Higher Order Interpolation

Leigh J. Little SUNY Brockport October 8, 2021

Contents

1 Introduction

We have seen that linear interpolation can be used to approximate missing entries from a table of values. Linear interpolation is easy to implement by hand, however it is not always as accurate as we might need. Is it possible to use other basic equations to approximate missing data values in a table?

2 Quadratic Interpolation

The next step in the process would be to ask: instead of using a line to approximate a short section of a table, can a quadratic be used instead? If so how is this quadratic computed?

An important theorem in numerical analysis says that if you have 3 points, $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ and none of the x-coordinates are the same, then there is exactly one quadratic function that goes through all three points. This quadratic is called the *quadratic interpolating polynomial*.

It is easy to obtain the equation of this quadratic using MATLAB. A quadratic function has the form

$$y(x) = ax^2 + bx + c.$$

We can state the problem as: Given three coordinates in the x-y plane with unique x-coordinates, find the values of a, b and c in the quadratic function

$$y(x) = ax^2 + bx + c \tag{1}$$

that goes through these three points.

To obtain values of a, b and c, we use the known points. It is easier to see how the process works using a specific case. Consider the points (-1, 1), (1, 3), (2,0). Substitute the first point (-1,1) into Equation (1). This gives

$$\begin{aligned} 1 &= a(-1)^2 + b(-1) + c \\ 1 &= a - b + c \end{aligned}$$

Similarly, substituting the second point (1,3) into Equation (1) gives

$$\begin{aligned} 3 &= a(1)^2 + b(1) + c \\ 3 &= a + b + c \end{aligned}$$

and substituting the third point (2,0) into Equation (1) gives

$$\begin{aligned} 0 &= a(2)^2 + b(2) + c \\ 0 &= 4a + 2b + c \end{aligned}$$

Note that we now have a 3×3 system of linear equations

$$\begin{aligned} a - b + c &= 1 \\ a + b + c &= 3 \\ 4a + 2b + c &= 0. \end{aligned}$$

This can be solved in MATLAB using

```
>> format rat
>> A = [1 -1 1; 1 1 1; 4 2 1];
>> b = [1 3 0]';
>> sol = A\b
sol =
      -4/3     % first element is a
       1       % second element is b
       10/3    % third element is c
```

Thus the values defining the quadratic are $a = -\frac{4}{3}, b = 1, c = \frac{10}{3}$. This can be verified by plotting the quadratic function and the data points:

```
xg = linspace(-2,3,21)';
a = -4/3;
b = 1;
c = 10/3;
yg = a*xg.^2 + b*xg + c;
hold on
plot(xg,yg,'b-')
plot(-1,1,'k*')
plot(1,3,'k*')
plot(2,0,'k*')
hold off
xlabel('x')
ylabel('y')
title('Quadratic Interpolation')
```

The resulting figure is shown in Figure 1.

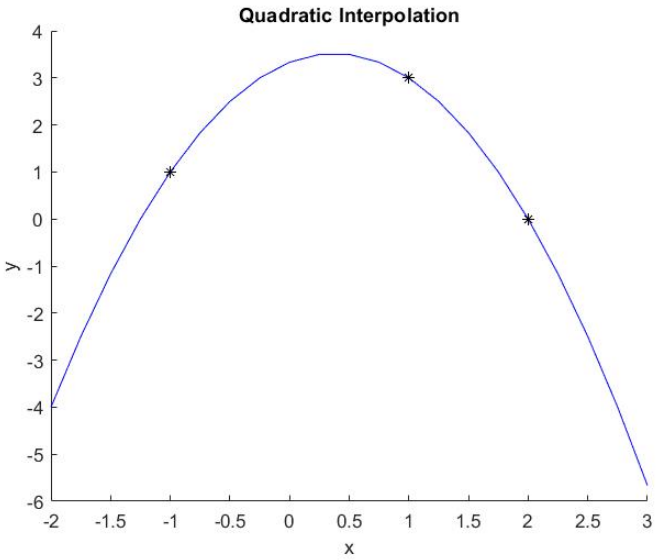

Figure 1: Example of quadratic interpolation. Note how the curve goes through all the points.

3 Cubic Interpolation

The quadratic interpolation process in the previous section can be extended to higher degree polynomials. If you have 4 points in the x-y plane and the x-coordinates are unique, there is exactly one cubic interpolating polynomial that goes through them.

Suppose you have the points (-1,0), (0,2), (1,1), (2,3). The general form of a cubic function is

$$y(x) = ax^3 + bx^2 + cx + d. \tag{2}$$

The process for this problem is the same as before, except that we will ultimately get a 4×4 system of linear equations for the values a, b, c and d.

Substituting the first point (-1,0) into Equation (2) gives

$$\begin{aligned} 0 &= a(-1)^3 + b(-1)^2 + c(-1) + d \\ 0 &= -a + b - c + d. \end{aligned}$$

Similarly, substituting the second point (0,2) into Equation (2) gives

$$\begin{aligned} 2 &= a(0)^3 + b(0)^2 + c(0) + d \\ 2 &= d. \end{aligned}$$

Substituting the third point (1,1) into Equation (2) gives

$$\begin{aligned} 1 &= a(1)^3 + b(1)^2 + c(1) + d \\ 1 &= a + b + c + d. \end{aligned}$$

Finally, substituting the fourth point (2,3) into Equation (2) gives

$$\begin{aligned} 3 &= a(2)^3 + b(2)^2 + c(2) + d \\ 3 &= 8a + 4b + 2c + d. \end{aligned}$$

The resulting 4×4 linear system becomes

$$\begin{aligned} -a + b - c + d &= 0 \\ d &= 2 \\ a + b + c + d &= 1 \\ 8a + 4b + 2c + d &= 3 \end{aligned}$$

This can be solved in MATLAB using

```
>> format rat
>> A = [-1 1 -1 1; 0 0 0 1; 1 1 1 1; 8 4 2 1];
>> b = [0 2 1 3]';
>> sol = A\b
sol =
       1       % first element is a
      -3/2     % second element is b
      -1/2     % third element is c
       2       % fourth element is d
```

The resulting graph is shown in Figure 2.

4 We Need a Better Way

The procedure for obtaining interpolating polynomials demonstrated above works, but the generation of the coefficient matrix required several manual steps. It would be better to let MATLAB do as much of the work as possible. The key to doing this is recognizing that there are patterns in the calculation that can be exploited.

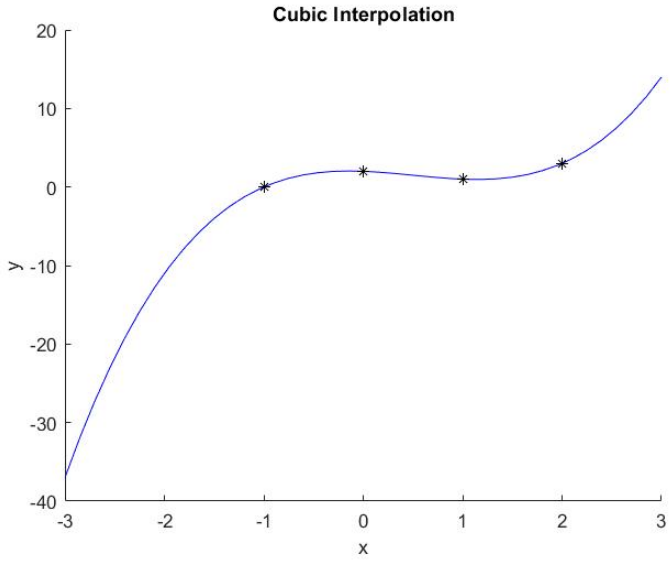

Figure 2: Example of cubic interpolation. Note how the curve goes through all the points.

4.1 Quadratic Example

Return to the quadratic function example. The x-coordinates were $x_1 = -1, x_2 = 1, x_3 = 2$ and the coefficient matrix was

$$A = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ 4 & 2 & 1 \end{pmatrix}.$$

However, writing the coefficient matrix in a different way is helpful in observing a pattern.

$$A = \begin{pmatrix} (-1)^2 & (-1)^1 & (-1)^0 \\ (1)^2 & (1)^1 & (1)^0 \\ (2)^2 & (2)^1 & (2)^0 \end{pmatrix}$$

provided we make the assumption that any number to the 0 power is 1 (this is not technically true for negative numbers or zero, but we can get away with it here).

4.2 Cubic Example

Similarly for the cubic example, the x-coordinates were $x_1 = -1, x_2 = 0, x_3 = 1, x_4 = 2$ and the coefficient matrix was

$$A = \begin{pmatrix} -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 \end{pmatrix}.$$

This can be written as

$$A = \begin{pmatrix} (-1)^3 & (-1)^2 & (-1)^1 & (-1)^0 \\ (0)^3 & (0)^2 & (0)^1 & (0)^0 \\ (1)^3 & (1)^2 & (1)^1 & (1)^0 \\ (2)^3 & (2)^2 & (2)^1 & (2)^0 \end{pmatrix}.$$

4.3 The Vandermonde Matrix

In these examples, it can be seen that each row of the coefficient matrix consists of descending powers of one of the x-coordinates. The highest power is equal to one less than the number of x-coordinates (a power of 2 with 3 coordinates and a power of 3 with 4 coordinates).

It is easier to generate the matrix if we examine it by columns instead of by rows. If we store the quadratic x-coordinates in the vector

$$x = \begin{pmatrix} -1 \\ 1 \\ 2 \end{pmatrix},$$

then the coefficient matrix has the form

```
>> x = [-1 1 2]';
>> A = [x.^2 x ones(3,1)]
A =
     1    -1     1
     1     1     1
     4     2     1
```

where `ones(3,1)` is a vector of all ones of length 3.

Similarly, if we store the cubic x-coordinates in the vector

$$x = \begin{pmatrix} -1 \\ 0 \\ 1 \\ 2 \end{pmatrix},$$

then the coefficient matrix has the form

```
>> x = [-1 0 1 2]';
>> A = [x.^3 x.^2 x ones(4,1)]
A =
    -1     1    -1     1
     0     0     0     1
     1     1     1     1
     8     4     2     1
```

We can quickly compute the quadratic case by doing

```
>> x = [-1 1 2]';
>> y = [1 3 0]';
>> A = [x.^2 x ones(3,1)];
>> p = A\y
p =
    -4/3
     1
     10/3
```

The cubic case would be

```
>> x = [-1 0 1 2]';
>> y = [0 2 1 3]';
>> A = [x.^3 x.^2 x ones(4,1)];
>> p = A\y
p =
     1
    -3/2
    -1/2
     2
```

This matrix A shows up quite frequently in computing. It is called the *Vandermonde* matrix. The formal definition is: Given a column vector $x = (x_1, x_2, \ldots, x_n)^T$, the elements of the Vandermonde matrix are given by

$$A_{i,j} = x_i^{n-j},\ i = 1, \ldots, n,\ j = 1, \ldots, n.$$

You can use the MATLAB function `vander` to generate it for a given x vector, for example

```
>> x = [-1 1 2]';
>> A = vander(x)
A =
     1    -1     1
     1     1     1
     4     2     1
```

4.4 The `polyval` Function

It would be really nice if there was also some MATLAB function that would take the output vector p of polynomial coefficients from the previous examples and use it to evaluate the quadratic/cubic at a given point (or even a set of points). Fortunately, there is. This is called the `polyval` function. The calling syntax for this function is

```
y = polyval(p,x)
```

The inputs to this function are a vector p of polynomial coefficients (in descending order of powers) and a scalar or vector x of coordinates where the polynomial is to be evaluated.

4.5 Compact Version of the Quadratic Example

Assembling all these concepts together, we can quickly generate and plot the quadratic interpolating polynomial

```
x = [-1 1 2]';
y = [1 3 0]';
A = vander(x);
p = A\y;
xg = linspace(-2,3,21)';
yg = polyval(p,xg);
figure(1)
hold on
plot(xg,yg,'b-')
plot(x,y,'k*')
hold off
xlabel('x')
ylabel('y')
title('Quadratic Interpolation')
```