Lines of Best Fit: Linear Programming

Doug Hundley
Whitman College

October 11, 2005

We consider constructing the line of best fit using errors other than the standard least squares error. In particular, we consider what line is obtained by using the uniform norm and the sum of absolute values. *These are particularly useful as case studies in how to construct linear programming problems.*

Before we begin, let's get some intuition about these measures of error.

1 The mean, the median and optimization

Let $\{x_1, x_2, \ldots, x_n\}$ be a set of n real numbers.

Exercise: Define $E(m) = \sum_{i=1}^{n}(m - x_i)^2$. Show, using calculus, that the number m that minimizes E is given by the mean of the data set.

Absolute Value and the Derivative

We know that, if $f(x) = |x|$, then the derivative of f can be written as:

$$f'(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases} = \frac{x}{|x|} \doteq \text{signum}(x)$$

Similarly, if $f(m) = |m - x_i|$ (x_i is a real number), then

$$\frac{df}{dm} = \begin{cases} -1 & \text{if } m < x_i \\ 1 & \text{if } m > x_i \end{cases}$$

Continuing, if $f(m) = |m - x_1| + |m - x_2|$, where $x_1 < x_2$, then we can rewrite the absolute value as:

$$f(m) = \begin{cases} (m - x_1) + (m - x_2) & \text{if } m \geq x_2 \\ (m - x_1) - (m - x_2) & \text{if } x_1 \leq m < x_2 \\ -(m - x_1) - (m - x_2) & \text{if } m < x_1 \end{cases} \quad \Rightarrow \quad \frac{df}{dm} = \begin{cases} 2 & \text{if } m > x_2 \\ 0 & \text{if } x_1 < m < x_2 \\ -2 & \text{if } m < x_1 \end{cases}$$

And another example, since the derivative changes slightly with an odd number of points: $f(m) = |m - x_1| + |m - x_2| + |m - x_3|$, with $x_1 < x_2 < x_3$:

$$\frac{df}{dm} = \begin{cases} 3 & \text{if } m > x_3 \\ 1 & \text{if } x_2 < m < x_3 \\ -1 & \text{if } x_1 < m < x_2 \\ -3 & \text{if } m < x_1 \end{cases}$$

In this last case, we might define the derivative at x_2 to be zero, since that is where f' jumps from 1 to -1.

Exercise: Extend the preceding discussion to show that, given $\{x_1, x_2, \ldots, x_n\}$ with $x_1 < x_2 < \ldots < x_n$, then the median of the data minimizes the error:

$$E(m) = \sum_{i=1}^{n} |m - x_i|$$

Hint: Consider $\frac{dE}{dm}$ in some sample intervals. It might be helpful to consider the cases where n is even, then when n is odd.

Minimize the Maximum

In this case, we want to explore the error we get by using the $\infty-$norm,

$$E(m) = \max_{1 \leq i \leq n} |m - x_i|$$

Before continuing, you might first consider the graphs of $|m - x_i|$. They are all "V-shaped", with the vertex at $(x_i, 0)$. In particular, plot these by hand for three x_i so that $x_1 < x_2 < x_3$. You should convince yourself that the maximum will be the line $-(m - x_3)$ until it intersects with the line $m - x_1$, so that E itself has a "V-shaped" graph. By solving for the vertex, you should find that the m that minimizes this error is $m = \frac{x_1 + x_3}{2}$.

Given a set of ordered n real numbers, use the preceding discussion to argue that the m that minimizes $E(m)$ is given by $m = \dfrac{x_1 + x_n}{2}$.

2 Uniform Norm

In this case, let the error function be defined by:

$$E(m,b) = \max_i \ |\epsilon_i| \doteq \epsilon_{\max}$$

By the definition of the max, we have the n constraints:

$$|\epsilon_i| \leq \epsilon_{\max}, \quad i = 1..n$$

which could be expressed using $\epsilon_i = y_i - mx_i - b$:

$$|y_i - mx_i - b| \leq \epsilon_{\max}, \quad i = 1..n$$

This is actually two inequalities per data point:

$$-\epsilon_{\max} \leq y_i - mx_i - b \leq \epsilon_{\max}, \quad i = 1..n$$

so that, given n data points, there will be $2n$ constraints, each pair of which will need to be written as:

$$\begin{aligned} mx_i + b - \epsilon_{\max} &\leq y_i \\ -mx_i - b - \epsilon_{\max} &\leq -y_i \end{aligned}$$

The unknowns here are the $\epsilon_{\max}$, m, b. We rewrite everything now in terms of a linear programming problem, where the objective function is explicit in terms of the unknowns:

$$\text{minimize} \quad 0 \cdot m + 0 \cdot b + \epsilon_{\max} \qquad s.t.$$

$$\begin{bmatrix} x_1 & 1 & -1 \\ x_2 & 1 & -1 \\ \vdots & & \vdots \\ x_n & 1 & -1 \\ -x_1 & -1 & -1 \\ -x_2 & -1 & -1 \\ \vdots & & \vdots \\ -x_n & -1 & -1 \end{bmatrix} \begin{bmatrix} m \\ b \\ \epsilon_{\max} \end{bmatrix} \leq \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ -y_1 \\ -y_2 \\ \vdots \\ -y_n \end{bmatrix}$$

In the Matlab implementation, we will want the program to automatically construct the matrix A and vector $\mathbf{b}$ for the `linprog` function. This can be quickly done using the following:

```
function [f,fopt]=unifbestline(x,y)

n=length(x);
A=zeros(2*n,3);  %Just sets up space for A
b=zeros(2*n,1);

A(1:n,1)=x(:);
A(n+1:2*n,1)=-x(:);
A(1:n,2)=ones(n,1);
A(n+1:2*n,2)=-ones(n,1);
```

```
A(:,3)=-ones(2*n,1);
b(1:n)=y(:);
b(n+1:2*n)=-y(:);
c=[0,0,1];

[f,fopt]=linprog(c,A,b);

%Plotting routine:
t=linspace(min(x),max(x));
yt=f(0)*t+f(1);
plot(x,y,'ro',t,yt,'k-');
```

The following script file will use our previous M-file `linreg` written for the least squares error and will compare the lines from the least squares versus the uniform norm:

```
%Script file to plot and compute errors for the lines of best fit using
%least squares and the uniform norm.

x=[10 20 30 40 50 60 70 80];
y=[25 70 380 550 610 1220 830 1450];
n=length(x);
[f,fopt]=unifbestline(x,y);
clf;
A=linreg2(x,y)
clf;

%Set up the plots:
t=linspace(min(x),max(x)); yt1=A.m*t+A.b; yt2=f(1)*t+f(2);
plot(x,y,'ro',t,yt1,'k-',t,yt2,'b-.');
legend('Data','Least Squares','Uniform Norm');

%List the epsilons as an array:
eps1=abs(y(:)-A.m*x(:)-A.b*(ones(n,1)));
eps2=abs(y(:)-f(1)*x(:)-f(2)*ones(n,1));

[x(:),eps1,eps2]
```

This returns the line:

$$y = 15.20x + 37.00$$

for the uniform error, and

$$y = 19.47x - 234.3$$

for the least squares error.

Here is the array of errors (the first column is the data in x, the second column is the ϵ_i for the least squares error, and the third column is the ϵ_i for the uniform error.

```
 Data:    Squares:  Uniform:
10.0000   64.5833  164.0000
20.0000   85.1190  271.0000
30.0000   30.1786  113.0000
40.0000    5.4762   95.0000
50.0000  129.2262  187.0000
60.0000  286.0714  271.0000
70.0000  298.6310  271.0000
80.0000  126.6667  197.0000
```

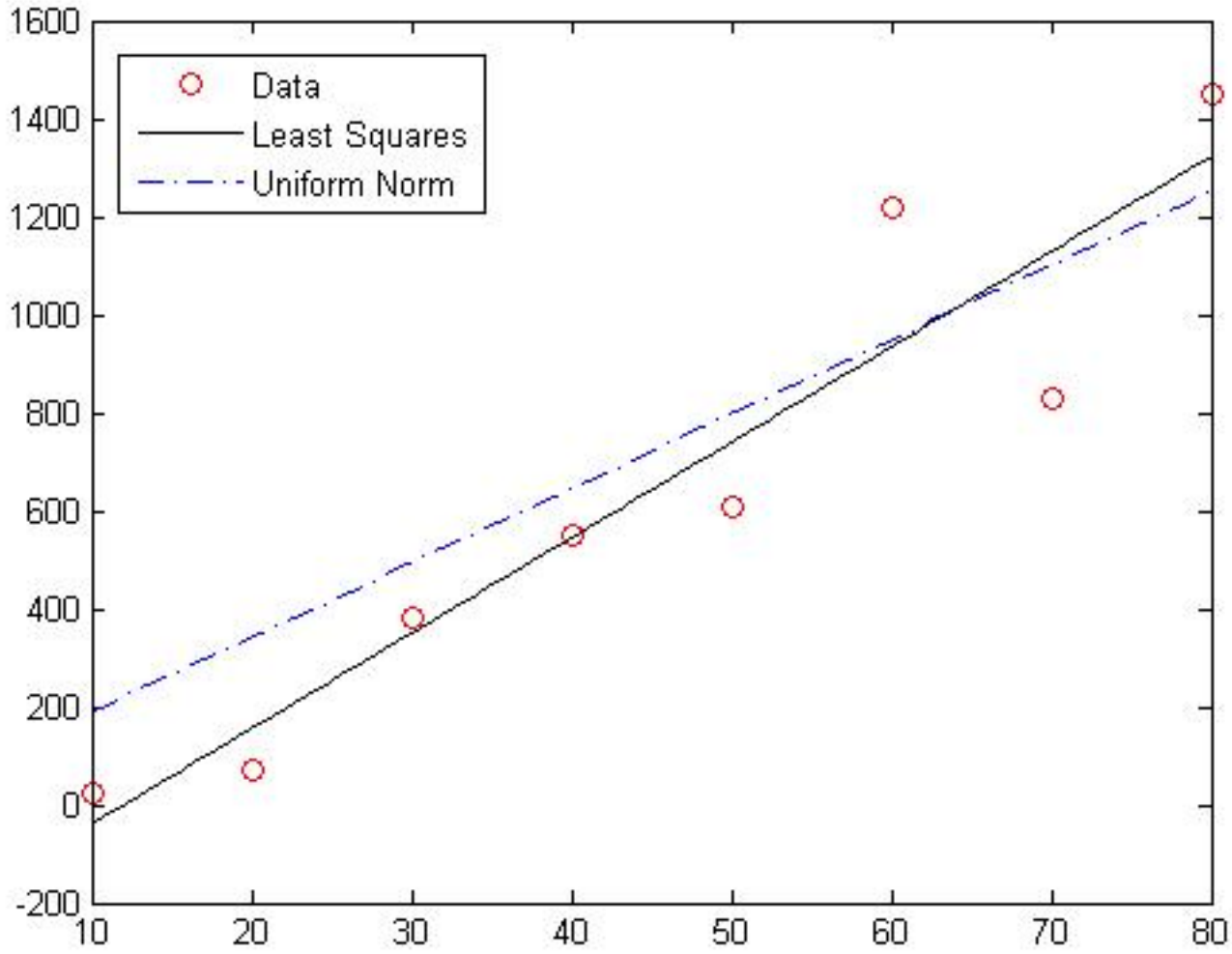

Figure 1: Comparing the Lines of Best Fit

3 The 1-norm

Using the 1−norm, we have an error function:

$$E(m,b) = \sum_{i=1}^{n} |\epsilon_i|$$

where again the $\epsilon_i = y_i - mx_i - b$.

The objective function for the linear program must be linear (and not involve the absolute value function). We get around this by creating a set of n new variables, a_i, where

$$a_i = |\epsilon_i| = |y_i - mx_i - b|$$

Here's the first key point for this problem: We will have an objective function with $2n+2$ variables. They are:

$$\mathbf{x} = [m,\ b,\ \epsilon_1,\ \ldots,\ \epsilon_n,\ a_1,\ \ldots,\ a_n]^T$$

and our objective function will be $\mathbf{c}^T\mathbf{x}$, where

$$\mathbf{c} = [0,\ 0,\ \underbrace{0,\ 0,\ \ldots\ 0}, \underbrace{1,\ 1,\ \ldots\ 1}]$$

Note the order in which the variables appear- That will be consistent in what is to follow. The braces are grouping the n constants for the ϵ_i, and the n constants for the a_i together.

The constraints will give the required relationships between these variables, and we will minimize the sum of the a_i.

We will loosen the definition of the a_i's somewhat so that:

$$\epsilon_i \le a_i \qquad \text{AND} \qquad -\epsilon_i \le a_i$$

This will give us $2n$ inequality constraints. This is imposed in Matlab by the matrix inequality $A\mathbf{x} \le \mathbf{b}$. In this case, the size of A will be $2n \times 2n+2$, the size of the unknown $\mathbf{x}$ is $2n+2 \times 1$, and the size of $\mathbf{b}$ is $2n \times 1$. The row of A corresponding to the constraint

$$\epsilon_1 - a_1 \le 0 \Rightarrow \left[0\ \ 0\ \ \underbrace{1\ \ 0\ \ 0 \ldots 0}\ \ \underbrace{-1\ \ 0\ \ \ldots 0}\right]$$

where the braces are grouping the coefficients for ϵ_i and a_i together respectively. Thus, taken together, the inequality constraints give a matrix A of the form (the separators are there just for easier reading):

$$A = \left[\begin{array}{cc|ccccc|cccc} 0 & 0 & 1 & 0 & \ldots & 0 & -1 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 1 & \ldots & 0 & 0 & -1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 1 & 0 & 0 & \ldots & -1 \\ \hline 0 & 0 & -1 & 0 & \ldots & 0 & -1 & 0 & \ldots & 0 \\ 0 & 0 & 0 & -1 & \ldots & 0 & 0 & -1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & -1 & 0 & 0 & \ldots & -1 \end{array}\right] \doteq \left[\begin{array}{c|c|c} 0 & I & -I \\ \hline 0 & -I & -I \end{array}\right]$$

Additionally, we will have the following **equality** constraints:

$$\epsilon_i = y_i - mx_i - b \qquad \text{or} \qquad mx_i + b + \epsilon_i = y_i$$

Writing these in matrix form $A_{\text{eq}}\mathbf{x} = \mathbf{b}_{\text{eq}}$ (NOTE: We need to keep the unknown vector $\mathbf{x}$ with length $2n+2$ even though we're only actually using the first $n+2$. This implies that A_{eq} has size $n \times 2n+2$.):

$$\left[\begin{array}{cc|cccc|ccc} x_1 & 1 & 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ x_2 & 1 & 0 & 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_n & 1 & 0 & 0 & \dots & 1 & 0 & \dots & 0 \end{array}\right] \left[\begin{array}{c} m \\ b \\ \epsilon_1 \\ \vdots \\ \epsilon_n \\ a_1 \\ \vdots \\ a_n \end{array}\right] = \left[\begin{array}{c} y_1 \\ y_2 \\ \vdots \\ y_n \end{array}\right]$$

Although these matrices may look complex, look at the patterns- they will be easy to construct in Matlab, and a function to accomplish this follows:

```
function [f,fopt]=onenormline(x,y)
n=length(x);

%Initialize the b's
b=zeros(2*n,1);
beq=y(:);

%Set up the matrices:
Z=zeros(n,2);
I=eye(n);
A=[Z,I,-I;Z,-I,-I];
Aeq=[x(:), ones(n,1), I, zeros(n,n)];

%The constant vector for the objective function:
c=[0;0; zeros(n,1); ones(n,1)];

[f,fopt]=linprog(c,A,b,Aeq,beq);

%Plotting routine:
t=linspace(min(x),max(x));
yt=f(1)*t+f(2);
plot(x,y,'ro',t,yt,'k-');
```

Changing our comparison script slightly:

```
x=[10 20 30 40 50 60 70 80];
y=[25 70 380 550 610 1220 830 1450];
```

```
n=length(x);
[f,fopt]=unifbestline(x,y);
clf;
A=linreg2(x,y)
clf;
[f2,fopt2]=onenormline(x,y);
clf;
%Set up the plots:
t=linspace(min(x),max(x));
yt1=A.m*t+A.b;
yt2=f(1)*t+f(2);
yt3=f2(1)*t+f2(2);
plot(x,y,'ro',t,yt1,'k-',t,yt2,'b-.',t,yt3,'k:');
legend('Data','Least Squares','Uniform Norm','1-Norm');

%List the epsilons as an array:
eps1=abs(y(:)-A.m*x(:)-A.b*(ones(n,1)));
eps2=abs(y(:)-f(1)*x(:)-f(2)*ones(n,1));
eps3=abs(y(:)-f2(1)*x(:)-f2(2)*ones(n,1));

[x(:),eps1,eps2,eps3]
```

Gives a line of best fit of $y = 20.4966x - 254.34$, and the Matlab output is on the next page. In particular, consider the residuals.

The following is a summary of the overall error for each of the three lines under each of the three methods. As expected, each method minimizes its error value:

	$\sum \lvert\epsilon_i\rvert$	$\sum \epsilon_i^2$	$\max \lvert\epsilon_i\rvert$
1-Norm	1015	226010	350.43
Least Square	1026	216120	298.63
Unif Norm	1569	342790	271

```
10.0000   64.5833  164.0000   74.3751
20.0000   85.1190  271.0000   85.5913
30.0000   30.1786  113.0000   19.4424
40.0000    5.4762   95.0000   15.5239
50.0000  129.2262  187.0000  160.4903
60.0000  286.0714  271.0000  244.5434
70.0000  298.6310  271.0000  350.4229
80.0000  126.6667  197.0000   64.6107
```

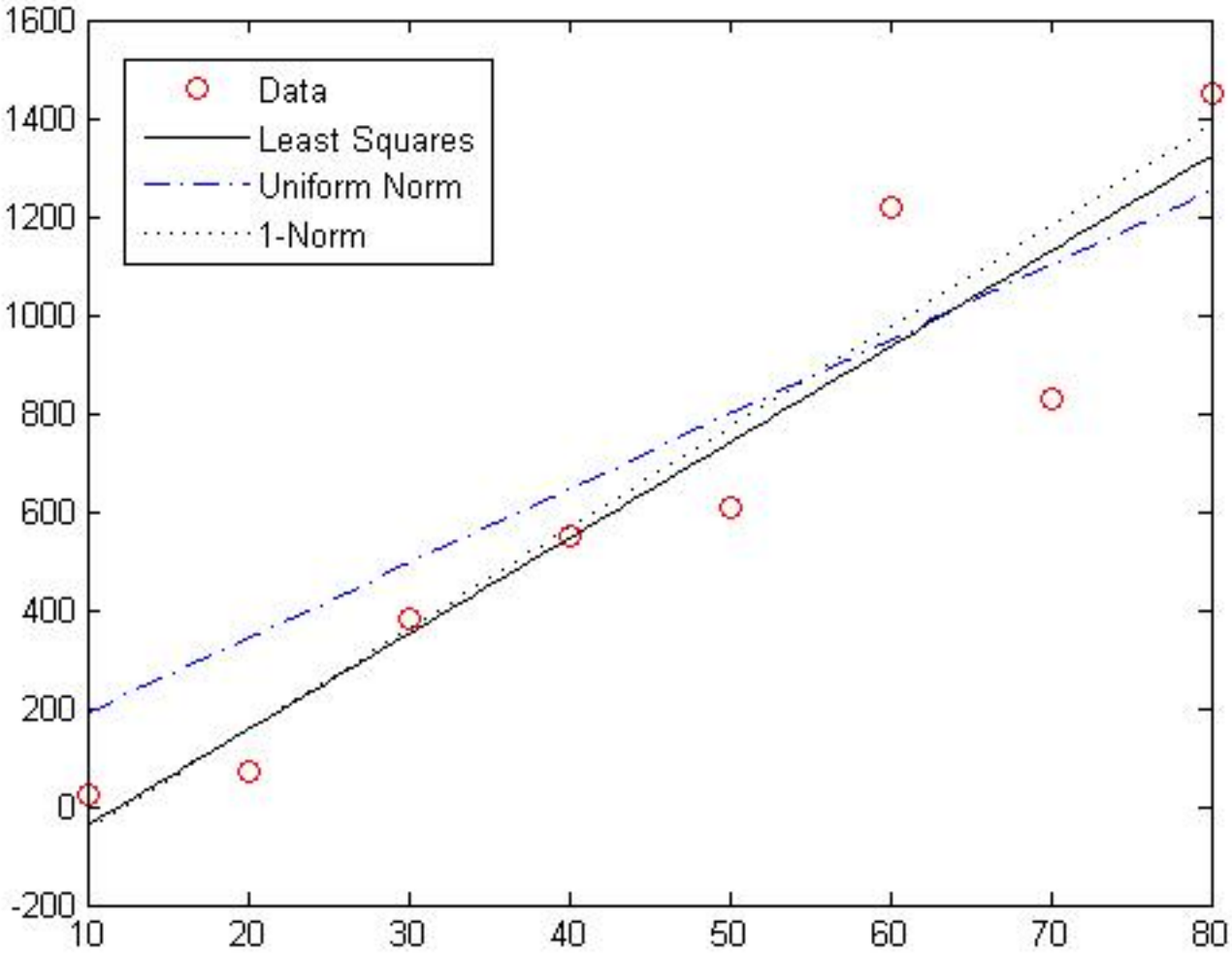

Figure 2: Comparison of lines of best fit using various measures of error.

4 The Median-Median Line

The median-median line is another technique for finding a line through data. It can be easily understood and easily constructed without regards to minimization or linear programming (we include it here for the sake of comparing it to our other techniques).

It should be used when there is a lot of noise and a lot of outliers in the data, as the median is very resistant to outliers.

In this case, order your data (using ascending x), and form 3 groups- Low, Middle, High. The groups should each have approximately the same number of data points; for example, if n is divisible by three, have three equally spaced groups. If there is 1 left over, put it in the middle group, and if there are two left over, make the middle group have one less than the low and high groups.

For each of the three groups, find the median values of x and y. Form the equation of the line through the high and low group median points. Shift this line 1/3 of the way towards the middle median point.

In Matlab, this is done by the following script file (most of the code is in splitting the data into the three groups):

```
function [m,b,resids]=medmedline(x,y);
x=x(:); y=y(:);
n=length(x);
[x,idx]=sort(x);
y=y(idx);

if rem(n,3)==0 %Three equal size groups
    Low=[x(1:(n/3)),y(1:n/3)];
    Middle=[x((n/3)+1:2*(n/3)),y((n/3)+1:2*(n/3))];
    High=[x(2*n/3+1:n),y(2*n/3+1:n)];
elseif rem(n,3)==1 %Make the middle group 1 bigger
    NumPts=floor(n/3);
    Low=[x(1:NumPts),y(1:NumPts)];
    Middle=[x(NumPts+1:2*NumPts+1),y(NumPts+1:2*NumPts+1)];
    High=[x(2*NumPts+2:n),y(2*NumPts+2:n)];
else %High and low groups have 1 more than middle
    NumPts=ceil(n/3);  %Number in Low and High groups
    Low=[x(1:NumPts),y(1:NumPts)];
    Middle=[x(NumPts+1:2*NumPts-1),y(NumPts+1:2*NumPts-1)];
    High=[x(2*NumPts:n),y(2*NumPts:n)];
end

%Form the median points:
LowMed=median(Low);
MidMed=median(Middle);
HiMed=median(High);

%Form the line between the low and high groups:
m1=(HiMed(2)-LowMed(2))/(HiMed(1)-LowMed(1));
```

```
b1=-m1*LowMed(1)+LowMed(2);

ytemp=m1*MidMed(1)+b1;
shift=(1/3)*(MidMed(2)-ytemp);
m=m1;
b=b1+shift;

resids=y-m*x-b;

t=linspace(min(x),max(x));
yt=m*t+b;
plot(x,y,'*',t,yt,'k-');
```