MATLAB Workshop 15 - Linear Regression in MATLAB

Objectives: Learn how to obtain the coefficients of a "straight-line" fit to data, display the resulting

equation as a line on the data plot, and display the equation and goodness-of-fit statistic on the graph.

MATLAB Features:

data analysis

Command	Action
polyfit(x,y,N)	finds linear, least-squares coefficients for polynomial
polylle(x,y,N)	equation of degree n that is best fit to the (x,y) data set.

graphics commands

Command	Action
<pre>plot(x,y,symbol)</pre>	creates a pop up window that displays the (x,y) data points specified on linearly-scaled axes with the symbol (and color) specified in the <i>string variable</i> symbol. The data points are supplied as separate \mathbf{x} and \mathbf{y} vectors. MATLAB automatically scales the axes to fit the data.
semilogy(x,y,symbol)	creates a pop up window that displays the (x,y) data points specified on a graph with the y -axis scaled in powers of 10 and the x -axis scaled linearly with the symbol (and color) specified in the $string\ variable\ symbol$. The data points are supplied as separate x and y vectors. MATLAB automatically scales the axes to fit the data.
loglog(x,y,symbol)	creates a pop up window that displays the (x,y) data points specified on a graph with both the x- and y-axes scaled in powers of 10 with the symbol (and color) specified in the string variable symbol. The data points are supplied as separate \mathbf{x} and \mathbf{y} vectors. MATLAB automatically scales the axes to fit the data.
<pre>xlabel(xname)</pre>	adds the text in the <i>string variable</i> xname below the <i>x</i> -axis.
ylabel(yname)	adds the text in the string variable yname below the y-axis.
title(graphname)	adds the text in the string variable graphname above the plot.
axis('equal')	forces equal-scaling on the x- and y-axes
hold on	maintains current plot for additional plotting overlay
hold off	turns off hold on
<pre>text(X,Y,'string')</pre>	displays 'string' at (X, Y)-coordinates on current plot
gtext('string')	displays 'string' at plot location designated by cross-hairs

grap<u>h symbol options</u>

Graph Symbol Options							
	Color		Symbol	Line			
Y	yellow	•	point	-	solid line		
m	magenta	0	circle	:	dotted line		
С	cyan	x	x-mark		dash-dot line		
r	red	+	plus		dashed line		
g	green blue	*	star				
b	blue	ន	square				
w	white	đ	diamond				
k	black	v	triangle (down)				
		٨	triangle (up)				
		<	triangle (left)				
		>	triangle (right)				
		р	pentagram				
		h	hexagram				

• Textbook costs

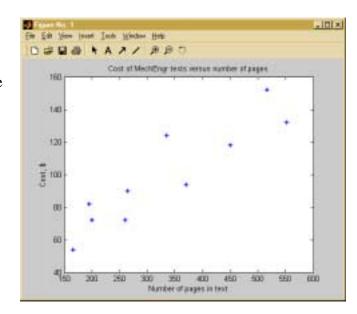
Concerned about the ever rising cost of textbooks, an engineering student decided to see whether the cost of textbooks in a particular subject was related to the number of pages. He went to the bookstore and found the following data for 10 mechanical engineering books:

Mechanical Engineering textbook cost versus number of pages										
Number of pages	166	195	200	260	265	335	370	450	517	552
Cost, \$	54.00	82.00	72.00	72.00	90.00	124.00	94.00	118.00	152.00	132.00

Using the MATLAB script developed in Workshop 14, the engineer produced the plot shown at the right. The data does look as if it fits a linear relationship. Several questions arise. The first is what are the appropriate values for the coefficients a_1 and a_0 in the linear equation,

$$C = a_1 P + a_0$$

where C is the textbook cost, \$, and P is the number of pages, that best describes the data. A second question is what does this line look like when plotted with the data. A third question is how well does the line actually represent the data.



(1) Create a plot of cost versus number of pages.

Create a data file containing the data. Use your script from Workshop 14 to create a figure showing the data points as illustrated above.

Check to see what variables are in the Workspace by typing who at the command prompt. You should have at least xdat, ydat, symbol, xname, yname, and graphname. Why?

(2) MATLAB connects the dots.

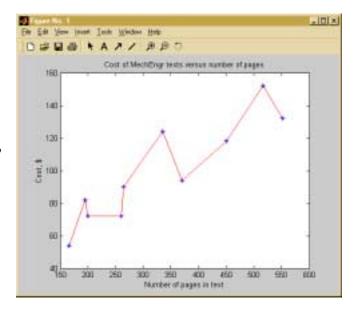
Because the graph variable information is present in the Workspace, we can use the Command Window to illustrate some more features of graphs and graph management in MATLAB.

What would happen if we let MATLAB "draw a line" for the data points? To observe this, enter

- » hold on
- » plot(xdat,ydat,'r-')

at the command prompt. The hold command is used to manage figure display. hold on says to keep the current figure and superimpose any additional plot commands on top of it. hold off says to replace the current figure with whatever the next plot command dictates. In this case, the plot command asks that the same data be plotted, but this time with a red line. The figure at the right results.

MATLAB by itself will "connect the dots" - not very useful if we are trying to find an equation that relates the cost to the number of pages.



Lets return to the original data plot. Unfortunately, there is no undo command that will remove the line just added. You will have to replace the figure - but it can be done from the Command Window by issuing the following commands (why?).

- » hold off
- » plot(xdat,ydat,symbol)
- » xlabel(xname)
- » ylabel(yname)
- » title(graphname)

Fitting a line to data

Many methods exist for finding a "best fit" line or curve to some data. One of the most popular is called least squares regression or linear regression. For a straight-line approximation, we are seeking the line

$$y = a_1 x + a_0$$

that best approximates the data. If we knew the values for a_1 and a_0 , we could estimate the y-values for each of the data points by

$$(yest)_i = a_1(xdat)_i + a_0$$

where i refers to an individual data point. The error associated with the estimate is defined as the vertical distance between the data point and the proposed line, i.e.,

$$e_i = (ydat)_i - (yest)_i$$

were e_i is the error. Linear regression finds values for a_1 and a_0 by a mathematical procedure that minimizes the sum of the error-squared for all of the data points.

(3) Least squares in MATLAB.

Because fitting a line to data is such a common activity, MATLAB has a single command that will find the estimates,

where coeff is a variable that will capture the coefficients for the best fit equation, xdat is the x-data vector, ydat is the y-data vector, and n is the degree of the polynomial line (or curve) that you want to fit the data to. A straight line is a 1st-degree polynomial, so the value for n would be 1.

Find the "best fit " to the book data by entering

```
» coeff = polyfit(xdat,ydat,1)
coeff =
    0.2048    31.2181
```

MATLAB responds with the coefficient vector in the order [a₁ a₀]. (How would you suppress display of coeff?) Thus, according to MATLAB and the least squares procedure, the best fit equation for the line representing a linear relation between the cost of a Mechanical Engineering text and the number of pages is

```
C = 0.2048P + 31.2181
```

(4) Displaying the best fit on the data graph.

Visual confirmation that the "best fit" equation is indeed representative of the data comes next. There are two problems at the moment. The first is that we have the coefficients for the equation, but not the *x*- and *y*- vectors that are required for the plot command. The *x*- and *y*-vectors will need to be generated.

This brings us to the second problem. Remember that MATLAB uses "connect the dots" for creating a line. If the plot points for the data are far apart, the line might have angles and corners and not appear smooth. In order to counter this, we need to use a large number of points when plotting a line. This will make any point to point distance small and make the resulting "connect the dots" picture look smooth. Generally 200 points are sufficient, but you might want to use more.

Thus, the steps that we need to follow to create a smooth line fit to the data are to

- 1. define a vector of 200 x-points in the range of the data
- 2. calculate the corresponding vector of *y*-points
- 3. display the x- and y-points as a line in the figure.

To see how this works, enter the following at the command prompt

```
» xline = linspace( min(xdat), max(xdat), 200);
```

What does the linspace command do? The min command? The max command? Why does this work to create a vector of x-values that span the data domain? Note that the variable name xline is being used to distinguish this vector from the data vector.

Now enter

```
» yline = coeff(1)*xline+coeff(2);
```

This command creates a vector of *y*-values corresponding to the best fit equation. Why?

We can now plot the best fit line

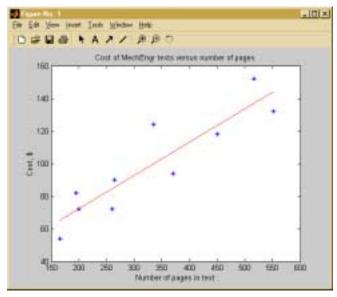
```
» hold on
```

```
» plot(xline,yline,'r-')
```

The result, displayed at the right, shows that the best-fit calculated by the polyfit command is a reasonable representation of the data. The next question is: how good?

• Error and goodness-of-fit estimation

As engineers, we should always be interested in knowing how close our approximations (in this case, the line) actually come to the measured, physical reality. As can be seen in the approximation at the right, only one data point actually seems to fall on or near the line!



The first question we can ask is what is the absolute error associated with the fit. This can be calculated as

$$e_i = |(ydat)_i - (yest)_i|$$

for each data point. Note that the absolute error treats positive and negative deviations of the data from the line in the same manner. In MATLAB code, this becomes

```
» yest = coeff(1)*xdat+coeff(2);
» abs_error = abs(ydat-yest);
```

Given abs_error, we can extract the magnitude of the maximum absolute error and data point at which it occurred by using a variation of the max command:

```
» [max_abs_error, maxpt] = max(abs_error);
```

max_abs_error will have the value of the maximum absolute error and maxpt will be the index where it is found in abs error. For the plot above,

```
max_abs_error =
    24.1809
maxpt =
    6
```

Thus the maximum error is found at the sixth data point (xdat = 335 where did this come from?). How could you find the minimum absolute error?

The absolute error provides the magnitude of the error. However, this does not tell us how serious the error actually is. For example, which is better: an absolute error of 50 units relative to an expected value of 100 units or an absolute error of 50 units relative to an expected value of 5000 units. Both have the same absolute error. But the percentage error in the first case is 50% while it is only 1% in the second case.

Relative error is like a percentage error in that how large the error is compared with the expected error. Relative error is sometimes referred to as the fractional error because it is obtained by dividing the absolute error by the magnitude of the corresponding *y*-value. The MATLAB command to do element by element division is

```
» rel_error = abs_error./ydat;
```

How would you find the greatest relative error and the location at which it occurs?

A commonly used statistic that is related to the error, but is not the same as the error is the goodness-of-fit r^2 (r-squared) statistic. The r^2 statistic ranges from a value of 0 for absolutely no relation between the data and the line to a value of 1 which occurs only if all of the data fall exactly on the line, i.e., no error. In some engineering disciplines, an equation fitted to data is acceptable only if $r^2 > 0.9$. Other engineering disciplines might find an r^2 as low as 0.7 acceptable for use.

The r² statistic is calculated from

$$r^{2} = 1 - \frac{SSE}{SST}$$
where
$$SSE = \sum_{i=1}^{n} [(ydat)_{i} - (yest)_{i}]^{2}$$
and
$$SST = \sum_{i=1}^{n} [(ydat)_{i} - y_{ave}]^{2}$$

MATLAB implementation of these equations is straight forward. For example, what (single) MATLAB command would you use to compute the average value of the y-data? The r^2 statistic for the text book cost versus number of pages fit is $r^2 = 0.8204$.

(5) Calculate the various error estimates.

Implement the MATLAB commands (in the Command Window) to find the following

- 1) Maximum absolute error
- 2) Index of the value where the maximum absolute error was found.
- 3) X-data point where maximum absolute error was found.
- 4) Maximum relative error
- 5) Index of the value where the maximum relative error was found.
- 6) X-data point where maximum relative error was found.
- 7) r² statistic for the fit.

• Displaying equation and r² statistic on the graph

The final bell and whistle in displaying data and a best line fit to the data on a graph is to also display the equation and r^2 statistic as text. In order to do this, we need to build both the equation and r^2 as a string variables for display. The equation can be built from the following commands

```
» alstr = num2str(coeff(1));
» a0str = num2str(coeff(2));
» eqnstr = ['y = (', a1str, ')*x + (', a0str, ')'];
```

where the first two command convert the numbers for the equation coefficients to their equivalent strings. The third command creates a string variable with the text and coefficients in order. The r² statistic string can be built by the commands

```
» rsqstr = ['r<sup>2</sup> = (', num2str(rsq)];
```

This command used the **num2str** command internally to create the string rather than create another variable to hold the conversion. The process of building the strings part by part is referred to as concatenation.

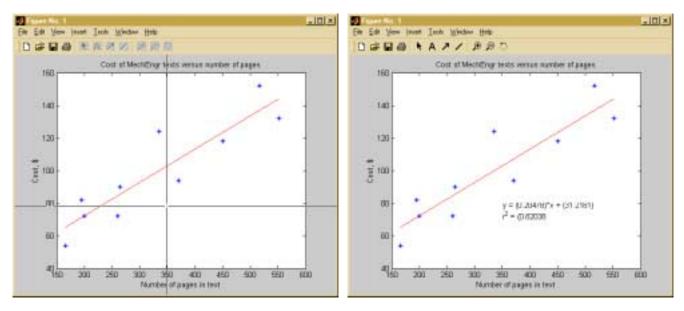
Both the equation and the r^2 statistic can be displayed by using the text command:

```
text(X,Y,'text to display')
```

where \mathbf{x} and \mathbf{y} are the (x,y)-coordinates on the current plot at which to start the text string. As always, the text string can be a string variable name.

```
An alternative is to use the gtext command
```

```
gtext({eqnstr,rsqstr})
```



This causes a "cross-hairs" to appear on the plot, as shown above and to the right, which can be moved by moving the mouse. A left-click on the mouse will cause the requested strings to be placed at the location of the cross-hairs as shown in the figure above and to the left. Note how the r² equation string is displayed with the number 2 showing as an exponent. Why?

(6) Display equation on graph.

- 1) Display the equation and r^2 statistic on the current graph using text.
- 2) Display the equation and r² statistic on the current graph using gtext.

Exercises:

- 1. Modify your linearplot function from Workshop 14 so that it will now
 - a Display the data points (as previously);
 - b Calculate a best-fit line to the data;
 - c Display the best-fit line as a line only;
 - d Calculate the r² statistic;
 - e Display the equation and r^2 statistic on the plot;
 - Note: you should tell the user what to do if you use gtext.
 - f Return the equation coefficients and r^2 statistic to the calling function.
- 2. Test your modified function by running your script from Workshop 14 and reproducing the graphs of this workshop.

Recap: You should have learned

- That MATLAB uses "connect-the-dots" to draw lines between points.
- How to use **polyfit** to find a best fit straight line to data.
- How to display a best fit straight line to data on the same plot as the data.

- That many points are required to have a smooth line displayed in MATLAB.
- The meaning of and how to calculate absolute error.
- How to find maximum and minimum absolute error and their *x*-location.
- The meaning of and how to calculate relative error.
- How to find maximum and minimum relative error and their *x*-location.
- How to calculate the r^2 statistic.
- How to display text strings on the plot.