Higher Order Interpolation Part 2

Leigh J. Little SUNY Brockport October 11, 2021

Contents

1 Introduction **1**

2 Quadratic Case **1**
2.1 Example 1: Using Two Points Below $x^* = 1.63$ and One Above 2
2.2 Example 2: Using One Point Below $x^* = 1.63$ and the Two Above 2
2.3 Some Other Considerations 2
2.4 The Cubic Interpolating Polynomial 3

3 Another Approach...Use the Whole Table **3**

1 Introduction

We have seen that interpolation is not limited to simple linear interpolation. We can also compute quadratic and cubic (and even higher order) functions that will will go through the points in a table. How do we use this new information to obtain more accurate estimates of missing entries in a table of values?

2 Quadratic Case

Consider the table from the linear interpolation discussion (note that one additional point has been added to the table) show in Table 1. As before, we would like to estimate the value of y^* when $x^* = 1.63$. This

x	$y = f(x)$
⋮	⋮
1.50	2.33
1.60	2.58
1.70	2.82
1.80	3.06
⋮	⋮

Table 1: Table of values for some $y = f(x)$.

value lies between 1.60 and 1.70 in the table (see Table 2).

x	$y = f(x)$
⋮	⋮
1.50	2.33
1.60	2.58
1.63	???
1.70	2.82
1.80	3.06
⋮	⋮

Table 2: Location of $x^* = 1.63$ in the table.

From what we learned about linear interpolation, we want to use the two points on either side (1.60, 2.58) and (1.70, 2.82), however to fit a quadratic, we also need one other point. We have two choices; we can use either (1.50, 2.42) or (1.80, 3.06). Which one do we use? It turns out that without further criteria or information, it won't matter too much which of these points we use. Let's test both points.

2.1 Example 1: Using Two Points Below $x^* = 1.63$ and One Above

For this first example, we will use the two points below $x^* = 1.63$ and the one above. From this point, the algorithm is a combination of the one from linear interpolation and the process of fitting a quadratic.

a) Locate where x^* lies in the table.

b) Choose the two points below x^* and the one above.

c) Use these three points to generate the Vandermonde matrix.

d) Using the corresponding y coordinates, solve the linear system for the quadratic coefficients.

e) Substitute the value of x^* into this quadratic to compute the estimate y^*.

The code to do this is in the script `quad1.m`. This example uses the same data file as from Homework 15, but in order to assess the accuracy of the process, we need to use a table with more precision in the y values. This table has 7 digits of accuracy instead of 4.

From the output in `quad1.m` we see that the relative error is about $4.46 \cdot 10^{-5}$. When linear interpolation was used the relative error was $9.93 \cdot 10^{-4}$. Thus, quadratic interpolation reduced the error by a factor of about 22.

2.2 Example 2: Using One Point Below $x^* = 1.63$ and the Two Above

For this second example, we will use the point below $x^* = 1.63$ and the two points above. The algorithm in this case is nearly the same as for Example 1, except for how the points are extracted from the x and y vectors to compute the quadratic. This example is in the script `quad2.m`

From the output in `quad2.m` we see that the relative error is about $6.27 \cdot 10^{-5}$. Here, the quadratic interpolation reduced the error by a factor of about 16 compared to the linear interpolation error.

2.3 Some Other Considerations

Here are some observations we can make about these examples and the quadratic interpolating process in general:

a) Quadratic interpolation improved the error in the estimated y^* value relative to linear interpolation.

b) Using the two points below and the one above x^* provided a more accurate result than one point below and two above. This is a consequence of this particular table of values. It is entirely possible that the opposite would be observed if a different table of values were used.

c) Without additional information, either approach is acceptable.

d) There are two special cases that need to be considered for our quadratic interpolation process to be completely general. In the event that the desired value x^* lies between the first two points in the table, the only choice is to use one point below and two points above.

   Similarly, if the value of x^* lies between the last two points in the table, then the only choice is to use the point above and two points below.

e) In our examples, the x-coordinates in the table have been equally spaced. This is not a requirement; our process will still work for non-equally spaced x-coordinates.

f) As with linear interpolation, it is feasible to perform a formal mathematical analysis of the process to obtain a bound on the maximum error possible using quadratic interpolation. Let the 3 x-coordinates used to construct the quadratic interpolating polynomial be $x_1 < x_2 < x_3$. Then the absolute error in the quadratic interpolating process is bounded by

$$|y^* - y_{exact}| \leq \frac{(x_2 - x_1)(x_3 - x_2)(x_3 - x_1)}{6} \max_{x_1 \leq x \leq x_3} |f'''(x)|.$$

   In the event that the x-coordinates are equally spaced, this becomes

$$|y^* - y_{exact}| \leq \frac{h^3}{6} \max_{x_1 \leq x \leq x_3} |f'''(x)|$$

   where $h = x_2 - x_1 = x_3 - x_2$. If you have taken Calc 2, you should recognize the right hand side of this expression as the remainder term in the quadratic Taylor series expansion of $f(x)$ about x^*.

2.4 The Cubic Interpolating Polynomial

If quadratic interpolation is not sufficiently accurate, the process can be extended to the cubic interpolating polynomial in a straightforward manner. Here are some comments on the cubic case.

a) The code to do this is nearly identical to the one for the quadratic case because we wrote the code in an abstract manner that takes advantage of powerful MATLAB commands.

b) A cubic interpolating polynomial requires 4 points. The most logical points to use would be the two points in the table that lie below x^* and the two points that lie above.

c) As with the quadratic case, you would need to adjust the points used in the event that x^* lies between the first two points in the table or the last two points in the table.

   If x^* lies between the first two points in the table, you must use the point below x^* and the three points above. If x^* lies between the last two points in the table, you must use the three points below and the point above.

d) The absolute error bound for the cubic interpolating polynomial is given by

$$|y^* - y_{exact}| \leq \frac{h^4}{24} \max_{x_1 \leq x \leq x_4} |f''''(x)|,$$

   assuming the x-coordinates are equally spaced.

e) If a cubic is still not accurate enough, the process can be extended to 4th, 5th or even higher degree interpolating polynomials.

3 Another Approach...Use the Whole Table

Rather than use small portions of the table to approximate small degree polynomial approximations to obtain missing table entries, another approach would be to use all the values in the table and build one polynomial that approximates the entire table. This would eliminate the need to search for where x^* lies in the table. We could build the whole-table polynomial and then use the `polyval` function to estimate y^* directly.

This can be done and the process is nearly identical to what we have done with small degree polynomials. This is shown in the `wholetable.m` script. The output from this script is given below.

```
>> wholetable
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND =  2.241142e-19.
> In whole1 (line 11)

xstar = 1.33
ae =
   2.1243e-07
re =
   5.6183e-08
```

The graph of the resulting 20th degree interpolating polynomial is given in Figure 1.

In this case, we obtained an extremely accurate relative error. This indicates that the estimated $y*$ value is accurate to the number of places in the raw data (plus or minus some digits in the last place).

So why don't we do this all the time? This process is easier to use and is more accurate. The answer lies in the warning message that was also output when the `wholetable.m` script was run

```
>> wholetable
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND =  2.241142e-19.
```

Recall the end of the linear systems notes. A comment was made that one consequence of rounding errors in scientific computing is that a linear system with two intersecting lines can behave like a linear system with parallel lines (though in this case we are talking about a system with 21 intersecting 21st dimensional hyperplanes).

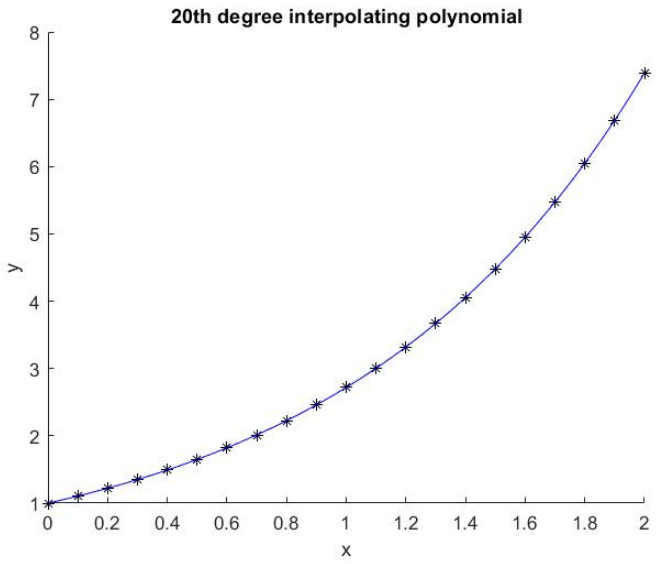

Figure 1: Building a 20th degree interpolating polynomial using the whole table of values. Like the other interpolating polynomials we have constructed, this goes through all 21 data points in the table.

This is what this warning means. Rounding errors are causing this linear system (which would have exactly 1 intersection point in 21 dimensional space if the problem were to be solved exactly by hand) is behaving like there are either no intersection points or an infinite number of intersection points.

If you ever solve a linear system in MATLAB and get this warning, it means that something very bad could be happening (*i.e.*, your computational results could be meaningless). In this example we were lucky, but this is very rarely the case if you see this error in MATLAB.

The important point you should be taking from this section is that extremely high degree polynomial interpolation (degrees larger than 7 or so) is usually a bad idea.