FCM 1 HW 3

Ryan Bausback

October 2020

1 Executive Summary

Our goal is to implement two different interpolation strategies: a piecewise interpolating polynomial of degree either 1 or 2, and a cubic interpolating spline with either a second derivative parameterization or B-spline parameterization. After the correctness of the codes is verified through simple examples and duplicating the theoretical rates of convergence, the algorithms will be used to evaluate three interrelated functions on a non-uniform mesh. Our results indicate that the cubic splines likely produce better estimates of the three functions than the piecewise interpolation due to their continuous differentiability.

2 Statement of Problem

A problem arises from normal polynomial interpolation in that there is no guarantee of convergence to the function being interpolated, even when the degree increases and the number of points increases.[1] Piecewise polynomial interpolation solves this problem because as the number of interpolating nodes increases, the length of each interval of interpolation decreases, resulting in convergence as the number of points goes to infinity.[2] The splines go one step further in that the spline within each interval is continuously differentiable. [4]

Our goal, therefore, is to empirically demonstrate these facts for a sampling of test functions. These functions will be $y = x^3$, as the interpolating splines will be cubic, $y = \frac{2}{1+10x^2}$, to illustrate the Runge's phenomenon does not occur for these piecewise strategies, and then onto variations of these functions with the coefficients randomly generated from [1,100]. Accuracy will be shown by first comparing the interpolating function's output for a series of x values to the true function and then by confirming convergence of the $\log_2$ of the errors to the appropriate values based on the error bounds.

Once the algorithms are shown to be correct, they will be applied to three interrelated functions: y(t), $f(t) = y(t) + ty'(t)$, and $G(t) = e^{-ty(t)}$ to see if the other two can be recovered if y(t) is specified. This is expected to be the case, although because of their continuous differentiability, the splines are expected to be more accurate at this task that the regular piecewise linear or quadratic interpolations.

3 Description of Mathematics

3.1 Piecewise Polynomials

For the piecewise polynomial, the interpolation is defined differently for each interval, using a chosen basis. In this case, the Lagrange basis was chosen, resulting in

$$p_{1,i}(x) = f(x_{i-1})\frac{x - x_i}{x_{i-1} - x_i} + f(x_i)\frac{x - x_{i-1}}{x_i - x_{i-1}}, \qquad x \in [x_{i-1}, x_i]$$

for each interval in the linear case, and

$$p_{2,i}(x) = f(x_{i-2})\frac{(x - x_{i-1})(x - x_i)}{(x_{i-2} - x_{i-1})(x_{i-2} - x_i)} + f(x_{i-1})\frac{(x - x_{i-2})(x - x_i)}{(x_{i-1} - x_{i-2})(x_{i-1} - x_i)}$$
$$+ f(x_i)\frac{(x - x_{i-2})(x - x_{i-1})}{(x_i - x_{i-2})(x_i - x_{i-1})}, \qquad x \in [x_{i-2}, x_i]$$

for each interval in the quadratic case.[3] This results in $p_{1,i}(x) \in C^{(1)}$ and $p_{2,i}(x) \in C^{(2)}$, but globally only $g(x) \in C^{(0)}$.

3.2 Interpolating Cubic Splines

However, this can be fixed by using splines. As discussed above, the splines match the derivatives at the nodes x_i up to the degree of the function being interpolated, without requiring extra information about the derivatives as with Hermite-Birkhoff interpolation. The general form of the cubic splines to be used is:

$$p_i(x) = s''_{i-1}\frac{(x_i - x)^3}{6h_i} + s''_i\frac{(x - x_{i-1})^3}{6h_i} + \gamma_{i-1}(x - x_{i-1}) + \tilde{\gamma}_{i-1}$$

where $h_i = x_i - x_{i-1}$.[4] Therefore, there are 4 different parameters that need to be specified for spline. Due to the spline's interpolatory nature, if we plug in x_i and x_{i-1}, we will get f_i and f_{i-1} respectively. The resulting equations can then be rearranged to get the following formulas for γ_{i-1} and $\tilde{\gamma}_{i-1}$:

$$\gamma_{i-1} = f_{i-1} - s''_{i-1}\frac{h_i^2}{6}, \quad 1 \le i \le n$$

$$\tilde{\gamma}_{i-1} = \frac{f_i - f_{i-1}}{h_i} - \frac{h_i}{6}(s''_i - s''_{i-1}), \quad 1 \le i \le n$$

If we then substitute the formulas for the γ's into the equations where the x_i's were plugged in and rearrange more, we get :

$$\mu_i s''_{i-1} + 2s''_i + \lambda_i s''_{i+1} = d$$

$$\mu_i = \frac{h_i}{h_i + h_{i+1}}; \qquad \lambda_i = \frac{h_{i+1}}{h_i + h_{i+1}}; \qquad d = \frac{6}{h_i + h_{i+1}} \left[\frac{(f_{i+1} - f_i)}{h_{i+1}} - \frac{(f_i - f_{i-1})}{h_i} \right]$$

We can now solve for the s''_i's in matrix form. However, to use Thomas's algorithm, a tridiagonal matrix is required, meaning that the first and last column must be eliminated. This is were the boundary conditions, which are also specified, become important. If we subtract the entire first column and the entire last column from the vector of d's, since only the first element of the first column and the last element of the last column are nonzero, the result becomes:

$$\begin{pmatrix} 2 & \lambda_1 & 0 & \dots & 0 \\ \mu_2 & 2 & \lambda_2 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \mu_{n-2} & 2 & \lambda_{n-2} \\ 0 & \dots & 0 & \mu_{n-1} & 2 \end{pmatrix} \begin{pmatrix} s''_1 \\ \vdots \\ s''_{n-1} \end{pmatrix} = \begin{pmatrix} d_1 - \mu_1 s''_0 \\ \vdots \\ d_{n-1} - \lambda_{n-1} s''_n \end{pmatrix}$$

We can now use Thomas's algorithm to solve for the second derivative values, given that the second derivative values for s''_0 and s''_n are specified. However, if instead the first derivatives at the boundaries are specified, two more equations are required.[4] These can be found by taking the derivative of $p_i(x)$ above, plugging in x_0 and x_n, and then, due to the problem's interpolatory nature, setting it equal to s'_0 and s'_n. This yields the following equations:

$$2s''_0 + s''_1 = \frac{6}{h_1} \left(\frac{f_1 - f_0}{h_1} - s'_0 \right) \quad \text{and} \quad 2s''_n + s''_{n-1} = \frac{6}{h_m} \left(s'_n - \frac{f_n - f_{n-1}}{h_n} \right)$$

And the following matrix with the 2 new equations at $i = 0$ and $i = n$:

$$\begin{pmatrix} 2 & 1 & 0 & 0 & \dots & 0 & 0 \\ \mu_1 & 2 & \lambda_1 & 0 & \dots & 0 & 0 \\ 0 & \mu_2 & 2 & \lambda_2 & & & \vdots \\ 0 & 0 & & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & 0 & \mu_{n-2} & 2 & \lambda_{n-2} & 0 \\ 0 & 0 & \dots & 0 & \mu_{n-1} & 2 & \lambda_{n-1} \\ 0 & 0 & \dots & 0 & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} s''_0 \\ s''_1 \\ \vdots \\ s''_{n-1} \\ s''_n \end{pmatrix} = \begin{pmatrix} \frac{6}{h_1} \left(\frac{f_1 - f_0}{h_1} - s'_0 \right) \\ d_1 \\ \vdots \\ d_{n-1} \\ \frac{6}{h_n} \left(s'_n - \frac{f_n - f_{n-1}}{h_n} \right) \end{pmatrix}$$

Since s_0'' and s_n'' are still unknown, we try to eliminate them by adding rows together and performing the above strategy of subtracting from the d vector. This is done by first multiplying row 1 by $\frac{\mu_1}{2}$ and subtracting it from row 2, and row n by $\frac{\lambda_{n-1}}{2}$ and subtract from row n-1. This yields the system below, which can be solved used in Thomas's algorithm as before. s_0'' and s_n'' can be found by substituting back into the equations above.

$$\begin{pmatrix} 2-\frac{\mu_1}{2} & \lambda_1 & 0 & \dots & 0 \\ \mu_2 & 2 & \lambda_2 & & \vdots \\ 0 & & \ddots & \ddots & 0 \\ \vdots & 0 & \mu_{n-2} & 2 & \lambda_{n-2} \\ 0 & \dots & 0 & \mu_{n-1} & 2-\frac{\lambda_{n-1}}{2} \end{pmatrix} \begin{pmatrix} s_1'' \\ \vdots \\ s_{n-1}'' \end{pmatrix} = \begin{pmatrix} d_1 - \frac{3\mu_1}{h_1}\left(\frac{f_1-f_0}{h_1} - s_0'\right) \\ \vdots \\ d_{n-1} - \frac{3\lambda_{n-1}}{h_n}\left(s_n' - \frac{f_n-f_{n-1}}{h_n}\right) \end{pmatrix}$$

3.3 Cubic B-Splines

The cubic B-splines have slightly different form than the regular cubic splines, given by $s(x) = \sum_{i=-1}^{n+1} \alpha_i B_i(x)$.[5] The basis $B_i(x)$ has the following forms on the uniform mesh:

$$\begin{aligned} B_i(x) &= \frac{1}{h^3}(x - x_{i-2})^3, \text{ if } x_{i-2} \le x \le x_{i-1} \\ &= \frac{1}{h^3}(h^3 + 3h^2(x - x_{i-1}) + 3h(x - x_{i-1})^2 - 3(x - x_{i-1})^3), \text{ if } x_{i-1} \le x \le x_i \\ &= \frac{1}{h^3}(h^3 + 3h^2(x_{i+1} - x) + 3h(x_{i+1} - x)^2 - 3(x_{i+1} - x)^3), \text{ if } x_i \le x \le x_{i+1} \\ &= \frac{1}{h^3}(x_{i+2} - x)^3, \text{ if } x_{i+1} \le x \le x_{i+2} \\ &= 0, \text{ otherwise} \end{aligned}$$

As is apparent from the above equations, although each spline is the sum of all basis functions, due to the property of limited support, the only nonzero ones are within the interval $[x_{i-2}, x_{i+2}]$, resulting in:

$$s(x) = \alpha_{i-1}B_{i-1}(x) + \alpha_i B_i(x) + \alpha_{i+1}B_{i+1}(x) + \alpha_{i+2}B_{i+2}(x)$$

If $x \in [x_i, x_{i+1}]$, then $B_{i-1}(x)$ will be the fourth equation, $B_i(x)$ should be the third equation, and so forth. However, the α_i's are still unknown so in order to find them, we must set up a system of linear equations by plugging in each x_i into $s(x)$ and solving using Thomas's algorithm, as with the regular cubic splines. Assuming we are given $s'(0) = f'(0)$ and $s'(n) = f'(n)$, then the following matrix results:

$$\begin{pmatrix} -\frac{3}{h} & 0 & \frac{3}{h} & \dots & 0 & 0 \\ 1 & 4 & 1 & & & \vdots \\ 0 & & \ddots & \ddots & & 0 \\ \vdots & & 0 & 1 & 4 & 1 \\ 0 & \dots & 0 & -\frac{3}{h} & 0 & \frac{3}{h} \end{pmatrix} \begin{pmatrix} \alpha_{-1} \\ \alpha_0 \\ \vdots \\ \alpha_n \\ \alpha_{n+1} \end{pmatrix} = \begin{pmatrix} f_0' \\ f_0 \\ \vdots \\ f_n \\ f_n' \end{pmatrix}$$

This matrix is not in tridiagonal form, so to get this form, we use the same strategy we used for the regular cubic splines. However, in this case, row 2 will be multiplied by $\frac{3}{h}$ and row n by $-\frac{3}{h}$, so that row 1 + row 2 results in column 1 being all 0's and the same for the last column. We can then completely ignore the first and last column, solve the inner matrix (which is now tridiagonal) using the same implementation of Thomas's algorithm, and then back-substitute using the equations in second and second-to-last rows to find a_{-1} and a_{n+1}. The resulting matrix to be solved by Thomas's algorithm is then given as:

$$\begin{pmatrix} \frac{12}{h} & \frac{6}{h} & \dots & 0 \\ & \ddots & \ddots & \vdots \\ \vdots & & 4 & 1 \\ 0 & \dots & -\frac{6}{h} & -\frac{12}{h} \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} f_0' + \frac{3f_0}{h} \\ \vdots \\ \frac{-3f_n}{h} + f_n' \end{pmatrix}$$

The same strategy is also true when $s_0'' = f_0''$ and $s_n'' = f_n''$ are given, except with the coefficients in the first and last equation changed. The resulting zeros in the first slot of the upper diagonal and the last slot of the lower diagonal do not affect Thomas's algorithm since the matrix is still diagonally dominate.

$$\begin{pmatrix} \frac{6}{h^2} & \frac{-12}{h^2} & \frac{6}{h^2} & \dots & 0 & 0 \\ 1 & 4 & 1 & & & \vdots \\ 0 & & \ddots & \ddots & & 0 \\ \vdots & & 0 & 1 & 4 & 1 \\ 0 & \dots & 0 & \frac{6}{h^2} & \frac{-12}{h^2} & \frac{6}{h^2} \end{pmatrix} \begin{pmatrix} \alpha_{-1} \\ \alpha_0 \\ \vdots \\ \alpha_n \\ \alpha_{n+1} \end{pmatrix} = \begin{pmatrix} f_0'' \\ f_0 \\ \vdots \\ f_n \\ f_n'' \end{pmatrix}$$

$$\begin{pmatrix} \frac{-36}{h^2} & 0 & \dots & 0 \\ & \ddots & \ddots & \vdots \\ \vdots & & 4 & 1 \\ 0 & \dots & 0 & \frac{-36}{h^2} \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} f_0'' - \frac{6f_0}{h^2} \\ \vdots \\ \frac{-6f_n}{h^2} + f_n'' \end{pmatrix}$$

3.4 Error Bounds

The absolute global error between a function and the piecewise polynomial interpolation is defined by $||f(x) - p(x)||_\infty \leq Ch^{k+1}||f^{k+1}||_\infty$, where k is the degree of the interpolant in each of the intervals.[2] Therefore, $||f(x)-p(x)||_\infty \sim h^{k+1}$ so if we change the size of the interval, there should be a proportionate change in the error. If $Err(h) = ||f(x) - p(x)||_\infty \sim h^p$ for an interval of size h and $Err(h/2) = ||f(x) - p(x)||_\infty \sim (\frac{h}{2})^p$ for an interval of size $\frac{h}{2}$, then:

$$\frac{Err(h)}{Err(h/2)} = \frac{Ch^p}{C\frac{h}{2}^p} = 2^p \Rightarrow \log_2\left(\frac{Err(h)}{Err(h/2)}\right) = p$$

where the ratio of the error with error with the interval halved should converge to p. This will be used to determine correctness. For piecewise linear $p = 2$, for piecewise quadratic $p = 3$, and for both of the cubic splines $p = 4$.[2][4]

4 Algorithm and Implementation

Separate codes were written for piecewise (both linear and quadratic), for cubic interpolating splines, and for cubic B-splines. For-loops were used to create uniform meshes for testing by starting at the a specified endpoint a and adding $i * h$ in all codes or $i * \frac{h}{2}$ for the half-sized interval used to test convergence. For-loops were also used to create a much larger evaluation mesh as well as to evaluate all of the meshes using specified test functions.

These arrays were then passed to different functions that combined both the determination of the parameters needed for each interpolation strategy with evaluation. This was done for simplicity, thereby avoiding having to return a large 2-D array full of all of the parameters that would simply be passed to a different evaluation function. This implementation therefore reduces the strain on processing power caused by such a large return.

For the piecewise polynomials, the evaluation points, labelled $xValues$, were passed individually to the function within a for-loop, where the lagrange basis form was used to create each polynomial on $[x_{i-1}, x_i]$. An IF-statement was then used to determine which local polynomial to evaluate the $xValues$ on. The combined creation and evaluation was thus $O(n^2)$ for both linear and quadratic.

For the cubic splines, since determination of the parameters requires solving a nearly-tridiagonal matrix, it was slightly more complicated. A series of for-loops was used to create a number of arrays containing the main, upper, and lower diagonals of the tridiagonal matrix. A flag was then checked to see which boundary condition would be implemented, and the arrays updating according to the specifications in Section 3 above. The two possible boundary conditions were the Hermite boundary conditions: $s'_0 = f'_0, s'_n = f'_n$ (denoted by 1 for the first derivative) and $s''_0 = f''_0, s''_n = f''_n$ (denoted by 2 for the second derivative).

These arrays were then passed to a separate function that implemented Thomas's algorithm, acquired from pg.96 of Numerical Mathematics by Quateroni,

Sacco, and Saleri.[6] This output was then used to finalize parameter computation.

Evaluation in this case was also $O(n^2)$ since an outside for-loop was used to iterate through the array of $xValues$, while an inner for-loop iterated through the nodes (x_j's) to check if $xValues_i \in [x_j, x_{j+1}]$. The evaluation was then assigned to an element of a results array to be returned.

5 Experimental Design and Results

5.1 Demonstrating Correctness

Correctness will be demonstrated in three different ways. First, the interpolating function's values will be plotted against the actual function values to demonstrate that the two graphs are indiscernible. Second, the interpolating function will be evaluated at the x_i's and the max norm taken to illustrate that the interpolant is actually interpolating. Finally, the ratio of the max norms of interpolations using interval sizes h and $\frac{h}{2}$ will be calculated to show that it converges to the predicted value of p =2, 3, or 4 respectively for each interpolation strategy.

For piecewise linear, the function $f(x) = \frac{2}{1+10x^2}$ was chosen, since this same function is used in the FCM 1 notes (Set 14, slide 23).[2] It also a variation on the Runge's function, with some of the coefficient changed, so that if there is increased numerical instability near the endpoints, then it will be present. This was not the case, as predicted. As Figure 1 illustrates, the implemented algorithm was able to mimic the graphs in notes exactly using both 4 and 9 subintervals. With each increase in the number of intervals, the piecewise functions mimics the true function better and better. Additionally, increasing the number of intervals showed that ratio of the errors converged to 2 as predicted (Figure 2). This is true even when the coefficients are randomized between 1 and 100. This random number generation was done outside of the code since the spline interpolations require the calculation of the 1st and 2nd derivatives at the endpoints, which was done outside of the code also.

For piecewise quadratic, the same function was used for easy comparison of results to the notes, as well as to piecewise linear. As Figure 3 shows, the implemented algorithm was able to mimic the results of the notes exactly.[2] The log of the ratio of the errors also converged to the predicted value of 3 for both the original and a randomized version of the function with different coefficients (Figure 4).

For the cubic interpolating splines, the interpolations values were plotted against $y = x^3$ on $[-4, 4]$ with no visually discernible difference for either the 1st or 2nd derivative boundary conditions, as Figure 5 shows. This was done because since the splines are cubics themselves, they should be able to exactly produce the true function value at every point. This was confirmed to be the case, as the max norm between the real function values and the spline was $1.421 * 10^{-14}$ even for $n = 5$ on both boundary conditions, close to machine

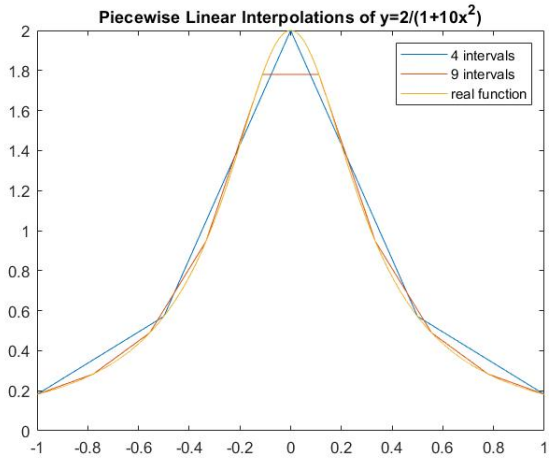

Figure 1

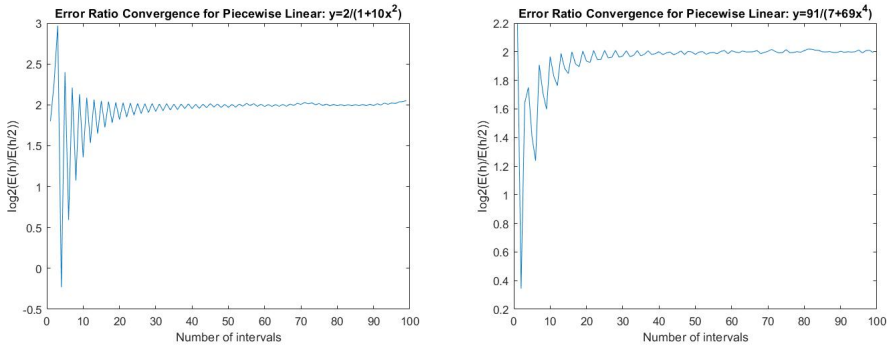

Figure 2: Convergence of the Ratio of the Error to the Error on the Half-sized Interval for $y = \frac{2}{1+10x^2}$ and randomized function

epsilon for double precision. This was the case for the B-spline interpolation as well, with an identical max norm for $n = 5$ that remained below 10^{-11} even up to $n = 640$ for both boundary conditions. This indicates that the B-spline is also able to identically interpolate any function value of a cubic polynomial, with some numerical instability at smaller intervals (where the matrix the algorithm is solving is significantly larger).

To further test each of the cubic spline's capabilities, both algorithms interpolated $y = \frac{2}{1+10x^2}$ on $[-4, 4]$, the same function used in the piecewise tests above. As Figure 6 illustrates, the interpolation is indiscernible for the true function to the naked eye. The same was also found to be true for the B-splines.

However, to truly ensure that the algorithm was performing as expected,

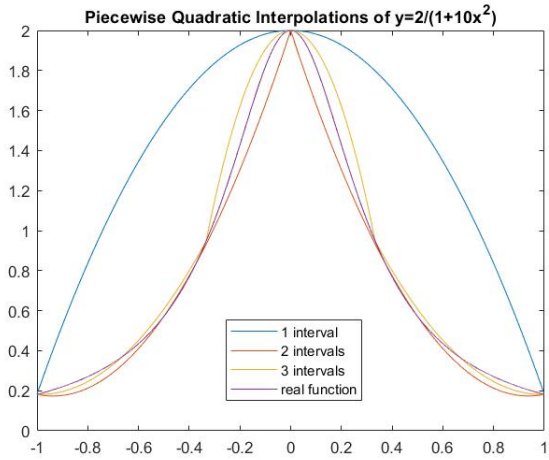

Figure 3

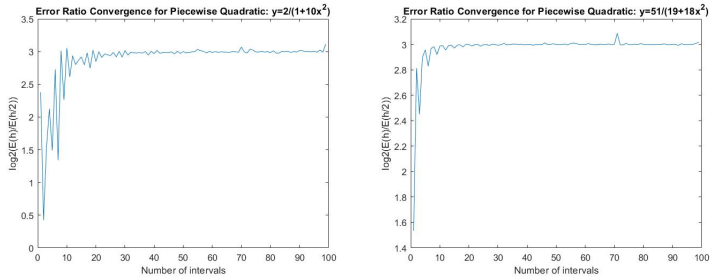

Figure 4: Convergence of the Ratio of the Error to the Error on the Half-sized Interval for $y = \frac{2}{1+10x^2}$ and randomized function

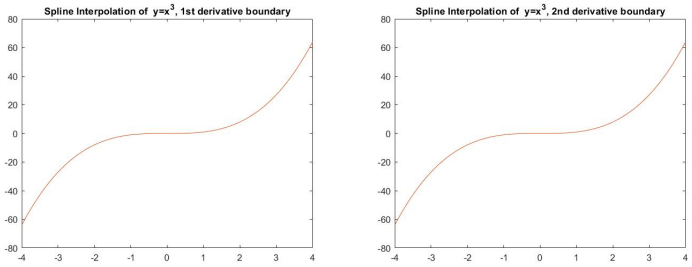

Figure 5: Real v. Cubic Spline Interpolation Function Values for $y = x^3$, n=640

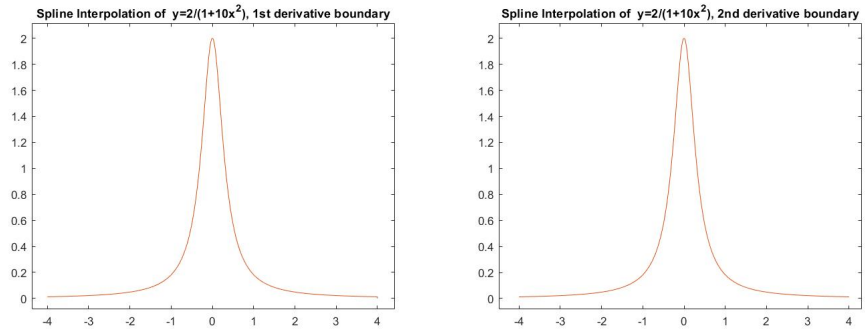

Figure 6: Real v. Cubic Spline Interpolation Function Values for $y = \frac{2}{1+10x^2}$, n=640

the $\log_2$ ratio of the errors was calculated. As discussed in Section 3, this was expected to converge to 4, which is exactly what happened. The same was also done for $y = \frac{1}{1+25x^2}$, the classical Runge's function, as well as randomized coefficient versions for the cubic splines and the B-splines respectively.

n	5	10	20	40	80	160	320	640
B-Spline, s' boundary	0.742985949	2.893849387	5.994139405	5.22245107	4.82985542	4.24477843	4.05994501	4.014982592
B-Spline, s'' boundary	0.758866505	2.893868058	5.994139405	5.22245107	4.82985542	4.24477843	4.05994501	4.014982592
Cubic Spline, s' boundary	0.742985949	2.893849387	5.994139405	5.22245107	4.829855419	4.24477843	4.05994525	4.014982608
Cubic Spline, s'' boundary	0.758866505	2.893868058	5.994139405	5.22245107	4.829855419	4.24477843	4.05994525	4.014982608

Figure 7: Convergence of $\log_2\left(\frac{Err(h)}{Err(h/2)}\right)$ for $y = \frac{2}{1+10x^2}$

n	5	10	20	40	80	160	320	640
B-Spline, s' boundary	0.742986631	2.893850646	5.994134852	5.22252202	4.83004067	4.24576178	4.06303405	4.031339692
B-Spline, s'' boundary	0.758866742	2.893869318	5.994134852	5.22252202	4.83004067	4.24576178	4.06303405	4.031339692
Cubic Spline, s' boundary	0.742864095	2.89385064	5.994134852	5.22252202	4.83004067	4.24576178	4.06303429	4.03134778
Cubic Spline, s'' boundary	0.758828312	2.893869317	5.994134852	5.22252202	4.83004067	4.24576178	4.06303429	4.03134778

Figure 8: Convergence of $\log_2\left(\frac{Err(h)}{Err(h/2)}\right)$ for $y = \frac{1}{1+25x^2}$

n	5	10	20	40	80	160	320	640
Cubic Spline, s' boundary	3.678902579	5.1110471	4.983811847	4.32509981	4.078387186	4.01921446	4.00474323	4.001333186
Cubic Spline, s'' boundary	3.741116513	5.111164114	4.983811827	4.32509981	4.078387186	4.01921446	4.00474323	4.001333186

Figure 9: Convergence of $\log_2\left(\frac{Err(h)}{Err(h/2)}\right)$ for $y = \frac{90}{95+76x^2}$

The B-splines and the cubic interpolating splines appear to be comparable in terms of accuracy on the chosen example functions as well as their randomized versions. However, convergence to the correct value predicted by the error ratio seems to be happening at a much slower pace than both piecewise interpolations.

n	5	10	20	40	80	160	320	640
B-Spline, s' boundary	0.458008187	1.857312302	4.140149386	5.10122602	5.154324315	4.55646521	4.14315271	4.035344584
B-Spline, s'' boundary	0.468161281	1.857320481	4.140149386	5.10122602	5.154324315	4.55646521	4.14315271	4.035344584

Figure 10: Convergence of $\log_2\left(\frac{Err(h)}{Err(h/2)}\right)$ for $y = \frac{9}{3+71x^2}$

Note that n=320 is 64 subintervals, which is where the cubic splines get close to the predicted value, while the piecewise interpolations are able to get the right value with only 30 subintervals. This might be related to the fact that the cubic splines require 5 points to specify on each interval, resulting in many more points needed to get the correct value.

5.2 Interrelated Functions on Non-uniform Mesh

As discussed in Section 2, the three functions to be estimated are $y(t)$, $f(t) = y(t) + ty'(t)$, and $G(t) = e^{-ty(t)}$. The following data was given for y(t):

t_i	0.5	1.0	2.0	4.0	5.0	10.0	15.0	20.0
$y(t_i)$	0.04	0.05	0.0682	0.0801	0.0940	0.0981	0.0912	0.0857

This data was then used as the initial mesh and corresponding function values for a cubic spline interpolation as well as both piecewise linear and quadratic interpolations. The boundary condition used for the cubic spline was the natural boundary (i.e. both endpoints set to 0) in the second derivatives. The results for cubic spline are shown in Figures 11 and 13. As is obvious from a comparison of the above given values and the y(t) column in Figure 13, the cubic spline is correctly interpolating the function values in both cases, giving us additional confidence that the method used to compute the rest of the values at least makes sense mathematically. For the computation of f(t) specifically, the derivative of the cubic spline form given in Section 3.2 was taken and each of the t_i's was evaluated using that functional form as well. This did not change the computational complexity beyond evaluation since the derivative uses the same parameters as the true function, making implementation simple.

For the piecewise interpolations on the data, the major problem is that $f(t)$ is defined in terms of $y'(t)$ and composite functions are generally not differentiable. To solve this problem, the derivative of the lagrange forms used for each local polynomial was taken individually away from the endpoints of each interval. However, at the endpoints, the piecewise polynomials are not continuous and so there are easily visible discontinuities in the plots of f(t) at the endpoints. Therefore, the piecewise polynomial interpolation for f(t) will not be very accurate, as Figure 12 illustrates. The Piecewise Quadratic functions were slightly closer to the cubic spline interpolation but overall, both piecewise interpolations were very far off. However, both $y(t)$ functions were still interpolatory, further confirming the algorithms' correctness.

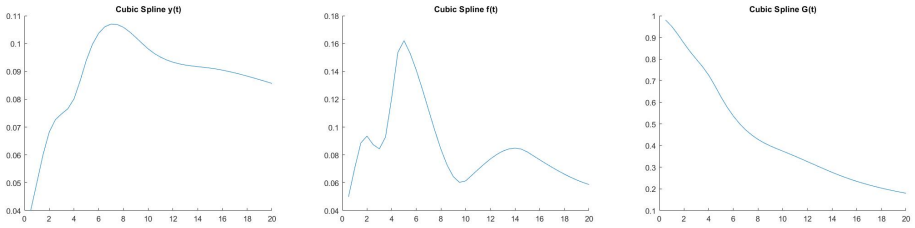

Figure 11: Cubic Spline Interpolations for y(t), f(t), and G(t)

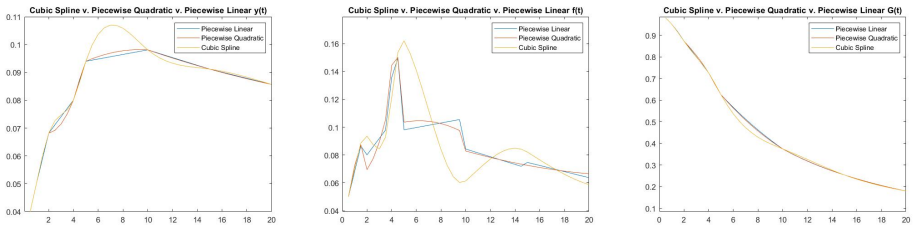

Figure 12: Piecwise Linear and Quadratic Interpolations for y(t), f(t), and G(t) compared against the Cubic Spline Interpolations

6 Conclusions

Overall, the errors for all of the piecewise linear and quadratic polynomials, the cubic splines, and the B-splines converged to the correct values under $\log_2\left(\frac{Err(h)}{Err(h/2)}\right)$. All forms were also shown to be interpolatory for both selected and randomized examples, indicating their correct funcitioning. It is interesting that the B-splines and cubic splines seemed to take longer to converge to the desired error estimated values, perhaps indicating that this implementation is not as efficient as others. However, since it takes 5 points to define the cubic splines in any interval, n will increase very fast and make it seem like it is taking longer than it truly is if only the number of intervals was investigated.

For the interrelated functions on the non-uniform mesh, the cubic splines were able to accurately interpolate the values, providing further evidence of correct functioning. Using piecewise linear or quadratic did not prove to be as effective, especially for the $f(t)$, which was defined in terms of $y'(t)$, since the local polynomials are not differentiable at the endpoints of each interval. Overall in this regard, the cubic splines gave what appear to be significantly better results, although we cannot directly compare to the unknown true function.

t	y(t)	f(t)	G(t)
0.5	0.04	0.0499	0.9802
1	0.05	0.07038	0.95123
1.5	0.06006	0.08862	0.91384
2	0.0682	0.09355	0.87249
2.5	0.07267	0.08736	0.83388
3	0.07476	0.08437	0.79908
3.5	0.07655	0.09282	0.76495
4	0.0801	0.12098	0.72586
4.5	0.08662	0.15362	0.67719
5	0.094	0.16214	0.625
5.5	0.0998	0.15289	0.57759
6	0.10374	0.1408	0.53664
6.5	0.10606	0.12686	0.50188
7	0.10702	0.11207	0.47277
7.5	0.10687	0.09743	0.44866
8	0.10585	0.08393	0.4288
8.5	0.10421	0.07258	0.41239
9	0.10221	0.06436	0.39857
9.5	0.10009	0.06028	0.38642
10	0.0981	0.06134	0.37494
10.5	0.09645	0.06538	0.36325
11	0.09513	0.06947	0.3512
11.5	0.0941	0.07343	0.33887
12	0.09331	0.07708	0.32635
12.5	0.09273	0.08023	0.31376
13	0.0923	0.0827	0.30123
13.5	0.09198	0.08431	0.2889
14	0.09172	0.08487	0.27692
14.5	0.09147	0.08421	0.26544
15	0.0912	0.08214	0.25462
15.5	0.09086	0.07931	0.24454
16	0.09046	0.07651	0.2352
16.5	0.08999	0.07377	0.22653
17	0.08948	0.07111	0.21847
17.5	0.08892	0.06858	0.21098
18	0.08832	0.0662	0.20399
18.5	0.08769	0.06399	0.19746
19	0.08704	0.06199	0.19134
19.5	0.08637	0.06023	0.18558
20	0.0857	0.05874	0.18014

t	y(t)	f(t)	G(t)
0.5	0.04	0.05	0.9802
1	0.05	0.0682	0.95123
1.5	0.0591	0.0864	0.91517
2	0.0682	0.0801	0.87249
2.5	0.07118	0.08605	0.83699
3	0.07415	0.092	0.80056
3.5	0.07713	0.09795	0.76343
4	0.0801	0.1357	0.72586
4.5	0.08705	0.1496	0.67589
5	0.094	0.0981	0.625
5.5	0.09441	0.09892	0.59496
6	0.09482	0.09974	0.56614
6.5	0.09523	0.10056	0.53849
7	0.09564	0.10138	0.51197
7.5	0.09605	0.1022	0.48657
8	0.09646	0.10302	0.46224
8.5	0.09687	0.10384	0.43894
9	0.09728	0.10466	0.41665
9.5	0.09769	0.10548	0.39532
10	0.0981	0.0843	0.37494
10.5	0.09741	0.08292	0.35958
11	0.09672	0.08154	0.3451
11.5	0.09603	0.08016	0.33143
12	0.09534	0.07878	0.31852
12.5	0.09465	0.0774	0.30632
13	0.09396	0.07602	0.29479
13.5	0.09327	0.07464	0.2839
14	0.09258	0.07326	0.27359
14.5	0.09189	0.07188	0.26384
15	0.0912	0.0747	0.25462
15.5	0.09065	0.0736	0.24535
16	0.0901	0.0725	0.23655
16.5	0.08955	0.0714	0.22819
17	0.089	0.0703	0.22025
17.5	0.08845	0.0692	0.2127
18	0.0879	0.0681	0.20552
18.5	0.08735	0.067	0.1987
19	0.0868	0.0659	0.1922
19.5	0.08625	0.0648	0.18602
20	0.0857	0.0637	0.18014

t	y(t)	f(t)	G(t)
0.5	0.04	0.0503	0.9802
1	0.05	0.07228	0.95123
1.5	0.06012	0.08742	0.91377
2	0.0682	0.0695	0.87249
2.5	0.06919	0.07744	0.84116
3	0.0715	0.08935	0.80694
3.5	0.07514	0.10524	0.76876
4	0.0801	0.14442	0.72586
4.5	0.0876	0.15015	0.67423
5	0.094	0.1036	0.625
5.5	0.09491	0.10426	0.59335
6	0.0957	0.10458	0.56316
6.5	0.09639	0.10458	0.53446
7	0.09696	0.10424	0.50727
7.5	0.09743	0.10358	0.48158
8	0.09778	0.10258	0.45738
8.5	0.09803	0.10126	0.43465
9	0.09816	0.0996	0.41336
9.5	0.09819	0.09762	0.39347
10	0.0981	0.0829	0.37494
10.5	0.09735	0.08168	0.35982
11	0.09661	0.0805	0.34553
11.5	0.09588	0.07937	0.33199
12	0.09517	0.07828	0.31916
12.5	0.09448	0.07723	0.30699
13	0.09379	0.07622	0.29544
13.5	0.09312	0.07525	0.28446
14	0.09247	0.07432	0.27402
14.5	0.09183	0.07344	0.26408
15	0.0912	0.0726	0.25462
15.5	0.09059	0.0718	0.24559
16	0.08999	0.07104	0.23697
16.5	0.0894	0.07033	0.22874
17	0.08883	0.06966	0.22088
17.5	0.08827	0.06902	0.21335
18	0.08773	0.06844	0.20614
18.5	0.0872	0.06789	0.19924
19	0.08669	0.06738	0.19261
19.5	0.08619	0.06692	0.18625
20	0.0857	0.0665	0.18014

Figure 13: Cubic Spline Interpolation for $s_0'' = s_n'' = 0$(Left), Piecewise Linear (Center), and Piecewise Quadratic(Right)

7 Program Files

All codes were compiled using Java SE 9 on my local machine using "javac" to compile and "java" to run. To implement different testing functions, uncomment the appropriate sections with "main" and/or change the appropriate labelly hard-coded values. Values do not have to be changed with the methods themselves. All graphs were produced by outputting .txt files to Matlab.

References

[1] /www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Sets/set12.pdf

[2] www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Sets/set14.pdf

[3] www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Sets/set15.pdf

[4] www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Sets/set16.pdf

[5] www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Sets/set17.pdf

[6] Quarteroni, Alfio M., et al. Numerical Mathematics. Springer Berlin Heidelberg, 2007.