Armstrong State University
Engineering Studies
MATLAB Marina – Curve Fitting Primer

Prerequisites
The Curve Fitting Primer assumes knowledge of the MATLAB IDE, MATLAB help, arithmetic operations, built in functions, scripts, variables, arrays, logic expressions, conditional structures, iteration, functions, debugging, characters and strings, cell arrays, structures, file input and output, 2D plotting, and interpolation. Material on these topics is covered in the MATLAB Marina Introduction to MATLAB module, MATLAB Marina Variables module, MATLAB Marina Arrays module, MATLAB Marina Logic Expressions module, MATLAB Marina Conditional Structures module, MATLAB Marina Iteration module, MATLAB Marina Functions module, MATLAB Marina debugging module, MATLAB Marina Character and Strings module, MATLAB Marina Cell Arrays module, MATLAB Marina Structures module, MATLAB Marina File Input and Output module, MATLAB Marina Plotting module, and MATLAB Marina Interpolation Module.

Learning Objectives
1. Be able to use MATLAB to fit polynomial curves to data.
2. Be able to determine the sum of squared error of a curve fit to data.
3. Be able to use MATLAB to estimate trends in data using curve fitting.
4. Understand when curve fitting is appropriate to use.

Terms
curve fitting, error, sum of squared error, noisy data

MATLAB Functions, Keywords, and Operators
polyval, polyfit, poly, roots

Curve Fitting
Curve fitting involves finding a function (typically a polynomial) that best matches a set of data. Typically this involves minimizing the sum of squared difference (least-squared error) between the value of the function and the actual value. Curve fitting is useful for data visualization and determining the physical meaning behind the data, determining trends in the data, and determining a formula describing the data.

Linear regression is a process that determines best fit linear equation to a set of data and polynomial regression is a process that determines best fit polynomial equation of order n to a set of data. Linear regression is a form of polynomial regression where the polynomial order n = 1. Best fit here means that the square of the difference between the original data points and the points on the polynomial function is minimized, i.e. minimizes square of the errors. Higher order polynomials are needed for data with more variation than relatively smoothly shaped data. For original data x and y with length L and an nth order polynomial $f(x)$ given by

$f(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n$, the sum of the squared error is $e = \sum_{k=1}^{L} \left(y_k - f(x_k)\right)^2$.

MATLAB has two built in functions for curve fitting using polynomial regression: `polyfit` and `polyval`. The function `polyfit` determines the coefficients of a polynomial of the specified degree that best fits the data. The function `polyval` evaluates a polynomial with the specified coefficients at the specified points.

The MATLAB program of Figure 1a illustrates using `polyfit` and `polyval` to determine a best fit polynomial to a set of data and then using the polynomial to evaluate new points on the function. The plot of Figure 1b shows the original noisy data and the best fit polynomial functions for degree 1, 2, and 3.

```
% generate 2nd order polynomial data
p = [1, -2, 7];   % x^2 - 2x + 7 = 0
x = [-5 : 0.1 : 5];
y = polyval(p, x);

% corrupt the data to simulate measured noisy data
ynoisy = y + 5*(rand(size(y))- 0.5);
figure(1)
plot(x,ynoisy,'x'), xlabel('x'),ylabel('y'), title('Measured Data')

% fit the data using polyfit and degree 1, 2 and 3 polynomials
coef1 = polyfit(x,ynoisy,1);
coef2 = polyfit(x,ynoisy,2);
coef3 = polyfit(x,ynoisy,3);

% plots corresponding to best fit curves using original points
yfit1 = polyval(coef1,x);
yfit2 = polyval(coef2,x);
yfit3 = polyval(coef3,x);
figure(2)
plot(x,ynoisy,'kx',x,yfit1,'c',x,yfit2,'b',x,yfit3,'g')
xlabel('x'),ylabel('y'), title('Polyfit Results')
legend('data','degree 1 fit','degree 2 fit','degree 3 fit');
```

Figure 1a, Curve Fitting Example using polyfit and polyval

The statement `coef1 = polyfit(x,ynoisey,1)` determines the polynomial coefficients for the best fit polynomial function of degree one. Polynomials are represented in MATLAB by a vector of the polynomial coefficients. For example, the polynomial $2x^2 - 4x + 7$ is represented in MATLAB as the vector $[2,\ -4,\ 7]$. The statement `yfit1 = polyval(coef1,x)` evaluates the polynomial function with coefficients `coef1` at the original x values.

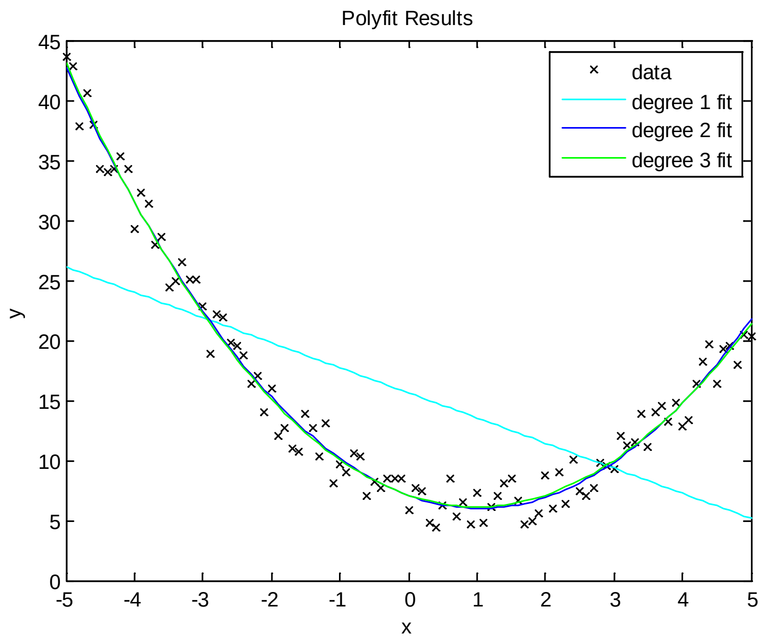

Figure 1b, Curve Fitting Example using polyfit and polyval

Note in the plot of Figure 1b that when using `polyfit` and `polyval` for curve fitting that the original data points are not necessarily on the best fit function.

The sum of the squared error (least squared error) for each fitted curve from the code in Figure 1a can be computed with the following MATLAB code segment

```
efit1 = sum((ynoisy – yfit1).^2);
efit2 = sum((ynoisy – yfit2).^2);
efit3 = sum((ynoisy – yfit3).^2);
```

For the example of Figure 1a and 1b, the sum of the squared errors is given in Figure 2.

Order n	Sum of Squared Error
1	6102.5
2	214.4
3	212.0
7	203.6
11	198.7

Figure 2, Sum of Squared Errors for Example of Figure 1a and 1b

In general, as the order of the polynomial increases, the error decreases but sometimes depending on the data a lower order polynomial fit yields a lower error. One must also keep in mind that the order of the polynomial function to fit to the data must be much less than the total number of points. For example, fitting a sixth order polynomial to a set of data with only 10 points would not generally yield a good result.

Additional Useful MATLAB Functions for Polynomials

Some other useful MATLAB functions when working with polynomials are `roots` and `poly`. The `roots` function determines the roots of a polynomial specified by the polynomial coefficients. The `poly` function determines the polynomial coefficients corresponding to the roots. The `roots` and `poly` functions are inverses of each other for vector data.

Figure 3 shows an example of using the MATLAB `roots` and `poly` functions to first determine the roots of the third order polynomial $f(x) = x^3 - x^2 - 2.75x + 1.5$ and then use the roots to determine the corresponding polynomial coefficients.

```
>> coef = [1, -1, -2.75, 1.5];
>> r = roots(coef);
r =
   -1.5000
    2.0000
    0.5000
>> c = poly(r);
c =    1.0000   -1.0000   -2.7500    1.5000
```

Figure 3, MATLAB `roots` and `poly` Functions

General Guidelines on Interpolation and Curve Fitting

Data collected from the field or experiments is typically discrete. The data often comes from sensors that periodically measure values of interest. The physical meaning of the data may not be known.

Interpolation involves estimating values between the known sampled data points. Estimating values beyond the range covered by the collected data is called extrapolation. The interpolation function passes through all the data points and can be thought of as "exact" curve fitting. Interpolation does not yield a single analytical function for the data. Interpolation is typically used when the data has little uncertainty (little or no error in the measurements) and one needs estimates of values in between the data values. Curve fitting involves finding an analytical function (typically a polynomial function) that best matches the data. The approximating function may not pass through the data points. Curve fitting is typically used when the data has error (noisy measurements) and one wants to determine trends in the data or determine an analytical formula describing the data.

Both interpolation and curve fitting can be used to estimate values from measured data. Interpolation works well when the data is reliable but not as well for less reliable data.

Last modified Thursday, November 13, 2014