Line of Best Fit using Matlab or Octave

The basic problem is the following: Given a set of points,

$$\{(x_1, y_1), (x_2, y_2), \cdots, (x_k, y_k)\}$$

Find values for slope m and intercept b so that:

$$y_i = mx_i + b \qquad \text{for } i = 1, 2, \cdots, k$$

Writing this system of equations out, we get a matrix-vector equation:

$$\begin{array}{rl} mx_1 + b & = y_1 \\ mx_2 + b & = y_2 \\ \vdots & \vdots \\ mx_p + b & = y_p \end{array} \quad \Rightarrow \quad \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_p & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix} \quad \Rightarrow \quad A\mathbf{c} = \mathbf{b}$$

Remember- If the data actually sat on a line, we would only need two points to determine the line. In this case, we assume that the data does not actually all sit on the line, and so the matrix equation **has no solution**.

As we saw in class, we can solve the matrix equation by using the normal equations or using the QR decomposition of A.

A couple of notes about finding the line of best fit- Once you have the line, you should evaluate the line using new $x-$values and plot the result. You'll see that in the Matlab code to follow.

Here is a sample Matlab script, and the published output is attached.

```
%Example 1 using the normal equation

x=[2 5 7 8]'; y=[1 2 3 3]';

numpts=length(x);  % This is how many data points we have

A=[x ones(numpts,1)]; % Build the matrix A

c=inv(A'*A)*A'*y;  % c is the least squares slope/intercept

fprintf('The slope is: %f, the intercept is: %f\n',c(1),c(2));

%Computing the error between y and the values on the line:
Err=norm(y-A*c);
fprintf('The error is: %f\n',Err);

xTest=linspace(0,8);   %Create test data
yTest=c(1)*xTest+c(2); %Create the line using the test data
plot(x,y,'k*',xTest,yTest,'k-');  % Plot the data with the line
```

The sample published output is attached.

Find a Polynomial of Degree n

Similar to the line of best fit, we might want to fit a polynomial of degree n to our data.

You might recall from algebra that it takes two points for a line (or degree 1 polynomial), three points for a parabola (or degree two polynomial), and generally it takes $n+1$ points to define a polynomial of degree n.

Suppose we have 100 points. We certainly should not try to fit a polynomial of degree 99- such a polynomial would have wild swings (we'll try an example below). Rather, suppose we think a parabola should fit our data pretty nicely- Then we look for a polynomial of degree 2:

$$y = a_0 + a_1 x + a_2 x^2$$

Building the matrix will be similar to the previous one:

$$\begin{array}{rl} a_0 + a_1 x_1 + a_2 x_1^2 & = y_1 \\ a_0 + a_1 x_2 + a_2 x_2^2 & = y_2 \\ \vdots & \\ a_0 + a_1 x_p + a_2 x_p^2 & = y_p \end{array} \quad \Rightarrow \quad \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \\ 1 & x_p & x_p^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix}$$

Here is some Matlab code for this example:

```
%Example 2: Best fitting quadratic

x=[2 5 7 8]'; y=[1 2 3 3]';

numpts=length(x);  % This is how many data points we have

A=[ones(numpts,1)  x  x.^2 ]; % Build the matrix A

c=inv(A'*A)*A'*y

%Computing the error between y and the values on the line:
Err=norm(y-A*c);
fprintf('The error is: %f\n',Err);

xTest=linspace(0,8)';   %Create test data
yTest=[ones(length(xTest),1) xTest xTest.^2]*c;
plot(x,y,'k*',xTest,yTest,'k-');  % Plot the data with the parabola
```

The published result is attached.

Other Models

Or the model may be something like the following, where the unknowns are the values of a_0, a_1, a_2 and a_3:

$$y = a_0 \sin(x) + a_1 \sin(2x) + a_2 \cos(x) + a_3 \cos(3x)$$

And even though this looks nonlinear, it is a linear problem in a_0, a_1, a_2 and a_3. The matrix equation would be:

$$\begin{bmatrix} \sin(x_1) & \sin(2x_1) & \cos(x_1) & \cos(3x_1) \\ \sin(x_2) & \sin(2x_2) & \cos(x_2) & \cos(3x_2) \\ \vdots & \vdots & & \\ \sin(x_p) & \sin(2x_p) & \cos(x_p) & \cos(3x_p) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_p \end{bmatrix}$$

Here's the example done up in Matlab. First, we make up some random noisy data to use, then we build the model.

```
% Example 3, Section 6.5

%Make up some noisy data.
x=3*pi*rand(40,1);  %40 random numbers between 0 and 3*pi
x=sort(x);
y=-sin(x)+3*sin(2*x)-0.5*cos(x)+4*cos(3*x)+randn(size(x));

%Create matrix A:
A=[sin(x) sin(2*x) cos(x) cos(3*x)];
c=inv(A'*A)*A'*y

%Computing the error between y and the values on the line:
Err=norm(y-A*c);
fprintf('The error is: %f\n',Err);

xTest=linspace(0,10)';   %Create test data
yTest=[sin(xTest) sin(2*xTest) cos(xTest) cos(3*xTest) ]*c;
plot(x,y,'k*',xTest,yTest,'k-');
```

Solving using QR

We'll solve this last equation using QR instead of the normal equations. Here's the code again- The first several lines are to create some data.

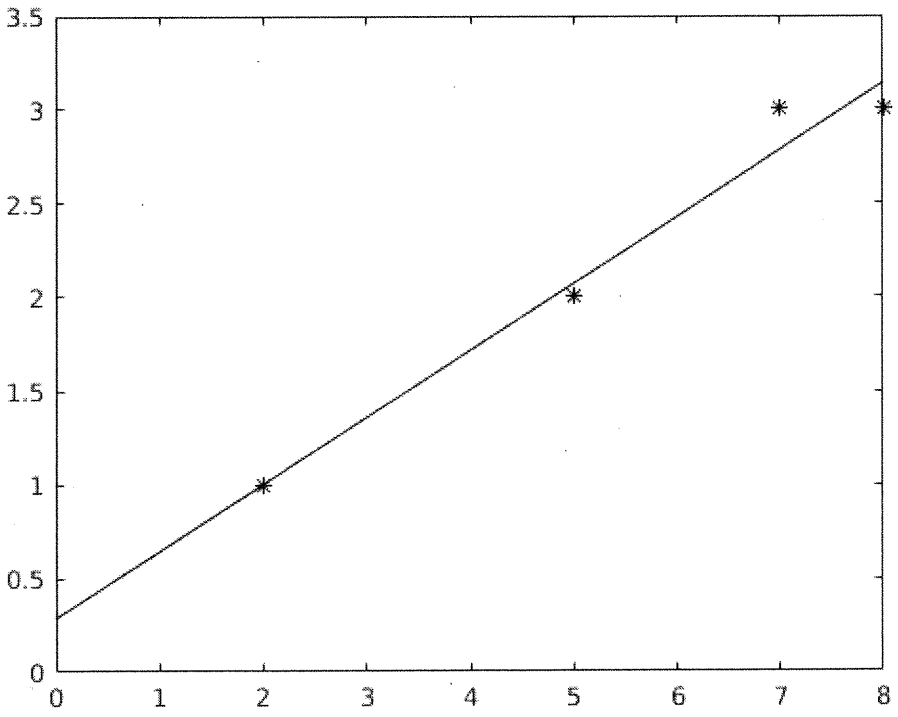
```
% Example 4, Section 6.5

%Make up some noisy data.
x=3*pi*rand(40,1);  %40 random numbers between 0 and 3*pi
x=sort(x);
y=-sin(x)+3*sin(2*x)-0.5*cos(x)+4*cos(3*x)+randn(size(x));

%Create matrix A:
```

```
A=[sin(x) sin(2*x) cos(x) cos(3*x)];
[Q,R]=qr(A,0);  %The option '0' is so that Q is not square
c=inv(R)*Q'*y

%Computing the error between y and the values on the line:
Err=norm(y-A*c);
fprintf('The error is: %f\n',Err);

xTest=linspace(0,10)';   %Create test data
yTest=[sin(xTest) sin(2*xTest) cos(xTest) cos(3*xTest) ]*c;
plot(x,y,'k*',xTest,yTest,'k-');
```

The published output is attached separately.

```
%Example 1 using the normal equation

x=[2 5 7 8]'; y=[1 2 3 3]';

numpts=length(x);  % This is how many data points we have

A=[x ones(numpts,1)]; % Build the matrix A

c=inv(A'*A)*A'*y;  % c is the least squares slope/intercept

fprintf('The slope is: %f, the intercept is: %f\n',c(1),c(2));
%Computing the error between y and the values on the line:
Err=norm(y-A*c);
fprintf('The error is: %f\n',Err);

xTest=linspace(0,8);   %Create test data
yTest=c(1)*xTest+c(2); %Create the line using the test data
plot(x,y,'k*',xTest,yTest,'k-');  % Plot the data with the line
```

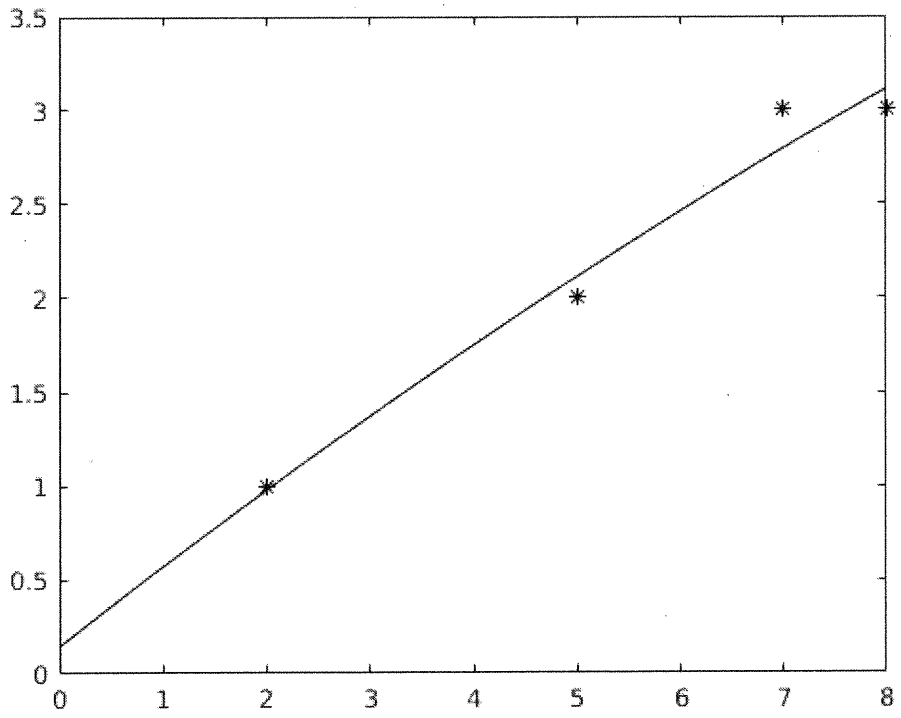
```
The slope is: 0.357143, the intercept is: 0.285714
The error is: 0.267261
```



Published with MA TLAB® R2016a

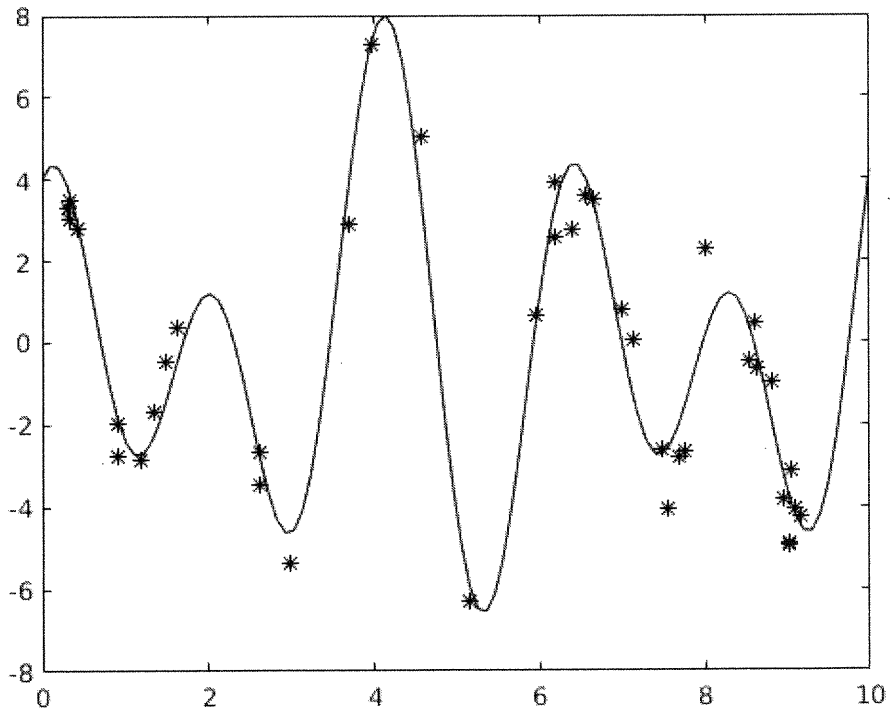
```
%Example 2: Best fitting quadratic

x=[2 5 7 8]'; y=[1 2 3 3]';

numpts=length(x);  % This is how many data points we have

A=[ones(numpts,1)  x  x.^2 ]; % Build the matrix A

c=inv(A'*A)*A'*y

%Computing the error between y and the values on the line:
Err=norm(y-A*c);
fprintf('The error is: %f\n',Err);

xTest=linspace(0,8)';   %Create test data
yTest=[ones(length(xTest),1) xTest xTest.^2]*c;
plot(x,y,'k*',xTest,yTest,'k-');  % Plot the data with the parabola
```

```
c =

    0.1439
    0.4318
   -0.0076

The error is: 0.261116
```



```
% Example 3, Section 6.5

%Make up some noisy data.
x=3*pi*rand(40,1);  %40 random numbers between 0 and 3*pi
x=sort(x);
y=-sin(x)+3*sin(2*x)-0.5*cos(x)+4*cos(3*x)+randn(size(x));

%Create matrix A:
A=[sin(x) sin(2*x) cos(x) cos(3*x)];
c=inv(A'*A)*A'*y

%Computing the error between y and the values on the line:
Err=norm(y-A*c);
fprintf('The error is: %f\n',Err);

xTest=linspace(0,10)';   %Create test data
yTest=[sin(xTest) sin(2*xTest) cos(xTest) cos(3*xTest) ]*c;
plot(x,y,'k*',xTest,yTest,'k-');  % Plot the data with the parabola
```

```
c =

   -0.8472
    3.0807
   -0.3253
    4.3433

The error is: 5.437065
```



```
% Example 4, Section 6.5

%Make up some noisy data.
x=3*pi*rand(40,1);  %40 random numbers between 0 and 3*pi
x=sort(x);
y=-sin(x)+3*sin(2*x)-0.5*cos(x)+4*cos(3*x)+randn(size(x));

%Create matrix A:
A=[sin(x) sin(2*x) cos(x) cos(3*x)];
[Q,R]=qr(A,0);  %The option '0' is so that Q is not square
c=inv(R)*Q'*y

%Computing the error between y and the values on the line:
Err=norm(y-A*c);
fprintf('The error is: %f\n',Err);

xTest=linspace(0,10)';   %Create test data
yTest=[sin(xTest) sin(2*xTest) cos(xTest) cos(3*xTest) ]*c;
plot(x,y,'k*',xTest,yTest,'k-');
```

```
c =

   -0.9462
    3.3968
   -0.5430
    4.2201

The error is: 6.087244
```