SIGNALS AND SYSTEMS LABORATORY 1:

Intro to MATLAB and Complex Numbers

INTRODUCTION

The goal of this lab is to acquaint you with MATLAB, in the content of complex numbers. Make this lab interactive by trying things, analyzing the outcomes, and asking questions. Then try something else based on your increased understanding.

The instructions are written for IBM PCs, which run WINDOWS NT, on the engineering network here at CSU. If you are using a computer in the Lockheed Martin Design Studio (Room B203 in the Engineering building), you will be using MATLAB 6.1.

GETTING STARTED

Create a folder on your u:\ drive to save your work. This can be done by clicking on 'Start', then going to 'Programs', and then clicking on 'Windows NT Explorer'. Once Explorer is open, click on your u:\ drive on the left-hand side of the window. Then, click on 'File', go to 'New', and select 'Folder'. A new folder will pop up on the right-hand side of the screen, where you can label the folder. Label it 'ee311'. Do not put any spaces or odd characters in your folder name, so that you can access your new folder in any version of MATLAB.

Once you have created a folder, you can start MATLAB. Click on 'Start', then 'Engineering Applications', then 'Matlab', and then MATLAB 6.1'. You should now be in the MATLAB command window.

You should see a cursor flashing just to the right of the » prompt. Assuming you named your new folder ee311, type:

```
»cd u:\ee311
```

You are now working in your ee311 folder on your u:\ drive. Type:

```
»pwd
```

This command should output u\ee311, which is the directory you are currently in. (NOTE: pwd stands for print working directory). This command is one of many that you can find on the class web page under MATLAB Operating System Commands. At this point, you may want to create another folder in your ee311 directory named 'lab_1' to help you organize your labs.

NB: Many of the commands look like UNIX commands (such as: pwd, ls, etc…) or DOS commands (dir, copy, etc).

IMAGINARY NUMBERS AND THE COMPLEX PLANE

We use two different coordinate systems to represent complex numbers. There is the Cartesian system, $z = a + jb$, and the polar system, $z = re^{j\theta}$, where $j = \sqrt{-1}$. In the Cartesian format, a is said to be the real part of z, and b is the imaginary part of z. In the polar format, r is the magnitude of z and θ is the angle of z. The complex number z can be plotted on the complex plane. The complex plane consists of a real axis (the x-axis or horizontal axis) and an imaginary axis (the y-axis or vertical axis). When a complex number z is plotted, we often draw a line from the origin to the point z on the complex plane. The length of the line is the magnitude of z and the angle that the line makes with the positive real axis of the complex plane is referred to as the angle of z. The location on the complex plane of z is defined by the Cartesian coordinate pair (a,b), or the Euler coordinate pair (r,θ).

The coordinate transformations from one coordinate system to another are

$$a = r\cos(\theta) = real(z) \qquad \text{and} \qquad r = \sqrt{a^2 + b^2} = \left(z^* z\right)^{1/2} = abs(z)$$
$$b = r\sin(\theta) = imag(z) \qquad \text{and} \qquad \theta = \arctan\left(b/a\right) = angle(z)\,.$$

The functions 'real', 'imag', 'abs', and 'angle' are MATLAB functions.

To see these ideas demonstrated, go to the web page for EE311 and click on the link 'complex_numbers_demo.m' under 'Demos for Lab1'. Save this file to your u:\ drive in the folder that you are working in (u:\ee311\lab_1, for example). Once the file is in the folder, type

```
»complex_numbers_demo
```

to run the program. After you have run the program, click on 'File', go to 'Open', and then select the file 'complex_numbers_demo.m'. Read through the file to familiarize yourself with the structure and syntax of MATLAB m-files. Notice the use of comments. It is a good idea to comment your programs liberally, so you or someone else can go back to the program later and understand what is going on.

THE EXCEPTIONAL NUMBER *e*

The next part of this lab deals with the exceptional number e = 2.718281828459045 …, which you may recognize as the base of the natural logarithm. This number comes up often in engineering, so it is important to understand where it comes from. The number e can be defined in several ways. Two of the most important are Euler's sequence and Taylor's series. Euler's sequence defines e as the limit of a sequence:

$$e = \lim_{n\to\infty}\left(1 + \frac{1}{n}\right)^n$$

This is the limit of a sequence. For n = 1, 2, 3, …, the sequence of numbers is $\left(1 + \frac{1}{n}\right)^n$ = 2, 2.25, 2.37, …, which converges to e. Taylor's series expansion for e is

$$e = \lim_{n\to\infty}\sum_{k=0}^{n}\frac{1}{k!}$$

This is called the limit of a series. For n = 1, 2, 3, …, the sequence of series is

$$\sum_k \frac{1}{k!} = 1, 1+1, 1+1+.5, \ldots = 1,\ 2,\ 2.5,\ \ldots,$$ which converges to e.

We are going to use these two definitions, along with an indirect definition to approximate e for different values of n. The indirect definition is

$$\int_1^e \frac{1}{t}dt = \ln(e) - \ln(1) = 1$$

This will be coded in MATLAB as numerical integration.

There is a demo called 'approx_e.m' that lets you enter the number of terms, n, and graph the *n-term* approximation of *e*. One of the approximations converges rapidly. Can you guess which one it is? Download 'approx_e.m' from the web page under 'Demos for Lab 1'. Run the program with different values of *n* to get a feel for how well each approximation works.

Assignment:

1. How many terms are required to have a 1% error or less in each approximation to *e*? (You should have three answers: one for the integral approximation; one for Euler's approximation; and one for the Taylor series approximation.)

THE FUNCTION e^x

The Euler and Taylor approximations of the function e^x are

$$e^x = \lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n \qquad \text{and} \qquad e^x = \lim_{n\to\infty}\sum_{k=0}^{n}\frac{x^k}{k!}$$

Assignment:

2. Without the aid of software, mathematically prove $\frac{d}{dx}e^x = e^x$ using both Euler's and Taylor's approximations.

THE FUNCTION $e^{j\theta}$ AND PLOTTING THE UNIT CIRCLE

The function $e^{j\theta}$ is defined by $e^{j\theta} = \lim_{n\to\infty}\left(1+\frac{j\theta}{n}\right)^n$ and $e^{j\theta} = \lim_{n\to\infty}\sum_{k=0}^{n}\frac{(j\theta)^k}{k!}$, with θ defined in radians, and NOT degrees. Euler's Identity is $e^{j\theta} = \cos\theta + j\sin\theta$. If you were to graph $e^{j\theta}$ vs. $0 < \theta \le 2\pi$ on the complex plane, you would have two functions oscillating between –1 and +1. The cosine function is oscillating on the real axis and the sine function is oscillating on the imaginary axis. The result, if you sweep θ through a range of 2π, is the unit circle. This is demonstrated in the demo 'eulers_id.m' located under 'Demos for Lab 1' on the web page. You may notice that some fancy tricks were used to format the output of the labels on the graph. Learn this nice printing format so you, too, can produce pretty graphs.

Assignment:

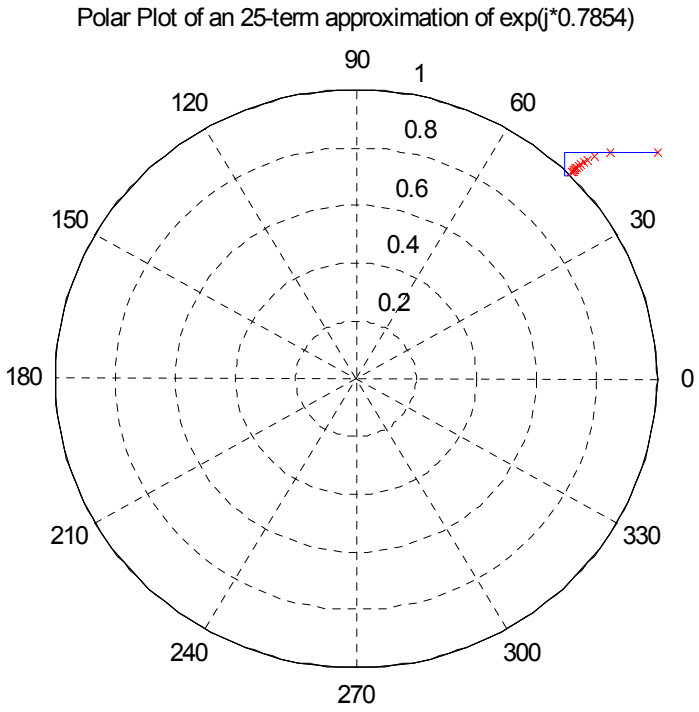
3. Using Euler's Identity, $e^{j\theta} = \cos\theta + j\sin\theta$, and the Taylor Series expansion,

   $e^{j\theta} = \lim_{n\to\infty}\sum_{k=0}^{n}\frac{(j\theta)^k}{k!}$, find Taylor series expansions for $\cos(\theta)$ and $\sin(\theta)$.

   NOTE: Do this problem mathematically and without the aid of software (i.e. without using MATLAB, Maple, Mathcad, etc.)

Assignment:

4. Download and run the program 'app_expj.m' from 'Demos for Lab 1' on the web page to demonstrate how the Euler and Taylor approximations converge to $e^{j\theta}$. For 25 terms approximation of θ, you should see this sequence approximations to $e^{j\pi/4}$, which is a unit vector at angle $\pi/4$:



Explain the path that each approximation takes. Why is one arc-like and the other a sequence of horizontal and vertical refinements? Use the zoom tool in MATLAB to see what is really going on. Use what you have learned about complex numbers and the two approximations to justify your answer.

ROOTS OF A QUADRATIC EQUATION

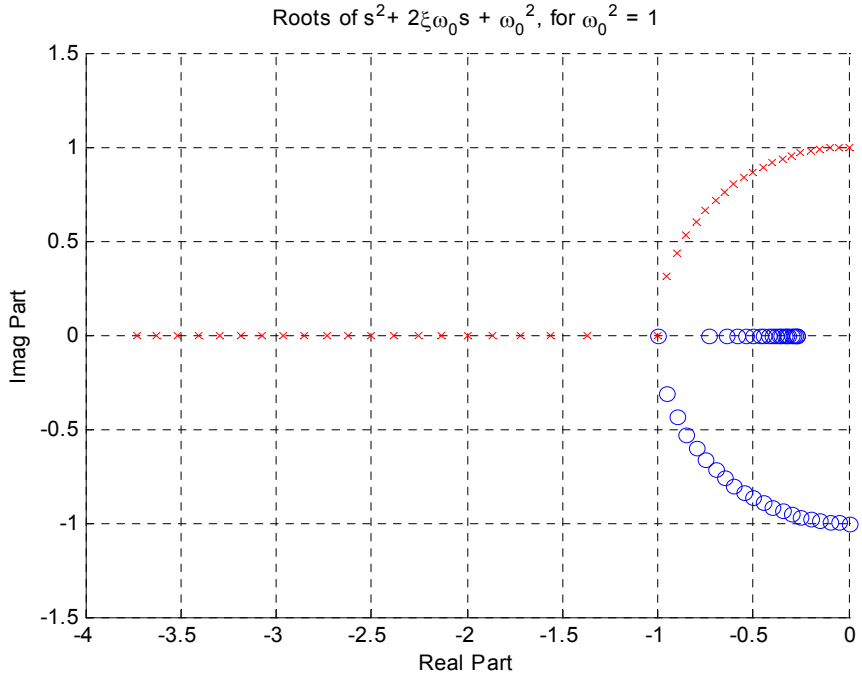
In our study of second-order *linear time-invariant* (LTI) systems, we will encounter second-order polynomials of the form

$$s^2 + 2\xi\omega_0 s + \omega_0^2 = 0,$$

where s is the Laplace transform variable, $\xi \geq 0$ is the damping factor and ω_0 is the undamped resonant frequency.

Assignment:

5. Complete the square in the equation $s^2 + 2\xi\omega_0 s + \omega_0^2 = 0$ to show that the solutions are $s_{1,2} = -\xi\omega_0 \pm j\omega_0\sqrt{1-\xi^2}$. Note that for $|\xi| > 1$, the roots are real, for $|\xi| = 1$, they are repeated, and for $|\xi| < 1$, they are complex conjugates. In linear systems these cases are called over-damped, critically damped, and under-damped, respectively. Download 'rootdemo.m' under 'Demos for Lab 1' on the web page to see the locus of roots for $\omega_0 = 1$. You should see something like this:



Now re-do the loop for ξ to make it go from –4 to +4 in steps of 0.2. Use '**help**' and '**type**' commands to figure out what 'rootdemo.m' does. (HINT: In order to see your new results, change the line that says axis([-4 0 -1.5 1.5]); to axis([-4 4 -1.5 1.5]);) Include a printout of your results.

FUNCTIONS

Functions are an important part of any program. Whenever you have a block of code that you want to re-use, it is a good idea to make a function out of that piece of code. Depending on the type of function, you may pass variables into and retrieve information out of the function. You have already seen a few of MATLAB's built-in functions such as plot(x,y), exp(x), disp('*text*'), real(z), and imag(z). To display the syntax of a function, type:

```
>> type angle

function p = angle(h)
%ANGLE  Phase angle.
%   ANGLE(H) returns the phase angles, in radians, of a matrix with
%   complex elements.

p = atan2(imag(h), real(h));
```

Some of the extra comments were stripped away, so you are encouraged to actually use the type command yourself. Using the 'type' command, you can see most of the functions in MATLAB, but some of them are built-in and invisible.

Notice that in the function definition, the first line of the function is p=angle(h). This tells MATLAB that, when the function is called using ang=angle(z), there is a matrix, or vector, named z to be passed in and temporarily stored in h. Then, whatever is stored in p will be returned by the function and stored in the variable ang in the program that called the function angle. Be sure that you understand how functions work. You will be using and creating many in this course.

END NOTES

Please refer to the website for lab write-up format.

Also, the following m-files are located on the web page under 'Lab 1'. These files are for your benefit. Please do not turn in any of the results from these exercises or demos.

To get a better understanding of complex numbers and functions, try the following extra exercises:

Powers of Complex Numbers:	'cn_powers.m'
Perpendicular Complex Numbers:	'perpendicular_cn.m'
The 'fac' function:	'fac.m'

For your amusement, run the following extra demos:

Implicit and Explicit 'for' loops:	'circle_demo.m'
Polar Plots:	'polar_plots_cart.m' 'polar_plots_polar.m' 'polar_plots_both.m' 'polar_plots_movie_cart.m' 'polar_plots_movie_polar.m' 'polar_plots_movie_both.m'
Using the 'plot' Command:	'rainbow_simple.m' 'rainbow_fancy.m' 'rainbow_arch.m'
Complex Numbers Using Euler's Identity:	'cn_eulers_id.m'