13.42: HELPFUL MATLAB PLOTTING COMMANDS

©A.H. TECHET

MATLAB will be a useful tool to know throughout the course of the semester. The first homework can be simplified using MATLAB and the first lab will ask each lab group to use MATLAB prior to the lab session to generate a waveform to study. Throughout the term there may be other assignments which call for the use of a software package such as MATLAB. The specific use of MATLAB is not required.

Running MATLAB on the MIT server is quite simple. At a workstation you can use the software pull-down menus at the top of the screen or at a server prompt type *add matlab* press return, then *matlab*.

There are some common functions that are used frequently in MATLAB. Several of these are listed below with descriptions of their use. If you are looking for a certain function and want to know how to use it type **help command** at the prompt, replacing "command" with the function name. It is also useful to look at the "Using MATLAB" manual that ships with the PC version of MATLAB if you have access to one. There will be a copy available at each of the lab sessions in the Towtank. Mathworks has extensive information on their web site about the functions and using MATLAB *http://www.mathworks.com.* If a problem requires complex MATLAB code, a guideline will be included in the homework. If you choose to use a program other than MATLAB that is fine.

In MATLAB it is simple to create a script that performs a series of commands similar to a C or FORTRAN code. Edit a file with a $*.m$ extension in the working directory. Type commands in order you wish them performed and save the file. Note: a script is different from a MATLAB function in that variables defined and calculated in a script are global, but within a function variables are only local unless passed in and out (**help function**).

who: lists all variables in the MATLAB workspace. Another option is **whos**, which is similar in effect to '$ls - l$' in UNIX, lists the variables and their size. To look at the contents of a variable just type the variable name at the MATLAB prompt omitting the semicolon at the end of a line.

cd dir: changes to the desired directory, where your script is located, unless you include the working directory in the path you must be in the correct directory to run a function or script.

path: displays all the directories that MATLAB searches in for the functions called. If your function is not in one of these directories you must change to that directory before calling a function.

figure(1): make figure 1 the current plot; future plotting commands will be directed into the window for figure 1. This is useful when you have multiple figure windows open and want a script to plot in a window different from the current figure. Call the **figure(#)** command before the plot command.

subplot(3,1,2): breaks the current figure window into a three by one array of distinct plotting regions and sets the current region to be the second of the three. You can change the current region with something like **subplot(3,1,3)**. Can also type **subplot 312** dropping the brackets and commas. **subplot 324** will make a figure with three rows and two columns of plots (six total), the third number refers to which of the six plots will be made active. Counting is from left to right, top to bottom.

hold: hold the current contents of the active figure; this means that any subsequent plot commands will draw on top of whatever is already there rather than starting from scratch. A second **hold** command will release the figure. It is also possible to type **hold on** and **hold off** to toggle the hold state.

clf: clears the current figure window. Also removes any hold and/or subplot sectioning. This does not toggle the hold command– if hold is on it will stay on.

plot(x1, y1, '-', x2, y2, '–', x3, y3, ':'): plot three curves defined by the x_i and y_i vectors, each with a different line style (in this case, solid, dashed, and dotted). The text arguments after the y vector for each curve can also be used to define marker style (circle, x, *, etc.) and curve color. Type **help plot** for a list of color and marker codes.

legend('curve 1', 'curve 2', 'curve 3'): assign and draw the legend for curves 1 (the solid line), etc. You can click on the legend with the mouse and move it around once it has been drawn.

axis([xmin, xmax, ymin, ymax]): change the limits on the plot axes. **V = axis** will return the axis limits for the current figure.

xlabel('time (s)'): set the label on the x axis of the current graph (and current suplot if applicable) to the given text. Same holds true for **ylabel('H (m)')**. Certain greek symbols can be used in plots, such as ω by typing similar commands used in LaTeX. **xlabel('\ omega')** for example.

gtext('message text'): allows you to place the text given in the argument onto a graph using the mouse. Helpful for labeling the interesting features of any given plot.

ginput: allows you to click on points in the graph with the mouse. When you hit return you will leave input mode and the x and y coordinates for the points that you clicked on will be displayed. To record the values you click type $[x, y] = ginput(2)$; for two values (any integer number will work).

zoom: toggles the ability to zoom in or out on the current graph. When zooming, clicking on the left mouse button zooms you in around the point you clicked on; clicking on the right button zooms you out. **zoom on; zoom off**.

orient tall: set the orientation of the printed version of the plot to be full page portrait. Other options are portrait (the default) and landscape.

print -deps filename.eps: send the current figure to the file filename.eps using the current page orientation. There are other formats you can use– listed under **help print**.

[p,f] = spectrum(x, 512, 0, [], 200): generate a power spectrum for the data in vector x, using 512 point FFT's, with no overlap between windows, the default Hanning window and given that the sampling rate was 200 Hz. In general increasing the size of your FFTs will increase the resolution of your spectrum (the number of points for which you actually get information back); it will not increase the maximum frequency for which you get information back. That is controlled by the Nyquist frequency which is defined as one-half of your sample rate. Having FFTs longer than the length of x does not make sense and you should make them small enough such that several windows will actually slide along x so that you will get good averaging. It's traditional (and faster) to use FFT lengths which are a power of two.

semilogy(f, p(:,1)): plot the power spectrum that you just got from spectrum with a logarithmic y axis and a linear x axis. The second argument simply means that you only want to plot the first column ((:,1) = all rows, first column, (1:100,2) would be first 100 rows, second column, etc.).

loglog(f, p(:,1)): plot the power spectrum on a log-log scale.

[b,a] = butter(10.0, 50/100): define filter coefficients for a tenth order Butterworth low-pass filter with cut-off frequency of 50 Hz if the Nyquist rate is 100 Hz.

xf = filtfilt(b, a, x): filter the data in vector x with the filter defined by coefficients b and a.

help plot: get help on the plot command. Very helpful for getting the many details of **plot** and all the other commands that are not discussed here.

%: indicates a comment. All text to the right of this marker is ignored until the next line break.

;: at the end of a line stops the results from scrolling across the screen. Good for math with big arrays and matrices.

:

OTHER USEFUL COMMANDS:: Type **help command** to find out more:

mean, median, std, min, max, sqrt, sort, grid, semilogx, semilogy, sin, cos, tan, acos, asin, atan, log, log10, exp, load, fread, fwrite, fopen, save

TRY RUNNING THESE SAMPLE PROGRAMS

SAMPLE CODE 1

```
a = 2.5;                        % Amplitude of sine wave in cm
x = 0 : pi/20 : 2 ∗ pi;         % Theta from zero to 2 PI with 40 points between.
y = a ∗ sin(x);                 % Calculate y = sin(theta).
plot(x, y, ‘r−′);               % Plot x vs. y as a red line
xlabel(‘\theta(radians)′);
ylabel(‘y(cm)′);
title(‘sin(\theta)′);
```

SAMPLE CODE 2

```
vec1 = 10 : 10 : 200;                % Vector 1
vec2 = [2 : 4 : 20 400 : 2 : 440];   % Vector 2
mat1 = [2 : 4 : 20; 400 : 2 : 440];  % Matrix is created using 2 arrays of the
                                     % same length; the semicolon indicates a row break.
y1 = vec1 ∗ vec2′;                   % Multiply vec1 by transpose of vec2.
                                     % Result is scalar.
y2 = vec1. ∗ vec2;                   % Mult. each entry in vec1 by the entry in vec2 with
                                     % the same index. Vectors must be the same length.
                                     % Result is vector of same length as vec1, vec2.
size(y1)                             % show that y1 is a 1 x 1 array (a scalar).
size(y2)                             % returns the vector/matrix dimensions m x n.
length(y2)                           % returns the vector length
```