consult the MATLAB guide on the wiki page if you have any questions, or search for it online.