MA 348 - Numerical Analysis
Project 4

Samuel Yaney, Julian Pryde, and Sean Holden

November 17, 2016

1 Introduction

When a line can only be described by a series of points on it, it is often useful to approximate the position of the line at points between the given ones. This can be done with several different techniques, such as spline approximation, and polynomial regression. The method looked at in this project will be spline approximation. The results of the team's MATLAB implementation of the spline approximation will be compared to MATLAB's own implementation of the spline approximation.

The points used to test the spline approximation method describe the roof of the car in the graphic below. At the end of the project, the authors will also be comparing the line obtained with both methods of interpolation with the image of the car to test the accuracy of the approximation.

The points that describe the car's roof are in the following table.

Figure 1: The car whose roof is described by the given points

Car roof points	
X	**Y**
1.25	1.15
1.60	1.48
2.21	1.71
2.78	1.83
3.60	2.17
4.26	2.47
4.82	2.64
5.50	2.74
6.21	2.76
6.88	2.74
7.58	2.60
8.31	2.30
8.93	2.10

2 Theory-Analysis

2.1 Divided Difference

Divided difference is an algorithm that can be used to calculate the coefficients of interpolation polynomials. The basic form of the polynomial is as follows:

$$f(x) = a_1+a_2(x-x_0)+a_3(x-x_0)(x-x_1)+...+a_n(x-x_0)(x-x_1)...(x-x_n) \quad (1)$$

Suppose there exist $n+1$ data points of the form $(x_0, f(x_0))...(x_n, f(x_n))$, then there are $n+1$ zeroth divided differences of the form

$$f[x_i] = f(x_i), \tag{2}$$

n first divided differences of the following form:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}, \tag{3}$$

and so on until there is a single n^{th}-divided difference of the following form:

$$f[x_0, x_1, ...x_n] = \frac{f[x_1, x_2, ...x_n] - f[x_0, ...x_{n-1}]}{x_n - x_0} \tag{4}$$

Each zeroth divided difference is substituted for its corresponding a coefficient in equation 1 with the 1st zeroth divided difference being substituted for the a_1 coefficient, the 2nd zeroth difference being substituted for the a_2 coefficient and so on.

2.2 Spline Approximation

The spline approximation works by creating a 3rd-degree polynomial between each of the given points $(x(n), y(n))$. At any given point, the polynomials on either side of it have the same position, slope, and concavity at the point where they touch. Since the position where they touch equals the position of the given point, this position can be defined exactly. With this information, the first and second derivative of the equations can be compared to find the slope and concavity of the equations at the point where they touch. A generalized third degree polynomial is defined below:

$$S_i(x) = a + b(x - x_i) + c(x - x_i)^2 + d(x - x_i)^4 \tag{5}$$

By finding separate equations for three of the a, b, c, and d coefficients in this equation, they can be substituted for in equation 5, its value can be found, and thus the values of all other coefficients can be found. From now on, $x_{i+1} - x_i$ will be referred to as h_i for conciseness.

Because two lines' positions, slopes, and concavities are equal at a point that they have in common, the following can be inferred for any equation i:

$$S_i(x_{i+1}) = S_{i+1}(x_{i+1}) \tag{6}$$

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \tag{7}$$

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \tag{8}$$

In addition, because adjacent polynomials have common positions at the given points, it can be assumed that

$$S_i(n) = y(n) \tag{9}$$

Finally, because $S_i(x)$ is equal to the a coefficient, it can also be assumed that

$$a = y \tag{10}$$

for all given points y_i. Now that a separate equation for the a coefficient has been found, equations for the other coefficients must also be found.

Using equations 5, 6, and 10, it can be inferred that

$$y_{i+1} - y_i = b_j(h_i) + c_j(h_i)^2 + d_j(h_i)^3 \tag{11}$$

and using equations 7 and 8, the following two equations can be inferred by taking the derivative and second derivative of S_i:

$$b_{i+1} = b_i + 2c_i(h_i) + 3d_i(h_i)^2 = S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \tag{12}$$

$$c_{i+1} = 2c_i + 6d_i(h_i) = S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \tag{13}$$

An equation for the d variable can be found by rearranging equation 12 as such:

$$d_i = \frac{2c_i - c_{i+1}}{h_i} \tag{14}$$

An equation for the b variable can be found in the following way.

$$y_{i+1} - y_i = b_i h_i + c_i h_i^2 + \frac{2c_{i+1} - 2c_i}{6h_i} h_i^3 \tag{15}$$

and by algebraic simplification,

$$y_{i+1} - y_i = b_i h_i + c_i h_i^2 + \frac{2c_{i+1} - 2c_i}{6} h_i^2 \tag{16}$$

$$y_{i+1} - y_i = b_i h_i + c_i h_i^2 + \frac{c_{i+1} h_i^2}{3} - \frac{c_i h_i^2}{3} \tag{17}$$

$$y_{i+1} - y_i = b_i h_i + \frac{2c_i h_i^2}{3} + \frac{c_{i+1} h_i^2}{3} \tag{18}$$

$$h_i f[x_i, x_{i+1}] = b_i h_i + \frac{2c_i h_i^2}{3} + c_{i+1}\frac{h_i^2}{3} \tag{19}$$

$$b_i = f[x_i, x_{i+1}] - c_i h_i - d_i h_i^2 \tag{20}$$

Another way of writing the equation for b is using equation 12 and replacing i with $i-1$ and $i+1$ with i. This moves the focus from the left endpoint of the polynomial to the right endpoint.

$$b_i = b_{i-1} + 2c_{i-1}h_{i-1} + 3d_{i-1}h_{i-1}^2 \tag{21}$$

Equation 20 can be substituted for the left hand side of this equation to form:

$$f[x_i, x_{i+1}] - c_i h_i - d_i h_i^2 = b_{i-1} + 2c_{i-1}h_{i-1} + 3d_{i-1}h_{i-1}^2 \tag{22}$$

If equations in terms of c are substituted for d coefficients in this equation, those c's can be solved for to find one of the three unknown coefficients. These replacements use equation 14 and 20.

$$\begin{aligned} f[x_i, x_{i+1}] - c_{i-1}h_{i-1} - \frac{2c_i - 2c_{i-1}}{6}h_i + 2c_{i-1}h_{i-1} + (c_i - c_{i-1})h_{i-1} = \\ f[x_{i-1}, x] - c_i h_i - \frac{c_{i-1}h_i}{3} \end{aligned} \tag{23}$$

Finally, this can be reorganized to be the following:

$$3(f[x_i, x_{i+1}] - f[x_{i-1}, x]) = c_{i-1}h_{i-1} + 2c_i[h_{i-1} + h_i] + c_{i+1}h_i \tag{24}$$

Because this equation references the given points on either side of the one being evaluated, the equation can only be evaluated for points 1 to $n-1$. For this project, the value of $S_0''(x_0)$ was assumed to be 0, meaning that c_0 was 0 and the value of $S_{n-1}''(x_n)$ was assumed to be 0, meaning that c_n was 0. When evaluating for equation 24 at $i = 1$ and $i = n-1$, the first and last terms on the right hand side of the equation were zero, as the c coefficient at those points was assumed to be zero.

To find all $n-2c$-values, equation 24 was evaluated in a matrix equation of the form $Ax = B$ with the row 1 of the A matrix containing the equation evaluated at $i = 1$, row 2 containing the equation evaluated at $i = 2$ and so on until row $n-1$. The first column of the matrix contains the coefficient to the c_1 variable in equation 24, the second column contains the coefficient to the c_2 variable in the equation and so on until row $n-1$.

3 Numerical Solution

Using MATLAB R2015b, the given coordinates were hardcoded as separate x and y vectors. Along with this a matrix of dimensions 11x11 was created and filled with zeros. This matrix was only a 11x11 because of the initial conditions given when forming a natural spline (c_0 and c_n are equal to 0) which leaves only 11 values that need to be calculated. This matrix was filled on the diagonal using the equations for h. This matrix was then converted into an augmented matrix and using Gaussian Elimination without pivoting the values for c_1 to c_{n-1} were calculated. Using these values, the coefficients b and d were calculated (the values of a did not have to be calculated as they are synonymous with the vector of y values). To solve for each of these sets, MATLAB's for-end loops were used to fill vectors with the data.

Once all of the coefficients had been solved for, a final for loop was used in conjunction with the 'hold on' command to ensure that each portion of the piecewise function could be plotted simultaneously. Since there was not a complete function available to be plotted, a set of coordinates had to be used. For each section (S_i) that was plotted, approximately 100 coordinate pairs were used to plot the spline. Ideally the function itself would have been used to create the plot but due to the density of the coordinates, and error that this may have induced is negligible.

4 Results and Discussion

The MATLAB implementation of the Spline Approximation created by the team proved to very closely approximate the roof of the car in the picture. Two graphics are shown below, of the result of MATLAB's own implementation of the spline approximation of the car, as well as the same image with the result of the team's implementation of the spline approximation overlayed.

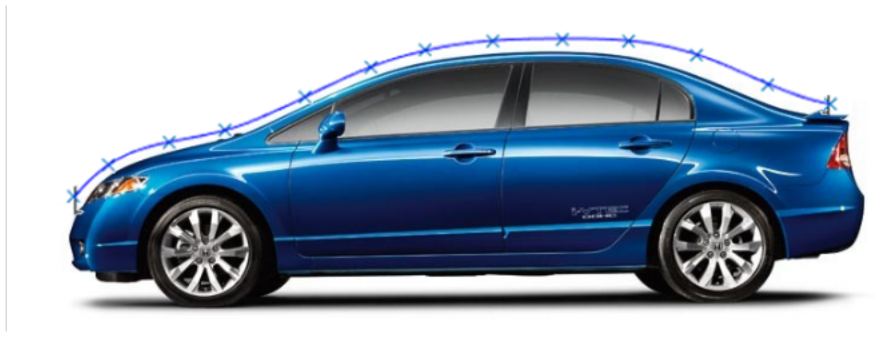

Figure 2: An image of the car with the result of MATLAB's implementation of the spline approximation overlayed

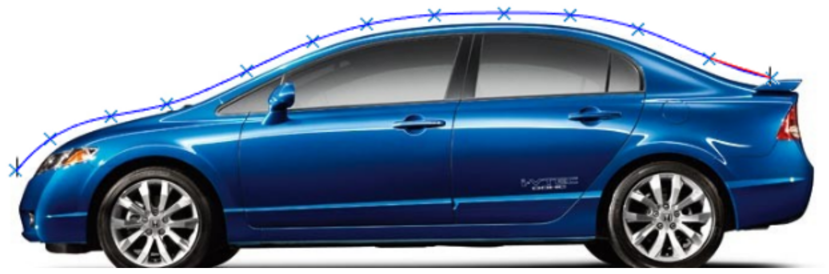

Figure 3: An image of the car with the result of the team's implementation of the spline approximation overlayed. The team's implementation is in red.

The only difference between the team's implementation of the spline approximation was between points 12 and 13, where MATLAB used a different set of end point conditions to create its spline than were used by the team.

5 Conclusions

A spline approximation is an efficient way of interpolating a line between a given set of points that meets each point. In this project, it has been proven algebraically, and reinforced numerically that a spline approximation can be used to estimate the y-value of any x-value along a line described by a series of points.

6 Appendix A: MATLAB Spline Approximation Implementation

```
%build matrix to find c
for i=1:11
    h1=(x(i+1)-x(i));
    h2=(x(i+2)-x(i+1));

    matrix(i,i)=(2*(h1+h2));
    if i<11
    matrix(i,i+1)=h2;
    end
    if i>1
    matrix(i,i-1)=h1;
    end
end

%build vector on right side
for j=1:11
     f1=(y(j+1)-y(j))/(x(j+1)-x(j));
     f2=(y(j+2)-y(j+1))/(x(j+2)-x(j+1));

     c(j) = f2-f1;
end

matrix(:,12)=c;
%use gaussian elimination to find c
run('Gaussian_Elimination_Example.m')

c_ans=[0;c_ans];
c_ans(13)=0;
```

```
%loop to find d
d=zeros(12,1);
for k=1:12
    h_k=(x(k+1)-x(k));
    d(k)=(c_ans(k+1)-c_ans(k))/(3*h_k);
end

%loop to find b
b=zeros(12,1);
for l=1:12
b(l)=((y(l+1)-y(l))/(x(l+1)-x(l))) -...
           ((2/3)*c_ans(l)*(x(l+1)-x(l))) -...
           ((1/3)*c_ans(l+1)*(x(l+1)-x(l)));
end
%build and plot Si
figure;
hold on;
ytotal=0;
for t = 1:12
    f =  @(q) y(t)+b(t).*(q-x(t))+...
          c_ans(t).*(q-x(t)).^2+d(t)*(q-x(t)).^3;
    q0=linspace(x(t),x(t+1));
    y0= f(q0);
    plot(q0,y0,'r')
    ytotal=[ytotal,y0];
end
ytotal(1)=[];
axis equal
plot(x,y,'bo')
hold off

%use MATLAB's spline function to compare
xx = linspace(x(1),x(13),1200);

yy = spline(x,y,xx);

figure;
plot(x,y,'x',xx,yy,'b')
```

```
axis equal

figure;
plot(x,y,'x',xx,yy,'b',q0,y0,'r')
axis equal

%find error
difference = ytotal-yy;
error=sum(difference)/1200;
```