Introduction to using MATLAB

MATLAB is a powerful mathematics package that can do many things. In this lab we are going to concentrate on three major areas:

1. Plotting data and functions on a 2 dimensional graph

2. Performing linear regression on a set of data to find the slope and y-intercept

3. Solving a system of simultaneous equations

The most straightforward way to learn to use MATLAB is to try examples and see what they do. Below three examples are presented and explained. Type these examples in and observe the results on the computer. Modify these examples and see how the results change. Finally, several problems are presented for you to create your own MATLAB files to solve.

Commands in MATLAB may be utilized in two basic formats. The first is to type in the MATLAB workspace the commands you wish MATLAB to perform. This is useful when you are experimenting with a new command for the first time. However if you have a repetitive task to perform this approach is not very efficient. The second way to have MATLAB execute commands is the use of an m file. m files are files containing lists of MATLAB commands. MATLAB then executes these commands. In this lab, the three examples are presented as m files. They are hydrorw2011.m, regressw2011.m, and simplexw2011.m

In the MATLAB workspace, which is the environment to which MATLAB starts when you begin MATLAB, you create an m file by clicking on the NEW option under the FILE menu choice. After selecting NEW you are presented with three options, choose M-FILE. You are then in the notepad utility. Type in the commands as given below exactly. Once the example is typed in save it making sure it has an extension of .m.

To run the example, return to the MATLAB workspace and type the name of the example such as hydrorw2011 MATLAB will then run the commands contained in the m file. Your files will be saved in the MATLAB\Work directory.

Note MATLAB is based upon the m file concept. All of MATLAB's functions are in reality m files themselves. For example, the sin function is located in c:\MATLAB\toolbox\MATLAB\elfun\sin.m However; all basic MATLAB functions are called by simply typing their name. sin will automatically call this function for you. As you create your own m files you can also call them at any time in your projects. Be careful not to name your files with names that already exist such as functions like sin, cos, exp, etc. MATLAB is often picky about filenames beginning with numbers such as 12lab.m, and MATLAB can be fussy about spaces within filenames; I would avoid them. Moreover, if you use related files such as .mat files and others, it is probably wise to use slightly different names so again MATLAB does

not confuse them. For example, if you had a hydror.m having a related file of hydror.mat is probably not wise. For this lab exercise, you will not need .mat files, but you may use them later.

Finally pay attention to letter case. MATLAB is very fussy about that. Do not capitalize unless you are sure.

Example #1 Plotting and integrating Hydrogen's Wave functions

The radial wave functions for the lowest six energy states of Hydrogen are shown below. Where a_0 is the Bohr radius, Z is the charge of the nucleus:

$$\boldsymbol{R_{100} = 2\left(\frac{Z}{a_0}\right)^{3/2} e^{-Zr/a_0}}$$

$$\boldsymbol{R_{200} = \left(\frac{Z}{2a_0}\right)^{3/2}\left(2 - \frac{Zr}{a_0}\right)e^{-Zr/2a_0}}$$

$$\boldsymbol{R_{210} = \frac{1}{\sqrt{3}}\left(\frac{Z}{2a_0}\right)^{3/2}\left(\frac{Zr}{a_0}\right)e^{-Zr/2a_0}}$$

$$\boldsymbol{R_{300} = \frac{2}{81\sqrt{3}}\left(\frac{Z}{a_0}\right)^{3/2}\left(27 - 18\frac{Zr}{a_0} + 2\left(\frac{Zr}{a_0}\right)^2\right)e^{-Zr/3a_0}}$$

$$\boldsymbol{R_{310} = \frac{4}{81\sqrt{6}}\left(\frac{Z}{a_0}\right)^{3/2}\left(6 - \frac{Zr}{a_0}\right)\left(\frac{Zr}{a_0}\right)e^{-Zr/3a_0}}$$

$$\boldsymbol{R_{320} = \frac{4}{81\sqrt{30}}\left(\frac{Z}{a_0}\right)^{3/2}\left(\frac{Zr}{a_0}\right)^2 e^{-Zr/3a_0}}$$

The probability density functions for these states are obtained by:

$$\boldsymbol{P(r) = r^2(R(r))^2}$$

Finally, the expected positions of an electron in these energy states is given by:

$$\boldsymbol{r_{expected} = \int_0^\infty rP(r)dr}$$

Since we are using discrete values in our calculations, expected positions are found from:

$$r_{expected} = \sum rP(r)\Delta r$$

Note: at this point in the course, you are not supposed to understand what these functions are. For now, they can be thought of as simply mathematical functions that you wish to see plotted. Type in the following m file exactly as shown here. Following the m file listing, the m file is repeated but with explanations for the various parts of the file. Type only the listing first given.

hydrorw2011.m - The Listing

```
%Example program to help students learn MATLAB
%this program plots the radial wave functions and radial
%probability density functions for the first six Hydrogen
%electron levels
%version 2011-01-10 D.W. Donovan

clear all;

a0=1;
Z=1;
r=(0:0.2:30)';
pnc=(Z/a0)^1.5;
pnc2=(Z/2*a0)^1.5;
Zr=Z*r;

R100=2*pnc*exp(-Zr);
R200=pnc2*(2-r).*exp(-Zr/2);
R210=pnc2/sqrt(3)*Zr.*exp(-Zr/2);
R300=2*pnc/(81*sqrt(3))*(27-18*Zr+Zr.^2).*exp(-Zr/3);
R310=4*pnc/(81*sqrt(6))*(6-Zr).*Zr.*exp(-Zr/3);
R320=4*pnc/(81*sqrt(30))*Zr.^2.*exp(-Zr/3);
P100=(r.*R100).^2;
P200=(r.*R200).^2;
P210=(r.*R210).^2;
P300=(r.*R300).^2;
P310=(r.*R310).^2;
P320=(r.*R320).^2;

figure;
hold on;

tt1a=('Radial Wave Functions for Hydrogen Atom');
```

```
tt2='D.W. Donovan - ';
tta=[tt1a,'\newline',tt2,date];
title(tta)
xlabel('Radial Distance, r(in units of a0)');
ylabel('Radial Wave Function, R');
axis([0 7 -.15 1.7]);

plot(r,R100,'r.');
text(.35, 1.65,'100','color','r');
plot(r,R200,'b.');
text(.35,.6,'200','color','b');
plot(r,R210,'g.');
text(3,.20,'210','color','g');
plot(r,R300,'m.');
text(.1,.25,'300','color','m');
plot(r,R310,'k.');
text(1,.03,'310 ->','color','k');
plot(r,R320,'c.');
text(2.8,-.01,'320','color','c');
hold off

figure;
hold on;
tt1='Radial Probability Density for Hydrogen Atom';
tt=[tt1,'\newline',tt2,date];
title(tt,'FontSize',14)
xlabel('Radial Distance, r(in units of a0)','FontSize',14);
ylabel('Radial Probability Density Function,  \Psi','FontSize',14);
axis([0 20 0 .60]);

plot(r,P100,'r.','MarkerSize',10);
text(1, .55,'100','color','r','FontSize',13);
plot(r,P200,'b.','MarkerSize',10);
text(6, .21,'200','color','b','FontSize',13);
plot(r,P210,'g.','MarkerSize',15);
text(3, .21,'210','color','g','FontSize',13);
plot(r,P300,'m.','MarkerSize',15);
text(6, .3,'300','color','m','FontSize',13);
plot(r,P310,'k.','MarkerSize',20);
text(13, .13,'310','color','k','FontSize',13);
plot(r,P320,'c.','MarkerSize',20);
text(10, .13,'320','color','c','FontSize',13);
hold off
 nr=r';
```

```
rexp100=(nr*P100)*.2
rexp200=(nr*P200)*.2
rexp210=(nr*P210)*.2
rexp300=(nr*P300)*.2
rexp310=(nr*P310)*.2
rexp320=(nr*P320)*.2

%{
rexp100 =  1.5001
rexp200 =  6.0000
rexp210 =  5.0000
rexp300 =  9.7457
rexp310 = 12.4741
rexp320 = 10.4922
%}
```

hydrorw2011.m – Explained

```
%Example program to help students learn MATLAB
%this program plots the radial wave functions and radial
%probability density functions for the first six Hydrogen
%electron levels
%version 2011-01-10 D.W. Donovan
```

% denotes a comment, so anything on a line after % is ignored by MATLAB If one starts a program with several lines of comments, then if you type help then the name of the program at the command prompt (>>) you will get all the initial comments printed. For example if you type help hydrorw2011 the command window will come back with the following:

```
Example program to help students learn MATLAB
 this program plots the radial wave functions and radial
 probability density functions for the first six Hydrogen
 electron levels
 version 2011-01-10 D.W. Donovan
```

```
clear all;
```

This command cleans out MATLAB=s workspace of all previously used variables. MATLAB gives error messages if variables are re-sized incorrectly. This allows one to use common variable names repeatedly in different m-files. Note a semicolon (;) prevents the commands on that line from being written to the workspace during execution.

```
a0=1;
```

Set a_0 to 1 to keep the scale useful.

```
Z=1;
```

Charge is 1 for Hydrogen

```
r=(0:0.2:30)';
```

This creates a range of data. The syntax is variable = (start value: step size : end value) Note a single quote (') transposes a matrix. This range command creates a row vector; however, we need a column vector for the equations. Transposing a row gives the column.

```
pnc=(Z/a0)^1.5;
pnc2=(Z/2*a0)^1.5;
Zr=Z*r;
```

These lines create normalization variables needed in the functions. Rather than writing these out each time the new constants can be used now.

```
R100=2*pnc*exp(-Zr);
R200=pnc2*(2-r).*exp(-Zr/2);
R210=pnc2/sqrt(3)*Zr.*exp(-Zr/2);
R300=2*pnc/(81*sqrt(3))*(27-18*Zr+Zr.^2).*exp(-Zr/3);
R310=4*pnc/(81*sqrt(6))*(6-Zr).*Zr.*exp(-Zr/3);
R320=4*pnc/(81*sqrt(30))*Zr.^2.*exp(-Zr/3);
```

Definitions of the 6 wave functions

```
P100=(r.*R100).^2;
P200=(r.*R200).^2;
P210=(r.*R210).^2;
P300=(r.*R300).^2;
P310=(r.*R310).^2;
P320=(r.*R320).^2;
```

Definitions of the 6 probability density functions

Note the period (.) before the * and the ^ that tells MATLAB to perform those operations on the individual elements of the matrices themselves.

If the period is not there, the multiplication would be normal MATRIX multiplication. With the periods the effect is $P100_i = (r_i * R100_i)^2$.

Without the periods the effect would be

$$P100_{im} = (r_{ij} * R100_{jk})(r_{kl} * R100_{lm}) .$$

```
figure;
```

Tells MATLAB to create a new figure window. If you did not have a figure window, any plot command would create a figure window; however, if you had an existing figure window, then any new plot commands writes over your last figure. By creating a new figure window, you keep your figures from being lost.

```
hold on;
```

Tells MATLAB to plot more than one plot on the same figure Without this command and without a figure command, MATLAB would draw over the old plots without saving them and only the last plot command would remain.

```
tt1a=('Radial Wave Functions for Hydrogen Atom');
tt2='D.W. Donovan - ';
tta=[tt1a,'\newline',tt2,date];
```

Defines string variables to be used to create a title for the plot. MATLAB uses LATEX commands. Thus, \newline makes the title have 2 lines instead of one. The date command prints the actual date on your plot automatically. tta=[ ] combines the different string elements into a single variable which is required for the title command next.

```
title(tta)
```

places a title on your graph

```
xlabel('Radial Distance, r(in units of a0)');
```

places a label on the x axis. Again LATEX commands can be used and will be done below to create appropriate Greek characters.

```
ylabel('Radial Wave Function, R');
```

places a label on the y axis

```
axis([0 7 -.15 1.7]);
```

Defines the axis ranges the syntax is axis ([xmin xmax ymin ymax]) No commas inside!

```
plot(r,R100,'r.');
```

Plot the points defined in the vectors. The syntax is plot (xvector, yvector, 'control string') If xvector and yvector are single

numbers you would plot a single point. If they are vectors, you plot the set of points. Control strings allow for colors and the symbols plotted. Here the r stands for red, b for blue and so on. The periods indicate a '.' should be plotted for the data point. Other symbols include: *, +, -, letters, etc.

```
text(.35, 1.65,'100','color','r');
```
text command allows you to place a string at a specified point within the figure. The syntax is text (x coordinate, y coordinate, 'string') The x and y coordinates are related to the x and y axis themselves and not any figure positions. Rescaling the x and y axes will cause the placed strings to be moved.

```
plot(r,R200,'b.');
text(.35,.6,'200','color','b');
plot(r,R210,'g.');
text(3,.20,'210','color','g');
plot(r,R300,'m.');
text(.1,.25,'300','color','m');
plot(r,R310,'k.');
text(1,.03,'310','color','k');
plot(r,R320,'c.');
text(2.8,-.01,'320','color','c');
hold off
```
Tells MATLAB not to add anymore plots to the current figure

```
figure;
```
Create a new figure

```
hold on;
tt1='Radial Probability Density for Hydrogen Atom';
tt=[tt1,'\newline',tt2,date];
title(tt,'FontSize',14)
```
FontSize allows one to change the size of the font used in titles, axis labels, inserted text. The number is in "points"

```
xlabel('Radial Distance, r(in units of a0)','FontSize',14);
ylabel('Radial Probability Density Function,  \Psi','FontSize',14);
```
Note the \Psi is used to create a capital Ψ character if \psi was used ψ would be created

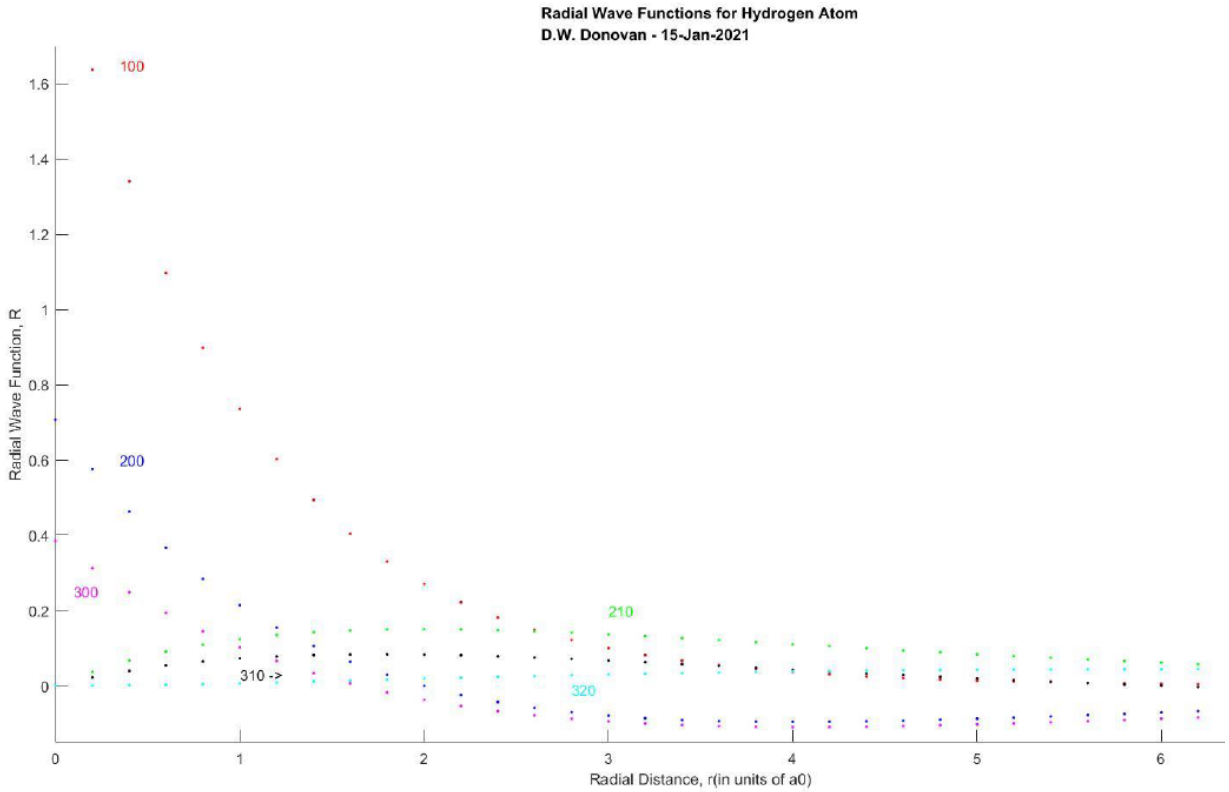
```
axis([0 20 0 .60]);

plot(r,P100,'r.','MarkerSize',10);
text(1, .55,'100','color','r','FontSize',13);
plot(r,P200,'b.','MarkerSize',10);
text(6, .21,'200','color','b','FontSize',13);
plot(r,P210,'g.','MarkerSize',15);
text(3, .21,'210','color','g','FontSize',13);
plot(r,P300,'m.','MarkerSize',15);
text(6, .3,'300','color','m','FontSize',13);
plot(r,P310,'k.','MarkerSize',20);
text(13, .13,'310','color','k','FontSize',13);
plot(r,P320,'c.','MarkerSize',20);
text(10, .13,'320','color','c','FontSize',13);
hold off
```
MarkerSize similar to FontSize allows us to change the size of the marker symbols. Note that here I have used different sizes to show how the numbers affect the size.

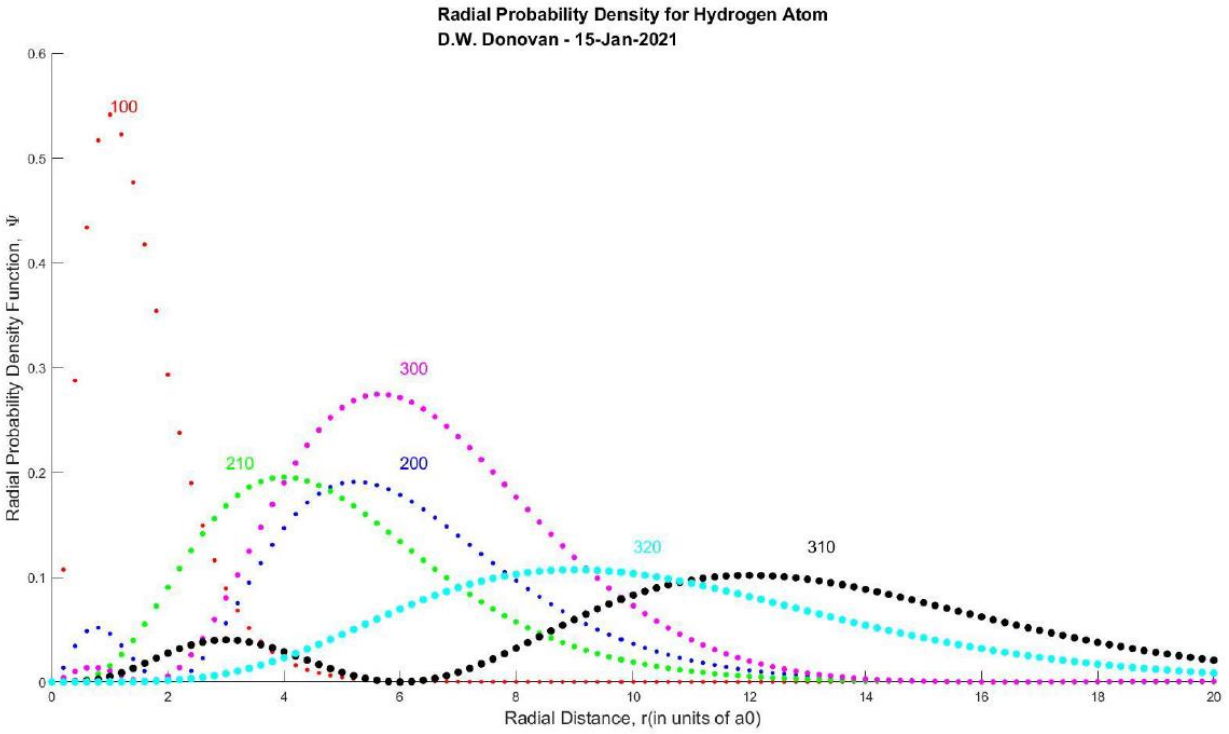
nr=r';	Create a new vector that is the transpose of r. Since r was a column vector, the new vector is now a row vector.
rexp100=(nr*P100)*.2 rexp200=(nr*P200)*.2 rexp210=(nr*P210)*.2 rexp300=(nr*P300)*.2 rexp310=(nr*P310)*.2 rexp320=(nr*P320)*.2	Calculating the expected positions of the electron in the given state. Note the lack of periods causes these to be Matrix multiplications. A scalar such as .2 is multiplied the way a scalar normally multiplies a matrix. Compare this equation with the expression for expected position given above. Note the .2 is ?r! Note lack of semicolons causes these results to printed in the workspace. This allows one then to see these values.
%{ rexp100 = 1.5001 rexp200 = 6.0000 rexp210 = 5.0000 rexp300 = 9.7457 rexp310 = 12.4741 rexp320 = 10.4922 %}	%{ causes the lines following to all be comments. It saves having to comment each and every line. To stop this you place a %}. I have pasted the workspace results of my expectation values here. I find it useful to put solutions to m-file problems in the m-file but commented out so that everything is in one place. All work requiring MATLAB needs to have the m-file as well as any plots and results turned in. The plots should be on full size pages, the answers should be commented as shown here.

Outputs from Program

Text:

```
%{
rexp100 =  1.5001
rexp200 =  6.0000
rexp210 =  5.0000
rexp300 =  9.7457
rexp310 = 12.4741
rexp320 = 10.4922
%}
```


Radial Wave Functions for Hydrogen Atom
D.W. Donovan - 15-Jan-2021
Radial Wave Function, R
Radial Distance, r(in units of a0)
100
200
300
210
310 ->
320
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
0
1
2
3
4
5
6


Radial Probability Density for Hydrogen Atom
D.W. Donovan - 15-Jan-2021
Radial Probability Density Function, Ψ
Radial Distance, r(in units of a0)
100
300
210
200
320
310
0
0.1
0.2
0.3
0.4
0.5
0.6
0
2
4
6
8
10
12
14
16
18
20

Example #2 - Performing Linear Regression on a set of Data

Assume you have a set of data that plots or should plot to a straight line. One would like to find the slope and y-intercept of the best straight line. If there are only two points on the line the slope and y-intercept are easily determined. If there are more than two points than a least squares approach must be used. In general one is trying to determine the coefficients from an equation of the form :

$$y = c_0 x^0 + c_1 x^1 + c_2 x^2 + \ldots$$

for a straight line this becomes :

$$y = c_0 + c_1 x$$

This can be written in a matrix for as:

$$Y = N*C$$

where Y is a column matrix of the y data values, C is a column matrix of the coefficients, and N is a matrix whose columns are the powers of the x data points. Column 1 is therefore each x data point raised to the zero power and thus must be all 1's. Column 2 is the x data raised to the first power or the x data points alone. If we were fitting to an arbitrary polynomial, we would have additional columns that would be higher order powers of the x data points.

To find the coefficients then we solve the above matrix equation for C. Thus:

```
C = N\Y
```

Note c_0 is the y intercept and c_1 is the slope of our best straight line.

Type in and experiment with the following m file.

regrexmpw2011.m

```
%Program to demonstrate how MATLAB does basic linear Regression on a set
%of data which is assumed to have a linear relationship
%version 2011-01-10 D.W. Donovan

clear all;

y =   [15.4500 ;
      15.6000 ;
      15.6000 ;
      15.7000 ;
```

```
    15.1500 ;
    15.0500 ;
    15.3000 ;
    15.3500 ;
    14.8000 ;
    14.9000 ;
    14.9000 ;
    14.7500 ;
    14.3000 ;
    14.4500 ;
    14.4000 ;
    14.5500 ;
    14.1500 ;
    14.0000 ;
    14.3000 ;
    13.9000 ;
    13.8500 ;
    13.8000 ;
    13.1500 ;
    12.1000 ;
    11.6500];

x =   [0.0981; 0.2211; 0.2460; 0.2730; 0.3372;
    0.4643; 0.5442; 0.6324; 0.6577; 0.6978;
    0.7252; 0.7262; 0.7988; 0.8228; 0.8741;
    0.9241; 1.0144; 1.0661; 1.0940; 1.1166;
    1.1294; 1.2240; 1.5043; 1.8182; 2.1038];

n(:,2) = x;
n(:,1)=ones(size(x));
c=n\y;

slope = c(2)
yintercept = c(1)

sp='The Slope is ';
s=num2str(slope);
st=[sp,s];

si='The Y intercept is ';
i=num2str(yintercept);
sti=[si,i];
```

```
xp= [min(x):(max(x)-min(x))/100 :max(x)]';
yp= c(2)*xp+c(1);

figure;
hold on
plot (x,y,'r*');
plot (xp,yp,'b-','LineWidth',3);
plot (max(x), max(y),'w.')
plot (max(x), max(y),'w.')
legend('Data','Fitted Line',st,sti,'Location','SouthWest')
legend('boxoff')
t1='Example demonstrating linear regression of data';
t3='D.W. Donovan -';
t0=' \newline ';
t=[t1,t0,t3,date];
title (t, 'FontSize', 15)
xlabel('x in unitless numbers','FontSize',13)
ylabel('f(x) in unitless numbers', 'FontSize',13)

%{
slope = -2.0332

yintercept = 16.1632
%}
```

regrexmpw2011.m – Explained

Note: I am not going to explain commands already explained above. If you have questions on these, of course come and ask me. I will be explaining new things in this program.

```
%Program to demonstrate how MATLAB does basic linear Regression on a set
%of data which is assumed to have a linear relationship
%version 2011-01-10 D.W. Donovan

clear all;

y =   [15.4500 ;            The y values are entered as a column matrix
      15.6000 ;
      15.6000 ;
      15.7000 ;
      15.1500 ;
      15.0500 ;
      15.3000 ;
      15.3500 ;
      14.8000 ;
```

```
14.9000 ;
14.9000 ;
14.7500 ;
14.3000 ;
14.4500 ;
14.4000 ;
14.5500 ;
14.1500 ;
14.0000 ;
14.3000 ;
13.9000 ;
13.8500 ;
13.8000 ;
13.1500 ;
12.1000 ;
11.6500];
```

```
x =  [0.0981; 0.2211; 0.2460; 0.2730; 0.3372;
     0.4643; 0.5442; 0.6324; 0.6577; 0.6978;
     0.7252; 0.7262; 0.7988; 0.8228; 0.8741;
     0.9241; 1.0144; 1.0661; 1.0940; 1.1166;
     1.1294; 1.2240; 1.5043; 1.8182; 2.1038];
```

The x values are also entered as a column matrix however, you cannot take up so much space in both the workspace, or on a page to turn in. The ; operator also serves the purpose of telling MATLAB that the current row is done. Here I am still providing 25 entries but I use 5 rows on a page rather than 25. Be very careful though to ensure that your x and y values are identical in number. MATLAB will complain if you do not have identical number of elements when you do matrix operations, plot, etc.

```
n(:,2) = x;
```

N is the matrix made up of the powers of the x data. The second column is declared first to establish the size of the matrix. Since we are doing linear regression the matrix will be a k by 2 matrix where k is the total number of data points involved. The first column is the x to the zero power and thus will be all ones. The second column is x to the first power and thus is the x values. By creating the matrix this way, we do not have to know in advance how many elements are being used. This makes this technique very portable.

```
n(:,1)=ones(size(x));
```

ones is a MATLAB function that provides a matrix of all ones automatically. Here the size function makes sure that enough ones are generated to match the number of x values present. Note: these two lines allow one to calculate the regression without having to know the exact number of data points available.

```
c=n\y;
```

This does the matrix equation that determines the two c coefficients.

```
slope = c(2)
```

Identify the slope as the second element of the matrix

```
yintercept = c(1)
```

Identify the y intercept as the first element of the matrix. Note the absence of ; in the previous two lines cause the slope and y intercept to be printed in the workspace.

```
sp='The Slope is ';
s=num2str(slope);
st=[sp,s];

si='The Y intercept is ';
i=num2str(yintercept);
sti=[si,i];
```

Creating a string variable to enable you to print on the graph the values of the slope and y-intercept.

```
xp= [min(x):(max(x)-min(x))/100 :max(x)]';
```

Create a new range of x data for plotting the resulting Best Straight line. This allows you to create line without knowledge of the actual range of x values. Note the 100 sets a resolution for your data. Sometimes when fitting rapidly changing variables, such as trig functions, it sometimes might be necessary to use a value like 1000, or even more.

```
yp= c(2)*xp+c(1);
```

Create the y values for the line

```
figure;
hold on
plot (x,y,'r*');
plot (xp,yp,'b-','LineWidth',3);
plot (max(x), max(y),'w.')
plot (max(x), max(y),'w.')
```

Plot the original data points

Plot the Best Straight line as calculated by the regression

LineWidth allows you to change thickness of line. This is similar to MarkerSize and FontSize mentioned above.

The legend command below requires a plot for every element printed in the legend. Since we are plotting raw data, regression data and we want to print the slope and y-intercept, we need four plots. These last two are single points chosen to be away from actual data. Notice that the data has a negative slope. Thus as x gets larger, y gets smaller. When y is large, x is not. The max ( ) function chooses the largest of the x data min( ) chooses the smallest. Often one should use max(x), min(y) to insure that you do not plot over real data. The w. says plot a white dot which is not visible on a white background. So we get our needed plots without adding artifacts.

```
legend('Data','Fitted Line',st,sti,'Location','SouthWest')
```

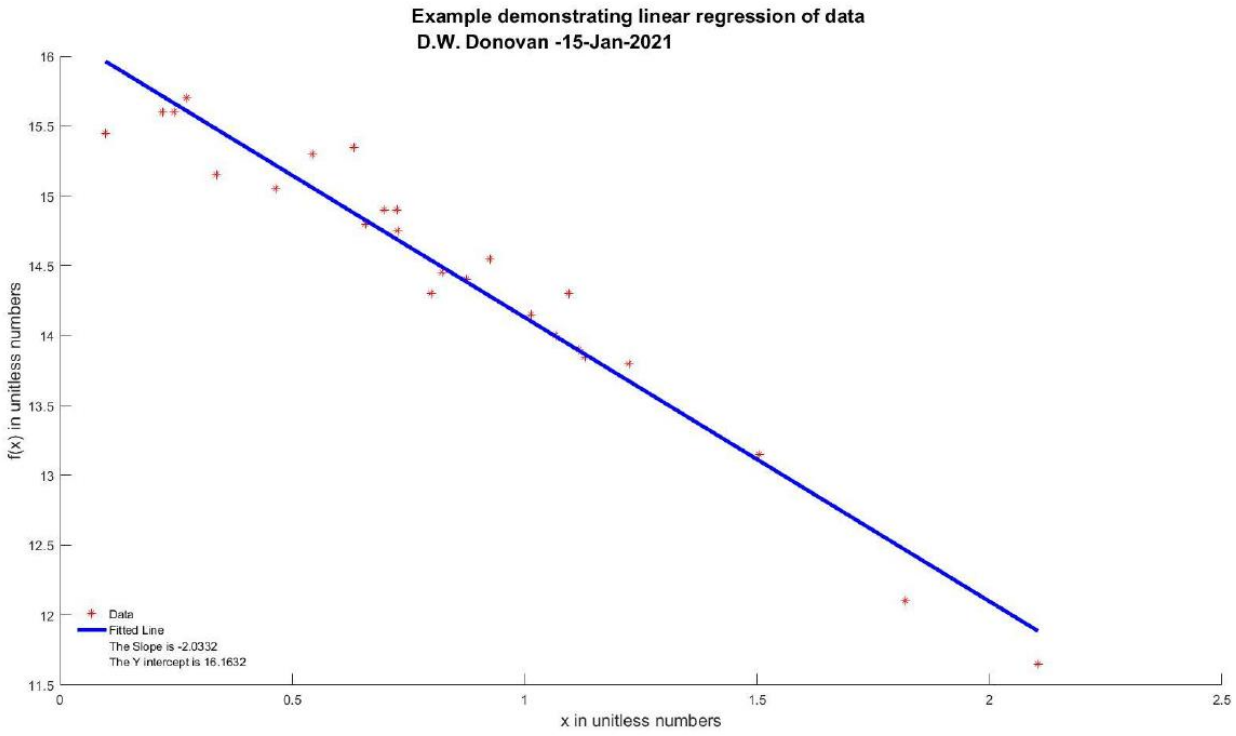
Produces a legend box. Note first two strings get assigned symbols used in the plot. Next two just get printed giving us slope and
y-ntercept. The Location tells MATLAB where to place the legend. SouthWest indicates the lower left hand corner. Often you will need to plot first and see where the legend is least obscuring of your plot.

```
legend('boxoff')
```
Causes the normal box drawn around the legend to be removed.

```
t1='Example demonstrating linear regression of data';
t3='D.W. Donovan -';
t0=' \newline ';
t=[t1,t0,t3,date];
title (t, 'FontSize', 15)
xlabel('x in unitless numbers','FontSize',13)
ylabel('f(x) in unitless numbers', 'FontSize',13)

%{
slope = -2.0332
```
Here we have commented out a place in the m-file to provide the calculated slope and y-intercept.

```
yintercept = 16.1632
%}
```

Output of example



Example #3 - Solving a Set of Simultaneous Equations

Often in Physics, a set of simultaneous equations must be solved. MATLAB makes this particularly easy since matrix methods for this have existed for a long time. In matrix terms a set of simultaneous equations can be written as:

AX = B

where A is the matrix made of the coefficients of the variables usually defined as x_1, x_2, .. , x_n. and B is the matrix made up of the constant terms.

To solve for X in MATLAB one solves the equation:

X = A\B.

$$\boldsymbol{3x_1 - 5x_2 + 7x_3 + 4x_4 - x_5 = 20}$$

$$\boldsymbol{4x_1 + 6x_2 - 13x_3 - 3x_4 + 2x_5 = -14}$$

$$\boldsymbol{x_2 + x_3 + x_4 - x_5 = 3}$$

$$\boldsymbol{x_1 - 4x_3 + 3x_4 = 7}$$

$$\boldsymbol{18x_1 - 10x_2 + 6x_5 = -4}$$

Type in the following m file and experiment with it.

simlexw2011.m - The Listing

```
%Program to demonstrate how MATLAB solves Simultaneous Equations
%version 2011-01-10 D.W. Donovan
clear all;

a=[3 -5 7 4 -1;
  4 6 -13 -3 2;
  0 1 1 1 -1;
  1 0 -4 3 0;
  18 -10 0 0 6];
b=[20 -14 3 7 -4]';

x=a\b;

x1=x(1)
```

```
x2=x(2)
x3=x(3)
x4=x(4)
x5=x(5)

test=a*x
xs1='x1';
xs2='x2';
xs3='x3';
xs4='x4';
xs5='x5';
ANS={'variable' 'value' 'Exp B' 'Calc B';
   xs1 x1 b(1) test(1);xs2 x2 b(2) test(2);xs3 x3 b(3) test(3);
   xs4 x4 b(4) test(4);xs5 x5 b(5) test(5)};
ANS

%{
ANS =

   'variable'    'value'      'Exp B'    'Calc B'
   'x1'         [  1.7152]    [   20]    [ 20.0000]
   'x2'         [ -8.3273]    [  -14]    [-14.0000]
   'x3'         [ -4.3394]    [    3]    [  3.0000]
   'x4'         [ -4.0242]    [    7]    [       7]
   'x5'         [-19.6909]    [   -4]    [      -4]
%}
```

simlexw2011.m - Explained

```
%Program to demonstrate how MATLAB solves Simultaneous Equations
%version 2008-01-16 D.W. Donovan
clear all;

a=[3 -5 7 4 -1;          Assigning the coefficients to a matrix “a”

  4 6 -13 -3 2;
  0 1 1 1 -1;
  1 0 -4 3 0;
  18 -10 0 0 6];
b=[20 -14 3 7 -4]';      Assigning the constants to a column matrix “b”

x=a\b;                   Solve for the x variables

x1=x(1)                  List the x variables in the workspace by leaving the ‘;’ out
x2=x(2)
```

```
x3=x(3)
x4=x(4)
x5=x(5)

test=a*x                Check the solution by performing the matrix multiplication with the original
                        coefficients and the new values to compare with the original constants.
xs1='x1';               Creating string variables to make an answer grid.
xs2='x2';
xs3='x3';
xs4='x4';
xs5='x5';
ANS={'variable' 'value' 'Exp B' 'Calc B';    '{' allow different types of variables to be grouped together
                                                in a matrix, while '[' only allow identical types to be
                                                grouped.  Here I am mixing strings and numbers.
   xs1 x1 b(1) test(1);xs2 x2 b(2) test(2);xs3 x3 b(3) test(3);      I prevent any printout until I have it all
   xs4 x4 b(4) test(4);xs5 x5 b(5) test(5)};
ANS                                             Now we print out the final answer

%{                              Again putting our results directly into the m-file to keep answers
                                                conveniently located.
ANS =

   'variable'    'value'      'Exp B'    'Calc B'
   'x1'         [  1.7152]    [   20]    [ 20.0000]
   'x2'         [ -8.3273]    [  -14]    [-14.0000]
   'x3'         [ -4.3394]    [    3]    [  3.0000]
   'x4'         [ -4.0242]    [    7]    [       7]
   'x5'         [-19.6909]    [   -4]    [      -4]
%}
```