DSP First

Laboratory Exercise #1

Introduction to MATLAB

The Warm-up section of each lab should be completed during a supervised lab session and the laboratory instructor should verify the appropriate steps by initialing on the **Instructor Verification** line. An instructor verification page can be found at the end of this lab.

1 Overview and Goals

MATLAB will be used extensively in all the succeeding labs. The goal of this first lab is to gain familiarity with MATLAB and build some basic skills in the MATLAB language. Please read the material on *Using* MATLAB in the text for a more complete overview. If you desire an in-depth presentation that covers most of the language, consult the MATLAB reference manual which is also available on-line with a web-browser interface.

There are several specific objectives in this lab:

1. Learn to use the help system to study basic MATLAB commands and syntax.
2. Write your own functions and `m-files`'s in MATLAB.
3. Learn some advanced programming techniques for MATLAB, i.e., vectorization.

2 Warm-up

Each lab will start with a warm-up section like this one. The warm-up usually consists of a few simple exercises to introduce MATLAB commands. Since this is the first lab, make sure that you have installed the *DSP First Toolbox* for MATLAB. Refer to the instructions on the *DSP First CD-ROM.* When you start MATLAB on your computer, the path should include a directory called `dspfirst`.

2.1 Basic Commands

The following exercises will begin your orientation in MATLAB.

(a) View the MATLAB introduction by typing `intro` at the MATLAB prompt. This short introduction will demonstrate some of the basics of using MATLAB.

(b) Explore the MATLAB help capability. Type each of the following lines to read about these commands:

```
help
help plot
help colon
help ops
help zeros
help ones
lookfor filter        %<--- keyword search
```

If the lines scroll past the bottom of the screen, it is possible to force MATLAB to display only one screen-full of information at a time by issuing the command `more on`.

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

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

(d) Variable names can store values and matrices in MATLAB. Try the following:

```
xx = sin( pi/5 );
cos( pi/5 )       %<--- assigned to what?
yy = sqrt( 1 - xx*xx )
ans
```

(e) Complex numbers are natural in MATLAB. Notice that the names of some basic operations are unexpected, e.g., `abs` for magnitude. Try the following:

```
zz = 3 + 4i
conj(zz)
abs(zz)
angle(zz)
real(zz)
imag(zz)
help zprint           %- requires DSP First Toolbox
exp( sqrt(-1)*pi )
exp(j*[ pi/4 -pi/4 ])
```

(f) Plotting is easy in MATLAB, for both real and complex numbers. The basic plot command will plot a vector `yy` versus a vector `xx`. Try the following:

```
xx = [-3 -1 0 1 3 ];
yy = xx.*xx - 3*xx;
plot( xx, yy )
zz = xx + yy*sqrt(-1);
plot( zz )        %<---- complex values can be plotted
```

Drop the semicolons, if you want to display the values in the `x`, `y`, and `z` vectors. Use `help arith` to learn how the operation `xx.*xx` works; compare to matrix multiply.

When unsure about a command, use `help`.

2.2 MATLAB Array Indexing

(a) Make sure that you understand the colon notation. In particular, explain what the following MATLAB code will produce

```
jkl = 2 :  4 :  17
jkl = 99 :  -1 :  88
ttt = 2 :  (1/9) :  4
tpi = pi * [ 2 :  (-1/9) :  0 ]
```

(b) Extracting and/or inserting numbers in a vector is very easy to do. Consider the following definition:

```
xx = [ ones(1,4), [2:2:11], zeros(1,3) ]
xx(3:7)
length(xx)
xx(2:2:length(xx))
```

Explain the result echoed from the last three lines of the above code.

(c) In the previous part, the vector `x` contains 12 elements. Observe the result of the following assignment:

```
xx(3:7) = pi*(1:5)
```

Now write a statement that will replace the odd-indexed elements of `xx` with the constant -77 (i.e., `xx(1)`, `xx(3)`, etc). *Use a vector indexing and vector replacement.*

Instructor Verification (separate page)

2.3 MATLAB Script Files

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

```
kset = -3:11;
kset
cos( pi*kset/4 )      %<---comment:  compute cosines
```

Explain how the last example computes the different values of cosine without a loop. The text following the `%` is a comment; it may be omitted. If you remove the semicolon at the end of the first statement, all the elements of `kset` will be echoed to the screen.

(b) *Vectorization* is an essential programming skill in MATLAB. Loops can be done in MATLAB, but they are *not* the most efficient way to get things done. It's better to *avoid loops and use the vector notation instead.* For example, the code below uses a loop to compute values of the sine function. Rewrite this computation without using the loop (as in the previous part).

```
xx = [ ];   %<--- initialize the x vector to a null
for k=0:7
   xx(k+1) = sin( k*pi/4 )    %-- xx(0) would fail
end
x
```

(c) Use the built-in MATLAB editor (on Windows or a Mac), or an external one such as `emacs` or `notepad` (on UNIX or DOS), to create a script file called `funky.m` containing the following lines:

```
tt = -2 :  0.05 :  3;
xx = sin( 2*pi*0.789*tt );
plot( tt, xx ), grid on      %<--- plot a sinusoid
title('TEST PLOT of SINUSOID')
xlabel('TIME (sec)')
```

(d) Run your script from MATLAB. To run the file `funky` that you created in part (c), try

```
funky          %<---will run the commands in the file
type funky     %<---will type out the contents of
               %    funky.m to the screen
which funky    %<---will show directory containing funky.m
```

(e) Add three lines of code to your script, so that it will plot a cosine on top of the sine. Use the `hold` function to add a plot of

```
0.5*cos( 2*pi*0.789*tt )
```

to the plot created in part (c). See `help hold` in MATLAB.

Instructor Verification (separate page)

2.4 MATLAB Demos

There are many demonstration files in MATLAB. Run the MATLAB demos from a menu by typing `demo`, and explore some of the different demos of basic MATLAB commands and plots.

When unsure about a command, use `help`.

2.5 MATLAB Sound

(a) Run the MATLAB sound demo by typing `xpsound` at the MATLAB prompt. If you are unable to hear the sounds in the MATLAB demo then check the sound hardware on your machine. Since there are so many variations in the types of sound hardware on different computers, this may require consultation with an expert in system configuration and/or MATLAB installation.

(b) Now generate a tone (i.e., a sinusoid) in MATLAB and listen to it with the `sound` command. The frequency of your tone should be 2 KHz and the duration should be 1 sec. The following lines of code should be saved in a file called `mysound.m` and run from the command line.

```
dur = 1.0;
fs = 8000;
tt = 0 :  (1/fs) :  dur;
xx = sin( 2*pi*2000*tt );
sound( xx, fs )
```

The sound hardware will convert the vector of numbers `xx` into a sound waveform at a certain rate, called the sampling rate. In this case, the sampling rate is 8000 samples/second, but other values might be used depending on the capability of sound hardware. What is the length of the vector `xx`? Read the online `help` for `sound` (or `soundsc`) to get more information on using this command.

Instructor Verification (separate page)

2.6 Functions

The following warm-up exercises are to help you in writing functions in MATLAB. Although these examples contain minor errors, they do exemplify the correct structure and syntax for writing functions.

(a) Find the mistake in the following function:

```
function  xx = cosgen(f,dur)
%COSGEN  Function to generate a cosine wave
%  usage:
%      xx = cosgen(f,dur)
%       f = desired frequency
%     dur = duration of the waveform in seconds
%
tt = [0:1/(20*f):dur];  % gives 20 samples per period
yy = cos(2*pi*f*tt);
```

(b) Find the mistake in the following function:

```
function  [sum,prod] = sumprod(x1,x2)
%SUMPROD  Function to add and multiply two complex numbers
%  usage:
%       [sum,prod] = sumprod(x1,x2)
%           x1 = a complex number
%           x2 = another complex number
%          sum = sum of x1 and x2
%         prod = product of x1 and x2
%
sum = z1+z2;
prod = z1*z2;
```

(c) Explain the following lines of MATLAB code:

```
yy = ones(7,1) * rand(1,4);

xx = randn(1,3);
yy = xx(ones(6,1),:);
```

(d) Write a function that performs the same task as the following without using a `for` loop. Consult the matrix multiplication section in the appendix on *Using* MATLAB for some clever solutions.

```
function Z = expand(xx,ncol)
%EXPAND  Function to generate a matrix Z with identical columns
%         equal to an input vector xx
%  usage:
%       Z = expand(xx,ncol)
%      xx = the input vector containing one column for Z
%    ncol = the number of desired columns
%
xx = xx(:);  %-- makes the input vector x into a column vector
Z = zeros(length(xx),ncol);
for i=1:ncol
   Z(:,i) = xx;
end
```

2.7 Vectorization

(a) Explain the following lines of MATLAB code:

```
A = randn(6,3);
A = A .* (A>0);
```

(b) Write a new function that performs the same task as the following function without using a `for` loop. Use the idea in part (a) and also consult section on vector logicals in the *Using* MATLAB section of the appendix. In addition, the MATLAB logical operators are summarized via `help relop`.

```
function  Z = replacez(A)
%REPLACEZ  Function that replaces the negative elements
%          of a matrix with the number 77
%   usage:
%        Z = replacez(A)
%        A = input matrix whose negative elements are to
%            be replaced with 77
%
[M,N] = size(A);
for i=1:M
   for j=1:N
      if A(i,j) < 0
         Z(i,j) = 77;
      else
         Z(i,j) = A(i,j);
      end
   end
end
```

Instructor Verification (separate page)

3 Exercises: Using MATLAB

The following exercise can be completed at your convenience. Results from each part should be included in a brief lab report write-up.

3.1 Manipulating Sinusoids with MATLAB

Generate two 3000 hertz sinusoids with different amplitudes and phases.

$$x_1(t) = A_1 \cos(2\pi(3000)t + \phi_1) \qquad x_2(t) = A_2 \cos(2\pi(3000)t + \phi_2)$$

(a) Select the value of the amplitudes as follows: let $A_1 = 13$ and use your age for A_2. For the phases, use the last two digits of your telephone number for ϕ_1 (in degrees), and take $\phi_2 = -30°$. *When doing computations in* MATLAB*, make sure to convert degrees to radians.*

(b) Make a plot of both signals over a range of t that will exhibit approximately 3 cycles. Make sure the plot starts at a negative time so that it will include $t = 0$, *and make sure that your have at least 20 samples per period of the wave.*

(c) Verify that the phase of the two signals $x_1(t)$ and $x_2(t)$ is correct at $t = 0$, and also verify that each one has the correct maximum amplitude.

(d) Use `subplot(3,1,1)` and `subplot(3,1,2)` to make a three-panel subplot that puts both of these plots on the same window. See `help subplot`.

(e) Create a third sinusoid as the sum: $x_3(t) = x_1(t) + x_2(t)$. In MATLAB this amounts to summing the vectors that hold the samples of each sinusoid. Make a plot of $x_3(t)$ over the same range of time as used in the previous two plots. Include this as the third panel in the window by using `subplot(3,1,3)`.

(f) Measure the magnitude and phase of $x_3(t)$ directly from the plot. In your lab report, explain how the magnitude and phase were measured by making annotations on each of the plots.

4 Lab Review Questions

In general, your lab write-up should indicate that you have acquired a better understanding of the topics treated by the laboratory assignment. Here are a few questions for you to answer in order to assess your understanding of this lab's objective: a working knowledge of the basics of MATLAB. If you do not know the answers to these questions go back to the lab and try to figure them out in MATLAB (remember the commands `help` and `lookfor`).

1. You saw how it easy it is for MATLAB to generate and manipulate vectors (i.e., 1-dimensional arrays of numbers). For example, consider the following:

```
yy = 0:10;
yy = zeros(1,25);
yy = 1:.25:5;
```

   (a) How would you modify one of the above lines of MATLAB code to create a vector that steps from 0 to 10 in steps of $\frac{1}{2}$?

   (b) How would you modify one of the lines in the code to create a vector of one hundred 100's?

2. You also learned that MATLAB has no problem handling complex numbers . Consider the following line of code:

$$\texttt{yy} = 3 + 5\texttt{j};$$

   (a) How do you get MATLAB to return the magnitude of the complex number `yy`?

   (b) How do you get MATLAB to return the phase of the complex number `yy`? What are the units of the answer?

3. In Section 2.3, you learned that multiple lines of MATLAB code can be stored in a file with a `.m` extension. MATLAB then executes the code in the order that it appears in the file. Consider the following file, named `example.m`:

```
f = 200;
tt = [0:1/(20*f):1];
z = exp(j*2*pi*f*tt);
subplot(211)
plot(real(z))
title('Real part of exp(j*2*pi*200*tt)')
subplot(212)
plot(imag(z))
title('Imaginary part of exp(j*2*pi*200*tt)')
```

(a) How do you execute the file from the MATLAB prompt?

(b) Suppose the file were named `example.dog`. Would it run? How could you change it to make it work in MATLAB?

(c) Assuming the M-file runs, what do you expect the plots to look like? If you're not sure type in the code and run it.

Lab 1
Instructor Verification Sheet

Staple this page to the end of your Lab Report.

Name: _____ Date: _____

Part 2.2 Vector replacement using the colon operator:

Verified: _____

Part 2.3 Run the modified function `funky` from a file:

Verified: _____

Part 2.5 Use `sound` to play a 2 kHz tone in MATLAB:

Verified: _____

Part 2.7 Modify `replacez` using vector logicals:

Verified: _____