A Curve-Fitting Cookbook for use with the NMM Toolbox

Gerald Recktenwald*

October 17, 2000

Abstract

Computational steps for obtaining curve fits with MATLAB are described. The steps include reading data into MATLAB variables, setting up the overdetermined system of equations, obtaining the least squares solution, and plotting a comparision of the fit function with the original data. The procedures and MATLAB routines from [1] are demonstrated for fits of data to lines, functions that may be transformed into lines, polynomials, arbitrary linear combination of functions, and multivariate (surface) fits.

1 Introduction

This document provides a minimal set of instructions for obtaining univariate and multivariate curve fits using MATLAB. Readers are expected to understand the nature of curve fitting, and the basic procedures involved. For a complete introduction to curve-fitting with MATLAB see [1] and the Numerical Methods with MATLAB (NMM) Toolbox described therein.

1.1 Objective of fitting

Given m data pairs, (x_i, y_i), find the coefficients, c_j, in the function

$$F(x) = c_1 f_1(x) + c_2 f_2(x) + \ldots + c_n f_n(x) \tag{1}$$

such that $F(x_i) \approx y_j$. $F(x)$ is called the *fit function*. The $f_j(x)$ are called the *basis functions* of $F(x)$. In curve fitting there are more data points, m, than basis functions, n.

The user chooses the basis functions. The c_i are determined by solving the least squares problem for an $m \times n$ system of equations (m equations in n unknowns, where $m > n$). The m equations are obtained by evaluating the fit function at each of the known data points.

1.2 Residuals

Since $F(x_i)$ will not, in general, exactly equal y_i, compute the residual, r_i, at the ith data point

$$r_i = y_i - F(x_i) \tag{2}$$

A *least squares* fit finds the c_j in equation (1) such that $\sum_{i=1}^{m} r_i^2$ is minimized. "Least" refers to the process of minimization. "Squares" refers to the quantity being minimized, i.e., the sum of the squares of the residuals, $\sum r_i^2$.

*Mechanical Engineering Department, Portland State University, Portland, Oregon, gerry@me.pdx.edu

1.3 Data with Errors

An alternative view of curve-fitting takes $F(x)$ as the true model of the (x, y) data. The x_i are assumed to be known exactly, and the y_i are assumed to contain a random error, ϵ_i. Because of the error in y_i, $F(x_i) \neq y_i$. In this view of curve-fitting the relationship between the fit function and the data is

$$y_i = F(x_i) + \epsilon_i \tag{3}$$

The ϵ_i in Equation (3) play an analogous role to the r_i in Equation (2). If we can further assume that the ϵ_i are normally distributed random variables, i.e. if the observed ϵ_i is taken from a Gaussian distribution of possible ϵ at each x_i, then the least squares fit provides the most likely estimate of the c_j that are the adjustable parameters in Equation (1).

1.4 Overdetermined System of Equations

Consider the task of fitting m data points to three basis functions, i.e.

$$y = F(x) = c_1 f_1(x) + c_2 f_2(x) + c_3 f_3(x) \tag{4}$$

Assume for the moment that the c_i can be found so that the $F(x)$ passes exactly through all the data, i.e., suppose that the data "line up" so that the curve fit function acts like an interpolant. Then each of the following m equations would be satisfied exactly

$$\begin{aligned} c_1 f_1(x_1) + c_2 f_2(x_1) + c_3 f_3(x_1) &= y_1 \\ c_1 f_1(x_2) + c_2 f_2(x_2) + c_3 f_3(x_2) &= y_2 \\ &\vdots \\ c_1 f_1(x_m) + c_2 f_2(x_m) + c_3 f_3(x_m) &= y_m \end{aligned}$$

This is equivalent to the overdetermined system

$$Ac = y \tag{5}$$

where

$$A = \begin{bmatrix} f_1(x_1) & f_2(x_1) & f_3(x_1) \\ f_1(x_2) & f_2(x_2) & f_3(x_2) \\ \vdots & \vdots & \vdots \\ f_1(x_m) & f_2(x_m) & f_3(x_m) \end{bmatrix} \qquad c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{6}$$

To develop equation (5) we momentarily assumed that the data fell along the curve given by equation (4). In general this will not be the case. Since there are more equations than free parameters ($m > n$), the exact solution to equation (5) cannot be obtained. In other words, when $m > n$ it is not usually possible to find the c_j that *simultaneously* satisfy all m equations.

The least squares method provides the compromise solution that minimizes $\|r\|_2^2 = \|y - Ac\|_2^2$. The minimum value of $\|r\|_2^2$ is obtained by the c vector that satisfies the *normal equations*

$$(A^T A)c = A^T y \tag{7}$$

$(A^T A)$ is an $n \times n$ matrix, and $A^T y$ is an n-element column vector. As long as $(A^T A)$ is not ill-conditioned,[1] the solution to Equation (7) may be obtained by Gaussian elimination. In fact, $A^T A$

[1] If two or more of the basis functions are linearly dependent, then A will be rank-deficient and $A^T A$ will be singular. In other cases the condition number of $A^T A$ will be large if the elements in A have large variations in the order of magnitude

is symmetric and positive definite (see, e.g. [2]) so that Equation (7) can be solved via Cholesky factorization. Instead of solving the normal equations one can use an orthogonal decomposition of A to compute the least squares solution directly from equation (5). MATLAB routines that implement both procedures are discussed below.

For the general case of fitting n basis functions to m data points we have

$$A = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \dots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_n(x_2) \\ \vdots & \vdots & & \vdots \\ f_1(x_m) & f_2(x_m) & \dots & f_n(x_m) \end{bmatrix} \qquad c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{8}$$

Typically $m \gg n$.

1.5 Numerical Procedures

To set up the least squares problem for computer solution

1. Read the (x, y) data into program variables
2. Compute A, the $m \times n$ matrix defined by the x_i data ($i = 1, \dots, m$) and the basis functions, f_j, $j = 1, \dots, n$
$$A = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \dots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_n(x_2) \\ \vdots & \vdots & & \vdots \\ f_1(x_m) & f_2(x_m) & \dots & f_n(x_m) \end{bmatrix} \tag{9}$$
3. Solve for c using one of the procedures described below.

1.5.1 Solution via the Normal Equations

Given A defined by equation (9), the c can be obtained by solving equation (7). These steps are automated by the `fitnorm` function in the NMM Toolbox.

1.5.2 Solution via QR Factorization

Given A defined by equation (9), the c can be obtained via QR factorization of A

$$A = QR$$

where Q is an orthogonal matrix and R is an upper triangular matrix. See [1, § 9.2.3] for details. The properties of Q and R give the least squares solution as

$$c = R^{-1}Q^T y \tag{10}$$

Since R is upper triangular the steps implied by equation (10) can be carried out efficiently once Q and R are available.

The `fitqr` function in the NMM Toolbox obtains a least squares curve fit using QR factorization. Use of the `fitqr` function is described in § 6, below.

2 Reading data into MATLAB variables

Data for the fit must first be stored in MATLAB variables. This can be achieved by assigning the values of x and y directly, i.e.,

```
x = [ ... ]
y = [ ... ]
```

or the data can be read from a file. The following procedures can be used to read data from a file

1. Read "naked" data with the built-in `load` function
2. Read data from a file containing a text header (optional) and text column headings with the NMM `loadColData` function.
3. Read data from a file containing a text header (optional), text column headings, and the first column in a (string) date format with the NMM `loadColDateData` function.

2.1 Using the built-in `load` function

The built-in `load` function reads numeric data from a plain text file into a MATLAB matrix. The file must contain only numbers that are arranged in rows and columns. The number of columns in each row must be the same.

The `load` function can be used in two ways: in *command* form and in `function` form. Both ways can be used to achieve the same effect, though the syntax of the methods is different. The syntax of the command form is

```
load filename
```

where ***filename*** is the name of the file containing data. The result of executing the command form is that the contents of ***filename*** are stored in a MATLAB variable named ***filename***. If the ***filename*** contains an extension, such as `.dat` or `.txt`, the MATLAB variable created by the statement `load` ***filename*** is the creation of a variable *without* the extension. For example

```
load myData.dat
```

creates a variable named `myData` that contains the contents of the file named `myData.dat`, and

```
load yourData
```

creates a variable named `yourData` that contains the contents of the file named `yourData`.

The function form of the `load` function has the syntax

```
D = load('filename')
```

In this usage, the data contained in ***filename*** is stored in the variable called `D`.

2.1.1 Example: Reading Data from a Plain Text File

The `airvisc.dat` file contains data for the viscosity of air as a function of temperature. The data in in two columns. The first column is the temperature in °C, and the second column is the viscosity in kg/(m · s). The first few lines of `airvisc.dat` look like

```
 0     1.720e-5
20     1.817e-5
40     1.911e-5

... additional data in file
```

If the `airvisc.dat` file is in the MATLAB path, the following statements load the data into the `p` and `t` variables

```
>> load airvisc.dat          %  Data is stored in "airvisc" matrix
>> t = airvisc(:,1);         %  Copy first column of airvisc into t
>> mu = airvisc(:,2);        %  and second column into mu
```

The same effect is achieved with

```
>> D = load('airvisc.dat');  %  Data is stored in D matrix
>> t = D(:,1);               %  Copy first column of D into t
>> mu = D(:,2);              %  and second column into mu
```

Or, if the full path to `airvisc.dat` file is (on a Windows™ computer) in `c:\matlab\toolbox\nmm\data` the following could be used

```
>> D = load('c:\matlab\toolbox\nmm\data\airSat.dat');
>> p = D(:,1);   t = D(:,2);
```

The function form of `load` is useful when the name of a data file is passed into another function that is responsible for reading, and possibly manipulating, the data. Consider the `plotMyData` function listed below.

```
function plotMyData(fname)
% plotMyData  Plot (x,y) data in a named text file
%
% Synopsis:  plotMyData(fname)
%
% Input:  fname = (string) name of file containing data
%
% Output:  A plot of the data in fname

D = load(fname);
plot(D(:,1),D(:,2));
```

2.2 Using the NMM `loadColData` function

Often data files contain descriptive text in addition to the numerical data. The NMM `loadColData` allows reading of data from such files providing that all of the descriptive text is at the top of the file.

The contents of the `pdxTave.dat` file are

```
Historically averaged temperature data
recorded at the Portland International Airport
Temperatures in degrees F
month  high    low     ave
  1   45.36   33.84   39.6
  2   50.87   35.98   43.43
  3   56.05   38.55   47.3
  4   60.49   41.36   50.92
  5   67.17   46.92   57.05
  6   73.82   52.8    63.31
  7   79.72   56.43   68.07
  8   80.14   56.79   68.47
  9   74.54   51.83   63.18
 10   64.08   44.95   54.52
 11   52.66   39.54   46.1
 12   45.59   34.75   40.17
```

The first three rows of the file contains a header describing the contents of the file. The fourth row contains labels for the data in the columns.

The contents of the `pdxTave.dat` file are read with

```
>> [month,T,labels] = loadColData('pdxTave.dat',4,3)
```

The "4" argument tells `loadColData` that the data is contained in four columns. The "3" argument tells `loadColData` that the data is preceded by a three line header of text information.

The advantage of using `loadColData` is that the important labeling data can be left in the file. To read the same file with the `load` function, one would have to delete the header and column labels. That would defeat the purpose of documenting the data with the header and column labels.

2.3 Using the NMM `loadColDateData` function

The contents of the `tualatin.dat` file are

```
Date    Temperature    Conductivity    DO  FlowRate
06/05/1980  14.50   125 7.20    364
07/10/1980  20.00   183 5.00    201
08/07/1980  19.40   134 6.50    189
09/11/1980  17.50   136 6.60    213
10/09/1980  14.00   109 8.00    374

... more data in file
```

The data is from water quality measurements in the Tualatin river in Oregon. The first row contains labels for the data in the columns. The first column is in a `mm/dd/yyyy` date format, where `mm` is the number of the month, `dd` is the number of the date, and `yyyy` is the year.

The contents of the `tualatin.dat` file are read with

```
>> [d,Y,labels] = loadColDateData('tualatin1.dat',5,0,1);
```

where `d` is a vector of dates stored in MATLAB's `datenum` format, `Y` is a matrix containing the data in columns 2 through 4, and `labels` is a string matrix holding the column labels. Type `help loadColDateData` or read the source code of the function to get more information on the options and inputs to the `loadColDateData` function.

3 Fitting a line to data: $y = c_1x + c_2$

3.1 Using `linefit`

The NMM `linefit` function fits a line to data. The equation of the line is $y = c_1x + c_2$. A typical usage is

```
>> x = ...
>> y = ...
>> c = linefit(x,y)
```

or

```
>> [c,R2] = linefit(x,y)
```

where `R2` is the R^2 statistic. See [1, § 9.1.4]. Given the `x`, `y`, and `c` vectors defined in the preceding statments, the following statements evaluate the fit function and plot a comparison of the fit with the original data

```
>> xfit = [min(x)  max(x)];
>> yfit = c(1)*xfit + c(2);
>> plot(x,y,'o',xfit,yfit,'--');
>> legend('data','fit');
```

3.2 Using `polyfit`

The built-in `polyfit` function obtains curve fits of polynomials to (x, y) data. Since a line is a polynomial of degree 1, the `polyfit` function can be used to fit a line as follows

```
>> x = ...
>> y = ...
>> c = polyfit(x,y,1)
```

or

```
>> [c,S] = polyfit(x,y,1)
```

where `S` is a data structure used by the built-in `polyval` to obtain error bounds that contain 50 percent of the data, *if* the errors in the data are independent, normally distributed, and have the same variance.

4 Fitting a Line to Transformed Data

With a suitable change of variables, some simple nonlinear functions can be transformed to a linear function. A linear least squares curve fit can then be performed on the transformed variables. For example, a simple exponential growth/decay curve is a line when plotted on semilogarithmic axes

$$y = c_1 e^{c_2 x} \quad \longrightarrow \quad \ln y = \ln c_1 + c_2 x$$

Similarly a simple power law transforms as follows

$$y = c_1 x^{c_2} \quad \longrightarrow \quad \ln y = \ln c_1 + c_2 \ln x$$

4.1 Fit $y = c_1 e^{c_2 x}$ to Data.

The following statements fit $y = c_1 e^{c_2 x}$ to a set of (x, y) data

```
>> x = ...
>> y = ...
>> ct = linefit(x,log(y));       %  Line fit to transformed data
>> c = [exp(ct(2));  ct(1)]      %  Extract parameters from transformation
```

4.2 Fit $y = c_1 x^{c_2}$ to Data.

The following statements fit $y = c_1 x^{c_2}$ to a set of (x, y) data

```
>> x = ...
>> y = ...
>> ct = linefit(log(x),log(y));  %  Line fit to transformed data
>> c = [exp(ct(2));  ct(1)]      %  Extract parameters from transformation
```

5 Fitting Polynomials to Data

The built-in `polyfit` function performs the least squares fit of a degree n polynomial to vectors of (x, y) data. The built-in `polyval` function evaluates a polynomial at given x points.

`polyfit`, `polyval`, and other built-in MATLAB routines for manipulating polynomials store the coefficients of the polynomial in a row vector. The coefficients of the polynomial are in order of *decreasing* powers of x.

$$y = c_1 x^n + c_2 x^{n-1} + \ldots + c_n x + c_{n+1} \tag{11}$$

The following statements obtain a polynomial curve fit and plot the data along with the fit on the same plot.

```
>> x = ...
>> y = ...
>> n = ...                          %  degree of the polynomial
>> c = polyfit(x,y,n);

>> xfit = linspace(min(x),max(x));  %  100 points in range of x
>> yfit = polyval(c,xfit);
>> plot(x,y,'o',xfit,yfit,'-');
>> legend('data',sprintf('degree %d polynomial'));
```

If specific terms in the fit function are to be omitted, e.g., if $F(x) = c_1 x + c_2$ (no constant term), then `polyfit` cannot be used. The following section shows how the NMM toolbox routines `fitnorm` and `fitqr` can be used to fit such polynomial expressions to data.

6 Fitting Arbitrary Linear Combinations of Basis Functions

The general least squares fitting problem involves the fit function

$$F(x) = c_1 f_1(x) + c_2 f_2(x) + \ldots + c_n f_n(x)$$

defined in terms of the *basis functions*, f_j, $j = 1, \ldots, n$.

Least squares fits to arbitrary, linear combinations of basis functions can be obtained with the NMM `fitnorm` and `fitqr` functions. These functions have the same syntax. The only difference is in the numerical algorithm used to solve the least squares problem. `fitnorm` solves the normal equations. `fitqr` uses QR factorization of the overdetermined coefficient matrix. In general, `fitqr` is recommended.

Both `fitnorm` and `fitqr` require the user to supply an m-file function that returns the A matrix in equation (9). This m-file is usually quite short — often only one line long. For example, the basis functions of

$$F(x) = \frac{c_1}{x} + c_2 x \tag{12}$$

are defined in the `xinvpxBasis` function

```
function A = xinvpxBasis(x)
% xinvpxBasis  Matrix with columns evaluated with 1/x and x
A = [1./x  x];
```

A curve fit of equation (12) to a set of (x, y) data is obtained with

```
>> x = ...
>> y = ...
>> c = fitqr(x,y,'xinvpxBasis');
```

Note that the `A` matrix returned by the basis function m-file *must* have m rows and n columns. If, for example, the `x` variable is a row vector then the `A` matrix returned by `xinvpxBasis` will be a long row vector. To guard agains this possibility the `xinvpxBasis` function should be rewritten as

```
function A = xinvpxBasis(x)
% xinvpxBasis  Matrix with columns evaluated with 1/x and x
A = [1./x(:)  x(:)];
```

The `x(:)` subexpressions convert `x` to a column vector before performing any mathematical operations on the elements of `x`.

6.1 Use of inline function objects

Version 5 and later of MATLAB supports the use of object oriented programming constructs. For our purposes the immediately useful aspect of the capability is in the definition of an *inline function object*. Instead of writing a separate `xinvpxBasis` function to evaluate the `A` matrix (in the preceding example), we could write

```
>> x = ...
>> y = ...
>> Afun = inline('[1./x(:)  x(:)]')
>> c = fitqr(x,y,Afun);
```

The `Afun = inline('[1./x(:)  x(:)]')` expression creates an inline function object called `Afun` that can be used just like a function defined in an external m-file. Notice that in the call to `fitqr`, `Afun` is not enclosed in quotes.

Example: Fitting Monomial Basis Functions

Suppose that the data to be fit corresponds to a known theoretical model. In that case an arbitrary polynomial fit would not be appropriate (unless, of course, the theoretical model is a polynomial). Here we demonstrate the procedure for the fit function

$$y = c_1 x^3 + c_2 \tag{13}$$

Because the x^2 and x terms are missing, Equation (13) cannot be fit with the built-in `polyfit` function.

The basis functions in Equation (13) are x^3 and 1. The following statement defines an in-line function object to evaluate the basis functions in the (two) columns of a matrix.

```
>> Afun = inline('[x(:).^3  ones(size(x(:)))]')
```

Now, suppose the data is stored in a plain text file called `myData.dat`. The following statements load the data from the file and use the `fitqr` function to obtain the coefficients of the fit. (Note that `myData.dat` is a ficticious file, so the coefficients of the fit are not printed here.)

```
>> D = load('myData.dat');     %  Read data into the D matrix
>> x = D(:,1);   y = D(:,2);   %  Store data in two column vectors
>> c = fitqr(x,y,Afun);        %  Obtain the fit using the inline fcn object
```

Finally, we evaluate the fit, and plot a comparison of the fit function with the original data.

```
>> xfit = linspace(min(x),max(x));  %  100 pts in range of original data
>> A = Afun(xfit);                  %  Columns of A are basis fcns eval at x
>> yfit = A*c;                      %  Fit function evaluated at xfit
>> plot(x,y,'o',xfit,yfit,'-');
```

7 Multivariate fits: $y = F(x_1, x_2, \ldots, x_p)$

When a single dependent variable, y, is a function of p independent variables, x_1, x_2, ..., x_p, a linear least squares fit can be obtained with fitting functions of the form

$$F(x_1, x_2, \ldots, x_p) = c_1 f_1(x_1, x_2, \ldots, x_p) + c_2 f_2(x_1, x_2, \ldots, x_p) + \ldots + c_n f_n(x_1, x_2, \ldots, x_p)$$

The simplest case is when the dependent variable is linearly related to each of the independent variables, i.e.,

$$y = F(x_1, x_2, \ldots, x_p) = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n + c_{n+1}$$

In this case the number of basis functions is one more than the number of of independent variables if the constant term c_{n+1} is included. The NMM Toolbox does not include general purpose routines for multivariate fitting. Using the \ operator it is easy to obtain such a fit in just a few lines of MATLAB code. For example, to fit

$$y = c_1 x_1 + c_2 x_3 + c_3 x_3 + c_4$$

use the following MATLAB statements

```
>> x1 = ...       %  store values of given data
>> x2 = ...
>> x3 = ...
>> y = ...
>> A = [ x1(:)  x2(:)  x3(:)  ones(length(x1(:)),1)]
>> c = A\y(:)
```

The `x1(:)`, `x2(:)`, and `x3(:)` expressions guarantee that `A` is formed from four column vectors. This will not be necessary if `x1`, etc. are already a column vectors. The `c = A\y` expression solves the least squares problem $Ac = y$. Since A is $m \times n$, the \ operator performs a QR factorization of A and then obtains the least squares solution, via equation (10).

References

[1] G. W. Recktenwald. *Numerical Methods with* MATLAB*: Implementations and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 2000.

[2] G. Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, Wellesley, MA, 1986.