Lab 1
Introduction to MATLAB and Scripts

EE 235: Continuous-Time Linear Systems
Department of Electrical Engineering
University of Washington

The development of these labs was originally supported by the National Science Foundation under Grant No. DUE-0511635. Any opinions, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

1 Introduction

This lab provides exercises to get you started in MATLAB. You will learn how to use vectors, complex numbers, basic math operations, and generate 2-D plots. You will learn how to use script files in MATLAB, which have the file extension `*.m` and which we will refer to as 'M-Files'. You will also learn to use the `help` and `lookfor` commands to assist you with debugging your code.

You may find it useful to try some of the built-in demos in MATLAB to get more practice. Type `demo` to see the choices. In particular, look at the demo on "Basic Matrix Operations" (under "Mathematics") and the demo on "2-D Plots" (under "Graphics").

There are a very large number of useful MATLAB tutorials online. This lab handout is not intended to replace them. For a more thorough explanation of any aspect of using MATLAB, just search for `matlab tutorial` on the Web (or follow the links on the lab homepage).

[1] Last revision: Fri Apr 9 01:56:34 EDT 2010
[2] http://www.creativecommons.org/licenses/by/2.0

2 Meta-Section

2.1 Victory Conditions

There are several of problems for you to solve in the labs. Some will ask you to **do** something, such as write some code and produce a plot. Others will simply ask you to **explain** something. The problems are indicated by boxed text, like so:

> **Exercise 0:**
> • Log in to your EE account on the lab computer, and start MATLAB.
> • What does the name MATLAB come from?

In order to successfully complete a lab, you must complete all the exercises. This includes: writing reasonable MATLAB code (with a header and comments), producing all required output, showing everything to the TA and archiving your work. You should also know how to answer all the questions.

2.2 Getting Unstuck

This course requires you to learn how to program specifically in MATLAB. However, a solid understanding of the programming **process** will make your life significantly easier, both in this course and in your future work.

Remember, programming is a process of iterative improvement. Error codes are your **diagnostics**, not signs of failure. Plan your code before you start working. If it doesn't run correctly the first time, then start from where your program crashes, and push onwards!

Be methodical. Though it is counter-intuitive, being slow and careful while debugging will let you finish faster overall. Here are some useful questions to ask yourself at various points in the process.

2.2.1 Before you start writing code

These questions may help you visualize everything you need to keep track of in order to answer the question you're working on.

- What is the goal?
- What are your inputs?
  "X and N" is not a good answer. "A sinusoid of constant amplitude and frequency, and also an integer" is good.
- What do your outputs need to be?
- What does your input look like? Don't assume, nor guess: plot it!

- What is the data structure holding your input?
  Again, "x" is a bad answer. "A single-row matrix with 4000 elements" is good.
- What is the data structure holding your output?
- Do you need any sort of loop or control structure in your code?
- If yes, then what are the conditions for when and how much to loop?
- What intermediate data structures do you probably need to define?

2.2.2 When you think you're finished

These questions can be useful to sanity-check your answer.

- Did you accomplish your goal?
  That is, did you do everything you needed to?
- What does your output data structure look like?
  Is that what you expected?
- What does your output look like? (don't assume, nor guess: plot it!)
  Does it look like you expected?
- Using your plots (input and output), and your data structures or variables, how would you convince someone that you did what you were supposed to?

2.2.3 Whenever something doesn't work

You will inevitably spend most of your time debugging code. It is very easy to kill a lot of time debugging ineffectively or inefficiently. Here is where being methodical can really pay off. When you see an error message, or when your output is incorrect, ask yourself the following:

- What is the error message?
- What line is it pointing to?
- What does the message mean?
  If you don't know, then search for the error text online!
- Have you seen this message before, or can you find an example online?
- If yes, what sorts of things have triggered it before?
  Example: "matrix dimensions must agree", when trying to multiply a row and a column
- How did you solve this problem last time?

- What is the specific function, variable, or operation that is not working?
- What arguments does this function/variable/operation take? Use `help`.
- What are the arguments that you *should* be passing to this function/variable/operator? i.e. What should their value be? What should their data structure look like?
- What are the arguments that you are *actually* passing? i.e. What is the value? What is the data structure? Do not assume! Display the actual value on the screen!
- Now, is what you are passing the same as what you ought to be passing? If 'No', how can you fix that? If 'Yes', then it's time to go take a look at your algorithm itself...

3 Using 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]

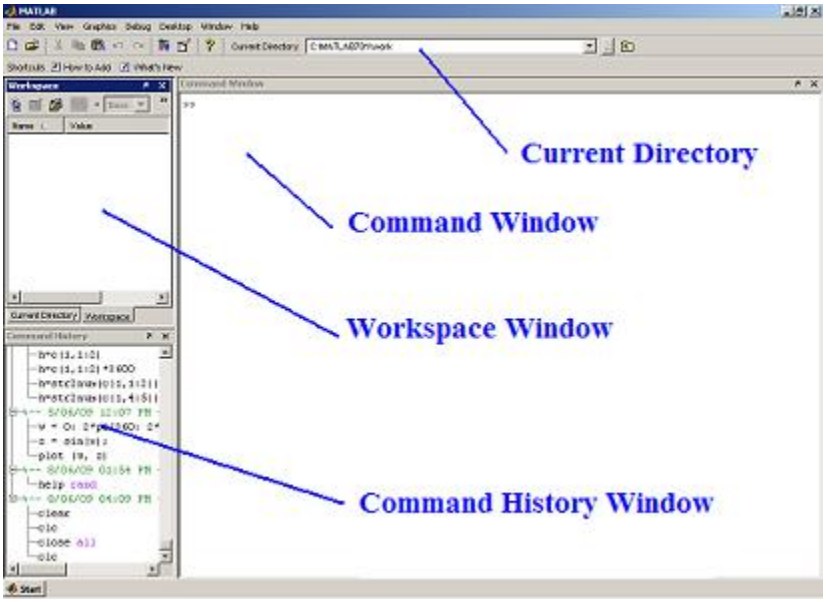

Figure 1: The MATLAB Graphical User Interface (GUI)

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

[3]Image from http://www.matrixlab-examples.com/image-files/starting_matlab.jpg

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.

> NOTE: There are several ways to use MATLAB on Linux. Typing `matlab` at the prompt will run MATLAB in X-Windows, which can be slow if done over a remote connection. To run MATLAB in just a terminal (without the GUI), type `matlab -nodisplay`. You can also request MATLAB use the current terminal for commands and use X-Windows for everything else (such as figures) by typing `matlab -nodesktop`.

4 MATLAB Commands

4.1 Help

Familiarize yourself with the `help` command, which displays the correct syntax for MATLAB functions as well as examples. Good programmers don't know everything, but they do know where to look things up! Just typing `help` (by itself) on the command line will give you a list of all help topics. Entering `help <function>` shows you the help page for a specific MATLAB function. You may need to read the `help plot` page in order to generate plots later in this lab.

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.

4.2 Matrix Operations

MATLAB is designed to operate efficiently on matrices (hence the name MATLAB = "Matrix Laboratory", which is the answer to Question 0). 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 `[ ]`. The elements within a row are separated by commas, and different rows are separated by semicolons. 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: $a = \binom{2}{4}$ and $b = \binom{3}{1}$
- 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?
Have you seen this message before, or can you find an example online?
What is the specific function, variable, or operation that is not working?
What arguments does this function/variable/operation take?
What are the arguments that you *should* be passing to this operator?
What are the arguments that you are *actually* passing?
What is the correct fix?

Exercise 3:
• Perform the following:
```
c = a + b
d = a + b;
```
• What does the `;` do?

The `size` command is extremely useful. This command can be used in several ways to find the dimensions of the matrix that MATLAB is using to represent the variable. You have already seen how to find more information about MATLAB commands, so it is assumed that you will do so as needed.

Exercise 4:
• Define `e = 17` in MATLAB. • Use `size` to find the dimensions of `a`, `b'`, and `e`?
• Use an alternate syntax for `size` to define a new variable `f` that is just the height of `b`.

It is important to remember that many commands in MATLAB can be called in multiple ways to get different results. Information about all these various syntaxes can be found together on the help page.

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 (and not 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`?
• What are the first, second, third, and last elements of each?
• What exactly do `g` and `h` contain?
• What does the syntax `v = x:y;` mean, in English?

• 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?
• What does the syntax `v = x:y:z;` mean, in English?

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 then. 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 plotting and 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. Useful for comparing plots.
- `hold off;` – Erases the current plot before drawing the next (this is the default).
- `figure` – Opens a new window on the screen. All subsequent plots will appear here.

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.

EE 235 deals with continuous-time signal processing. However, MATLAB only handles digitized representations of signals. As such, you need a way to represent time 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 $x_o(t) = te^{-|t|}$ using the following:

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

• Create the related signals $x_e(t) = |t|e^{-|t|}$ and $x(t) = 0.5 * [x_o(t) + x_e(t)]$.
• What are $x_o(t)$ and $x_e(t)$, relative to $x(t)$?
There is a name for functions with the particular symmetry that x_o and x_e have.
• Plot all three signals in one window, using `subplot`.

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 $z_1 = 1+j$. 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 $z_1 + z_2$, where $z_2 = 2e^{\frac{1}{3}j\pi}$
5. What do you expect j^j to be? Compute it. Can you explain how to derive this?

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.

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;
s = linspace(0,9,73113);
plot(s,y);
sound(y);
```

• What does `load` do?
• What does `linspace` do?
• How does `linspace` compare to the `x:y:z` operator?
• Why do you need the 73113?

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 with a 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. In this class, you should also write your lab section and email address in the file. For example:

```
% james e tetazoo, III
% tetazoo at ee dot washington dot edu
% lab section 1337
% 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 lab webpage.[4] 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:
• How can you show that the output plot of `dampedCosine.m` matches the MATLAB code?

Now you will create your own scripts.

Exercise 11:
• Create a copy of `dampedCosine.m`, and call it `dampedCosine_YourName.m`
(this keeps the files sorted next to each other in the directory, so they're easy to find).
• 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).

[4] http://cnx.org/content/m13554/latest/dampedCosine.m

Exercise 12:
• 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 the lab webpage,[5] 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 13:
• 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.
Put these plots in a separate window; use the `figure` command.
• 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 14:
• This 3-D plot is just a 2-D process in the complex plane, seen over time. Explain intuitively what the underlying 2-D process looks like.
• 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 slower" and "decay faster"?

[5] http://cnx.org/content/m13554/latest/compexp.m

Exercise 15:
- Returning to the notion of looking at the 3-D process in the complex plane:
- Are you going in a circle, spiraling inward, or out? Why?
- Are you accelerating, or are you slowing down? Why?
- What direction do you travel in (clockwise, counter-clockwise)? Why?
- Where do you start at time 0?

EOF