Programming Assignment 6

Jenny Petrova

March 2025

1 Executive Summary

This report investigates several methods of polynomial interpolation. We begin by exploring the Barycentric Form 1 of the Lagrange interpolating polynomial $p_n(x)$, constructed on both uniform and Chebyshev mesh points (of the first or second kinds) over a specified global interval $[a, b]$. Next, we develop the piecewise cubic and piecewise Hermite cubic interpolating polynomials, $g_3(x)$, on sub-intervals $[a_i, b_i]$ within the global mesh. Finally, we construct a cubic interpolatory spline, $s(x)$, using two types of parametrization: $s''(x)$ parametrization (i.e. we apply natural boundary conditions), as well as parametrization in terms of the cubic B-spline basis. The report discusses the mathematical intuition behind these methods, and their performance is evaluated on various functions $f(x)$, through routines constructed using the Python programming language. The accuracy and space and time complexity of each method is examined.

2 Statement of the Problem

Suppose we construct a single interpolating polynomial to approximate the function $f(x) = \frac{1}{1+25x^2}$ on a uniform mesh (equidistant nodes on an interval $[a, b]$). Examine the following plot:

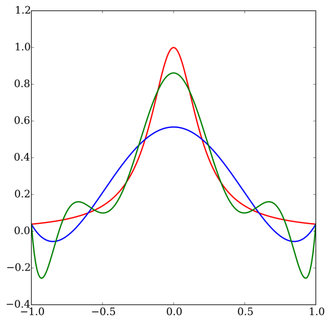

Figure 1: Runge's Phenomenon

This is a well-known result in the field of numerical analysis, called **Runge's phenomenon**. The original function $f(x)$ is shown by the red line, and the fifth order and ninth order polynomial interpolations are shown by the blue and green lines, respectively. The n-th order derivative of this function increases as the number of nodes $(n+1)$ increase. For higher order polynomial interpolations, we examine an increase in the magnitude of the oscillations near the boundaries of the interval. For such a polynomial, we may attempt to minimize the oscillations by using Chebyshev nodes of the first or second kind, which have closer spacing near the endpoints of the interval and wider spacing near the center of the interval. Chebyshev nodes can minimize the approximation error for high-degree polynomial interpolation.

Now consider an arbitrary function $f(x)$ of high degree n. An alternative approach to the interpolation problem is to partition the interval $[a, b]$, such that

$$a = x_0 < x_1 < \cdots < x_n = b,$$

and interpolate the function on each sub-interval $[x_i, x_{i+1}]$ with a low-degree polynomial. This approach constructs a **piecewise interpolating polynomial**, $g_3(x)$, which can improve the interpolation accuracy across the chosen interval.

While piecewise polynomial interpolation ensures continuity at each end point x_i, it does not address the differentiability of $g_3(x)$. This can be resolved with a **piecewise cubic Hermite interpolating polynomial**, which constructs a third-degree polynomial on each sub-interval $[x_i, x_{i+1}]$ such that the polynomial is continuous at each $f(x_i)$ *and* at the value of the first derivative of the function, $f'(x_i)$, for each $x_i \in [a, b]$. This forces the construction of a continuous interpolating polynomial $g_3(x)$ with a continuous first derivative, $g_3'(x)$.

If we went to improve the smoothness of the interpolating polynomial, we can construct an **interpolatory cubic spline**, $s(x)$, by requiring the polynomial to be continuous at each $f(x_i)$, $f'(x_i)$, *and* $f''(x_i)$, for all $x_i \in [a, b]$. This ensures the continuity of $s(x)$, $s'(x)$, and $s''(x)$ at each x_i.

We will explore these methods and compare their accuracy in polynomial interpolation and assess the behavior of their interpolating polynomials, in comparison to our chosen function $f(x)$.

3 Description of the Mathematics

3.1 Lagrange Interpolating Polynomial

Lagrange interpolation requires the computation of the Lagrange basis functions $l_0(x), \ldots, l_n(x)$ given by

$$l_i(x) = \frac{\prod_{j=0}^{n} \ \ (x - x_j)}{\prod_{j=0, j \neq i}^{n} (x_i - x_j)}, \tag{1}$$

where $l_i(x)$ is a degree n polynomial and

$$l_i(x_j) = \begin{cases} 0 & \text{if } \ i \neq j \\ 1 & \text{if } \ i = j. \end{cases} \tag{2}$$

The Lagrange Form of the unique interpolating polynomial is constructed by

$$p_n(x) = \sum_{i=0}^{n} y_i l_i(x), \tag{3}$$

where $y_i = f(x_i)$. Consider the constant function $f(x) = 1$, which is the unique interpolating polynomial for points $(x_0, 1), \ldots, (x_n, 1)$. Equation (3) gives

$$p_n(x) = \sum_{i=0}^{n} (1) l_i(x) = f(x).$$

Since $p_n(x) = f(x) = 1$, the above equation simplifies to

$$\sum_{i=0}^{n} l_i(x) = 1. \tag{4}$$

This result is necessary for the derivation that follows.

3.1.1 Barycentric Form 1

From the Lagrange interpolating polynomial (1), define the following:

$$w_{n+1}(x) = \prod_{j=0}^{n} (x - x_j) \tag{5}$$

and

$$w'_{n+1}(x_i) = \prod_{j=0, j \neq i}^{n} (x_i - x_j). \tag{6}$$

Equation (1) becomes

$$l_i(x) = \frac{w_{n+1}(x)}{(x - x_j) w'_{n+1}(x_i)} \tag{7}$$

and the **Barycentric Form 1** interpolation formula is given by

$$\begin{aligned} p_n(x) &= w_{n+1}(x) \sum_{i=0}^{n} \frac{y_i}{(x-x_j){w'}_{n+1}(x_i)} && (8) \\ &= w_{n+1}(x) \sum_{i=0}^{n} y_i \frac{\gamma_i}{(x-x_j)}, && (9) \end{aligned}$$

where

$$\gamma_i^{-1} = {w'}_{n+1}(x_i). \tag{10}$$

3.1.2 Uniform Nodes

Suppose we take an interval $[a, b]$ of equally spaces nodes $x_i = a + ih$ for $h = (b-a)/n$. Our equation for γ_i^{-1} (10) can be expanded such that

$$\begin{aligned} \gamma_i^{-1} &= \prod_{j=0}^{i-1}(x_i - x_j) \prod_{j=i+1}^{n} (x_i - x_j) \\ &= \prod_{j=0}^{i-1}(a + ih - (a + jh)) \prod_{j=i+1}^{n} (a + ih - (a + jh)) \\ &= \prod_{j=0}^{i-1}(i-j)h \prod_{j=i+1}^{n} (i-j)h. \end{aligned}$$

Using factorial notation,

$$\prod_{j=0}^{i-1}(i-j) = i!,$$

and

$$\prod_{j=i+1}^{n} (j-i) = (n-i)!,$$

so we obtain

$$\gamma^{-1} = (-1)^{n-i} h^n (i!)(n-i)!. \tag{11}$$

Then

$$\begin{aligned} \gamma_i &= \frac{(-1)^{n-i}}{h^n (i!)(n-i)!} \\ &= \frac{(-1)^{n-i}}{h^n n!} \binom{n}{i} \\ &= \frac{(-1)^n}{h^n n!} \left[(-1)^i \binom{n}{i} \right] \\ &= \frac{(-1)^n}{h^n n!} \beta_i \end{aligned}$$

such that

$$\beta_{i+1} = -\beta_i \frac{n-i}{i+1}. \tag{12}$$

Therefore the Barycentric Form 1 interpolation formula (8) can be adapted such that

$$\begin{aligned} p_n(x) &= w_{n+1}(x) \sum_{i=0}^{n} y_i \frac{\gamma_i}{(x-x_j)} \\ &= \tilde{w}_{n+1}(x) \sum_{i=0}^{n} y_i \frac{\beta_i}{(x-x_j)}. \end{aligned}$$

3.1.3 Chebyshev Nodes

Chebyshev polynomials of the first kind (T_n) are defined by

$$T_n(\cos\theta) = \cos(n\theta).$$

Chebyshev nodes of the first kind are given by the zeros of polynomial T_n, where

$$x_i = \cos\left(\frac{(2i+1)\pi}{2n+2}\right), \quad 0 \le i \le n. \tag{13}$$

Therefore the coefficients (β_i) for the Barycentric form interpolating polynomial (13), for these nodes, are

$$\beta_i = (-1)^i \sin\left(\frac{(2i+1)\pi}{2n+1}\right). \tag{14}$$

Similarly, Chebyshev polynomials of the second kind (U_n) are defined by

$$U_n(\cos\theta)\sin\theta = sin((n+1)\theta).$$

Chebyshev nodes of the second kind are given by the zeros of polynomial U_n, where

$$x_i = \cos\left(\frac{i\pi}{n}\right), \quad 0 \le i \le n. \tag{15}$$

The coefficients (β_i) for the Barycentric form interpolating polynomial (13), for these nodes, are thus

$$\beta_i = (-1)^i \delta_i, \tag{16}$$

where

$$\delta_i = \begin{cases} \frac{1}{2} \text{ if } j = 0 \text{ and } j = n \\ 1 \text{ otherwise.} \end{cases}$$

3.2 Newton Interpolating Polynomial

The formula for the **Newton divided differences** is given by

$$f\left[x_0, \ldots, x_n\right] = \frac{f\left[x_1, \ldots, x_n\right] - f\left[x_0, \ldots, x_{n-1}\right]}{x_n - x_0}. \tag{17}$$

Consider a set of nodes $(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)$ with distinct values x_i. For the given nodes, the divided difference formula is used to construct the following table:

$$\begin{array}{cccc} x_0 & x_1 & x_2 & x_3 \\ f[x_0] & f[x_1] & f[x_2] & f[x_3] \\ f[x_0, x_1] & f[x_1, x_2] & f[x_2, x_3] & \\ f[x_0, x_1, x_2] & f[x_1, x_2, x_3] & & \\ f[x_0, x_1, x_2, x_3] & & & \end{array}, \tag{18}$$

where

$$\begin{aligned} f[x_0] &= y_0, \\ f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0}, \\ f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}, \\ f[x_0, x_1, x_2, x_3] &= \frac{f[x_0, x_1, x_2] - f[x_1, x_2, x_3]}{x_3 - x_0}. \end{aligned}$$

We construct the Newton interpolating polynomial using the recursive derivation of the Newton form, using the first column of the divided difference table as the coefficients of the interpolating polynomial, such that

$$\begin{aligned} p_n(x) =& f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &+ f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2). \end{aligned}$$

Therefore the definition of the **Newton interpolating polynomial** is given by

$$p_3(x) = \sum_{i=1}^{n} w_i(x)\, f[x_0, \ldots, x_n]. \tag{19}$$

3.2.1 Hermite-Birkhoff Interpolation (Newton Form)

Suppose we have a divided difference of the form $f[x_i, x_i]$. Using derivative rules, we can show

$$\begin{aligned} f[x_i, x_i] &= \lim_{x_j \to x_i} f[x_i, x_j] \\ &= \lim_{x_j \to x_i} \frac{f(x_j) - f(x_i)}{x_j - x_i} \\ &= f'(x_i) \end{aligned}$$

Now consider a set of nodes $(x_0, y_0), (x_0, y_0'), (x_1, y_1), (x_1, y_1')$, where $f'(x_0) = y_0'$, $f'(x_1) = y_1'$, and $x_0 \neq x_1$. We use the derivative rules to construct the following divided difference table:

$$\begin{array}{cccc}
x_0 & x_0 & x_1 & x_1 \\
f[x_0] & f[x_0] & f[x_1] & f[x_1] \\
f'[x_0] & f[x_0, x_1] & f'[x_1] & \\
f[x_0, x_0, x_1] & f[x_0, x_1, x_1] & & \\
f[x_0, x_0, x_1, x_1] & & &
\end{array} . \tag{20}$$

where

$$\begin{aligned}
&f[x_0] = y_0, \\
&f[x_0, x_0] = f'[x_0] = y_0', \\
&f[x_0, x_0, x_1] = \frac{f[x_0, x_1] - f[x_0, x_0]}{x_1 - x_0} = \frac{f[x_0, x_1] - f'[x_0]}{x_1 - x_0}, \\
&f[x_0, x_0, x_1, x_1] = \frac{f[x_0, x_0, x_1] - f[x_0, x_1, x_1]}{x_1 - x_0}.
\end{aligned}$$

As with the Newton interpolating polynomial, we use the first column of the divided difference table as the coefficients for the **Hermite interpolating polynomial**, $H_3(x)$, such that

$$\begin{aligned}
H_3(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_0, x_1](x - x_0)(x - x_0) \\
&\quad + f[x_0, x_0, x_1, x_1](x - x_0)(x - x_0)(x - x_1) \\
&= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_0, x_1](x - x_0)^2 \\
&\quad + f[x_0, x_0, x_1, x_1](x - x_0)^2(x - x_1)
\end{aligned}$$

We end at this definition for the cubic Hermite interpolating polynomial, as higher degree Hermite interpolation will not be necessary for this report.

3.3 Piecewise Interpolating Polynomial

Piecewise polynomial interpolation requires local interpolants of lower order rather than a global interpolating polynomial. Let $[a, b]$ denote the global interval and let $\{x_i\}_{i=0}^n$ be the set of ordered, distinct mesh points such that $x_i \in [a, b]$ for all $0 \leq i \leq n$. The global interval can be partitioned into a set of disjoint sub-intervals at the mesh points, such that

$$[a, b] = [x_0, x_1) \cup [x_1, x_2) \cup \cdots \cup [x_{n-1}, x_n) \cup [x_n] = \cup_s I_s. \tag{21}$$

For M intervals, constructing a degree k polynomial requires $Mk + 1$ mesh points on the global interval, where each sub-interval consists of $nk+1$ local mesh points. We enforce continuity on interpolating polynomial $g_k(x)$ by interpolating at the shared endpoints of each interval, such that $g_k(x_i) = f(x_i)$ for all $I_i =$

$[x_i, x_{i+1}]$, where $i = 0, 1, \dots, M-1$. By constructing a local polynomial at each sub-interval, $g_k(x)$ becomes

$$g_k(x) = \begin{cases} p_{k,1}(x) & x \in [x_0, x_1] \\ p_{k,2}(x) & x \in [x_1, x_2] \\ \vdots & \\ p_{k,M-1}(x) & x \in [x_{n-1}, x_n] \end{cases} . \tag{22}$$

3.3.1 Piecewise Linear Polynomial

Each sub-interval contains two mesh points as the endpoints, i.e. $I_i = [x_i, x_{i+1}]$ is a sub-interval where the end points are $\{x_i, x_{i+1}\} \in [a, b]$. To construct a piecewise linear (degree 1) polynomial, $g_1(x)$, we interpolate the two end points of each sub-interval. We may choose any interpolating form (Lagrange or Newton) for each interval. We choose to construct a local linear Newton interpolating polynomial $p_{1,i}(x)$ on the sub-interval, such that

$$p_{1,i}(x) = f(x_i) + f[x_i, x_{i+1}](x - x_i). \tag{23}$$

for $x \in [x_i, x_{i+1}]$.

3.3.2 Piecewise Quadratic Polynomial

For a piecewise quadratic (degree 2) polynomial, we requires 3 mesh points. To construct a local interpolating polynomial on $[x_i, x_{i+1}]$, we require 1 additional mesh point in each sub-interval. The choice of additional mesh point depends on the choice of the local mesh. For a uniform local mesh, let $h = x_{i+1} - x_i$. We consider the midpoint of each interval as the additional interpolation point, i.e. we interpolate the set of points

$$\{x_i, x_i + \frac{h}{2}, x_{i+1}\} \tag{24}$$

for each sub-interval $[x_i, x_{i+1}] \in I_i$. With the additional mesh point in each sub-interval, the global interval is now $[x_0, x_{2n}]$, meaning each local interval becomes $[x_i, x_{i+2}]$ with interpolating nodes $\{x_i, x_{i+1}, x_{i+2}\}$.

For a Chebyshev local mesh, we generate three Chebyshev nodes of the second kind on the interval $[-1, 1]$, and cast the nodes back to each interval $[x_i, x_{i+1}]$. The Chebyshev nodes for $n = 2$ are given by

$$\begin{aligned} x_0 &= \cos\left(\frac{0 \cdot \pi}{2}\right) = 1, \\ x_1 &= \cos\left(\frac{1 \cdot \pi}{2}\right) = 0, \\ x_2 &= \cos\left(\frac{2 \cdot \pi}{2}\right) = -1. \end{aligned}$$

Notice that 0 is in fact the midpoint of the interval $[-1,1]$. When we cast the distribution of these nodes back to the sub-interval $[x_i, x_{i+1}]$, we again obtain the set of interpolation nodes

$$\{x_i, x_i + \frac{h}{2}, x_{i+1}\} \quad \text{with } h = x_{i+1} - x_i$$

for each sub-interval. Therefore, in the case of quadratic local interpolation, the choice of uniform or Chebyshev local meshes does not affect the interpolating nodes. Hence we construct the local quadratic interpolating polynomial in the Newton basis for either choice of local mesh by the formula

$$p_{2,i}(x) = f(x_i) + f[x_i, x_{i+1}](x - x_i) + f[x_i, x_{i+1}, x_{i+2}](x - x_i)(x - x_{i+1}).$$

for $x \in [x_i, x_{i+2}]$.

3.3.3 Piecewise Cubic Polynomial

For a piecewise cubic (degree 3) polynomial, we require 4 mesh points for the interpolation. To construct a local interpolating polynomial on each sub-interval $[x_i, x_{i+1}]$, we require 2 additional mesh points in each sub-interval. Again, the choice of additional mesh points depends on the choice of the local mesh. Let $h = x_{i+1} - x_i$. For a uniform local mesh, interpolate the set of points

$$\{x_i, x_i + \frac{h}{3}, x_i + \frac{2h}{3}, x_{i+1}\}$$

for each sub-interval $[x_i, x_{i+1}] \in I_i$. With the additional mesh points in each sub-interval, the global interval is now $[x_0, x_{3n}]$, meaning each local interval becomes $[x_i, x_{i+3}]$ with interpolating nodes $\{x_i, x_{i+1}, x_{i+2}, x_{i+3}\}$.

For a Chebyshev local mesh, we generate four Chebyshev nodes of the second kind on the interval $[-1,1]$, and cast the nodes back to each of the intervals $[x_i, x_{i+1}]$. The Chebyshev nodes for $n = 3$ are given by

$$\begin{aligned}
x_0 &= \cos\left(\frac{0 \cdot \pi}{3}\right) = 1, \\
x_1 &= \cos\left(\frac{1 \cdot \pi}{3}\right) = \frac{1}{2}, \\
x_2 &= \cos\left(\frac{2 \cdot \pi}{3}\right) = -\frac{1}{2}, \\
x_3 &= \cos\left(\frac{3 \cdot \pi}{3}\right) = -1.
\end{aligned}$$

When we cast the distribution of these nodes back to the sub-interval $[x_i, x_{i+1}]$, we obtain the set of interpolation nodes

$$\{x_i,\ \frac{x_i + x_{i+1}}{2} - \frac{h}{4},\ \frac{x_i + x_{i+1}}{2} + \frac{h}{4},\ x_{i+1}\}, \quad \text{with } h = x_{i+1} - x_i.$$

Therefore, in the case of cubic local interpolation, the choice of uniform or Chebyshev local meshes *does* affect the values of the interior interpolating nodes on each sub-interval. For either choice of local mesh, we construct the local cubic interpolating polynomial in the Newton basis by the formula

$$\begin{aligned} p_{3,i}(x) =& f(x_i) + f[x_i, x_{i+1}](x - x_i) + f[x_i, x_{i+1}, x_{i+2}](x - x_i)(x - x_{i+1}) \\ &+ f[x_i, x_{i+1}, x_{i+2}, x_{i+3}](x - x_i)(x - x_{i+1})(x - x_{i+2}) \end{aligned}$$

for $x \in [x_i, x_{i+3}]$.

3.3.4 Piecewise Cubic Hermite Polynomial

Suppose we have the derivative values at each mesh point, meaning $f'(x_i)$ is known for each $x_i \in [a, b]$. We can construct the piecewise cubic interpolating polynomial via Hermite interpolation (20) on each interval $[x_i, x_{i+1}] \in I_i$. We enforce the following constraints:

$$\begin{aligned} H_3(x_i) = f(x_i), &\quad H_3'(x_i) = f'(x_i) \\ H_3(x_{i+1}) = f(x_{i+1}), &\quad H_3'(x_{i+1}) = f'(x_{i+1}) \end{aligned}$$

where the Newton form of the cubic Hermite interpolating polynomial is

$$\begin{aligned} H_{3,i}(x) =& f(x_i) + f'(x_i)(x - x_i) + f[x_i, x_i, x_{i+1}](x - x_i)^2 \\ &+ f[x_i, x_i, x_{i+1}, x_{i+1}](x - x_i)^2(x - x_{i+1}) \end{aligned}$$

for $x \in [x_i, x_{i+1}]$. Note that this does not require a choice for the type of local mesh points. We require only the endpoints (and the derivatives at the end points) for the interpolation.

3.4 Cubic Spline Polynomial

Definition. Given an interval $[a, b]$, let the distinct points $a = x_0 < x_1 < \cdots < x_n = b$ define a partition in to intervals $[x_i, x_{i+1})$. A cubic polynomial spline, $s(x)$, of degree 3, is a piecewise polynomial of degree 3, such that $s(t) = p_{3,i}(x)$ on $[x_{i-1}, x_i)$ for $i \leq 1 \leq n$. The cubic spline polynomials are such that their values and the values of their $1^{\underline{\text{st}}}$ and $2^{\underline{\text{nd}}}$ derivatives match at x_i for $1 \leq i \leq n-1$.

To develop the set of cubic splines, $S_3(x)$, we require n intervals each with a cubic polynomial, i.e. we require $4n$ parameters. The continuity of $s(x), s'(x), s''(x)$ at x_i, $1 \leq i \leq n-1$, each require $n-1$ parameters. Therefore the function has $4n + 3(n-1) = n + 3$ degrees of freedom and we assume

$$S_3(x) = \begin{cases} p_{3,1}(x) & x \in [x_0, x_1] \\ p_{3,2}(x) & x \in [x_1, x_2] \\ \vdots & \\ p_{3,n}(x) & x \in [x_{n-1}, x_n] \end{cases}$$

For all $0 \leq i \leq n$, let us define and denote

$$\begin{aligned} s_i = s(x_i) &= f(x_i) = f_i, \\ s'_i = s'(x_i) &= f'(x_i) = f'_i, \\ s''_i = s''(x_i) &= f''(x_i) = f''_i. \end{aligned}$$

Let $h_i = x_i - x_{i+1}$. Since $s(x)$ is piecewise cubic, then each $p''_i(x)$ is linear. Using Lagrange interpolation, we can interpolate each $s''(x_i)$ on $[x_{i+1}, x_i]$, by constructing

$$p''_i(x) = p''_i(x_{i-1})\frac{x_i - x}{h_i} + p''_i(x_i)\frac{x - x_{i-1}}{h_i}.$$

Imposing the condition that $s''(x)$ is continuous at the interval nodes, we therefore have

$$p''_i(x) = s''_{i-1}\frac{x_i - x}{h_i} + s''_i\frac{x - x_{i-1}}{h_i} \tag{25}$$

for $x \in [x_{i-1}, x_i]$. By integrating the above equation and using the condition that $s_{i-1} = f_{i-1}$ and $s_i = f_i$, equation (25) becomes

$$\begin{aligned} p_i(x) = s''_{i-1}&\frac{(x_i - x)^3}{6h_i} + s''_i\frac{(x_i - x)^3}{6h_i} \\ &+ \left(\frac{f_i - f_{i-1}}{h_i} - \frac{h_i}{6}(s''_i - s''_{i-1})\right)(x - x_{i-1}) + \left(f_i - s''_{i-1}\frac{h_i^2}{6}\right). \end{aligned}$$

When $x \in [x_{i-1}, x_i]$, we obtain the first derivative equation

$$p'_i(x) = -s''_{i-1}\frac{(x_i - x)^2}{2h_i} + s''_i\frac{(x - x_{i-1})^2}{2h_i} + \frac{f_i - f_{i-1}}{h_i} - \frac{h_i}{6}(s''_i - s''_{i-1}). \tag{26}$$

Therefore

$$p'_i(x_i) = s''_i\frac{h_i}{2} + \frac{f_i - f_{i-1}}{h_i} - \frac{h_i}{6}(s''_i - s''_{i-1}).$$

Similarly, when $x \in [x_i, x_{i+1}]$, we can shift the index to obtain

$$p'_{i+1}(x) = -s''_i\frac{(x_{i+1} - x)^2}{2h_{i+1}} + s''_{i+1}\frac{(x - x_i)^2}{2h_{i+1}} + \frac{f_{i+1} - f_i}{h_{i+1}} - \frac{h_i}{6}(s''_{i+1} - s''_i).$$

Therefore

$$p'_{i+1}(x_i) = -s''_i\frac{h_{i+1}}{2} + \frac{f_{i+1} - f_i}{h_{i+1}} - \frac{h_{i+1}}{6}(s''_{i+1} - s''_i).$$

Since $p_i(x_i) = p_{i+1}(x_i)$ is an interpolation condition, we can derive the formula

$$\mu_i s''_{i-1} + 2s''_i + \lambda_i s''_{i+1} = d_i, \tag{27}$$

for $1 \leq i \leq n-1$, where

$$\mu_i = \frac{h_i}{h_i + h_{i+1}} < 1, \tag{28}$$

$$\lambda_i = \frac{h_{i+1}}{h_i + h_{i+1}} < 1, \tag{29}$$

and

$$d_i = \frac{6}{h_i + h_{i+1}} \left[\frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i} \right] = 6f[x_{i-1}, x_i, x_{i+1}]. \tag{30}$$

To solve for each s_i'', $0 \le i \le n$, we use equation (25) to create the following system of equations:

$$\begin{pmatrix} 2 & \lambda_0 & 0 & \cdots & 0 \\ \mu_1 & 2 & \lambda_1 & \cdots & 0 \\ 0 & \mu_2 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \mu_{n-1} & 2 & \lambda_{n-1} \\ 0 & 0 & 0 & \mu_n & 2 \end{pmatrix} \begin{pmatrix} s_0'' \\ s_1'' \\ \vdots \\ s_{n-1}'' \\ s_n'' \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_n \\ d_n \end{pmatrix}. \tag{31}$$

In order to solve the above tridiagonal system, we must impose certain boundary conditions. We use the **natural boundary condition**, where we set $s''(x_0) = s''(x_n) = 0$ and let $\mu_0 = \mu_n = 0$, $d_0 = d_1$, and $d_{n-1} = d_n$. Therefore the system (31) becomes

$$\begin{pmatrix} 2 & \lambda_0 & 0 & \cdots & 0 \\ \mu_1 & 2 & \lambda_1 & \cdots & 0 \\ 0 & \mu_2 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \mu_{n-1} & 2 & \lambda_{n-1} \\ 0 & 0 & 0 & \mu_n & 2 \end{pmatrix} \begin{pmatrix} s_0'' \\ s_1'' \\ \vdots \\ s_{n-1}'' \\ s_n'' \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_n \\ d_n \end{pmatrix}. \tag{32}$$

The matrix is diagonally dominant and therefore non-singular. Hence a unique solution to this system exists.

3.4.1 Cubic B-Splines

Given a partition π on an interval $[a, b]$ where $a = x_0 < x_1 < \cdots < x_n = b$, we can construct the linear space of cubic splines $S_3(\pi)$ with dimension $n+3$. To find a basis for $S_3(\pi)$ consisting of $n+3$ linearly independent cubic splines, we first derive the **cubic B-spline**, $B_i(t)$. Assume we have a set of five uniformly spaced nodes on an interval $[a, b]$, such that $a = x_0 < x_1 < \cdots < x_4 = b$, where $h = x_{i+1} - x_i$ for all $0 \le i \le 3$. Given a function $f(x)$, we can define the forward and backward differences of $f(x)$ as follows:

$$\begin{aligned} \Delta f(x) &= f(x+h) - f(x) \\ \nabla f(x) &= f(x) - f(x-h) \\ \Delta^2 f(x) &= \Delta f(x+h) - \Delta f(x) \\ \Delta^m f(x) &= \Delta^m f(x+h) - \Delta^m f(x). \end{aligned}$$

We can write the Newton divided differences in terms of the forward differences by scaling, such that

$$\begin{aligned} f[x_0, x_1] &= \frac{\Delta f(x_0)}{h} \\ f[x_0, x_1, \dots, x_k] &= \frac{\Delta^k f(x_0)}{k! h^k}. \end{aligned}$$

Now consider the function $F_t(x)$, where

$$F_t(x) = (x-t)_+^3 = \begin{cases} (x-t)^3 & t \le x \\ 0 & t > x \end{cases}. \tag{33}$$

$F_t(x)$ is twice continuously differentiable with respect to x for a fixed t, and is piecewise cubic with respect to t for a fixed x (and vice versa). Therefore Δ^4 will have intervals where it is identically 0. We define a function $K(t)$ such that

$$\begin{aligned} K(t) &= \Delta^4 F_t(x_0) \\ &= \binom{4}{4} F_t(x_4) - \binom{4}{3} F_t(x_3) + \binom{4}{2} F_t(x_2) - \binom{4}{1} F_t(x_1) + \binom{4}{0} F_t(x_0) \\ &= F_t(x_4) - 4F_t(x_3) + 6F_t(x_2) - 4F_t(x_1) + F_t(x_0) \\ &= (x_4 - t)_+^3 - 4(x_3 - t)_+^3 + 6(x_2 - t)_+^3 - 4(x_1 - t)_+^3 + (x_0 - t)_+^3. \end{aligned}$$

We examine the behavior of $K(t)$ for all t, for a fixed x, to be

$$\begin{aligned} t \le x_0 &\to K(t) \equiv 0 \\ x_0 \le t \le x_1 &\to K(t) = (x_4 - t)^3 - 4(x_3 - t)^3 + 6(x_2 - t)^3 - 4(x_1 - t)^3 + 0 \\ x_1 \le t \le x_2 &\to K(t) = (x_4 - t)^3 - 4(x_3 - t)^3 + 6(x_2 - t)^3 - 0 + 0 \\ x_2 \le t \le x_3 &\to K(t) = (x_4 - t)^3 - 4(x_3 - t)^3 + 0 - 0 + 0 \\ x_3 \le t \le x_4 &\to K(t) = (x_4 - t)^3 - 0 + 0 - 0 + 0 \\ x_4 \le t &\to K(t) \equiv 0. \end{aligned}$$

Now, let $x_i = x_0 + ih$. The above equations for $K(t)$ become

$$\begin{aligned} t \le x_0 &\to K(t) \equiv 0 \\ x_0 \le t \le x_1 &\to K(t) = (x_1 - t + 3h)^3 - 4(x_1 - t + 2h)^3 + 6(x_1 - t + h)^3 - 4(x_1 - t)^3 \\ x_1 \le t \le x_2 &\to K(t) = (x_2 - t + 2h)^3 - 4(x_2 - t + h)^3 + 6(x_2 - t)^3 \\ x_2 \le t \le x_3 &\to K(t) = (x_3 - t + h)^3 - 4(x_3 - t)^3 \\ x_3 \le t \le x_4 &\to K(t) = (x_4 - t)^3 \\ x_4 \le t &\to K(t) \equiv 0. \end{aligned}$$

Recalling $K(t) = \Delta^4 F_t(x_0)$, we can develop a formal definition for the cubic B-spline around reference point x_i defined on a set of uniform mesh points $x_{i-2} < x_{i-1} < x_i < x_{i+1} < x_{i+2}$. The cubic B-spline is defined as

$$B_i(t) = \frac{1}{h^3} \Delta^4 F_t(x_{i-2}), \tag{34}$$

where $h = x_i - x_{i-1}$. Therefore, the cubic B-spline is twice-continuously differentiable on $\mathbb{R}$, and is identically 0 when $x \geq x_{i+2}$ or $x \leq x_{i-2}$. On a uniform mesh, we can simplify the above equations to show the cubic B-spline has the following values:

	x_{i-2}	x_{x-1}	x_i	x_{i+1}	x_{i+2}
$B_i(x)$	0	1	4	1	0
$B_i'(x)$	0	$\frac{3}{h}$	0	$-\frac{3}{h}$	0
$B_i''(x)$	0	$\frac{6}{h^2}$	$-\frac{12}{h^2}$	$\frac{6}{h^2}$	0

Therefore, given we have values f_0', f_n', f_i, and a set of uniform nodes $x_0 < x_1 < \cdots < x_n$, the unique cubic spline $s(x) \in \mathcal{B}$, where $\mathcal{B} = span\{B_{-1}, B_0, \ldots, B_{n+1}\}$, such that

$$s(x_i) = f_i, \quad s'(x_0) = f_0', \quad s'(x_n) = f_n'$$

for $0 \leq i \leq n$, is given by

$$s(x) = \sum_{i=-1}^{n+1} a_i B_i(x). \tag{35}$$

To solve for each a_i, we can construct a linear system of equations, where

$$\begin{pmatrix} -\frac{3}{h} & 0 & \frac{3}{h} & \cdots & 0 \\ 1 & 4 & 1 & \cdots & 0 \\ 0 & 1 & 4 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & -\frac{3}{h} & 0 & \frac{3}{h} \end{pmatrix} \begin{pmatrix} a_{-1} \\ a_0 \\ \vdots \\ a_n \\ a_{n+1} \end{pmatrix} = \begin{pmatrix} f_0' \\ f_0 \\ \vdots \\ f_n \\ f_n' \end{pmatrix}. \tag{36}$$

The above system is again a tri-diagonal system for which we can use a triangular solve to find the vector of a_i coefficients.

4 Description of the Methods, Algorithms and Implementation

For the construction and testing of the various polynomial interpolation methods, we develop several routines in Python. We present the relevant pseudo-code below. Each function runs in double precision (IEEE standard floating point arithmetic).

4.1 Lagrange Interpolating Polynomial Routines

We develop a routine to generate Chebyshev nodes of the first or second kind. The algorithm accepts a the end points of an interval $[a.b]$, the specified number of nodes n, and the type of Chebyshev nodes to be generated ($kind = 1$ indicates Chebyshev nodes of the first kind, $kind = 2$ indicates Chebyshev nodes of the second kind). The routine evaluates $n + 1$ Chebyshev nodes on the interval $[-1, 1]$ and casted their values to the specified input interval $[a, b]$, and returns an array of the casted values. These values will be used at the interpolation nodes to construct an interpolating polynomial on an interval $[a, b]$.

Algorithm 1 Chebyshev Nodes

```
1:  procedure CHEBYSHEV_NODES($n, a, b, kind$)
2:      if kind $= 1$ then
3:          $k \leftarrow [\,0, 1, \ldots, n\,]$
4:          mesh $\leftarrow \cos\left(\frac{(2\,k+1)\,\pi}{2\,(n+1)}\right)$
5:      else if kind $= 2$ then
6:          $k \leftarrow [\,0, 1, \ldots, n\,]$
7:          mesh $\leftarrow \cos\left(\frac{\pi\,k}{n}\right)$
8:      end if
9:      $x_i \leftarrow \dfrac{(a+b)}{2} + \left(\dfrac{(b-a)}{2} \cdot \text{mesh}\right)$
10:     return $x_i$
11: end procedure
```

The following algorithm computes the weights of the Lagrange interpolating polynomial. The routine accepts the interpolation nodes x_i and returns an array of the weights γ_i.

Algorithm 2 Barycentric Weights

```
1: procedure BARYCENTRIC_WEIGHTS($x_i$)
2:     $n \leftarrow |x_i|$
3:     $\gamma \leftarrow [1, 1, \ldots, 1]$                    ▷ length $n$
4:     for $j \leftarrow 0$ to $n-1$ do
5:         for $i \leftarrow 0$ to $n-1$ do
6:             if $i \neq j$ then
7:                 $\gamma[j] \leftarrow \gamma[j] \;/\; \big(x_i[j] - x_i[i]\big)$
8:             end if
9:         end for
10:    end for
11:    return $\gamma$
12: end procedure
```

Using the γ values generated above, we construct the Barycentric Form 1 of the Lagrange interpolating polynomial. The following routine accepts the array of weights γ, and an array of the interpolation nodes (x_i, y_i) and evaluation nodes x. The routine outputs an array of the values of the interpolating polynomial, $p_i(x)$, at each of the evaluation nodes x.

Algorithm 3 Barycentric Interpolation

```
1: procedure BARYCENTRIC_INTERPOLATION($x, x_i, \gamma, y_i$)
2:     $n \leftarrow |\text{nodes}|$
3:     $p \leftarrow [0, 0, \ldots, 0]$                ▷ same length as $x$
4:     $w \leftarrow [1, 1, \ldots, 1]$                ▷ same length as $x$
5:     for $k \leftarrow 0$ to $|x| - 1$ do
6:         for $j \leftarrow 0$ to $n-1$ do
7:             $w[k] \leftarrow w[k] \times (x[k] - x_i[j])$
8:         end for
9:         for $i \leftarrow 0$ to $n-1$ do
10:            if isClose$\big(x[k], x_i[i], 10^{-12}\big)$ then
11:                $p[k] \leftarrow y_i[i]$
12:                break
13:            end if
14:            $p[k] \leftarrow p[k] \;+\; \dfrac{w[k] \cdot y\_i[i] \cdot \gamma[i]}{x[k] - x_i[i]}$
15:        end for
16:    end for
17:    return $p$
18: end procedure
```

4.2 Piecewise Interpolating Polynomial Routines

We will construct piecewise interpolating polynomials in the Newton basis. We first develop an algorithm to compute the divided differences for a set of nodes

(x_i, y_i). The function parameters set $f'(x) = None$ by default. If we want to compute the divided differences for a Hermite interpolating polynomial, we pass the first derivative function $f'(x)$ in the algorithm parameters. We store the coefficients (corresponding to the first column of the divided difference table) in an array. The routine outputs the array of coefficients.

Algorithm 4 Newton Divided Difference

```
1: procedure NEWTON_DIVIDED_DIFF($x_i$, $y_i$, $f' = None$)
2:     $n \leftarrow |x_i|$
3:     coeffs $\leftarrow [\, y_i[0] \,]$
4:     for $j \leftarrow 1$ to $n - 1$ do
5:         for $i \leftarrow 0$ to $n - j - 1$ do
6:             if $j = 1$ and $f' \neq$ None and $x_i[i] = x_i[i+1]$ then
7:                 $y_i[i] \leftarrow \frac{f'(x_i[i])}{j!}$
8:             else
9:                 $y_i[i] \leftarrow \frac{y_i[i+1] - y_i[i]}{x_i[i+j] - x_i[i]}$
10:            end if
11:        end for
12:        Append $y[0]$ to coeffs
13:    end for
14:    return (coeffs
15: end procedure
```

The following algorithm constructs a piecewise polynomial. The routine passes the following parameters: the known function $f(x)$, end points of the global interval $[a, b]$, number of sub-intervals M, evaluation nodes x, degree of the interpolating polynomial k, and the "local method" (0 indicates a uniform mesh on each sub-interval, 1 indicates a mesh of Chebyshev nodes of the first kind, 2 indicates a mesh of Chebyshev nodes of the second kind). By default, the parameter sets the arguments *hermite* $= None$, $f'(x) = None$ by default. If we want to construct a piecewise Hermite cubic interpolating polynomial, we pass the arguments *hermite* $= True$ and the first derivative function $f'(x)$.

Algorithm 5 Piecewise Polynomial Interpolation

```
procedure PIECEWISE_POLYNOMIAL($f$, $a$, $b$, $M$, $x$, $k$, $local\_method$, $hermite = None$, $f' = None$)
    $I \leftarrow \text{linspace}(a, b, M+1)$,   $sub\_data \leftarrow [\,]$
    for $i \leftarrow 0$ to $M-1$ do
        $a_i \leftarrow I[i]$,   $b_i \leftarrow I[i+1]$
        if $hermite$ then
            $x_i \leftarrow [\,a_i,\, a_i,\, b_i,\, b_i\,]$
            $y_i \leftarrow [\,f(a_i),\, f(a_i),\, f(b_i),\, f(b_i)\,]$
            $newton\_coeff \leftarrow \text{newton\_divided\_diff}(x_i,\, y_i,\, f')$
        else
            if $k = 1$ then
                $x_i \leftarrow [\,a_i,\, b_i\,]$
            else if $k = 2$ then
                if $local\_method = 0$ or $1$ or $2$ then
                    $x_{mid} \leftarrow (a_i + b_i)/2$
                    $x_i \leftarrow [\,a_i,\, x_{mid},\, b_i\,]$
                end if
            else if $k = 3$ then
                if $local\_method = 0$ then
                    $x_{mid1} \leftarrow a_i + (b_i - a_i)/3$
                    $x_{mid2} \leftarrow a_i + 2(b_i - a_i)/3$
                    $x_i \leftarrow [\,a_i,\, x_{mid1},\, x_{mid2},\, b_i\,]$
                else
                    $x_i \leftarrow \text{chebyshev\_nodes}(3,\, a_i,\, b_i,\, local\_method)$
                end if
            end if
            $y_i \leftarrow [\,f(x)$ for each $x \in x_i\,]$
            $newton\_coeff \leftarrow \text{newton\_divided\_diff}(x_i,\, y_i)$
        end if
        Append the record $\{a_i,\, b_i,\, x_i,\, coeff\}$ to $sub\_data$
    end for
    $x \leftarrow \text{atleast\_1d}(x)$,    $g \leftarrow$ zeros array with same shape as $x$
    for $i \leftarrow 0$ to $|x| - 1$ do
        if $x[i] \le a$ then
            $sub\_int \leftarrow 0$
        else if $x[i] \ge b$ then
            $sub\_int \leftarrow M - 1$
        else
            $sub\_int \leftarrow \text{searchsorted}(I,\, x[i]) - 1$
        end if
        Let $data \leftarrow sub\_data[sub\_int]$
        $g[i] \leftarrow \text{newton\_polynomial}\big(x[i],\, data.x_i,\, data.newton\_coeff\big)$
    end for
    return $g$
end procedure
```

The piecewise polynomial interpolation routine above returns an array of the interpolation function values, $g_k(x)$, for each evaluation node x, evaluated over the sub-intervals containing each evaluation node x.

4.3 Cubic Spline Interpolating Polynomial Routines

To construct the cubic spline interpolating polynomial, we develop a routine to construct and solve the linear system described in (32). The routine accepts the interpolation nodes t_i and the corresponding y_i nodes. We recursively solve for the vector of s_i'' values. This is possible since the matrix is tri-diagonal, so we can simplify the computations to $O(n)$ complexity. The routine outputs the vector, denoted as M.

Algorithm 6 Cubic Spline Coefficients

```
 1: procedure CUBIC_SPLINE_COEFFICIENTS($t_{\text{nodes}}$, $y_{\text{nodes}}$)
 2:     $n \leftarrow \text{length}(t_{\text{nodes}})$
 3:     Initialize array $a[0 \dots n-1] \leftarrow 0$
 4:     $h[i] = t_{\text{nodes}}[i+1] - t_{\text{nodes}}[i]$ for $i = 0, \dots, n-2$ for all $i$
 5:     Initialize arrays $\lambda[0 \dots n-1] \leftarrow 1$, $\mu[0 \dots n-1] \leftarrow 0$, $d[0 \dots n-1] \leftarrow 0$
 6:     for $i \leftarrow 1$ to $n-2$ do
 7:         $a[i] \leftarrow \frac{3}{h[i]}\Big(y_{\text{nodes}}[i+1] - y_{\text{nodes}}[i]\Big) - \frac{3}{h[i-1]}\Big(y_{\text{nodes}}[i] - y_{\text{nodes}}[i-1]\Big)$
 8:     end for
 9:     for $i \leftarrow 1$ to $n-2$ do
10:         $\lambda[i] \leftarrow 2\Big(t_{\text{nodes}}[i+1] - t_{\text{nodes}}[i-1]\Big) - h[i-1] \cdot \mu[i-1]$
11:         $\mu[i] \leftarrow \frac{h[i]}{\lambda[i]}$
12:         $d[i] \leftarrow \frac{a[i] - h[i-1] \cdot d[i-1]}{\lambda[i]}$
13:     end for
14:     Initialize array $M[0 \dots n-1] \leftarrow 0$    ▷ $M$ holds the second derivatives $s_i''$
15:     for $j \leftarrow n-2$ downto 1 do
16:         $M[j] \leftarrow d[j] - \mu[j] \cdot M[j+1]$
17:     end for
18:     return $M$
19: end procedure
```

Once we obtain the array of s_i'' values, we can develop a routine to evaluate the cubic spline polynomial. The routine below accepts the interpolation nodes (t_i, y_i), the evaluation nodes x, and the array of s_i'' values (denoted as s) as the parameters. This routine accepts any set of ordered t_i, meaning the mesh does not need to be uniform for this procedure. The algorithm applies the natural boundary conditions (specified in Section **3.4**), and returns the value of the cubic spline evaluated at each node x, corresponding to the interval of t_i nodes containing each value x.

Algorithm 7 Cubic Spline Polynomial Interpolation

```
1: procedure CUBIC_SPLINE_POLYNOMIAL($t_{\text{nodes}}$, $y_{\text{data}}$, $M$, $x$)
2:     $n \leftarrow \text{length}(t_{\text{nodes}})$
3:     if $x \le t_{\text{nodes}}[0]$ then
4:         $i \leftarrow 0$
5:     else if $x \ge t_{\text{nodes}}[n-1]$ then
6:         $i \leftarrow n-2$
7:     else
8:         $i \leftarrow$ index such that $t_{\text{nodes}}[i] \le x < t_{\text{nodes}}[i+1]$
9:     end if
10:    $h \leftarrow t_{\text{nodes}}[i+1] - t_{\text{nodes}}[i]$
11:    $A \leftarrow \frac{t_{\text{nodes}}[i+1] - x}{h}$
12:    $B \leftarrow \frac{x - t_{\text{nodes}}[i]}{h}$
13:    $y_{\text{val}} \leftarrow A\, y_{\text{data}}[i] + B\, y_{\text{data}}[i+1] + \frac{h^2}{6}\left[(A^3 - A)M[i] + (B^3 - B)M[i+1]\right]$
14:    return $y_{\text{val}}$
15: end procedure
```

In the testing of our routines, we also wish to evaluate the first derivative of the polynomial $p(x)$ at each evaluation node x. We develop a routine that constructs and evaluates equation (26):

Algorithm 8 Cubic Spline First Derivative Evaluation

```
1: procedure CUBIC_SPLINE_PRIME($t_{\text{nodes}}$, $y_{\text{nodes}}$, $M$, $x$)
2:     $n \leftarrow \text{length}(t_{\text{nodes}})$
3:     if $x \le t_{\text{nodes}}[0]$ then
4:         $i \leftarrow 0$
5:     else if $x \ge t_{\text{nodes}}[n-1]$ then
6:         $i \leftarrow n-2$
7:     else
8:         $i \leftarrow$ index such that $t_{\text{nodes}}[i] \le x < t_{\text{nodes}}[i+1]$
9:     end if
10:    $h \leftarrow t_{\text{nodes}}[i+1] - t_{\text{nodes}}[i]$
11:    $A \leftarrow \frac{t_{\text{nodes}}[i+1] - x}{h}$,    $B \leftarrow \frac{x - t_{\text{nodes}}[i]}{h}$
12:    term1 $\leftarrow \frac{y_{\text{nodes}}[i+1] - y_{\text{nodes}}[i]}{h}$
13:    term2 $\leftarrow -\frac{(3A^2 - 1)M[i]\, h}{6}$
14:    term3 $\leftarrow \frac{(3B^2 - 1)M[i+1]\, h}{6}$
15:    return term1 + term2 + term3
16: end procedure
```

Our final routines construct the cubic B-spline $s(x)$ evaluated at fixed nodes x. We first create a routine to the value of each cubic B-spline basis function, $B_i(x)$, at each given point x. The routine passes the particular index i associated with the function, the evaluation node x, the reference point x_i, and the value of the uniform spacing h. The routine uses the formula $x_{i\pm h} = x_i \pm ih$ to define the set of points $x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}$. The routine evaluates the function $B_i(x)$ over the interval $[x_{-2}, x_2]$, given $x \in [x_{-2}, x_2]$, and returns the value of $B_i(x)$.

Algorithm 9 Cubic B-Spline Coefficients

```
1: procedure CUBIC_BSPLINE_COEFFICIENTS($i$, $x$, $x_i$, $h$)
2:     Convert $x$ to a scalar
3:     $x_0 \leftarrow x_i + (i-2)h$
4:     $x_1 \leftarrow x_i + (i-1)h$
5:     $x_2 \leftarrow x_i + ih$
6:     $x_3 \leftarrow x_i + (i+1)h$
7:     $x_4 \leftarrow x_i + (i+2)h$
8:     $B \leftarrow 0$
9:     if $x < x_0$ or $x > x_4$ then
10:        return $B$
11:    end if
12:    if $x_0 \le x < x_1$ then
13:        $B \leftarrow \frac{(x-x_0)^3}{h^3}$
14:        return $B$
15:    end if
16:    if $x_1 \le x < x_2$ then
17:        $B \leftarrow \frac{h^3 + 3h^2(x-x_1) + 3h(x-x_1)^2 - 3(x-x_1)^3}{h^3}$
18:        return $B$
19:    end if
20:    if $x_2 \le x < x_3$ then
21:        $B \leftarrow \frac{h^3 + 3h^2(x_3-x) + 3h(x_3-x)^2 - 3(x_3-x)^3}{h^3}$
22:        return $B$
23:    end if
24:    if $x_3 \le x \le x_4$ then
25:        $B \leftarrow \frac{(x_4-x)^3}{h^3}$
26:        return $B$
27:    end if
28:    return $B$
29: end procedure
```

After obtaining the $B_i(t)$ values, we construct a routine to build the linear system of equations (36) in the form of a matrix, and solve for the vector of a_i

values for each evaluation node x, for $-1 \leq i \leq n+1$. The routine returns the a_i values.

Algorithm 10 Cubic B-Spline Interpolation

```
1: procedure CUBIC_BSPLINE_INTERPOLATION($x_{\text{nodes}}$, $f$, $f'$)
2:     $n \leftarrow \text{length}(x_{\text{nodes}}) - 1$
3:     $h \leftarrow x_{\text{nodes}}[1] - x_{\text{nodes}}[0]$          ▷ Assume uniform spacing
4:     $\xi \leftarrow x_{\text{nodes}}[0]$
5:     $N \leftarrow n + 3$
6:     Initialize matrix $M[0 \ldots N-1,\, 0 \ldots N-1] \leftarrow 0$
7:     Initialize vector $B[0 \ldots N-1] \leftarrow 0$          ▷ Set up first two rows:
8:     $M[0,0] \leftarrow -3/h$, $M[0,2] \leftarrow 3/h$
9:     $M[1,0] \leftarrow 1$, $M[1,1] \leftarrow 4$, $M[1,2] \leftarrow 1$          ▷ Set up last two rows:
10:    $M[n+1,n] \leftarrow -3/h$, $M[n+1,n+2] \leftarrow 3/h$
11:    $M[n+2,n] \leftarrow 1$, $M[n+2,n+1] \leftarrow 4$, $M[n+2,n+2] \leftarrow 1$
12:    $B[0] \leftarrow f'(x_{\text{nodes}}[0])$, $B[1] \leftarrow f(x_{\text{nodes}}[0])$
13:    $B[n+1] \leftarrow f'(x_{\text{nodes}}[n])$, $B[n+2] \leftarrow f(x_{\text{nodes}}[n])$
14:    for $i \leftarrow 1$ to $n-1$ do
15:        $row \leftarrow i + 1$
16:        $M[row,i] \leftarrow 1$, $M[row,i+1] \leftarrow 4$, $M[row,i+2] \leftarrow 1$
17:        $B[row] \leftarrow f(x_{\text{nodes}}[i])$
18:    end for
19:    Solve linear system: $a \leftarrow M^{-1}B$
20:    return $a$
21: end procedure
```

To solve for the interpolating polynomial, $p(x)$, we compute the inner product of the a_i and B_i vectors (35) (i.e. sum the elements of corresponding indices).

5 Description of the Experimental Design and Results

5.1 Task 1: Testing Various Functions f(x)

5.1.1 Testing $\mathbf{f(x) = x^3}$

To analyze the interpolatory behavior of the polynomials constructed from the above routines, we carefully select several different functions $f(x)$ to test. Since we are testing functions for which the results are known, we are able to compare the approximation error in the interpolation to the true value of the function $f(x)$ evaluated at each node x. For each interpolation, several global intervals $[a, b]$ are tested and a dense set of $n = 100$ uniform nodes (evaluation nodes) are generated on each interval, which will be used to evaluate the interpolating polynomial. For each polynomial error, we consider the maximum error in the interpolation and compare the results. Note that all functions tested with the piecewise interpolating methods will use Chebyshev nodes of the second kind for the local mesh points.

First, consider the following simple, cubic polynomial:

$$f(x) = x^3.$$

We can easily differentiate this polynomial to find the first and second derivative functions, which will be used to construct the cubic Hermite interpolating polynomial and the cubic spline polynomials, respectively. The differentiated functions are

$$f'(x) = 3x^2,$$
$$f''(x) = 6x.$$

Examine the behavior of a single cubic interpolating polynomial in the Lagrange basis, for varying degrees of interpolation n:

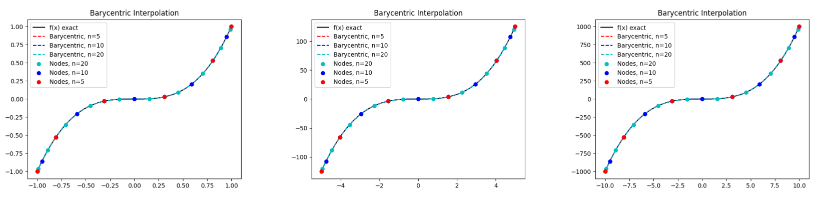

Figure 2: Intervals $[-1, 1]$, $[-5, 5]$, $[-10, 10]$

The Barycentric Form 1 of the Lagrange interpolating polynomial on each interval, for each degree of interpolation, appears to interpolate the function $f(x)$ with strong accuracy.

We now examine the performance of the cubic piecewise interpolating polynomial (left) compared to the cubic piecewise Hermite interpolating polynomial (right) on the same intervals, for $M = 20$ sub-intervals that will partition each chosen global interval. One each sub-interval, we generate Chebyshev nodes of the second kind for the interpolation:

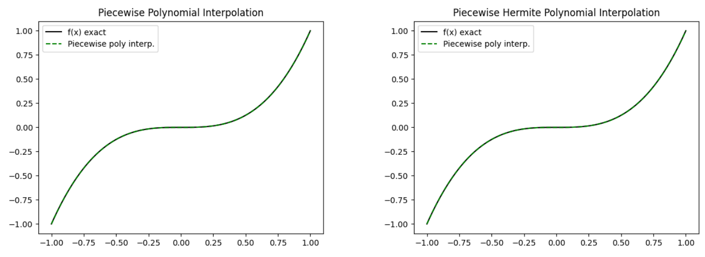

Figure 3: Interval $[-1, 1]$, $M = 20$

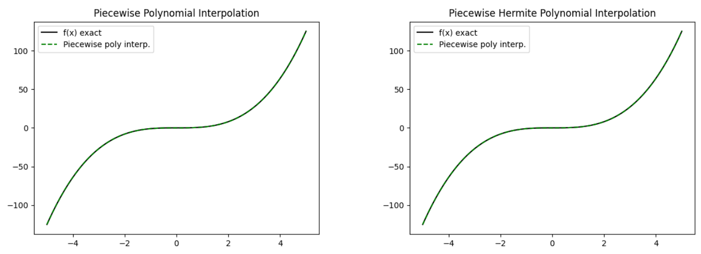

Figure 4: Interval $[-5, 5]$, $M = 20$

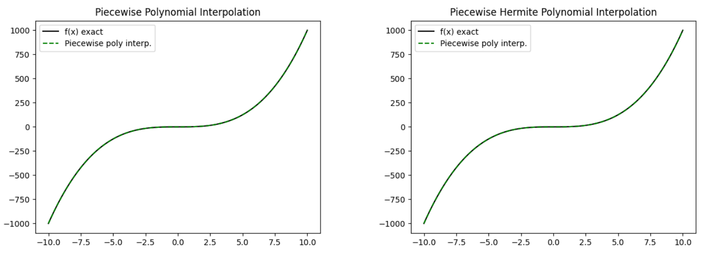

Figure 5: Interval $[-10, 10]$, $M = 20$

From the plots above, we do not visually observe a difference in the interpolation. We turn to the maximum interpolation error on each interval:

Interval	Cubic Piecewise	Cubic Hermite Piecewise
$[-1, 1]$	$1.11022302 * 10^{-16}$	$1.66533453 * 10^{-16}$
$[-5, 5]$	$1.42108547 * 10^{-14}$	$1.42108547 * 10^{-14}$
$[-10, 10]$	$1.13686837 * 10^{-13}$	$1.13686837 * 10^{-13}$

Table 1: Maximum Piecewise Approximation Error $f(x) = x^3$

As the interval size increases, but the number of sub-intervals remains the same, we see that the error is approximately equivalent for both the cubic and cubic Hermite piecewise interpolating polynomials. On the smallest interval tested, $[-1, 1]$, the cubic Hermite polynomial has marginally less maximum approximation error in the interpolation.
Lastly, for our chosen function $f(x) = x^3$, we also compare the cubic spline interpolation (left) and cubic B-spline polynomial interpolation (right) accuracies on the same chosen intervals, for varying numbers of interpolation nodes n. Observe the following plots:

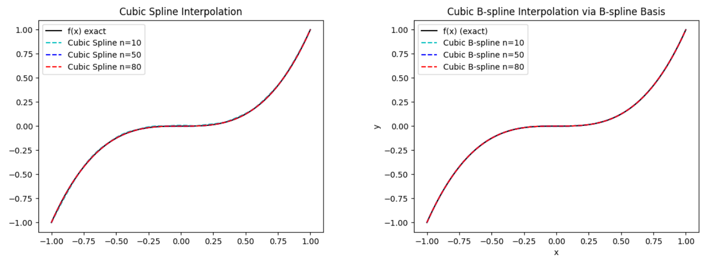

Figure 6: Interval $[-1, 1]$

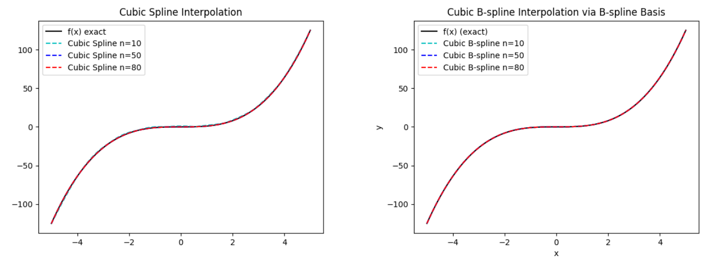

Figure 7: Interval $[-5, 5]$

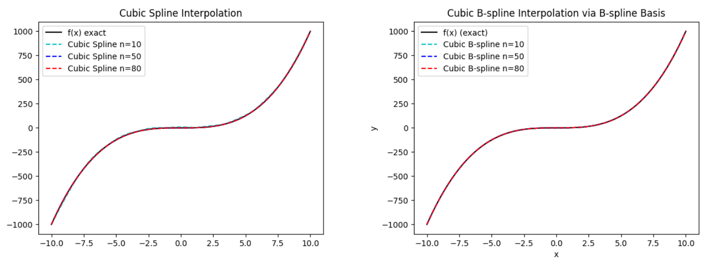

Figure 8: Interval $[-10, 10]$

From the plots above, we can visually observe a difference in the interpolation, where the cubic B-spline achieves much better accuracy in the interpolation. The the cubic spline interpolating polynomial results in several oscillations, which are larger in range as the intervals increase. We examine the maximum interpolation error on each interval, for both interpolation methods:

Interval	Interpolation Nodes n	Cubic Spline	Cubic B-Spline
$[-1, 1]$	10	0.01523654	$6.66133815 * 10^{-16}$
$[-1, 1]$	50	0.00060137	$6.66133815 * 10^{-15}$
$[-1, 1]$	80	0.00011949	$8.65973959 * 10^{-15}$
$[-5, 5]$	10	1.90456756	$5.68434189 * 10^{-14}$
$[-5, 5]$	50	0.07517101	$8.10018719 * 10^{-13}$
$[-5, 5]$	80	0.0149365	$1.73372428 * 10^{-12}$
$[-10, 10]$	10	15.23654049	$4.54747351 * 10^{-13}$
$[-10, 10]$	50	0.60136811	$6.4801498 * 10^{-12}$
$[-10, 10]$	80	0.11949199	$1.3869794 * 10^{-11}$

Table 2: Maximum Spline Approximation Error $f(x) = x^3$

In the table above, we notice that for the cubic spline interpolation, the maximum approximation error increases with each interval, but decreases as we increase n, the number of interpolation nodes. We also note that the cubic B-spline produces a significantly better approximation compared to the alternative cubic spline method.

5.1.2 Testing f(x) = 1 / (1 + x²)

We examine the interpolatory behavior of the Runge polynomial $f(x) = \frac{1}{1+10x^2}$, as discussed in the introduction (Section **1**) of this report. We will use cubic interpolation to approximate this quadratic rational function. The first and second order derivatives of $f(x)$ are

$$f'(x) = -\frac{20x}{(1+10x^2)^2},$$
$$f''(x) = -\frac{20(1-30x^2)}{(1+10x^2)^3}.$$

As the second order derivative is a more complex rational function, we may expect to see some errors in the interpolation. We again examine the interpolatory behaviors on several global interval sizes, where we generate $n = 100$ evaluation nodes for testing the accuracy of the interpolation. We compare the visual results produced in the plots, and summarize the maximum approximation errors for each interpolation method. We first examine the performance of the Barycentric Form 1 of the Lagrange interpolating polynomial for varying intervals, and varying numbers of interpolation nodes ($n = 5$, $n = 10$, $n = 20$).

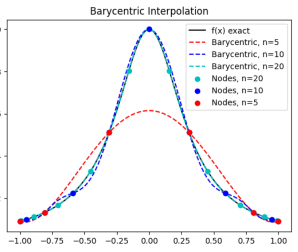

Figure 9: Interval $[-1, 1]$

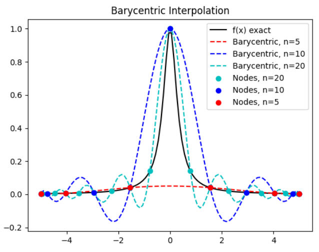

Figure 10: Interval $[-5, 5]$

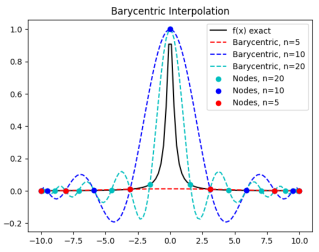

Figure 11: Interval $[-10, 10]$

The above plots confirm the the Runge phenomenon, as we see more oscillations closer to the end points of the interval, for higher degrees of interpolation. We also compare the performance of the cubic piecewise polynomial (left) and the cubic piecewise Hermite interpolating polynomial (right), for $M = 20$ subintervals:

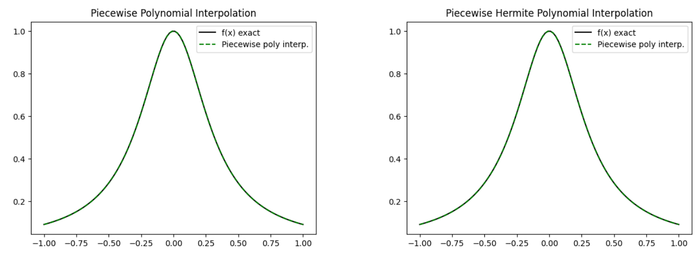

Figure 12: Interval $[-1, 1]$, $M = 20$

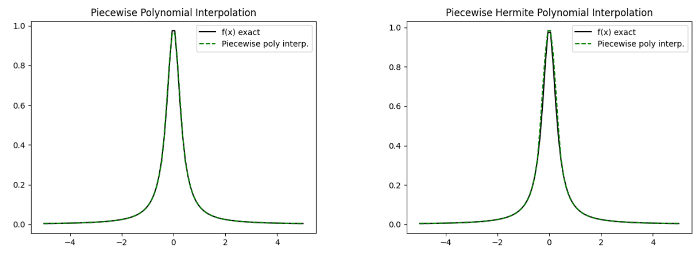

Figure 13: Interval $[-5, 5]$, $M = 20$

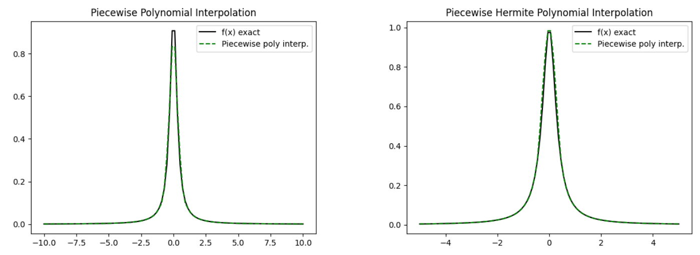

Figure 14: Interval $[-5, 5]$, $M = 20$

Interval	Cubic Piecewise	Cubic Hermite Piecewise
$[-1, 1]$	0.00010175	0.00040235
$[-5, 5]$	0.02388304	0.07836042
$[-10, 10]$	0.07401699	0.28943192

Table 3: Maximum Piecewise Approximation Error $f(x) = \frac{1}{1+10x^2}$

In the piecewise interpolation plots above, we see some interpolation error occurring near the midpoint of the interval, where $x = 0$. We know that $f(0) =$

$\frac{1}{1+0} = 1$, meaning the interpolating function $p(x)$ at $x = 0$ should have a value of $p(0) \approx 1$. In the table, we also notice that the maximum approximation error for the cubic Hermite piecewise interpolation is higher than for the cubic piecewise interpolation, for each interval tested.
Lastly, we again compare the cubic spline interpolation (left) and cubic B-spline interpolation (right) methods. Within each plot, we also examine the differences in the interpolation for varying numbers of initial interpolation nodes ($n = 10$, $n = 50$, $n = 150$, $n = 80$):

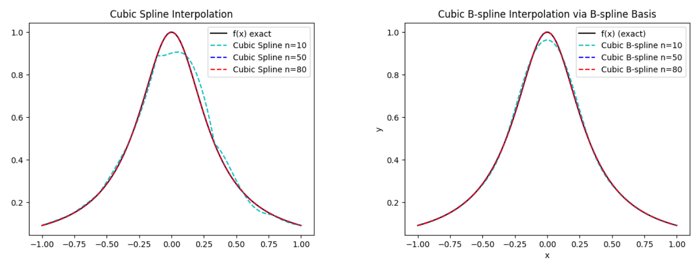

Figure 15: Interval $[-1, 1]$

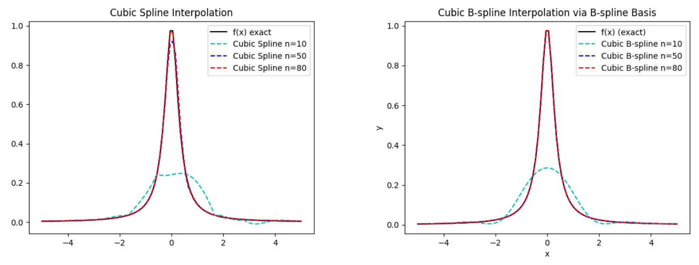

Figure 16: Interval $[-5, 5]$

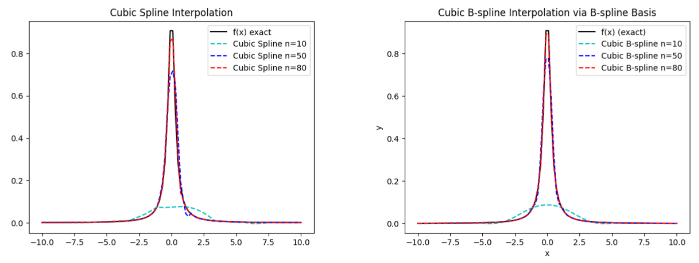

Figure 17: Interval $[-10, 10]$

Interval	Interpolation Nodes n	Cubic Spline	Cubic B-Spline
$[-1,1]$	10	0.09999227	0.03680164
$[-1,1]$	50	0.00099463	$1.167 * 10^{-5}$
$[-1,1]$	80	0.00026843	$1.82 * 10^{-5}$
$[-5,5]$	10	0.73516832	0.68965694
$[-5,5]$	50	0.06691395	0.01895045
$[-5,5]$	80	0.02321064	0.00105509
$[-10,10]$	10	0.83410522	0.81961181
$[-10,10]$	50	0.21410292	0.13088337
$[-10,10]$	80	0.0934866	0.02414161

Table 4: Maximum Spline Approximation Error $f(x) = \frac{1}{1+10x^2}$

From the plots above, we notice that for all the intervals, a higher degree of interpolation improves the accuracy of the interpolation. This difference is particular visible in the plots for the larger intervals $[-5,5]$ and $[-10,10]$. This confirms that a higher degree of interpolation improves the interpolation accuracy for both spline methods tested. For each interval and each number of initial interpolation nodes, the cubic B-spline constructs a better approximation to the true function.

5.1.3 Testing $\mathbf{f(x) = sin(x)}$

We now test the behavior of cubic polynomial interpolation on the non-cubic trigonometric function $f(x) = sin(x)$. For this function, the first and second order derivatives of $f(x)$ are

$$\begin{aligned} f'(x) &= cos(x), \\ f''(x) &= -sin(x). \end{aligned}$$

For each interpolation, we examine the interpolatory behavior on varying global intervals $[a, b]$. On each interval, we construct a set of $n = 100$ evaluation nodes for testing the accuracy of the interpolating polynomials. We will again consider and compare the results of the interpolation and maximum approximation error. We first compare the performance of the cubic piecewise polynomial (left) and the cubic piecewise Hermite interpolating polynomial (right), for $M = 20$ subintervals:

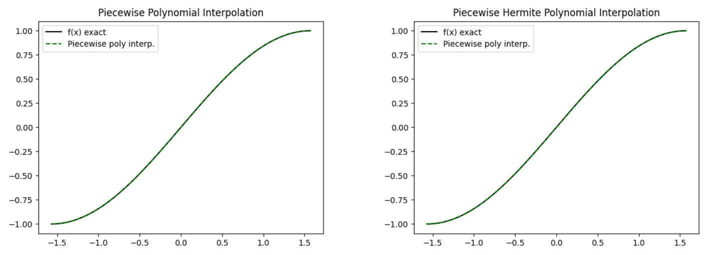

Figure 18: Interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$, $M = 20$

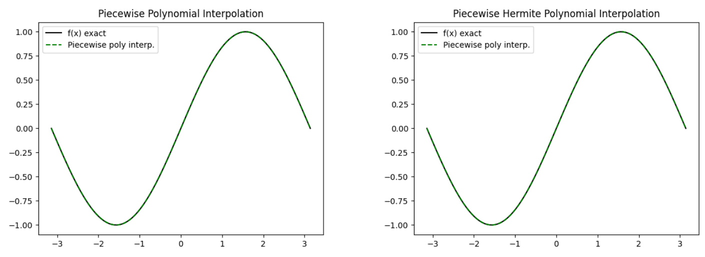

Figure 19: Interval $[-\pi, \pi]$, $M = 20$

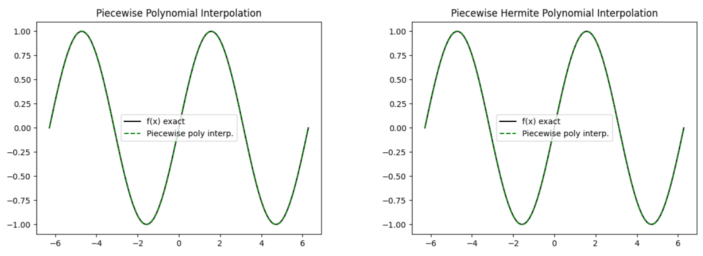

Figure 20: Interval $[-2\pi, 2\pi]$, $M = 20$

Interval	Cubic Piecewise	Cubic Hermite Piecewise
$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$3.3 * 10^{-7}$	$1.47 * 10^{-6}$
$[-\pi, \pi]$	$6 * 10^{-6}$	$2.461 * 10^{-5}$
$[-2\pi, 2\pi]$	$9.976 * 10^{-5}$	0.00040115

Table 5: Maximum Piecewise Approximation Error $f(x) = sin(x)$

From the plots and approximation error table above, we see that both piecewise methods produce closely accurate approximations to the true function $f(x)$. For this function, we notice that the cubic piecewise interpolation produces a better approximation than the Hermite cubic interpolation. This could be due to the fact that the Hermite interpolation requires us to input $f'(x)$ into the function parameters. Unlike the differentiation of a polynomial, the first derivative of this trigonometric function behaves like a "shift" to the graph, which may result in slight errors to arise in the approximation. This may explain why the maximum approximation error is slightly higher for the cubic Hermite piecewise interpolating polynomial.
We now compare the behaviors of the cubic spline (left) interpolation and cubic B-spline (right) interpolation methods for the chosen function $f(x)$, for varying sizes n of interpolation nodes:

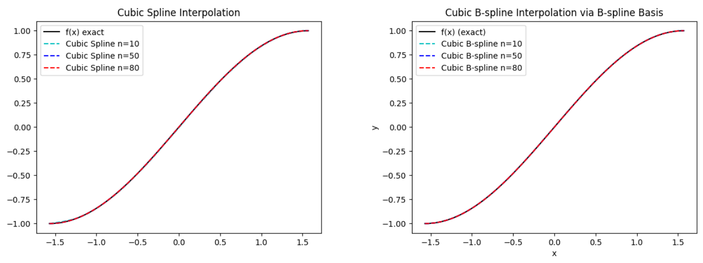

Figure 21: Interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$

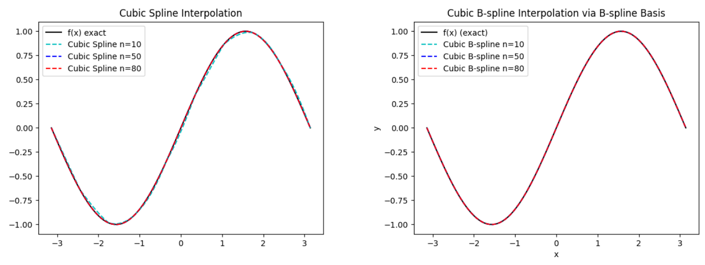

Figure 22: Interval $[-\pi, \pi]$

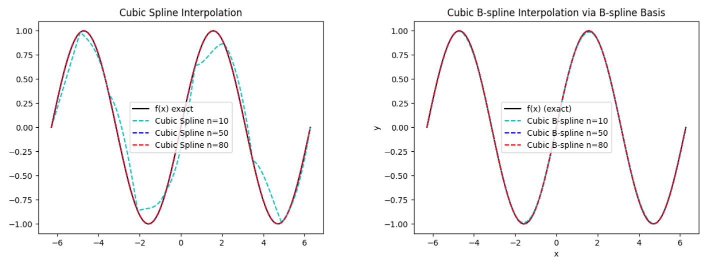

Figure 23: Interval $[-2\pi, 2\pi]$

Interval	Interpolation Nodes n	Cubic Spline	Cubic B-Spline
$[-\frac{\pi}{2}, \frac{\pi}{2}]$	10	0.0075277	$3.87 * 10^{-5}$
$[-\frac{\pi}{2}, \frac{\pi}{2}]$	50	0.00025745	$4 * 10^{-8}$
$[-\frac{\pi}{2}, \frac{\pi}{2}]$	80	$5.103 * 10^{-5}$	$1 * 10^{-8}$
$[-\pi, \pi]$	10	0.03860667	0.00066839
$[-\pi, \pi]$	50	0.00026096	$6.3 * 10^{-7}$
$[-\pi, \pi]$	80	$6.242 * 10^{-5}$	$1 * 10^{-7}$
$[-2\pi, 2\pi]$	10	0.27763567	0.01508764
$[-2\pi, 2\pi]$	50	0.00203139	$1.105 * 10^{-5}$
$[-2\pi, 2\pi]$	80	0.00048833	$1.67 * 10^{-5}$

Table 6: Maximum Spline Approximation Error $f(x) = sin(x)$

For the larger intervals $[-\pi, \pi]$ and $[-2\pi, 2\pi]$, the cubic B-spline produces a better approximation to the true function $f(x)$. For the largest interval tested, $[-2\pi, 2\pi]$, the cubic spline interpolation results in visible approximation errors when a smaller degree of nodes ($n = 10$) is tested. The above error table confirms the visual results. For each interval and value of n tested, the cubic B-spline produces smaller maximum approximation errors compared to the cubic spline interpolation. For each interval tested, both methods improve in their approximation accuracy when the number of interpolation nodes is increased.

5.1.4 Testing Randomly Generated Polynomials

$\mathbf{f(x) = ax^3 + bx^2 + cx + d}$

For the following tests, we randomly generate floating point coefficients a, b, c, d from a Normal distribution $N \sim (0, 1)$ and scale by 10 to construct several cubic polynomials of the form $f(x) = ax^3 + bx^2 + cx + d$. Since these are standard polynomials, the first and second derivatives of these functions can easily be found by taking

$$f'(x) = 3ax^2 + 2bx + x,$$
$$f''(x) = 6ax + 2b.$$

We generate $n = 20$ Chebyshev nodes for testing, and examine the maximum approximation errors in the cubic piecewise, cubic Hermite piecewise, cubic spline, and cubic B-spline interpolations. We test each of the four interpolation methods listed for each randomly generated polynomial. For the piecewise interpolating polynomials, we set the number of sub-intervals to 10 and continue to choose Chebyshev nodes of the second kind for the local (sub-interval) mesh points. The global interval chosen for the experiments is $[-10, 10]$. We evaluate the functions for a set of 100 uniformly spaced points x on the interval $[-10, 10]$. We tabulate the results below:

Test	Coefficients	Piecewise Error	Piecewise Hermite Error	Spline Error	BSpline Error
1	[-2.212, 18.045, 2.044, -8.188]	4.547e-13	4.547e-13	6.815e+05	4.209e+03
2	[3.145, -1.679, -0.564, 0.34]	4.547e-13	4.547e-13	1.651e+06	5.430e+03
3	[-9.768, -4.609, -3.423, -3.822]	4.547e-13	4.547e-13	5.233e+05	2.629e+03
4	[-9.739, 6.601, -3.576, -1.887]	9.095e-13	9.095e-13	2.858e+06	7.985e+03
5	[8.843, 9.694, -0.839, 4.676]	1.819e-12	1.819e-12	4.730e+06	1.362e+04
6	[-0.302, 6.84, 2.777, -0.812]	6.821e-13	4.547e-13	1.263e+06	3.586e+03
7	[2.149, -4.52, 3.178, 10.8]	1.819e-12	1.819e-12	3.875e+06	1.088e+04
8	[-4.503, -2.028, -2.306, 6.515]	3.638e-12	3.638e-12	7.124e+06	2.114e+04
9	[2.36, -9.21, 2.734, 17.01]	1.819e-12	3.638e-12	5.376e+06	1.583e+04
10	[-8.227, 23.09, -8.454, 15.816]	1.819e-12	1.819e-12	5.266e+06	1.615e+04
11	[8.945, -4.161, 11.449, -14.956]	1.819e-12	1.819e-12	5.612e+06	1.690e+04
12	[-6.81, 0.188, -7.02, 15.115]	1.819e-12	3.638e-12	6.176e+06	1.719e+04
13	[12.257, 3.181, -10.497, 20.069]	9.095e-13	9.095e-13	2.326e+06	6.730e+03
14	[20.601, -2.207, -2.215, -10.656]	1.819e-12	1.819e-12	3.939e+06	1.107e+04
15	[1.45, 4.375, -2.01, -3.43]	1.137e-13	1.137e-13	2.602e+05	1.021e+03
16	[-20.772, -22.072, -4.18, -12.959]	9.095e-13	9.095e-13	1.679e+06	4.848e+03
17	[9.186, -5.33, 0.067, -7.916]	4.547e-13	2.274e-13	7.243e+05	2.189e+03
18	[-11.674, 5.87, -8.343, 1.724]	4.547e-13	9.095e-13	1.421e+06	4.220e+03
19	[-20.042, -8.728, 14.406, 4.096]	5.684e-14	5.684e-14	1.160e+05	3.763e+02
20	[-15.567, -7.905, 16.99, -2.447]	1.819e-12	1.819e-12	2.933e+06	8.210e+03

Table 7: Maximum Interpolation Error Comparison for Randomly Generated Cubic Polynomials f(x)

For all tests performed, the above table suggests the cubic piecewise interpolation methods better approximate the true polynomial $f(x)$, in comparison to the cubic spline interpolation methods, which have higher maximum approximation error for each randomly generated polynomial $f(x)$. We also notice that for all the cubic polynomials, the maximum approximation errors for cubic piecewise and cubic Hermite piecewise interpolating polynomials are equal.

5.2 Task 2: Estimating Functions y(t), f(t), D(t)

Suppose we have functions $y(t)$, $f(t)$, $D(t)$ related by the following equation:

$$D(t) = e^{-ty(t)} = e^{-\int_0^t f(\tau)d\tau}. \tag{37}$$

Since $-\int_0^t f(\tau)d\tau = -ty(t)$, we can differentiate to get

$$\begin{aligned} \frac{d}{dt}(-\int_0^t f(\tau)d\tau) &= \frac{d}{dt}(-ty(t)) \\ -f(t) + f(0) &= -y(t) - ty'(t). \end{aligned}$$

Let $f(0) = 0$. Then the above equation becomes

$$f(t) = y(t) + ty'(t). \tag{38}$$

Now suppose we have available the following set of nonuniform discrete values (t_i, y_i), for $0 \leq i \leq n$ and $y(t_i) = y_i$, rather than a continuous function, which are given to be

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

Table 8: Discrete Nodes (t_i, y_i)

We use a natural boundary condition to construct an interpolatory cubic spline, $s(t)$, based on the data above, to estimate the functions $y(t)$, $f(t)$, and $D(t)$. A set of evaluation nodes from $t_1 = 0.5$ to $t_{40} = 20$ are generated to evaluate the interpolating polynomial, with each value incremented by $\Delta t = 0.5$. We also want to observe the behavior of piecewise interpolation on the set of points. Since our piecewise routine requires the true function $y(t)$ to be input into the routine parameters, we need to construct a function for these points. The Python NumPy library package ”interp” can be used to develop a piecewise linear function that passes through the given set of discrete nodes. We pass this function into the piecewise interpolation routine parameters. Since this is a linear interpolant, we set the degree of the piecewise interpolation to $k = 1$, and construct a linear piecewise interpolating polynomial $g_1(x)$. The plots for the interpolated values of the functions $y(t)$ and the computed values of the functions $f(t)$ and $D(t)$ are given below.

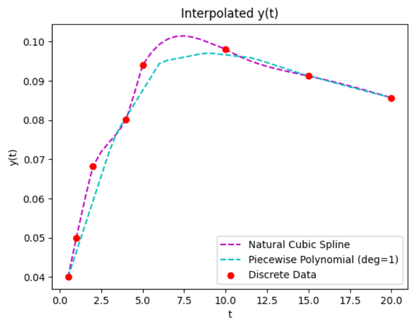

Figure 24: $y(t)$

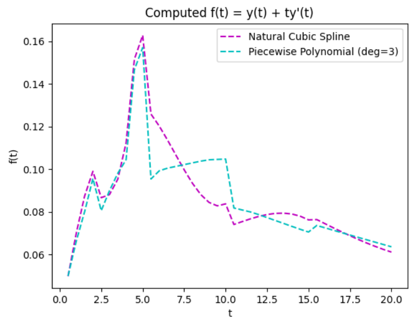

Figure 25: $f(t)$

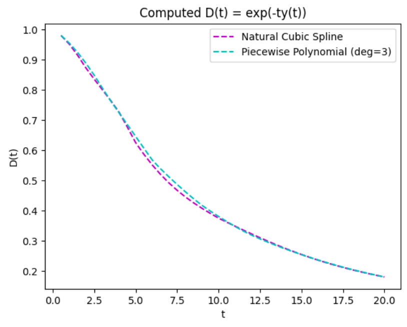

Figure 26: $D(t)$

Visually, both interpolation strategies produce similar results for each of the plots observed. Looking at the first plot, for $y(t)$, we notice that the cubic spline polynomial $s(t)$ behaves similarly to the linear interpolating polynomial $g_1(t)$.

The values of $y(t_i)$ at the interpolation nodes t_i are monotonically increasing up until $t_i = 10.0$, after which the values begin to decrease. As such, the interpolating polynomials are increasing up until $t = 10.0$, after which the polynomials exhibit a downward trend. The second plot, for $f(t)$, plots the sum of the interpolated function $f(t)$ and the first derivative of the function $f'(t)$, scaled by t. The first derivative of the cubic spline interpolation, $s'(t)$, is computed using the "Cubic Spline First Derivative Evaluation" routine discussed in Section **4** of this report. Since $s(t)$ is a cubic polynomial, we expect $s'(t)$ to behave as a quadratic polynomial. To find the first order derivative of $g_1(x)$, we computed the slope between every interpolation node, i.e. $slope = \frac{y(t_{i+1})-y(t_i)}{t_{i+1}-t_i}$. We located each of the evaluation nodes such that $t \in [t_i, t_{i+1}]$, and plotted their values according to their approximate slope. Since the slope of $y(t)$ has a steep upward trend up to $t = 5.0$, we see the graph of $f(t)$ increase up $t = 5.0$. Since the slope between points $t = 5.0$ to $t = 10.0$ is nearly zero, the value of $f(t) \approx y(t)$ on the interval $[5.0, 10.0]$. After node $t = 10.0$, the value of $y(t)$ decreases, hence the slope of $y(t)$ becomes negative. Since $f(t)$ contains the term $tf'(t)$, this means that the negative slope is scaled by larger and increasing values of t. This explains why the sum of the constructed functions begin to experience downward behavior towards the end of the plot. Lastly, we view the plot of $D(t)$. The function $D(t)$ only requires the interpolating polynomials for $y(t)$ as a parameter. Since the cubic spline and linear piecewise polynomials produced similar approximations for $y(t)$, we expect similar behaviors such that $D(t) \approx e^{-ts(t)} \approx e^{-tg_1(t)}$. This is confirmed in the third plot above. Since $f(0) = 0$, we have $D(0) = e^0 = 1$. We know $y(t) > 0$ for all t, hence the plot of $D(t)$ decreases exponentially, approaching the value 0 as t increases.

6 Conclusion

This report discussed the mathematical derivation of several interpolation methods. We chose specific functions to examine the behavior of the Barycentric Form 1 of Lagrange interpolating polynomials, cubic piecewise and cubic piecewise Hermite interpolating polynomials, and cubic spline and B-spline interpolating polynomials. After conducting several tests, we examined the possible oscilatory behavior that can arise from Lagrange interpolation, which was eliminated when the piecewise and spline methods were applied. For certain functions, the cubic piecewise interpolating polynomials resulted in smaller maximum approximation errors, compared to the cubic spline interpolating polynomials, and vice versa. The piecewise and spline interpolating polynomials are capable of being solved in $O(n)$ complexity, since a linear system can be constructed and solved recursively using forward or backward recursion. The research conducted in this report can be extended further, by examining a larger variety of functions, with degrees $n \leq 2$ or $n \geq 4$. Further, for Task 2, we can explore a stronger method for constructing the piecewise polynomial $g_d(t)$ for a given discrete set of nodes.

References

[1] Alfio Quarteroni and Fausto Saleri, “Numerical Methods for Scientists and Engineers,” *Springer*, 2000. `https://link.springer.com/book/10.1007/978-3-662-04166-0`.

[2] M. B. Rivas, *On the Convergence of Some Cubic Spline Interpolation Schemes*, *SIAM Journal on Numerical Analysis*, Vol. 23, No. 4 (Aug. 1986), pp. 903–912.

[3] Dr. Kyle A. Gallivan, “Set 14: Hermite Interpolation and Osculatory Polynomials”, Class Lecture, MAD5403: Foundations of Computational Mathematics II, Florida State University.

[4] Dr. Kyle A. Gallivan, “Set 15: Piecewise Polynomial Interpolation”, Class Lecture, MAD5403: Foundations of Computational Mathematics II, Florida State University.

[5] Dr. Kyle A. Gallivan, “Set 16: Piecewise Polynomial Interpolation Bases”, Class Lecture, MAD5403: Foundations of Computational Mathematics II, Florida State University.

[6] Dr. Kyle A. Gallivan, “Set 17: Polynomial Splines Part 1”, Class Lecture, MAD5403: Foundations of Computational Mathematics II, Florida State University.

[7] Dr. Kyle A. Gallivan, “Set 18: Polynomial Splines Part 2”, Class Lecture, MAD5403: Foundations of Computational Mathematics II, Florida State University.

[8] “Polynomial Interpolation,” *Wikipedia, The Free Encyclopedia*, `https://en.wikipedia.org/wiki/Polynomial_interpolation`.

[9] “Hermite Polynomial,” *Wikipedia, The Free Encyclopedia*, `https://en.wikipedia.org/wiki/Hermite_interpolation`.

[10] ‘Spline Interpolation,” *Wikipedia, The Free Encyclopedia*, `https://en.wikipedia.org/wiki/Spline_interpolation`.

[11] ‘Cubic Spline Interpolation,” *Wikipedia, The Free Encyclopedia*, `https://en.wikipedia.org/wiki/Cubic_spline_interpolation`.

[12] ‘Runge’s Phenomenon,” *Wikipedia, The Free Encyclopedia*, `https://en.wikipedia.org/wiki/Runges_phenomenon`.

[13] ‘Cubic Spline Interpolation,” *Wikiversity*, `https://en.wikiversity.org/wiki/Cubic_Spline_Interpolation`.