Lab 1: Introduction to MATLAB

The objective of this lab is to introduce you to the basic operations of MATLAB. Read through the handout sitting in front of a computer that has a MATLAB software. Practice each new command by completing the examples and exercise. Turn-in the answers for all the exercise problems as your lab report. When answering the problems, indicate the commands you entered and the output displayed. It may be easier to open a word document, and after each command you may copy and paste the commands and outputs to the word document. Similarly, plots can also be pasted to your word document.

1. Introduction

In this first lab, we will learn how to

- perform basic mathematical operations on simple variables, vectors, matrices and complex numbers.
- generate 2-D plots.
- use and write script files (MATLAB programs). MATLAB script files have a file extension name .m and are, therefore, usually referred as M-files.
- Use the 'help' and 'lookfor' commands to debug codes.
-

2. Starting MATLAB

Start MATLAB by clicking on it in the **Start** menu.

Once the program is running, you will see a screen similar to Figure 1.3

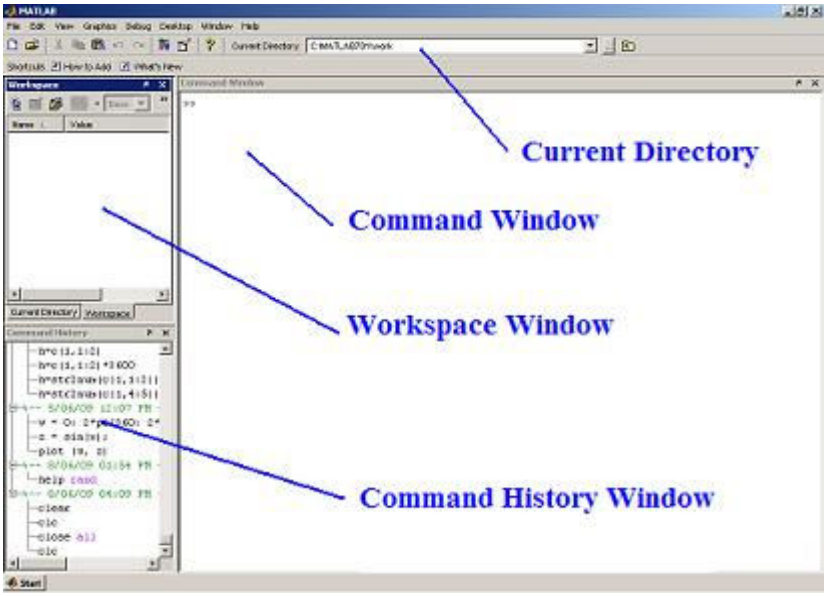

Figure 1: The MATLAB Graphical User Interface (GUI) (Image taken from http://www.matrixlab-examples.com/using-MATLAB.html)

The Command Window: is where you type commands. Hit Enter to run the command you just typed.

The Current Directory: shows the directory that you are working in. This directory is where MATLAB expects to find your files (M-files, audio, images, etc). If you encounter a 'file not found' error, it means the file is not in your Current Directory. You can change the working directory by typing into it or clicking through the file browser.

The Workspace Window: displays information about all the variables that are currently active. In particular, it will tell you the type (int, double, etc) of variable, as well as the dimensions of the data structure (such as a 2x2 or 8000x1 matrix). This information can be extremely useful for debugging!

The Command History Window: keeps track of the operations that you've performed recently. This is handy for keeping track of what you have or haven't already tried.

4 MATLAB Commands

4.1 Help

MATLAB has two important help commands to find the right syntax and a description of a command with its all options.

Typing ***help*** by itself on the command line gives you a list of help topics.

If you want to find more about the syntax of a certain command, you type in

```
>> help function_name
```
where the word function_name is in the above entry is substituted by the actual name of a function for which description is sought. For example, we can type in

```
>> help plot
```
and MATLAB will display the description, syntax and options of the plot function.

The other helpful command is ***lookfor***. If we are not sure the exact name of the function, we can make a search for a string which is related to the function, and MATLAB displays all m-files (commands) that have a matching string. MATLAB searches for the string on the first comment line of the m-file. For example:

```
>>lookfor  inverse
```

will display all m-files which contain the string 'inverse' in the comment line.

Two other useful commands are

```
>>whos
```

which lists all your active variables (this info also appears in your Workspace Window), and
>> clear
which clears and deletes all variables (for when you want to start over).

MATLAB has **tab-completion** for when you're typing long function names repeatedly. This means that you can type part of a command and then press <Tab> and it will fill in the rest of the command automatically. Moreover, MATLAB **stores command history**, and previous commands can be searched by pressing the up or down arrow keys.

4.2 Matrix Operations

MATLAB is designed to operate efficiently on matrices (hence the name MATLAB = "Matrix Laboratory). Consequently, MATLAB treats all variables as matrices, even scalars!

Like many other programming languages, variables are defined in MATLAB by typing:

<VariableName> = <Assignment>

MATLAB will then acknowledge the assignment by repeating it back to you. The following are what you would see if you defined a **scalar x**, a **vector y**, and a **matrix z**:

```
>> x = 3
x =
     3
>> y = [1, 2, 3]
y =
     1     2     3
>> z = [1, 2, 3; 4, 5, 6]
z =
     1     2     3
     4     5     6
```

You can see from the above examples that scalar variables require no special syntax; you just type the value after the equals sign. Matrices are denoted by square brackets [ ]. Commas separate the elements within a row, and semicolons separate different rows. A row array, such as y, is just a special case of a matrix that has only one row.

Exercise 1:

Define the following column arrays in MATLAB: $a = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$ and $b = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$

Then issue the following commands:
a'
a * b'
a' * b
a .* b
3 .* b

What do each of these three operators do?
’ * .*

Exercise 2:

Perform the following operation:
a * b
What is the error message?
What does the message mean?
What is the correct fix?

Exercise 3:

Perform the following:
c = a + b
d = a + b;
What does the ; do?

4.3 size command

The size command is extremely useful. This command tells you the dimensions of the matrix that MATLAB is using to represent the variable. To determine the dimension of a matrix x, for example, you type in

```
>> size(x)
```

Exercise 4:

Define e = 17 in MATLAB. Use size to find the dimensions of a, b’, and e?
There is an alternate syntax for size to determine the length of a vector. Can you guess? Search for the command either by using the help commands of MATLAB or by searching online.

Use the alternate syntax for size to determine the height of b.

You can also just use whitespace to separate elements within a row. The following two ways to define the variable are equivalent:

```
>> y = [1, 2, 3]
y =
```

```
      1      2    3
>> y2 = [1 2 3]
y2 =
      1     2     3
```

You now know how to define matrices. The ( ) operator allows you to access the contents of a matrix. MATLAB is 1-indexed, meaning that the first element of each array has index 1 (indexing starts from 1 not from 0).

```
>> y = [1, 2, 3];
>> y(1)
ans =
      1
```

To access a single element in a multidimensional matrix, use (i,j). The syntax is matrix(row,column):

```
>> y= [1, 2; 3, 4];
>> y(2, 1)
ans =
      3
```

There is a quick way to define arrays that are a range of numbers, by using the **: operator**.

Exercise 5:

Define the following:
g = 0:25;
h = -10:10;
How big are g and h? Write the command to find their sizes.
What are the first, second, third, and last elements of each?
What exactly do g and h contain?

Create the following vector k as follows:
k = -10:0.1:10;
How big is k?
What are the first, second, third, and last elements?
What exactly does k contain?

4.3 Plot

The MATLAB command **plot** allows you to graphically display vector data in the form of (surprise!) a plot. Most of the time, you'll want to graph two signals and compare them. If you had a variable t for time, and x for a signal, then typing the command

```
>> plot(t,x)
```

will display a plot of the signal x against time. See help plot if you haven't done so already.

You MUST label your plots in order to communicate clearly. Your graphs must be able to tell a story without you being present! Here are a few useful annotation commands:

```
>> title('Here is a title');
```
%– Adds the text "Here is a title" to the top of the plot.

```
>> xlabel('Distance traveled (m)');
```
%– Adds text to the X axis.

```
>> ylabel('Distance from Origin (m)');
```
% – Adds text to the Y axis.

```
>> grid on;
```
%– Adds a grid to the plot.

```
>> grid off;
```
%– Removes the grid (sometimes the plot is too cluttered with it on).

```
>> hold on;
```
%– Draws the next plot on top of the current one (on the same axes). Useful for comparing plots.

```
>> hold off;
```
%– Erases the current plot before drawing the next (this is the default).

In order to display multiple plots in one window , you must use the subplot command.
This command takes three arguments as follows: subplot(m,n,p). The first two arguments break the window into an m by n grid of smaller graphs, and the third argument p selects which of those smaller graphs is being drawn right now.

For example, if you had three signals x, y, and z, and you wanted to plot each against time t, then we could use the subplot command to produce a single window with three graphs stacked vertically:

```
>> subplot(3,1,1);
>> plot(t,x);
>> subplot(3,1,2);
>> plot(t,y);
>> subplot(3,1,3);
>> plot(t,z);
```

See help subplot or look online for more examples.

MATLAB only handles discrete representations of signals. As such, you need a way to represent time for continuous signals in MATLAB. All functions of time in MATLAB must be defined over a particular range of time (and with a particular granularity or increment). Note that the granularity of your time vector will impact the resolution of your plots.

Exercise 6:

Create and plot a signal $x_0(t) = te^{-|t|}$ using the following commands:

```
>> t = -10:0.1:10;
>> xo = t .* exp(-abs(t));
>> plot(t,xo)
```

Create the related signals $x_e(t) = |t|e^{-|t|}$ and $x(t) = 0.5 * [x_0(t) + x_e(t)]$.
What are $x_0(t)$ and $x_e(t)$, relative to $x(t)$?
Plot all three signals together in one window.

4.4 Complex numbers

One of the strengths of MATLAB is that most of its commands work with complex numbers, too. MATLAB, by default, uses the letter i for the square root of (-1). However, electrical engineers typically prefer using j, and so MATLAB has both predefined. Because of this, you may wish to avoid using i and j as variables if your code deals with complex numbers. That being said, everything in MATLAB is a variable. You may redefine the variables i and j to be anything you like.

Exercise 7:

---

1. Enter sqrt(-1) into MATLAB. Does the result make sense?
2. Enter i+j. Does the result make sense?
3. Define z1 = 1+j. Find the magnitude, phase, and real part of z using the following operators: abs(), angle(), real(), imag(). Is the phase in degrees or radians?
4. Find the magnitude of z1 + z2, where $z_2 = 2e^{\frac{1}{3}j\pi}$
5. Compute the magnitude of j^j . What do you expect the result to be?

4.5 Complex functions

MATLAB handles complex functions of time in the same way as real ones: defined over a range of time.

Exercise 8:

Create a signal $x_1(t) = te^{jt}$ over the range [-10,10] as in Exercise 4.
Plot the real and imaginary parts of x_1 in one window (using subplot, of course).
Notice that one plot is odd, and the other is even. Why is that?

4.6 Loading and plotting a sound

Exercise 9:

You will be playing and visualizing a lot of sound files in this course.
Load the built-in sound named ***handel***, plot it, and play it.
TURN THE VOLUME DOWN ON YOUR COMPUTER FIRST!

```
>>load handel;
>>plot(linspace(0,9,73113),y);
>> sound(y);
```

What does load do?
What does linspace do?

---

5 Script Files

Scripts are M-files that contain a sequence of commands that are executed exactly as if they were manually typed into the MATLAB console. Scripts are useful for writing things in MATLAB that you want to save and re-run later. A script file also gives you the ability to go back later and edit your commands, such as when you would like to re-run a function witha different parameter.

To create script files, you need to use a text editor such as Notepad on a Windows PC or emacs on Linux and Mac computers. MATLAB also has an internal editor that you can use within the MATLAB GUI. You can start the editor by clicking on an M-file within the MATLAB file browser. All of these editors are standard tools and will produce plaintext files that MATLAB can read.

Your M-files should each contain a header. In general, a header is a block of commented text that contains the name of the file, a description, and your name and the date. For example:

```
% john smith
% ee235 spring 2010, lab 1
% dampedCosine.m
% produces a plot of a cosine with frequency 1 Hz, with amplitude
% scaled by a decaying exponential (y).
```

<code goes here>

Download the dampedCosine.m script from the webpage

http://cnx.org/content/m13554/latest/dampedCosine.m

Save the script in your current working directory, otherwise MATLAB will not be able to find the file. You need to run the dampedCosine.m script by typing dampedCosine at the MATLAB prompt. Open the script in an editor, and read the code. You may safely ignore the 'diary' commands entirely.

Exercise 10:

Create a copy of dampedCosine.m, and call it dampedCosine_YourName.m. Open dampedCosine_YourName.m in an editor and give it a proper header. Edit dampedCosine_YourName.m to create a second signal that is a cosine with twice the period as the original signal. Comment your code.

Add commands to plot the two signals together, with the original signal on top and your second signal on the bottom. You will need to use the subplot and plot commands. Comment your code.

Run your script, and save the output plot as dampedCosine_YourName.jpg. (You can typically just right-click save the image). Turn-in the code and output with your lab report.

Exercise 11:

What is the period of the second signal?
How do the envelopes of the two signals compare?
What is the difference between the second signal that you just made, and a third signal which has half the frequency of the first one?

Download the compexp.m script from http://cnx.org/content/m13554/latest/compexp.m and save it in your current working directory. This script specifies a complex exponential funtion y(t). This script generates five plots: one 3-D (X,Y,Time), and two pairs of 2-D plots: one pair in Cartesian coordinates (Real and Imaginary axes) and one pair in polar coordinates (radius and angle).

Run compexp.m, and look at the graphs. You can rotate the 3-D graph around by clicking on the "Rotate 3D" toolbar button, then clicking on the graph and dragging the pointer around.

Exercise 12:

Create a copy of compexp.m called compexp_YourName.m
Add a second signal to compexp_YourName.m that has half the oscillation frequency of the original. Generate a new set of 5 plots for the second signal.
Add a third signal to compexp_YourName.m that decays noticeably faster than the original. Generate a new set of 5 plots for the third signal.
Check that your code is commented and has a header.
Run your script, and save the output plots as compexp_YourName_1.jpg through Compexp_YourName_3.jpg.

Exercise 13:

Explain intuitively what the 3-D process looks like in 2-D.
Explain how the 3-D graph is consistent with each of the pairs of 2-D graphs.
Explain how the plots of the second and third signals confirm that your code is correct.
In other words: how do your output signals satisfy your goals of "oscillate faster" and "decay faster"?
