

---

---

# CHAPTER 2

---

## INTERPOLATION

1	General description . . . . .	156
2	Interpolation methods . . . . .	156
2.1	Lagrange's interpolation formula . . . . .	156
2.2	Newton's forward interpolation formula . . . . .	158
2.3	Newton's backward interpolation formula . . . . .	162
2.4	Newton's divided difference formula . . . . .	165
2.5	Spline interpolation . . . . .	168
3	Tasks . . . . .	175

# 1 General description

Interpolation is a fundamental technique in numerical analysis, widely used for data fitting and function approximation, as well as supporting other numerical methods including root-finding, optimization, integration, and differentiation. The method involves constructing new data points within the range of a discrete set of known data points  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$  by determining an interpolating function  $f(x)$  - called the *interpolant* - that satisfies  $f(x_i) = y_i$  for all given points. When this interpolating function is specifically an algebraic polynomial, it is commonly known as an *interpolation polynomial*.

This chapter provides a rigorous examination of fundamental interpolation techniques, presenting both their theoretical underpinnings and practical implementation considerations. The study begins with classical polynomial interpolation methods, progressing to more advanced approaches for specialized applications. For datasets with uniform spacing, we thoroughly investigate two complementary Newtonian formulations: the *forward interpolation formula*, particularly suited for estimating values near the start of the data range, and its counterpart, the *backward interpolation formula*, which demonstrates superior performance when working with terminal data points. When dealing with non-uniformly distributed data points, the chapter develops the elegant *Lagrange interpolation formula*, notable for its conceptual simplicity and direct construction. Parallel to this, we present the computationally efficient *Newton's divided difference method*, which offers a systematic recursive approach to polynomial generation. The discussion then advances to modern piecewise interpolation techniques, with particular emphasis on *cubic spline interpolation*. This method provides smooth approximations through carefully constructed piecewise polynomials that maintain continuity while effectively minimizing undesirable oscillatory behavior between nodes. For each method, we provide detailed algorithmic implementations and clear block diagrams that illustrate the computational workflow, enabling readers to both understand the theoretical foundations and implement these techniques in practical applications.

# 2 Interpolation methods

## 2.1 Lagrange's interpolation formula

Consider a set of  $n + 1$  data points  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$  for  $n$  unequal intervals, where each  $y_i = f(x_i)$  for a smooth function  $f$  (i.e., a function possessing derivatives of all orders). The process of determining a unique  $n$ -th degree polynomial  $P(x)$  that satisfies the interpolation conditions  $P(x_i) = y_i$  for all  $i = 0, 1, \dots, n$  is known as *Lagrange interpolation*. The resulting polynomial  $P(x)$  is referred to as the Lagrange interpolating polynomial or simply the *Lagrange interpolant*, named in honor of the renowned French mathematician Joseph-Louis Lagrange (1736 - 1813).

The Lagrange interpolation polynomial  $P(x)$  is unique and passes through the specified

points. The degree of the polynomial is at most  $n - 1$  and it is defined as

$$\begin{aligned} P(x) &= \frac{(x - x_1)(x - x_2) \cdots (x - x_n)}{(x_0 - x_1)(x_0 - x_2) \cdots (x_0 - x_n)} y_0 + \frac{(x - x_0)(x - x_2) \cdots (x - x_n)}{(x_1 - x_0)(x_1 - x_2) \cdots (x_1 - x_n)} y_1 + \\ &\quad + \cdots + \frac{(x - x_0)(x - x_1) \cdots (x - x_{n-1})}{(x_n - x_0)(x_n - x_1) \cdots (x_n - x_{n-1})} y_n = \sum_{i=0}^n y_i L_i(x), \end{aligned} \quad (2.1)$$

where the Lagrange basis polynomials  $L_i(x)$  are given by

$$L_i(x) = \prod_{\substack{0 \leq j \leq n \\ j \neq i}} \frac{x - x_j}{x_i - x_j}.$$

### *Example: Lagrange's interpolation formula*

Consider the function  $f(x) = 1/x$  as an illustrative example. We will construct the Lagrange interpolating polynomial using the nodes  $x_0 = 1$ ,  $x_1 = 1.5$ , and  $x_2 = 4$ . The corresponding function values are  $y_0 = f(x_0) = 1$ ,  $y_1 = f(x_1) = 2/3$ , and  $y_2 = f(x_2) = 1/4$ . Note that Lagrange interpolation does not require the nodes to be equally spaced. Since we have three points ( $n = 2$ ), the resulting interpolating polynomial will be quadratic.

First, we compute the Lagrange basis polynomials

$$\begin{aligned} L_0(x) &= \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{(x - 1.5)(x - 4)}{(1 - 1.5)(1 - 4)} = \frac{2}{3}x^2 - \frac{11}{3}x + 4 \\ L_1(x) &= \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = \frac{(x - 1)(x - 4)}{(1.5 - 1)(1.5 - 4)} = -\frac{4}{5}x^2 + 4x - \frac{16}{5} \\ L_2(x) &= \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(x - 1)(x - 1.5)}{(4 - 1)(4 - 1.5)} = \frac{2}{15}x^2 - \frac{1}{3}x + \frac{1}{5} \end{aligned}$$

The interpolating polynomial is then given by

$$P(x) = \sum_{i=0}^2 y_i L_i(x) = 1 \cdot L_0(x) + \frac{2}{3} \cdot L_1(x) + \frac{1}{4} \cdot L_2(x) = \frac{1}{6}x^2 - \frac{13}{12}x + \frac{23}{12}.$$

Let's evaluate the polynomial  $P(x)$  at  $x = 2$  and compare with the exact value  $f(2) = 0.5$ .

$$P(2) = \frac{1}{6}(2)^2 - \frac{13}{12}(2) + \frac{23}{12} = \frac{5}{12} \approx 0.4167$$

The approximation error at  $x = 2$  is

$$|P(2) - f(2)| \approx |0.4167 - 0.5| = 0.0833.$$

The procedure for the computation of the Lagrange polynomials is presented in Figure 2.1. The block diagram illustrates the step-by-step process of Lagrange interpolation, which

computes an interpolated value at a given point  $x^*$  using  $n$  data points  $(x_i, y_i)$ . The algorithm begins by initializing the parameters and reading the input data. It then initializes a sum variable to zero  $sum = 0$ , which will accumulate the final result. The core of the method involves nested loops. The outer loop  $i$  iterates over each data point to construct the corresponding Lagrange basis polynomial  $L_i(x^*)$ . The inner loop  $j$  calculates the product of terms for  $L_i(x^*)$  by skipping the case where  $i = j$  to avoid division by zero. For each valid term, the algorithm updates the product using the formula  $\frac{x^* - x_j}{x_i - x_j}$ . After computing the basis polynomial, the result is multiplied by  $y_i$  and added to the running sum. Once all iterations are complete, the algorithm outputs the accumulated sum, which represents the interpolated value  $P(x^*)$ , and terminates. This structured approach efficiently combines the contributions of all basis polynomials to produce the final result.

## 2.2 Newton's forward interpolation formula

Newton's forward interpolation formula provides an efficient polynomial approximation for equally spaced data points, particularly effective when estimating values near the *beginning* of a dataset. Given a set of  $n + 1$  equally spaced data points  $(x_i, y_i)$ , where  $x_i = x_0 + ih$  for  $i = 0, 1, 2, \dots, n$ , an interpolation polynomial  $P(x)$  is constructed based on direct differences using the formula

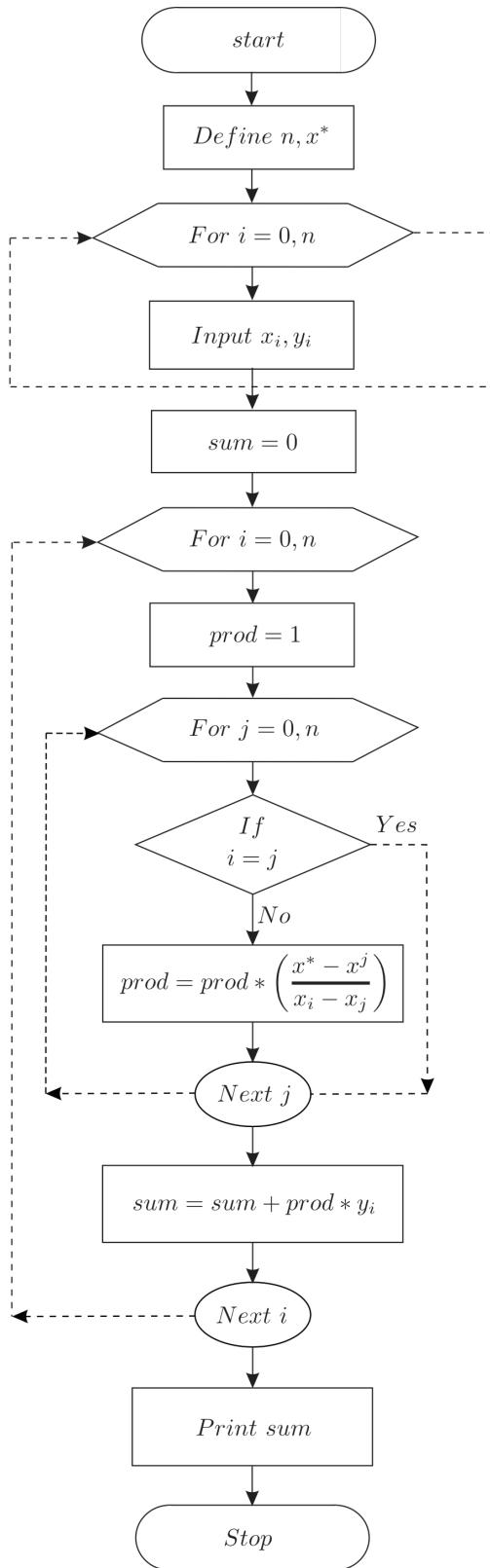
$$\begin{aligned} P(x) &= y_0 + \frac{\Delta y_0}{h}(x - x_0) + \frac{\Delta^2 y_0}{2!h^2}(x - x_0)(x - x_1) + \frac{\Delta^3 y_0}{3!h^3}(x - x_0)(x - x_1)(x - x_2) + \dots \\ &\quad \dots + \frac{\Delta^n y_0}{n!h^n}(x - x_0)(x - x_1) \dots (x - x_{n-1}) = \\ &= y_0 + \sum_{k=1}^n \left[ \frac{\Delta^k y_0}{h^k k!} \prod_{i=0}^{k-1} (x - x_{n-i}) \right], \end{aligned} \tag{2.2}$$

where  $P(x)$  is the polynomial approximation of the function  $f(x)$  at point  $x$  and  $y_0 = f(x_0)$  represents the value of the function at the initial data point  $x_0$ ,  $\Delta y_i = y_{i+1} - y_i$  represents the first-order forward difference for  $i = 0, 1, \dots, n - 1$ ,  $\Delta^2 y_i = \Delta y_{i+1} - \Delta y_i$  is the second-order forward difference for  $i = 0, 1, \dots, n - 2$ ,  $\Delta^k y_i = \Delta^{k-1} y_{i+1} - \Delta^{k-1} y_i$  represents the  $k$ -th order forward difference for  $i = 0, 1, \dots, n - k$ ,  $h = x_{i+1} - x_i$  is constant step size between points and  $k!$  is the factorial of  $k$ .

To simplify the formula (2.2), we introduce a new variable  $u$ , defined as

$$u = \frac{x - x_0}{h}.$$

This variable represents the normalized distance between the point  $x$  and the initial data point  $x_0$ , scaled by the spacing  $h$ . Substituting  $u$  into the formula, we can rewrite Newton's forward interpolation formula as



**Figure 2.1:** Block diagram of the Lagrange interpolation formula

$$\begin{aligned}
P(x) &= y_0 + u\Delta y_0 + \frac{u(u-1)}{2!}\Delta^2 y_0 + \frac{u(u-1)(u-2)}{3!}\Delta^3 y_0 + \dots \\
&\dots + \frac{u(u-1)\cdots(u-n+1)}{n!}\Delta^n y_0 = y_0 + \sum_{k=1}^n \left[ \frac{u(u-1)\cdots(u-k+1)}{k!} \Delta^k y_0 \right]. \tag{2.3}
\end{aligned}$$

The Newton forward interpolation formula is computationally efficient and particularly suitable for algorithmic implementation. The required forward differences  $\Delta^n y_i$  can be systematically computed through a forward difference table (Table 2.1), where the boxed elements in the first row are the only differences used in Newton's forward interpolation formulas (2.2) and (2.3).

$x$	$y$	$\Delta y$	$\Delta^2 y$	$\Delta^3 y$	$\dots$	$\Delta^{n-1} y$	$\Delta^n y$
$x_0$	$y_0$	$\boxed{\Delta y_0}$	$\boxed{\Delta^2 y_0}$	$\boxed{\Delta^3 y_0}$	$\dots$	$\boxed{\Delta^{n-1} y_0}$	$\boxed{\Delta^n y_0}$
$x_1$	$y_1$	$\Delta y_1$	$\Delta^2 y_1$	$\Delta^3 y_1$	$\dots$	$\Delta^{n-1} y_1$	—
$x_2$	$y_2$	$\Delta y_2$	$\Delta^2 y_2$	$\Delta^3 y_2$	$\dots$	—	—
$x_3$	$y_3$	$\Delta y_3$	$\Delta^2 y_3$	$\Delta^3 y_3$	$\dots$	—	—
...	...	...	...	...	...	—	—
$x_{n-1}$	$y_{n-1}$	$\Delta y_{n-1}$	—	—	...	—	—
$x_n$	$y_n$	—	—	—	...	—	—

**Table 2.1:** Tableau for the computation of the coefficients of the polynomial by Newton's forward interpolation method

Figure 2.2 presents the Newton's forward interpolation algorithm, which begins by accepting the number of data points  $n$ , target point  $x^*$ , and dataset  $(x_i, y_i)$ . It first computes the step size  $h = x_2 - x_1$  and normalized position  $u = (x^* - x_1)/h$ . The algorithm then constructs a forward difference table through recursive relations  $\Delta^k y_i = \Delta^{k-1} y_{i+1} - \Delta^{k-1} y_i$  for  $k = 1$  to  $n-1$ , where only the first row elements (boxed in Figure 2.2) are used in subsequent calculations. Initializing  $y^* = y_1$  and a temporary variable  $u_1 = 1$ , it iteratively updates the estimate  $y^*$  for each difference order  $j$ . This computationally efficient ( $\mathcal{O}(n^2)$ ) process systematically incorporates higher-order differences while avoiding explicit polynomial construction, making it particularly suitable for equally-spaced data interpolation.

### Example: Newton's forward interpolation formula

To illustrate the application of Newton's forward interpolation formula, suppose we have the following data for the population of Kazakhstan (in millions) for the years 2010, 2015, 2020, and 2025:

Year (x)	Population (y)
2010	16.5
2015	17.5
2020	18.7
2025	20.0

We want to estimate the population in the year  $x = 2018$ . Using the steps outlined above, we can compute the forward differences  $\Delta^k y_i$ , determine the value of  $u$ , and apply the formula (2.3) to obtain the interpolated value  $P(x)$ .

### Step 1. Calculate the forward differences

First, we calculate the forward differences ( $\Delta y$ ,  $\Delta^2 y$  and  $\Delta^3 y$ ):

$x$	$y$	$\Delta y$	$\Delta^2 y$	$\Delta^3 y$
$x_0 = 2010$	$y_0 = 16.5$	1.0	0.2	-0.1
$x_1 = 2015$	$y_1 = 17.5$	1.2	0.1	
$x_2 = 2020$	$y_2 = 18.7$	1.3		
$x_3 = 2025$	$y_3 = 20.0$			

where  $\Delta y_0 = y_1 - y_0 = 17.5 - 16.5 = 1.0$ ,  $\Delta y_1 = y_2 - y_1 = 18.7 - 17.5 = 1.2$ ,  $\Delta y_2 = y_3 - y_2 = 20.0 - 18.7 = 1.3$ ,  $\Delta^2 y_0 = \Delta y_1 - \Delta y_0 = 1.2 - 1.0 = 0.2$ ,  $\Delta^2 y_1 = \Delta y_2 - \Delta y_1 = 1.3 - 1.2 = 0.1$ ,  $\Delta^3 y_0 = \Delta^2 y_1 - \Delta^2 y_0 = 0.1 - 0.2 = -0.1$ .

### Step 2. Determine the value of $u$

The formula for  $u$  is

$$u = \frac{x - x_0}{h},$$

where  $x$  is the year we want to estimate (2018),  $x_0$  is the first year in the data (2010),  $h$  is the interval between years (5 years) and then

$$u = \frac{2018 - 2010}{5} = \frac{8}{5} = 1.6.$$

### Step 3. Apply Newton's forward interpolation formula

The formula is

$$P(x) = y_0 + u \cdot \Delta y_0 + \frac{u(u-1)}{2!} \cdot \Delta^2 y_0 + \frac{u(u-1)(u-2)}{3!} \cdot \Delta^3 y_0.$$

Plugging in the values

$$P(2018) = 16.5 + 1.6 \cdot 1.0 + \frac{1.6 \cdot 0.6}{2} \cdot 0.2 + \frac{1.6 \cdot 0.6 \cdot (-0.4)}{6} \cdot (-0.1).$$

Calculate each term and sum all the terms

$$P(2018) = 16.5 + 1.6 + 0.096 + 0.0064 = 18.2024$$

The estimated population of Kazakhstan in the year 2018 is approximately 18.20 million using Newton's forward interpolation formula.

### 2.3 Newton's backward interpolation formula

Unlike Newton's forward interpolation formula, which starts from the first data point, the backward formula starts from the last data point and works backward. This method is particularly useful when the point of interest is near the end of the data range.

The general form of Newton's backward interpolation formula is given by

$$\begin{aligned}
 P(x) &= y_n + \frac{\nabla y_n}{h}(x - x_n) + \frac{\nabla^2 y_n}{2!h^2}(x - x_n)(x - x_{n-1}) + \frac{\nabla^3 y_n}{3!h^3}(x - x_n)(x - x_{n-1})(x - x_{n-2}) + \dots \\
 &\quad \dots + \frac{\nabla^n y_n}{n!h^n}(x - x_n)(x - x_{n-1}) \dots (x - x_1) = \\
 &= y_n + \sum_{k=1}^n \left[ \frac{\nabla^k y_n}{h^k k!} \prod_{i=0}^{k-1} (x - x_{n-i}) \right], 
 \end{aligned} \tag{2.4}$$

where  $P(x)$  is the interpolated value at point  $x$ ,  $y_n$  is the value of the function at the last data point  $x_n$ ,  $\nabla y_n = y_n - y_{n-1}$  is the first-order backward difference,  $\nabla^k y_n$  is the  $k$ -th order backward difference, defined recursively as  $\nabla^k y_n = \nabla^{k-1} y_n - \nabla^{k-1} y_{n-1}$ ,  $h = x_{i+1} - x_i$  is the spacing between consecutive data points,  $x_{n-i}$  are the  $x$ -values in descending order from the last data point,  $n$  is the degree of interpolation.

To simplify the formula for Newton's backward interpolation method, we introduce a new variable  $v$ , defined as

$$v = \frac{x - x_n}{h}$$

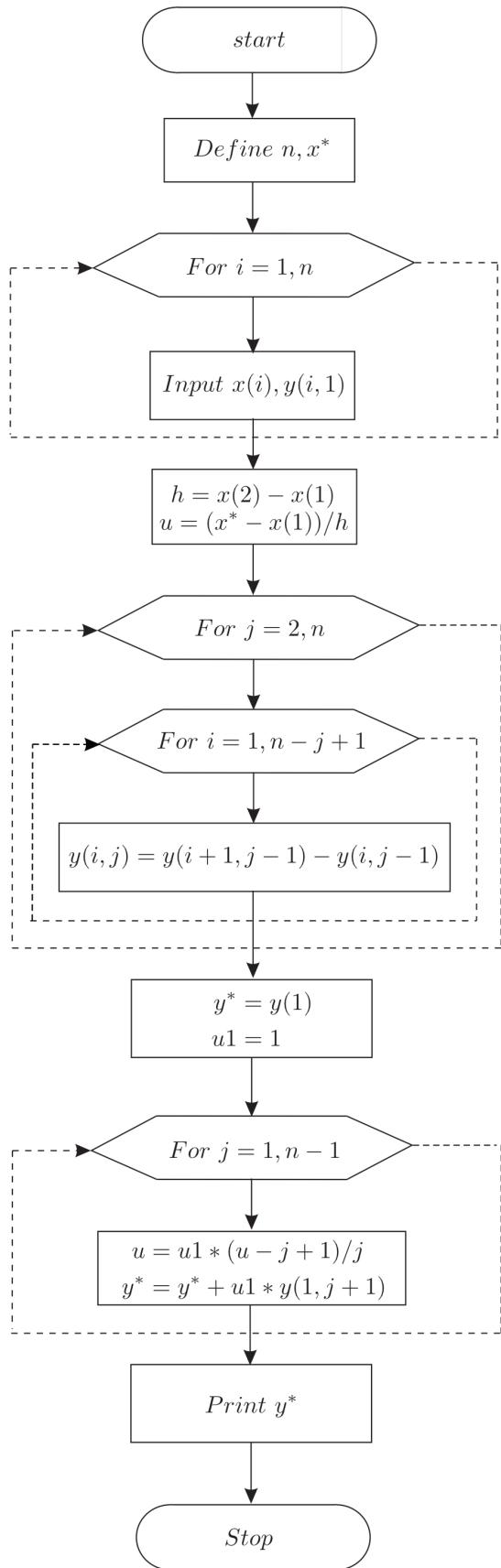
This variable represents the normalized distance between the point  $x$  and the last data point  $x_n$ , scaled by the spacing  $h$ . Substituting  $v$  into the formula, we can rewrite Newton's backward interpolation formula (2.4) as following

$$\begin{aligned}
 P(x) &= y_n + v \nabla y_n + \frac{v(v+1)}{2!} \nabla^2 y_n + \frac{v(v+1)(v+2)}{3!} \nabla^3 y_n + \dots \\
 &\quad \dots + \frac{v(v+1) \dots (v+n-1)}{n!} \nabla^n y_n = y_n + \sum_{k=1}^n \left[ \frac{v(v+1) \dots (v+k-1)}{k!} \nabla^k y_n \right]. 
 \end{aligned} \tag{2.5}$$

The backward differences required for the formulas (2.4) and (2.5) can be systematically computed using a backward difference table (see Table 2.2).

**Example: Newton's backward interpolation formula**

To illustrate the application of Newton's backward interpolation formula, consider the previous data for the population of Kazakhstan (in millions) for the years 2010, 2015,



**Figure 2.2:** Block diagram of the Newton's forward interpolation method

$x$	$y$	$\nabla y$	$\nabla^2 y$	$\nabla^3 y$	$\dots$	$\nabla^{n-1} y$	$\nabla^n y$
$x_0$	$y_0$	$\nabla y_0$	—	—	—	—	—
$x_1$	$y_1$	$\nabla y_1$	$\nabla^2 y_1$	—	...	—	—
$x_2$	$y_2$	$\nabla y_2$	$\nabla^2 y_2$	$\nabla^3 y_2$	...	—	—
$x_3$	$y_3$	$\nabla y_3$	$\nabla^2 y_3$	$\nabla^3 y_3$	...	—	—
...	...	...	...	...	...	—	—
$x_{n-1}$	$y_{n-1}$	$\nabla y_{n-1}$	$\nabla^2 y_{n-1}$	$\nabla^3 y_{n-1}$	...	$\nabla^{n-1} y_{n-1}$	—
$x_n$	$y_n$	$\nabla y_n$	$\nabla^2 y_n$	$\nabla^3 y_n$	...	$\nabla^{n-1} y_n$	$\nabla^n y_n$

**Table 2.2:** Tableau for the computation of the coefficients of the polynomial by Newton's backward interpolation method

2020, and 2025. Suppose we want to estimate the population in the year 2023. Using the steps outlined above, we can compute the backward differences, determine the value of  $v$ , and apply the formula (2.5) to obtain the interpolated value  $P(x)$ .

### Step 1. Compute backward differences

The backward differences are calculated as follows

$x$	$y$	$\nabla y$	$\nabla^2 y$	$\nabla^3 y$
2010	16.5	—	—	—
2015	17.5	1.0	—	—
2020	18.7	1.2	0.2	—
2025	20.0	1.3	0.1	-0.1

### Step 2. Determine $v = (x - x_3)/h$

For  $x = 2023$ ,  $x_3 = 2025$ , and  $h = 5$ :

$$v = \frac{2023 - 2025}{5} = -0.4.$$

### Step 3: Apply Newton's backward interpolation formula

Using the formula

$$P(2023) = y_3 + v\nabla y_3 + \frac{v(v+1)}{2!}\nabla^2 y_3 + \frac{v(v+1)(v+2)}{3!}\nabla^3 y_3.$$

Substituting the values

$$P(2023) = 20.0 + (-0.4)(1.3) + \frac{(-0.4)(-0.4+1)}{2}(0.1) + \frac{(-0.4)(-0.4+1)(-0.4+2)}{6}(-0.1).$$

Calculating each term

$$P(2023) = 20.0 - 0.52 + 0.012 - 0.0048 = [19.4872].$$

Algorithm ?? presents Newton's backward interpolation method for estimating function values near the end of a dataset, systematically constructing a backward difference table from equally spaced points  $(x_i, y_i)$  with step size  $h$  before evaluating the interpolating polynomial. The algorithm first initializes the difference table with  $y$ -values ( $\nabla^0 f_i = y_i$ ) and recursively computes higher-order backward differences  $\nabla^j f_i = \nabla^{j-1} f_i - \nabla^{j-1} f_{i-1}$  for  $j = 1$  to  $n$  and  $i = j$  to  $n$ . It then calculates the normalized distance  $v = (x - x_n)/h$  and evaluates the polynomial through an accumulating sum  $P(x) = y_n + \sum_{j=1}^n \left( \prod_{k=1}^j \frac{v+k-1}{k} \right) \nabla^j f_n$ , utilizing only the terminal diagonal entries ( $\nabla^j f_n$ ) of the difference table. This approach is computationally efficient ( $\mathcal{O}(n^2)$  for table construction,  $\mathcal{O}(n)$  per evaluation) and numerically stable for moderate  $n$  and small  $|v|$ , making it particularly suitable when new data points are appended sequentially or when interpolating near the end of tabulated values.

---

**Algorithm 26** Newton's backward difference interpolation

---

**Require:** Data points  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$  with  $x_i = x_0 + ih$  (equally spaced)

**Require:** Target value  $x$  where  $x_n - x_0 \geq x - x_n \geq 0$

**Ensure:** Interpolated value  $P(x)$  at target point

```

1: Initialize Backward Difference Table:
2: Allocate  $\nabla^j f_i$  for  $i, j \in \{0, \dots, n\}$ 
3: for  $i \leftarrow 0 \dots n$  do
4:    $\nabla^0 f_i \leftarrow y_i$                                       $\triangleright$  Initialize 0th differences
5: end for
6: Compute Backward Differences:
7: for  $j \leftarrow 1 \dots n$  do
8:   for  $i \leftarrow j \dots n$  do
9:      $\nabla^j f_i \leftarrow \nabla^{j-1} f_i - \nabla^{j-1} f_{i-1}$        $\triangleright$  Recursive difference calculation
10:    end for
11: end for
12: Construct Interpolating Polynomial:
13:    $v \leftarrow \frac{x-x_n}{h}$                                  $\triangleright$  Normalized distance from endpoint
14:    $P \leftarrow \nabla^0 f_n$                                      $\triangleright$  Start with  $y_n$ 
15:   term  $\leftarrow 1$                                           $\triangleright$  Initialize product term
16:   for  $j \leftarrow 1 \dots n$  do
17:     term  $\leftarrow$  term  $\times \frac{v+j-1}{j}$                    $\triangleright$  Update falling factorial term
18:      $P \leftarrow P + \text{term} \times \nabla^j f_n$              $\triangleright$  Add next term
19:   end for
20: return  $P$                                              $\triangleright$  Return interpolated value

```

---

## 2.4 Newton's divided difference formula

Newton's divided difference formula is particularly useful when the data points are not evenly spaced. The formula is derived from the concept of divided differences, which provides a systematic way to compute the coefficients of the polynomial.

At this stage, in order to simplify notation, we will denote the divided difference approximations of the first order derivatives by  $f[x_i, x_{i+1}]$ , the divided difference approximation of

$x$	$y$	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$	$\dots$	$f[x_0, x_1, \dots, x_n]$
$x_0$	$y_0 = f[x_0]$	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$	$\dots$	$f[x_0, x_1, \dots, x_n]$
$x_1$	$y_1$	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	$f[x_1, x_2, x_3, x_4]$	$\dots$	-
$x_2$	$y_2$	$f[x_2, x_3]$	$f[x_2, x_3, x_4]$	$f[x_2, x_3, x_4, x_5]$	$\dots$	-
$x_3$	$y_3$	$f[x_3, x_4]$	$f[x_3, x_4, x_5]$	-	$\dots$	-
...	...	...	...	...	...	-
$x_{n-1}$	$y_{n-1}$	$f[x_{n-1}, x_n]$	-	-	$\dots$	-
$x_n$	$y_n$	-	-	-	$\dots$	-

**Table 2.3:** Newton's divided differences tableau for the computation of the coefficients of the polynomial

the second order derivatives by  $f[x_i, x_{i+1}, x_{i+2}]$  and that of the  $i$ -th order by  $f[x_0, x_1, \dots, x_i]$ .

Using these notations, Newton's form of the interpolating polynomial is written in the form

$$P(x) = \sum_{i=0}^n \left[ f[x_0, x_1, \dots, x_i] \prod_{j=0}^{i-1} (x - x_j) \right] \quad (2.6)$$

where the first divided difference is defined as

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

and the second divided difference by

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}.$$

General form of the  $i$ -th divided difference is given by

$$f[x_0, x_1, \dots, x_i] = \frac{f[x_1, x_2, \dots, x_i] - f[x_0, x_1, \dots, x_{i-1}]}{x_i - x_0}, \quad x_i \neq x_0$$

and

$$f[x_i, x_{i+1}, \dots, x_j] = \frac{f[x_{i+1}, x_{i+2}, \dots, x_j] - f[x_i, x_{i+1}, \dots, x_{j-1}]}{x_j - x_i}, \quad x_i \neq x_j.$$

Given  $n + 1$  distinct data points  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ , the Newton interpolation polynomial  $P(x)$  in equation (2.6) can be expressed as:

$$\begin{aligned} P(x) &= f[x_0] + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + \dots \\ &\quad \dots + (x - x_0)(x - x_1) \cdots (x - x_{n-1})f[x_0, x_1, \dots, x_n] \end{aligned}$$

where  $f[x_i] = f(x_i) = y_i$  is the function value at  $x_i$ ,  $f[x_i, x_j]$  the first divided difference and higher-order divided differences are defined recursively and are presented as in the divided differences Table 2.3.

***Example: Newton's divided difference formula***

The values of  $x$  and  $y$  are as follows:

$x$	$y$
300	2.4771
304	2.4829
305	2.4843
307	2.4871

We construct Newton's divided difference table as follows:

$x$	$y$	1st Order	2nd Order
300	2.4771		
304	2.4829	$\frac{2.4829 - 2.4771}{304 - 300} = 0.0014$	
305	2.4843	$\frac{2.4843 - 2.4829}{305 - 304} = 0.0014$	$\frac{0.0014 - 0.0014}{305 - 300} = 0$
307	2.4871	$\frac{2.4871 - 2.4843}{307 - 305} = 0.0014$	$\frac{0.0014 - 0.0014}{307 - 304} = 0$

We want to find the value of  $f(x)$  at  $x = 301$ . The Newton's divided difference interpolation formula is given by

$$f(x) = y_0 + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2]$$

Substituting the known values for  $x = 301$ , we get

$$y(301) = 2.4771 + (301 - 300) \times 0.0014 + (301 - 300)(301 - 304) \times 0$$

Calculating each term:

$$y(301) = 2.4771 + (1) \times 0.0014 + (1)(-3) \times 0$$

Thus,

$$y(301) = 2.4771 + 0.0014 + 0 = 2.4785$$

The solution of the divided difference interpolation method at  $y(301)$  is

$$y(301) \approx [2.4785].$$

**Algorithm 27** Newton's divided difference interpolation

**Require:** Set of data points  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$  with distinct  $x_i$

**Require:** Target value  $x$  for interpolation

**Ensure:** Interpolated value  $P(x)$  at target point

```

1: Initialization:
2: Allocate divided difference table  $f[0..n][0..n]$ 
3: for  $i \leftarrow 0..n$  do
4:    $f[i][0] \leftarrow y_i$                                  $\triangleright$  Initialize first column with y-values
5: end for
6: Compute Divided Differences:
7: for  $j \leftarrow 1..n$  do
8:   for  $i \leftarrow 0..n-j$  do
9:      $f[i][j] \leftarrow \frac{f[i+1][j-1] - f[i][j-1]}{x_{i+j} - x_i}$        $\triangleright$  Recursive difference calculation
10:    end for
11: end for
12: Construct Interpolating Polynomial:
13:  $P \leftarrow f[0][0]$                                  $\triangleright$  Start with constant term
14: product  $\leftarrow 1$                                  $\triangleright$  Initialize product term
15: for  $i \leftarrow 1..n$  do
16:   product  $\leftarrow$  product  $\times (x - x_{i-1})$        $\triangleright$  Update product term
17:    $P \leftarrow P + f[0][i] \times$  product            $\triangleright$  Add next polynomial term
18: end for
19: return  $P$                                       $\triangleright$  Return final interpolated value

```

## 2.5 Spline interpolation

One effective approach to address the limitations of Lagrange interpolation is through *spline interpolation*. Unlike traditional polynomial interpolation that uses a single high-degree polynomial to approximate an entire dataset, spline interpolation employs *piecewise polynomial functions* defined over subintervals of the data range. This method is formally known as *piecewise-polynomial interpolation*. The term "spline" originates from the word "splint," referring to a flexible tool used to draw smooth curves by passing through specified points. The simplest form of spline interpolation is *piecewise linear interpolation*, where data points are connected with straight lines. However, this method results in a non-smooth interpolant with sharp corners at the nodes. To achieve smoother results, *cubic spline interpolation* is often used.

For a given set of  $n + 1$  data points  $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$ , the simplest form of spline interpolation is *piecewise linear interpolation*, which connects consecutive points with straight lines, where for each interval  $[x_i, x_{i+1}]$ , the interpolating function  $S_i(x)$  is

$$S_i(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i). \quad (2.7)$$

While this guarantees continuity ( $C^0$  smoothness), it lacks smooth derivatives at the knots

(junction points). More sophisticated splines provide greater smoothness as given in Table 2.4.

Type	Degree	Smoothness
Linear	1	$C^0$
Quadratic	2	$C^1$
Cubic	3	$C^2$

**Table 2.4:** Common spline types and their properties

A *cubic spline* is a piecewise cubic polynomial interpolant that ensures continuity and smoothness across the nodes. Specifically, each interval  $[x_i, x_{i+1}]$  is approximated by a cubic polynomial  $S_i(x)$ . The spline is constructed such that it is twice continuously differentiable, meaning the function, its first derivative, and its second derivative are all continuous across the nodes.

Let's study here the construction of a cubic spline. Consider the problem of finding the cubic spline interpolant  $P(x)$  that is a cubic polynomial  $S_i(x)$  in each interval  $[x_i, x_{i+1}]$  for  $i = 0, 1, \dots, n - 1$

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

where each cubic polynomial  $S_i(x)$  is defined on the interval  $[x_i, x_{i+1}]$ . It can be expressed as

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

where the coefficients  $a_i, b_i, c_i, d_i$  are determined based on the values of  $y_i$  at each data point and continuity conditions for  $P(x)$  and its first and second derivatives at each knot.

For a cubic spline interpolant  $P(x)$  defined on nodes  $a = x_0 < x_1 < \dots < x_n = b$ , the following *conditions* must hold:

- $S_i(x_i) = f(x_i)$  and  $S_i(x_{i+1}) = f(x_{i+1})$  for each  $i$  interpolates the data.
- Continuity condition  $S_{i+1}(x_{i+1}) = S_i(x_{i+1})$  for each  $i$ .
- First derivative continuity  $S'_{i+1}(x_{i+1}) = S'_i(x_{i+1})$ .
- Second derivative continuity  $S''_{i+1}(x_{i+1}) = S''_i(x_{i+1})$ .

To uniquely determine the spline, *additional boundary conditions* are imposed at the end-points  $x_0$  and  $x_n$ . Common choices include:

- Free (natural) boundary conditions  $P''(x_0) = P''(x_n) = 0$ . This results in a "natural" spline that minimizes curvature at the boundaries.
- Clamped boundary conditions  $P'(x_0) = f'(x_0)$  and  $P'(x_n) = f'(x_n)$ . This ensures the spline matches the function's derivatives at the endpoints.

- Not-a-knot boundary conditions, where the third derivative is continuous at  $x_1$  and  $x_{n-1}$ , effectively treating these nodes as non-knots.
- Periodic boundary conditions  $P(x_0) = P(x_n)$ ,  $P'(x_0) = P'(x_n)$ , and  $P''(x_0) = P''(x_n)$ .

### Algorithm for clamped cubic spline

Consider a function  $P(x)$  that interpolates  $f$  such that  $P(x_i) = f(x_i)$ , and satisfies the clamped boundary conditions  $P'(x_0) = f'(x_0)$  and  $P'(x_n) = f'(x_n)$ . For convenience, we denote  $f(x_i) = y_i$  and  $P''(x_i) = z_i$ , where  $z_i$  are the unknown moments. Since  $P''(x)$  is linear in each interval  $[x_i, x_{i+1}]$ , we can express it as

$$P''(x) = S''_i(x) = \frac{z_{i+1}}{h_i}(x - x_i) + \frac{z_i}{h_i}(x_{i+1} - x), \quad x \in [x_i, x_{i+1}]$$

where  $h_i = x_{i+1} - x_i$ . Integrating twice gives the spline function  $S_i(x)$  as

$$P(x) = S_i(x) = \frac{z_{i+1}}{6h_i}(x - x_i)^3 + \frac{z_i}{6h_i}(x_{i+1} - x)^3 + c_i(x - x_i) + d_i(x_{i+1} - x).$$

Using the interpolation conditions  $S_i(x_i) = y_i$  and  $S_i(x_{i+1}) = y_{i+1}$ , we define integration constants  $c_i$ ,  $d_i$  and get the cubic spline

$$S_i(x) = \frac{z_{i+1}}{6h_i}(x - x_i)^3 + \frac{z_i}{6h_i}(x_{i+1} - x)^3 + \left(\frac{y_{i+1}}{h_i} - \frac{h_i}{6}z_{i+1}\right)(x - x_i) + \left(\frac{y_i}{h_i} - \frac{h_i}{6}z_i\right)(x_{i+1} - x).$$

To specify the  $z_i$ , we consider the first derivative continuity at interior nodes

$$S'_{i-1}(x_i) = S'_i(x_i), \quad \text{for } i = 1, 2, \dots, n-1$$

where the derivatives are computed as

$$S'_i(x) = \frac{z_{i+1}}{2h_i}(x - x_i)^2 - \frac{z_i}{2h_i}(x_{i+1} - x)^2 + \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{6}(z_{i+1} - z_i) \quad (2.8)$$

Evaluating at the nodes gives

$$\begin{aligned} S'_i(x_i) &= -\frac{h_i}{6}z_{i+1} - \frac{h_i}{3}z_i + b_i, \\ S'_{i-1}(x_i) &= \frac{h_{i-1}}{6}z_i + \frac{h_{i-1}}{3}z_{i-1} + b_{i-1}, \end{aligned}$$

where

$$b_i = \frac{1}{h_i}(y_{i+1} - y_i).$$

Taking into account the continuity of  $P'(x)$  at every internal node  $x_i$ , we enforce  $S'_i(x_i) = S'_{i-1}(x_i)$  in the spline conditions to derive equations for the unknown moments  $z_i$ . This yields the system of equations

$$\frac{h_{i-1}}{h_{i-1} + h_i}z_{i-1} + 2z_i + \frac{h_i}{h_{i-1} + h_i}z_{i+1} = 6\frac{b_i - b_{i-1}}{h_{i-1} + h_i}, \quad (2.9)$$

for  $i = 1, 2, \dots, n - 1$ . These provide  $n - 1$  equations for the  $n + 1$  unknowns  $\{z_i\}_{i=0}^n$ . To complete the system, we incorporate the boundary conditions  $P'(x_0) = f'(x_0)$  and  $P'(x_n) = f'(x_n)$ . Evaluating the spline derivative conditions at the endpoints gives

$$2z_0 + z_1 = 6 \frac{b_0 - f'(x_0)}{h_0}, \quad (2.10)$$

$$z_{n-1} + 2z_n = 6 \frac{b_{n-1} - f'(x_n)}{h_{n-1}}. \quad (2.11)$$

For notational convenience, we define the following coefficients

$$\begin{aligned} u_i &= \frac{h_i}{h_{i-1} + h_i}, & i &= 1, 2, \dots, n - 1, \\ l_i &= \frac{h_{i-1}}{h_{i-1} + h_i}, & i &= 1, 2, \dots, n - 1, \\ v_i &= 6 \frac{b_i - b_{i-1}}{h_{i-1} + h_i}, & i &= 1, 2, \dots, n - 1, \end{aligned}$$

with boundary terms

$$\begin{aligned} u_0 &= 1, & l_n &= 1, \\ v_0 &= 6 \frac{b_0 - f'(x_0)}{h_0}, & v_n &= 6 \frac{b_{n-1} - f'(x_n)}{h_{n-1}}. \end{aligned}$$

The diagonal elements are constant  $c_i = 2$  for  $i = 0, 1, \dots, n$ . The complete system can then be expressed as a tridiagonal matrix equation  $A\mathbf{z} = \mathbf{v}$

$$\begin{pmatrix} c_0 & u_0 & & & \\ l_1 & c_1 & u_1 & & \\ \ddots & \ddots & \ddots & & \\ & l_{n-1} & c_{n-1} & u_{n-1} & \\ & & l_n & c_n & \end{pmatrix} \begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ z_{n-1} \\ z_n \end{pmatrix} = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \\ v_n \end{pmatrix}.$$

We denote this tridiagonal matrix compactly as  $A = \text{tridiag}(\mathbf{l}, \mathbf{c}, \mathbf{u})$ , where sub-diagonal  $\mathbf{l} = (l_1, \dots, l_n)^T$ , main diagonal  $\mathbf{c} = (c_0, \dots, c_n)^T$ , super-diagonal  $\mathbf{u} = (u_0, \dots, u_{n-1})^T$ .

**Remark:** The boundary conditions only affect the first and last rows of the system. For natural spline boundary conditions ( $P''(x_0) = P''(x_n) = 0$ ), we would set

$$c_0 = c_n = 1, \quad u_0 = l_n = 0, \quad v_0 = v_n = 0.$$

The tridiagonal system can be solved efficiently using the tridiagonal LU decomposition with  $O(n)$  operations, making cubic spline interpolation computationally attractive. The algorithm for the computation of the moments  $z_0, z_1, \dots, z_n$  for cubic splines with clamped boundary conditions is summarized in Algorithm ??.

To evaluate the cubic spline interpolant  $P(x)$  at any point  $x$  using the computed moments  $z_i$ , we employ the piecewise cubic polynomial representation on each subinterval  $[x_i, x_{i+1}]$  as

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, \quad \text{for } x \in [x_i, x_{i+1}]. \quad (2.12)$$

The coefficients  $a_i, b_i, c_i$ , and  $d_i$  are determined as follows:

- By interpolation condition the spline must pass through the data points, so  $S_i(x_i) = y_i$ . This directly gives

$$a_i = y_i. \quad (2.13)$$

- By the first derivative condition the derivative  $S'_i(x_i)$  is derived from the continuity and moment conditions

$$b_i = -\frac{h_i}{6}z_{i+1} - \frac{h_i}{3}z_i + \frac{y_{i+1} - y_i}{h_i}, \quad (2.14)$$

where  $h_i = x_{i+1} - x_i$ .

- By the second derivative condition the second derivative  $S''_i(x_i)$  is given by the moment  $z_i$

$$c_i = \frac{z_i}{2}. \quad (2.15)$$

- By the third derivative condition the coefficient  $d_i$  is determined by enforcing continuity of the second derivative at  $x_{i+1}$

$$d_i = \frac{z_{i+1} - z_i}{6h_i}. \quad (2.16)$$

For computational efficiency, the polynomial  $S_i(x)$  can be rewritten in nested form

$$S_i(x) = y_i + (x - x_i) \left( B_i + (x - x_i) \left( \frac{z_i}{2} + \frac{(x - x_i)(z_{i+1} - z_i)}{6h_i} \right) \right). \quad (2.17)$$

**Algorithm 28** Cubic spline construction with clamped boundary conditions

**Require:** Data points  $(x_0, y_0), \dots, (x_n, y_n)$  with  $x_0 < \dots < x_n$

**Require:** Boundary derivatives  $f'(x_0)$  and  $f'(x_n)$

**Ensure:** Piecewise cubic polynomials  $\{S_i(x)\}_{i=0}^{n-1}$  for each interval  $[x_i, x_{i+1}]$

- 1: **Initialization:**
- 2: **for**  $i \leftarrow 0 \dots n-1$  **do**
- 3:    $h_i \leftarrow x_{i+1} - x_i$  ▷ Interval widths
- 4:    $b_i \leftarrow \frac{y_{i+1}-y_i}{h_i}$  ▷ Slopes between points
- 5: **end for**
- 6: **Construct Tridiagonal System:**
- 7: Initialize vectors  $\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{q}$  of size  $n+1$
- 8:  $\mathbf{u}[0] \leftarrow 2h_0, \mathbf{v}[0] \leftarrow h_0, \mathbf{q}[0] \leftarrow 6(b_0 - f'(x_0))$  ▷ First boundary condition
- 9: **for**  $i \leftarrow 1 \dots n-1$  **do**
- 10:    $\mathbf{u}[i] \leftarrow 2(h_{i-1} + h_i)$  ▷ Main diagonal
- 11:    $\mathbf{v}[i] \leftarrow h_i$  ▷ Upper diagonal
- 12:    $\mathbf{w}[i] \leftarrow h_{i-1}$  ▷ Lower diagonal
- 13:    $\mathbf{q}[i] \leftarrow 6(b_i - b_{i-1})$  ▷ RHS components
- 14: **end for**
- 15:  $\mathbf{u}[n] \leftarrow 2h_{n-1}, \mathbf{w}[n] \leftarrow h_{n-1}, \mathbf{q}[n] \leftarrow 6(f'(x_n) - b_{n-1})$  ▷ Last boundary condition
- 16: **Solve for Moments:**
- 17: Solve  $\mathbf{T}\mathbf{z} = \mathbf{q}$  where  $\mathbf{T}$  is the tridiagonal matrix:

$$\mathbf{T} = \begin{pmatrix} \mathbf{u}[0] & \mathbf{v}[0] & & & \\ \mathbf{w}[1] & \mathbf{u}[1] & \mathbf{v}[1] & & \\ & \ddots & \ddots & \ddots & \\ & & \mathbf{w}[n-1] & \mathbf{u}[n-1] & \mathbf{v}[n-1] \\ & & & \mathbf{w}[n] & \mathbf{u}[n] \end{pmatrix}$$

for moment vector  $\mathbf{z} = (z_0, \dots, z_n)^\top$

- 18: **Construct Spline Segments:**

- 19: **for**  $i \leftarrow 0 \dots n-1$  **do**
- 20:    $S_i(x) \leftarrow \frac{z_{i+1}}{6h_i}(x - x_i)^3 + \frac{z_i}{6h_i}(x_{i+1} - x)^3$  ▷ Cubic terms
- 21:   +  $\left(\frac{y_{i+1}}{h_i} - \frac{h_i z_{i+1}}{6}\right)(x - x_i)$  ▷ Linear terms
- 22:   +  $\left(\frac{y_i}{h_i} - \frac{h_i z_i}{6}\right)(x_{i+1} - x)$
- 23: **end for**
- 24: **return**  $\{S_i(x)\}_{i=0}^{n-1}$  ▷ Return all spline segments

*Example: Cubic spline interpolation*

Find the natural cubic spline that interpolates the points  $(1, 2)$ ,  $(2, 3)$ , and  $(4, 6)$ .

**Step 1. Define the spline segments**

For  $n = 3$  points, we have  $n - 1 = 2$  intervals with corresponding cubic polynomials:

$$\text{Interval } [1, 2] : S_0(x) = a_0 + b_0(x - 1) + c_0(x - 1)^2 + d_0(x - 1)^3$$

$$\text{Interval } [2, 4] : S_1(x) = a_1 + b_1(x - 2) + c_1(x - 2)^2 + d_1(x - 2)^3$$

### Step 2. Apply interpolation conditions

The spline must pass through all data points:

$$S_0(1) = 2 \Rightarrow a_0 = 2 \quad (1)$$

$$S_0(2) = 3 \Rightarrow a_0 + b_0 + c_0 + d_0 = 3 \quad (2)$$

$$S_1(2) = 3 \Rightarrow a_1 = 3 \quad (3)$$

$$S_1(4) = 6 \Rightarrow a_1 + 2b_1 + 4c_1 + 8d_1 = 6 \quad (4)$$

### Step 3. Apply derivative continuity

First and second derivatives must match at  $x = 2$ :

$$S'_0(2) = S'_1(2) \Rightarrow b_0 + 2c_0 + 3d_0 = b_1 \quad (5)$$

$$S''_0(2) = S''_1(2) \Rightarrow 2c_0 + 6d_0 = 2c_1 \quad (6)$$

### Step 4. Apply natural boundary conditions

Second derivatives vanish at endpoints:

$$S''_0(1) = 0 \Rightarrow 2c_0 = 0 \quad (7)$$

$$S''_1(4) = 0 \Rightarrow 2c_1 + 12d_1 = 0 \quad (8)$$

### Step 5. Solve the system

From eqn.(7)  $c_0 = 0$

From eqn.(1)  $a_0 = 2$

From eqn.(3)  $a_1 = 3$

From eqn.(2)  $2 + b_0 + 0 + d_0 = 3 \Rightarrow b_0 + d_0 = 1$

From eqn.(6)  $0 + 6d_0 = 2c_1 \Rightarrow c_1 = 3d_0$

From eqn.(8)  $2(3d_0) + 12d_1 = 0 \Rightarrow d_0 = -2d_1$

From eqn.(5)  $b_0 + 0 + 3d_0 = b_1$

From eqn.(4)  $3 + 2b_1 + 4c_1 + 8d_1 = 6$

Solving these equations yields

$$\begin{aligned} a_0 &= 2, & b_0 &= \frac{11}{12}, & c_0 &= 0, & d_0 &= \frac{1}{12}, \\ a_1 &= 3, & b_1 &= \frac{7}{6}, & c_1 &= \frac{1}{4}, & d_1 &= -\frac{1}{24}, \end{aligned}$$

the final spline function is

$$P(x) = \begin{cases} 2 + \frac{11}{12}(x-1) + \frac{1}{12}(x-1)^3, & x \in [1, 2] \\ 3 + \frac{7}{6}(x-2) + \frac{1}{4}(x-2)^2 - \frac{1}{24}(x-2)^3, & x \in [2, 4] \end{cases}$$

Let's check by substituting the given points into the final spline functions

- At  $x = 1$   $P(1) = 2 + \frac{11}{12}(0) + \frac{1}{12}(0)^3 = 2$  ✓ (matches given point (1,2))
- At  $x = 2$  both pieces yield  $P(2) = 3$  ✓ (matches given point (2,3))
- At  $x = 4$   $P(x) = 3 + \frac{7}{6}(2) + \frac{1}{4}(2)^2 - \frac{1}{24}(2)^3 = 6$  ✓ (matches given point (4,6))

All interpolation conditions are satisfied, confirming the correctness of our cubic spline solution.

### 3 Tasks

1. Construct and analyze the polynomial interpolation of the function  $f(x) = \sin(x)$  on the interval  $[0, 2\pi]$ .
  - (a) Create a uniform partition of the interval using six equally spaced nodes  $x_i$  (for  $i = 1, \dots, 6$ ) and compute the corresponding function values  $y_i = \sin(x_i)$ . Using these data points, determine the unique fifth-degree interpolating polynomial  $P(x)$  that passes through all pairs  $(x_i, y_i)$  using Lagrange interpolation.
  - (b) Evaluate both the original function  $f(x)$  and the interpolating polynomial  $P(x)$  at 101 equidistributed points  $z_j$  across  $[0, 2\pi]$  to assess the approximation quality.
  - (c) Compare the true function value  $f(x)$  with the polynomial approximation  $P(x)$  to examine the behavior of polynomial interpolation for this smooth periodic function.
2. Using Newton's forward interpolation formula, find the polynomial  $f(x)$  that satisfies the data points below. Evaluate  $f(1.5)$ .

$x$	1	2	3	4
$y$	1	3	8	16

3. Using Newton's backward interpolation formula, construct an interpolating polynomial of degree 3 for the data below. Find  $f(1.5)$ .

$x$	-1	0	1	2
$y$	0.5	1.0	2.5	4.0

4. Find the solution of the equation  $x^3 - x + 1$  using Newton's divided difference interpolation formula with nodes  $x_1 = 2$  and  $x_n = 4$ , at  $x = 3.8$ , with step size  $h = 0.5$ .
5. Find the cubic splines for the following table of values and compute  $f(2.5)$ :

$x$	1	2	3	4	5
$y$	32	17	34	20	27

6. Given the data points  $f(0) = 2$ ,  $f(1) = 4$ , and  $f(3) = 7$ :
- (a) Approximate the value  $f(2)$  using an appropriate interpolating polynomial written in Lagrange's form.
  - (b) Approximate the same value  $f(2)$  using Newton's divided difference formula.
  - (c) Construct an interpolating cubic spline and approximate the value  $f(2)$ .
7. Consider the following population data for a major city:

$t_i$	1990	1995	2000	2005	2010	2015
$y_i$	2,450,800	2,710,500	2,890,200	3,150,700	3,420,300	3,810,600

where  $t_i$  represents the census year and  $y_i$  represents the city's population in that year.

- (a) Construct the divided difference table and derive by hand the Newton's interpolating polynomial for this data.
- (b) Implement a numerical method to compute the fifth-degree interpolating polynomial  $P(x)$  that matches all data points  $(t_i, y_i)$ .
- (c) Verify your numerical solution by comparing it with the polynomial obtained in part (a).
- (d) Estimate the city's population for each year between 1990 and 2015 using your interpolating polynomial.
- (e) Repeat the estimation using cubic spline interpolation and compare the results with your polynomial approximation.
- (f) Use your interpolating polynomial to predict the city's population for the years 2018 and 2025.
- (g) Investigate how using integer values for both  $t_i$  (years since 1990) and  $y_i$  (population in millions) affects your results.