1.1 Overview

MATLAB will be used extensively in all the labs. The primary goal of this lab is to familiarize yourself with using MATLAB. Please read Appendix B: *M-file Programming in* MATLAB for an overview. Here are three specific goals for this lab:

1. Learn basic MATLAB commands and syntax, including the `help` system.
2. Write and edit your own script files in MATLAB, and run them as commands.
3. Learn a little about advanced programming techniques for MATLAB, i.e., vectorization.

1.2 Getting Started

After logging in, you can start MATLAB by double-clicking on a MATLAB icon, typing `matlab` in a terminal window, or by selecting MATLAB from a menu such as the `START` menu under Windows. The following steps will introduce you to MATLAB.

(a) Run the MATLAB help desk by typing `helpdesk`. The help desk provides a hypertext interface to the MATLAB documentation. Two links of interest are **Getting Help** and **Getting Started with MATLAB**.

(b) One of the tabs in the Help Desk is called Demos, where you can see information on getting started and some other basics of MATLAB.

(c) Explore the MATLAB help capability available at the command line. Try the following:

```
help
help edit
help plot
help colon        %<--- a VERY IMPORTANT notation
help ops
help zeros
help ones
lookfor filter      %<--- keyword search
```

Note: it is possible to force MATLAB to display only one screen-full of information at once by issuing the command `more on`).

(d) **control-C** will stop the execution of any MATLAB command. For example, using **ctl-C** while `lookfor` is running will force it to stop and print out all the results it has found so far.

1.3 Calculate and Plot

(a) Use MATLAB as a calculator. Try the following:

```
pi*pi - 10
sin(pi/4)
ans ^ 2       %<--- "ans" holds the last result
```

(b) Do variable name assignment in MATLAB. Try the following:

```
x = sin( pi/5 );
cos( pi/5 )       %<--- assigned to what?
y = sqrt( 1 - x*x )
ans                 %<--- what value is contained in ans?
```

(c) Complex numbers are natural in MATLAB. The basic operations are supported. Try the following:

```
z = 3 + 4i, w = -3 + 4j
real(z), imag(z)
abs([z,w])      %<-- Vector (or Matrix) constructor
conj(z+w)
angle(z)
exp( j*pi )
exp(j*[ pi/4, 0, -pi/4 ])
```

(d) Plotting is easy in MATLAB for both real and complex numbers. The basic plot command will plot a vector `y` versus a vector `x`. Try the following:

```
x = [-3 -1 0 1 3 ];
y = x.*x - 3*x;
plot( x, y )
z = x + y*sqrt(-1)
plot( z )    %<---- complex values: plot imag vs. real
```

Use `help arith` to learn how the operation `xx.*xx` works when `xx` is a vector; compare array multiplication (`dot-star`) to matrix multiplication. When unsure about a command, use `help`.

2.1 MATLAB Array Indexing

(a) Make sure that you understand the **colon** notation. In particular, explain in words what each of the following MATLAB statements will produce (array length and values):

```
jkl =  0 : 6
nnn =  2 : 4 : 17
mmm =  99 : -1 : 88
ttt =  2 : (1/9) : 4    %<--- How many elements in the ttt vector?
tpi = pi * [ 0:0.1:2 ];
```

(b) Extracting and/or inserting numbers in a vector is very easy to do. Consider the definition of `xx` in the first line:

```
xx = [ zeros(1,3), linspace(0,1,5), ones(1,4) ]
xx(4:6)
size(xx)
length(xx)
xx(2:2:length(xx))
```

Explain the results echoed from each of the last four lines of the above code.

(c) Observe the result of the following two assignments, where `xx` was defined in part (b):

```
xx = [ zeros(1,3), linspace(0,1,5), ones(1,4) ];
yy = xx;
yy(4:6) = exp(1)*(1:3)
```

Now write a statement that will take the vector `xx` and replace the even indexed elements (i.e., `xx(2)`, `xx(4)`, etc) with the constant π^e. *Use a vector replacement, not a loop.*

2.2 MATLAB Array Operations

There are two kinds of multiplication in MATLAB: matrix multiplication and *array* multiplication. Many other operations such as division and exponentiation also have this dual character. MATLAB uses a period to change the behavior from a matrix operator to an array operator, e.g., dot-star ($.*$) instead star ($*$) for multiplication.

(a) The default is matrix multiplication when the "$*$" symbol is used; execute the following:

```
AA = [ 1, 2; 3, 4; 5, 6; 7, 8]
BB = [ 1, 2, 3; 4, 5, 6]
CC = AA * BB
DD = BB * AA
```

Explain why one of the multiplications fails. In the case where the multiplication succeeds, explain what the dimensions of the resulting product matrix will be.

(b) In DSP it is often useful to perform an *element-by-element* multiplication, which we will call array multiplication. For example, to multiply one sinusoid by another we would use the ".$*$" operator:

```
nn = 1:9;
qq1 = 7*cos(0.1*pi*nn-pi/2);  qq2 = cos(pi*nn);
qq = qq1 .* qq2
zz = qq1 * qq2
```

Explain why the multiplication that attempts to create `zz` fails. Notice that the dimensions of the array-product matrix `qq` are the same as the dimensions of `nn`. Explain the values in the vector `qq2`.

2.3 MATLAB Script Files

(a) Experiment with vectors in MATLAB. Think of the vector as a list of numbers. Try the following:

```
xk = cos( pi*(0:11)/4 );  %<---comment: compute cosines
```

How many values of the cosine are stored in the vector `xk`? What is `xk(1)`? Is `xk(0)` defined? *Notes:* the semicolon at the end of a statement will suppress the echo to the screen. The text following the `%` is a comment; it may be omitted.

(b) (A taste of vectorization) Loops can be written in MATLAB, but they are NOT the most efficient way to get things done. It's better to **always avoid loops** and use the colon notation instead. The following code has a loop that computes values of the cosine function. *Note:* the index of `cc()` must start at 1.

```
cc = [ ];      %<--- initialize the cc vector to be empty
for k=-50:50
   cc(k+51) = cos( pi*k/30 );
end
plot(cc)
```

Explain why it is necessary to write `cc(k+51)` inside the loop. What happens if you use `cc(k)` instead? Also, explain the labels on the x-axis of the plot.

(c) **Rewrite the computation in the previous without using the loop** by following the style in part (a). Two MATLAB statements plus the `plot` command should suffice.

(d) Use the built-in MATLAB editor to create a script file[1] called `mylab0.m` containing the following lines:

```
tt = -5 : 0.01 : 10;
xx = cos( 0.5*pi*tt );
zz = 0.6*exp(-j*pi/4)*exp(j*0.5*pi*tt);
%
%<-- plot the real part, which is a sinusoid
plot( tt, xx, 'b-', tt, real(zz), 'r--' ), grid on
title('Test Plot of a TWO sinusoids')
xlabel('Time (sec)')
```

Explain why the plot of `real(zz)` is a sinusoid. Determine its phase (φ) and amplitude (A) from

[1] Use the MATLAB command `addpath()` to allow MATLAB to "see" your personal directory (usually the `C:` drive).

its definition. Make a calculation of the phase from a time-shift measured on the plot; zoom in to measure a peak location very accurately. Compare this value to the expected value from the formula that defines zz.

(e) Run your script from MATLAB. To run the file mylab0 that you created previously, try

```
mylab0              %<---will run the commands in the file
type mylab0         %<---will type out the contents of
                    %    mylab0.m to the screen
```