MA3105 Final Project
Cubic Spline Interpolation

Shreya Ganguly (22MS110)
Abhratanu Ray (22MS052)
Niravra Nag (22MS072)

Autumn, 2024

Introduction

What is interpolation?

Given a set of $n+1$ points $\{x_0, x_1, ..., x_n\}$ at which the values of a function $f(x_0) = y_0, f(x_1) = y_1, ..., f(x_n) = y_n$ are known, interpolation is the process of finding the function that satisfies the condition $f(x_i) = y_i \forall i \in \{0, 1, 2, ..., n\}$.

Introduction

Polynomial Interpolation

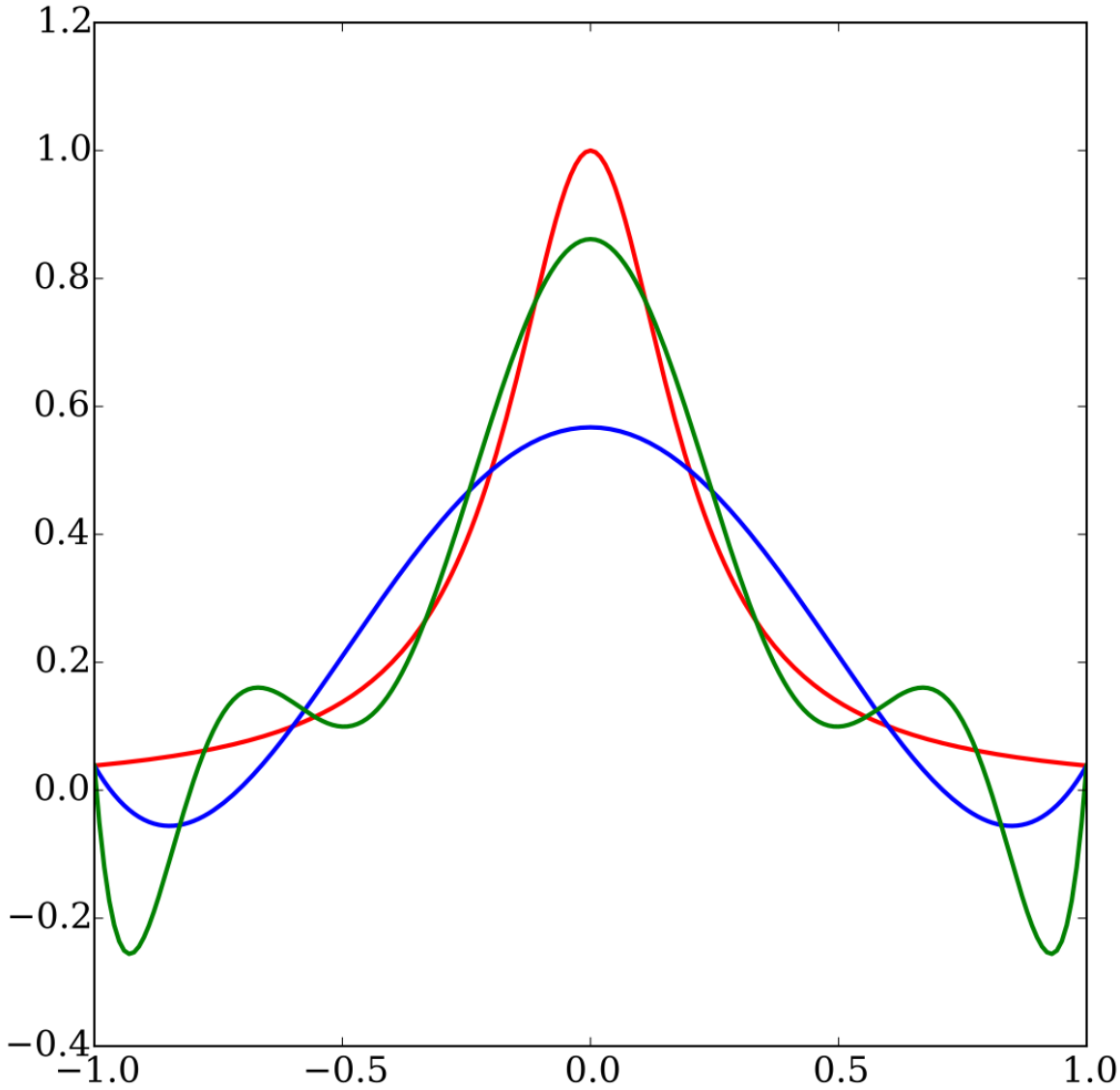
Polynomial interpolation involves fitting a single polynomial of degree n to a given set of $n+1$ data points. While this approach ensures that the polynomial passes through all the given points, it has notable disadvantages when used for a large number of points.

Introduction

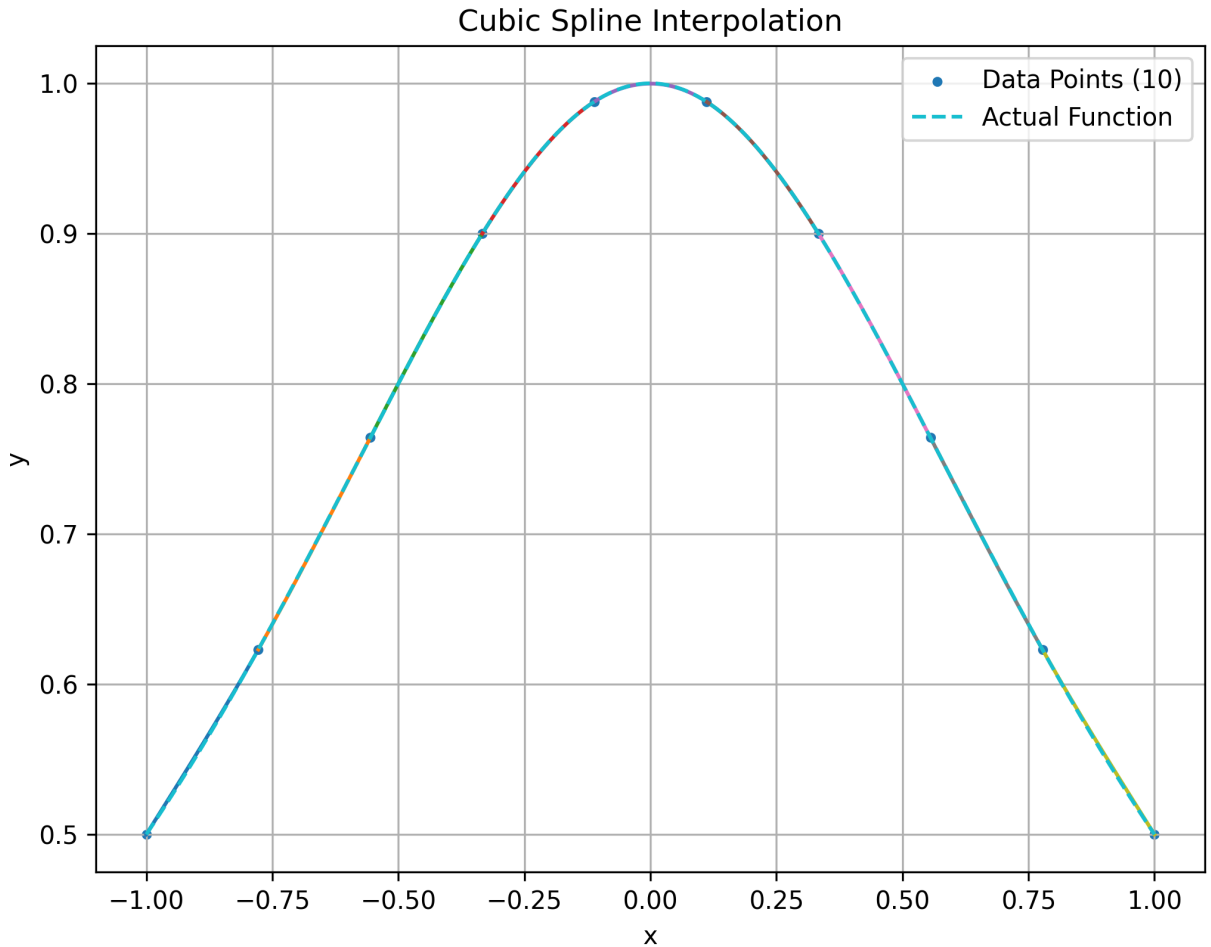
A primary concern is the occurrence of **Runge's phenomenon**, which is characterized by large oscillations near the endpoints of the interpolation interval, particularly when equally spaced nodes are used. This behavior is caused by the high degree of the polynomial. Consequently, the error grows significantly, making the interpolation unreliable.

Additionally, the computational cost of evaluating and differentiating high-degree polynomials can be prohibitive. These drawbacks highlight the need for an alternative approach that balances accuracy and computational efficiency, such as *piecewise interpolation*.

Introduction



Via Cubic Spline Interpolations



Introduction

Piecewise Interpolation

Piecewise interpolation addresses the limitations of polynomial interpolation by dividing the data into smaller intervals and fitting simpler functions, such as linear or quadratic polynomials, within each interval. Piecewise interpolation avoids Runge's phenomenon because it uses low-degree polynomials over small intervals. Furthermore, the use of lower-degree polynomials reduces computational complexity and improves numerical stability.

Introduction

Linear Piecewise Interpolation

Linear interpolation fits a straight line between two consecutive data points. For two points (x_i, y_i) and (x_{i+1}, y_{i+1}), the linear interpolant is:

$$L_i(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i), \quad x \in [x_i, x_{i+1}].$$

Quadratic Piecewise Interpolation

Piecewise quadratic interpolation divides the entire domain into subintervals, where each subinterval spans three consecutive points. A unique quadratic polynomial is constructed for each subinterval using the Lagrange formula. Let the function be interpolated over a domain divided into n points, where n is a multiple of 3.

Quadratic Piecewise Interpolation

For any three consecutive points (x_{i-1}, y_{i-1}), (x_i, y_i), and (x_{i+1}, y_{i+1}), the Lagrange polynomial is expressed as:

$$Q_i(x) = y_{i-1} \cdot \ell_{i-1}(x) + y_i \cdot \ell_i(x) + y_{i+1} \cdot \ell_{i+1}(x),$$

where $\ell_{i-1}(x)$, $\ell_i(x)$, and $\ell_{i+1}(x)$ are the Lagrange basis polynomials defined as:

$$\ell_{i-1}(x) = \frac{(x - x_i)(x - x_{i+1})}{(x_{i-1} - x_i)(x_{i-1} - x_{i+1})},$$

$$\ell_i(x) = \frac{(x - x_{i-1})(x - x_{i+1})}{(x_i - x_{i-1})(x_i - x_{i+1})},$$

$$\ell_{i+1}(x) = \frac{(x - x_{i-1})(x - x_i)}{(x_{i+1} - x_{i-1})(x_{i+1} - x_i)}.$$

Quadratic Interpolation: Deriving the Coefficients

The resulting interpolating polynomial $Q_i(x)$ satisfies the following conditions:

- $Q_i(x_{i-1}) = y_{i-1}$,
- $Q_i(x_i) = y_i$,
- $Q_i(x_{i+1}) = y_{i+1}$.

Advantages of Quadratic Interpolation

- Provides smoother approximations compared to linear interpolation.
- Ensures continuity of the first derivative across intervals.

Cubic Spline Construction

Given $n+2$ points (x_i, y_i), the spline is:

$$S(x) = P_i(x), \quad x \in [x_i, x_{i+1}],\ i = 0, 1, \ldots, n$$

Conditions:

1. $P_i(x_i) = y_i \qquad i = 0, 1, 2, ..., n$
2. $P_{i-1}(x_i) = P_i(x_i) \qquad i = 0, 1, 2, ..., n$
3. $P'_{i-1}(x_i) = P'_i(x_i) \qquad i = 0, 1, 2, ..., n$
4. $P''_{i-1}(x_i) = P''_i(x_i) \qquad i = 0, 1, 2, ..., n$

Boundary Conditions

Approaches for boundary conditions:

1. **Natural Spline:** $P_0''(x_0) = P_{n+1}''(x_{n+1}) = 0$
2. **Clamped Spline:** $P_0'(x_0) = d_0$, $P_{n+1}'(x_{n+1}) = d_{n+1}$
3. **Not-a-knot Spline:** Higher-order derivatives are continuous.

Finding Equations

We will be considering the natural boundary conditions. Now, we have $4n$ variables and $4n$ equations to solve. A straightforward approach is to plug in $P_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$ to obtain

$$P_i'(x) = 3a_i x^2 + 2b_i x + c_i$$

$$P_i''(x) = 6a_i x + 2b_i$$

This gives us the set of equations,

$$a_i x_{i+1}^3 + b_i x_{i+1}^2 + c_i x_{i+1} + d_{i+1} = y_{i+1}$$

$$a_{i+1} x_{i+1}^3 + b_{i+1} x_{i+1}^2 + c_{i+1} x_{i+1} + d_{i+1} = y_{i+1}$$

$$3a_i x_{i+1}^2 + 2b_i x_{i+1} + c_{i+1} - 3a_{i+1} x_{i+1}^2 - 2b_{i+1} x_{i+1} - c_{i+1} = 0$$

$$6a_i x_{i+1} + 2b_i = 6a_{i+1} x_{i+1} + 2b_{i+1}$$

$$6a_0 x_0 + 2b_0 = 0$$

$$6a_n x_{n+1} + 2b_n = 0$$

Finding Equations

Then this system of $4n$ equations can be directly solved by Gaussian elimination, to obtain the $4n$ coefficients.
However, there are ways to increase the efficiency and reduce the computational expense rather than directly brute-forcing the solution like above.

Efficient Solution for Cubic Splines

A more calculated approach is to construct $S''(x)$, and then obtain $S(x)$ by successively integrating it twice. In the following section we consider $P_i(x)$ to be the cubic spline in $[x_{i-1}, x_i]$
Let us denote $P_i''(x_i) = M_i = P_{i+1}''(x_i)$. We note, $M_0 = M_n = 0$.
Since P_i are cubics, S'' will be a piecewise linear function which interpolates the set of points $\{(x_i, M_i)\}$. Then, in $[x_{i-1}, x_i]$,

$$S''(x) = P_i''(x) = M_{i-1}\frac{x_i - x}{x_i - x_{i-1}} + M_i\frac{x - x_{i-1}}{x_i - x_{i-1}}$$

.

Efficient Solution for Cubic Splines

Integrating, we obtain

$$P_i'(x) = -\frac{M_{i-1}}{2(x_i - x_{i-1})}(x_i - x)^2 + \frac{M_i}{2(x_i - x_{i-1})}(x - x_{i-1})^2 + A_i$$

where A_i is a constant.

$$P_i(x) = \frac{M_{i-1}}{6(x_i - x_{i-1})}(x_i - x)^3 + \frac{M_i}{6(x_i - x_{i-1})}(x - x_{i-1})^3 + A_i x + B_i$$

where A_i, B_i are constants.

Efficient Solution for Cubic Splines

We have, $S(x_{i-1}) = y_{i-1}$ and $S(x_i) = y_i$. Using these,

$$P_i(x_{i-1}) = \frac{M_{i-1}}{6(x_i - x_{i-1})}(x_i - x_{i-1})^3 + A_i x_{i-1} + B_i$$

$$\implies y_{i-1} = \frac{M_{i-1}}{6}(x_i - x_{i-1})^2 + A_i x_{i-1} + B_i$$

$$P_i(x_i) = \frac{M_i}{6(x_i - x_{i-1})}(x_i - x_{i-1})^3 + A_i x_i + B_i = \frac{M_i}{6}(x_i - x_{i-1})^2 + A_i x_i + B_i$$

Final System of Equations

We have two equations and two unknowns A_i and B_i. Using these conditions to eliminate these two unknowns, we can simplify the expression to

$$\frac{x_i - x_{i-1}}{6} M_{i-1} + \frac{x_{i+1} - x_{i-1}}{3} M_i + \frac{x_{i+1} - x_i}{6} M_{i+1} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}}$$

along with the natural boundary conditions,

$$M_0 = 0$$

$$M_n = 0$$

The system of $n - 1$ equations can be solved for $M_1, M_2, ..., M_{n-1}$, from which $S''(x)$ and subsequently $S(x)$ can be determined.

Efficient Computation: Tridiagonal Systems

The resulting system of equations can be represented as:

$$\begin{bmatrix} \frac{x_1-x_0}{3} & \frac{x_2-x_1}{6} & 0 & \dots & 0 \\ \frac{x_1-x_0}{6} & \frac{x_2-x_0}{3} & \frac{x_3-x_2}{6} & \dots & 0 \\ 0 & \frac{x_2-x_1}{6} & \frac{x_3-x_1}{3} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \frac{x_n-x_{n-1}}{3} \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_n \end{bmatrix} = \begin{bmatrix} \frac{y_2-y_1}{x_2-x_1} - \frac{y_1-y_0}{x_1-x_0} \\ \frac{y_3-y_2}{x_3-x_2} - \frac{y_2-y_1}{x_2-x_1} \\ \vdots \\ \frac{y_n-y_{n-1}}{x_n-x_{n-1}} - \frac{y_{n-1}-y_{n-2}}{x_{n-1}-x_{n-2}} \end{bmatrix}.$$

Efficient algorithms (e.g., Jacobi Iterations) reduce the computational cost.

Methodology: Setting up our linear equations

We can write the equation for an interpolating cubic polynomial, given $n+1$ points (x_i, y_i) where $i \in \{0, 1, 2, \ldots, n\}$ in the following Newton divided difference form: Then, our cubic polynomials take the form

$$P_i = (1-t)\, y_{i-1} + t\, y_i + t(1-t)\big((1-t)\, a_i + t\, b_i\big),$$

where $i \in \{1, 2, \ldots, n\}$ and $t = \frac{x - x_{i-1}}{x_i - x_{i-1}}$.
Each of these n polynomials interpolate y in the intervals $[x_{i-1}, x_i]$, such that $P_i'(x_i) = P_{i+1}'(x_i)$.

Methodology: Setting up our linear equations

Then, these cubic polynomials together form a differentiable function in $[x_0, x_n]$. We have

$$a_i = k_{i-1}(x_i - x_{i-1}) - (y_i - y_{i-1}),$$

$$b_i = -k_i(x_i - x_{i-1}) + (y_i - y_{i-1})$$

for $i \in \{1, 2, \ldots, n\}$, where

$$k_0 = P_1'(x_0)$$

$$k_i = P_i'(x_i) = P_{i+1}'(x_i), \text{ when } i \in \{1, 2, \ldots, n-1\}$$

$$k_n = P_n'(x_n).$$

Methodology: Setting up our linear equations

In addition, if $P_i''(x_i) = P_{i+1}''(x_i)$ holds for $i \in \{1, 2, \ldots, n-1\}$, then the resulting function has a continuous second derivative.

From all the imposed conditions and the formulas for a_i and b_i, it is easy to see that this is the case if and only if

$$\frac{k_{i-1}}{x_i - x_{i-1}} + \left(\frac{1}{x_i - x_{i-1}} + \frac{1}{x_{i+1} - x_i} \right) 2k_i + \frac{k_{i+1}}{x_{i+1} - x_i} = 3 \left(\frac{y_i - y_{i-1}}{(x_i - x_{i-1})^2} + \right.$$

for $i \in \{1, 2, \ldots, n-1\}$. This gives us $n-1$ linear equations for the $n+1$ unknowns $k_0, k_1, \ldots, k_n$.

Methodology: Setting up our linear equations

A popular condition, which we will be using, is called the "natural spline" condition. The 2 extra equations are as follows

$$P_1''(x_0) = 2\frac{3(y_1 - y_0) - (k_1 + 2k_0)(x_1 - x_0)}{(x_1 - x_0)^2} = 0,$$

and

$$P_n''(x_n) = -2\frac{3(y_n - y_{n-1}) - (2k_n + k_{n-1})(x_n - x_{n-1})}{(x_n - x_{n-1})^2} = 0.$$

Simplifying further, we get

$$\frac{2}{x_1 - x_0}k_0 + \frac{1}{x_1 - x_0}k_1 = 3\frac{y_1 - y_0}{(x_1 - x_0)^2}$$

and

$$\frac{1}{x_n - x_{n-1}}k_{n-1} + \frac{2}{x_n - x_{n-1}}k_n = 3\frac{y_n - y_{n-1}}{(x_n - x_{n-1})^2}.$$

Methodology: Setting up our linear equations

Observe that the matrix we get looks like this:

$$\begin{bmatrix} a_{11} & a_{12} & 0 & \dots & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \dots & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{(n-1)(n-2)} & a_{(n-1)(n-1)} & a_{(n-1)n} \\ 0 & 0 & 0 & \dots & 0 & a_{n(n-1)} & a_{nn} \end{bmatrix}$$

where $a_{11} = \frac{2}{k_0+k_1}$, $a_{i(i+1)} = a_{(i+1)i} = \frac{1}{k_i+k_{i-1}}$ and $a_{ii} = 2 \cdot (\frac{1}{k_i+k_{i-1}} + \frac{1}{k_i+k_{i+1}})$. It is easy to see that this matrix is strictly diagonally dominant.

Jacobi Iteration Method

The Jacobi Iteration Method starts with an arbitrary guess for the solution to a set of linear equations written in the matrix form as $Ax = b$.

Rewriting as linear equations, we get:

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n)$$
$$x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n)$$
$$\vdots$$
$$x_n = \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1})$$

Jacobi Iteration Method

Then make an initial guess of the solution

$$x^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \ldots, x_n^{(0)}\right).$$

Substitute these values into the right-hand side of the rewritten equations to obtain the *first approximation*,

$$\left(x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, \ldots, x_n^{(1)}\right).$$

This accomplishes one *iteration*.
By repeated iterations, we form a sequence of approximations

$$x^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \ldots, x_n^{(k)}\right), \quad k = 1, 2, 3, \ldots$$

Jacobi Iteration Method

We now see that this method will converge for diagonally dominant matrices, such as the one we deal with in our method for cubic spline interpolation.
We split A into

$$A = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} - \begin{bmatrix} 0 & 0 & \cdots & 0 \\ -a_{21} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ -a_{n1} & \cdots & -a_{n,n-1} & 0 \end{bmatrix} -$$

$$\begin{bmatrix} 0 & -a_{12} & \cdots & -a_{1n} \\ 0 & 0 & \cdots & -a_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}.$$

This can be written as $A = D - L - U$.

Jacobi Iteration Method

The equation $A\mathbf{x} = \mathbf{b}$ is transformed into

$$(D - L - U)\mathbf{x} = \mathbf{b}.$$

Then,

$$\mathbf{x} = D^{-1}(L + U)\mathbf{x} + D^{-1}\mathbf{b}.$$

Jacobi Iteration Method

Define $T := D^{-1}(L+U)$ and $c := D^{-1}\mathbf{b}$. Then we have

$$\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c}, \quad k = 1, 2, 3, \ldots$$

.

Theorem 1 *If A is a diagonally dominant matrix, then for any choice of $x^{(0)}$, the Jacobi method gives a sequence $\{x^{(k)}\}_{k=0}^{\infty}$ which converges to the unique solution of $Ax = b$.*

This is true because the spectral radius of the matrix will be less than 1, by Gershgorin's Circle Theorem. After that, we can apply the following lemmas.

Lemma 1 If the spectral radius satisfies $\rho(T) < 1$, then $(I-T)^{-1}$ exists, and

$$(I-T)^{-1} = I + T + T^2 + \cdots = \sum_{j=0}^{\infty} T^j.$$

Jacobi Iteration Method

Lemma 2 For any $\mathbf{x}^{(0)} \in \mathbb{R}^n$, the sequence $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$ defined by

$$\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c} \quad \text{for each } k \geq 1$$

converges to the unique solution of $\mathbf{x} = T\mathbf{x} + \mathbf{c}$ if $\rho(T) < 1$. Here, $\rho(T)$ is the spectral radius of T.

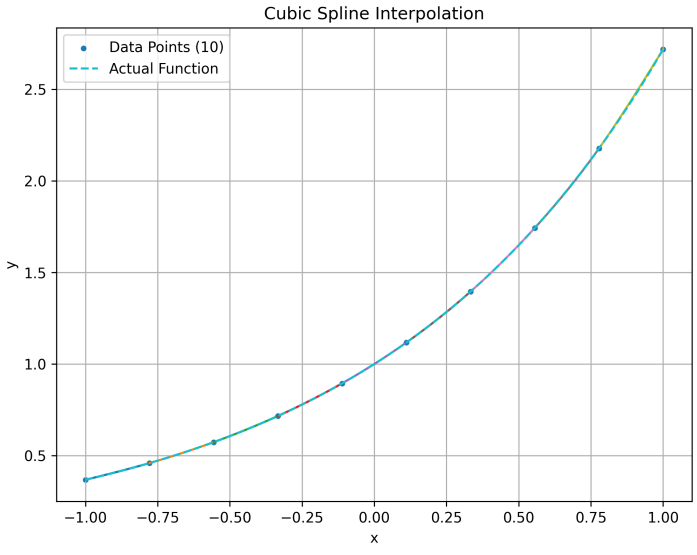
Proof.

$$\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c} = T\left(T\mathbf{x}^{(k-2)} + \mathbf{c}\right) + \mathbf{c} = \cdots = T^k\mathbf{x}^{(0)} + (T^{k-1} + \cdots +$$

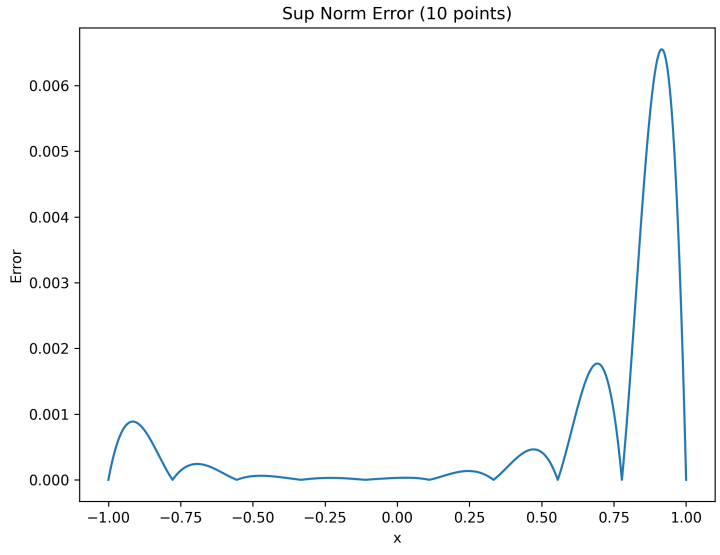
Since $\rho(T) < 1$, $\lim_{k\to\infty} T^k\mathbf{x}^{(0)} = 0$.

$$\lim_{k\to\infty} \mathbf{x}^{(k)} = 0 + \lim_{k\to\infty} \left(\sum_{j=0}^{k-1} T^j\right)\mathbf{c} = (I - T)^{-1}\mathbf{c}.$$

Plots

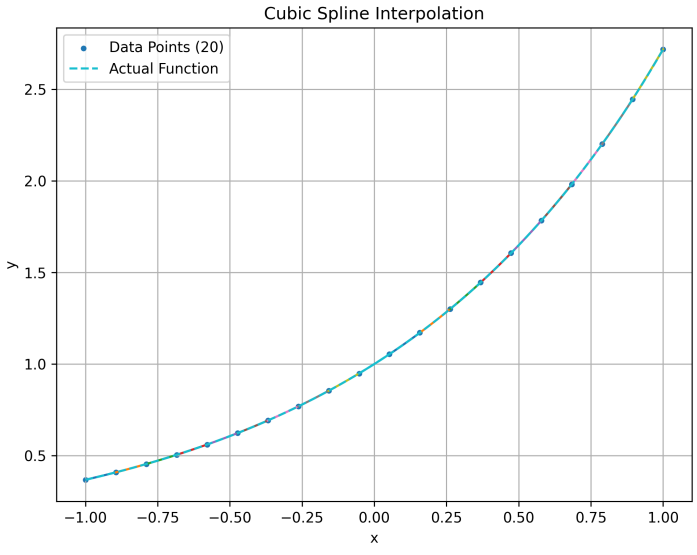


(a) Interpolation using 10 points

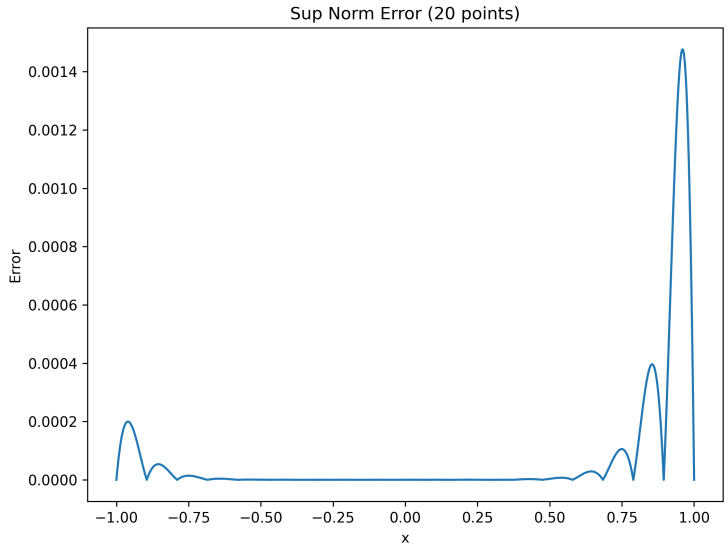


(b) Error plot for 10 points

Plots

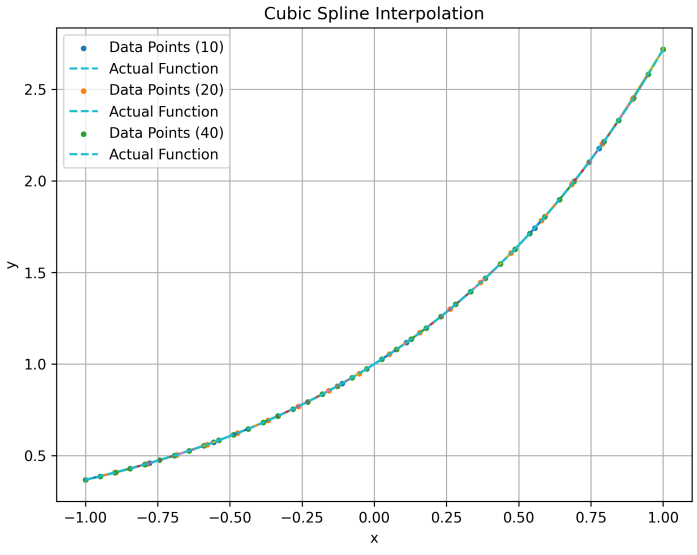


(a) Interpolation using 20 points

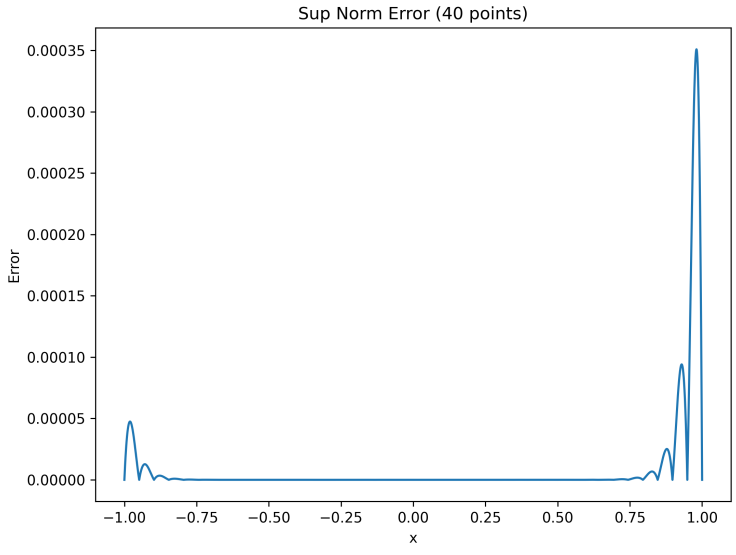


(b) Error plot for 20 points

Plots



(a) Interpolation using 40 points



(b) Error plot for 40 points

Plots

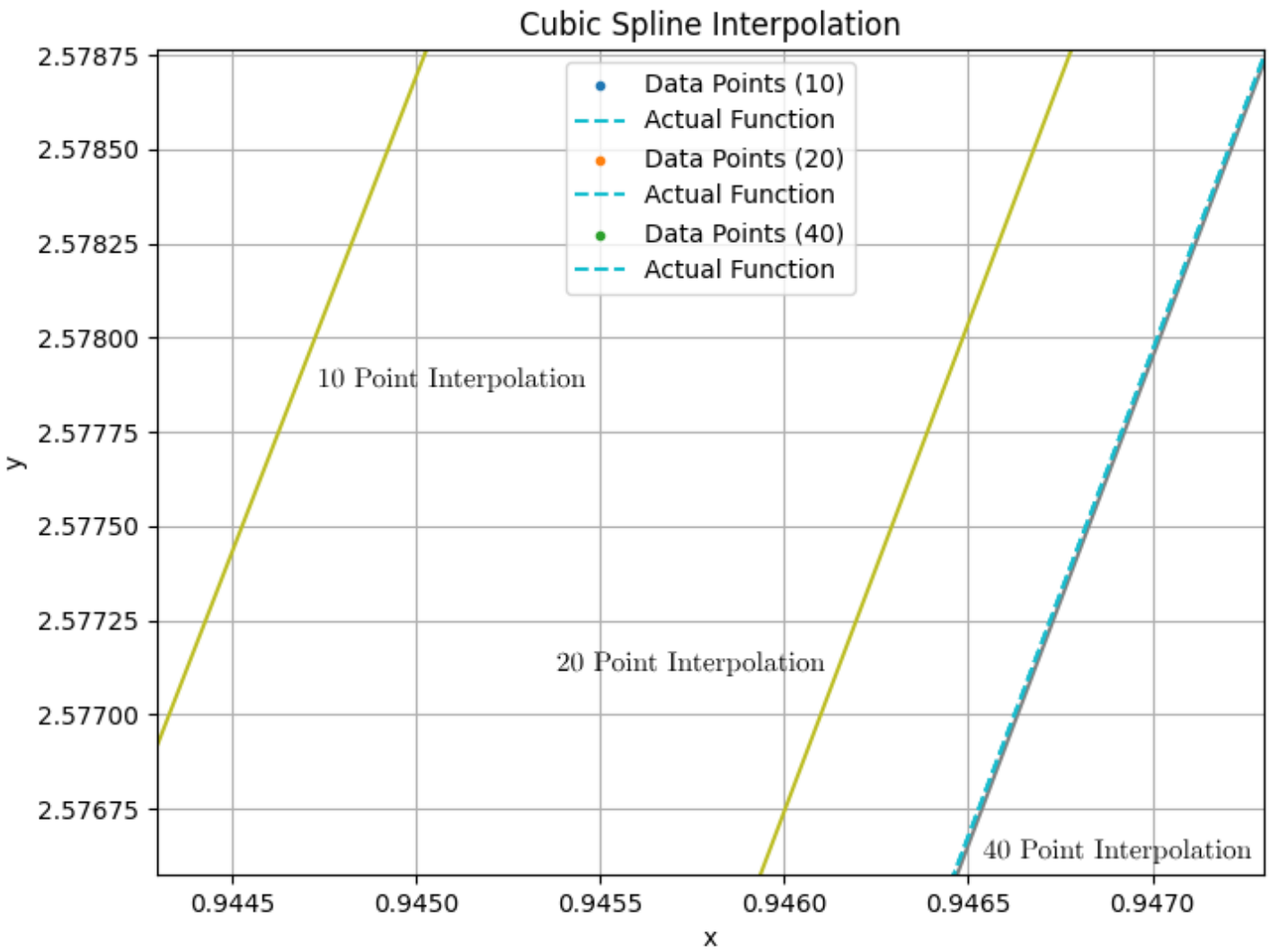
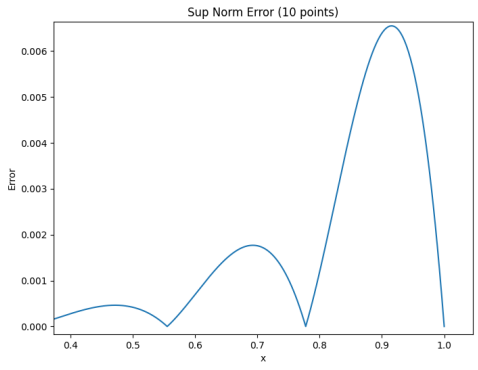
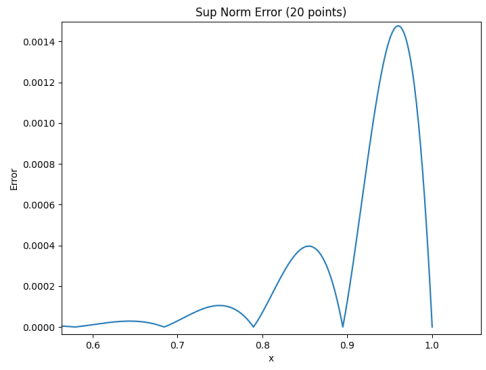


Figure: Zoomed in plot of the three interpolation attempts, using 10, 20 and 40 points

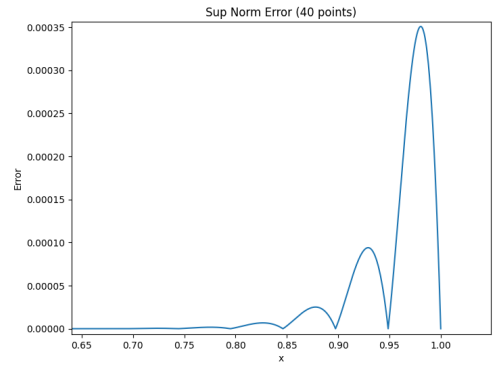
Plots



(a) Error plot for 10 points



(b) Error plot for 20 points



(c) Error plot for 40 points

Figure: Plots to visualize error differences with increasing number of points.

Error Analysis

Error in General Polynomial Interpolation

For a standard polynomial interpolation, for interpolating a function at $n+1$ points by a polynomial of degree n, the error is given by,

$$f(x) - p_n(x) = f[x_0, x_1, ..., x_n, x] \prod_{i=0}^{n} (x - x_i)$$

Lagrange's Form of the Remainder

Let us assume $f \in C^{n+1}[x_0, x_n]$. Let $f(x) - p_n(x) = R_n$. We define an auxiliary function

$$Y(t) = R_n(t) - R_n(x) \prod_{i=0}^{n} \frac{(t - x_i)}{(x - x_i)}$$

Then,

$$Y^{(n+1)}(t) = R_n^{(n+1)}(t) - R_n(x)(n+1)! \prod_{i=0}^{n} \frac{1}{(x - x_i)}$$

Lagrange's Form of the Remainder

Since, the degree of the interpolating polynomial $p_n(x)$ is $\leq n$, we have

$$R_n^{(n+1)}(t) = f^{(n+1)}(t)$$

and hence,

$$Y^{(n+1)}(t) = f^{(n+1)}(t) - R_n(x)(n+1)! \prod_{i=0}^{n} \frac{1}{(x - x_i)}$$

Since $R_n(x)$ and $\prod_{i=0}^{n}(x - x_i)$ are both 0, we have

$$Y(x) = Y(x_i) = 0$$

that is, Y has at least $n + 2$ roots.

Lagrange's form of the Remainder

By applying Rolle's theorem $n+1$ times, we get that $Y^{(n+1)}(t)$ has at least one root in $[x_0, x_1, ..., x_n]$.

$$\exists\, \xi \in [x_0, x_1, ..., x_n] \text{ such that}$$

$$Y^{(n+1)}(\xi) = f^{(n+1)}(\xi) - R_n(x)(n+1)! \prod_{i=0}^{n} \frac{1}{(x - x_i)} = 0$$

$$\implies R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

Error in Cubic Spline Interpolation

At any point $x \in [x_{i-1}, x_i]$, the error is given by

$$\xi(x) = |f(x) - S(x)| = |f(x) - P_i(x)|$$

Now, the general representation of $S(x) = P_i(x)$ is in the form of a cubic, $a_i x^3 + b_i x^2 + c_i x + d_i$. We have interpolated f with S such that $S(x_i) = f(x_i)$, and S' as well as S'' are continuous at all x_i. We assume, for all the following steps, that $f \in C^4$.

Error in Cubic Spline Interpolation

Using Taylor's theorem with Lagrange's form of the remainder in $[x_{i-1}, x_i]$, the Taylor expansion of $f(x)$ in $[x_i, x_{i+1}]$ about x_i is given by

$$f(x) = f(x_i) +$$

$$f'(x_i)(x - x_i) + \frac{f''(x_i)}{2!}(x - x_i)^2 + \frac{f^{(3)}(x_i)}{3!}(x - x_i)^3 + \frac{f^{(4)}(\xi)}{4!}(x - x_i)^4,$$

where $\xi \in [x_i, x_{i+1}]$.

From the error term in the standard polynomial interpolation, we have that the error in cubic spline interpolation will be $\approx O((x - x_i)^4)$.

Error in Cubic Spline Interpolation

The error can thus be bounded by the 4^{th} order term in the Taylor expansion. For uniformly spaced nodes, let $h = x_{i+1} - x_i$. On the interval $[x_i, x_{i+1}]$,

$$|\epsilon(x)| \leq \frac{\|f^{(4)}\|_\infty}{24}(x - x_i)^4$$

where $\|F\|_\infty$ denotes the infinity norm or the sup-norm of a function F and is defined as $\max_{i \in \mathbf{R}} |F(i)|$.
The maximum error occurs at $x = x_{i+1}$, where:

$$|\epsilon(x_{i+1})| \approx \frac{\|f^{(4)}\|_\infty}{24} h^4$$

Error in Cubic Spline Interpolation

The full computation is done by integrating the contributions of the error term over all intervals using the cubic spline basis functions. Let $S_i(x)$ be the cubic spline basis function associated with the node x_i, such that

$$S(x) = \sum_{i=0}^{n} c_i S_i(x)$$

where c_i are elements of the coordinate vector. The contribution to the error from $S_i(x)$ can be expressed as,

$$\int_{x_0}^{x_n} \epsilon(x) S_i(x)\, dx$$

Error in Cubic Spline Interpolation

For natural cubic splines, the $S_i(x)$ is symmetric, and its fourth derivative's integral, under the natural boundary conditions $S''(x_0) = S''(x_n) = 0$, evaluates to

$$\int_{x_0}^{x_n} \left(S_i^{(4)}(x)\right)^2 \, dx = \frac{1}{4!16} = \frac{1}{384}$$

Optimizing over all basis functions, the contribution of $S_i(x)$ evaluates to $\frac{5}{384}$.
Hence, the error for the cubic spline interpolation can be given by,

$$\|\epsilon(x)\|_\infty \leq \frac{5}{384} \|f^{(4)}\|_\infty h^4.$$

Error in Jacobi Iteration Method

From the theorem and lemmas in the section on the acobi iteration method, we note that if $\|T\| < 1$ for any natural matrix norm and $\mathbf{c}$ is a given vector, then the sequence $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$ defined by

$$\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c}$$

converges, for any $\mathbf{x}^{(0)} \in \mathbb{R}^n$, to a vector $\mathbf{x} \in \mathbb{R}^n$, with $\mathbf{x} = T\mathbf{x} + \mathbf{c}$, and the following error bound holds: $\|\mathbf{x} - \mathbf{x}^{(k)}\| \leq \|T\|^k \|\mathbf{x}^{(0)} - \mathbf{x}\|$.

Error in Jacobi Iteration Method

So, the error is approximated as $\|\mathbf{x} - \mathbf{x}^{(k)}\| \approx \rho(T)^k \|\mathbf{x} - \mathbf{x}^{(0)}\|$.

In the worst case, where every element of the matrix is operated in each iteration, the time complexity for each iteration is $\mathcal{O}(n^2)$. The overall time complexity of the algorithm depends on the matrix properties, as shown by the error bound.

Comparison with Linear and Quadratic Interpolation

For interpolating the same set of equally spaced $n+1$ nodes with an n^{th} order polynomial, the error would be given by

$$\frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x-x_i) = \frac{f^{(n+1)}(\xi)}{(n+1)!}h^{n+1}$$

Hence, for a piecewise linear interpolation, the error would be bounded by

$$\frac{1}{2}\|f^{(2)}\|_\infty h^2$$

and for piecewise quadratic interpolation, the error would be bounded by

$$\frac{1}{6}\|f^{(3)}\|_\infty h^3.$$

Some more plots

We present some error plots below, which compare linear, quadratic and cubic spline interpolation techniques. We are interpolating e^x over the interval $[-1, 1]$, for 10, 20 and 40 points respectively.

The sup-norm error for cubic spline interpolations was found to be 0.00655 for 10 points, 0.00148 for 20 points and 0.00035 for 40 points. It is clear that the error decreases proportionally with $1/n^2$, where n is the number of points.

Some more plots

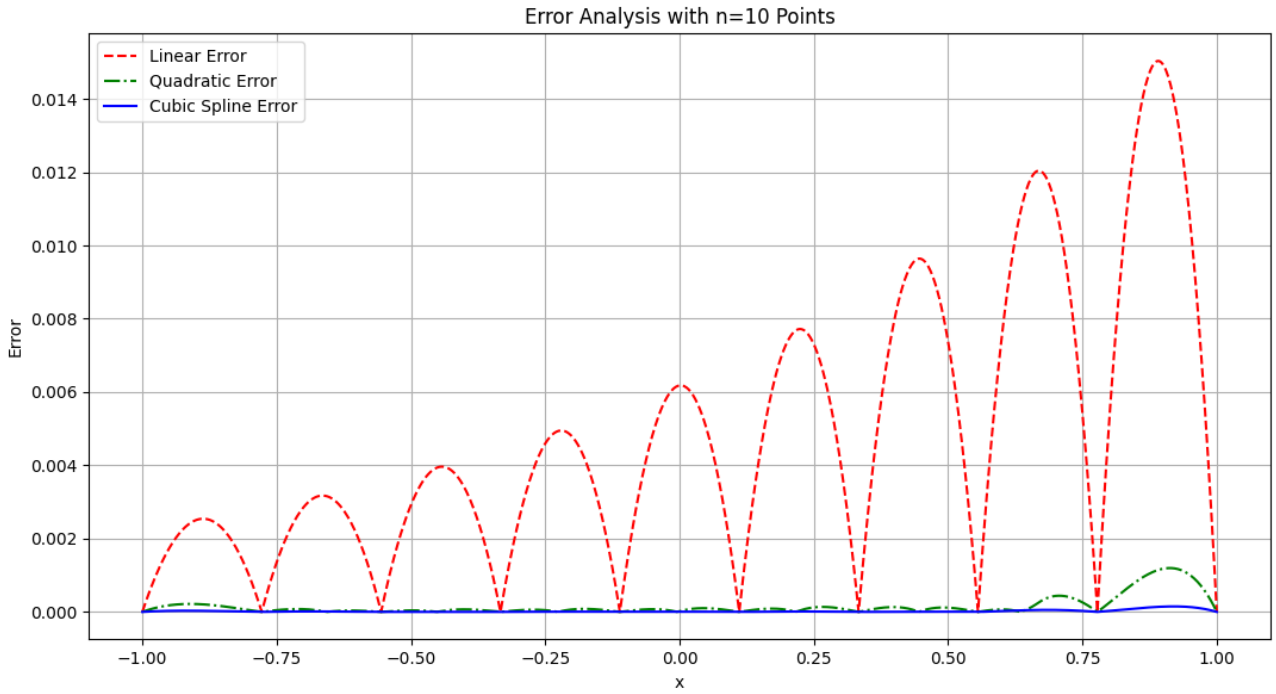


Figure: Error comparison for 10 points

Some more plots

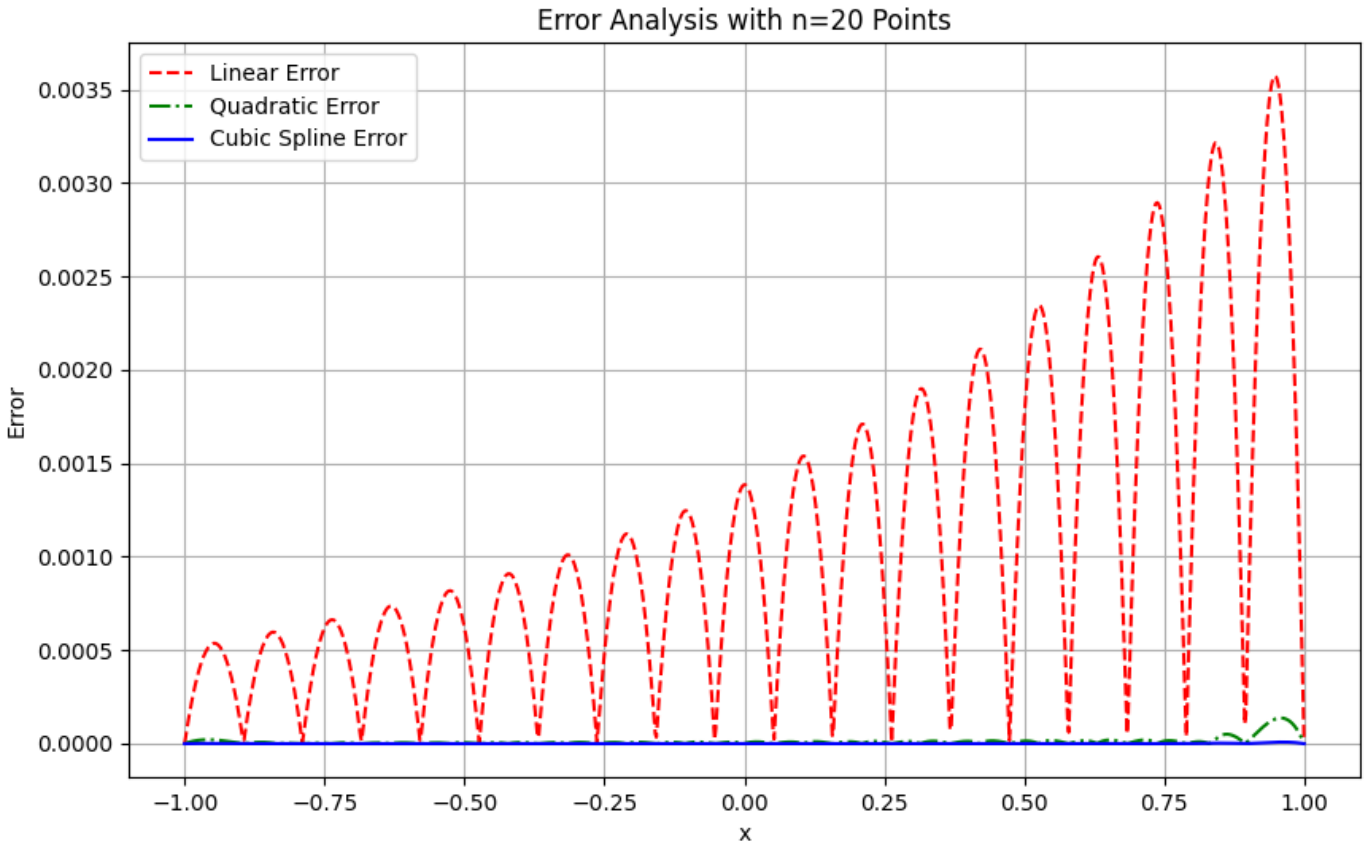


Figure: Error comparison for 20 points

Some more plots

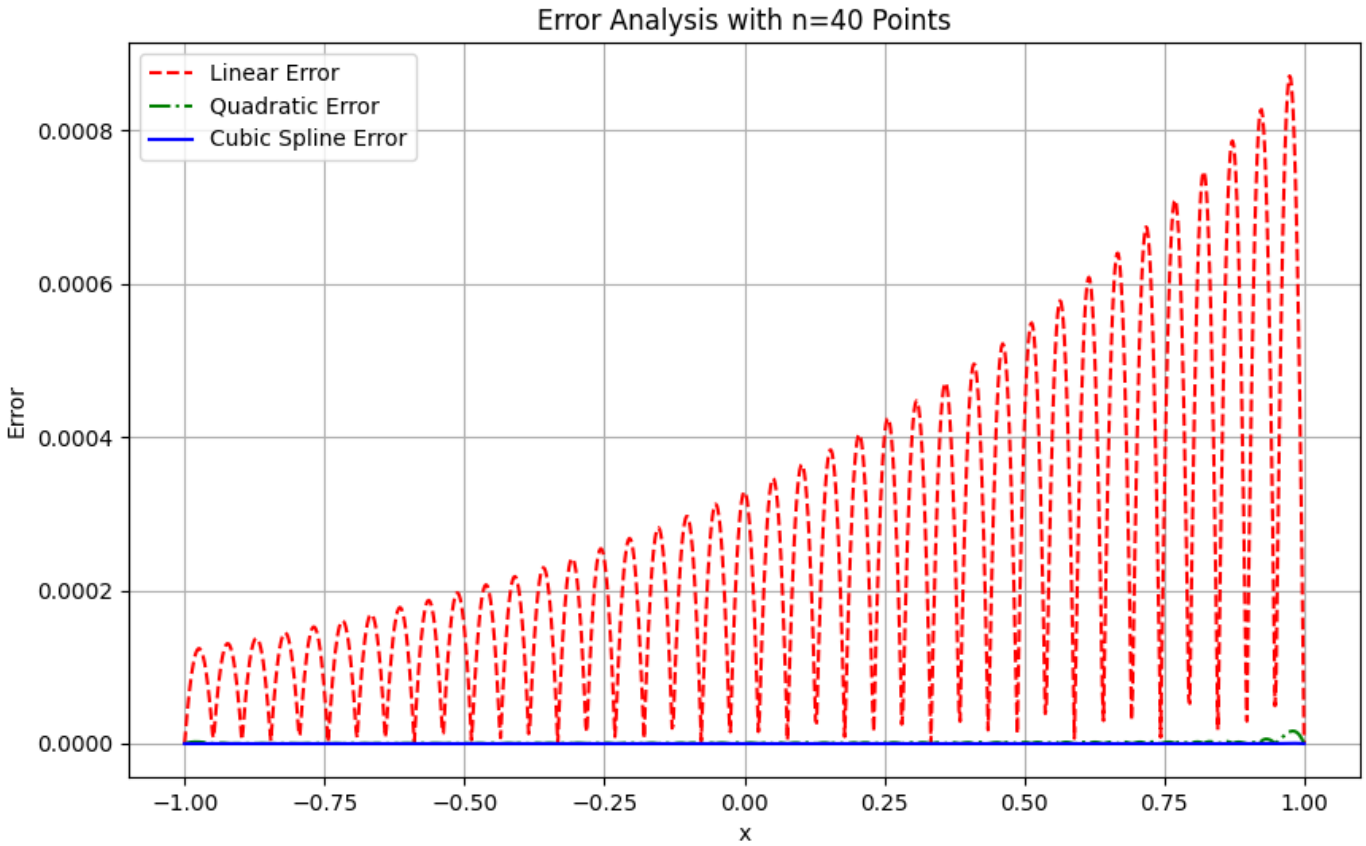


Figure: Error comparison for 40 points

Some more plots

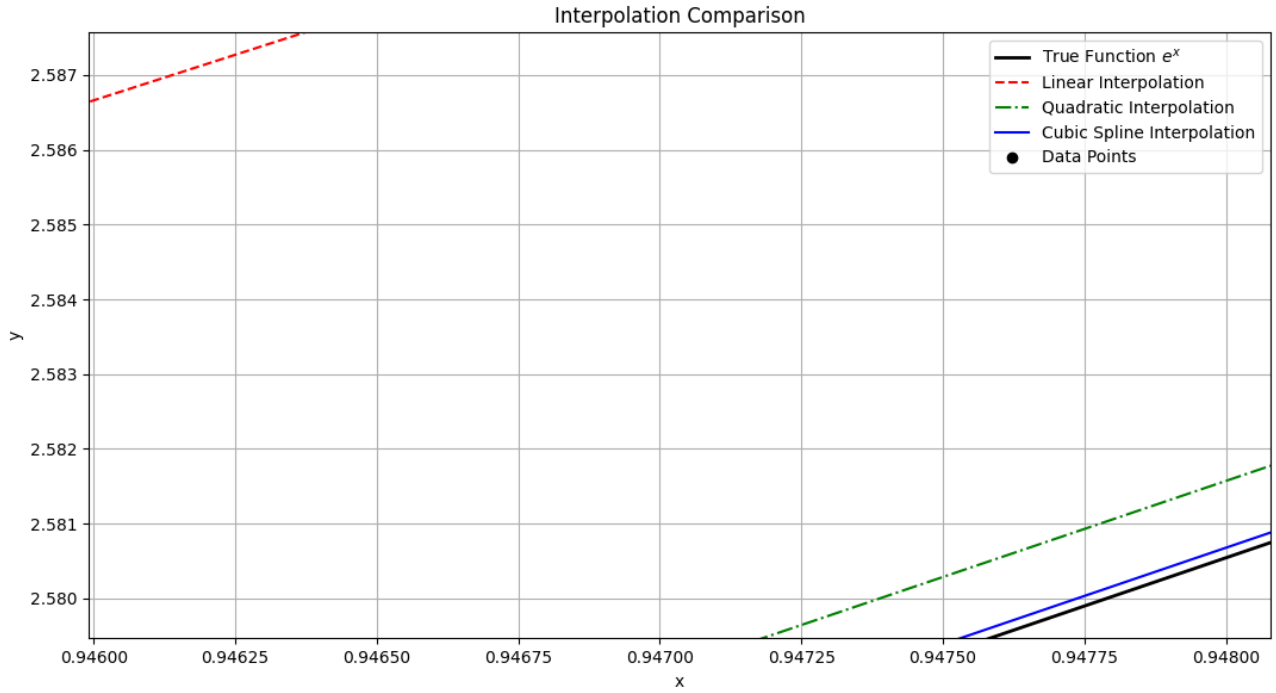


Figure: Comparison between the interpolation methods for 10 points