

CHEMICAL ENGINEERING 2E04

Chapter 3 – Curve Fitting

Module 3B: Interpolation

Dr. Jake Nease

Kieran McKenzie

Steven Karolat

Cynthia Pham

Chemical Engineering

McMaster University



Updated November 8, 2019

Contents

Supplementary Material	2
Overview	3
Primary Learning Outcomes	3
Interpolating using Lagrange Polynomials	3
Standard Polynomials	3
Lagrange Polynomials.....	3
Benefits, Drawbacks, and Failures: Lagrange Polynomials	5
Spline Interpolation.....	6
Linear Splines	6
Quadratic Splines.....	7
Drawbacks of Quadratic Splines.....	8
Cubic Splines	9
Data Scaling and Normalization.....	11
Shameless Optimization (4G03) Plug	13
Conclusion and Summary.....	14

Supplementary Material

Suggested Readings

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

- Lagrange Interpolating Polynomials
 - Chapter 6 → 6.5.1
- Spline Interpolation
 - Chapter 6 → 6.6
- Using MATLAB Built-in Functions for Curve Fitting (Useful for Checking Your Work!)
 - Chapter 6 → 6.7

Online Material

[MathTheBeautiful > Polynomial Interpolation According to Lagrange](#) (Click to follow hyperlink)

- Proves why/how Lagrange Polynomials work to fit data from a Linear Algebra standpoint

[Computerphile > Bicubic Interpolation](#) (Click to follow hyperlink)

- For those who are interested!
- Compares linear and cubic splines simply, and speaks about the popular use of spline interpolation in imaging and animation applications
- *Note:* Refers to cubic spline interpolation as ‘bicubic interpolation’

[Management Information Systems > Intro to Designing Optimization Models Using Excel](#) (Click to follow hyperlink)

- For those who are interested!
- Uses Excel’s built-in optimization solver to solve a simple optimization problem: maximizing business revenue

Overview

Recall, interpolation is when we wish to *estimate values within the range of our data set*. For example, having data points for minutes 12, 13 and 14, but wishing to find the value at 13.5 minutes.

The Interpolation Process

In interpolation, a curve is uniquely fit through a set of data with the intent of estimating the value of a function between those points.

Interpolation can be done in several ways, such as (1) single polynomials to represent the entire range or (2) many connected polynomials tied together to cover the range.

Primary Learning Outcomes

- Introduce interpolation using *standard* and *Lagrange polynomials*.
- Apply MATLAB to form an algorithm for interpolation using Lagrange polynomials.
- Familiarize ourselves with spline interpolation (linear, quadratic and cubic).
- Identify when we have *data scaling* and require *normalization*.

Interpolating using Lagrange Polynomials

It's possible to use a single polynomial to represent the range. This method of interpolation is *most common when few points are involved*. You will see that fitting one polynomial to a set can have its drawbacks when too many data points are used, due to the introduction of noise in the graph.

Standard Polynomials

Since we are interpolating, we must fit every point exactly. This means we must use a polynomial of n^{th} order to fit $n + 1$ data points. As previously discussed, high order standard polynomials can exhibit erratic behaviour between data points. Lower order polynomials are better behaved. When we get beyond 6 data points, standard polynomials are usually a poor choice.

Lagrange Polynomials

Lagrange polynomials are one way to formulate a polynomial – no optimization required! *Only the data points themselves are needed*. Let's start with two points, (x_1, y_1) and (x_2, y_2) . The first order Lagrange polynomial through these points has the form:

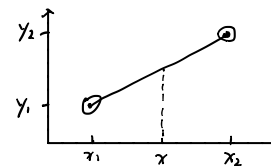
$$f(x) = y = a_1(x - x_2) + a_2(x - x_1)$$

In other words, we can find a *linear combination* of two coefficients a_1 and a_2 that will fit between these points. Substituting our two known points in for x , we achieve two equations. Isolating for a_1 and a_2 :

$$\begin{aligned} y_1 &= a_1(x_1 - x_2) + a_2(\underbrace{x_1 - x_1}_0) \rightarrow a_1 = \frac{y_1}{(x_1 - x_2)} \\ y_2 &= a_1(\underbrace{x_2 - x_2}_0) + a_2(x_2 - x_1) \rightarrow a_2 = \frac{y_2}{(x_2 - x_1)} \end{aligned}$$

Putting these back into our first order form gives the polynomial:

$$f(x) = \frac{(x - x_2)}{(x_1 - x_2)}y_1 + \frac{(x - x_1)}{(x_2 - x_1)}y_2$$

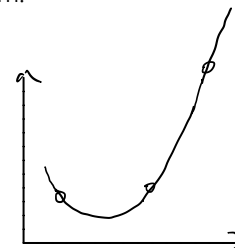


Remember that $\{x_1, x_2, y_1, y_2\}$ are all points we already know; hence they can be *treated as numbers*! Expanding Lagrange polynomials to second order, using three points, (x_1, y_1) , (x_2, y_2) and (x_3, y_3) has the form:

$$f(x) = y = a_1(x - x_2)(x - x_3) + a_2(x - x_1)(x - x_3) + a_3(x - x_1)(x - x_2)$$

Which turns into:

$$f(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)}y_1 + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)}y_2 + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}y_3$$



From this, you can probably notice the pattern emerging, and the mechanics behind Lagrange polynomials may become apparent. Imagine using x_1 as x - you can see that the first term's coefficient in front of y_1 becomes 1, and the other two terms disappear to 0!

Lagrange Polynomials

The general form for an n^{th} order Lagrange Polynomial using $n + 1$ data points is:

$$f(x) = \frac{(x - x_2)(x - x_3) \dots (x - x_{n+1})}{(x_1 - x_2)(x_1 - x_3) \dots (x_1 - x_{n+1})}y_1 + \frac{(x - x_1)(x - x_3) \dots (x - x_{n+1})}{(x_2 - x_1)(x_2 - x_3) \dots (x_2 - x_{n+1})}y_2 + \dots + \frac{(x - x_1)(x - x_2) \dots (x - x_n)}{(x_{n+1} - x_1)(x_{n+1} - x_2) \dots (x_{n+1} - x_n)}y_{n+1}$$

The i^{th} term does not contain $(x - x_i)$ in the numerator. The denominator is $(x_i - x_{j|j \neq i})$. The formula can also be written more compactly as a summation, where $N = n + 1$ is the number of data points:

$$f(x) = \sum_{i=1}^N y_i L_i(x) = \sum_{i=1}^N y_i \prod_{\substack{j=1 \\ j \neq i}}^N \frac{(x - x_j)}{(x_i - x_j)}$$

Data	
x	y
1	5
2	15
3	20
4	32
5	50

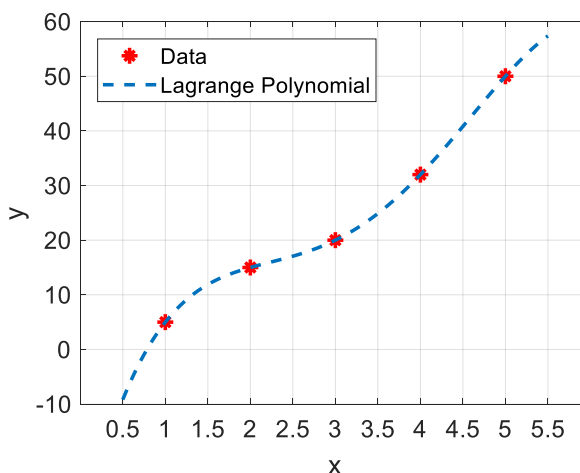


Figure 1. Using Lagrange polynomial to fit the data

5 points
4th poly Exactly.

Workshop: Lagrange Polynomials

Given a set of N data points, write a MATLAB code that determines the Lagrange polynomial for this set. Most of the code is given – your job is to interpret what you need to add to the code to make it work!

```
clc; clearvars; close all

%------%
% Inputting Known Data:
%------%

xvec = [0:10];          % Example xvec and corresponding yvec of collected data
yvec = sin(xvec);
x = 5.5;                % Value we want to evaluate the function at

N = length(xvec);

%------%
% Calculating the Lagrange Approximation of  $y = f(x)$ :
%------%



Summ = 0;                % Do not reset 'Summ' EVER!

for i = 1:N
    Num = 1; Den = 1;    % Must reset these to 1 each time!

    for j = 1:N
        if j ~= i
            Num = Num.* (x - xvec(j));
            Den = Den.* (xvec(i) - xvec(j));
        end
    end

    Summ = Summ +  $yvec(i) * \frac{Num}{Den}$ ;
end

y = Summ;
```



Benefits, Drawbacks, and Failures: Lagrange Polynomials

Benefits:

- Only the data points are required to construct a Lagrange polynomial.
- *Data points do not need to be evenly spread.* i.e. the spacing between points is not required to be constant. If x -values were (1, 2.25, 4.17, 9) a Lagrange polynomial could still be used to fit the data.

Disadvantages:

- They are *not easily "updateable"* – imagine you have three data points and use the above formulation to set up your Lagrange polynomial. The next day you get a 4th point by conducting your experiment again – *all the coefficients need to be calculated again* ☹.

Failures:

- The Lagrange polynomial is *for interpolation use only* - once outside the range of the data the polynomial may inaccurately represent the trend
 - Using the same data set as Figure 1, the associated Lagrange polynomial diverges far away from the true trend outside of the data range, as shown in Figure 2:

Data	
x	y
1	5
2	15
3	20
4	32
5	50

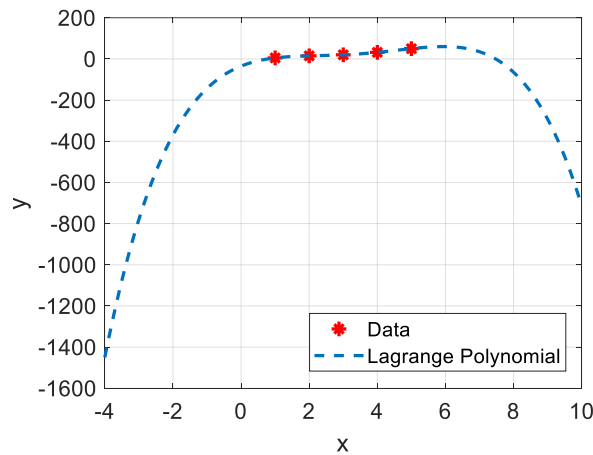


Figure 2: Using a Lagrange polynomial to fit the data, showing wild behaviour outside the data range!

Spline Interpolation

Instead of having one polynomial describe the entire set of data, we can use *splines* (AKA piecewise representation). This is where *the data range is broken into sections, with each section having their own polynomial*. The points where one polynomial switches over to the next are called "*knots*".

When using high order polynomials to fit many data points, we often see erratic behaviour from the turning points of the polynomial. We can instead use a series of lower order polynomials (1st, 2nd and 3rd) to cover the entire range. Constraints to ensure that each spline perfectly blends into the next will be set, resulting in a smooth curve.

Linear Splines

Linear Splines (AKA linear interpolation) is, in essence, "connecting one point to the next using a straight line". The easiest way to do so is to use a first order Lagrange polynomial to fit the points (avoids any calculations).

$$f(x) = \frac{(x - x_2)}{(x_1 - x_2)} y_1 + \frac{(x - x_1)}{(x_2 - x_1)} y_2$$

Generalization for Linear Splines

For n points, there are $n - 1$ intervals between points, therefore we have $n - 1$ linear Lagrange polynomials to represent the data's range. Because we need $n - 1$ functions, subscripts are used to distinguish intervals. Each interval in linear spline interpolation is denoted:

$$f_i(x) = \frac{(x - x_{i+1})}{(x_i - x_{i+1})} y_i + \frac{(x - x_i)}{(x_{i+1} - x_i)} y_{i+1}$$

The data set on the following page describes the relationship between time and reactor conversion. If we applied a linear spline to the data, we would find that for 10 minutes of reactor time, we should expect a conversion of 36.5%.

Data	
Time (minutes)	Reactor Conversion (%)
1	5
2	12
5	16
8	32
12	41
15	43
20	73
27	81

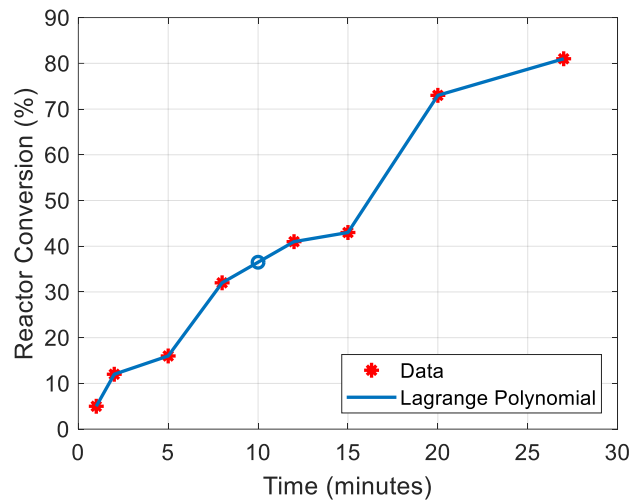


Figure 3: Using a Linear Spline to find the point at 10 minutes, from this conversion is 36.5%

Linear splines are by far the most common, we actually use them all the time in engineering and everyday life without thinking about it. Take for example, the last time you thought about the weather. If the temperature is 15°C at 9 am, and is forecasted to be 25°C at 1 pm, we typically assume it'll be near 20°C by 11 am.

Quadratic Splines

Quadratic Splines use a 2nd order polynomial known as, $s_{2,i}$, to tie together two data points in a series. Here, we keep the standard form of writing out a polynomial for an i th second-order $s_{2,i}$:

$$f(x) = s_{2,i} = a_i + b_i x + c_i x^2$$

As with linear splines, the same goes for quadratic – there are $n - 1$ equations required to describe the intervals between n points. Since each equation has three coefficients, a_i , b_i and c_i , there are $3(n - 1)$ *variables to solve for*. To reduce the system down to zero degrees of freedom (to make it solvable) we need to implement a series of conditions to help define our system:

Explanation and Equation(s)	New Equations	Remaining DOF
I know that the <i>first spline (1)</i> prediction must give me the exact value of my first point, and the <i>last spline (n-1)</i> prediction must give me the exact value of my last point: $s_{2,1}(x_1) = y_1 = a_1 + b_1 x_1 + c_1 x_1^2$ $s_{2,n-1}(x_n) = y_n = a_{n-1} + b_{n-1} x_n + c_{n-1} x_n^2$	2	$3n - 5$
At all the <i>interior points</i> (excluding 1 and n, so $n - 2$ total), the value of the spline to the <i>right</i> and the <i>left</i> of the point should both give me the exact value of that point: $s_{2,i}(x_i) = s_{2,i-1}(x_i)$ $a_i + b_i x_i + c_i x_i^2 = a_{i-1} + b_{i-1} x_i + c_{i-1} x_i^2$ $\forall i = 2 \dots (n - 1)$	$2(n - 2)$	$n - 1$
The <i>first derivative</i> of <i>spline i</i> and <i>spline i - 1</i> are <i>equal</i> at each of the interior points (maintains smoothness between splines): $s'_{2,i}(x_i) = s'_{2,i-1}(x_i)$ $b_i + 2c_i x_i = b_{i-1} + 2c_{i-1} x_i$ $\forall i = 2 \dots (n - 1)$	$n - 2$	1

The *second derivative* is 0 at the first point, which is a user decision since I desire the first interval to establish concavity on its own accord:

Condition 1 0

$$\begin{aligned} s''_{2,1}(x_1) &= 0 \\ 2c_1 &= 0 \end{aligned}$$

A plot using a quadratic spline to interpolate between the same data points as before can be seen below. Now, we would expect that for 10 minutes of reactor time we should yield a conversion of 39.0%.

Data

Time (minutes)	Reactor Conversion (%)
1	5
2	12
5	16
8	32
12	41
15	43
20	73
27	81

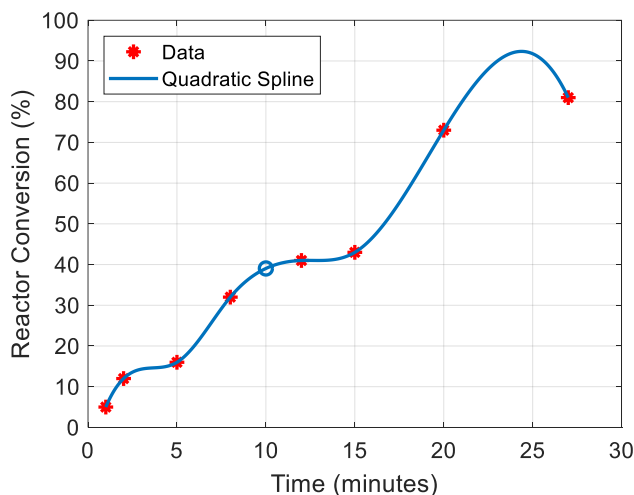


Figure 4: Using a Quadratic Spline to find the point at 10 minutes, conversion is 39.0%

Drawbacks of Quadratic Splines

- *Quadratic splines cannot change concavity* within an interval.
 - This is due to the nature of quadratic equations - they're either concave up or down.
- The third condition (slopes between neighboring splines must be equal across the knot), can restrict quadratic splines and may force them to act erratically to meet their conditions, which reduces their usefulness for interpolation.
 - Take a look at the final segment between $x = 20$ and $x = 27$. The incoming slope was a large positive value, so the segment was forced to do the same... but this slope was too large to easily hit the final point at $x = 27$. As a result, that section needed to be sharply concave down to correct itself - this could lead to erroneous values given by that spline.

Note: Quadratic splines are NOT the same as plotting a bunch of second order Lagrange polynomials over the data set!

If you were to do this for a set of data points very similar to the example above, you'd get something like this:

Note that the conditions of quadratic splines are NOT met.

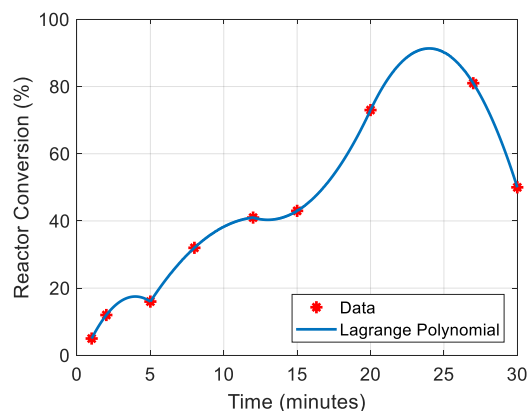


Figure 5: Plotting a bunch of second order Lagrange polynomials over a set of data. This is NOT the same as quadratic splines.

Cubic Splines

The standard form of a cubic spline s_3 that connects two points in a series of n points is:

$$f(x) = s_3 = a_i + b_i x + c_i x^2 + d_i x^3$$

This time each spline $s_{3,i}$ has four coefficients (a_i, b_i, c_i and d_i). With the same number of intervals, $n - 1$, that means there are a total of $4(n - 1)$ coefficients to solve for. Where am I going to get that many unique conditions to remove all my DOFs? Luckily, I can define the behaviour of the splines to get the information I need as follows (hopefully you find the colour coding helpful):

Explanation and Equation(s)	New Conditions	Remaining DOF
I know that the <i>first spline (1)</i> prediction must give me the exact value of my first point, and the <i>last spline (n-1)</i> prediction must give me the exact value of my last point: $s_{3,1}(x_1) = y_1 = a_1 + b_1 x_1 + c_1 x_1^2 + d_1 x_1^3$ $s_{3,n-1}(x_n) = y_n = a_{n-1} + b_{n-1} x_n + c_{n-1} x_n^2 + d_{n-1} x_n^3$	2	$4n - 6$
At all the <i>interior points</i> (total again, $n - 2$), the value of the spline to the <i>right</i> and the <i>left</i> of the point should both give me the exact value of that point: $a_i + b_i x_i + c_i x_i^2 + d_i x_i^3 = \overbrace{a_{i-1} + b_{i-1} x_i + c_{i-1} x_i^2 + d_{i-1} x_i^3}^{s_{3,i-1}(x_i)} = \overbrace{a_i + b_i x_i + c_i x_i^2 + d_i x_i^3}^{s_{3,i}(x_i)}$ $\forall i = 2 \dots (n - 1)$	$2(n - 2)$	$2n - 2$
The <i>first derivative</i> of <i>spline i</i> and <i>spline i - 1</i> are <i>equal</i> at each of the interior points (maintains smoothness between splines): $b_i + 2c_i x_i + 3d_i x_i^2 = s'_{3,i-1}(x_i) = s'_{3,i}(x_i) = b_{i-1} + 2c_{i-1} x_i + 3d_{i-1} x_i^2$ $\forall i = 2 \dots (n - 1)$	$n - 2$	n
The <i>second derivative</i> of <i>spline i</i> and <i>spline i - 1</i> are <i>equal</i> at each of the interior points (maintains concavity between splines): $2c_i + 6d_i x_i = s''_{3,i-1}(x_i) = s''_{3,i}(x_i) = 2c_{i-1} + 6d_{i-1} x_i$ $\forall i = 2 \dots (n - 1)$	$n - 2$	2
We CHOOSE the second derivatives at the <i>first</i> and <i>last</i> point to be equal to 0 since, again, we do not want to constrain the concavities of the first and last splines: $s''_{3,1}(x_1) = 0 = 2c_1 + 6d_1 x_1$ $s''_{3,n-1}(x_n) = 0 = 2c_{n-1} + 6d_{n-1} x_n$	2	0

The good news is that we don't have to go through this whole procedure every time – once we figure it out we can use it in general! If we set up all these equations as a system of linear equations ($Ax = b$ AGAIN!?), we will get something that looks like the following figure.

- The first row of the matrix corresponds to the second derivative of the first spline being zero at the first point, and the last row corresponds to the same for the last spline at the last point.
- All other rows follow a very distinct pattern. See if you can piece it out. Remember that we HAVE all the x_i and y_i values, and we are looking for the values for a, b, c and d .

$$\begin{bmatrix}
 0 & 0 & 2 & 6x_1 \\
 1 & x_1 & x_1^2 & x_1^3 \\
 1 & x_2 & x_2^2 & x_2^3 \\
 0 & 1 & 2x_2 & 3x_2^2 & 0 & -1 & -2x_2 & -3x_2^2 \\
 0 & 0 & 2 & 6x_2 & 0 & 0 & -2 & -6x_2 \\
 & & & & 1 & x_2 & x_2^2 & x_2^3 \\
 & & & & 1 & x_3 & x_3^2 & x_3^3 \\
 & & & & & & & \ddots \\
 & & & & 0 & 1 & 2x_{n-1} & 3x_{n-1}^2 & 0 & -1 & -2x_{n-1} & -3x_{n-1}^2 \\
 & & & & 0 & 0 & 2 & 6x_{n-1} & 0 & 0 & -2 & -6x_{n-1} \\
 & & & & & & & & 1 & x_{n-1} & x_{n-1}^2 & x_{n-1}^3 \\
 & & & & & & & & 1 & x_n & x_n^2 & x_n^3 \\
 & & & & & & & & 0 & 0 & 2 & 6x_n
 \end{bmatrix}
 \begin{bmatrix}
 a_1 \\
 b_1 \\
 c_1 \\
 d_1 \\
 a_2 \\
 b_2 \\
 c_2 \\
 d_2 \\
 \vdots \\
 a_{n-1} \\
 b_{n-1} \\
 c_{n-1} \\
 d_{n-1}
 \end{bmatrix}
 =
 \begin{bmatrix}
 0 \\
 y_1 \\
 y_2 \\
 0 \\
 0 \\
 y_2 \\
 y_3 \\
 0 \\
 \vdots \\
 0 \\
 y_{n-1} \\
 y_n \\
 0
 \end{bmatrix}$$

Continuing with the previous data points, a cubic spline has been fitted. Now, when $x = 10, y = 37.6$. Note that the curvature of the last spline is much less aggressive due to the concavity change in the second-last spline (compared to the quadratic spline).

Data	
Time (minutes)	Reactor Conversion (%)
1	5
2	12
5	16
8	32
12	41
15	43
20	73
27	81

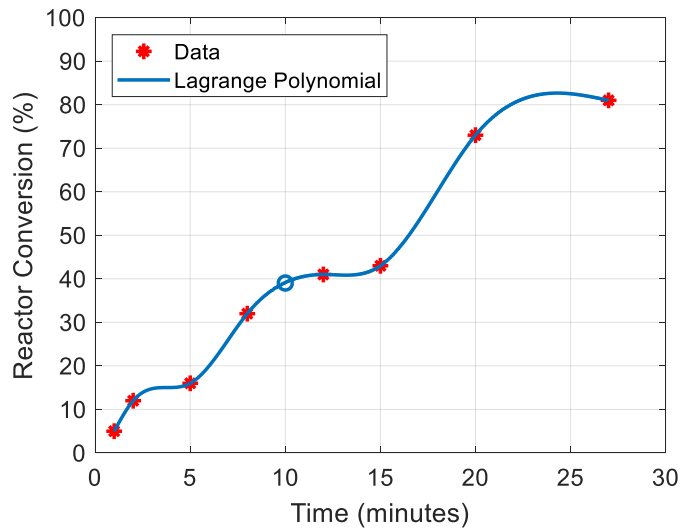


Figure 6. Using a Cubic Spline to find the point at 10 minutes, conversion is 37.6%

Cubic splines typically give better fits than quadratic splines. This is because cubic equations can inflect, whereas quadratic ones cannot. This gives them extra "maneuverability" allowing them to better fit the points and the curve.

It's also possible to use other polynomial forms with spline interpolation, for example, the premise of splines to "cut-up" a data set can also be used with 2nd and 3rd order Lagrange polynomials. The derivation for that can get a little hairy, just know it's possible.

Data Scaling and Normalization

Depending on the units used and their relative magnitude to each other, *issues can arise from data scaling*. Fix this by normalizing your inputs before using the trend or spline lines. This isn't a very typical problem to run into, but when it does happen, normalizing can make a world of difference.

For example, your trend line describes the effect of varying concentration of the reactant in the infeed, and the temperature of the reactor to conversion. However, the units for concentration are $\frac{\text{mol}}{\text{L}}$, and temperatures vary between 300 to 500 K. You may have a data set like this:

Reactor Conversion (%)	10	20	40	60	70	80
Inlet Concentration of reactant ($\frac{\text{mol}}{\text{L}}$)	0.001	0.001005	0.0012	0.0012	0.0013	0.0013
Inlet Flow Temperature (K)	300	300	305	310	310	315

Basis function regression was used for this example, the basis functions used are:

$$f_1(x, y) = x^2 \quad f_2(x, y) = y \quad f_3(x, y) = 1$$

Using the data as-is, the following plot and basis function coefficients are found:

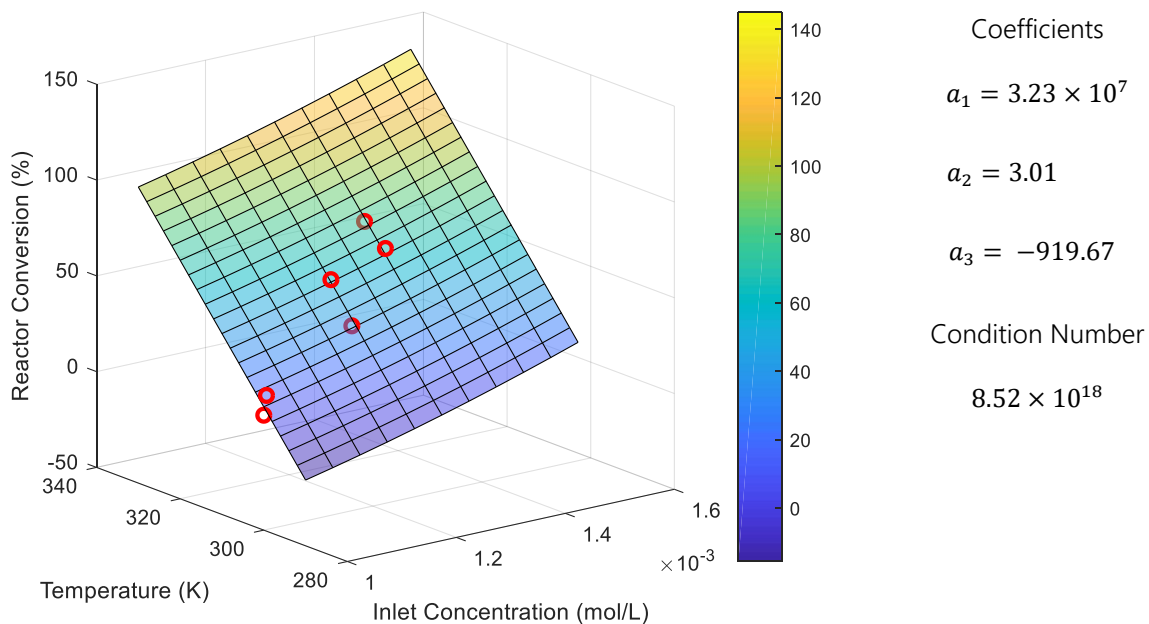


Figure 7. The plane represents the regressed function, and the red dots are the data points

MATLAB's Warning:

Warning: Matrix is close to singular or badly scaled. Results may be inaccurate.

At first glance this looks like a reasonable fit, but the coefficients say otherwise. The huge difference between coefficients is a bad sign. This model essentially only relies on f_1 and its x^2 relationship - nothing else has much 'pull'. MATLAB realizes this and warns you!

The *condition number* of the matrix also warns us. Remember back to Chapter 1: you want a condition number to be near 1 to be nonsingular – in this case we're at 8.52×10^{18} ! Our regression may lead to inaccurate readings, and we cannot trust it. But, thankfully, we have a fix!

Normalize the data, *and then* regress! To normalize the data, use the following equation for all inputs (x, y):

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Note: This is one of the many ways to normalize data, other methods alter the spread of the data to varying degrees.

This will normalize all values to a value within [0,1], so our input data is now:

Reactor Conversion (%)	10	20	40	60	70	80
Normalized Inlet Concentration of reactant ($\frac{mol}{L}$)	0	0.0167	0.667	0.677	1	1
Normalized Temperature (K)	0	0	0.333	0.667	0.667	1

Now, after normalization we get:

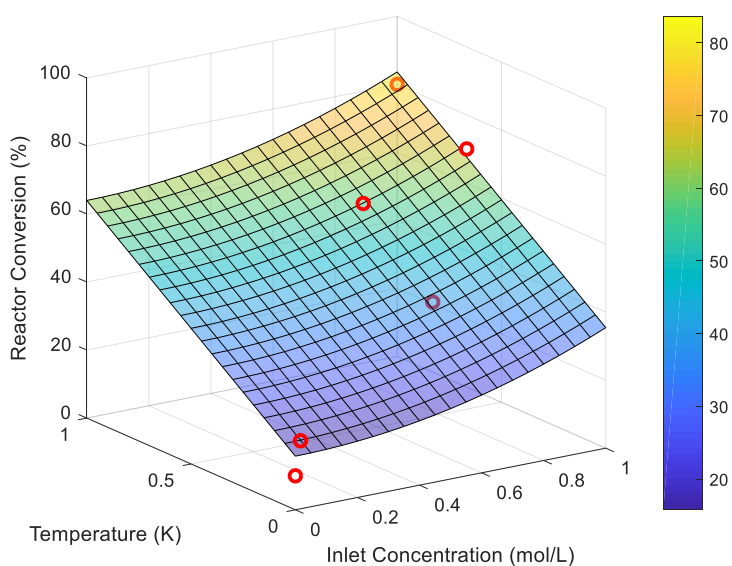


Figure 8. The plane represents the regressed function, and the red dots are the data points

Coefficients

$$a_1 = 19.66$$

$$a_2 = 48.22$$

$$a_3 = 15.77$$

Condition Number

119

Note: This isn't an ideal condition number, but it is significantly improved.

Now, only 2 digits of accuracy may be lost in our solutions.

Much better! The weighting between basis functions is much more even, *preventing any one function or one variable from dominating the rest.*

In order to use our normalized function with a new point, (*Inlet Concentration* = $0.0015 \frac{mol}{L}$, *Temperature* = 300 K) we must *first normalize* it to (1.667, 0), then use it in our formula. *This process adds an extra step, but in many cases it is well worth it for a more accurate result.* In this case, our normalized function results in a reactor conversion of 70.3%. If we did the exact same thing without normalization, we would find a reactor conversion of 54.8% - significantly different!

Data Scaling and Normalization Are NOT the Same as Converting Units



The purpose of normalization is to make the values relative to each other by giving them a value within $[0, 1]$. Changing the input's units doesn't do this (changing our units from $\frac{\text{mol}}{\text{L}}$ to $\frac{\text{mol}}{\text{ml}}$ will still lead to a singular matrix). Instead, the singular matrix problem has everything to do with relative closeness/spread of the data.

You can think of normalizing as increasing the resolution of 'the picture'. This helps distinguish the effects of the variables from one data point to the next, which in turn helps us withdraw the impact of changing specific variables to find our coefficients.

Shameless Optimization (4G03) Plug

The way we've done everything so far has involved setting the derivative to zero (giving us a stationary point) and then solving, but there are other ways of doing it – we could use optimization!

The main difference between optimization and our "normal" way is how the system is defined. For our current methods to work we need zero DOF (n equations for n unknowns), we need everything equaling something. Think back to the first module's production allocation problem, we "NEEDED to produce X barrels in plant Y " to solve, but with optimization, we can rewrite that to be "we need AT LEAST or AT MOST X barrels produced at plant Y ". We can also use optimization to minimize the cost of production, instead of meeting production.

There are no "real" solutions to optimization problems – they are always under-defined and can have infinite answers that make the math work. Rather, optimization is about finding the *best* or *worst* combination by either maximizing or minimizing a certain parameter.

These traits of optimization have vast applications:

- Minimizing cost
- Minimizing travel distance (what google maps does for you)
- Deciding how to best distribute resources,
- Scheduling employees to have similar hours

The list goes on-and-on, and it even includes regression! The entire concept of optimizing is actually a chemical engineering idea that spread to other fields (major flex).

We can write out a regression problem as an optimization one in the following way which minimizes SSE :

$$\begin{aligned} \min_{a_2, a_1, a_0} \phi &= \sum_{i=1}^N (a_2 x_i^2 + a_1 x_i + a_0 - y_i)^2 \\ \text{Subject to} \quad a_n &\leq 1000 \\ a_n &\geq -1000 \end{aligned}$$

These are constraints restricting/limiting the possible values of a_n . Unless a specific scenario calls for them, regression doesn't need them – there just here to give you an example of how constraints are handled

Or we can get more specific and use basis functions - we can use variables within the basis functions themselves and let optimization choose the best values for them!

$$\min_{a_1, a_2, a_3, j, k, r} \phi = \sum_{i=1}^N (a_1 x^j + a_2 \sin(x + ky) + a_3 e^{rx} - y_i)^2$$

Now here comes the true power of optimization – solving equations that are nonlinear in the parameters. Let's take Antoine's Equation which describes the saturated vapour pressure of a pure component as a function of temperature:

$$\log(P_{sat}(T)) = A - \frac{B}{T + C}$$

Where, P_{sat} is saturated vapour pressure, T is temperature, A , B , and C are chemical dependant coefficients. We can't make this equation linear, so all of our previous methods fall apart. The beauty of optimization is it doesn't care! This goes to show the robustness of optimization methods. Obviously we didn't cover any actual math on how to solve problems like these in this section, just the general principle. If you want to learn more about this interesting field of math take 4G03!

Conclusion and Summary

In this module, we have covered interpolation using polynomials and spline interpolation.

- Interpolation is useful when the unknown point is within the range of the data.
- The primary benefit of interpolation (fitting a curve to each data point exactly) is that it captures the error inherent in the data.
 - This helps better consider influences/factors outside of the "pure" model which better represent the true result within the range.

We've made it to the end of curve fitting! Relax, and watch a video about eigenvalues and electron volts (0.23 eV, to be exact): <https://www.youtube.com/watch?v=nL8hVXSDmNM>

Next up: Numerical Differentiation and Integration