Laboratory # 1

An Introduction to Matlab

This first laboratory is primarily based around a piece of sample code, `sample_function.m`. This code can be found as an appendix to this lab, and can also be downloaded from the course webpage at `http:\www.eecs.umich.edu\courses\eecs206`. In this lab, you will explore `sample_function.m` and start to become comfortable with some very important Matlab tasks. All line numbers in this laboratory refer to lines in `sample_function.m`.

Before you begin this laboratory, you should read the Matlab tutorial that can be found on the course webpage then Appendix B of the text. The first is a basic introduction to Matlab, while the second introduces some advanced topics in Matlab programming. Also, you are responsible for understanding the code that is contained in `sample_function.m`. Make sure that you execute the function and follow along with what it does. If necessary, step through the code line-by-line using the debugger. Use Matlab's `help` when you encounter a command you are not familiar with. When you are finished, you should be able to explain the behavior of every line in this function.

The first tasks in this lab will exercise your debugging skills in Matlab. You should use the command `pause off` before beginning to debug to prevent the file from stopping at the `pause` commands while you are debugging it. `pause on` will re-enable pausing.

1. [8] Immediately after the execution of line #195,
   (a) What are the dimensions of `envelope` (i.e., how many rows and columns does it have)?
   (b) What are the dimensions of `y`?
   (c) What are the dimensions of `y_fade`?
   (d) What number is stored in the 50^{th} element of `y_fade`?
2. [2] What is stored in `b` after line #309?
3. [2] What is stored in `b` after line #310?
4. [6] In the vicinity of lines #233 through 239 (the first clip routine),
   (a) What is the value of `counter` when the positive threshold is first exceeded?
   (b) What is the value of `counter` when the negative threshold is first exceeded?

Next, you'll provide some explanations of portions of `sample_function.m`. If you're not sure what a particular section of code does, remember to check the `help` of commands you aren't familiar with and examine intermediate variables until you have determined what it does.

5. [8] Look at lines #37 and #43. Each of these lines has two separate conditions.

(a) Explain in detail what these lines of code are doing. When will the code inside the `if` statements be executed?

(b) What would happen if `isempty` came before `~exist`? (Hint: try it) Why do you think this might happen?

6. [8] Explain the operation of lines #252 and #253.

7. [8] Explain what happens if you take line #326 and modify it so that you call `all` on a matrix only once. You might want to try this command on a larger matrix. (Note: most MATLAB operators that operate on matrices work in the same way. This includes commands like `min` and `max`, `mean` and `sum`, and a host of others.)

8. [8] Suppose that `c` is a 5x7 matrix. What *single* command will return just the four corners of the matrix? (Hint: there are several ways to do this. Look at lines #82 through 86).

Now, consider the following code:

```
x = [];
for i = 1:40
    x = [x; i/40];
end
```

9. [8] What happens to `x` as the `for` loop executes?

10. [8] Give a single command that produces the same `x` as these four lines of code.

Finally, you'll write some simple MATLAB code to work with some signals.

11. [14] Create two sinusoidal signals. The first should have an amplitude (i.e., a maximum excursion from zero) of 2 and a phase of -$\frac{\pi}{3}$. The second should have an amplitude of 3 and a phase of $\frac{5\pi}{6}$. Let your time axis be `0:0.001:0.1`.

    (a) Set the frequencies of the two signals so that they each contain three periods. (The two signals should have the same frequency). What frequency did you use?

    (b) Create a third signal by summing your two sinusoids.

    (c) Using `subplot`, plot all three signals on the same figure.

    (d) What are the amplitude, frequency, and phase of the sum signal?

12. [12] Load `handel.mat`, as is done in the sample code. Recall that the signal itself is stored in the variable `y`. Suppose that we want to get an estimate of the large-scape amplitude of this signal. We can do this simply by considering the signal to be made up of blocks of n samples. For each block of samples, simply find the minimum and maximum values in that block and store the results in two separate arrays.

    (a) Do this for the signal `y` that you loaded from `handel.mat`. Use a block size of 100 samples. Call your arrays `min_v` and `max_v`.

    (b) Plot the signal `y`.

    (c) On the same plot (use `hold on`), plot your two arrays. Since your arrays are only $1/100^{th}$ of the size of `y`, use `(1:length(max_v))*100` as your time axis. The result should roughly outline the signal `y`. Make sure that you zoom in on a portion of the signal so that we can see both the positive and negative envelopes.

13. [8] Use `subplot` to create a figure with two subplots. In the first, plot the function $y = \sqrt{x}$. In the second, plot $y = \ln x$. (MATLAB uses `log` for the natural logarithm). Recall that these functions are only properly defined for positive values of x.

Appendix: sample_function.m

```
function [result] = sample_function(num_in, string_in)

%function [result] = sample_function(num_in, string_in)
%
% SAMPLE_FUNCTION:  A MATLAB function that illustrates the use of various
%                MATLAB commands
%
% Input Parameters:
%    num_in:     optional numeric parameter (default: 42)
%    string_in:  optional string parameter (default: 'Slime mold')
%
% Output Parameters:
%    result:    output based on the two inputs

% Notice the comment block above.  When you type 'help funtion_name',
% the first comment block in the file will be printed.  There are
% various styles for header comments, but they should always contain a
% description of what the code does and descriptions of the input and
% output parameters.

% We'll close all of the figures, but this can be annoying if you want your
% figures to stick around.  In general, I wouldn't recommend doing this in
% your functions.
close all

% If you want to hear the sound demonstrations in this function, make sure you
% have headphones and set 'play_sounds' to 1.  On a UNIX system in the CAEN labs, you
% need to run /usr/demo/SOUND/gaintool and select headphone output; otherwise, you'll
% disturb the other users in the lab.
play_sounds = 0;

% This following piece of code performs input parameter checking.  If either
% of the parameters is missing or empty ('[]'), these IF statements will
% set default values.  This is especially useful if you have a long parameter
% list, but you only want to set the value of one of them.

if ~exist('num_in','var') | isempty(num_in)
    % The '~' and '|' are logical connectors, meaning NOT and OR, respectively.
    %  AND is '&'.
    num_in = 42;
end

if ~exist('string_in','var') | isempty(string_in)
    string_in = 'Slime mold';
end

% It is also a good idea to check the size of your input parameters so that
% you aren't given a vector or matrix when you expect a scalar.

if any(size(num_in) > 1)
    error('num_in must be a scalar!');
end
```

```
if ~ischar(string_in)
    error('string_in must be a character array!');
end


% First, some basic MATLAB skills.  You should be able to explain in detail
% what each of these commands is doing.

row_vector1 = [1 3 5 7 9 11 13 15]  % Horizontal concatenation
row_vector2 = 1:2:15                % Colon operator makes row vectors
row_vector3 = linspace(1,15,8)      % --> linspace(start,end,# elements)
disp('Hit a key to continue.');
pause

col_vector1 = [6 4 2 0 -2 -4]'      % ' performs transposition
col_vector2 = [6; 4; 2; 0; -2; -4]  % Vertical concatenation
col_vector3 = (6:-2:-4)'            % Transposing the colon operator
disp('Hit a key to continue.');
pause

M = ones(2,3)       % Two rows, three columns (i.e., 2x3)
N = zeros(3,4)      % Three rows, four columns (i.e., 3x4)
A = eye(3)          % 3x3 square matrix
B = [1 2 3; 4 5 6; 7 8 9; 10 11 12]  % 3x4 matrix
disp('Hit a key to continue.');
pause

% Indexing:
B(2,3)              % Item at second row, third column
B([2 3],[3 1])      % Second and third rows, third and first columns
B(3,:)              % Third row, all columns
B(3)                % 3rd element in the order (1,1), (2,1), (3,1), (1,2), ...
B(6:end)            % All elements from #6 to the end
disp('Hit a key to continue');
pause

B(4,2) = 100                % Assigning to a single element
B(6) = -num_in
B(2:end,3) = [21; 24; 27]   % Left and right hand sides must be the same size!
B([1 3],:) = zeros(2,3)
disp('Hit a key to continue');
pause

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% To use an output parameter, we just assign a value to it.  Our output
% will be a string containing the two input parameters.  Notice that
% a string is simply an array of characters.  We can concatenate two strings
% the same way we concatenate arrays.  'num2str' converts a number into its
% string representation.

result = [ '"' string_in '" was your string. ' num2str(num_in) ' was your number.'];
disp(result);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Suppose we want to make a sinusoid with amplitude equal to 'num_in'.
% In order to represent a signal (like a sinusoid) on a computer, we need
% to "sample" it.  That is, we store a number of equally-spaced values of
% the signal every second.  Thus, we first need to define a time axis:

fs = 8192;
t = 0:1/fs:2;

% This gives us two seconds worth of a time axis at a sampling rate of 8192
% samples per second.  Then:

frq = 300;     % In hertz
phase = pi/4;  % In radians
cos_wave = cos(2*pi*t*frq + phase);

% And now, we plot.  Make sure you zoom in to see the sinusoid.

figure(1);              % Creates or activates figure #1
plot(t,cos_wave);
title('A simple cosine');
xlabel('Time (s)');
ylabel('Amplitude');
zoom on
disp('Hit a key to continue.');
pause

% We can listen to the sound as well:
if play_sounds
    soundsc(cos_wave,fs);
end

% Suppose we only want to plot the first one hundred samples so that we can clearly
% see the sinusoid and its phase.  We can do this:

plot(t(1:100),cos_wave(1:100),'k:o');
title('Fewer samples of a simple cosine');
disp('Hit a key to continue.');
pause

% The third parameter to 'plot' is a line-style specifier.  It is optional, but
% it can be very useful.  There are other styles of plots that we can use, too:

subplot(3,1,1);  % In a matrix of figures (3 rows, 1 column), select the 1st one
stem(t(1:100),cos_wave(1:100));
axis tight
subplot(3,1,2);
bar(t(1:100),cos_wave(1:100));
axis tight
subplot(3,1,3);
stairs(t(1:100),cos_wave(1:100));
axis tight
```

```
disp('Hit a key to continue.');
pause
subplot(1,1,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Now, let's load a built-in signal from MATLAB.  The variables stored in
% this file are 'y' (the signal) and 'Fs' (the sampling frequency).  Note that
% variable names are case sensitive!
load handel;

% Now, we plot and play.  The first parameter of 'plot' shows another way to
% build up a time axis.  Zoom in on this one, too.

plot(linspace(0,length(y)/Fs,length(y)),y);
title('Handel''s Halleluia Chorus');
xlabel('Amplitude');
ylabel('Time (s)');
zoom on;
if play_sounds
    soundsc(y,Fs);
end
disp('Hit a key to continue.');
pause

% Suppose we want this sound to fade in and fade out rather than starting
% and ending suddenly.  We can accomplish this by multiplying the signal by an
% envelope.  Notice that we need to use the .* operator to accomplish this.
% We build our envelope by horizontally concatenating three row vectors.
% Notice that our envelope has to be the same size as the signal. Since 'y' is
% a column vector and 'envelope' is a row vector (as are anything generated by the
% colon operator or 'linspace'), we need to transpose 'envelope'.

envelope = [linspace(0,1,Fs) ones(1,length(y)-2*Fs) linspace(1,0,Fs)];
y_fade = y.*envelope';

% Now we plot.  Notice what happens when we plot with only one parameter.

subplot(3,1,1);  % In a matrix of figures (3 rows, 1 column), select the 1st one
plot(y);
axis tight;
title('Faded Sound Sample');
subplot(3,1,2);  % Now select the second of the matrix of figures
plot(envelope);
axis([0 length(y) 0 1.2]);  % So we can see the envelope clearly
subplot(3,1,3);  % Third subplot...
plot(y_fade);
axis tight;
if play_sounds
    soundsc(y_fade,Fs);
end
disp('Hit a key to continue');
pause

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Let's clean up our workspace.  'clear' removes variables, 'close' closes figures.

close(1);  % Close figure 1.  We could close all figures with 'close all'
clear frq phase A B N M row_vector1 row_vector2 row_vector3;
clear col_vector1 col_vector2 col_vector3;
who        % These are the variables we have left in our workspace.
disp('Hit a key to continue');
pause

% In Appendix B, the text describes a "clip" command.  Let's implement this
% in a number of different ways.  The most straightforward (but slowest) is
% to use a FOR loop.  This is how you'd do it in most programming languages.

cos_wave = cos_wave(1:300);
thresh = 0.78;
clip1 = cos_wave;
for counter = 1:length(clip1)
    if clip1(counter) > thresh
        clip1(counter) = .78;
    elseif clip1(counter) < -thresh
        clip1(counter) = -.78;
    end
end

% There are a number of better ways to do this in MATLAB, because relational
% operators (<, >, ==) work on vectors.  The text gives the following
% (somewhat confusing) example:

clip2 = thresh*(cos_wave > thresh) - thresh*(cos_wave < -thresh) + ...
    cos_wave.*(abs(cos_wave) <= thresh);

% The text also shows one way to use the 'find' command.  Here's another way to
% use the find command:

clip3 = cos_wave;
too_big = find(abs(clip3) > thresh);
clip3(too_big) = thresh.*sign(clip3(too_big));

% Finally, the easiest way to implement this is to use MATLAB's built-in functions
% 'min' and 'max'

clip4 = min(max(cos_wave,-thresh),thresh);

subplot(2,2,1);
plot(clip1); title('Clip1');
subplot(2,2,2);
plot(clip2); title('Clip2');
subplot(2,2,3);
plot(clip3); title('Clip3');
subplot(2,2,4);
plot(clip4); title('Clip4');
disp('Hit a key to continue');
```

```
pause;

% As is usually the case, there are a LOT of different ways to perform this task.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 'max' and 'min' also let you locate the maximum or minimum value in a vector.

[max_value,max_index] = max(y);
subplot(1,1,1);
plot(y);
hold on;
plot(max_index, max_value,'rx');    % Why does this work?
hold off;
title('Maximum value of handel waveform.');
disp('Hit a key to continue');
pause;

% But there are actually multiple maximum values...

hold on;
plot(find(y == max_value),max_value,'ro');
hold off;
title('Maximum values of handel waveform.');
disp('Hit a key to continue');
pause;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Here are some other useful things we may need to do:

sum(cos_wave > thresh)  % Count the number of elements greater than 'thresh'
sum(cos_wave(find(cos_wave > thresh)))  % Sum of elements greater than 'thresh'
prod([1 2 3 4 5 6])     % Product of elements in vector.  This equals 1*2*3*4*5*6.
mean(cos_wave)          % Mean value of a vector
median(cos_wave)        % Median value of a vector
std(cos_wave)           % Standard deviation of a vector
disp('Hit a key to continue');
pause;

b = reshape(1:8,[2 4])  % Change the size of a matrix
repmat(b,[3,2])         % Replicate a matrix
fliplr(b)               % "Mirror" a matrix left-to-right
flipud(b)               % "Mirror" a matrix top-to-bottom
disp('Hit a key to continue');
pause;

size(b)                 % Returns a vector: [# rows, # columns]
size(b,1)               % Number of rows
size(b,2)               % Number of columns
length(b)               % Maximum of (# rows, # columns).  Good for vectors.
prod(size(b))           % Total number of elements in b
disp('Hit a key to continue');
pause;
```

```
all([1 1 0 1])          % Are all elements in a vector nonzero?
any([1 1 0 1])          % Are any elements in a vector nonzero?
all(all([1 1; 0 1]))    % Are all elements in a matrix nonzero?
any(any([1 1; 0 1]))    % Are any elements in a matrix nonzero?
disp('Hit a key to continue');
pause;

% Sorting: Let x and y be sets of ordered pairs (i.e., for a scatter plot).
x = [1 4 6 7 2 3];
y = [6 2 1 2 5 7];
disp([x; y]);
shg                      % Bring current graph to the foreground
plot(x,y,'mx');
axis([0 8 0 8]);
[x_srt,ind] = sort(x);   % We can sort x, but we need to reorder y as well...
y_srt = y(ind);          % That is what 'ind' is for
disp([x_srt; y_srt]);
hold on;
plot(x_srt,y_srt,'go');  % We've preserved the x-y pairs
hold off;

xy = [x' y'];            % Alternately...
xy_srt = sortrows(xy,1)  % We can use the 'sortrows' command
disp('Hit a key to continue');
pause;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% We can represent and graph mathematical functions easily in MATLAB

x = -2:.01:2;
x2 = x.^2;
x3 = x.^3;
exp_x = exp(-x);

plot(x,[x' x2' x3' exp_x']);  % Plot will also plot columns of a matrix
% If we want to specify different line styles, we can plot like this:
% plot(x,x,'k-',x,x2,'g:',x,x3,'r--',x,exp_x,'c-.');
grid on
title('Mathematical graphing');
xlabel('x');
ylabel('y');
legend('y = x','y = x^2','y = x^3','y = e^{-x}',0);
disp('Hit a key to continue');
pause;
```