Team members: Jeff Duong

Numerical Methods COSC 4364

Project Report – Final

Cubic B-Spline Interpolation and Error Estimation

Project Description:
First, we explore concept of cubic spline interpolation. We may include a case of Lagrange polynomial interpolation to see a more basic approach. Using cubic spline interpolation, we can create a less jagged, more consistent piecewise polynomial. With this concept, we can possibly reduce error at the endpoints, given an interval for a polynomial.
We will use a set of nodes to follow a curve and figure out the curvature of a curve. We may have a function in which we can derive, let's say, f(x). Each node must be greater or less than the next.

Concepts:
We can take a look at Lagrange to see a more basic approach to interpolating.
With the interpolating polynomial, we can approximate the value of the function at any "x" in the given interval.
A Lagrange polynomial can be given with the following formula:

$$L(x) = (x - x1)(x - x2) / (x0 - x1)(x0 - x2)$$

The x values will be our nodes, and it is a degree two since we compute with two nodes. This will change depending on the case.
And the polynomial given by:

$$p(x) = f(x0)Ln,0(x) + f(x1)Ln,1(x) + \cdots + f(xn)Ln,n(x)$$

The n will be our degree of polynomial.

Our result will be some polynomial that we can use to approximate values, like **f(x).** And we can approximate the derivative as well, **f'(x).** On graph, we can see the many approximations, and the approximations have pretty small error.

Now, we move on to cubic spline with that familiar concept in mind.
Here are some common formulas to work with:
"Interpolant" – This is the interpolant of the spline

$$Sj(x) = aj + bj(x - xj) + cj(x - xj)^2 + dj(x - xj)^3$$
on interval [xj, xj+1]

For error estimation we can use:

$$\max a \le x \le b \mid f^{(k)}(x) - S^{(k)}(x) \mid \le C_k(\max 0 \le j \le n-1 \mid x_{j+1} - x_j \mid)^{4-k}$$

For error estimation, we take some function, f(x), and subtract our interpolant, S(x). Another variable we might see and use often is **h(step size)**, which is basically the difference or increment between each node.

The Code (**using Matlab**):

To explain more of the process, we will walk through the code.
We start out by taking the input of the node values, x and f(x). The 'x' values are placed into a matrix. We just treat it as a list. The 'x' values should be increasing. Our f(x) values are placed into variable, A_coefficient. These will be the values of our A(i). coefficients from the interpolant equation.

$$Si(x) = ai + bi(x - xi) + ci(x - xi)^2 + di(x - xi)^3$$
$$\text{on interval } [xi, xi+1]$$

Next, we have 'h_stepSize', which is 'h' or the step size. This is the difference or increment between each node. So, for example, if $x_1 = 1$ and $x_2 = 2$, then the step size would be 2-1 = 1.

We will be dealing with Crout's factorization in many of the next parts. Based on the general form of tridiagonal system we find:

$$\begin{bmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_{n-1} \\ D_n \end{bmatrix} = \begin{bmatrix} 3(y_1 - y_0) \\ 3(y_2 - y_0) \\ 3(y_3 - y_1) \\ \vdots \\ 3(y_{n-1} - y_{n-3}) \\ 3(y_n - y_{n-2}) \\ 3(y_n - y_{n-1}) \end{bmatrix}.$$

We calculate the sub interval values. From the form 3(y-y) and the step size we use the equation:

$3*(y_{n+2}*h - y_{n+1}*(x_{n+2} - x_n) + y_n*h) / (h_{n+1}*h_n)$

From LU decomposition in Crout factorization, we create L(lower) values and U(upper) values. We also create a variable to store the y values as in **Ly = b** of LU decomposition. The first value in lower matrix is 1, and the first value in upper matrix is 0 so we set those accordingly.

Then, we calculate the next L and U values. To find an L value, we will use from Crout's factorization:

L = 2(x – x) – h*U

Then, to find U, we use **U = h / L.**

At the end of Crout factorization, we calculate the spline coefficient values using the equations:

B(i) = y(i+1) – y(i) / h – h(c(i) – 2*c(i))/3

C(i) = y(i+1) – U*c(i)

D(i) = y(i+1) -y(i) / 3*h

The equations are derived from the conditions of cubic splines.

We get the A(i) coefficients from the f(x) inputs.

Lastly, we plot our nodes and the spline.

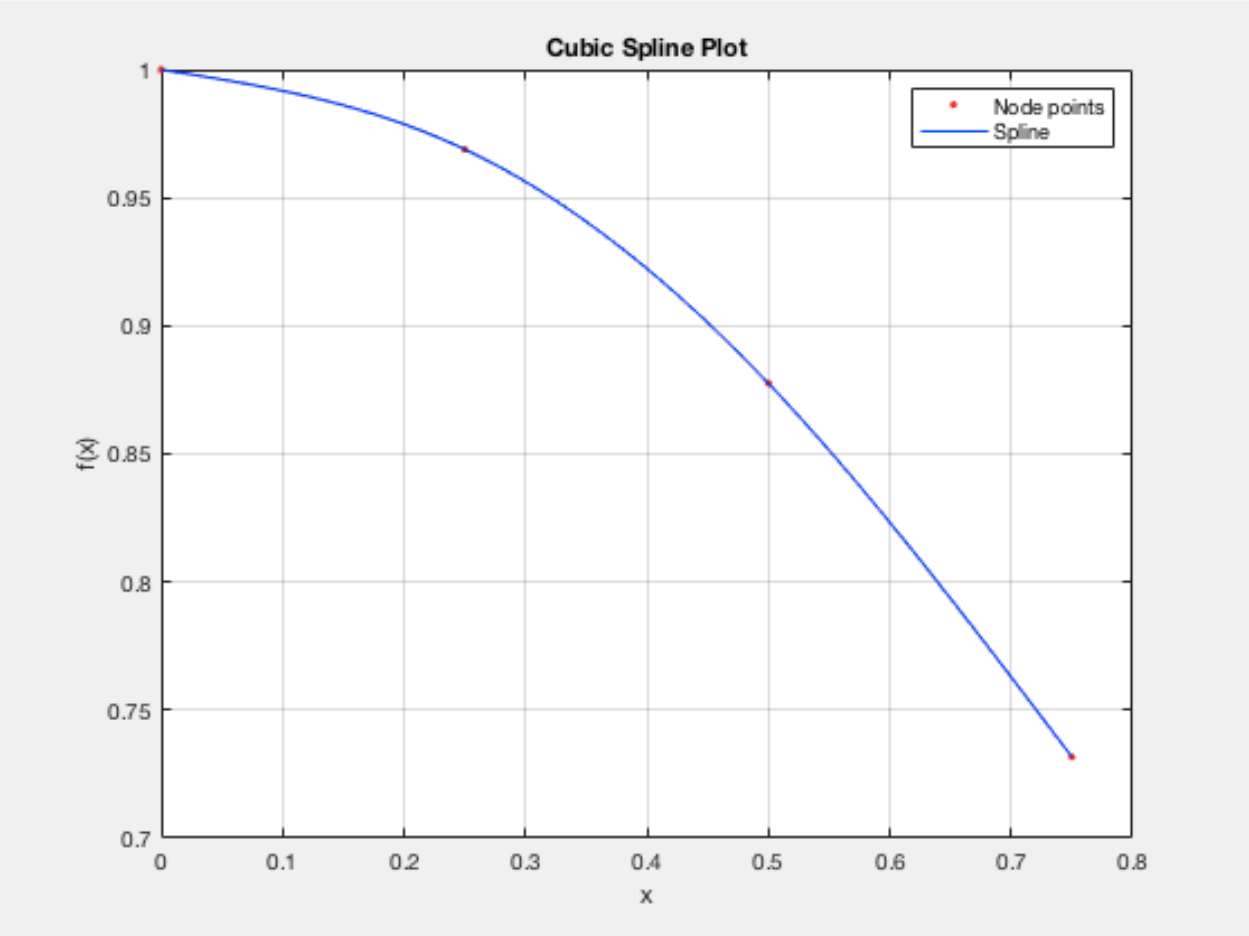
Here is our test case input and output:

```
>> cubicBSpline
------ Cubic B-spline Interpolation ------
Enter n, number of nodes, for x(1), x(2), ... , x(n):
 4
Input x(1), press enter. Then input f(x(1)), press enter:
 0
 1
Input x(2), press enter. Then input f(x(2)), press enter:
 0.25
 0.968912
Input x(3), press enter. Then input f(x(3)), press enter:
 0.5
 0.877582
Input x(4), press enter. Then input f(x(4)), press enter:
 0.75
 0.731688
------ Cubic Spline Interpolation Data ------
Nodes x(1), x(2), ... , x(n):
x(1) = 0.000000
x(2) = 0.250000
x(3) = 0.500000
x(4) = 0.750000
```

Coefficients of the cubic spline, S, on each subinterval:

	A(i)	B(i)	C(i)	D(i)
x(1):	1.000000	-0.074644	0.000000	-0.795324
x(2):	0.968912	-0.223767	-0.596493	0.121131
x(3):	0.877582	-0.499302	-0.505645	0.674193
x(4):	0.731688	0.000000	0.000000	0.000000

>>



The input we use is based on the values of **f(x) = cos(x)**

As we can see, our spline runs through our nodes, which is our real data. The curve should represent fairly accurate approximation.

Conclusion:
We have only touched the surface, but we can see that cubic spline can approximate fairly accurately. It is a more complex process than Lagrange and other interpolation techniques. They both require interpolating polynomials that we must construct with nodes, function values, and given equations.

Sources:

Weisstein, Eric W. "Cubic Spline." From MathWorld--A Wolfram Web Resource.
http://mathworld.wolfram.com/CubicSpline.html

http://www.physics.utah.edu/~detar/phys6720/handouts/crout.txt

https://www.geeksforgeeks.org/l-u-decomposition-system-linear-equations/

http://vis.berkeley.edu/courses/cs184-fa10/WWW/slides/13-SplinesAndNURBS.pdf

http://www.maths.lth.se/na/courses/FMN081/FMN081-06/lecture11.pdf

Numerical Analysis 10th Edition, R.L. Burden, D.J. Faires, A.M. Burden