Fitting equations to experimental data: systems with two variables

by Richard K. Herz <herz@ucsd.edu>

Notation used below

x_j = independent variable values

y_j = dependent variable values

a_i = unknown parameters in equations whose values are to be determined in the fitting process, where i = 0 to n, for total of (n+1)

(n+1) = number of unknown parameter values

m = number of experimental measurements, i.e., number of x_j, y_j pairs, where we must have m > = (n+1).

Comments on "straight line" and "polynomial" fits

Straight lines are polynomials of order one. Fitting straight lines and higher-order polynomials to data is a problem in linear algebra, since the problem is linear in the (n+1) unknown coefficents, a_i .

$$a_n x_1^n + a_{n-1} x_1^{n-1} + \ldots a_0 = y_1$$
$$a_n x_2^n + a_{n-1} x_2^{n-1} + \ldots a_0 = y_2$$
$$\vdots$$
$$a_n x_m^n + a_{n-1} x_m^{n-1} + \ldots a_0 = y_m$$

Each equation (row) corresponds to one experiment measuring the pair, x_j, y_j. Remember that all the x_j's and y_j's are measured, known values. The (n+1) a_i values are the unknowns that we want to determine. With the ordering of terms shown here, the ordering of coefficients in the **a** column vector is a_n at the top (element 1) , followed by a_{n-1}, and continuing to a_0 at the bottom (element n+1).

In matrix notation the system is,

$$\mathbf{X}\,\mathbf{a} = \mathbf{y}$$

where **X** is the m-row-by-(n+1)-column matrix of measured values (each row: $x_j^n, x_j^{n-1}, \ldots x_j^1, x_j^0 = 1$) , **a** is the column vector of unknown coefficient values to be determined $(a_n, a_{n-1}, \ldots, a_0)^T$, and **y** is the column

vector of measured values y_j.

We know that if the rank of the X matrix and the rank of the augmented, [X, y] matrix are equal and, thus, m = (n+1), there is a unique solution given by

$$\mathbf{a} = \mathbf{X}^{-1}\,\mathbf{y}$$ % for m = (n+1), in Matlab notation, a = inv(X) * y

Since there is random scatter in experimental measurements, we usually will have more experimental measurements than there are unknowns, i.e., m > (n+1), in order to get the best estimate of parameter values that we can. In this case, the matrix **X** is not square and cannot be inverted. This is an "overdetermined" lienar system. By multiplying both sides of the starting matrix equation by the transpose of **X**, however, we obtain the square, invertable matrix [$\mathbf{X}^T\,\mathbf{X}$], and the solution becomes:

$$\mathbf{a} = [\mathbf{X}^T\,\mathbf{X}]^{-1}\,[\mathbf{X}^T\,\mathbf{y}]$$ % for m > (n+1), in Matlab notation, a = inv(X' * X)*(X' * y)

The coefficients a_i returned are those that minimize the sum of the squared errors between the experimental and model predictions of y_j. The proof of this can be seen for the example of a first-order polynomial by comparing this solution to the derivation given in Appendix M of Himmelblau's text (6th ed.).

The Matlab backslash operator can also be used to solve overdetermined linear systems:

```
a = X\y
```

Fitting polynomial coefficients to data is such a common task that Matlab and Excel have special functions and tools that can also be used instead of this direct linear algebra solution. In fact, in each package there is more than one option to choose from.

One thing to note below is that the Matlab function fminsearch() and the Excel tool Solver can be applied to straight lines and higher-order polynomial equations, as well as nonlinear equations. The reason for using these general tools for straight line and polynomial fits, however, would be to minimize an objective function other than the standard sum of the square errors. An example would be when you want to minimize the sum of the squared relative errors, e.g., Sum((measured - model)/model)^2.

(1) Straight line

MATLAB

(a) Standard function polyfit().

A straight line is a polynomial of order one.

```
coef = polyfit(x, y, 1)  % last argument,  n = 1 here, is order of polynomial
```

Function polyfit() returns (n+1) best-fit coefficient values in the order: coefficient of highest power of independent variable first (a_n) to coefficient of zero-power of independent variable last (a_0, i.e., the "constant"). Note, the following are arbitrary variable names: coef, x, y.

(b) Standard function fminsearch().

Use fminsearch() instead of polyfit() when the objective function you want to minimize is something other than the standard "sum of the squared errors" that is minimized by polyfit(). An example would be when you want to minimize the sum of the squared relative errors, e.g., Sum((measured - model)/model)^2.

EXCEL

(a) Use the Regression Tool

(b) Add a linear trendline to a chart with the Trendline Tool

Make sure you check the box "Display equation on chart" under the Options tab in the Add Trendline dialog window.

(c) Solver Tool

Use Solver instead of Regression or Trendline when the objective function you want to minimize is something other than the standard "sum of the squared errors." Example would be when you want to minimize the sum of the squared relative errors, e.g., Sum((measured - model)/model)^2.

(d) Standard function linest()

The other tools usually are used to operate once on an entire set of data. The function linest() is more "programmable." Use linest() when you need to fit more than one subset of a larger set of data. For example, in order to find the derivative of a data set: fit the first five points to a straight line and get the slope (derivative) at the midpoint, then drop the first point and add the sixth point, the fit this second subset of five points, etc., through all subsets of five in the entire set.

(2) Polynomial

MATLAB

(a) Use the standard function polyfit().

```
coef = polyfit(x, y, n)  % last argument, n, is order of polynomial
```

Function polyfit() returns (n+1) best-fit coefficient values in the order: coefficient of highest power of independent variable first (a_n) to coefficient of zero-power of independent variable last (a_0, i.e., the "constant"). Note, the following are arbitrary variable names: coef, x, y.

(b) Standard function fminsearch().

Use fminsearch() instead of polyfit() when the objective function you want to minimize is something other than the standard "sum of the squared errors" that is minimized by polyfit(). An example would be when you want to minimize the sum of the squared relative errors, e.g., Sum((measured - model)/model)^2.

EXCEL

(a) Trendline Tool, add a polynomial trendline to a chart with this tool.

Check the box "Display equation on chart" under Options tab in Add Trendline dialog window.

(b) Solver Tool

UseSolver instead of Regression or Trendline when the objective function you want to minimize is something other than the standard "sum of the squared errors." Example would be when you want to minimize the sum of the squared relative errors, e.g., Sum((measured - model)/model)^2.

(c) Standard function linest()

The other tools usually are used to operate once on an entire set of data. The function linest() is more "programmable." Use linest() when you need to fit more than one subset of a larger set of data. For example, in order to find the derivative of a data set: fit the first five points to a quadratic equation and get the derivative at the midpoint, then drop the first point and add the sixth point, the fit this second subset of five points, etc., through all subsets of five in the entire set.

(3) Nonlinear equation

MATLAB

(a) Standard function fminbnd() for one unknown parameter, or the standard function fminsearch() for one or more unknown parameters.

EXCEL

(a) Trendline Tool

Add exponential or logarithmic or "power" (y = ax^b) trendlines to charts. Make sure you check the box "Display equation on chart" under the Options tab in the Add Trendline dialog window.

(b) Solver Tool.