USING MATLAB

Basic info, Examples for Graphing Equations, Creating GUI Interfaces, etc.

INDEX

SOME BASICS The help files and online manual give good information about the commands but it can be extremely difficult to get basic information from these sources. An online manual is located at **<http://www.utexas.edu/cc/math/Matlab/Manual/ReferenceIX.html>**. Often a guess will lead you to the proper command. Other times the information is just not there. Depending on what version you have and what files are loaded into it, there may be many functions available that are not listed in the command reference. A good way to find information is by typing **`lookfor`** followed by a keyword in the Matlab window and hit enter. It will return a list of available function filenames that contain that word. Since the functions have comments that explain what they do, you can enter plain English words and get good results. Open a function file and the comments should describe what it does and how to use it. Function files have .m extensions and can be opened with a text editor such as notepad. The introductory comment can be displayed in the Matlab window by simply by typing help followed by the filename, e.g. **`help semilogx`** will display the initial comments in the matlab file semilogx.m. When you write your own functions, include a description in the first few lines and you can access it the same way.

THE TWO MOST IMPORTANT THINGS TO KNOW If Matlab embarks on some lengthy process that you didn't intend it to do you can put a stop to it by pressing **Ctrl + c**. The second important thing is that if you don't want Matlab to echo the stuff you type and the calculations it makes, then type a semicolon at the end of each command. Forgetting to use the semicolon is the most common reason for having to us the **Ctrl + c** command. The program execution time is greatly increased if all the calculations are being displayed in the Matlab window.

EXECUTING COMMANDS Although you can use Matlab by just typing commands in the window and hitting enter, it is much better to type your commands in a text file and save it in a directory where Matlab can find it. Give the text file a **`.m`** extension. Now you can run those commands by typing the filename (without the **`.m`**) in the Matlab window. This is the way to do it because you can refine the commands by editing the text file and you don't have to retype the whole thing every time you want to run it. In UNIX, a file of the commands can be created with a text editor such as Pico or Emacs and saved with a `.m` extension. The windows version of Matlab has it's own editor.

`eval(s)` will execute the commands in the text string "s". This is what I was looking for so that I could pass equations to a function for evaluation. An alternate way to prepare an expression for the eval command is to first initialize one of its variables using the syms command, e.g. **`syms x`**, and then store an expression using the variable **`x`** to another variable, e.g. **`Var=3*x^2+t`**, and evaluate, e.g. **`eval(Var)`**. The fact that the expression **`3*x^2+t`** contains at least one variable initialized using the **`syms`** command causes Matlab to save the expression into the variable **`Var`** rather than evaluate it at that time.

EXECUTING THE PROGRAM In the UT Math lab we start Matlab by typing `matlab` at the linux prompt.

CREATE A SIMPLE PLOT This sample plots $y = x^2 + 3x$ on one graph. The "for" loop is required because taking the square is not a valid matrix operation:

```
newplot                       % starts a new graphing window
X=0; Y=0;                     % initialize variables
for x = 0 : .1 : 5            % repeat for values of x   start : increment : end
y = x^2 + 3*x;                % the equation
X = [X,x]; Y = [Y,y];         % form matrices for plotting
end                           % end of "for" statements
plot(X,Y);                    % variables to plot
grid on                       % make the grid visible
```

CREATING MULTIPLE PLOTS This sample plots 11 lines on one graph:

```
newplot                                 % starts a new graphing window
x = -6 : 0.025 : 1.5;                   % plot from : resolution : plot to
axis([-6 1.5 -1 2.5])                   % extents of plotting window ([x1 x2 y1 y2])
hold on                                 % complete the following instructions before
plotting
for b = -5 : 1 : 5                      % list of values for b   start : increment : end
y = (2-b)*exp(x) + (b-1)*exp(2*x);      % the equation
plot(x,y);                              % variables to plot
end                                     % end of instructions
hold off                                % create the plot
grid on                                 % make the grid visible
```

3D PLOTS Example:

```
[X,Y] = meshgrid(-8:.5:8);              %
R = sqrt(X.^2+Y.^2)+eps;                % eps prevents a divide by zero error
Z = sin(R)./R;                          % create 2 × 2 matrix
surf(Z)                                 % Z is a matrix of heights above an i × j grid
colormap hot                            % defines colors used for 3D
shading interp                          % or shading flat eliminates mesh lines
```

Another example:

```
[X,Y] = meshgrid([-2:.15:2],[-4:.3:4]);  %
Z = X.*exp(-X.^2-Y.^2);                 % function Z = Xe^(-X^2-Y^2)
surf(X,Y,Z)                             % or mesh(X,Y,Z)
```

LOG SCALE PLOTS

```
semilogx(X,Y);          % linear y-axis, log x-axis
semilogy(X,Y);          % linear x-axis, log y-axis
loglog(X,Y);            % log y-axis, log x-axis
```

An Example:

```
N=[]; X=[]; Y=[]; x=10^-2;                        % initialize variables
while x < 10^8,                                   % repeat for values of x   start : increment : end
s = i*x;                                          % get the j-omega
y = 10^10*s * (s+.1)^2 / ((s+1) * (1+s/10^4)^2);  % the function to plot
X = [X,x]; N = [N,y];                             % form matrices for X and complex Y values
x = x*1.05;                                       % controls the resolution of the plot
end                                               % end of "while" statements
Y = 20*log10(abs(N));                             % get the Y-matrix in dB
semilogx(X,Y);                                    % variables to plot
```

WORKING WITH FIGURES (PLOT WINDOWS)

To create a figure and store its handle: `FigName = figure`.

To specify figure location and size `figure('Position',[Left Bottom Width Height])`

Special characters can be used in figure labeling:

```
{\itA{_0}e}^{-\alpha{\itt}sin\beta{\itt} \alpha<<\beta
```

makes $A_0 e^{-\alpha t} \sin\beta t \quad \alpha << \beta$

The title of a figure and its properties can be set:

set(get(h,'Title'),'String','This is the Title Text','FontName', 'times','FontAngle','italic','FontSize',14)

The get command gets the figure handle; the set command then sets the properties.

To superimpose two plots with additional x-y axes:

```
hl1 = line(x1,y1,'Color','r');
ax1 = gca;
set(ax1,'Xcolor','r','Ycolor','r')
ax2 = axes('Position',get(ax1,'Position'),'XaxisLocation','top',
   'YaxisLocation','right','Color','none','Xcolor','k','Ycolor','k');
hl2 = line(x2,y2,'Color','k','Ycolor','k');
% to make the grid lines line up:
xlimits = get(ax1,'Xlim');
ylimits = get(ax1,'Ylim');
xinc = (xlimits(2)-xlimits(1))/5;
yinc = (ylimits(2)-ylimits(1))/5;
set(ax1,'Xtick',[xlimits(1):xinc:xlimits(2)],'Ytick',
   [ylimits(1):yinc:ylimits(2)])
```

To plot two functions with **separate y-axes**:

```
plotyy(X1,Y1,X2,Y2);
```

To return handles for the left axis AX(1), the right axis AX(2), and the graphics objects for each plot in H1 and H2:

```
[AX,H1,H2] = plotyy(X1,Y1,X2,Y2);
set(AX(1),'XLim',[0,5])              %set X1 axes limits
set(AX(2),'XLim',[0,5])              %set X2 axes limits
```

Setting **tick marks**, font, color, etc.:

```
set(gca,'Ytick',[0 2 4 6 8],'FontSize',16,'Xcolor','k','Ycolor',[0 0 0])
```

"gca" means "**get current axes**" and refers to getting the "handle" for the axes so that properties may be set. If the handle has already been stored in a variable, then the variable name may be substituted for gca. Axes has numerous properties such as "YTick" and "YColor." There are several ways of **specifying colors**. In the above example, `'k'` and `[0 0 0]` are equivalent and mean black. The three zeros are RGB values which may vary from 0 to 1.

The following statements give the plot a **title** and **label the X and Y axes**. These statement should be placed after the plot statement.

```
title('Shock Hazard for a 1300\Omega Person','FontSize',18,'Color',[0 0 0])
xlabel('Horizontal Distance in Feet','FontSize',16, 'Color',[0 0 0])
ylabel('Milliamps per Kilometer of Fence','FontSize',16, 'Color',[0 0 0])
```

Text Notes: To make an italic *t*: {\itt}. To make an ω: {\omega}. To make superscripts and subscripts like k_1^2: k{_1}{^2}.

The following statement sets the **limits of the X and Y axes** and goes somewhere after the plot statement. Note that the spelling is axis, which refers to something different than axes.

```
axis([Xmin Xmax Ymin Ymax])
```

To **copy a figure** (plot) to the clipboard in Windows metafile format, type `print –dmeta` in the matlab window. This seems to do the same thing as the menu command Edit/Copy Figure. Color and size formatting is lost. What I do is to make the figure as large as possible, use the Print Screen button to copy the whole screen to the clipboard, then paste it into Paintshop Pro, reduce it to two colors if desired, and paste it into a Word document. If you are running a high screen resolution, you can get a reasonably high resolution image with all the formatting intact.

CREATING A FUNCTION, USE OF MATRICES

The following sample is a function file to be saved as `brine.m` and called from another file. It illustrates the use of matrices:

```
function  X = brine(t)                 % Function declaration to be called with the name
brine
                                       % passing the variable  t  and returning the value  X.

A = [-0.167,    0,      0,    0,  0;          % This is a 5 x 5 matrix.
      0.167, -0.5,      0,    0,  0;          % Rows are separated by semicolons.
          0,  0.5, -0.125,    0,  0;          % Columns are separated by commas.
          0,    0,  0.125, -0.2,  0;          % The final semicolon just means that we
don't
          0,    0,      0,  0.2, -0.25];      % want Matlab to redisplay the contents.

[V,D] = eig(A);                        % This command has the program find a matrix of
                                       % eigenvalues  D  and a matrix of eigenvectors  V.
                                       % The trailing semicolon may be omitted if we wish
                                       % the results to be displayed.

C = inv(V)*[10;0;0;0;0];               % This is a 1 x 5 matrix of initial values

X = V*expm(t*D)*C;                     % expm  means the exponential function of a matrix
```

CALLING THE FUNCTION FROM A SEPARATE FILE

The following file works together with the above to solve and graph a problem:

```
A = [10;0;0;0;0];                      % Define a matrix  A.
for n=1:480                            % Draw the graph in 481 segments
 A(:,n+1) = brine(0.125*n);            % Calls the function  brine.m  above and passes the
                                       % value product 0.125 x n.
end

newplot;                               % Begin a graph
t=0:0.125:60;                          % Plot t from 0 with resolution 0.125 to 60
axis([0 60 0 10 ]);                    % Extents of the plotting window x1 x2 y1 y2
hold on                                % Draw all lines before finishing the graph.
for b = 1 : 1 : 5                      % Draw five lines
plot(t+(b-1),A(b,:));
end
hold off                               % Complete the graph
grid on                                % Show the x-y grid
xlabel('Time in Seconds')              % optional label for x-axis
ylabel('Brine Concentration')          % optional label for y-axis
```

WORKING WITH MATRICES

`A = [1 2 3;4 5 6;7 8 9];`	% A matrix is entered. Column elements are separated by spaces % (or commas) and rows are separated by semicolons. The % semicolon at the end is optional and supresses the output, % i.e. Matlab does not echo the matrix in the Matlab window.
`A = [1 2 3]'`	% An apostrophe transposes a matrix. This would ordinarily be % a row matrix; the apostrophe makes it a column matrix.
`A * B`	% Matrix multiplication is just like regular multiplication.
`det(A)`	% The **determinant** is computed.
`inv(A)`	% The **inverse matrix** is computed.
`rref(A)`	% The **reduced row echelon form** is computed.
`Rows = size(A,1)`	% The **number of rows** in matrix A.
`Cols = size(A,2)`	% The **number of columns** in matrix A.

To specify matrix elements, **`A(3,4)`** is the element of the 3rd row, 4th column of matrix **`A`**. If **`A`** is a 4x4 matrix, then **`A(12)`** would refer to the same element. **`A(1:3,4)`** refers to the first 3 elements of the 4th column; **`sum(A(1:3,4)`** is their sum. **`A(:,3)`** means all of column 3. **`A(:,3) = []`** deletes column 3.

CELL ARRAYS Regular matrices can hold only one type of data, i.e. strings or numbers, not both. A **cell array** can hold multiple data types. There are two types of assignment statements.

Cell Indexing:

```
A(1,1) = {[1 4 3;0 7 8]};
A(1,2) = {'Ann Smith'};
```

Content Indexing:

```
A{1,1} = [1 4 3;0 7 8];
A{1,2} = 'Ann Smith';
```

You can preallocate a cell array:

```
B = cell(2,3);
```

To access data from a cell array:

```
C = B{1,2};
```

Or to access an element of a submatrix from a cell array:

```
C = B{1,2}(3,4);
```

MULTIPLICATION OF FUNCTIONS

When multiplying two functions together it is necessary to use a period with asterisk .* for the operator i.e. **`y = exp(-x).*cos(x).`**

USING COMPLEX NUMBERS

A complex number is stored in a single variable. The variables **`i`** and **`j`** are reserved for the square root of negative one.

`A = 3+2*i;`	% The number `(3+2i)` is saved to variable **A**.
`B = phase(A);`	% The phase angle of `(3+2i)` is saved to variable **B**.
`C = CP2MP(A);`	% The magnitude of `(3+2i)` is saved to variable **C**.
`D = real(A);`	% The real part of `(3+2i)` is saved to variable **D**.
`E = imag(A);`	% The imaginary part of `(3+2i)` is saved to variable **E**.
`F = abs(A);`	% The absolute value of `(3+2i)` is saved to variable **F**.

FORMATTING THE OUTPUT (DECIMAL PLACES)

Matlab has some shortcomings in this area, i.e. few choices. **`format short`** is the default. This is 5-digit precision. **`format long`** is 15-digit. **`format bank`** gives two decimal places. It is difficult to get anything else. If you change the format, it will remain that way until you change it back. If you want to arrange the data in columns, you can put it in a matrix, but it has to all be formatted the same way.

You can use **`num2str()`** to convert to a string. Trailing zeros will be dropped from decimal numbers, so decimals will not line up. One could use this command with a little code to deal with formatting.

WORKING WITH POLYNOMIALS

To represent the **polynomial** $x^3 - 2x - 5$, save the coefficients in a matrix:

```
p = [1 0 –2 –5]
```

To **factor the polynomial** equation $x^3 - 6x^2 + 11x - 6 = 0$, enter the coefficients into a matrix and use the roots() command:

```
p = [1 -6 11 -6];            % Matrix of coefficients of the polynomial
r = roots(p)'                % Finds the roots of the polynomial. The apostrophe simply
                             % makes the result r a row matrix (takes less room) rather
                             % than a column matrix.
```

The result will be 3.0000 2.0000 1.0000. This means that the factors are (x-3)(x-2)(x-1).

To find a polynomial to **fit a curve**:

```
p = polyfit(x,y,n)
```

where **`x`** is a matrix of x-values, **`y`** is a matrix of y-values, and **`n`** is the order of polynomial desired.

SWITCH/CASE STATEMENTS (LIKE SELECT/CASE)

In Matlab, the select case statement is switch case:

```
switch expression
   case value 1
      statements
   case value 2
      more statements
   otherwise
      other statements
end
```

DERIVATIVES (Van der Pol equation)

A second order derivative must be rewritten as a system of equations. Example:

$y_1'' - u\left(1 - y_1^{\,2}\right) y_1' + y_1 = 0$ where u>0 is a scalar parameter

System:

$y_1' = y_2$

$y_2' - u\left(1 - y_1^{\,2}\right) y_2 - y_1$

Create an ODE file (m-file) for u=1:

```
Function dy = vdp1(t,y)
Dy = [y(2); (1-y(1)^2)*y(2)-y(1)];
```

Note that the *t* argument is not used, but must appear in the argument list along with *y*.

To solve over the time interval 0-20 with initial values *y*(1)=2, *y*(2)=0:

```
[T,Y] = ode45(‘vdp1’,[0 20],[2;0]);
```

T is a column vector of time points. Y is a solution array of *y*(1) and *y*(2) values, each row corresponding to a value of T.

To plot both *y*(1) and *y*(2):

```
plot(t,y(:,1),'-',t,y(:,2),'—');
```

For u = 1000, the character of the problem changes, called a *still* problem. We must use the **`ode15s`** command to evaluate it. Syntax is the same. There are other ode solvers also. Some operations require the function to be saved to an m-file. Here is an example function called **`humps.m`**:

```
Function y=humps(x)
Y = 1./((x-p.3).^2+0.01)+1./((x-0.9).^2+0.04)-6;
```

To integrate this function from 0 to 1:

```
Q = quad('humps',0,1)
```

To perform the double integration $\int_{Y_{\min}}^{Y_{\max}} \int_{X_{\min}}^{X_{\max}} f(x,y)\,dx\,dy$:

```
R = quad2('name',Xmin,Xmax,Ymin,Ymax)
```

FOURIER TRANSFORM

There are several tools for Fourier transforms and I don't fully understand this yet. The one I have used is undocumented except for the comments found by typing **`help fourier`** in the matlab window and that is **`fourier()`**. It only works for certain functions. **`x`** is the independent variable for the time domain and **`s`** is the independent variable for the frequency domain. Example:

```
syms s x                          % Treat expressions with these variables as expressions
EX = exp(-0.2*x^2);               % A gaussian
F = Fourier(EX,x,s); x=-8; s=x; X=[]; Y1=[]; Y2=[];
while x <= 8
   y1 = eval(expression); y2 = eval(F);
   Y1=[Y1,y1]; Y2=[Y2,y2]; X=[X,x];
   x=x+0.01; s=x;
end
plot(X,Y1,'LineWidth',3)          % Plot the expression with a bold line
hold on
plot(X,real(Y2))                  % Plot the Fourier transform with a thin line
hold off, grid on
```

In the example above, the line **`F = Fourier(EX,x,s);`** says to perform the fourier transform on the expression stored in **`EX`** with respect to the independent variable **`x`** and store the result in **`F`** as a function of the variable **`s`**.

The function for the inverse fourier transform is **`ifourier()`**. I haven't tried this yet. It's use is in the following form; note the reversal of s and x.

```
Orig = iFourier(F,s,x);           % Should return the original function from the example above
```

EXPERIMENT WITH DFIELD AND PPLANE.

DFIELD5 is an interactive tool for studying single first order differential equations. When DFIELD5 is executed, a DFIELD5 Setup window is opened. The user may enter the differential equation and specify a display window using the interactive controls in the Setup window.

When the Proceed button is pressed on the Setup window, the DF Display window is opened. At first this window displays a direction line field for the differential equation. When the mouse button is depressed in the DFIELD5 Display window, the solution to the differential equation with that initial condition is calculated and plotted.

Other options are available in the Options menu. These are fairly self explanatory. The Settings option allows the user to change several parameters. Included here are the possibilities of using a vector field instead of the default line field, and of changing the number of field points computed and displayed.

PPLANE5 is an interactive tool for studying planar autonomous systems of differential equations. When PPLANE5 is executed, a PPLANE5 Setup window is opened. The user may enter the differential equation and specify a display window using the interactive controls in the Setup window. Up to 4 parameters may also be specified. In addition the user is give a choice of the type of field displayed and the number of field points.

When the Proceed button is pressed on the Setup window, the PPLANE5 Display window is opened, and a field of the type requested is displayed for the system. When the mouse button is depressed in the PPLANE5 Display window, the solution to the system with that initial condition is calculated and plotted.

Other options are available in the menus. These are fairly self explanatory.

This is version 5.0b10, and will only run on version 5 of MATLAB.

MAKING MATLAB PROGRAMS RUN FASTER

Matlab handles matrix operations efficiently. Loops such as if, for, while, etc., are handled much more slowly. For example one program I wrote using loops ran for 8 hours before I stopped it. The same program rewritten using matrices ran in 90 seconds. Many of the examples here use loops but could be rewritten to use matrices.

CREATING A GUI INTERFACE

To invoke the GUI editor, type **`guide`** in the Matlab window. This brings up the Guide Control Panel. From this window, buttons, listboxes, text windows, etc., can be added to the figure window (which also comes up when the guide command is given). To make a button do something, use the Guide Control Panel to open the Callback Editor. Make sure the button is selected, and in the Callback Editor window, type the commands or function filename to be run. Program execution will be faster if functions are used. The manual suggests using a single function file with a switch/case statement to handle multiple buttons in your figure. This reduces the number of files associated with the figure and makes maintenance easier. For example:

```
function myGui(action)
switch(action)
     case 'load'
          load mydata
          set(gcbf,"UserData',XYData)
     case 'plot'
          XYData=get(gcbf,'UserData')
          x=XYData(:,1);
          y=XYData(:,2);
          plot(x,y)
     case 'close'
          close(gcbf)
end
```

This example executes code when one of the three buttons, **`load`**, **`plot`**, or **`close`**, is clicked. The term **`gcbf`** (Get Callback Figure) gets the figure handle. Similar terms are **`gca`** (Get Current Axes) and **`gcbo`** (Get Callback Object). The corresponding statement in the load button's callback window would be:

```
myGui load
```

To edit the current (previously created) figure, type **`guide(gcf)`** in the Matlab window.

There seems to be a problem with saving a figure in some directories. I am not sure if the directory needs to be on the Matlab path, or if it has to do with non-standard directory names such as those containing spaces, or if it is because there is another file somewhere with the same name. There will be no indication that there is a problem, so the user should verify that the files have actually been created. There will be 2 files, a **`.m`** and a **`.mat`**.

MORE ABOUT HANDLES

Handles have been mentioned elsewhere in this document. They are needed in order to reference and change a property. An example is:

```
set(gca,'FontSize',16)
```

This code sets the fontside of the axes. **`gca`** is a command that obtains the handle. The handle name (in single quotations) or a variable containing it could also be used here. **`'FontSize'`** is the property to be set and **`16`** is the new value of the property. Additional properties could be set here by following the last value with another comma, then another property name, comma, its value, etc.

Some commands to retrieve handles are:

`gca` (Get Current Axes)
`gcf` (Get Current Figure)
`gcbf` (Get Callback Figure)
`gcbo` (Get Callback Object)

Using `findobj()` - A handle can be saved to a variable so that it can be used again without retrieving it each time. This practice is discouraged because the item to which the handle refers could be terminated and an error would result when using its handle. Using the **`gc*`** commands will prevent this. Another alternative is the **`findobj()`** command. This command will retrieve the handle for the object having a specified property. For example:

```
myHandle = findobj('Tag','My Line');
```

This code searches the current objects for one whose **`'Tag'`** property is **`'My Line'`**, and saves its handle in the variable **`myHandle`**.