CHEMICAL ENGINEERING 2E04

Chapter 3 – Curve Fitting

Module 3A: Introduction & Function Regression

Dr. Jake Nease

Kieran McKenzie

Steven Karolat

Cynthia Pham

Chemical Engineering

McMaster University

Updated October 21, 2019

Contents

Supplementary Material

Suggested Readings

Gilat, V. Subramaniam: *Numerical Methods for Engineers and Scientists.* Wiley. Third Edition (2014)

- Least-Squares Regression using Polynomials
  - Chapter 6 → *6.2.2, 6.3, 6.4*
- Basis Function Regression
  - Chapter 6 → *6.8*

Overview/Applications

In many real-world scenarios, a mathematical formula for a phenomenon is not known, rather, a set of collected data points exists instead. This is where curve fitting will come to play!

Curve Fitting

Curve fitting is the process by which a *mathematical expression* that *best fits* a set of data is chosen.

What does it mean to "best fit": Best fit refers to the *least amount of error*. There are various methods that exist to minimize error between a curve and data points on a graph, each based on unique processes – therefore, one method of minimizing error may give a slightly different 'curve of best fit' than another method.

Our approach to finding the best fit: *Least squares regression*.

Curve fitting can be used in almost any scenario or experiment in which we need to go from collected data points to a continuous model. This occurs very often in chemical engineering, as well as in countless other disciplines – science, finance, economics, statistics – it's everywhere!

Curve fitting is known as a *GREY BOX* modeling technique. We do not always know the fundamental physics or phenomena of our system, but that is not as important as being able to relate independent information to outcomes of interest.

Example: Cynthia's Avocado Tree

Suppose Cynthia is growing an avocado tree and she decides to take measurements once every day at 8:00 am and plots them on a graph. What if she wants to find out how tall her plant has grown at 3.5 days? Or what if she wants to estimate how tall her plant will be beyond her recorded time measurements? How can she do this?

Interpolation and Extrapolation

- *Interpolation:* Estimating values *between* a set of non-continuous data points by fitting a continuous equation to the data set.
- *Extrapolation:* Estimating values *outside* the collected data points using the continuous equation fitted to the data set.

Note: There is typically error inherent in the collection of data points, so the fitted curve will also contain a certain amount of error. Keep in mind, a 'best fit' is usually not a 'perfect fit'.

Together, we will go over a selected number of curve fitting methods. Depending on the situation, a certain method of curve fitting may be selected over another.

Primary Learning Outcomes

- Introduce curve fitting using *least-squares regression*.
- Recognize *patterns* for linear, quadratic and polynomial least-squares regression to put it into $\boldsymbol{Ax} = \boldsymbol{b}$ form!
- Expand our patterns to form *basis function regression*.

Note: Sometimes curve fitting is referred to as regression. For example, linear regression corresponds to fitting a linear polynomial to a set of data, quadratic regression refers to the use of a quadratic polynomial, and so on.

Linear Least-Squares Regression

Least squares regression is the process of *minimizing the sum of the square of the errors (SSE)*. See Figure 1 on the next page to see what 'errors' are defined as. The curve with parameters that give the minimum SSE will be the regression curve! Let's have a look at different forms of least squares regression…

Using regression tools, such as linear least-squares regression, we can identify the parameters of a *linear polynomial* explicitly. Remember drawing lines of best fit by hand? That is your past now.

Recall that least squares regression squares the errors between plotted data points on the graph and an arbitrary line through the data – the *sum of these errors* is dependent on the parameters of the line (a_1 and a_0 in the line equation, $y = a_1x + a_0$), and must be *minimized* in order to achieve the minimum squared error.

Imagine This:

You're sitting in your high school math class, exhausted – you cannot wait for the day to be over. Your mind wanders… Suddenly, your teacher calls on you, asking you to please explain how to find the extrema (maximum or minimum) of a polynomial, $f(x)$. After a long pause, you remember!

- Find the derivative of the polynomial
- Solve for $f'(x) = 0$. Because $f'(x)$ represents the slope of the original function at any point, solving where $f'(x) = 0$ will output a "stationary" point - the location of a local maximum or minimum at which the slope of $f(x)$ is zero!

The concept of least squares regression uses this same idea. We wish to *find the minimum* of the sum of squared errors. This process also uses derivatives, setting them equal to zero. However, instead of one derivative, we'll be required to use several partial derivatives – we'll see why!

Suppose you have a set of data points:

$$(x_1, y_1), (x_2, y_2), \dots (x_N, y_N)$$

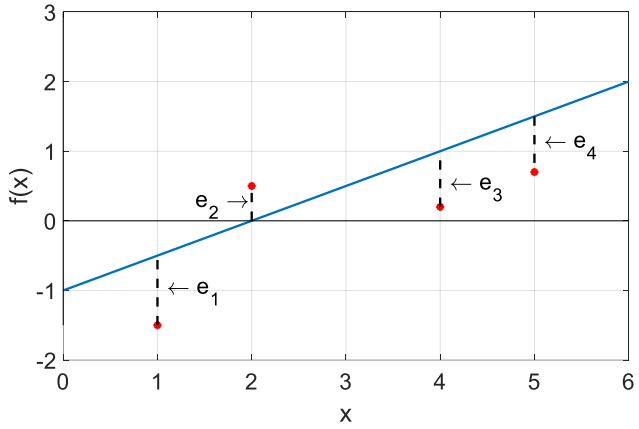
You wish to approximate these data points by a regressing a line to the data in the form:

$$\hat{y} = a_1x + a_0$$

A Note on Notation

To indicate a function's value relating to an input x-value, it is customary to use the notation $y = f(x)$. Alternatively, to indicate the *predicted* value of a function at an input x-value, the notation $\hat{y} = f(x)$ is often used.

We wish to minimize the errors between the plotted line and the data points to achieve a regression curve. Graphically, the error e_i at any point x_i can be represented as the difference between $\hat{y}_i$ and y_i at x_i. The plot below shows a line plotted through a set of data points $(x_1, y_1) \dots (x_4, y_4)$, with errors $e_1, \dots, e_4$ with $e_i = (y_i - \hat{y}_i)$ shown.



x_i	y_i	$\hat{y}_i$	$\lvert e_i \rvert$
1	-1.5	0.5	2
2	0.5	0	0.5
4	0.2	1	0.8
5	0.7	1.5	0.8

Figure 1: An example of plotted data points showing error between an arbitrary line of best fit and the data set. Note: The line plotted above is arbitrary – no attempt has (yet) been made to make this a regression line for our data!

We seek the terms a_1 and a_0 in the equation $\hat{y} = a_1 x + a_0$ that will minimize the deviation of $\hat{y}$ from y (minimizing e_i as much as possible). We achieve this result through the rigorous use of *unconstrained optimization*.

For reasons that may become clear later, the typical selection for an objective function for linear regression is the sum of squared errors (SSE). SSE is represented as:

$$SSE = \sum_{i=1}^{N} e_i^2$$

$$\Downarrow$$

$$SSE = \sum_{i=1}^{N} (\hat{y}_i(x_i) - y_i)^2$$

Notice that the upper limit on the summation, N, refers the number of data points in the set. If we are using the form of a linear regression, we may replace $\hat{y}_i(x_i)$ with the expression $\hat{y}_i(x_i) = a_0 + a_1 x_i$:

$$SSE = \sum_{i=1}^{N} (a_0 + a_1 x_i - y_i)^2$$

Since we seek the terms a_1 and a_0 that will minimize the deviation of $\hat{y}_i$ from y_i, we want to find the *minimum* of SSE. The formal optimization formulation for this problem becomes:

$$\min_{a_0 \; a_1} SSE \triangleq \sum_{i=1}^{N} (a_0 + a_1 x_i - y_i)^2$$

(unconstrained)

Note: We have chosen to define $SSE = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$*. However, the term inside the parentheses is squared, so the order of subtraction does not matter (i.e. you can also do* $y_i - \hat{y}_i$ *in the parentheses).*

Visualizing SSE as a Function of Parameters

Using the data points from the plot above, it may help to visualize SSE as a function of a_0 and a_1. In Figure 2, SSE is plotted as a function of a_0 and a_1 for our data set. Figure 2 shows an overall view of SSE, while Figure 3 shows a contour plot zooming in around the minimum of the surface. It should be clear what values of a_0 and a_1 should be used.

Formal Optimization

As it turns out, the optimization problem above is *unconstrained* and *convex*, which means that we may apply derivative-based methods to locate the so-called *global solution* (best possible answer). This is very rarely the case but is one of the *most important aspects of least-squares regression* and is fundamental to its implementation. For information on how to solve constrained optimization programs with real-world applications, consider ChE 4G03 in your final year.

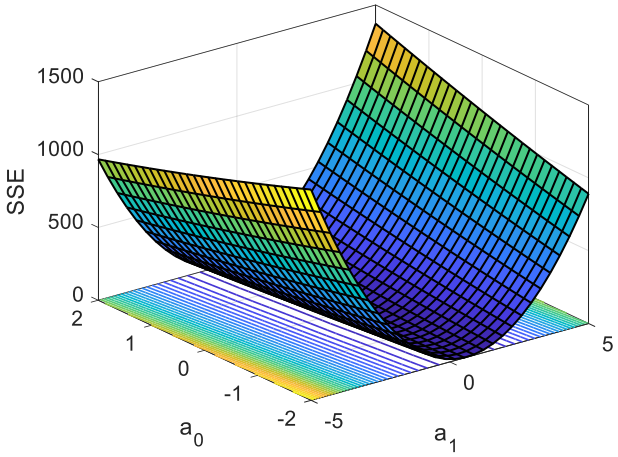


Figure 2: A surface plot showing SSE as a function of a_0 and a_1 for our data set.

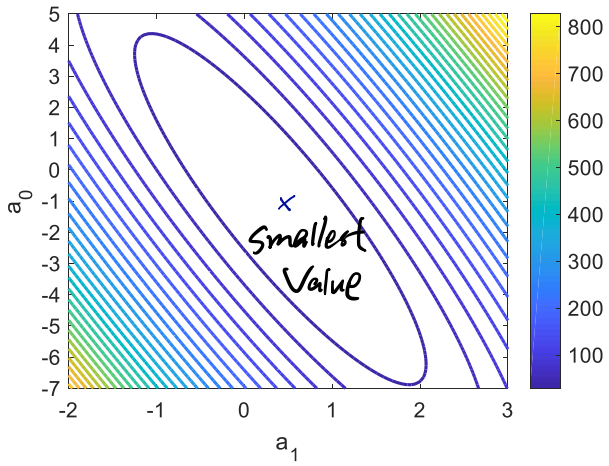


Figure 3: The contour plot of the SSE as a function of a_0 and a_1 for our data set showing where we can find the minimum SSE.

Alright, so we can see a general region in which we can find the minimum SSE. But how can we *calculate* this minimum value analytically? That is, how do we minimize a function of two variables, instead of just one variable?

Solving an Unconstrained Optimization for the Minimum of SSE

Again, flashback to how you are used to finding the minimum of a function of x:

1. Derive that function with respect to our unknown variable (x).
2. Set this derivative equal to zero – there is now one unknown, and one equation, so we can solve for the unknown x-value. When this x is inputted into our function of x, that function will output its *minimum value*.

Question: How can we be sure that we're finding the minimum and not the maximum when we take the derivative and set it equal to zero?

Answer: Of course, you can't be certain to get one or the other on *all* equations. However, we're *guaranteed to find the minimum of our SSE equation*! This is because of the nature of the function - SSE is a *CONVEX quadratic*, so there's no maximum to find, only a minimum. As matter of fact, it can be mathematically shown that there is only *one* minimum (not always the case for nonlinear equations).

We have SSE defined as a function of a_0 and a_1. Because we are fitting a polynomial to pre-gathered data points, x_i *and* y_i *are known!* We have two unknowns, so we'll need two equations to solve for them! How do we form two equations, though?

1. Take the partial derivative of SSE with respect to one unknown, say a_0.
2. Take the partial derivative of SSE with respect to the second unknown, a_1.
3. Now we have two equations and two unknowns! Set each derivative to zero and solve for the two unknown parameters!

This can be done by use of either analytical or matrix methods – for practical purposes, *matrix methods are by far preferred*. Let's do it together.

Starting with the formula we defined previously for SSE:

$$SSE \quad = \quad \sum_{i=1}^{N} (a_0 + a_1 x_i \; - y_i)^2$$

Let's take the partial derivative of the above expression with respect to each of the *regression coefficients* and set them to 0:

Partial WRT $\boldsymbol{a_0}$		Partial WRT $\boldsymbol{a_1}$	
$\frac{\partial SSE}{\partial a_0} =$	$\sum_{i=1}^{N} 2(a_0 + a_1 x_i - y_i)$	$\frac{\partial SSE}{\partial a_1} =$	$\sum_{i=1}^{N} 2(x_i)(a_0 + a_1 x_i - y_i)$
$0 =$	$\sum_i 2a_0 + \sum 2a_1 x_i - 2\sum y_i$	$0 =$	$\sum_i a_0 x_i + \sum_i a_1 x_i^2 - \sum y_i x_i$
$0 =$	$\sum a_0 + \sum a_1 x_i - \sum y_i$ $a_0 \sum 1$	$0 =$	

Next, Isolate the terms attached to variables and move all constant terms to the other side of the equation:

$a_0 \sum 1 + a_1 \sum x_i$	$= \sum_{i=1}^{N} y_i$
$\sum a_0 x_i + \sum a_1 x_i^2$ $a_0 \sum x_i + a_1 \sum x_i^2$	$= \sum_{i=1}^{N} y_i x_i$

Do these equations bring back memories? I hope you see what I see! This is a set of linear equations for which we can solve the system $A\boldsymbol{x} = \boldsymbol{b}$:

$$\underbrace{\begin{bmatrix} \sum 1 & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} y_i x_i \end{bmatrix}}_{b}$$

Solve,

Solve this system using any numerical method from our first unit, and voilà! We now have the necessary values to plug into our regression line equation, $\hat{y} = a_1 x + a_0$. Keep in mind that the entries in our matrix A and $\boldsymbol{b}$ are all constants (after the summations are performed using the data).

Linear Least-Squares Regression

For a set of data points (x_i, y_i), the equation of a line that minimizes SSE will have the form:

$$\hat{y} = a_0 + a_1 x$$

Where the parameters a_0 and a_1 can be solved for by use of a linear system, $Ax = b$, where N is the number of data points being regressed:

$$\underbrace{\begin{bmatrix} \sum_{i=1}^{N} 1 & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} y_i x_i \end{bmatrix}}_{b}$$

After performing row operations, the matrix system above can be solved for both a_1 and a_0.

Workshop: Creating a **MATLAB** code for linear least squares regression.

The MATLAB code for linear least squares regression will incorporate a method for solving linear systems, but with a twist at the beginning: we have to set up the matrices ourselves! The simplified code for solving your matrix system has been provided with blank space to create your linear least-squares regression matrix from the given set of data points:

```
function [a1,a0] = linregress(xvec,yvec)
% INPUTS:
%   xvec = vector containing x data information
%   yvec = vector containing y data information
% OUTPUTS:
%   a1 and a0 within the equation y_R = a1x + a0
  n = length(xvec)
A = [ n          sum(xvec)
      sum(xvec)  sum(xvec.^2) ];
b = [sum(yvec)  sum(yvec.*xvec)]' ;

% Solving the matrix can be done with the built-in function backslash
solution= A\b;

a1=solution(2);

a0=solution(1);

end
```

Quadratic Least-Squares Regression

To fit a higher-order polynomial, we follow a very similar process to linear least-squares regression! The error between data points and our curve must still be minimized – the only difference being that we will have more unknown parameters to solve for, depending on the order of polynomial used. This will require us to form more equations to solve for those unknowns, but otherwise, the process remains the exact same.

Now we'll fit a quadratic polynomial, $\hat{y} = a_0 + a_1 x_i + a_2 x_i^2$, to a set of data points by minimizing the error between the curve and those points. The formula for the sum of squared errors becomes:

$$SSE = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 = \sum_{i=1}^{N} (a_0 + a_1 x_i + a_2 x_i^2 - y_i)^2$$

Now solving for the parameters a_2, a_1, and a_0 which will minimize SSE. We will require the use of three partial derivatives now to solve for these three unknowns, setting each derivative equal to zero to minimize it:

Partial WRT $\boldsymbol{a_0}$		Partial WRT $\boldsymbol{a_1}$	
$\frac{\partial SSE}{\partial a_0} =$	$\sum 2 \cdot 1 \cdot (a_0 + a_1 x_i + a_2 x_i^2 - y_i)$	$\frac{\partial SSE}{\partial a_1} =$	$\sum 2 \cdot x_i (a_0 + a_1 x_i + a_2 x_i^2 - y_i)$
$0 =$	$\sum 2a_0 + \sum 2a_1 x_i + \sum 2a_2 x_i^2 - \sum 2y_i$	$0 =$	$\sum 2a_0 x_i + \sum 2a_1 x_i^2 + \sum 2a_2 x_i^3 - \sum 2x_i y_i$
$0 =$		$0 =$	
Partial WRT $\boldsymbol{a_2}$		As a LSOE	
$\frac{\partial SSE}{\partial a_2} =$	$\sum 2 \cdot x_i^2 (a_0 + a_1 x_i + a_2 x_i^2 - y_i)$	$\underbrace{\begin{bmatrix} N & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \sum y_i \\ \sum y_i x_i \\ \sum y_i x_i^2 \end{bmatrix}}_{b}$	
$0 =$	$\sum 2a_0 x_i^2 + \sum 2a_1 x_i^3 +$		
$0 =$			

After solving this system, you will have all parameters required to plot a quadratic regression curve in the form $\hat{y} = a_2 x^2 + a_1 x + a_0$! Using this algorithm, check out an example:

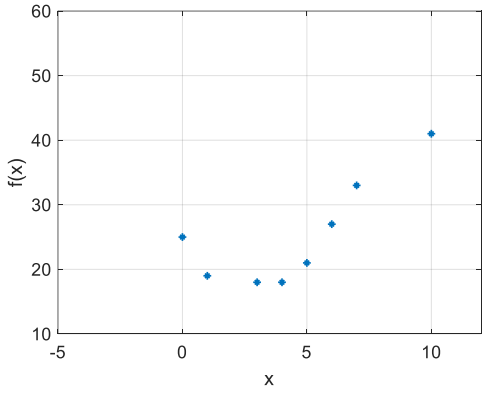


Figure 4: Quadratic shaped scatter plot.

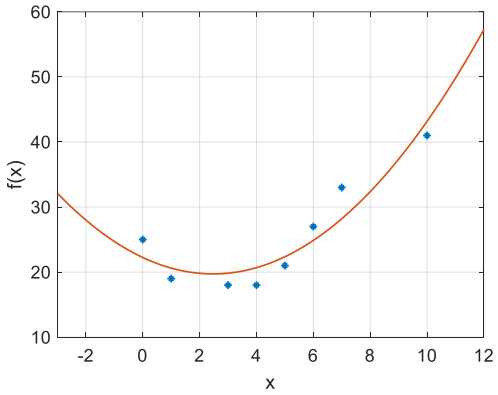


Figure 5: Quadratic regression curve fitted to the data points.

Now that's powerful stuff.

Quadratic Regression

For a given set of known data points (x_i, y_i), a quadratic regression curve will have the form:

$$\hat{y} = a_0 + a_1 x_i + a_2 x_i^2$$

Where the parameters can be solved for by use of a linear system, $A\boldsymbol{x} = \boldsymbol{b}$:

$$\begin{bmatrix} \sum_{i=1}^{N} 1 & \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i^3 \\ \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i^3 & \sum_{i=1}^{N} x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} y_i x_i \\ \sum_{i=1}^{N} y_i x_i^2 \end{bmatrix}$$

Isn't that great? I suppose it looks like we've done a lot of hard-core mathematics to get here, but all we have been doing to this point has been finding the minima of the SSE function. It is a simple idea in theory, yet we can expand it to do so much!

There are a few observations we can make from this exercise:

1. Quadratic regression looks *eerily similar* to linear regression
2. The matrix of coefficients A in our system of equations is *symmetric*
3. The RHS vector $\boldsymbol{b}$ also exhibits a pattern directly related to the form of our regression equation $\hat{y} = f(x)$

Figure 6: All we're doing is minimizing SSE!

Generalization: Least-Squares Regression using Polynomials

Both quadratic and linear polynomial regression have been covered so far, but our process can be extended to any order of polynomial!

Recall the formulaic approaches used to solve for linear and quadratic polynomial regression:

Linear Regression

$$\underbrace{\begin{bmatrix} \sum_{i=1}^{N} 1 & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} y_i x_i \end{bmatrix}}_{b}$$

Quadratic Regression

$$\begin{bmatrix} \sum_{i=1}^{N} 1 & \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i^3 \\ \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i^3 & \sum_{i=1}^{N} x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} y_i x_i \\ \sum_{i=1}^{N} y_i x_i^2 \end{bmatrix}$$

What if I told you that we can identify a pattern (great for coding!) and fit any order of polynomial as follows:

Least-Squares Polynomial Regression of Order $\boldsymbol{n}$

For a set of known data points (x_i, y_i), an n^{th}-order polynomial regressed through these points will have the form:

$$\hat{y} = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} + a_n x^n$$

Where the parameters $a_0 \ldots a_n$ can be solved for via the system of linear equations, A is an $(n+1) \times (n+1)$ symmetric matrix of the form, x is the solution vector consisting of the unknown parameters and b is a vector that holds all known constants.

$$\begin{bmatrix} \sum_{i=1}^{N} 1 & \cdots & \sum_{i=1}^{N} x_i^{n-1} & \sum_{i=1}^{N} x_i^n \\ \vdots & \vdots & \cdots & \vdots \\ \sum_{i=1}^{N} x_i^{n-1} & \ddots & \sum_{i=1}^{N} x_i^{2n-2} & \sum_{i=1}^{N} x_i^{2n-1} \\ \sum_{i=1}^{N} x_i^n & \cdots & \sum_{i=1}^{N} x_i^{2n-1} & \sum_{i=1}^{N} x_i^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \vdots \\ \sum y_i x_i^{n-1} \\ \sum y_i x_i^n \end{bmatrix}$$

Note: The degree of x_i in the A matrix decreases by 1 each time when descending either a row or a column. The coefficient on x_i^0 will always be N, the number of data points.

And there you have it! The ultimate general formula, which can be used to fit *ANY* order of standard polynomial to a set of data - linear, quadratic, cubic, you name it.

Choosing a Polynomial Form to Represent a Set of Data

With a general formula to fit any n^{th} order polynomial to a set of data, the question arises: "Which order of polynomial should I choose to represent my data?" Before this question is answered, we'll investigate a different question: "Which order of polynomial will always perfectly fit my data?" You may be shocked to discover that the answers to these questions are not always the same. Let's think why:

You have *one point* on a graph:

- An infinite number of possible lines/higher order polynomials can be drawn through this point with zero error.

You have *two points* on a graph:

- One unique line exists which can be plotted to *perfectly* pass through both points.
- An infinite number of higher order polynomials can be drawn through these points with zero minimum error.

You have *three points* on a graph:

- Data can no longer be fit by a line without error involved.
- One unique quadratic exists which can perfectly fit all points with zero error.
- An infinite number of higher order polynomials can be drawn through these points with zero minimum error.

You are likely noticing the pattern:

Perfectly Fitting a Set of Data

- An n^{th} order polynomial can fit any set of $n+1$ data points perfectly, with minimum error $= 0$.
  - $n+1$ equations can be formed to solve for $n+1$ parameters, allowing for zero degrees of freedom in our system $Ax = b$! Here, only one possible solution exists.
- Plotting an i^{th} order polynomial to a set of $n+1$ data points, where $i < n$, will require polynomial regression (minimum error $\neq 0$, but is minimized).
  - The system $Ax = b$ will have $(n+1-i)$ degrees of freedom and will be underspecified. More parameters than equations exist, and so infinite solutions exist; however, only one optimized solution (which is what we solve for).

Great! So, we know that if we have $n+1$ data points, we can always perfectly represent the points with an n^{th} order polynomial, with *zero error*. So, why wouldn't we do this every time? The answer lies below, where various ordered polynomials are fit to the same set of 12 data points:

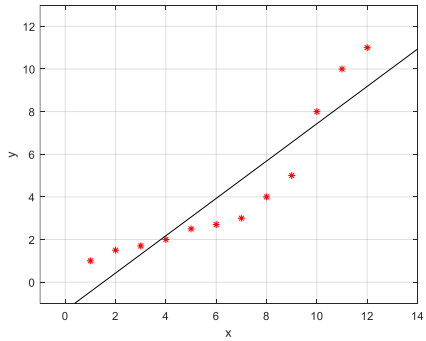


Figure 7: Regression for 1^{st} order polynomial for 12 data points.

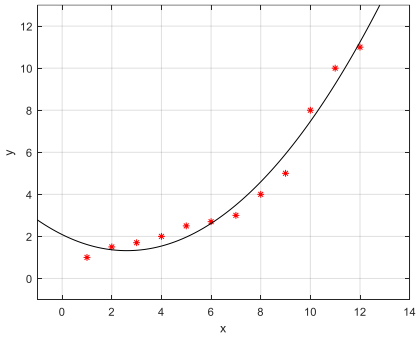


Figure 8: Regression for 2^{nd} order polynomial for 12 data points.

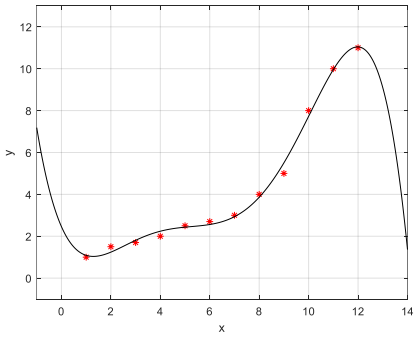


Figure 9: Regression for 3^{rd} order polynomial for 12 data points.

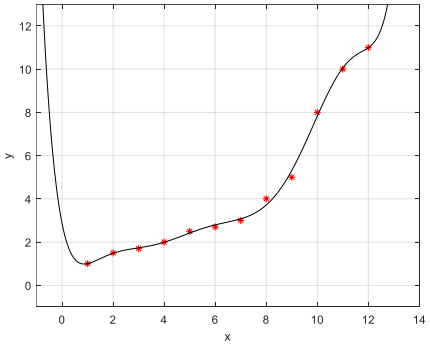


Figure 10: Regression for 8^{th} order polynomial for 12 data points.

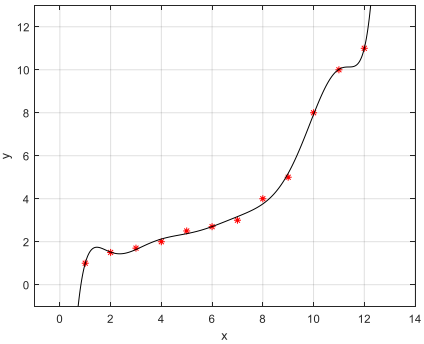


Figure 11: Regression for 9^{th} order polynomial for 12 data points.

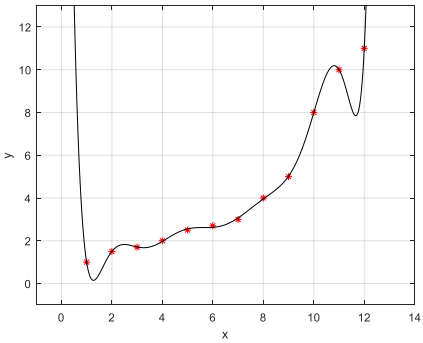


Figure 12: Regression for 11^{th} order polynomial for 12 data points.

$$\hat{y} = \underbrace{1}_{f_1} a_0 + a_1 \underbrace{\sin(x)}_{f_2} + a_2 \underbrace{x}_{f_3}$$

only for single x func. inside.

$$\min_{a_0, a_1, a_2} SSE \triangleq \sum_i (\hat{y}_i - y_i)^2$$

$$\begin{bmatrix} \sum 1 & \sum \sin(x_i) & \sum x_i^3 \\ \sum \sin(x_i) & \sum (\sin(x_i))^2 & \sum x_i^3 \sin(x_i) \\ \sum x_i^3 & \sum x_i^3 \sin(x_i) & \sum x_i^6 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum y_i \sin(x_i) \\ \sum y_i x_i^3 \end{bmatrix}$$

$$\begin{bmatrix} f_1 f_1 & f_1 f_2 & f_1 f_3 \\ f_2 f_1 & f_2 f_2 & f_2 f_3 \\ f_3 f_1 & f_3 f_2 & f_3 f_3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y f_1 \\ y f_2 \\ y f_3 \end{bmatrix}$$

There are vast differences in the shapes of the fitted curves. We know that the higher order polynomials fit the data set more closely however there are some drawbacks.

Drawbacks of Higher Order Polynomials

1. Higher order polynomials exhibit more *erratic behaviour* (as seen in the plots).
   - For an extreme example, imaging the results of extrapolating this data with a 9^{th} order polynomial, compared to the results of extrapolation with an 11^{th} order polynomial – there is a big difference!
2. They can become *inaccurate between* data points.
   - The 11^{th} order polynomial fits this data perfectly, but there are large deviations in the curve between some plotted points (especially between the two last points)
   - In these areas, the general trend of the curve is clearly represented inaccurately
   - This tends to mean we *over fit* our data
   - In some cases, these deviations can become frequent and troublesome and are referred to as *noise*

Both issues can be managed by fitting a lower order polynomial which can be decided depending on the error or any background information you have about your data set.

For the reasons above, higher ordered standard polynomials are often considered *unreliable for both extrapolation and interpolation* of data sets. However, there are various ways to check if you have over fit your data.

Checking if You Have Over-fit Your Data – Training and Testing

Training and testing data sets is a method to check if you have over fit your data set. This method involves the following two steps:

1. Randomly taking 80% of a data set as *training* data – fit any n^{th} order polynomial to this data.
2. The remaining data points are then used to *test* your data to see if the fitted curve accurately represents these points.

If there is excessive noise in your fitted curve, you will find that the testing data will not be accurately represented by the curve that you fit!

Basis Function Regression

Univariate Basis Function Regression

We can tweak what we know about regression to tackle systems that are best described by a *basis function*. A basis function is a function that can be used to further describe the relationship *beyond* just a coefficient and a variable.

Instead of regressing around the variable (*i.e.* finding a_1 that best fits $a_1 x$ to the data), we can find the coefficient, a_1, that *best fits* $a_1 f_1(x)$ *to the data*. Examples of $f_n(x)$ could be any function such as, 4^x, $tan\,(x + x^2)$! Putting it all together, an example of a three-basis function system, we regress the below function to the data:

$$\hat{F}(x) = a_1 f_1(x) + a_2 f_2(x) + a_3 f_3(x)$$

It is important to note $\hat{F}$ can include *as many* basis functions f_n as we deem appropriate – the below derivation continues off the above $\hat{F}$ system, but you will notice a pattern that can be expanded to any size system.

Using the function above, the equation for SSE becomes:

$$SSE = \sum_{i=1}^{N_x} (\hat{F}(x_i) - F(x_i))^2 \quad = \quad \sum_{i=1}^{N_x} \left(\sum_{m=1}^{N_c} a_m f_m(x_i) - F(x_i) \right)^2$$

Where, $m \in \{1, \dots, N_c\}$ is the index of the basis functions, and a_m is the coefficient for the m^{th} basis function. N_x refers to x's data points. For the sake of the derivation, we can ignore the first two summations that sum across the points for now. We can do this because these summations are across every term – like a common factor, we'll add them back in the end. For now, we'll just talk about the *squared error term* (SE):

$$SE = \left(\sum_{m=1}^{N_c} a_m f_m(x_i) - F(x_i) \right)^2$$

Take the derivative of SE with respect to each a_m (using the chain rule), setting all resulting equations equal to 0:

$$\frac{\partial(SE)}{\partial a_1} = 2[\{ a_1 f_1(x_i) + a_2 f_2(x_i) + a_3 f_3(x_i) \} - F(x_i)] \times \big(f_1(x_i)\big) = 0$$

To keep things tidy, we'll drop the (x_i) for the remainder of the derivation, making the previous equation:

$$\frac{\partial(SE)}{\partial a_1} = 2[\{ a_1 f_1 + a_2 f_2 + a_3 f_3 \} - F] \times (f_1) = 0$$

By rearranging the above, we get:

$$a_1 f_1 f_1 + a_2 f_2 f_1 + a_3 f_3 f_1 = f_1 F$$

Notice that f_1 is a part of each term, while a_m is still paired with its corresponding basis function, f_m, and our original data points, $F(x_i)$ are on the RHS.

As complicated and as busy as the above equation *seems*, the *only pieces that are unknown are the basis function coefficients,* a_m. When you perform the same derivative with respect to the other coefficients, you will get the following:

$$\frac{\partial(SE)}{\partial a_2} = 2[\{ a_1 f_1 + a_2 f_2 + a_3 f_3 \} - F] \times (f_2) = 0$$
$$\frac{\partial(SE)}{\partial a_3} = 2[\{ a_1 f_1 + a_2 f_2 + a_3 f_3 \} - F] \times (f_3) = 0$$

After rearranging, a 3×3 system of equations is formed in which a_m can easily be solved for. The only differences across the equations is which basis function is the common multiplier – these have been color coded.

$$a_1 f_1 f_1 + a_2 f_2 f_1 + a_3 f_3 f_1 = f_1 F$$
$$a_1 f_1 f_2 + a_2 f_2 f_2 + a_3 f_3 f_2 = f_2 F$$
$$a_1 f_1 f_3 + a_2 f_2 f_3 + a_3 f_3 f_3 = f_3 F$$

Writing this in matrix form:

$$\underbrace{\begin{bmatrix} f_1 f_1 & f_2 f_1 & f_3 f_1 \\ f_1 f_2 & f_2 f_2 & f_3 f_2 \\ f_1 f_3 & f_2 f_3 & f_3 f_3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} f_1 F \\ f_2 F \\ f_3 F \end{bmatrix}}_{b}$$

Now that everything is nicely set up, let's reintroduce our summations (because we can't forget about them).

$$\underbrace{\begin{bmatrix} \sum_{i=1}^{N_x} f_1 f_1 & \sum_{i=1}^{N_x} f_2 f_1 & \sum_{i=1}^{N_x} f_3 f_1 \\ \sum_{i=1}^{N_x} f_1 f_2 & \sum_{i=1}^{N_x} f_2 f_2 & \sum_{i=1}^{N_x} f_3 f_2 \\ \sum_{i=1}^{N_x} f_1 f_3 & \sum_{i=1}^{N_x} f_2 f_3 & \sum_{i=1}^{N_x} f_3 f_3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \sum_{i=1}^{N_x} f_1 F \\ \sum_{i=1}^{N_x} f_2 F \\ \sum_{i=1}^{N_x} f_3 F \end{bmatrix}}_{b}$$

What we just walked though was the derivation of a system of equations to fit a curve with three basis functions to a 1-dimensional data set. But from that, you can see the pattern that we can exploit to extend to N_c basis functions.

Multivariate Basis Function Regression

This pattern can even be exploited beyond univariate basis functions! We can apply basis function regression to multivariate systems, or we can separate variables using $f_1 = x$, and $f_2 = y$... we can really use anything we want, with just one small change. Let's take for example our three-basis function system but this time with *two variables*, x and y.

$$\hat{F}(x, y) = a_1 f_1(x, y) + a_2 f_2(x, y) + a_3 f_3(x, y)$$

The SSE would be very similar to before, but this time we have *two summations* (one for each variable):

$$SSE = \sum_{i=1}^{N_x} \sum_{k=1}^{N_y} \left(\hat{F}(x_i, y_k) - F(x_i, y_k) \right)^2 = \sum_{i=1}^{N_x} \sum_{k=1}^{N_y} \left(\sum_{m=1}^{N_c} a_m f_m(x_i, y_k) - F(x_i, y_k) \right)^2$$

After going through the same derivation as before and adding back in the summations we get:

$$\underbrace{\begin{bmatrix} \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_1 f_1 & \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_2 f_1 & \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_3 f_1 \\ \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_1 f_2 & \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_2 f_2 & \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_3 f_2 \\ \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_1 f_3 & \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_2 f_3 & \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_3 f_3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_1 F \\ \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_2 F \\ \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_3 F \end{bmatrix}}_{b}$$

Let's make a general rule for N_c number of basis functions and two variables:

Multivariate Regression with Basis Functions

$$\underbrace{\begin{bmatrix} \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_1 f_1 & \cdots & \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_{N_c} f_1 \\ \vdots & \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_m f_m & \vdots \\ \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_1 f_{N_c} & \cdots & \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_{N_c} f_{N_c} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_1 \\ \vdots \\ a_{N_c} \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_1 F \\ \vdots \\ \sum_{i=1}^{N_x}\sum_{k=1}^{N_y} f_{N_c} F \end{bmatrix}}_{b}$$

Where N_x and N_y are the number of x and y data points, and N_c is the index of basis functions. Together this creates an $N_c \times N_c$ matrix. Solving for $x = (a_1, \dots, a_{N_c})$ will return the coefficients of the basis functions.

Applying the above two variable, three basis function to an example:

Basis Functions

$f_1 = x^2$
$f_2 = y$
$f_3 = \sin(x + y)$

Data

x	y	z
0	0	5
2	1	10
2.5	2	9
1	3	0
4	6	3
7	2	80

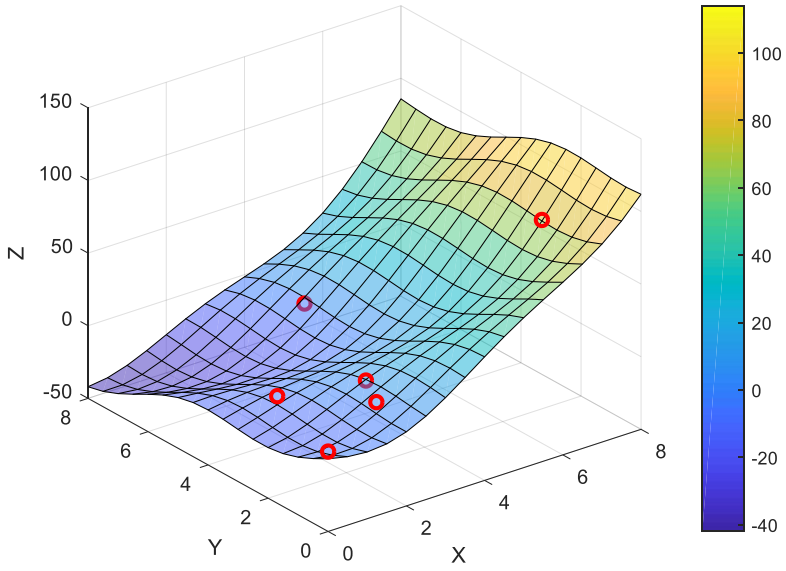


Figure 13: The resulting function, $\hat{F} = 1.86x^2 - 4.38y - 7.61\,sin(x + y)$ from our basis function regression.

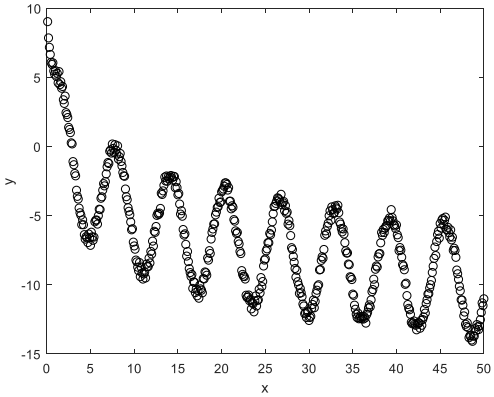
Linear Within the Parameters

By now, you may have realized that we have been using matrices to solve systems which include obviously nonlinear functions within them. But, how are we able to do this – matrices can only be used to solve linear systems of equations, right?

It works because all the functions/summations within the matrix are *known* and will result to be scalars. The unknowns are the parameters, $a_1, \ldots, a_{N_c}$, which are linear (no squared terms, etc.). Looking at it in this way, our system IS technically linear, because it is *linear within the parameters*!

Workshop: Basis Function Regression

Take a look at the data plotted below. Propose a few basis function combinations that you think will lead to a good curve fit to the data.



$\hat{F} = a_1 \sin(x) + a_2 x + a_3 \cdot 1 + a_4 \cdot e^{-x}$

Benefits and Drawbacks: Basis Function Regression

Benefits

- Easily allows non-linear functions into regression
  - We can now regress functions like $\sin(xy)$, 3^{x+2y} or any function we can think of to our data while still remaining linear within the parameters
- We can use basis functions to *make the system multivariate*
  - If we want $f_1 = x$ and $f_2 = y$, we can do that!
- Any prospective basis functions can be fine-tuned by comparing a bunch of options and their resulting curves to the data; from there, residuals, and other error tools can be used to select the most appropriate one
  - Ex: If you believe your basis function should be something in the form of $f_1 = \cos(\#xy + \#)$, you may choose to fit a few variations of that to your data, such as $f_1 = \cos(xy + 1)$ or $f_1 = \cos(2xy + 0.2)$ → Compare the resulting curves and determine the best option.

Drawbacks

- Choosing the correct basis function (there are many!) largely comes down to your intuition and knowledge of underlying systems. Visualizing the data can also help

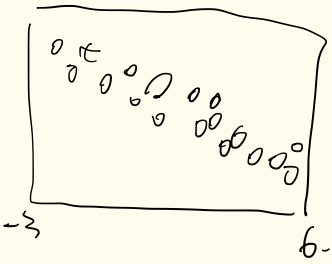
Conclusion and Summary

In this module we have covered:

- Curve fitting using least-squares regression to fit linear, quadratic and any order polynomial.
- Basis function regression when we want to fit data points that can be multivariate or that can be described by basis functions.

Next up: Interpolation

Matlab Date Set:



x y

A = [length(x) sum(x) sum(x.^2)
sum(x) sum(x.^2) sum(x.^3)
sum(x.^2) sum(x.^3) sum(x.^4)];

b = [sum(y); sum(x.*y); sum(x.^2.*y)];

C = A\b;

x_plot = -3:0.1:6;

y_plot = C(1) + C(2)*x_plot + C(3)*x_plot.^2;

hold on;

plot ( )

C: a number very small that make x^2 coefficient almost 0.

∴ the data plot should make more sense as linear.