Chapter 6

Interpolation

***** Updated **5/21/12 HG**

What's Ahead

- Polynomial Interpolation: Existence and Uniqueness
- Interpolation Error
- Interpolating Methods: Lagrange, Newton, Hermite, Monomial, and Chebyshev
- Piecewise Interpolation
- Spline Interpolation
- Bezier Curves

6.1 Introduction

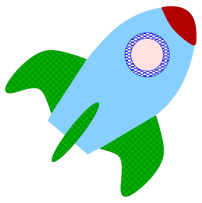
The word "interpolation" refers to interpolating some unknown information from a given set of known information. The technique of interpolation is widely used as a valuable tool in science and engineering. The problem is a classical one and dates back to the time of Newton and Kepler, who needed to solve such a problem in analyzing data on the position of stars and planets.

Mathematical applications of interpolation include derivation of computational techniques for

- Numerical differentiation
- Numerical integration
- Numerical solutions of differential equations

We now mention a simple real-life application.

6.2 A Case Study: Upward Velocity of a Rocket



As a rocket is launched from the ground, its upward velocity, $v(t)$ (m/s), is measured at certain time instants $t(s)$. Suppose that one measures the upward velocity of a rocket for time $0 \le t \le 30$ and the measurements are tabulated as follows:

$t(s)$	$v(t)$ (m/s)
0	0
10	250
15	350
22	655
25	890
30	910

Using the above table, one would like to predict the velocity of the rocket at certain nontabulated times, say, $t = 5$s, $t = 20$s, $t = 23$s, $t = 29$s. Such a problem of predicting the values of the dependent variable at nontabulated values of the independent variable in a given interval is called **interpolation**, which will be the subject matter of this chapter.

For the above rocket example, if we can find a function $v(t)$, that interpolates the above data, then it can be immediately used to predict its value for any value of t in that interval.

For this example, of course, once $v(t)$ is determined, it can also be used to find the acceleration of the rocket at a certain given time just by differentiating the function. Similarly, the distance covered from time $t = t_1$ to time $t = t_2$ ($t_2 > t_1$) in the given interval, can be obtained by evaluating the integral

$$\int_{t_1}^{t_2} v(t)\, dt.$$

Numerical differentiation and integration will be discussed in later chapters.

6.3 Problem Statement

Suppose that the following table is generated from some experiment.

x_0	f_0
x_1	f_1
x_2	f_2
$\vdots$	$\vdots$
x_k	f_k
$\vdots$	$\vdots$
x_n	f_n

In the above table, f_k, $k = 0, \ldots, n$ are known to be the values of a certain function $f(x)$, evaluated at x_k, $k = 0, \ldots, n$ in an interval containing these points. *Note that only the functional values are known, but not the function $f(x)$.*

The problem is to find the functional value f_I corresponding to a nontabulated intermediate value $x = x_I$.

Such a problem is called an **Interpolation Problem**. The numbers x_0, x_1, …, x_n are called the **nodes**. Here is a formal definition.

Interpolation Problem

Given

$$\begin{cases} \text{The } (n+1) \text{ nodes: } x_0, x_1, \ldots, x_n \\ \text{The functional values: } f_0, f_1, \ldots, f_n \text{ at these nodes} \\ \text{An Intermediate (nontabulated) point: } x_I \end{cases}$$

Predict f_I : the value at $x = x_I$.

Note: For illustration purposes and to estimate the error of interpolation, we will very often assume that $f(x)$ is explicitly known. But in practice, only the functional values are known. Even if some representation of $f(x)$ is known, it may be too difficult to evaluate the functional values at certain points.

There are various forms of interpolation including:

- Algebraic polynomial interpolation
- Piecewise polynomial interpolation

- Trigonometric interpolation
- Rational function interpolation

etc. We will discuss only the **polynomial interpolation** (including piecewise polynomial interpolation) in this chapter. Discussions of the other forms of interpolation will be deferred until later chapters.

6.4 Existence and Uniqueness of the Interpolation Polynomial

It is well-known that a continuous function $f(x)$ on $[a, b]$ can be approximated as close as possible by means of a polynomial. This is a classical result, known as **Weierstrass Approximation Theorem**.

Weierstrass Approximation Theorem

Given a continuous function $f(x)$ on $[a, b]$, and a small number $\epsilon (> 0)$.

There exists a polynomial $P(x)$ such that

$$|f(x) - P(x)| < \epsilon$$

for all x in $[a, b]$.

Knowing that f_k, $k = 0, \ldots, n$ are the values of a certain function at x_k, the most obvious thing then to do is to construct a polynomial $P_n(x)$ of degree at most n that passes through the $(n+1)$ points: (x_0, f_0), (x_1, f_1), …, (x_n, f_n).

Indeed, if the nodes x_0, x_1, …, x_n are assumed to be distinct, then such a polynomial always does exist and is unique, as can be seen from the following discussion.

Even though the polynomial is unique, it can be constructed using different bases. We will first show below how this polynomial can be constructed using the **monomial basis**: $\{1, x, \ldots, x^n\}$; that is, $P_n(x)$ will be constructed in the following form: $P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$.

If $P_n(x)$ interpolates at $x_0, x_1, \ldots, x_n$, we must have, by definition

$$\begin{aligned} P_n(x_0) &= a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n = f_0 \\ P_n(x_1) &= a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n = f_1 \\ &\vdots \\ P_n(x_n) &= a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n = f_n \end{aligned} \tag{6.1}$$

The system (6.1) contains $(n+1)$ equations in $(n+1)$ unknowns: $a_0, a_1, \ldots, a_n$ which are the coefficients of $P_n(x)$. In matrix notation, the system can be written in the form: $Ax = b$.

$$\underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}}_{x} \underset{=}{} \underbrace{\begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}}_{b} \tag{6.2}$$

Because $x_0, x_1, \ldots, x_n$ are distinct, it can be shown [**Exercise**] that the matrix of the above system is nonsingular. Thus, the linear system for the unknowns $a_0, a_1, \ldots, a_n$ has a unique solution, in view of the following well-known result, available in any linear algebra textbook (e.g., Datta (2010)).

The algebraic linear system $Ax = b$ has a unique solution for every b if and only if A is nonsingular.

This means that $P_n(x)$ exists and is unique.

We summarize the result in the following theorem.

Theorem 6.1 (Existence and Uniqueness Theorem for Polynomial Interpolation). Given $(n+1)$ **distinct** points: $x_0, x_1, \ldots, x_n$, and the associated functional values $f_0, f_1, \ldots, f_n$ of a function $f(x)$ at these points; that is,

$$f(x_i) = f_i, \quad i = 0, 1, \ldots, n.$$

There exists a **unique polynomial** $P_n(x)$ of degree at most n such that

$$P_n(x_i) = f_i, \quad i = 0, 1, \ldots, n.$$

(*That is, the values of $P_n(x)$ coincide with the functional values at $x = x_i$.*)

Definition 6.2. The polynomial $P_n(x)$ in **Theorem 6.1** is called the **interpolating polynomial**. The point x_I at which $f(x)$ has to be interpolated is called the **interpolating point**.

6.5 Construction of the Interpolating Polynomial using Monomial Basis

The constructive proof of Theorem 6.1 provides us a method to construct the interpolating polynomial in the form: $P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$.

The method is stated in the following algorithmic form:

Algorithm 6.3 (Interpolation via Monomial Basis).

Inputs: (i) The $(n+1)$ distinction nodes: $x_0, x_1, \dots, x_n$

(ii) The $(n+1)$ functional values at these nodes: $f_0, f_1, \dots, f_n$

Output: The interpolating polynomial $P_n(x)$.

Step 1. Form the matrix A from the nodes, and the vector b from the functional values, as follows:

$$A = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_1^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}, \quad b = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}$$

Step 2. Solve the $(n+1) \times (n+1)$ linear system:

$$Ax = b$$

for the unknown vector $x = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}$. (The vector x consists of the coefficients of $P_n(x)$.)

Step 3. Construct the interpolating polynomial in the form: $P_n(x) = a_0 + a_1 x + \cdots + a_n x^n$.

Conditioning and Accuracy

The matrix A above is known as the **Vandermonde matrix** and is notoriously **ill-conditioned**, if the nodes are *close to each other* or *the matrix is of high-order*. In such a case the system may not be accurately solved. For example, the condition number of a Vandermonde matrix of order 10 with nodes on 1 through 10 is of order $O(10^{13})$. (The readers are invited to verify this fact using the built-in MATLAB functions, **vandermonde** and **Cond**).

Lagrange and Newton Methods of Interpolation

In view of the above remarks, very often the interpolating polynomial $P_n(x)$ is not constructed in practice using the monomial basis.

We will describe two other computationally different methods in the sequel which have better numerical properties. (We want to remind the readers that, *though these methods are different, the interpolating polynomial is still the same.*)

- **Lagrangian Interpolation:** The basis functions for the Lagrange method is a set of n polynomials $L_i(x), i = 0, \ldots, n$, called **Lagrange polynomials**. Because of some special properties of these polynomials (see next section), the matrix A is an identity matrix and *therefore is well-conditioned.*

- **Newton Interpolation:** The basis functions for the Newton interpolation is the set of $n+1$ polynomials: $\{1, (x-x_0), (x-x_0)(x-x_1), \ldots, (x-x_0)(x-x_1)(x-x_{n-1})\}$.

  The matrix A is *lower triangular* in this case; *whose condition number depends upon the ordering of the nodes.* (**Do an example to convince yourself**.)

6.6 Lagrange Interpolation

A classical method, due to the famous French astronomer, **Joseph Lewis Lagrange** (1736-1813), is the **Lagrange interpolation**.

Idea for Lagrange Interpolation

The **idea** is to construct the interpolating polynomial $P_n(x)$ in the form:

$$P_n(x) = L_0(x)f_0 + L_1(x)f_1 + \cdots + L_n(x)f_n \tag{6.3}$$

where each $L_k(x), k = 0, 1, \ldots, n$ is a polynomial of degree at most n, called the **Lagrange polynomial**, defined in the following.

Lagrange Polynomials: The Lagrange polynomials $\{L_k(x)\}$ each of degree n are defined as follows:

$$L_0(x) = \frac{(x-x_1)\cdots(x-x_n)}{(x_0-x_1)\cdots(x_0-x_n)} \textbf{ (the factor } (x-x_0) \textbf{ is missing)}.$$
$$L_1(x) = \frac{(x-x_0)(x-x_2)\cdots(x-x_n)}{(x_1-x_0)(x_1-x_2)\cdots(x_1-x_n)} \textbf{ (the factor } (x-x_1) \textbf{ is missing)}.$$

In general,

$$L_k(x) = \frac{(x-x_0)(x-x_1)(x-x_2)\cdots(x-x_{k-1})}{(x_k-x_0)(x_n-x_1)(x_n-x_2)\cdots(x_k-x_{k-1})} \quad \textbf{(the factor } (x-x_k) \textbf{ is missing).}$$
$$k = 0, 1, 2, \ldots, n. \tag{6.4}$$

Lagrange polynomial properties:

From the definition, the following useful properties of Lagrange polynomials can be easily proved [**Exercise**]:

$$\begin{aligned} L_0(x_0) &= 1, \quad L_0(x_1) = L_0(x_2) = \cdots = L_0(x_n) = 0 \\ L_1(x_1) &= 1, \quad L_1(x_0) = L_1(x_2) = \cdots = L_n(x_n) = 0 \end{aligned}$$

In general,

$$\begin{cases} L_k(x_k) = 1, \\ L_k(x_i) = 0, \quad i \neq k. \end{cases}$$

Validation of (6.3)**:** Using the above properties, we will now show that $P_n(x)$, defined by (6.3), is indeed the interpolating polynomial.

n=1. The nodes are x_0 and x_1:

$$L_0(x) = \frac{x-x_1}{x_0-x_1}, \quad L_1(x) = \frac{x-x_0}{x_1-x_0}.$$
$$P_1(x) = L_0(x)f_0 + L_1(x)f_1$$

Substituting the values of $L_0(x)$ and $L_1(x)$ in $P_1(x)$ we obtain

$$P_1(x) = \frac{x-x_1}{x_0-x_1}f_0 + \frac{x-x_0}{x_1-x_0}f_1$$

Substituting x_0 and x_1 in $P_1(x)$, we see:

$$P_1(x_0) = f_0, \quad P_1(x_1) = f_1$$

This proves that $P_1(x)$ is an interpolating polynomial of degree 1, for the two nodes, x_0 and x_1.

General case. The general case is similar.

Substituting the values of $L_0(x), L_1(x), \ldots, L_n(x)$ from (6.4) (with $k = 0, 1, \ldots, n$) into (6.3), and using the above Lagrange polynomial properties, it is easy to see that

$$\begin{aligned}
P_n(x_0) &= L_0(x_0)f_0 + L_1(x_0)f_1 + \cdots + L_n(x_0)f_n = f_0 \\
P_n(x_1) &= L_0(x_1)f_0 + L_1(x_1)f_1 + \cdots + L_n(x_1)f_n = 0 + f_1 + \cdots + 0 = f_1 \\
&\vdots \\
P_n(x_n) &= L_0(x_n)f_0 + L_1(x_n)f_1 + \cdots + L_n(x_n)f_n = 0 + 0 + \cdots + 0 + f_n = f_n
\end{aligned}$$

That is, the polynomial $P_n(x)$ has the property that

$$P_n(x_k) = f_k, \quad k = 0, 1, \ldots, n.$$

which means that the polynomial $P_n(x)$ is the interpolating polynomial, for the nodes, $x_0, x_1, \ldots, x_n$. This polynomial is called the **Lagrange Interpolating Polynomial**.

Algorithm 6.4 (Lagrange Interpolation).

Inputs: (i) The degree of the interpolation n.

(ii) The $(n + 1)$ nodes: $x_0, x_1, \ldots, x_n$

(iii) The function values of a function $f(x)$ at the nodes: $f_0, f_1, \ldots, f_n$.

(iv) The interpolating point X_I

Output: The interpolated value Y_I of $f(X_I)$.

Step 1. Compute $L_k(X_I)$ where $L_k(X)$ are defined by (6.4); $k = 0, 1, \ldots, n$.

Step 2. Compute the interpolated value of $f(X_I)$:

$$Y_I = \sum_{k=0}^{n} L_k(X_I) f_k.$$

Example 6.5

Using the following table,

(a) Explicitly construct the Lagrange Interpolating polynomial $P_3(x)$.

(b) Interpolate $f(3)$.

0	7
1	13
2	21
4	43

Input-Data:

$$\begin{cases} \text{Degree of the interpolating polynomial: } n = 3. \\ \text{Nodes: } x_0 = 0, x_1 = 1, x_2 = 2, x_3 = 4. \\ \text{Functional values: } f_0 = 7, f_1 = 13, f_2 = 21, f_3 = 43. \\ \text{Interpolating point: } X_I = 3. \end{cases}$$

Formulas to be used:

$$\begin{cases} P_3(x) = L_0(x)f_0 + L_1(x)f_1 + L_2(x)f_2 \\ Y_I = P_3(X_I) \end{cases} \qquad \begin{cases} L_0(x) &= \dfrac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_1-x_2)(x_0-x_3)} \\ L_1(x) &= \dfrac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} \\ L_2(x) &= \dfrac{(x-x_0)(x-x_1)(x-x_2)}{(x_2-x_0)(x_2-x_2)(x_2-x_3)} \end{cases}$$

Solution (a).

Step 1. Construct the Lagrange polynomials:

$$L_0(x) = \frac{(x-1)(x-2)(x-4)}{(-1)(-2)(-4)} = \frac{(x-1)(x-2)(x-4)}{-8}$$

$$L_1(x) = \frac{(x-0)(x-2)(x-4)}{1\cdot(-1)(-3)} = \frac{x(x-2)(x-4)}{3}$$

$$L_2(x) = \frac{(x-0)(x-1)(x-4)}{2\cdot 1\cdot(-2)} = \frac{x(x-1)(x-4)}{-4}$$

$$L_3(x) = \frac{(x-0)(x-1)(x-2)}{4\cdot 3\cdot 2} = \frac{x(x-1)(x-2)}{24}$$

Step 2. From the Lagrange interpolating polynomial $P_3(x)$ of degree 3:

$$P_3(x) = 7L_0(x) + 13L_1(x) + 21L_2(x) + 43L_3(x)$$

Solution (b).

We will compute $P_3(3)$ and accept it as an interpolated value of $f(3)$.

$L_0(3) = \frac{1}{4}$, $L_1(3) = -1$, $L_2(3) = \frac{3}{2}$, $L_3(3) = \frac{1}{4}$.

So, $P_3(3) = 7L_0(3) + 13L_1(3) + 21L_2(3) + 43L_3(3) = 31$.

Interpolated value of $f(3) = P_3(3) = 31$.

Accuracy Check: Note that $f(x)$ in this case is $f(x) = x^2 + 5x + 7$, and the exact value of $f(x)$ at $x = 3$ is 31. Thus, the interpolation error for this example is zero. This is because the function to be interpolated is itself a polynomial of degree 2, and the interpolating polynomial $P_3(x)$ is of degree 3.

Example 6.6

Given

i	x_i	$f(x_i)$
0	2	$\frac{1}{2}$
1	2.5	$\frac{1}{2.5}$
2	4	$\frac{1}{4}$

(a) Explicitly find the Lagrange interpolating polynomial $P_2(x)$,

(b) Use $P_2(x)$ to interpolate $f(3)$.

Input Data:

$$\begin{cases} \text{Degree of the interpolating polynomial: } n = 2. \\ \text{Nodes: } x_0 = 2, x_1 = 2.5, x_2 = 4 \\ \text{Functional values: } f_0 = \dfrac{1}{2}, f_1 = \dfrac{1}{2.5}, f_2 = \dfrac{1}{4} \end{cases}$$

Details of the problem to be solved: Compute $P_2(3)$ and accept it as the interpolated value of $f(3)$.

Solution (a).

Step 1. Find Lagrange Polynomials:

$$L_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{(x-2.5)(x-4)}{(-0.5)(-2)} = (x-2.5)(x-4) = x^2 - 6.5x + 10$$

$$L_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} = \frac{(x-2)(x-4)}{(0.5)(-1.5)} = -\frac{1}{0.75}(x-2)(x-4) = \frac{x^2-6x+8}{0.75}$$

$$L_2(x) = \frac{(x-x_1)(x-x_0)}{(x_2-x_1)(x_2-x_0)} = \frac{1}{3}(x-2.5)(x-2) = \frac{x^2-4.5x+5}{3}$$

Step 2. Find the interpolating polynomial $P_2(x)$:

$$P_2(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x) = \frac{1}{2}L_0(x) + \frac{1}{2.5}L_1(x) + \frac{1}{4}L_2(x)$$

$$= \frac{1}{2}(x^2 - 6.5x + 10) + \frac{1}{2.5}\frac{(x^2-6x+8)}{0.75} + \frac{1}{4}\frac{(x^2-4.5x+5)}{3}$$

Solution (b).

Interpolated value of $f(3) = P_2(3) = 0.3250$.

Accuracy Check:

(i) The value of $f(x)$ at $x = 3$ is $\frac{1}{3} \approx 0.3333$.

Absolute Error: $|f(3) - P_2(3)| = |0.3333 - 0.3250| = 0.0083$.

Barycentric Interpolation: A Variation of Lagrange Interpolation

There exists a variation of Lagrange interpolation, known as the **Barycentric interpolation** which has better numerical properties. It is **fast** and **stable**. "It deserves to be known as the standard method of polynomial interpolation" (Berrut and Trefethen [ ]).

Setting

$$w_k = \frac{1}{\underset{\substack{j=0\\ j\neq k}}{\Sigma^n}} \Pi(x_k - x_j)$$

the Lagrange interpolating polynomial $P_n(x)$ can be written as [**Exercise**]:

$$P_n(x) = \frac{\displaystyle\sum_{k=0}^{n} f_k \frac{w_k}{x-x_k}}{\displaystyle\sum_{k=0}^{n} \frac{w_k}{x-x_k}}.$$

6.7 Error in Interpolation

Since $f(x)$ is approximated by an interpolating polynomial $P_n(x)$, we would like to obtain an expression for the error of interpolation at a give intermediate point, say, $\bar{x}$.

That is, we would like to calculate

$$E(\bar{x}) = f(\bar{x}) - P_n(\bar{x}).$$

Note that, since $P_n(x_i) = f(x_i)$, $E(x_i) = 0$, $i = 0, 1, 2, \ldots, n$, that is, *there are no errors of interpolating at a tabulated point.*

Here is a result for the error expression at any point $\bar{x}$, $E(\bar{x})$, in $[a, b]$.

Theorem 6.7 (Interpolation-Error Theorem). Let

(i) $f(x)$ be $(n+1)$ times continuously differentiable on the closed interval $[a, b]$,

(ii) $x_0, x_1, \ldots, x_n$ be $(n+1)$ distinct numbers in $[a, b]$,

(iii) $P_n(x)$ be the interpolating polynomial of degree at most n that interpolates $f(x)$ in $[a, b]$.

Then for every $\bar{x}$ in $[a, b]$, there exists a number $\xi = \xi(\bar{x})$ in (a, b) (depending on $\bar{x}$) such that

$$E_n(\bar{x}) = f(\bar{x}) - P_n(\bar{x}) = \frac{f^{(n+1)}(\xi(\bar{x}))}{(n+1)!} \prod_{i=0}^{n} (\bar{x} - x_i). \tag{6.5}$$

For proof of **Theorem 6.7** we need the following result from Calculus:

Generalized Rolle's Theorem

Suppose $f(x)$ satisfies the following hypotheses:

(i) it is continuous on $[a, b]$

(ii) it is n times differentiable on (a, b)

(iii) it has $(n+1)$ distinct zeros in $[a, b]$.

Then there exists a number c in (a, b) such that

$$f^{(n)}(c) = 0$$

Proof of Theorem 6.7. **Case A.** Suppose that $\bar{x}$ is one of the numbers $x_0, x_1, \ldots, x_n$.

The proof in this case is trivial; because $f(\bar{x}) = P_n(\bar{x})$. Thus, the error is identically zero. The result will, therefore, hold for any arbitrary number ξ.

Case B. $\bar{x}$ is not one of the numbers $x_0, x_1, \ldots, x_n$.

To prove the theorem in this case, we define a function $g(t)$ in the variable t in $[a, b]$ as:

$$g(t) = f(t) - P_n(t) - [f(\bar{x}) - P_n(\bar{x})]\left[\frac{(t-x_0)(t-x_1)\cdots(t-x_k)(t-x_n)}{(\bar{x}-x_0)(\bar{x}-x_1)\cdots(\bar{x}-x_n)}\right]. \tag{6.6}$$

We will now show $g(t)$ satisfies all the hypotheses of the generalized Rolle's theorem so that this theorem can be applied to obtain our result.

First, it follows from the definition of $g(t)$ that it is $(n+1)$ times continuously differentiable in $[a, b]$, since $f(x)$ is so. **(Hypothesis (i) and (ii) are satisfied.)**

Second, we now show that $g(t)$ becomes identically zero at $(n+2)$ points: $x_0, x_1, x_2, \ldots, x_n$ and $\bar{x}$.

Substituting $x = x_k$ in (6.6), we get

$$\begin{aligned} g(x_k) &= f(x_k) - P_n(x_k) - [f(\bar{x}) - P_n(\bar{x})]\left[\frac{(x_k-x_0)(x_k-x_1)\cdots(x_k-x_k)(x_k-x_n)}{(\bar{x}-x_0)\cdots(\bar{x}-x_n)}\right] \\ &= P_n(x_k) - P_n(x_k) - [f(\bar{x}) - P_n(\bar{x})] \times 0 \\ &= 0 \quad (\text{Since } f(x_k) = P_n(x_k)). \end{aligned} \tag{6.7}$$

(Note that the numerator of the fraction appearing above contains the term $(x_k - x_k) = 0$). Thus, $g(x_k) = 0$ at $(n+1)$ distinct points: $x_0, x_1, \ldots, x_n$.

Furthermore, at $x = \bar{x}$, we have

$$\begin{aligned} g(\bar{x}) &= f(\bar{x}) - P_n(\bar{x}) - [f(\bar{x}) - P_n(\bar{x})]\left[\frac{(\bar{x}-x_0)\cdots(\bar{x}-x_n)}{(\bar{x}-x_0)\cdots(\bar{x}-x_n)}\right] \\ &= f(\bar{x}) - P_n(\bar{x}) - f(\bar{x}) + P_n(\bar{x}) \\ &= 0 \end{aligned} \tag{6.8}$$

Thus, $g(t)$ becomes identically zero at $(n+2)$ distinct points: $x_0, x_1, \ldots, x_n$, and $\bar{x}$. **(Hypothesis (iii) is satisfied.)**

Since $g(t)$ satisfies all the hypotheses of the **Generalized Rolle's theorem**, we are now ready to apply this theorem to $g(t)$.

According to this theorem, there exists a number $\xi(\bar{x})$ in (a, b) such that $g^{(n+1)}(\xi) = 0$.

Let's compute $g^{(n+1)}(t)$ now. From (6.6) we have

$$g^{(n+1)}(t) = f^{(n+1)}(t) - P_n^{(n+1)}(t) - [f(\bar{x}) - P_n(\bar{x})]\frac{d^{n+1}}{dt^{n+1}}\left[\frac{(t-x_0)(t-x_1)\cdots(t-x_n)}{((\bar{x}-x_0)(\bar{x}-x_1)\cdots(\bar{x}-x_n)}\right] \quad (6.9)$$

Again, since the expression $(\bar{x}-x_0)(\bar{x}-x_1)\cdots(\bar{x}-x_n)$ is just a number, we obtain

$$\frac{d^{n+1}}{dt^{n+1}}\left[\frac{(t-x_0)(t-x_1)\cdots(t-x_n)}{(\bar{x}-x_0)(\bar{x}-x_1)\cdots(\bar{x}-x_n)}\right] = \frac{1}{(\bar{x}-x_0)(\bar{x}-x_1)\cdots(\bar{x}-x_n)}\cdot\frac{d^{n+1}}{dt^{n+1}}[(t-x_0)(t-x_1)\cdots(t-x_n)]$$

Since $(t-x_0)(t-x_1)\cdots(t-x_n)$ is a polynomial in t of degree $(n+1)$, it can be shown [**Exercise**] by **induction** that

$$\frac{d^{n+1}}{dt^{n+1}}(t-x_0)(t-x_1)\cdots(t-x_n) = (n+1)!$$

So,

$$\frac{d^{n+1}}{dt^{n+1}}\left(\frac{(t-x_0)(t-x_1)\cdots(t-x_n)}{(\bar{x}-x_0)(\bar{x}-x_1)\cdots(\bar{x}-x_n)}\right) = \frac{(n+1)!}{(\bar{x}-x_0)(\bar{x}-x_1)\cdots(\bar{x}-x_n)}$$

Also, $P_n^{(n+1)}(t) = 0$, because P_n is a polynomial of degree at most n and thus, its $(n+1)$th derivative must be zero.

Thus, from (6.9), we have

$$g^{(n+1)}(t) = f^{(n+1)}(t) - [f(\bar{x}) - P_n(\bar{x})] \times \frac{(n+1)!}{(\bar{x}-x_0)(\bar{x}-x_1)\cdots(\bar{x}-x_n)}.$$

Substituting $t = \xi$ in the last expression, we have

$$g^{(n+1)}(\xi) = f^{(n+1)}(\xi) - \frac{(f(\bar{x}) - P_n(\bar{x}))}{(\bar{x}-x_0)\cdots(\bar{x}-x_n)}(n+1)! \quad (6.10)$$

Setting $g^{(n+1)}(\xi) = 0$, from (6.10), we have

$$E_n(\bar{x}) = f(\bar{x}) - P_n(\bar{x}) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(\bar{x}-x_0)\cdots(\bar{x}-x_n).$$

□

Remark: To obtain the polynomial interpolation error using the above theorem, we need to know the $(n+1)^{\text{th}}$ derivative of the $f(x)$ or its absolute maximum value on the interval $[a,b]$. Since in practice this value is hardly known, this error formula is of limited use only. However, knowing an upper bound of $f^{(n+1)}(x)$, one can compute from Theorem 6.7, the maximum value (absolute) of interpolation, which can be of some practical value.

Computing the Maximum Interpolation Error

Step 1. Compute $f^{(n+1)})(x)$

Step 2. Compute $M = \max_{x_0 \le x \le x_n} |f^{(n+1)}(x)|$

Step 3. Compute $N = \max\limits_{x_0 \le x \le x_n} |(x-x_0)(x-x_1)\cdots(x-x_n)|$

Step 4. Compute the maximum interpolation error: $E_n(\overline{x}) = \dfrac{MN}{(n+1)!}$.

A special case: Maximum Linear Interpolation Error

It can be shown [**Exercise**] that for linear interpolation with x_0 and x_1 as nodes:

$$N = \max_{x_0 \le x \le x_1} |(x-x_0)(x-x_1)| = \frac{(x_1-x_0)^2}{4}$$

Thus, in this case

$$|E_n(\bar{x})| \le \frac{M}{8}(x_1-x_0)^2$$

where $|f''(x)| \le M$ on $[x_0, x_1]$.

So, the **maximum absolute error for linear interpolation** is:

$$\frac{M}{8}(x_1-x_0)^2, \text{ where } M = \max_{x_0 \le x \le x_1} |f(x)|.$$

Example 6.8

Find the maximum error for linear interpolation of $f(x) = \frac{1}{x}$ using $x_0 = 2$, and $x_1 = 4$, as nodes.

Input Data:

$$\begin{cases} \text{The degree of the interpolating polynomial: } n = 1. \\ \text{The nodes: } x_0 = 2, x_1 = 4. \end{cases}$$

Solution.

Step 1. Compute $f''(x) = -\frac{1}{x^2}$.

Step 2. Compute $M = \max\limits_{2 \le x \le 4} |f''(x)| = \dfrac{1}{4}$.

Step 3. Compute $N = \max\limits_{2 \le x \le 4} |(x - x_0)(x - x_1)| = \dfrac{(x_1 - x_0)^2}{4} = 1$.

Step 4. Compute the maximum error:

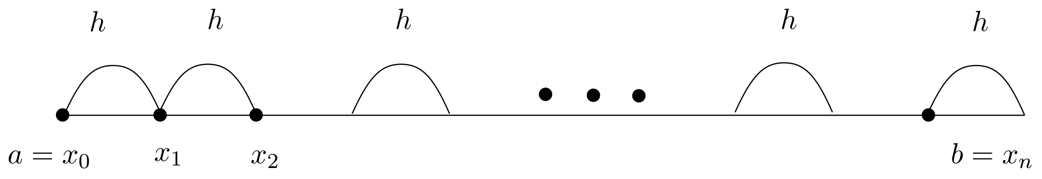
$$\frac{MN}{(n+1)!} = \frac{1}{4} \times 1 \times \frac{1}{2} = \frac{1}{8}.$$

That is, the **maximum absolute error** is $\frac{1}{8}$.

6.8 Simplification of the Error Bound for Equidistant Nodes

*The error formula in **Theorem 6.7** can be simplified in case the tabulated points (nodes) are* equally spaced*; because, in this case it is possible to obtain a nice bound for the expression: $(\bar{x} - x_0)(\bar{x} - x_1) \cdots (\bar{x} - x_n)$.*

Suppose the nodes are equally spaced with spacing h; that is $x_{i+1} - x_i = h$.



Then it can be shown [**Exercise**] that

$$|(\bar{x} - x_0)(\bar{x} - x_1) \cdots (\bar{x} - x_n)| \le \frac{h^{n+1}}{4} n! \qquad \text{on } [x_0, x_n].$$

If we also assume that $|f^{(n+1)}(x)| \le M$, in this interval, then we have

$$|E_n(\bar{x})| = |f(\bar{x}) - P_n(\bar{x})| \le \frac{M}{(n+1)!} \, \frac{h^{n+1}}{4} \, n! = \frac{Mh^{n+1}}{4(n+1)}. \tag{6.11}$$

Computing Maximum Interpolation Error with Equidistant Nodes

Step 0. Set $x_{i+1} - x_i = h, \; i = 0, 1, 2, \ldots, n-1$.

Step 1. Compute $f^{(n+1)}(x)$, the $(n+1)$th derivative of $f(x)$.

Step 2. Compute $\max_{x_0 \le x \le x_n} |f^{(n+1)}(x)| = M$.

Step 3. Compute the maximum interpolation error: $\frac{Mh^{n+1}}{4(n+1)}$.

Example 6.9

Suppose a table of values for $f(x) = \cos x$ has to be prepared in $[0, 2\pi]$ with equally spaced nodes h, using **linear interpolation**, with an error of interpolation of at most 5×10^{-7}. How small should h be?

Input Data:

$$\begin{cases} f(x) = \cos x \\ \text{The degree of the interpolating polynomial: } n = 1. \\ \text{The error tolerance: } \epsilon = 5 \times 10^{-7}. \\ \text{The end points of the interval: } x_0 = 0, x_n = 2\pi. \\ \text{The nodes are equally spaced with spacing } h. \end{cases}$$

Problem to be solved: Find h such that the maximum absolute error is less than ϵ.

Formula to be used:
$$\begin{cases} |E_1(x)| \le \dfrac{Mh^2}{8}, \text{ where} \\ M = \max\limits_{0 \le x \le 2\pi} |f''(x)|. \end{cases}$$

Solution.

Step 1. Compute M:

$$f(x) = \cos x, \quad f'(x) = -\sin x, \quad f^2(x) = f''(x) = -\cos x.$$
$$\text{So, } M = \max |f''(x)| = 1, \text{ for } 0 \le x \le 2\pi$$

Step 2. Compute maximum absolute error: $M\frac{h^2}{8} = \frac{h^2}{8}$

Step 3. Compute h:

Since the maximum error cannot exceed $\epsilon = 5 \times 10^{-7}$, we must have: $\frac{h^2}{8} \le 5 \times 10^{-7} = \frac{1}{2} \times 10^{-6}$. That is, $h \le 0.002$.

Example 6.10

Suppose a table is to be prepared for the function $f(x) = \sqrt{x}$ on $[1, 2]$ with equal spacing, h.

Determine h such that the interpolation with a polynomial of degree 2 will give an accuracy of $\epsilon = 5 \times 10^{-8}$.

Input Data:

$$\begin{cases} f(x) = \sqrt{x} \\ \text{The degree of the interpolating polynomial: } n = 2. \\ \text{The error tolerance: } \epsilon = 5 \times 10^{-8}. \\ \text{The end points of the interval: } a = 1, b = 2. \\ \text{The nodes are equally spaced with spacing } h. \end{cases}$$

Problem to be solved: Find h such that the maximum absolute error is less than ϵ.

Formula to be used: $|E_2(x)| \leq M\dfrac{h^3}{12}$, where $M = \max|f^{(3)}(x)|$, for $1 \leq x \leq 2$.

Solution.

Step 1. Compute M: $f^{(3)}(x) = \frac{3}{8}x^{\frac{-5}{2}}$,

$$M = \left|f^{(3)}(x)\right| \leq \frac{3}{8}, \quad \text{for } 1 \leq x \leq 2.$$

Step 2. Compute the maximum absolute error: $M\dfrac{h^3}{12} = \dfrac{3}{8} \times \dfrac{h^3}{12} = \dfrac{1}{32}h^3$.

Step 3. Compute h to have an accuracy of $\epsilon = 5 \times 10^{-8}$:

$$\frac{1}{32}h^3 < 5 \times 10^{-8} \text{ or } h^3 < 160 \times 10^{-8}.$$

That is,

$$h < \sqrt[3]{160 \times 10^{-8}} = 0.017.$$

6.9 Newton Interpolation

Sometimes, it may not be necessary to use the whole given data set of $(n+1)$ points, to compute the interpolating polynomial $P_n(x)$ to achieve a desired accuracy of an interpolated value of $f(x)$. A lower-order polynomial may just be enough. Also, *higher order polynomials may not always give a better accuracy* (see **Section 6.11**).

In such situations, one would like to have an interpolating scheme which will recursively compute the polynomials $P_0(x), P_1(x), P_2(x), \ldots$, etc., so that as soon as a polynomial yielding a desired accuracy is obtained, one can stop.

Newton's idea is:

- Start with a small number of data $(x_0, y_0), \dots, (x_k, y_k)$ (in principle, one can even start with (x_0, y_0)), and construct the associated interpolating polynomial $P_k(x)$ passing through these data set. Test if $P_k(x)$ gives an acceptable accuracy.
- If not, add the point (x_{k+1}, y_{k+1}) to the existing data set and construct the associated interpolating polynomial $P_{k+1}(x)$ by making use of the work done for $P_k(x)$. Test if $P_{k+1}(x)$ gives a desirable accuracy.
- If not, continue the process by adding one data at a time until satisfied.

Implementation of Newton's Idea:

One way to implement the ideas is to generate the polynomials $\{P_k(x)\}$ of successive higher degrees in the following recursive fashion:

$$\begin{aligned}
P_0(x) &= a_0 \\
P_1(x) &= P_0(x) + a_1(x - x_0) \\
P_2(x) &= P_1(x) + a_2(x - x_0)(x - x_1) \\
&\vdots \\
P_k(x) &= P_{k-1}(x) + a_k(x - x_0(x - x_1)(x - x_{k-1}),
\end{aligned}$$

where the constants $a_0, a_1, \dots, a_k$, called the **Newton coefficients**, need to be found so that each $P_k(x)$ interpolates at the nodes: $x_0, x_1, \dots, x_k$.

Note that $P_k(x)$ can be written as:

$$P_k(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_k(x - x_0)(x - x_1) \cdots (x - x_{k-1}).$$

That is, $\{1, (x - x_0), (x - x_0)(x - x_1), \dots, (x - x_0)(x - x_1) \cdots (x - x_{k-1})\}$ form the basis of $P_k(x)$.

Note: that each polynomial $P_k(x)$ has the following properties:

(i) it is of degree k.

(ii) it is formed out of $P_{k-1}(x)$.

We will now describe the following *equivalent ways* to obtain the Newton coefficients:

(a) via solution of a lower triangular system.

(b) using **divided differences**.

(c) using forward differences (when the nodes are equally spaced).

6.9.1 Generating the Newton Coefficients by Solution of a Lower Triangular System

Using the fact that the polynomials $\{P_k(x)\}$ interpolate at $x_0, x_1, \ldots, x_k$, it immediately follows that $a_0, a_1, \ldots, a_k$ satisfy the following triangular system:

$$\begin{aligned}
&a_0 &&= f_0\\
&a_0 + a_1(x_1 - x_0) &&= f_1\\
&a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) &&= f_2\\
&\vdots && \vdots\\
&a_0 + a_1(x_k - x_0) + \cdots + a_k(x_k - x_0)(x_k - x_1)\cdots(x_k - x_{k-1}) &&= f_k
\end{aligned}$$

The solution of the above system by **forward substitution** (Chapter 5) yields:

$$\boxed{a_0 = f_0}$$

$$a_1(x_1 - x_0) = f_1 - f_0 \Longrightarrow \boxed{a_1 = \frac{f_1 - f_0}{x_1 - x_0}}$$

$$a_2(x_2 - x_0)(x_2 - x_1) = f_2 - f_0 - a_1(x_2 - x_0) = f_2 - f_0 - \frac{f_1 - f_0}{x_1 - x_0}(x_2 - x_0)$$

(Substituting the value of a_1 from the last equation)

$$\Longrightarrow a_2 = \frac{f_2 - f_0}{(x_2 - x_0)(x_2 - x_1)} - \frac{(f_1 - f_0)(x_2 - x_0)}{(x_1 - x_0)(x_2 - x_0)(x_2 - x_1)}$$

$$= \frac{f_2 - f_1}{(x_2 - x_0)(x_2 - x_1)} - \frac{f_1 - f_0}{(x_1 - x_0)(x_2 - x_1)}$$

$$= \frac{\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0}.$$

Thus, $\boxed{a_2 = \frac{\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0}}$, etc.

An expression for a_k can also be written similarly. However, we will wait until we introduce the notion of **divided differences** in **Section 6.9.4**.

6.9.2 Evaluating the Newton Interpolating Polynomial

The Newton polynomial $P_k(x)$ in the form:

$$P_k(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_k(x - x_0)(x - x_1)\cdots(x - x_{k-1})$$

can be evaluated at a point $x = z$ by *modifying the standard Horner's method* as follows:

Substituting the value of $x = z$ in the above expression of $P_k(x)$, we obtain

$$P_k(z) = a_0 + a_1(z - x_0) + a_2(z - x_0)(z - x_1) + \cdots + a_k(z - x_0)(z - x_1)\cdots(z - x_{k+1}),$$

which can be written in the following **nested form**:

$$P_k(z) = a_0 + (z - x_0)\left(a_1 + (z - x_1)(a_2 + (z - x_2)(\cdots(x_{k-1} + x_k(z - x_{k-1})))\cdots)\right).$$

For example, for $\mathbf{k} = \mathbf{2}$: $P_2(z) = a_0 + a_1(z - x_0) + a_2(z - x_0)(z - x_1)$ can be written in the form: $P_2(z) = a_0 + (z - x_0)(a_1 + a_2(z - x_1))$.

The above computations can be arranged in the algorithmic form as follows:

Algorithm 6.11 (Horner's Method to Evaluate Newton Polynomial $P_k(x)$).

Inputs: $\begin{cases} \text{(i) The coefficients of the Newton polynomial } P_k(x): a_0, a_1, \ldots, a_k \\ \text{(ii) The point } x = z \text{ where } P_k(x) \text{ needs to be evaluated.} \end{cases}$

Output: The value of $P_k(x)$ at $x = z$, $P_k(z)$.

```
Step 0. Set $b_k = a_k$.
Step 1. Compute $b_{k-1}$ through $b_1$ recursively:
        For $i = k-1, k-2, \ldots, 0$ do
            $b_i = a_i + (z - x_i)b_{i+1}$
        End
Step 2. $P_k(z) = b_0$.
```

6.9.3 Divided Differences

Traditionally, the Newton coefficients a_k's are written in terms of **divided differences**, which are defined and denoted as follows:

- Zeroth Divided Differences: $f[x_i] = f(x_i) = f_i, \quad i = 0, 1, \ldots$.
- 1st Divided Differences: $f[x_i, x_{i+1}] = \dfrac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}, \quad i = 0, 1, \ldots$.

- 2nd Divided Differences: $f[x_i, x_{i+1}, x_{i+2}] = \dfrac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}, \quad i = 0, 1, \ldots.$
- k-th Divided Differences: $f[x_i, x_{i+1}, \ldots, x_{i+k}] = \dfrac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i},$

  $i = 0, 1, \ldots, n-k.$

It follows from above that *each divided difference can be obtained from the two previous ones of lower orders*. For example, $f[x_0, x_1, x_2, x_3]$ can be computed from $f[x_1, x_2, x_3]$ and $f[x_0, x_1, x_2]$. These differences can be computed and arranged in a systematic way in the form of the following table (illustrated for $\mathbf{n = 4}$).

6.9.4 The Newton Coefficients using Divided Differences and Newton Divided Differences Form

Using the expressions of the coefficients, $a_0, a_1, \ldots, a_k$ obtained in Section 6.9.1, we can now write them in terms of the divided differences, as follows:

$$\begin{aligned}
a_0 &= f_0 = f[x_0] \\
a_1 &= \tfrac{f_1 - f_0}{x_1 - x_0} = \tfrac{f[x_1] - f[x_0]}{x_1 - x_0} = f[x_0, x_1] \\
a_2 &= \tfrac{\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0} = \tfrac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = f[x_0, x_1, x_2]
\end{aligned}$$

In general, $\boxed{a_k = f[x_0, x_1, \ldots, x_k)}$, $k = 0, 1, 2, \ldots, n.$

Newton Divided Difference Interpolation Formula

$$\begin{aligned}
P_0(x) &= f_0 \\
P_k(x) &= P_{k-1}(x) + f[x_0, x_1, \ldots, x_k](x - x_0)(x - x_1)\cdots(x - x_{k-1}), \\
&\quad k = 1, 2, \ldots, n.
\end{aligned}$$

Generating the Divided Differences

Denoting $D_{i,0} = f_i$, $i = 0, 1, \ldots, n$ and $D_{ij} = f[x_{i-j}, x_{i-j+1}, \ldots, x_i]$, $j \leq i$, it is easy to see that

$$D_{ij} = \frac{D_{i,j-1} - D_{i-1,j-1}}{x_i - x_j}$$

Table 6.1: Table of Divided Differences $\mathbf{n = 4}$

x	$f(x)$	1^{st} Divided Difference	2^{nd} Divided Difference	3^{rd} Divided Difference	4^{th} Divided Difference
x_0	f_0				
		$f[x_0, x_1]$			
x_1	f_1	$= \frac{f_1 - f_0}{x_1 - x_0}$	$f[x_0, x_1, x_2]$		
		$f[x_1, x_2]$	$= \frac{f[x_1,x_2] - f[x_1,x_0]}{x_2 - x_0}$		
				$f[x_0, x_1, x_2, x_3]$	
x_2	f_2	$= \frac{f_2 - f_1}{x_2 - x_1}$	$f[x_1, x_2, x_3]$	$= \frac{f[x_1,x_2,x_3] - f[x_0,x_1,x_2]}{x_3 - x_4}$	
		$f[x_2, x_3]$	$= \frac{f[x_2,x_3] - f[x_1,x_2]}{x_3 - x_1}$		$f[x_0, x_1, x_2, x_3, x_4]$
				$f[x_1, x_2, x_3, x_4]$	$= \frac{f[x_1,x_2,x_3,x_4] - f[x_0,x_1,x_2,x_3]}{x_4 - x_0}$
x_3	f_3	$= \frac{f_3 - f_2}{x_3 - x_2}$	$f[x_2, x_3, x_4]$	$= \frac{f[x_2,x_3,x_4] - f[x_1,x_2,x_3]}{x_4 - x_1}$	
		$f[x_3, x_4]$	$= \frac{f[x_3,x_4] - f[x_2,x_3]}{x_4 - x_3}$		
x_4	f_4	$= \frac{f_4 - f_3}{x_4 - x_3}$			

Thus, we can generate the divided differences recursively by the following algorithm.

Algorithm 6.12 (Algorithm for Generating Divided Differences).

Inputs: The distinct numbers $x_0, x_1, \ldots, x_n$ and the corresponding functional values $f_0, f_1, \ldots, f_n$.

Output: Entries of the divided difference table.

Step 1. (Initialization). Set $D_{i,0} = f_i$, $i = 0, 1, 2, \ldots, n$.

```
Step 2. For $i = 1, 2, \ldots, n$ do
            For $j = 1, 2, \ldots i$ do
                $D_{ij} = \frac{D_{i,j-1} - D_{i-1,j-1}}{x_i - x_{i-j}}$
            End
        End
```

Example 6.13

Given the following table of values:

x	1	1.5	2.5
$f(x)$	0	0.4055	0.9163

(a) Compute the coefficients of the Newton interpolating polynomial $P_2(x)$ using

(i) lower-triangular system associated with the Newton basis

(ii) divided differences

(b) Interpolate $f(1.9)$.

Input Data:

$$\begin{cases} \text{Nodes: } x_0 = 1, x_1 = 1.5, x_2 = 2.5 \\ \text{Functional Values: } f_0 = 0, f_1 = 0.4055, f_2 = 0.9163 \\ P_0 = f_0 = 0. \\ \text{Interpolating Point: } X_I = 1.9 \end{cases}$$

Solution (a): Let $P_2(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$

(i) Computations of a_0, a_1, and a_2 using **Lower-Triangular System**:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & x_1 - x_0 & 0 \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \end{pmatrix}$$

$$= \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0.5 & 0 \\ 1 & 1.5 & 1.5 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 0 \\ 0.4055 \\ 0.9163 \end{pmatrix}}_{b}$$

(ii) Solution of the lower-triangular system by forward elimination: $x = \begin{pmatrix} 0 \\ 0.8110 \\ -0.2002 \end{pmatrix}$.

So, $\begin{cases} a_0 = 0 \\ a_1 = 0.8110 \\ a_2 = -0.2002 \end{cases}$.

(iii) Using Divided Differences (See Section 6.9.4):

$$\begin{aligned}
a_0 &= f[x_0] = f_0 = 0 \\
a_1 &= f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{f_1 - f_0}{x_1 - x_0} = 0.8118 \\
a_2 &= f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_2, x_1]}{x_2 - x_0} \\
&= \frac{\dfrac{f[x_2] - f[x_1]}{x_2 - x_1} - \dfrac{f[x_1] - f[x_0]}{x_1 - x_0}}{x_2 - x_0} \\
&= \frac{\dfrac{f_2 - f_1}{x_2 - x_1} - \dfrac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0} = \frac{0 - 5108 - 0.81108}{1.5} = -0.2002
\end{aligned}$$

Remark: Note that the coefficients a_0, a_1, a_2 obtained by both methods are the same, which was expected, because the *interpolating polynomial is unique.*

Solution (b).

(i) **Recursive Construction of the Newton Polynomials using Divided Differences**

$$\begin{aligned}
P_0(x) &= f_0 = 0 \\
P_1(x) &= P_0(x) + (x - x_0) f[x_0, x_1] = 0 + (x - 1) \times 0.8118 \\
&= 0.8118(x - 1) \textbf{ (using the value of } f[x_0, x_1] \textbf{ from solution of Part (a)(iii)).} \\
P_2(x) &= P_1(x) + (x - x_0)(x - x_1) f[x_0, x_1, x_2] \\
&= 0.8118(x - 1) + (x - 1)(x - 1.5) \times (-0.2002) \\
&\textbf{(using the value of } f[x_0, x_1, x_2] \textbf{ from solution of Part (a)(iii)).}
\end{aligned}$$

(ii) **Interpolation of $f(1.9)$ using $P_2(x)$:**

$$\begin{aligned}
P_1(1.9) &= 0.8118(0.9) = 0.7306 \\
P_2(1.9) &= P_1(1.9) + (1.9 - 1)(1.9 - 1.5) \times (-0.2002) = 0.6585
\end{aligned}$$

(iii) **Accuracy Check:** The tabulated values of $f(x)$ correspond to $f(x) = \ln(x)$.

Exact Value of $\ln(1.5) = 0.6419$.

Absolute Error: $|P_2(1.9) - \ln(1.5)| = |0.6585 - 0.6419| = 0.0166$.

Example 6.14

Given the following table:

x	1	1.5	2	3	3.5
$f(x)$	0	0.17609	0.30103	0.4772	0.54407

A. Construct the table of divided differences.

B. Using the divided differences, construct $P_3(x)$ and $P_4(x)$ recursively and use them to interpolate $f(2.5)$.

Input Data:

$$\begin{cases} \text{Nodes: } x_0 = 1, x_1 = 1.5, x_2 = 2, x_3 = 3, x_4 = 3.5 \\ \text{Function Values: } f_0 = 1, f_1 = 0.17609, f_2 = 0.30103, f_3 = 0.4772, f_4 = 0.54407 \\ \text{Degrees of the Interpolating Polynomials: } k = 3 \text{ and } k = 4 \end{cases}$$

Formula to be Used: $P_k(x) = P_{k-1}(x) + (x - x_0)(x - x_1)\cdots(x - x_{k-1})f[x_0, x_1, \ldots, x_k], k = 1, 2, 3, 4.$

Solution A. The Divided Differences Table for the Data of Example 6.14

i	x_i	f_i	$1st$ diff.	$2nd$ diff.	$3rd$ diff.	$4th$ diff.
0	1.0	0				
			$\boxed{0.3522}$			
1	1.5	0.17609		$\boxed{-0.1023}$		
			0.2499		$\boxed{0.0265534}$	
2	2.0	0.30103		−0.0491933		$\boxed{-0.006409}$
			0.1761		0.01053	
3	3.0	0.47712		−0.0281333		−0.002169
			0.1339		0.005107	
4	3.5	0.54407		−0.01792		
			0.11598			

Solution B.

(i) **Recursive Construction of the Newton Polynomials using Divided Differences for Example 6.14**

$$\begin{aligned}
\text{Set}\quad & P_0(x) &&= f_0 = 0\\
\mathbf{k=1}:\quad & P_1(x) &&= P_0(x) + f[x_0,x_1](x-x_0)\\
& &&= 0 + 0.3522 \times (x-1)(x-1.5) = 0.3522(x-1)\\
\mathbf{k=2}:\quad & P_2(x) &&= P_1(x) + f[x_0,x_1,x_2](x-x_0)(x-x_1)\\
& &&= 0.3522(x-1) - 0.1023(x-1)(x-1.5)\\
\mathbf{k=3}:\quad & P_3(x) &&= P_2(x) + f[x_0,x_1,x_2,x_3](x-x_0)(x-x_1)(x-x_2)\\
& &&= 0.3522(x-1) - 0.1023(x-1)(x-1.5) + 0.0265534(x-1)(x-1.5)(x-2)\\
\mathbf{k=4}:\quad & P_4(x) &&= P_3(x) + f[x_0,x_1,x_2,x_3,x_4](x-x_0)(x-x_1)(x-x_2)(x-x_3)\\
& &&= 0.3522(x-1) - 0.1023(x-1)(x-1.5) + 0.0265534(x-1)(x-1.5)(x-2)\\
& &&\quad -0.006409(x-1)(x-1.5)(x-2)(x-3)
\end{aligned}$$

Interpolation of $f(2.5)$ using $P_3(x)$ and $P_4(x)$:

$$\begin{aligned}
P_3(2.5) &= (2.5-1.0)(0.35222) + (2.5-1.0)(2.5-1.5)(-0.1023)\\
&\quad + (2.5-1.0)(2.5-1.5)(2.5-2.0)(.0265534)\\
&= .52827 - .15345 + .019915\\
&= \boxed{0.394795}
\end{aligned}$$

Similarly, with $n = 4$, we have

$$\begin{aligned}
P_4(2.5) &= P_3(2.5) + (2.5-1.0)(2.5-1.5)(2.5-2.0)(2.5-3.0)(-.006409)\\
&= 0.394795 + .002403 = \boxed{0.397198}.
\end{aligned}$$

Note that $P_4(2.5) - P_3(2.5) = \boxed{0.002403}$.

(Note that in computing $P_4(2.5)$, $P_3(2.5)$ computed previously has been gainfully used).

((iii) **Accuracy Check:** The above is a table for $f(x) = \log_{10}(x)$.

$\log(2.5)$ (*correct up to 5 decimal places*) is 0.39794.

Absolute Error: $|\log(2.5) - P_4(2.5)| = 7.4200 \times 10^{-4}$.

Example 6.15

Suppose the data $(4, 0.60206)$ is added to the table of Exercise 6.15. Using computations of Example 6.14, now find $P_5(x)$ and use it to interpolate $f(2.5)$ and compute the absolute error.

Solution. The 5th divided difference

$$
\begin{aligned}
f[x_0,x_1,x_2,x_3,x_4,x_5] &= \frac{f[x_1,x_2,x_3,x_4,x_5]-f[x_0,x_1,x_2,x_3,x_4]}{x_5-x_0}\\
P_5(x) &= P_4(x)+f[x_0,x_1,x_2,x_3,x_4,x_5]\times(x-x_0)(x-x_1)\cdots(x-x_4)\\
&= -0.006409(x-1)(x-1.5)(x-2)(x-3)\\
&\quad -0.001413(x-1)(x-1.5)(x-2)(x-3)(x-3.5)
\end{aligned}
$$

Interpolation:

$$
\begin{aligned}
P_5(2.5) &= P_4(2.5)-0.001413(1.5)(1)(0.5)(-0.5)(-1)\\
&= 0.39668.
\end{aligned}
$$

Absolute Error: $|\log(2.5)-P_5(2.5)| = 0.001271.$

A Relationship Between n^{th} Divided Difference and the n^{th} Derivative

The following theorem shows how the n^{th} derivative of a function $f(x)$ is related to the n^{th} divided difference. The proof is left as an [**Exercise**].

Theorem 6.16. Suppose that

(i) $f(x)$ is n times continuously differentiable in $[a,b]$

(ii) $x_0,x_1,\ldots,x_n$ are $(n+1)$ distinct numbers in $[a,b]$

Then the nth divided difference and the nth derivative of $f(x)$, $f^{(n)}(x)$, are related by

$$f[x_0,x_1,\ldots,x_n] = \frac{f^{(n)}(\xi)}{n!}$$

where ξ is a number between a and b.

6.9.5 The Newton Interpolation with Equally Spaced Nodes: Forward Differences

We now show that if the nodes are equally spaced, then the formulas for the Newton interpolating polynomials can be computationally simplified, because in this case:

- The expression $\Psi_{k-1}(x) = (x-x_0)(x-x_1)\ldots(x-x_{k-1})$ can be written in a compact way
  and
- The divided differences can be computed without any divisions by the associated factors.

Compact Representation of $\Psi_{k-1}(x)$

Set $x - x_0 = sh$.

Then

$$x - x_1 = x - x_0 + x_0 - x_1 = (x - x_0) - (x_1 - x_0) = sh - h = (s-1)h$$

Similarly, $x - x_2 = (s-2)h$.

In general, $x - x_i = (s-i)h, \quad i = 0, 1, 2, \ldots, (k-1)$.

So,

$$\begin{aligned} \Psi_{k-1}(x) &= (x - x_0)(x - x_1) \ldots (x - x_{k-1}) \\ &= (sh)(s-1)h \ldots (s-k+1)h \\ &= h^k (s)(s-1) \ldots (s-k+1). \end{aligned}$$

Invoke now the **binomial notation**:

$$\binom{s}{k} = \frac{s(s-1)(s-2)\ldots(s-k+1)}{k!}$$

Then using this notation, $\Psi_{k-1}(x)$ can be compactly written as:

$$\boxed{\Psi_{k-1}(x) = h^k k! \binom{s}{k}} \tag{6.12}$$

Simplification of the Divided Differences: Forward Differences

We now show how $f[x_0, x_1, \ldots, x_k]$ can be computed without divisions by the factor $(x_k - x_0)$. To do so, we introduce the notion of **forward difference** and show how divided differences can be expressed in terms of forward differences.

Define

$$\Delta f_i = f(x_{i+1}) - f(x_i), \quad i = 0, 1, 2, \ldots \tag{6.13}$$

So, for $i = 0$, we have

$$\begin{aligned} &\Delta f_0 = f(x_1) - f(x_0) \\ &\text{Thus,} \qquad f[x_0, x_1] = \tfrac{f(x_1) - f(x_0)}{x_1 - x_0} = \tfrac{\Delta f_0}{h}. \end{aligned} \tag{6.14}$$

$$\begin{aligned}\Delta^2 f_0 &= \Delta(\Delta f_0) = \Delta(f(x_1) - f(x_0)) \\ &= \Delta f(x_1) - \Delta f(x_0) \\ &= f(x_2) - f(x_1) - f(x_1) + f(x_0) \\ &= f(x_2) - 2f(x_1) + f(x_0)\end{aligned} \tag{6.15}$$

Again,

$$\begin{aligned}f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{(x_2 - x_0)} \\ &= \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{(x_2 - x_0)} \\ &= \frac{f(x_2) - 2f(x_1) + f(x_0)}{h \times 2h}\end{aligned} \tag{6.16}$$

So, using (6.15) in (6.16), we have

$$\boxed{f[x_0, x_1, x_2] = \tfrac{\Delta^2 f_0}{2h^2}.} \tag{6.17}$$

Formula (6.17) can be immediately generalized as shown below in Theorem 6.17.

Theorem 6.17 (Relationship Between Divided and Forward Differences).

$$f[x_0, x_1, \ldots, x_k] = \frac{1}{k!h^k}\Delta^k f_0. \tag{6.18}$$

Proof. By induction on k.

- For $k = 0$, the result is trivially true.
- The relation (6.14) shows that Theorem is true for $k = 1$.
- The relation (6.17) shows that Theorem is true for $k = 2$.

In the following, we will show that Theorem is true for any k. This will be done by **induction**.

First, assume that the result is true $k = m$. Then we will show that the result is also true for $k = m + 1$.

By definition, we have

$$f[x_0, \ldots, x_{m+1}] = \frac{f[x_1, \ldots, x_{m+1}] - f[x_0, \ldots, x_m]}{x_{m+1} - x_0}$$

Now, because of our assumption that Theorem is true for $k = m$, we have

$$f[x_1, \ldots, x_{m+1}] = \frac{\Delta^m f_1}{m! h^m}$$

and

$$f[x_0, \ldots, x_m] = \frac{\Delta^m f_0}{m! h^m}.$$

Also, by definition

$$x_{m+1} - x_0 = (m+1)h.$$

So,

$$f[x_0, \ldots, x_{m+1}] = \frac{\left(\frac{\Delta^m f_1}{m! h^m} - \frac{\Delta^m f_0}{m! h^m}\right)}{(m+1)h}.$$

$$= \frac{\Delta^m (f_1 - f_0)}{m!(m+1)h^{m+1}} \quad = \frac{\Delta^{m+1} f_0}{(m+1)! h^{m+1}}$$

□

Definition 6.18. The numbers $\Delta^k f_i$ are called ***k*th order forward differences** of f at $x = i$.

Summarizing,

$$\begin{aligned} f[x_0] &= f_0 \\ f[x_0, x_1] &= \frac{\Delta f_0}{h} \\ f[x_0, x_1, x_2] &= \frac{\Delta^2 f_0}{2h^2} \\ &\vdots \\ f[x_0, x_1, x_2, \ldots, x_k] &= \frac{\Delta^k f_0}{k! h^k} \end{aligned}$$

Generating the Forward Differences

From the above formula, it is clear that the functional values $f_0, f_1, \ldots, f_n$ are used only to construct the first order forward differences Δf_i, $i = 0, 1, \ldots, n-1$. Thereafter, the k-th order differences $\Delta^k f_i$, $k > 1$ can be generated from $\Delta^{k-1} k_i$.

- First order Forward Differences:

$$\Delta f_i = f_{i+1} - f_i, \quad i = 0, 1, \ldots, n-1$$

- Second-order Forward Differences:

$$\Delta^2 f_i = \Delta f_{i+1} - \Delta f_i, \quad i = 0, 1, \ldots, n-2$$

Table 6.2: The Forward Difference Table with $n = 4$

x	$f(x)$	Δf	$\Delta^2 f$	$\Delta^3 f$	$\Delta^4 f$
x_0	f_0				
		$\rangle\ \Delta f_0 = f_1 - f_0$			
x_1	f_1		$\rangle\ \Delta^2 f_0 = \Delta f_1 - \Delta f_0$		
		$\rangle\ \Delta f_1$		$\rangle\ \Delta^3 f_0 = \Delta^2 f_1 - \Delta^2 f_0$	
x_2	f_2		$\rangle\ \Delta^2 f_1 = \Delta f_2 - \Delta f_1$		$\rangle\ \Delta^4 f_0 = \Delta^3 f_1 - \Delta^3 f_0$
		$\rangle\ \Delta f_2 = f_3 - f_2$		$\rangle\ \Delta^3 f_1 = \Delta^2 f_2 - \Delta^2 f_1$	
x_3	f_3		$\rangle\ \Delta^2 f_2 = \Delta f_3 - \Delta f_2$		
		$\rangle\ \Delta f_3 = f_4 - f_3$			
x_4	f_4				

- Third-order Forward Differences:

$$\Delta^3 f_i = \Delta^2 f_{i+1} - \Delta^2 f_i, \quad i = 0, 1, \ldots, n-3$$

and so on.

These differences can be conveniently arranged in the form of a table as shown below ($n = 4$).

Newton Interpolating Polynomials in Terms of Forward Differences

We are now ready to express the Newton interpolating polynomial

$$P_k(x) = P_{k-1}(x) + f[x_0, x_1, \ldots, x_k]\Psi_{k-1}(x)$$

in terms of forward differences rather than divided differences.

From (6.12), we have

$$\Psi_{k-1}(x) = h_k k! \binom{s}{k}$$

Also from (6.18), we know

$$f[x_0, x_1, \ldots, x_k] = \frac{\Delta^k f_0}{k! h^k}.$$

Substituting these values in the above expression of $P_k(x)$, we obtain

Newton's Forward Difference Interpolation Formula

$$\begin{aligned} P_0(x) &= f_0 \\ P_k(x) &= P_{k-1}(x) + \binom{s}{k} \Delta^k f_0, \qquad k = 1, 2, 3, \ldots, n \end{aligned}$$

Remark: Note that only the differences along the diagonal; namely, $\Delta f_0, \Delta^2 f_2, \ldots, \Delta^k f_n$, are needed to form the interpolating polynomial of degree k. Also, all these entries need not be computed all at one time - they can be generated as needed depending upon the degree of the desired interpolating polynomial. *This is a real computational advantage of the Newton's interpolation over the Lagrange's.*

Algorithm 6.19 (Newton's Forward Difference Interpolation).

```
Inputs:   (i) A positive integer $n$
          (ii) $(n+1)$ equidistant points: $(x_0, f_0)$, $(x_1, f_1)$, ..., $(x_n, f_n)$
          (iii) $h$ = Distance between two points
          (iv) Interpolating point $X_I$, $x_0 \le X_I \le x_n$
Output: $Y_I$ = Interpolated value of $f(X_I)$
Step 0. Set $h = x_{i+1} - x_i$, $i = 0, 1, \ldots, n-1$
Step 1. Set $P_0 = f_0$.
Step 2. For $k = 1, 2, \ldots$ do until satisfied
    2.1 Generate the forward differences $\Delta f_0$, $\Delta^2 f_0$, ..., $\Delta^k f_0$, using the Forward Difference Table.
    2.2 Compute $Y_I = P_k(X_I) = P_{k-1}(X_I) + \binom{s}{k} \Delta^k f_0$.
End
```

Example 6.20

Given the following table for $f(x) = e^x$

x	0	1	2	3	4
$f(x)$	1	2.7183	7.3891	20.0855	54.5982

A. Construct a forward-difference table using all data.

B. Interpolate $f(1.5)$ recursively from $P_1(1.5), P_2(1.5), P_3(1.5)$, and $P_4(1.5)$.

C. Compare the interpolated values with the actual value of $e(1.5)$.

Input Data:

$$\begin{cases} \text{Nodes: } x_0 = 0, x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4 \\ \text{Functional Values: } f_0 = 1, f_1 = 2.7183, f_2 = 7.3891, f_3 = 20,0885, f_4 = 54.5982 \\ \text{Degrees of the Interpolating Polynomials: } k = 1, k = 2, k = 3, k = 4 \\ \text{The Interpolated Point: } x_I = 1.5 \\ \text{The Distance between Two Nodes: } h = 1 \end{cases}$$

Formulas to be Used:

$$\begin{cases} s = \dfrac{x - x_0}{h} \\ P_0(x) = f_0 \\ P_k(x_I) = P_{k-1}(x_I) + \binom{s}{k}\Delta^k f_0 \\ k = 1, 2, 3, 4 \end{cases}$$

Solution A. Construction of Forward Difference Table

Zeroth-Order Differences:

$$\begin{aligned} &f[x_0] = f_0 = 1; \quad f[x_1] = f_1 = 2.7183 \\ &f[x_2] = f_2 = 7.3891; \quad f[x_3] = f_3 = 20.0855 \\ &f[x_4] = f_4 = 54.5982 \end{aligned}$$

First-Order Differences:

$$\begin{aligned} f[x_0, x_1] &= f[x_1] - f[x_0] = 2.7183 - 1 = 1.7183 \\ f[x_1, x_2] &= f[x_2] - f[x_1] = 7.3891 - 2.7183 = 4.6708 \\ f[x_2, x_3] &= f[x_3] - f[x_2] = 20.0855 - 7.3891 = 12.6964 \\ f[x_3, x_4] &= f[x_4] - f[x_3] = 54.5982 - 20.0855 = 34.5127 \end{aligned}$$

Second-Order Differences:

$$\begin{aligned} f[x_0, x_1, x_2] &= f[x_1, x_2] - f[x_0, x_1] = 4.6708 - 1.783 = 2.9525 \\ f[x_1, x_2, x_3] &= f[x_2, x_3] - f[x_1, x_2] = 12.6964 - 4.6708 = 8.0256 \\ f[x_2, x_3, x_4] &= f[x_3, x_4] - f[x_2, x_3] = 34.5127 - 12.6964 = 21.8163 \end{aligned}$$

Third-Order Differences:

$$f[x_0,x_1,x_2,x_3] = f[x_1,x_2,x_3] - f[x_0,x_1,x_2] = 8.0256 - 2.9525 = 5.0731$$
$$f[x_1,x_2,x_3,x_4] = f[x_2,x_3,x_4] - f[x_1,x_2,x_3] = 21.8163 - 8.0256 = 13.7907$$

Fourth-Order Differences:

$$f[x_0,x_1,x_2,x_3,x_4] = f[x_1,x_2,x_3,x_4] - f[x_0,x_1,x_2,x_3] = 13.7907 - 5.0731 = 8.7176$$

The Difference Table for the Data

x	f	Δf	$\Delta^2 f$	$\Delta^3 f$	$\Delta^4 f$
0	1				
		1.7183			
1	2.7183		2.9525		
		4.6708		5.0731	
2	7.3891		8.0256		8.7176
		12.6964		13.7907	
3	20.0855		21.8163		
		34.5127			
4	54.5982				

B. Interpolation at $x = 1.5$

Compute: $s = \frac{X_I - x_0}{h} = \frac{1.5-0}{1} = 1.5.$

Compute the polynomial values recursively:

$$P_1(1.5) = P_0(1.5) + s \times \Delta f_0 = 1 + 1.5 \times 1.7183 = 3.5774$$

$$P_2(1.5) = P_1(1.5) + \binom{s}{2}\Delta^2 f_0 = 4.6846$$

$$P_3(1.5) = P_2(1.5) + \binom{s}{3}\Delta^3 f_0 = 4.4846 + \frac{(1.5)(1.5-1)(1.5-2)}{6} \times 5.0731 = 4.3675$$

$$P_4(1.5) = P_3(1.5) + \binom{s}{4}\Delta^4 f_0 = 4.3675 + \frac{(1.5)(1.5-1)(1.5-2)(1.5-3)}{24} \times 8.7176 = 4.5716$$

C. Error Comparison

(i) $|e^{1.5} - P_3(1.5)| = |4.4817 - 4.3675| = 0.1142$

(ii) $|e^{1.5} - P_4(1.5)| = |4.4817 - 4.5716| = 0.0899$

6.9.6 Interpolation towards the End of the Table: Newton's Backward Difference

While interpolating at some value of x near the end of the difference table, it is logical to reorder the nodes so that the end-differences can be used in computation.

Assume that the nodes are equally spaced with spacing h.

- Set $x_0 \equiv x_k$, where x_k is a point near the end of the table close to $x = XI$.
- Label the nodes above x_0 as $x_{-1}, x_{-2}, \ldots x_{-k}$ and the corresponding diagonal divided differences as $\Delta f_{-1}, \Delta f_{-2}, \ldots, \Delta f_{-k}$.
- Set $s = \dfrac{x - x_0}{h}$.

Then it can be shown [**Exercise**] that the interpolating polynomial $P_k(x)$ of degree k that interpolates at $x_0, x_{-1}, \ldots, x_{-k}$ is given by

$$P_k(x) = f_0 - \binom{-s}{1} \Delta f_{-1} + \binom{-s}{2} \Delta^2 f_{-2} + \cdots + (-1)^k \binom{-s}{k} \Delta^k f_{-k}. \tag{6.19}$$

For the obvious reason, the above equation is called the **Newton backward difference interpolating polynomial**.

Example 6.21

Using the difference table of Example 6.19, interpolate $f(3.5)$ using a four degree interpolating polynomial.

Solution. Since the point $x = 3.5$ is near the end of the table, we take $x_0 = 4$. Then

$$\begin{array}{lll}
s = \frac{3.5-4}{1} & = -.5 & = -\frac{1}{2} \\
x_{-1} = 3, & x_{-2} = 2, & x_{-3} = 1 \\
\Delta f_0 = 54.5982, & \Delta f_{-1} = 34.5127, & \Delta f_{-2} = 21.8163, \\
\Delta f_{-3} = 13.7907, & \Delta f_{-4} = 8.7176. & \\
\binom{-s}{1} & = -s & = \frac{1}{2} \\
\binom{-s}{2} & = \frac{(-s)(-s-1)}{2} & = \frac{-1}{8}
\end{array}$$

6.10 Hermite Interpolation

Given $(n+1)$ distinct points $(x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n)$, the Lagrange or Newton interpolating polynomial $P_n(x)$ matches the function values at the nodes. that is,

$$P_n(x_i) = f_i, \quad i = 0, 1, \ldots, n.$$

Some applications, however, demand that the interpolant be such that not only the functional values are matched, but also the derivatives at the nodes. The procedure of constructing such interpolants is called **Hermite interpolation**.

Hermite Interpolation

Given $(n+1)$ distinct points (x_i, f_i), $i = 0, 1, \ldots, n$

Find the polynomial $H_{2n+1}(x)$ of degree at most $(2n+1)$ such that

(i) $H_{2n+1}(x_i) = f(x_i)$, $i = 0, 1, \ldots, n$

(ii) $H'_{2n+1}(x_i) = f'(x_i)$, $i = 0, 1, \ldots, n$

Definition 6.22. The polynomial $H_{2n+1}(x)$ as defined above is called the **Hermite polynomial**.

Existence and Uniqueness of Hermite Interpolation

Below we show in a constructive way, using Lagrange polynomials, the existence and uniqueness of the Hermite interpolation.

Theorem 6.23 (Uniqueness of Hermite Interpolation). Let $f(x)$ be differentiable in $[a, b]$ and $\{x_i\}_{k=0}^{n}$ are $(n+1)$ distinct numbers in $[a, b]$. Then there is a unique polynomial $H_{2n+1}(x)$ of degree at most $(2n+1)$ such that

(i) $H_{2n+1}(x_i) = f(x_i)$, $i = 0, 1, \ldots, n$

(ii) $H'_{2n+1}(x_i) = f'(x_i)$, $i = 0, 1, \ldots, n$

Proof. The idea is to construct a set of $(2n+1)$ polynomials $\{B_k\}_{k=0}^{n}$ and $\{\hat{B}\}_{k=0}^{n}$ such that these polynomials form a basis of all polynomials of degree at most $(2n+1)$ and write the polynomial $H_{2n+1}(x)$ using this basis. Then $H_{2n+1}(x)$ will have the following form:

$$H_{2n+1}(x) = \sum_{k=0}^{n} f(x_k)B_k(x) + \sum_{k=0}^{n} f'(x_k)\hat{B}_k(x).$$

The polynomials $B_k(x)$ and $\hat{B}_k(x)$ now are chosen appropriately so that $H_{2n+1}(x)$ is an interpolant for both $f(x)$ and $f'(x)$.

Choose

$$B_k(x) = (1 - 2(x - x_k)L_k'(x_k))L_k^2(x)$$

and

$$\hat{B}_k(x) = (x - x_k)L_k^2(x),$$

where $L_k(x)$ is the k-th Lagrange polynomial:

$$L_k(x) = \prod_{\substack{j=0 \\ j\neq k}}^{n} \frac{(x - x_j)}{(x_k - x_j)}$$

and $L_k'(x)$ denotes its first derivative. Since each Lagrangian polynomial $L_k(x)$ is a polynomial of degree at most n, B_k and $\hat{B}_k$ are polynomials of degree at most $(2n + 1)$. Moreover, it is easy to see that the following properties of $B_k(x)$ and $\hat{B}_k(x)$ hold:

$$B_k(x_j) = \begin{cases} 0 & \text{for } j \neq k \\ 1 & \text{for } j = k \end{cases}$$

$$\hat{B}_k(x_j) = 0 \text{ for } j = 0, \dots, n$$

and

$$B_k'(x_j) = 0 \text{ for } j = 0, \dots, n$$

$$\hat{B}_k'(x_j) = \begin{cases} 0 & \text{for } j \neq k \\ 1 & \text{for } j = k \end{cases}$$

In view of these properties of B_k and $\hat{B}_k$, it can be verified [**Exercise**] that $H_{2n+1}(x)$ so defined satisfies the interpolant properties (i) and (ii) of this theorem.

Uniqueness: Uniqueness follows from the fact that $H_{2n+1}(x)$ is interpolant for both $f(x)$ and $f'(x)$. This is left as an [**Exercise**]. □

Generating Hermite Interpolating Polynomial

Given $\{x_k, f_k, f'_k\}_{k=0}^n$.

Step 1. Form the Lagrangian polynomials $L_k(x)$ and find their derivatives $L'_k(x)$, $k = 0, 1, \ldots, n$.

Step 2. Compute the polynomials $\{B_k\}_{k=0}^n$ and $\{\hat{B}_k\}_{k=0}^n$ as follows:

$$\begin{aligned} B_k(x) &= ((1 - 2(x - x_k))L'_k(x_k))L_k^2(x) \\ \hat{B}_k(x) &= (x - x_k)L_k^2(x). \end{aligned}$$

Step 3. Form the Hermite polynomial $H_{2n+1}(x)$:

$$H_{2n+1}(x) = \sum_{k=0}^{n} f(x_k)B_k(x) + \sum_{k=0}^{n} f'(x_k)\hat{B}_k(x)$$

Example 6.24

Find the Hermite polynomial for the following data and use it to approximate $f(1.5)$ and $f'(1.5)$

x	$f(x)$	$f'(x)$
1	0	1
2	0.6931	0.5

Input Data:

$$\begin{cases} \text{Number of distinct points: 2 because } (n = 1) \\ \text{Nodes: } x_0 = 1, x_1 = 2 \\ \text{Functional values and the derivative: } f_0 = 0, f_1 = 0.6931, f'_0 = 1, f'_2 = 0.5 \\ \text{Degree of the Hermite interpolating polynomial: 3} \\ \text{Interpolating point: } x_I = 1.5 \end{cases}$$

Problem Details: Compute $H_3(1.5)$ and $H'_3(1.5)$ as approximations to $f(1.5)$ and $f'(1.5)$.

Formulas to be Used:

$$\begin{cases} H_3(x) = f_0B_0(x) + f_1B_1(x) + f'_0B_0(x) + f'_1B_1(x) \\ B_0(x) \text{ and } B_1(x) \text{ are as defined above.} \end{cases}$$

Step 1. Compute the Lagrangian polynomials $L_0(x)$ and $L_1(x)$ and their derivatives:

$$\begin{aligned} L_0(x) &= \frac{x - x_1}{x_0 - x_1} = \frac{x - 2}{-1} = 2 - x \\ L_1(x) &= \frac{x - x_0}{x_1 - x_0} = \frac{x - 1}{1} = x - 1. \end{aligned}$$

$$L_0'(x) = -1, \quad L_1'(x) = 1.$$

Step 2. Compute the polynomials $B_0(x), B_1(x), \hat{B}_0(x)$ and $\hat{B}_1(x)$ from the Lagrange polynomials:

$$\begin{aligned} B_0(x) &= (1 - 2(x - x_0)L_0'(x))L_0^2(x) \\ &= (1 - 2(x-1)(-1))(2-x)^2 \\ &= (1 + 2(x-1))(2-x)^2 \\ &= (2x-1)(2-x)^2 \end{aligned}$$

$$\begin{aligned} B_1(x) &= (1 - 2(x - x_1)L_1'(x))L_1^2(x) \\ &= (1 - 2(x-2))(x-1)^2 \\ &= (5-2x)(x-1)^2 \end{aligned}$$

$$\begin{aligned} \hat{B}_0(x) &= (x - x_0)(L_0^2(x)) = (x-1)(2-x)^2 \\ \hat{B}_1(x) &= (x - x_1)L_1^2(x) = (x-2)(x-1)^2 \end{aligned}$$

Step 3. Form the Hermite polynomial $H_3(x)$ of degree 3:

$$\begin{aligned} H_3(x) &= f_0 B_0(x) + f_1 B_1(x) + f_0' \hat{B}_0(x) + f_1' \hat{B}_1(x) \\ &= 0.6931(5-2x)(x-1)^2 + (x-1)(2-x)^2 + 0.5(x-2)(x-1)^2 \end{aligned}$$

Interpolate *f*(1.5) by computing: $H_3(1.5) = 0.4091$.

Accuracy Check:

(i) The above table corresponds to $f(x) = \ln(x)$. Thus, the **exact value** is

$$\ln(1.5) = 0.4055$$

(ii) **Absolute Error:** $|0.4055 - 0.4091| = 0.0036$.

6.10.1 Error in Hermite Interpolation

The following result for error bound on Hermite interpolation can be established in the same way as the error bound for the standard polynomial interpolation (**Theorem 6.7**) and is thus left as an [**Exercise**].

Theorem 6.25 (Error in Hermite Interpolation). Let

(i) $f(x)$ be $(2n+2)$ times continuously differentiable on $[a,b]$, and

(ii) $a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$.

Then for any $\bar{x}$ in $[a,b]$, there exists a number $\xi = \xi(\bar{x})$ in (a,b) such that the error in Hermite Interpolation is

$$E_H(\bar{x}) = f(\bar{x}) - H_{2n+1}(\bar{x}) = \frac{f^{(2n+2)}(\xi(\bar{x}))}{(2n+2)!}\psi^2(x),$$

where $\psi(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$.

*** ATTACHMENT #2 ?? ***

6.11 Piecewise-Interpolation vs Single-Polynomial Interpolation

A major drawback with a single-polynomial interpolation is that the process becomes computationally intensive if a large number of data has to be included to achieve better accuracy. *The more distressing fact is that there is no guarantee that interpolation with a high degree polynomial will yield better accuracy. In fact, there are cases where error increases as the degree of interpolation becomes higher.*

To illustrate this fact, we cite the following famous example by **Carl David Tolmel Runge**.

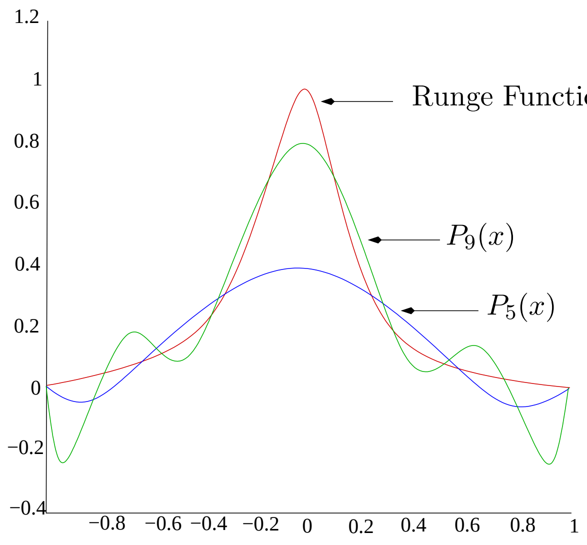
Interpolation with Runge's Function

Consider interpolating

$$f(x) = \frac{1}{1+25x^2}$$

in $[-1,1]$, using interpolating polynomials of degree 5 and 9, $P_5(x)$ and $P_9(x)$, respectively, with equidistant nodes.

It can be seen from the enclosed graph that *the resulting interpolating polynomials oscillate towards the end of the interval and the interpolation errors get worse with higher-degree polynomials* [see **Exercise**].



The above behavior is known as **Runge's phenomenon**.

Mathematical Reasoning for Runge's Phenomenon

Runge's phenomenon can be easily explained from the interpolation error formula:

$$E_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)\cdots(x-x_n)$$

It can be shown that

$$\lim_{n\to\infty}\left(\max_{-1\le x\le 1}|E_n(x)|\right) = +\infty$$

Thus, *the absolute maximum error of interpolation in this case grows as n becomes larger.*

Chebyshev Interpolation

Let's give a closer look at the polynomial interpolation error:

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

The error term $E_n(x)$ has two parts. The size of the first part

$$\frac{|f^{(n+!)}(\xi(x))|}{n+1!}$$

depends upon the bound M of $f^{(n+1)}(x)$ and n.

Except for some very special functions, such as $f(x) = \sin x$ or $f(x) = e^x$, etc., **usually, this part of the error steadily grows as n increases** (*as in the case of the Runge polynomial*, shown above.)

The next thing to do is to see if something can be done with the second part. Ideally, we would like to minimize

$$\max_{a \le x \le b} |(x - x_0)(x - x_1) \cdots (x - x_n)|.$$

If the nodes are given in advance (as has been the case so far), then we can not do much. But we can think of choosing these nodes so that this will happen.

The question is how?: The answer is provided in the following theorem, due to Russian mathematician, **A. Chebyshev**. We will discuss the theorem in details in **Chapter 10**.

Theorem 6.26 (Chebyshev Interpolation Theorem). Let the nodes, x_i, $i = 0, 1, \ldots, n$ be chosen as:

$$x_i = \frac{1}{2}\left[b + a - (b - a)\cos\left(\frac{2j+1}{2(n+1)}\pi\right)\right] ,$$

then

(i) $\max_{a \le x \le b} |(x - x_0)(x - x_1) \cdots (x - x_n)|$ will be minimized and the minimum value $= 2\left(\frac{b-a}{4}\right)^{n+1}$.

(ii) The interpolating error, E_{CH}, with these nodes satisfies:

$$|E_{CH}(x)| \le \frac{2(b-a)^{n+1}}{4^{n+1}(n+1)!} \max_{a \le x \le b} |f^{(n+1)}(x)|$$

□

Remarks: (i) The numbers $x_i, i = 0, 1, \ldots, n$ in Theorem 6.26 are the zeros of an orthogonal polynomial, called the **Chebyshev polynomial**.
(ii) The quantity $2\left(\frac{b-a}{4}\right)^{n+1}$, and hence, the worst case Chebyshev interpolation error, are much smaller than those obtained if x_i's were given arbitrarily (see the example below).

Example 6.27

Compute the maximum error of **linear interpolation** of $f(x) = \frac{1}{x}$ with use of the Chebyshev nodes on $[2, 4]$ and compare this with that obtained without using them (see **Example 6.8**).

Input Data: $\begin{cases} f(x) = \frac{1}{x}, \quad n = 1 \\ a = 2, \quad b = 4 \end{cases}$

Solution.

(i) Compute $f''(x) = \frac{2}{x^3}$.

(ii) Compute $M = \max_{2 \le x \le 4} |f''(x)| = \frac{1}{4}$.

(iii) Maximum Error with Chebyshev nodes (from **Theorem 6.24**): $\dfrac{2 \times 2^2}{16 \times 2} \times \dfrac{1}{4} = \dfrac{1}{16}$.

Comparison: Recall from Example 6.8, *the maximum error without Chebyshev Nodes is* $\frac{1}{8}$. *Thus, the maximum error with Chebyshev nodes is* $\frac{1}{2}$ *of the error without their uses.*

Piecewise polynomial interpolation. Even with the use of the Chebyshev nodes, the error can still be large if the interval size $(b-a)$ is large.

The thing to consider then is to divide the interval $[a, b]$ into several smaller intervals and then perform interpolation at each of these intervals. This process is referred to as the **piecewise interpolation**.

The **rationale for using piecewise interpolation** instead of high-degree interpolation is:

The maximum interpolation error for interpolating $f(x)$ for all x in $[a, b]$ using a piecewise interpolation contains a factor involving the largest of all the individual subinterval sizes that divide the interval $[a, b]$.

Specifically, let r be the degree of each interpolant and $h_i = x_{i+1} - x_i$, $i = 0, 1, \ldots, n-1$. Then the maximum interpolation error contains the factor: $\max_k (h_k)^{r+1}$. For example, the maximum error with piecewise linear interpolation (assuming that $f(x)$ is continuously twice differentiable on $[a, b]$) is:

$$\max_{a \le \xi \le b} |f''(\xi)| \times \frac{1}{8} \left[\max_k (h_k) \right]^2$$

where $h_k = x_{k+1} - x_k$.

Thus, *this error can be made as small as desired by choosing each h_k small enough for all k.*

The piecewise interpolation, of course, increases the number of interpolating points, but that does not substantially increase the computational complexity, because the interpolating polynomials in this case, are simple polynomials (usually of degree 1, 2, or 3).

Piecewise Polynomial Interpolation

(i) Given $(n+1)$ distinct points $(x_0, f_0), \ldots, (x_n, f_n)$ where $a = x_0 < x_1 < x_2 < \ldots < x_n = b$.

(ii) Let $I_1 = [x_0, x_1], I_2 = [x_1, x_2], \ldots, I_n = [x_{n-1}, x_n]$ be n subintervals of $[a, b]$.

Then the **piecewise polynomial interpolation** amounts to:

computing a polynomial $S(x)$, composed of n interpolating polynomials, $S_0(x), S_1(x), \ldots, S_{n-1}(x)$ in $I_1, I_2, \ldots, I_n$, respectively, each of the same degree, such that $S(x)$ interpolates the given data.

Definition 6.28. The abscissas x_i where the interpolant changes from one polynomial to the other are called **knots**.

Piecewise Linear Interpolation

The simplest piecewise interpolants are straight lines. Here a line is drawn in each subinterval I_k connecting the two points. The resulting piecewise line interpolant is continuous but its first derivative is not. Each piecewise linear interpolant has different slopes. Thus, the *resulting interpolant is not smooth.*

Piecewise Hermite Cubic Polynomial

The piecewise cubic Hermite interpolation is widely used. I has a continuous derivative on $[x_0, x_n]$. The difficulty with the piecewise Hermite interpolation is that not only the functional values but also the derivatives at the data points must be known. IN many practical situations, these derivatives are not known - the slopes must be defined somehow. Some special choices have been made in the MATLAB implementation, **pchip**, of the piecewise cubic Hermite interpolation. This particular "**pchip** is shape preserving, visually pleasing"[1].

Spline Interpolation

Though the MATLAB implementation of the piecewise Hermite cubic polynomial preserves the shape, it does so at the cost of smoothness. For smooth interpolations, we now consider splines. A **spline** of degree r is a piecewise polynomial which is $(r-1)$ times continuously differentiable. The cubic splines are most commonly used. A cubic spline is a piecewise cubic with continuous first and second derivatives.

The cubic spline is formally defined as follows:

[1]For details, see Moler (2004).

Cubic Spline

Given $(n+1)$ points $(x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n)$. Assume that $x_0 < x_1 < x_2 < \ldots < x_n$. We seek a **cubic polynomial** $S(x)$, denoted by $S_j(x)$ in $x_j \leq x \leq x_{j+1}$. That is,

$$S(x) = \begin{cases} S_0(x), & x_0 \leq x \leq x_1 \\ S_1(x), & x_1 \leq x \leq x_2 \\ \vdots & \\ S_{n-1}(x), & x_{n+1} \leq x \leq x_n \end{cases}$$

The cubic polynomials $S_j(x)$ have the following properties:

Property (i). $S_j(x_j) = f_j$ for $j = 0, 1, \ldots, n$. *(The spline passes through each point (x_j, f_j)).*

Property (ii). $S_j(x_{j+1}) = S_{j+1}(x_{j+1})$ for $j = 0, 1, 2, \ldots, n-2$. *(The spline is continuous over $[a, b]$).*

Property (iii). $S'_j(x_{j+1}) = S'_{j+1}(x_{j+1})$ for $j = 0, 1, \ldots, n-2$. *(The spline is a smooth function).*

Property (iv). $S''_j(x_{j+1}) = S''_{j+1}(x_{j+1})$ for $j = 0, 1, \ldots, n-2$. *(The second derivatives of the spline are continuous).*

Number of Unknowns vs. Number of Equations

Each $S_j(x)$ is a cubic polynomial. So, there are 4 coefficients to be determined to construct each $S_j(x)$. There are n such cubic polynomials to be determined. Thus, the total number of coefficients to be determined is **4n**.

However, the properties of $S(x)$ specified by the four properties (i)-(iv) gives us only $\mathbf{4n - 2}$ equations, as can be seen from the following count:

- Property (i) gives $(n+1)$ equations.
- Each of the Properties (ii)-(iv) gives $(n-1)$ equations.

Thus, to determine n polynomials $S_j(x), j = 0, 1, \ldots, k-1$, which contain $4n$ unknowns, *we still need two more equations.*

End Conditions

To obtain the remaining 2 unknowns, we can use some convenient end conditions. There are several end conditions that can be used. Here are some of the most commonly used ones.

- **Natural or Free Boundary Conditions**: In addition to satisfying the above four properties, $S(x)$ must satisfy the following second-derivative boundary conditions:

$$S^{''}(x_0) = S^{''}(x_n) = 0$$

- **Clamped Boundary Conditions**: In addition to satisfying the above four properties, $S(x)$ must satisfy the following first-derivative boundary conditions:

$$\begin{array}{rl} \text{(i)} & S'(x_0) = f'(x_0) \\ \text{(ii)} & S'(x_n) = f'(x_n) \end{array}$$

  In the sequel, we will show how to construct both the natural and clamped cubic splines, using properties (i)-(iv) and the relevant boundary conditions.

- **Not a knot condition**: This condition refers to the continuity of $S^{'''}(x)$ at the knot x_1 or x_{n-1}.

  Since by definition, the cubic spline $S(x)$, $S'(x)$, and $S^{''}(x)$ are continuous at the knots, the continuity of $S^{'''}(x)$ at the knot x_1 or x_{n-1} signifies that the two adjacent spline segments become identical, meaning that this knot is no longer a true knot. That is why the name, "**not-a-knot**" end condition.

6.11.1 Determining the Natural Cubic Spline

Let

$$h_j = x_{j+1} - x_j. \tag{6.20}$$

Since $S_j(x)$ is a cubic polynomial on the interval $[x_j, x_{j+1}]$, on this interval its second derivative, $S_j''(x)$, is a straight line. Let's write this straight line as:

$$S_j''(x) = z_j + m_j(x - x_j) \tag{6.21}$$

where m_j = slope of the line $S_j^{''}(x) = \frac{z_{j+1}-z_j}{h_j}$, and z_j's *still to be determined.*

Substituting the value of m_j from above in (6.26), we get

$$S_j''(x) = \frac{z_{j+1}}{h_j}(x - x_j) + \frac{z_j}{h_j}(x_{j+1} - x) \tag{6.22}$$

Our problem, however, is to compute $S_j(x)$, not $S_j^{''}(x)$. But not that $S_j^{''}(x)$ is a function of x and therefore, we can integrate it twice to obtain $S_j(x)$. This gives us

$$S_j(x) = \frac{z_{j+1}}{6h_j}(x - x_j)^3 + \frac{z_j}{6h_j}(x_{j+1} - x)^3 + A_j(x - x_j) + B_j(x_{j+1} - x), \tag{6.23}$$

where A_j and B_j are constants of integration.

To determine these two constants, we use *Property (i)* and *Property (ii)*:

$$S_j(x_j) = f_j \quad \text{Property (i)}$$

and

$$S_j(x_{j+1}) = S_j(x_{j+1}) = f_{j+1} \quad \text{Property (ii)}$$

Determining B_j: Substituting $x = x_j$ in (6.28), we get

$$S_j(x_j) = f_j = \frac{z_j}{6h_j} \times h_j^3 + B_j h_j$$

Thus,

$$\boxed{B_j = \frac{f_j}{h_j} - \frac{h_j}{6} z_j} \tag{6.24}$$

Determining A_j: Substituting $x = x_{j+1}$ in (6.28), we get

$$S_j(x_{j+1}) = f_{j+1} = \frac{z_{j+1}}{6h_j} \times h_j^3 + h_j A_j$$

Thus,

$$\boxed{A_j = \frac{f_{j+1}}{h_j} - \frac{h_j}{6} z_{j+1}} \tag{6.25}$$

Knowing A's and B's, all that we need to know is what z_j's are to obtain $S_j(x)$. We now show how to do so.

Computation of the numbers $z_0, z_1, \ldots z_n$:

- The numbers z_0 and z_n are computed from the boundary conditions:

  $$S^{''}(x_0) = S^{''}(x_n) = 0.$$

  These conditions give

  $$\boxed{z_0 = z_n = 0.}$$

- The remaining numbers z_1 through z_{n-1} are computed by using Property (iii):

$$S'_{j-1}(x_j) = S'_j(x_j), \quad j = 1, 2, \dots, n-1. \tag{6.26}$$

To use Condition (6.31), we

(i) **first** compute $S'_j(x)$ and $S'_{j-1}(x)$ by differentiating (6.28).

(ii) **Then** evaluate these expressions at $x = x_j$.

(iii) **Finally**, using the values of A_j and B_j from (6.30) and (6.29), respectively, and substituting $h_{j-1} = x - x_{j-1}$ in the expressions of $S'_{j-1}(x_j)$ and $S'_j(x_j)$, we obtain:

$$\begin{aligned} S'_{j-1}(x_j) &= \tfrac{h_{j-1}}{6} z_{j-1} + \tfrac{h_{j-1}}{3} z_j + b_{j-1}, \\ S'_j(x_j) &= -\tfrac{h_j}{6} z_{j+1} - \tfrac{h_j}{3} z_j + b_j, \\ \text{where} & \\ b_j &= \tfrac{1}{h_j}(f_{j+1} - f_j). \end{aligned}$$

The condition (6.26) then gives us

$$\frac{h_{j-1}}{6} z_{j-1} + \frac{h_{j-1}}{3} z_j + b_{j-1} = -\frac{h_j}{6} z_{j+1} - \frac{h_j}{3} z_j + b_j$$

or

$$h_{j-1} z_{j-1} + 2h_{j-1} z_j + 6b_{j-1} = -h_j z_{j+1} - 2h_j z_j + 6b_j$$

or

$$h_{j-1} z_{j-1} + 2(h_{j-1} + h_j) z_j + h_j z_{j+1} = 6(b_j - b_{j-1}), \quad j = 1, 2, \dots, n-1. \tag{6.27}$$

Equation (6.27) can be written as a system of n equations in n unknowns, $z_1, z_2, \dots, z_{n-1}$ as follows:

$$\begin{pmatrix} 2(h_0+h_1) & h_1 & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & \\ & h_2 & 2(h_2+h_3) & h_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{pmatrix} = \begin{pmatrix} 6(b_1-b_0) \\ 6(b_2-b_1) \\ 6(b_3-b_2) \\ \vdots \\ 6(b_{n-2}-b_{n-3}) \\ 6(b_{n-1}-b_{n-2}) \end{pmatrix} \tag{6.28}$$

The matrix of the system (6.28) is *symmetric* and *strictly diagonally dominant. We can safely solve this system using Gaussian elimination without pivoting (recall from* ***Chapter 5*** *that the growth factor ρ in this case is less than or equal to* 2).

Uniqueness of Natural Cubic Spline

In view of our above discussion, the following theorem immediately follows.

Theorem 6.29 (Uniqueness of Natural Cubic Spline). If a function $f(x)$ is defined in $[a,b]$, and if $x_0, x_1, \dots, x_n$ are distinct nodes, where $a = x_0 < x_1 < x_2 < \cdots < x_n = b$. Then there is a unique natural cubic spline for $f(x)$.

Proof. The system of linear algebraic equations in (6.28) has a unique solution for $z_1, z_2, \dots, z_{n-1}$, because the matrix of this system is strictly diagonally dominant which is nonsingular [**Exercise**].

The quantities z_0 and z_n are equal to zero because of the natural boundary conditions:

$$S''(x_0) = S''(x_n) = 0.$$

Thus z_j *are uniquely determined.* Once z_j are determined, the quantities A_j and B_j are uniquely determined, respectively, from (6.30) and (6.29). Thus S_j are uniquely determined from (6.23). $\square$

Based on the above discussions, an algorithm for computing the natural cubic spline $S_j(x)$ can be stated as follows:

Algorithm 6.30 (Computing Natural Cubic Spline).

```
Inputs:  (i) The nodes: $x_0, x_1, \ldots, x_n$
         (ii) The functional values: $f(x_i) = f_i$, $i = 0, 1, \ldots, n$.
         (iii) $S''(x_0) = S''(x_n) = 0$ (Natural Boundary Condition)
Outputs: The cubic splines $S_0(x), S_1(x), \ldots, S_{n-1}(x)$, given by (6.23).
Step 0. Compute the spacing between the nodes: $h_j = x_{j+1} - x_j, j = 0, 1, 2, \ldots, n-1$.
Step 1. For $j = 0, 1, \ldots, n-1$ do
            Compute the right-hand side of (6.33): $b_j = \frac{1}{h_j}(f_{j+1} - f_j)$
        End
Step 2. Set $z_0 = z_n = 0$ (End-point boundary conditions).
Step 3. Solve the symmetric strictly diagonally dominant system (6.33) to obtain $z_1, z_2, \ldots, z_{n-1}$.
Step 4. Compute $A_j$ and $B_j$, $j = 0, 1, \ldots, n-1$ using (6.30) and (6.29), respectively.
Step 5. Compute $S_j(x)$, $j = 0, 1, \ldots, n-1$ using (6.23).
End
```

Example 6.31

Find the cubic spline passing through $(1, 0)$, $(1.5, 0.6082)$ and $(2, 1.3863)$.

Input Data: $\begin{cases} \text{Nodes: } x_0 = 1, x_1 = 1.5, x_2 = 2 \\ \text{Functional Values: } f_0 = 0, f_1 = 0.6082, f_2 = 1.3863 \end{cases}$

Problem Details: $\begin{cases} \text{Find the cubic spline } S(x) \text{ defined by} \\ S_0(x), \quad 1 \le x \le 1.5 \\ S_1(x), \quad 1.5 \le x \le 2 \end{cases}$

Formulas to be used: Follow the steps of Algorithm 6.28.

Solution.

Step 0. Compute h_0 and h_1:

$$h_0 = x_1 - x_0 = 1.5 - 1.0 = 0.5$$
$$h_1 = x_2 - x_1 = 2 - 1.5 = 0.5$$

Step 1. Compute b_0 and b_1:

$$b_0 = \frac{1}{h_0}(f_1 - f_0) = \frac{1}{0.5}(0.6082) = 1.2164$$
$$b_1 = \frac{1}{h_1}(f_2 - f_1) = \frac{1}{0.5}(1.3863 - 0.6082) = 1.5562$$

Step 2. $z_0 = 0 \quad z_2 = 0.$

Step 3. Solve the system (6.33) to compute z_1: **(Note in this case, there is only one equation)**:

$$2(h_0 + h_1)z_1 = 6(h_1 - h_0) = 2.0358 \text{ or } z_1 = 1.0194.$$

Step 4. Compute A_0, B_0, and A_1, B_1 from (6.30) and (6.29), respectively:

$$\begin{aligned} A_0 &= \tfrac{f_1}{h_0} - \tfrac{h_0}{6} \qquad z_1 = \tfrac{0.6082}{0.5} - \tfrac{0.5}{6}(1.0194) = 1.1314 \\ B_0 &= \tfrac{f_0}{h_0} - \tfrac{h_0}{6} \qquad z_0 = 0 - \tfrac{0.5}{6} \times 0 = 0 \\ A_1 &= \tfrac{f_2}{h_1} - \tfrac{h_1}{6} \qquad z_2 = 2.7726 \\ B_1 &= \tfrac{f_1}{h_1} - \tfrac{h_1}{6} \qquad z_1 = 1.1314 \end{aligned}$$

Step 5. Write the cubic polynomials $S_0(x)$ and $S_1(x)$:

$$\begin{aligned} S_0(x) &= \frac{z_1}{6h_0}(x - x_0)^3 + \frac{z_0}{6h_1}(x_1 - x)^3 + A_0(x - x_0) + B_0(x_1 - x) \\ &= \frac{1.0194}{3}(x-1)^3 + 0 + 1.1314(x-1) = 0.3398(x-1)^3 + 1.1314(x-1). \\ S_1(x) &= \frac{z_2}{6h_1}(x - x_1)^3 + \frac{z_1}{6h_1}(x_2 - x)^3 + A_1(x - x_1) + B_1(x_2 - x) \\ &= 0 + \frac{1.0194}{3}(2-x)^3 + 2.7726(x - 1.5) + 1.1314(2 - x). \end{aligned}$$

Natural Cubic Spline for the Data: $(1, 0)$**,** $(1.5, 0.6082)$ **and** $(2, 1.3863)$

$$\begin{cases} S_0(x) = 0.3398(x-1)^3 + 1.1314(x-1), \quad 0 \le x \le 1.5 \\ S_1(x) = \frac{1.0194}{3}(2-x)^3 + 2.7726(x-1.5) + 1.1314(2-x), \quad 1.5 \le x \le 2 \end{cases}$$

Verification: $j = 0$. We verify here that $S_0(x)$ is a cubic spline in the interval $[0, 1.5]$.

- Verification of Condition (i): $S_0(x_0) = f_0$.

$$\begin{aligned} S_0(x_0) &= S_0(1) = 0 \\ f_0 &= 0 \end{aligned}$$

So,

$$\boxed{S_0(x_0) = f_0}$$

- Verification of Condition (ii): $S_0(x_1) = S_1(x_1)$.

$$\begin{aligned} S_0(x_1) &= S_0(1.5) = 0.3398 \times (0.5)^3 + 1.1314 \times (0.5) = 0.6082. \\ S_1(x_1) &= \tfrac{1.0194}{3}(0.5)^3 + 2.7726 \times 0 + 1.1314(0.5) = 0.6082 = f_1 \end{aligned}$$

So, $\boxed{S_0(x_1) = S_1(x_1) = 0.6082}$

- Verification of Condition (iii): $S_0'(x_1) = S_1'(x_1)$.

$$\begin{aligned} S_1'(x_1) &= \tfrac{z_1}{6h_1} \times 3(x_2 - x_1)^2(-1) + A_1 - B_1 = 1.3863 \\ S_0'(x_1) &= 3 \times 0.3398(0.5)^2 + 1.1314 = 1.3863. \end{aligned}$$

Thus,

$$\boxed{S'(x_1) = S_0'(x_1).}$$

- Verification of Condition (iv): $S_1''(x_1) = S_0''(x_1)$.

Compute the Derivatives:

$$\begin{aligned} S_0'(x) &= 3 \times 0.3398(x-1)^2 + 1.1314 \\ S_0''(x) &= 6 \times 0.3398(x-1) \\ S_1'(x) &= -1.0194(2-x)^2 - 1.1314 \\ S_2''(x) &= 2.0388(2-x) \end{aligned}$$

$$\begin{aligned} S_1''(x_1) &= 2.0388 \times 0.5 = 1.0194 \\ S_0''(x_1) &= 6 \times 0.3398 \times 0.5 = 1.0194. \end{aligned}$$

Thus,

$\boxed{S_1''(x_1) = S_0''(x_1)}$

- Verification of the boundary condition: $S_0''(x_0) = S_0''(x_2) = 0$.

$$\begin{aligned} S''(x_0) &= 6 \times 0.3398 \times 0 = 0 \\ S''(x_2) &= 2.0388 \times 0 = 0 \end{aligned}$$

Thus, $\boxed{S_0''(x_0) = S_0''(x_2) = 0}$.

The readers are invited to carry out the computations to verify the spline conditions when $j = 1$.

6.11.2 Determining Clamped Cubic Spline

The clamped cubic spline can be determined in the same way as in the last section.

Set

$$S'(x_0) = \alpha$$
$$S'(x_n) = \beta$$

Using these boundary conditions, the linear algebraic system of equations that needs to be solved in this case for determining the unknowns $z_1, z_2, \ldots, z_{n-1}$ are given by [**Exercise**]:

$$\begin{cases} \left(\frac{3}{2}h_0 + 2h_1\right) z_1 + h_1 z_2 = 6(b_1 - b_0) - 3(b_0 - \alpha) \\ h_{k-1}z_{k-1} + 2(h_{k-1} + h_k)z_k + h_k z_{k+1} = 6(b_k - b_{k-1}), \text{ for } k = 2, 3, \ldots, n-2 \\ h_{n-2}z_{n-2} + \left(2h_{n-2} + \frac{3}{2}h_{n-1}\right) z_{n-1} = 6(b_{n-1} - b_{n-2}) - 3(\beta - b_{n-1}). \end{cases} \quad (6.29)$$

Computing z_0 and z_n.

These quantities are computed by (6.34) from the above two boundary conditions. It can be shown that

$$\begin{aligned} z_0 &= \frac{3}{h_0}(b_0 - \alpha) - \frac{z_1}{2} \\ z_n &= \frac{3}{h_{n-1}}(\beta - b_{n-1}) - \frac{z_{n-1}}{2}. \end{aligned}$$

Computing A's and B's: Once all the z's are computed, A_j's and B_j's are computed, respectively, from (6.30) and (6.29).

Based on the above discussion, the algorithm for computing the **damped cubic spline** can be stated as follows:

Algorithm 6.32 (Computing Clamped Cubic Spline).

Inputs: (i) The nodes $x_0, x_1, \ldots, x_n$.

(ii) The functional values: $f(x_i) = f_i$, $i = 0, 1, \ldots, n$.

(iii) $f'(x_0) = \alpha = S'(x_0)$ and $f'(x_n) = f'(b) = \beta = S'(x_n)$.
(Clamped Boundary Conditions).

```
Step 0. For $i = 0, 1, \ldots, n-1$ do
            Set $h_j = x_{j+1} - x_j$
Step 1. Compute $b_0, b_1, \ldots, b_{n-1}$:
        For $i = 0, 1, \ldots, n-1$ do
            $b_j = \frac{1}{h_j}(f_{j+1} - f_j)$.
        End
Step 2. Solve the symmetric tridiagonal strictly diagonally dominant system (6.34) to compute
        $z_1, z_2, \ldots, z_{n-1}$.
Step 3. Compute $z_0$ and $z_n$:
```

$$\begin{cases} z_0 &= \frac{3}{h_0}(b_0 - \alpha) - \frac{z_1}{2} \\ z_n &= \frac{3}{h_{n-1}}(\beta - b_{n-1}) - \frac{z_{n-1}}{2} \end{cases} \tag{6.30}$$

```
Step 4. Compute $A_j$ and $B_j = j = 0, 1, 2, \ldots, n-1$ from equations (6.30) and (6.29), respectively.
Step 5. Construct the cubic splines $S_j(x)$, $j = 0, 1, \ldots, n-1$ using (6.23).
```

Example 6.33

Construct the clamped cubic spline $S(x)$ for the following data:

i	x	$f(x)$
0	1	0
1	1.5	0.4055
2	2	0.6931
3	2.5	0.9163

with the clamped boundary conditions: $S'(1) = 1$, $S'(2.5) = 0.4000$.

Step 0. Compute h_0, h_1, and h_2:

$$h_0 = x_1 - x_0 = 0.5$$
$$h_1 = x_2 - x_1 = 0.5$$
$$h_2 = x_3 - x_2 = 0.5$$

Step 1. Compute b_0, b_1, and b_2:

$$\begin{aligned} b_0 &= \frac{1}{h_0}(f_1 - f_0) = 0.8109 \\ b_1 &= \frac{1}{h_1}(f_2 - f_1) = 0.5754 \\ b_2 &= \frac{1}{h_2}(f_3 - f_2) = 0.4463 \end{aligned}$$

Step 2. Solve the system (**??**) to compute z_1 and z_2:

$$\begin{pmatrix} 1.7500 & 0.5000 \\ 0.5000 & 1.7500 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} -0.8462 \\ 7.8937 \end{pmatrix}$$
$$z_1 = -1.9298, \quad z_2 = 5.0621.$$

Step 3. Compute z_0 and z_3 from the formulas of Step 3::

$$\begin{aligned} z_0 &= \frac{3}{0.5}(0.8109 - 1) + \frac{1.9298}{2} \\ &= -0.1695 \\ z_3 &= \frac{3}{0.5}(0.4000 - 7.8937) - \frac{5.0621}{2} \\ &= -47.4933. \end{aligned}$$

Step 4. Compute A_0, B_0, A_1, B_1, A_2, and B_2 from (6.30) and (6.29):

$$\begin{aligned} A_0 &= \frac{f_1}{h_0} - \frac{h_0}{6} z_1 = 0.9718. \\ B_0 &= \frac{f_0}{h_0} - \frac{h_0}{6} z_0 = 0.0141. \end{aligned}$$

$$\begin{aligned} A_1 &= \frac{f_2}{h_1} - \frac{h_1}{6} z_2 = 0.9645. \\ B_1 &= \frac{f_1}{h_1} - \frac{h_1}{6} z_1 = 0.9718. \end{aligned}$$

$$\begin{aligned} A_2 &= \frac{f_3}{h_2} - \frac{h_2}{6} z_3 = 5.7904. \\ B_2 &= \frac{f_2}{h_2} - \frac{h_2}{6} z_2 = 0.9645. \end{aligned}$$

Step 5. Construct the clamped cubic spline:

$$\begin{aligned} S_0(x) &= \frac{z_1}{6h_0}(x - x_0)^3 + \frac{z_0}{6h_1}(x_1 - x)^3 + A_0(x - x_0) + B_0(x_1 - x) \\ &= -0.6433(x - 1)^3 - 0.0565(1.5 - x)^3 + 0.9718(x - 1) + 0.0141(1.5 - x) \end{aligned}$$

$$\begin{aligned}S_1(x) &= \frac{z_2}{6h_1}(x-x_1)^3 + \frac{z_1}{6h_1}(x_2-x)^3 + A_1(x-x_1) + B_1(x_2-x)\\ &= 1.6874(x-1.5)^3 - 0.6433(2-x)^3 + 0.9645(x-1.5) + 0.9718(2-x).\end{aligned}$$

$$\begin{aligned}S_2(x) &= \frac{z_3}{6h_2}(x-x_2)^3 + \frac{z_2}{6h_2}(x_3-x)^3 + A_2(x-x_2) + B_2(x_3-x)\\ &= -15.8311(x-2)^3 + 1.6874(2.5-x)^3 + 5.7904(x-2) + 0.9645(2.5-x)\end{aligned}$$

The clamped cubic spline for the data: $(1,0)$, $(1.5,0.4055)$, $(2,0.6931)$, and $(2.5,0.9163)$:

$$\begin{cases} S(x) = -0.6433(x-1)^3 - 0.0565(1.5-x)^3 \\ \qquad + 0.9718(x-1) + 0.0141(1.5-x), \text{ for } 1 \le x \le 1.5. \\ S(x) = 1.6874(x-1.5)^3 - 0.6433(2-x)^3 \\ \qquad + 0.9645(x-1.5) + 0.9718(2-x), \text{ for } 1.5 \le x \le 2. \\ S(x) = -15.8311(x-2)^3 + 1.6874(2.5-x)^3 \\ \qquad + 5.7904(x-2) + 0.9645(2.5-x), \text{ for } 2 \le x \le 2.5. \end{cases}$$

6.11.3 Uniqueness of the Clamped Cubic Spline

The following result on the uniqueness of the clamped cubic spline immediately follows from the above discussion. The proof os similar to theorem on uniqueness of the natural cubic spline. The readers are invited to fill-in the details [**Exercise**].

Theorem 6.34 (Uniqueness of the clamped cubic spline). The function $f(x)$ defined at $a = x_0 < x_1 < \cdots < x_n = b$ has a unique clamped cubic spline at these nodes.

6.11.4 Error in Cubic Spline Interpolation

As noted before, in piecewise interpolation, the error term contains a factor involving the maximum lengths of all the subintervals. Here is such an error formula for the **clamped cubic spline interpolation**.

Theorem 6.35 (Cubic Spline Error). Let

(i) $f(x)$ be four times continuously differentiable in $[a,b]$

(ii) $S(x)$ be a unique clamped cubic spline interpolant of $f(x)$ at the nodes: $a = x_0 < x_1 < \ldots < x_{n-1} < x_n = b$.

(iii) $h_i = x_{i+1} - x_i, i = 0, 1, \ldots, n-1$.

Then for all x in $[a, b]$, the following result on the absolute error holds:

$$|f(x) - S(x)| \leq 5 \max_{\xi \in [a,b]} |f^{(4)}(\xi)| \times \frac{(\max_i h_i)^4}{384}.$$

Significance of Theorem 6.33

The theorem says that unless $\max\limits_{\xi \in [a,b]} |f^{(4)}(\xi)|$ is too large, then choosing each interval size h_i small enough, one can obtain a small error with clamped cubic spline interpolation.

6.12 Bezier Curves and Bezier Splines

The Bezier curves, named after the French engineer, **Pierre Bézier**, who popularized its use in automobile body designes, is a parametric curve. Nowadays, they are frequently used in

- Computer graphics to model smooth curves
- Animation, such as Adobe Flush
- Fonts

The reason for such popularity of Bezier curves is that these curves are simple to draw and easy to manipulate to give desired shapes.

Definition 6.36. A Bezier curve of degree n is defined by

$$P(t) = \Sigma_{i=0}^{n} P_i B_{i,n}(t), \quad 0 \leq t \leq 1$$

where $P_0, P_1, \ldots, P_n$ are called the **control points**, and the $B_{i,n}(t)$ are polynomials each of degree n, called **Bernstein polynomials**.

Definition 6.37. Bernstein polynomials $B_{i,n}(t)$ of degree n are defined by:

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, 1, \ldots, n.$$

where

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}.$$

Special cases: **n=1**. The Bernstein polynomials of degree 1 are:

$$B_{0,1}(t) = 1,\ B_{1,1}(t) = t.$$

n=2. The Bernstein polynomials of degree 2 are:

$$B_{0,2}(t) = (1 - t^2),\ B_{1,2}(t) = 2t(1 - t),\ \text{and } B_{2,2}(t) = t^2.$$

n=3. The Bernstein polynomials of degree 3 are:

$$B_{0,3}(t) = (1 - t)^3,\ B_{1,3}(t) = 3t(1 - t^2),$$
$$B_{2,3}(t) = 3t^2(1 - t),\ \text{and } B_{3,3}(t) = t^3.$$

6.12.1 Bezier Curves of Degree 1, 2, and 3

Using the above expressions of the Bernstein polynomials, the Bezier curves in the special cases of $n = 1, 2$, and 3 are given by:

Linear Bezier curve: A linear Bezier curve, defined by two points P_0 and P_1, is given by:

$$B(t) = P_0 B_{0,1}(t) + P_1 B_{0,1}(t) = (1 - t)P_0 + tP_1$$

• The value $t = 0$ corresponds to the point P_0.

• The value $t = 1$ corresponds to the point P_1.

Quadratic Bezier curve: Similarly, a **quadratic Bezier curve** is defined by three given points P_0, P_1, and P_2 which are written in the form:

$$B(t) = (1 - t)^2 P_0 + 2(1 - t)P_1 + t^2 P_2, \quad t \in [0, 1].$$

Cubic Bezier curve: A cubic Bezier curve is more frequently used in applications. It has four control points: P_0, P_1, P_2, and P_3. The curve starts at P_0, goes towards P_1, and then ends at P_3, coming from the direction of P_2.

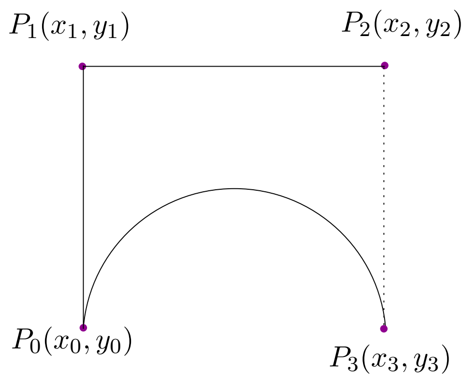


Figure 6.1 A cubic Bezier curve

A cubic Bezier curve in terms of the Bernstein polynomials is written as:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3, \quad t \in [0,1].$$

It is clear from the above expression that a cubic Bezier curve is composed of two quadratic Bezier curves $B_{P_0P_1P_2}(t)$ passing through P_0, P_1, and P_2, and $B_{P_1P_2P_3}(t)$ passing through P_1, P_2, and P_3.

6.12.2 Parametric Representation of the Bezier Curve

Taking $P(t) = (x(t), y(t))$, a Bezier curve of degree n can be written as:

$$\begin{cases} x(t) &= \Sigma_{i=0}^{n} x_i B_{i,n}(t) \\ y(t) &= \Sigma_{i=0}^{n} y_i B_{i,n}(t) \end{cases}$$

In particular, the cubic Bezier curve has the following parametric form:

$$\begin{cases} x(t) &= x_0 B_{0,3}(t) + x_1 B_{1,3}(t) + x_2 B_{2,3}(t) + x_3 B_{3,3}(t) \\ y(t) &= y_0 B_{0,3}(t) + y_1 B_{1,3}(t) + y_2 B_{2,3}(t) + y_3 B_{3,3}(t) \end{cases}$$

Substituting the values of $B_{0,3}(t)$, $B_{1,3}(t)$, and $B_{2,3}(t)$, we have

Parametric Representation of the Bezier Cubic

$$\begin{aligned} x(t) &= x_0(1-t)^3 + x_1 3t(1-t)^2 + x_2 3t^2(1-t) + x_3 t^3 \\ y(t) &= y_0(1-t)^3 + y_1 3t(1-t^2) + y_2 3t^2(1-t) + y_3 t^3 \end{aligned}$$

where $P_i = (x_i, y_i)$, $i = 0, 1, 2, 3$.

6.12.3 Properties of Bezier Curves

The following properties are some of the important properties of the Bezier curves, that can be proved rather easily [**Exercise**]. Some of them follow immediately from the definition of the Berstein polynomials.

Property I. The Bezier curve passes through the two end points P_0 and P_n. So, $P(0) = P_0$ and $P(1) = P_n$; but the other control points may not be on the curve.

Property II. The Bezier curve is a convex combination of control points; that is, *the Bernstein polynomials are nonnegative* in $0 \le t \le 1$ and $\Sigma_{i=0}^{n} B_{i,n}(t) = 1$.

Property III. $P'(0) = n(P_1 - P_0)$ and $P'(1) = n(P_n - P_{n-1})$.

This property says that the tangent lines at each of the end points, P_0 and P_n, are parallel to the lines that pass through that end point and the adjacent control point.

See Figure 6.2 of the Bezier cube. *The lines P_0P_1 and P_3P_2 are parallel.*

Example 6.38 **A.** Find the parametric form of the Bezier cube with $(0, 0)$ and $(3, 0)$ as endpoints and $(1, 3)$ and $(4, 3)$ as the other two control points.

B. Verify the Bezier cubic properties.

C. Draw the graph.

Input Data: $\begin{cases} (x_0, y_0) & = (0,0), (x_1, y_1) = (1,3), (x_2, y_2) = (4,3), \text{ and } (x_3, y_3) = (3,0) \\ n & = 3 \end{cases}$

Formula to be used: $\begin{cases} x(t) & = x_0(1-t)^3 + x_1 3t(1-t^2) + x_2 3t^2(1-t) + x_3 t^3 \\ y(t) & = y_0(1-t)^3 + y_1 3t(1-t^2) + y_2 3t^2(1-t) + y_3 t^3 \end{cases}$

Solution.

A. Parametric Construction of the Bezier Cubic

$$\begin{cases} x(t) & = 3t(1-t^2) + 12t^2(1-t) + 3t^3 \\ y(t) & = -18t^3 + 9t^2 + 9t, \quad 0 \le t \le 1. \end{cases}$$

B. Verification of the Bezier Properties

I. $\begin{cases} x(0) = 0, & y(0) = 0; \\ x(1) = 3, & y(1) = 0. \end{cases}$

Thus, the curve passes through the end points $P_0 = (0, 0)$ and $P_3 = (3, 0)$.

II. $\begin{cases} x'(0) = 3, & y'(0) = 9; \\ x'(1) = -9, & y'(1) = -27 \end{cases}$

Thus, the tangent lines at the end points are parallel. *The readers are invited to verify the other properties.*

C.

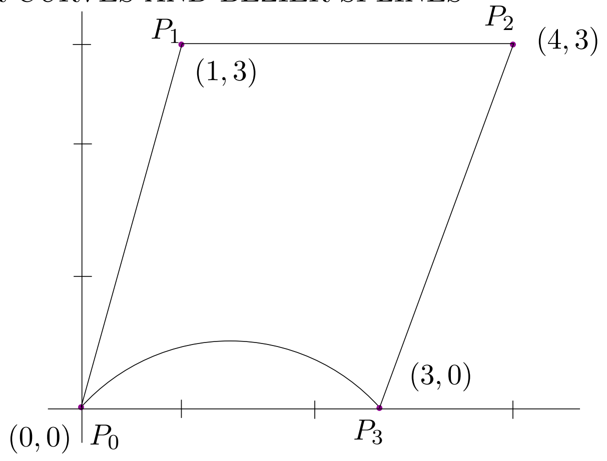


Figure 6.2: Bezier Graph (for the data)

Composite Bezier Curve: A curve that is composed of several Bezier curves is called a **composite Bezier curve**. In particular, when each curve is of the same degree, it is called a **Bezier spline**. Composite Bezier curves play an important role in certain applications, such as CAGD, where a complicated shape has to be formed. A single Bezier curve is not enough to draw such a shape. Indeed, in most practical applications, a sequence of Bezier curves with common endpoints are used to produce a complicated shape.

Postscripts, **SVG** (scalable vector graphics), the typographic system **METAFONT**, etc., use Bezier splines composed of Bezier cubics.

It is desirable that the curves are smooth at the common endpoints. One can ensure the smoothness of two Bezier curves $P(t)$ and $Q(t)$ by requiring that the derivatives at the common endpoints match. This can be assured by making the control points P_{n-1}, $P_n = Q_0$, and Q_1 collinear. Note this is just a sufficient condition.

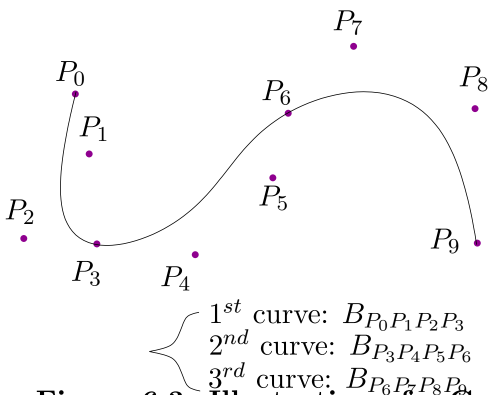

P_7
P_0
P_6
P_8
P_1
P_5
P_2
P_9
P_3
P_4
1st curve: $B_{P_0P_1P_2P_3}$
2nd curve: $B_{P_3P_4P_5P_6}$
3rd curve: $B_{P_6P_7P_8P_9}$

Figure 6.3: Illustration of a Composite Bezier Curve Consisting of Three Bezier Curves

6.13 MATLAB Function and Implementation

Several forms of **piecewise polynomial interpolation** are offered by MATLAB built-in function **interpl**. MATLAB does not offer any built-in functions for single polynomial interpolations. But the function "**polyfit**" can be used to fit a set of discrete data in a least-squares sense.

Usages for interpl:

$YI =$	**interp1** $(X, Y, XI,$ 'method')
$X, Y-$	Data vectors
$XI-$	A scalar or a vector at which interpolation is to be performed.
$YI-$	The output scalar or a vector of the interpolated values at XI.
'method' –	**nearest, linear, spline, pchip, cubic**.
nearest –	nearest neighboring interpolation; returns the value(s) that are nearest to the interpolated point(s).
linear –	linear spline interpolation, based on piecewise linear interpolation.
spline –	cubic spline interpolation, based on piecewise cubic interpolation with **not-a-knot** or **clamped cubic** end conditions. The default is "**not-a-knot**" condition.
pchip –	piecewise cubic Hermite interpolation
cubic –	same as **pchip**.

In addition to the **spline** option offered in **interpl**, there is a function, called, '**spline**' for cubic spline interpolation. Usually, it uses "**not-a-knot**" condition, but there are options for other types.

Usage: $PP =$ **spline** (X, Y)

provides the piecewise polynomial form of the cubic spline interpolation for later use with **ppval** to evaluate interpolated values at XI.

6.13.1 Implementation Using the Data of Velocity Profile of Rocket in Introduction

$$\begin{aligned} \Rightarrow\quad & X = (0, 10, 15, 22, 25, 30)^T \\ \Rightarrow\quad & Y = (0, 250, 350, 655, 890, 910) \\ \Rightarrow\quad & XI = (5, 20, 23, 29). \end{aligned}$$

- Linear Interpolation

$$\Rightarrow \quad YI = \textbf{interp1}\ (X, Y, XI,\ \text{`linear'})$$
$$YI = (125, 567.8571, 733.3333, 906)^T$$

Results of Linear Interpolation (in tabular form)

Time(s)	Predicted Velocity (m/s)
5	125
20	567.8571
23	733.333
29	906

- Cubic Spline Interpolation (same as **pchip**)

$$\Rightarrow YI = \ \textbf{interp1}\ (X, Y, XI, \text{`cubic'})$$
$$YI = \ (132.9776, 546.1006, 739.9677, 909.2369)^T$$

- Spline Interpolation

$$\Rightarrow YI = \ \textbf{spline}\ (X, Y, XI)$$
$$YI = \ (153.1696, 529.5772, 732.7615, 981.4770)^T$$

Results of Cubic Spline Interpolation

$t(s)$	$v(t)$ (m/s)
5	153.1696
20	529.577
23	732.7615
29	981.4770

- Nearest Neighboring Interpolation

$$\Rightarrow \quad YI = \ \textbf{interp1}\ (X, Y, XI,\ \text{`nearest'})$$
$$YI = (250, 655, 655, 910)$$

Results of Nearest Neighboring Interpolation

$t(s)$	$v(s)$ (m/s)
5	250
20	655
23	655
29	910

Exercises

1. (**Conceptual**) Answer **True** or **False** and give reasons for your answer.
   - (i) Different methods of interpolation give different interpolating polynomials.
   - (ii) The condition numbers of the linear systems arising from interpolation with different bases are the same.
   - (iii) The interpolation error is the same no matter what method of interpolation is used.
   - (iv) The polynomial of degree $\leq n$ which interpolates $f(x)$ at $(n+1)$ distinct points is $f(x)$, if $f(x)$ itself is a polynomial of degree $\leq n$.
   - (v) The kth divided difference $p[x_0, x_1, \ldots, x_k]$ of a polynomial $p(x)$ of degree $\leq k$ is independent of the ordering of interpolation points $x_0, x_1, \ldots, x_k$.
   - (vi) Spline interpolation is better than interpolation with a single polynomial of higher order.
   - (vii) The kth divided difference of a polynomial $P_k(x)$ of degree $\leq k$ is always a constant different from zero.
   - (viii) Newton's backward difference formula gives more accurate answers when interpolating towards the end of a table.
   - (ix) If the nodes are equally spaced, the divided differences become identical with the forward differences.
   - (x) The tangent lines at the end points of a Bezier curve are parallel.

2. (**Analytical**) Prove that the Vandermonde matrix A of System (6.2) is nonsingular if x_k, $k = 0, 1, \ldots, n$ are all distinct.

3. (**Analytical**) Given three distinct points $(x_0, f_0), (x_1, f_1)$, and (x_2, f_2), show that the interpolating polynomial $P_2(x)$ obtained by using any of the three bases: **monomial basis**, **Lagrange basis**, and the **Newton basis**, is the same.

4. (**Analytical**) Show that the matrix A of the linear system: $Ax = f$ for the Newton interpolation is lower triangular, and the coefficients $a_i, i = 0, 1, \ldots, k$ of the Newton interpolating polynomial $P_k(x)$ obtained by solving this system are the same as those obtained by using the Vandermonde system.

5. (**Analytical**) If $P_k(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_k(x - x_0)(x - x_1)\ldots(x - x_{k-1})$, is the Newton interpolating polynomial of degree k, then prove that

$$a_i = f[x_0, x_1, \ldots, x_i], \quad i = 0, 1, \ldots, k.$$

6. (**Analytical**) Give a proof of the existence and uniqueness of the interpolating polynomial $P_n(x)$ using Lagrange and Newton basis functions.

7. (**Analytical**) **Barycentric polynomial**: (i) Set $\hat{L}_k(x) = (x - x_0)(x - x_1)\cdots(x - x_n)$. (ii) $w_k = \dfrac{1}{\Pi_{\substack{j=0\\j\neq k}}^{n}(x_k - x_j)}$.

   (a) Prove that the jth Lagrange polynomial can be written as:

   $$L_j(x) = \hat{L}_j(x)\frac{w(x)}{(x - x_1)}.$$

   (b) Prove that the Lagrange interpolating polynomial $P_n(x)$ in this case becomes:

   $$P_n(x) = \frac{\Sigma_{k=0}^{n} f_k \dfrac{w_k}{x - x_k}}{\Sigma_{k=0}^{n} \dfrac{w_k}{x - x_k}}$$

   (c) What are the advantages of using the expression in (b) over the standard Lagrange interpolynomial $P_n(x)$?

8. (**Analytical**) Prove that the error bound for linear interpolation with the nodes x_0 and x_1 is $(x_1 - x_0)^2\frac{M}{8}$, where $|f^{(2)}(x)| \leq M$ on $[x_0, x_1]$.

9. (**Analytical**) Let $\psi_n(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$. Let $x_{i+1}-x_i = h$, $i = 0, 1, \ldots, n-1$. Then prove that

   (a) $\max\limits_{x_0 \leq x \leq x_1} |\psi_1(x)| = \dfrac{h^2}{4}$

   (b) $\max\limits_{x_0 \leq x \leq x_2} |\psi_2(x)| = \dfrac{2\sqrt{3}}{9}h^3$
   [**Hint:** To bound $|\psi_2(x)|$, shift this polynomial along the x-axis: $\hat{\psi}_2(x) = (x + h)x(x - h)$ and obtain the bound of $\hat{\psi}_2(x)$ and then find the bound of $\hat{\psi}_2(x)$ in $[-h, h]$.]

   (c) $\max\limits_{x_0 \leq x \leq x_3} |\psi_3(x)| = h^4$

   (d) $\max\limits_{x_1 \leq x \leq x_2} |\psi_3(x)| = \dfrac{9}{16}h^4$.

10. (**Analytical**) Prove the error-bound (6.11).

11. (**Computational**) Using the results of (a), (b), and (c) of Exercise 9, find the maximum absolute error for linear, quadratic, and cubic interpolation of each of the following functions and compare the results. Assume that the nodes are equally spaced. Present the results in tabular form.

    $f(x) = e^x$, $0 \leq x \leq 2$

    $f(x) = \sin x$, $0 \leq x \leq \frac{\Pi}{2}$

12. (**Computational**) The interpretation of the results of Parts (a), (b), and (c) of Exercise 9 is that if the interpolation nodes are chosen so that the interpolating point x is close to the midpoint of $[x_0, x_3]$, then the interpolation error is smaller than if x were chosen anywhere between x_0 and x_3. Verify this statement with both of the functions in Problem 11.

13. (**Computational**)

    (a) If $f(x) = \sin x$ is interpolated by a polynomial of degree 5 in $[0, 1]$, with equidistant nodes, how large can the (absolute) error be?

    (b) Compare the error in (a) with that obtained by a clamped cubic interpolant in the same interval and using the same nodes.

14. (**Computational**) Suppose that a table is to be prepared for the function $f(x) = \ln x$ on $[1, 3.5]$ with equal spacing nodes such that the interpolation with third degree polynomial will give an accuracy of $\epsilon = 5 \times 10^{-8}$. Determine how small h has to be to guarantee this accuracy.

15. (**Computational**) Repeat the previous exercise with $f(x) = \cos x$ in $[0, \pi]$.

16. (**Analytical**) Prove **Theorem 6.15**.

17. (**Computational**) Consider the following table

x	$f(x)$
1	0
1.01	0.01
1.02	0.0198
1.03	0.0296

    (a) Find an approximation of $f(1.025)$ using

        i. The monomial interpolation

        ii. Lagrange interpolating polynomial of degree 3

        iii. Newton interpolating polynomial of degree 3 based on forward differences

    (b) Compute the maximum absolute error for interpolation using the interpolating error formula. (**Theorem 6.7**) (Note that $f(x) = \ln x$.)

    (c) Compare the absolute error made in each case (i)-(iii) with the maximum absolute error.

    (d) Compare the condition numbers of each of the matrices A of the linear system $Ax = f$, for each of the above three methods.

18. (**Analytical**) Prove by induction that

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

19. (**Computational**) Consider the following table

x	$f(x)$
1	0
3	1.0986
4	1.3865
5	1.6094

(a) Using Newton's divided differences, construct the interpolating polynomial $P_3(x)$ of degree 3.

(b) Suppose now two entries $(6, 1.7918)$ and $(8, 2.0794)$ are added to the table. Construct now the Newton interpolating polynomial of degrees 4 and 5 making use of the polynomial $P_3(x)$.

(c) Using an interpolating polynomial of appropriate degrees, interpolate at $x = 3.5, 4.5, 5.5, 6.5$ and 7.5.

20. (**Analytical**) Prove that the interpolating polynomial $P_n(x)$ with $(n+1)$ distinct nodes $x_0, x_1, \ldots, x_n$ obtained by both Lagrange interpolation and Newton interpolation are the same. That is

$$\sum_{i=0}^{n} f_i L_i(x) = \sum_{i=0}^{n} f[x_0, x_1, \ldots, x_i] \times \prod_{j=0}^{i-1}(x - x_j).$$

21. (**Computational**) Consider interpolating $f(x) = \cos x$ at the nodes $x_0 = 0$, $x_1 = 0.5$, $x_2 = 1$, $x_3 = 1.5$, using

(i) Lagrange interpolation

(ii) Newton's divided differences

(a) Compute $P_3(x)$ using both Lagrange and Newton interpolation.

(b) Estimate $f(0.9)$ using $P_3(x)$ obtained by each method and compare the results.

(c) What is the maximum (absolute) error of interpolation?

(d) Compare the actual error with the maximum error.

22. (**Computational**) Repeat the previous exercise with each of the following functions: (i) $f(x) = \sin x$, (ii) $f(x) = xe^x$ and (iii) $f(x) = \sin x + \cos x$.

23. (**Analytical**) Develop the Newton's backward interpolating polynomial formula (6.19).

24. () For each function in Exercise 6.11, find the maximum error of interpolation with equally spaced Chebyshev nodes and compare this error with that obtained without the use of the Chebyshev nodes. Write your conclusions.

25. () For each function in Exercise 6.11, find the maximum error of interpolation by a clamped cubic spline with equally spaced nodes. Compare this error with the maximum errors using Lagrange interpolation and interpolation with Chebyshev nodes.

26. (**Computational**)

    (a) Find an approximate value of $\log_{10}(5)$ using Newton's forward difference polynomial of degree 3 with $x_0 = 1$, $x_1 = 2.5$, $x_2 = 4$, and $x_3 = 5.5$.

    (b) Repeat Part (a) using Newton's backward difference polynomial of degree 3.

    (c) Compare the results.

27. (**Computational**) Consider the following table:

x	$f(x)$
1	2.7185
2	14.7781
3	60.2565
4	218.3928

Interpolate at $x = 3.5$ by the interpolating polynomial of degree 3 using both Newton's forward and backward differences. Compare the results. Which one is better?

28. (**Analytical**) Show that if $f(x)$ is a polynomial of degree n, then its $(n+1)^{\text{th}}$ divided difference is zero.

29. (**Analytical**) Prove $f[x_0, x_1, \ldots, x_k] = \sum_{i=0}^{k} \frac{f(x_i)}{\xi_k'(x_i)}$, where $\xi_k(x) = (x-x_0)(x-x_1)\cdots(x-x_k)$.

30. (**Analytical**) Show that $f[x_0, x_1, \ldots, x_n]$ does not change if the nodes are rearranged. That is,

    $f[x_0, x_1, \ldots, x_n] = f[y_0, y_1, \ldots, y_n]$, where $(y_0, y_1, \ldots, y_n)$ is a permutation of $(x_0, x_1, \ldots, x_n)$.

31. (**Analytical**) **(Cubic Spline with Monomial Basis)**

    Suppose a natural cubic spline $S(x)$ has to be determined as follows:

$$\begin{cases} S_1(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3, \quad x_1 \le x \le x_2, \\ S_2(x) &= b_0 + b_1 x + b_2 x^2 + b_3 x^3, \quad x_2 \le x \le x_3. \end{cases}$$

(a) Write down the system of equations to determine the unknowns: a_0, a_1, a_2, a_3 and b_0, b_1, b_2, b_3.

(b) Explain why the above cubic spline with monomial basis is not computationally competitive with the method described in the text. Do an example to support your statement.

32. A clamped cubic spline $S(x)$ of a function $f(x)$, is defined as follows:

$$\begin{cases} S(x) & = 2x + 3x^2 + 4x^3 \qquad 0 \le x \le 1 \\ S(x) & = a + b(x-1) + c(x-1)^2 + d(x-1)^3 \end{cases}$$

with $f'(0) = f'(4)$. Determine a, b, c, and d.

33. (**Analytical**) Give a proof of the Hermite interpolation error theorem (Theorem 6.23).

34. (**Computational**) For **Example 6.29**, verify the cubic spline interpolant properties (including the natural boundary condition) for $j = 1$.

35. (**Computational**) Find the natural cubic spline function for the data of **Example 6.14** and use this function to interpolate at $x = 2.5$ and compare the result with those obtained with Newton's method. Then compare these results with the actual error.

36. (**Computational**) Using the data of Exercise 17, estimate $f(1.025)$ using a cubic clamped interpolating polynomial and compare the result with those obtained by each of the interpolating polynomials in Exercise 17(a).

37. (**Computational**) Compute approximations of the function $f(x) = x\sin(2\pi x + 1)$ on $-1 \le x \le 1$ with $h = 0.5$, using

(a) Lagrange interpolation

(b) Clamped cubic spline

(c) Hermite interpolation

Illustrate the results graphically.

38. (**Computational**) Consider the table

x	-3	-2	-1	0	1	2	3
f	1	2	5	10	5	2	1

Show by means of graphs that a natural cubic spline is a better approximation to $f(x) = 10/(1 + x^2)$ than using a single interpolating polynomial of degree 6. (Try with both Lagrange and Newton's interpolation.)

39. (**Computational**)

(a) Find the clamped cubic spline for each of the following functions with equally spaced nodes with spacing $h = 0.5$ and in the interval as indicated:

i. $f(x) = \cos x$, $0 \leq x \leq 1$

ii. $f(x) = x \ln x$, $1 \leq x \leq 2$

iii. $f(x) = x^3 - 7x + 6$, $-1 \leq x \leq 1$

iv. $f(x) = \cos \pi x$, $0 \leq x \leq 2$

(b) Interpolate $\cos(0.2)$ and $1.2 \ln(1.2)$, and $\cos(0.7854)$ using the appropriate splines and compare the results with the actual functional values.

(c) Compute $\int_0^1 \cos x \, dx$ and $\int_1^2 x \ln x \, dx$ using the obtained splines and compare the results with actual values of the integrations.

40. (**Analytical**)

(a) Show that any cubic polynomial on any closed interval is its own clamped cubic spline.

(b) Is the statement in Part (a) true for the natural cubic spline-that is, is any cubic polynomial in a closed interval its own natural cubic spline? If not, why not? Support your answer with an example.

41. (**Analytical**) Prove that a strictly diagonally dominant matrix is nonsingular and hence prove that a function $f(x)$ defined at $a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$, and differentiable at $x = a$ and $x = b$, has a unique clamped cubic spline interpolant at the nodes $x_0, x_1, \ldots, x_n$.

42. (**Analytical**)

(a) Show that it is possible to construct a quadratic spline defined by

$$S_i(x) = a_i x^2 + b_i + c_i, \quad x_i \leq x \leq x_{i+1}$$

for $i = 0, 1, \ldots, n-2$ that satisfies the following properties

i. $S_i(x_i) = f_i$, $i = 0, 1, \ldots, n-2$

ii. $S_i(x_{i+1}) = f_{i+1}$, $i = 0, 1, \ldots, n-2$

iii. $S_i'(x_i) = S_{i-1}'(x_i)$, $i = 1, 2, \ldots, n-2$

iv. $S_0''(x_0) = 0$.

(b) Find the quadratic spline that passes through the points $(0, 0)$, $(1, 1)$ and $(2, 2)$.

43. (**Applied**) A certain decaying animal population is counted every five years and the

following table is generated.

t (in years)	$P(t)$ (in millions)
0	100
5	89.5560
10	78.4905
15	67.2706
20	56.3897
25	46.2842
30	37.2687

(a) Predict the animal population in 9 years using

i. Lagrange interpolation

ii. Newton's interpolation

iii. Clamped cubic interpolation using $p'(0) = 1.9775$ and $p'(50) = 22.7112$.

(b) Compare the predicted population by each method with the actual population at $t = 9$ knowing that the population is modeled by $p(t) = \frac{300}{2+e^{0.06t}}$

(c) Using each of the interpolating polynomials obtained in Part (a), predict the rate of change of population at $t = 9$.

44. (**Applied**) The following table gives the US census population (in millions) every ten years from 1940 to 2010:

year	US population
1940	132.165
1950	151.326
1960	179.323
1970	203.302
1980	226.542
1990	248.710
2000	281.422
2010	308.400

(a) Predict the US population in the year 1995 using

i. Lagrange interpolation

ii. Newton Forward Differences interpolation

(b) What is the rate of change of population in 1995? Give results using both Lagrangian and Newton Forward Difference interpolating polynomials.

45. (**Applied**) Consider the velocity profile table of the rocket given in Section 6.2.

(a) Predict the velocity and acceleration at $t = 23$ seconds using

i. the Lagrange interpolation

ii. Newton's divided differences interpolation

(b) Predict the distance traveled by the rocket from $t = 5$ to $t = 20$ seconds using both Lagrange and Newton polynomials and compare the results.

46. (**Analytical**) Give a constructive proof to show that there exists a unique cubic spline $S(x)$ on $[a, b]$ with $S''(a) = \alpha$, $S''(b) = \beta$ where α and β are nonzero numbers.

47. Construct a cubic spline $S(x)$ for $f(x) = \sin x$ on $[0, \frac{\pi}{2}]$, with $x_i = 0 : \frac{\pi}{8} : \frac{\pi}{2}$ as nodes and compare $S(x)$ with $f(x)$ by plotting their graphs on the same co-ordinate system.

48. (**Computational and Analytical**)

(a) Prepare a table of maximum errors for a polynomial interpolation of

$$f(x) = \frac{1}{1 + x^2}$$

with $n = 2, 5, 10$ and 20 in $[-5, 5]$ using equidistant nodes. Using this table, confirm the statement: *"the error decreases initially but then increases quite rapidly as n increases"*.

(b) Give a mathematical reasoning for this behavior.

49. (**Analytical**)

(a) Prove that the properties of the polynomials $B_k(x)$ and $B_k(x)$ stated in the Hermite Interpolation Theorem (**Theorem 6.23**) hold; show that the polynomial $H_{2n+1}(x)$ defined there is a Hermite interpolating polynomial.

(b) Using the fact that $H_{2n+1}(x)$ interpolates both $f(x)$ and $f'(x)$, prove that $H_{2n+1}(x)$ is an unique Hermite interpolating polynomial.

50. Repeat Exercise 44 using Chebyshev nodes and compare the results.

51. (**Computational**) Give a graphical representation of the behavior of the higher degree interpolating polynomials for the Runge's function:

$$f(x) = \frac{1}{1 + 25x^2}$$

in $[-1, 1]$, choosing $n = 2, 5, 10$, and 15. Plot the graphs of each interpolating polynomial along with the graph of the Runge function.

52. (**Computational**) Using the error theorem for cubic spline interpolation (**Theorem 6.35**), prepare a table of maximum errors for cubic spline interpolation of

$$f(x) = \frac{1}{1 + x^2}$$

in $[-5, 5]$; at the nodes $x_i = \dfrac{(i-1)10}{n} - 5, i = 1, \ldots, n + 1$ for $n = 2, 4, \ldots, 16$.

53. (**Applied**) (**Windmill Problem Revisited**)

Recall the windmill problem described in Chapter 1. With the fixed diameter $D = 50$ meters in the formula:

$$EP = 0.01328D^2V^3,$$

we have generated the following table relating electric output (EP) to velocity of the wind (V):

$V =$ Wind Speed (m/s)	5	10	15	20
$EP =$ Electric Power (w)	41	332	1120	2656

(a) Use Newton's polynomials of degree 1, 2, and 3 to predict the EP when the wind speed is 12 m/s.

(b) Use Lagrange polynomials of degree 3 to predict EP for the same speed as part (a).

(c) Compare the results in both parts (a) and (b) with the actual EP obtained by the given formula.

54. (**Analytical**) Prove Properties I, II, and III of the Bezier curve.

55. (**Analytical**) Prove that the Bezier curve, as defined in Definition 6.34, is a convex combination of its control points.

56. (**Computational**) Find the Bezier curve of degree 3 for each of the following sets of control points:

(a) $(0,0),(1,1),(2,1),(4,3)$

(b) $(2,2),(1,2),(3,0),(4,2)$

(c) $(2,2),(4,3),(5,5),(10,0)$

57. (**Computational**) Find the control points of the Bezier curves defined by each of the following parametric equations:

$$\begin{cases} \text{(i) } x(t) &= -2t^3 + 9t^2 - 6t \\ \text{(ii) } y(t) &= -7t^3 + 6t^2 + 3t + 1 \end{cases}$$

58. (**Analytical**) Prove that the Bezier curve with control points $P_0(x_0, y_0) = P_1(x_1, y_1)$ and $P_2(x_2, y_2) = P_3(x_3, y_3)$ is just a line.

6.14 MATLAB Problems on Interpolation

M6.1 (Implementations of Lagrange and Monomial Interpolation) Write MATLAB functions in the following formats to implement the monomial and Lagrange methods:

$$\begin{aligned}\text{function } [YI, PC] &= \textbf{monomial } (X, Y, XI)\\ \text{function } [YI, PC] &= \textbf{lagrange } (X, Y, XI)\end{aligned}$$

You should use Horner's scheme for polynomial evaluation in the program for "**monomial**" interpolation.

$$\textbf{Inputs}: \begin{cases} X & \text{– The vector containing the abscissas of the data}\\ Y & \text{– The vector containing the ordinates of the data}\\ XI & \text{– The vector containing the abscissas of interpolating points}\end{cases}$$

(X, Y have the same length)

$$\textbf{Outputs}: \begin{cases} PC & \text{– The vector containing the coefficients of the interpolating polynomial}\\ YI & \text{– The vector containing the interpolated values}\end{cases}$$

M6.2 Write MATLAB functions in the following formats to implement Newton's methods with divided and forward differences.

Your *program should contain the codes for* the Newton-Horner's scheme to evaluate the Newton interpolating polynomials:

$$\begin{aligned}\text{function } [YI, DD, PC] &= \textbf{newintdd } (X, Y, n, XI)\\ \text{function } [YI, FD, PC] &= \textbf{newintfd } (X, Y, XI)\end{aligned}$$

$$\textbf{Inputs}: \begin{cases} X- & \text{same as in M6.1}\\ Y- & \text{same as in M6.1}\\ n- & \text{degree of the interpolating polynomial}\\ XI- & \text{same as in M6.1}\end{cases}$$

$$\textbf{Outputs}: \begin{cases} YI- & \text{same as in M6.1}\\ PC- & \text{same as in M6.1}\\ DD- & \text{The diagonal entries of the divided differences table}\\ FD- & \text{The diagonal entries of the forward differences table}\end{cases}$$

M6.3 Solve the following problems using MATLAB functions **monomial**, **lagrange**, and **newtintfd**, written in problems M6.1 and M6.2. Present your results in both tabular and graphical forms. The table should contain results from all the methods for comparison purposes.

A single plot should contain the graphs of the function (or data set) and the interpolating polynomials obtained by all three methods.

(a) (**US Population Prediction**) Using the Table in Exercise #, predict the US population in the years: 1945 : 10 : 2005.

(b) (**Decaying Animal Population**) Using the Table in Exercise #, predict the animal population in the years: $t = 6, 8, 12, 15, 18, 22,$ and 29.

(c) (**Velocity Profile of the Rocket**) Using the Table of the upward velocity profile of a rocket in Section 6.2, find the distance traveled by the rocket in 30 seconds. (Compute the interpolating polynomial $v(t)$ and integrating it from $t = 0$ to $t = 30$).

(d) (**Predicting Electric Output (EP) from Windmill Velocity (V)**) Recall the formula

$$EP = 0.01328D^2V^3$$

relating the windmill velocity $V(s)$ with electric power output $EP(w)$. Taking $D =$ 50 meters, generate a table of values (V, EP) with $V = 5 : 5 : 50$.

Use the table to predict $EP(w)$ when the wind speed is $VI = 8 : 4 : 48$ m/s.

M6.4 (**Implementations of Natural and Cubic Splines**) Write two MATLAB functions, **csplinat** and **csplincl**, to implement the algorithms for natural and clamped splines, as follows:

$$\begin{aligned}\text{function } [YI, PC] &= \textbf{csplinat } [X, Y, XI] \\ \text{function } [YI, PC] &= \textbf{csplind } [X, Y, XI, DERX_0, DERX_N]\end{aligned}$$

$$\textbf{Inputs}: \begin{cases} X, Y- & \text{same as in problem M6.1} \\ XI- & \text{same as in M6.1} \\ DERX_0- & \text{the first derivative at } x_0 \\ DERX_N- & \text{the first derivative at } x_n \end{cases}$$

$$\textbf{Outputs}: \begin{cases} YI- & \text{same as in problem M6.1} \\ PC- & \text{same as in M6.1} \end{cases}$$

M6.5 (**Interpolating Natural Log with Cubic Splines**) Find both the natural and clamped cubic splines for $f(x) = \ln x$, $1 \le x \le 2$ with equally spaced nodes of $h = 0.1$. Use these splines and MATLAB function, **spline** to interpolate $\ln x$ at $XI = [1.05 : 0.4 : 2]^T$.

(a) Present the results in tabular form containing the interpolated values obtained by each method at each point of interpolation, the corresponding functional values, and errors.

(b) Present the graphs of $f(x) = \ln x$ and interpolating splines in a single plot.

M6.6 (**Interpolating** $\sin \mathbf{x}$ **with Cubic Splines**). Repeat problem M6.5 with $f(x) = \sin x$, $0 \le x \le 2\pi$ using equally spaced nodes of $h = \frac{\pi}{8}$. Interpolate at $XI = [\frac{\pi}{16} : \frac{\pi}{2} : 2\pi]^T$.

M6.7 Approximate $f(x) = x\sin(2\pi x + 1)$, $-1 \leq x \leq 1$ with 11 equally spaced nodes, using a clamped cubic spline and MATLAB function "spline". Draw the splines and the the function in a single plot.

M6.8 (**Experiment with the Runge Function**). [The purpose of this experiment is to demonstrate the fact that the approximations obtained for the Runge function using the Chebyshev nodes and cubic splines are much more accurate than these obtained by the higher-order single interpolating polynomials.]

(a) Interpolate $f(x) = \dfrac{1}{1+25x^2}$ using Newton polynomials of both degrees 5 and 10 based on equally spaced nodes over $[-1, 1]$.

Draw the graph of $f(x)$ and those of the interpolating polynomials in a single plot.

(b) Repeat step (a) with the Chebyshev nodes (that is, with the nodes at the zeros of the Chebyshev polynomial in $[-1, 1]$).

(c) Interpolate using a clamped cubic spline with 11 equally spaced nodes.

(d) Prepare a table of interpolation errors in the following format with $x = -1 : 0.2 : 1$.

x	$f(x)$	Newton $P_{10}(x)$	Chebyshev $P_{10}(x)$	Spline (x)	Newton Error	Chebyshev Error	Spline Error

Newton $P_{10}(x)$–	The value of the 10th degree Newton interpolating polynomial at x with standard nodes
Chebyshev $P_{10}(x)$–	The value of the 10th degree Newton interpolating polynomial with Chebyshev nodes
spline (x)–	The value of the spline function at x

M6.9 If $x_0, x_1, \ldots, x_n$ are the Chebyshev nodes on $[-1, 1]$, then verify the assertion

$$\max_{-1 \leq x \leq 1} |(x - x_0)(x - x_1)\cdots(x - x_n)| = 2^{-n}$$

numerically for $n = 4, 8, 16$, and 32.

M6.10 (a) Write MATLAB programs to construct and plot a Bezier curve with a set of N control points.

(b) Extend your program to construct and plot a composite Bezier curve.

(c) Using your program(s), draw the following characters by choosing an appropriate set of control points for each character:

i. Letter T

ii. Letter B

iii. Letter D

iv. Number 5

v. The Greek Letter β

M6.11 (**Interpolating the Gamma Function**). The well-known gamma function is defined by $\Gamma(x) = \int_0^\infty t^{x-1}e^{-x}dx$, $x > 0$; one calculates from the above integral, $\Gamma(n+1) = n!$, for an integer n.

(a) Find an interpolating polynomial of degree 5 using the data: $(1, \Gamma(0)), (2, \Gamma(2)), (3, \Gamma(3)), (4, \Gamma(4))$, and $(5, \Gamma(5)), (6, \Gamma(6))$.

(b) Find a cubic spline to interpolate with the same data in (a).

(c) Plot the interpolating polynomial, cubic spline, and the gamma functions using the MATLAB built-in function **gamma** in a single plot.

(d) Interpolate $\Gamma(\frac{1}{2} : 1 : 5)$ using both polynomial and spline interpolants and compare your answers.

M6.12 Using the built-in MATLAB functions, **interpl**, **spline**, **pchip**, and **newintdd** from Exercise M6.2, present a graphical comparison of the following interpolants of the Runge function

$$f(x) = \frac{1}{1 + 25x^2}$$

in $-1 \leq x \leq 1$ with equally spaced points.

(a) a 9-degree interpolating polynomial.

(b) piecewise linear spline.

(c) piecewise cubic spline.

(d) piecewise Hermite cubic interpolant.

Plot the interpolants either in a single or separate graphs.

M6.13 Using MATLAB functions **spline** and **pchip**, design an experiment to compare a cubic spline interpolant with a piecewise cubic Hermite interpolant. Your experiment should support the following facts:

(a) a spline interpolant is smoother than pchip,

(b) spline interpolates the data of a smooth function accurately,

(c) **pchip** "has no overshoots, and less oscillations if the data are not of a smooth function".