ENGR 231
Linear Engineering Systems

Lab 7 - Curve Fitting via Least Squares in MATLAB

Goals:

1. Introduce the concept of a least squares regression in MATLAB.
2. Apply least squares for linear regression against a data set.
3. Apply least squares for polynomial curve fitting against a data set.

Introduction

A common task in science and engineering is to model and analyze relationships existing in the natural world. As an example, consider the relationship between running speed and an individual's heart rate. If we could quantize this phenomenon within the context of a mathematical model, then we could stand to make more detailed conclusions of the true relationship that exists.

In the simplest terms, we can cast this problem as modeling via some mathematical equation having a standard form (i.e. linear, exponential, polynomial, logarithmic etc.). Once we have decided upon the standard form of the model, we can then collect data and come up with an estimation of the parameters for the model we chose. Finally, we can see how closely the data points predicted by our model lines up with the actual data points found.

To do this parameter estimation, we will use ***least squares estimation***. This will be introduced in lecture later, however here we introduce the minimum needed for implementation.

Derivation of Least Squares Estimation

Let's assume the simplest model: the relationship being ***linear***. We can write in general for any given running speed, x_i, and heart rate, y_i:

$$y_i = \beta_0 + \beta_1 x_i \tag{1}$$

where β_0 and β_1 are the parameters corresponding the y-offset and slope of the linear equation respectively. If we write out these equations for a set of ***N (x, y)*** coordinate pairs we have:

$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_1 \\ y_2 &= \beta_0 + \beta_1 x_2 \\ &\cdots\cdots \\ y_N &= \beta_0 + \beta_1 x_N \end{aligned} \tag{2a}$$

In other words, we have independently measured N values for an individual's running speed and at the same times the individual's heart rate. Combining these using the same model yields N equations in two unknowns. Writing this in matrix form yields the following:

$$\begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdots & \cdots \\ 1 & x_N \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \tag{2b}$$

$$\mathbf{y} = D\beta \tag{2c}$$

where D is known as the design matrix, $\mathbf{y}$ is known as the observation vector, and $\boldsymbol{\beta}$ is known as the parameter vector. Our goal is to determine the parameter vector, $\boldsymbol{\beta}$, with as much certainty possible.

If the design matrix D was invertible, then we would be able to solve for the parameter vector in a relatively straightforward manner. However, since D is a Nx2 matrix we see immediately by definition that it is not invertible. We will rectify this through the *least squares method*. The actual derivation is in-depth and done in class in week 10 (agood reference Lay, sections 6.5 and 6.6). The end result is that we can solve the equation $\mathbf{y} = D\boldsymbol{\beta}$ in the following way: We multiply the equation by D^T, the transpose of D,

$$D^T\mathbf{y} = D^T D\beta. \tag{2}$$

We can show that the matrix D^TD is square and invertible. Therefore, we can now solve for $\boldsymbol{\beta}$ via:

$$\beta = \left(D^T D\right)^{-1} D^T\mathbf{y}\ . \tag{3}$$

The value of $\boldsymbol{\beta}$ obtained from this formula produces the least error.

This method is used in the following way:

1. We obtain a large number of (x, y) data points. We either choose or are given an equation we wish to fit the data ,
2. From the data set, create the observation vector $\mathbf{y}$ and the design matrix D
3. Use the boxed equation given above to calculate $\boldsymbol{\beta}$from D and $\mathbf{y}$.
4. Use the $\boldsymbol{\beta}$ values from step 3, in equation (2c) to compute the *predicted* values of at the specified x values.
5. Compare the predicted and observed y values, to see how good is formula obtained.

The example code below illustrates an implementation in MATLAB:

Ex 1:

```
%%  Linear Least Square Estimation
% Given the following set of points:
% First row:  x values
% Second row: y values (observed)
pts = [-0.04 0.35 0.57 1.23 2.17 2.15 3.21 3.44 3.90 4.52 5.48
        4.06 5.78 7.14 8.39 10.3 11.9 13.0 14.4 16.0 17.8 19.1];

% Make the design matrix, D, and observation vector, Y
D = [ones(length(pts),1), pts(1,:)']
Y = pts(2,:)'   % The symbol ' transposes the matrix

% Find Beta based on the equations
beta = (D'*D)^-1*(D'*Y)

% Use the matrix equation to get all of the estimated y values
Y_est = D*beta;

%% Plot the points as circles and the fitted line
plot(pts(1,:),pts(2,:),'o'), hold on
plot(pts(1,:),Y_est,'r')
legend('Observations','Estimated Linear Fit')
```

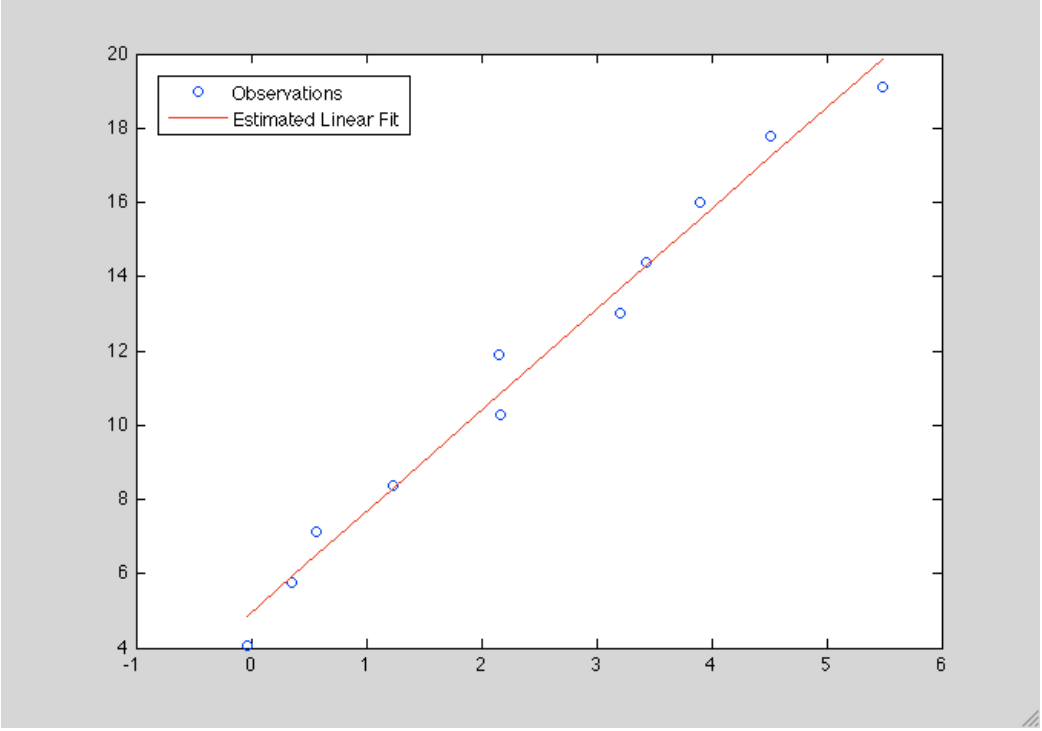

Figure 1. Example 1 Output graph showing the estimated fit compared to the actual data.

Calculating a Goodness of Fit metric (RMS error)

It important to find simple numerical measure that shows how close is our model to the actual data points given. If we use the estimated parameter vector, β_{EST}, we can calculate the estimated observations, $\mathbf{y}_{EST}$, with the following matrix equation:

$$\mathbf{y}_{EST} = D\beta_{EST}. \tag{4}$$

The vector of error values can now be found from the equation

$$\varepsilon = \mathbf{y} - \mathbf{y}_{EST},$$

the differences between the observed and the estimated values. From this we compute

$$\varepsilon_{RMS} = \left(\frac{1}{N} \varepsilon^T \varepsilon \right)^{1/2} \tag{5}$$

In example 2, we repeat the calculations of example 1 and append the computation of the RMS error. We also illustrate the use of the 'load' command. Rather than defining the *x* and *y* values in a statement, we load a variable 'pts' from a file†.

---

† The data used resides in file 'pts_ex.mat'. The command 'load pts_ex' reads this file and creates variables that were saved in this file. In the current example there is only one variable whose name is pts. The values in this are identical to the ones in Ex. 1.

Ex 2:

```
% Read the data file pts_ex.mat
% First row:  x values
% Second row: y values (observed)
load pts_ex
N = length(pts);
% Make the design matrix, D, and observation vector, Y
D = [ones(N,1), pts(1,:)']
Y = pts(2,:)'   % The symbol ' transposes the matrix
                % This makes it a column vector

% Find Beta_est based on the equations
% Use the matrix equation to get all of the estimated y values
beta_est = (D'*D)^-1*(D'*Y)
Y_est = D*beta_est;

%% Plot the points as circles and the fitted line
plot(pts(1,:),pts(2,:),'o'), hold on
plot(pts(1,:),Y_est,'r')
legend('Observations','Estimated Linear Fit')

%% Calculate the RMS Error
% Notice that both Y and Y_est are column vectors.
err = Y-Y_est;
RMSE = (err'*err/N)^0.5
title(['RMS error = ', num2str(RMSE)])
```

Least Squares with a Second Degree Polynomial

Following the exact same methodology presented in the previous discussion, one can also perform curve fitting using a second order polynomial. The general equation for a second order polynomial is:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2$$

Following the same method, we end up with the matrix form as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \cdots & \cdots & \cdots \\ 1 & x_N & x_N^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

$$\mathbf{y} = D\beta$$

Though matrix ***D*** and vector β are different from the equations for linear regression, it turns out that the matrix equations for finding the least squares curve fit are identical to those shown in the linear case. As before, the equation is

$$\beta = \left(D^T D\right)^{-1} D^T \mathbf{y}$$

with ***X***, ***Y***, and $\boldsymbol{\beta}$ as defined above. Again, once β is known, $\boldsymbol{Y_{EST}}$ can be found via

$$\mathbf{y}_{EST} = D\beta_{EST}$$

Clearly the same process can be used for higher order curve fits.

Appendix – Squaring a set of values

Consider the following set of data in MATLAB: `a = [3 4 5 7]`. Say we wanted to find the values given in `b` squared. If we intuitively tried the command `a^2`, MATLAB would produce the following error:

```
??? Error using ==> mpower
Inputs must be a scalar and a square matrix.
```

This is because MATLAB thinks we are trying to do the matrix multiplication `a*a`. This obviously can't work, let alone is it what we are actually trying to do. To rectify this, we must use the ***dot operator***. The command becomes `a.^2,` and the correct result of `[9 16 25 49]` is produced.