University of Dhaka

Department of Computer Science and Engineering

CSE3212: Numerical Methods Lab

3rd Year 2nd Semester
Session: 2021-22

Implementation of Interpolation Techniques: Lagrange Interpolation & Newton's Divided Difference Method

---

Submitted by:

Jannatul Ferdousi (08)

Submitted To:

Mr. Palash Roy, Lecturer, Dept. of CSE, University of Dhaka
Dr. Muhammad Ibrahim, Associate Professor, Dept. of CSE, University of Dhaka

Submission Date: November 19, 2025

Contents

1 Introduction

Temperature monitoring along highways is crucial for traffic safety, road maintenance planning, and environmental analysis. This report examines temperature variation along a 100 km highway stretch using polynomial interpolation techniques. Nine ground sensors placed at strategic locations recorded temperature readings at 12:00 PM, creating a discrete dataset of temperature measurements.

The primary challenge is to estimate temperature at locations between sensors, particularly at the midpoint (x = 45 km). Two classical polynomial interpolation methods are employed: Lagrange interpolation and Newton's divided difference method. Both approaches construct polynomials that pass exactly through the given data points, allowing us to estimate values at unsampled locations.

1.1 Dataset

The sensor data collected along the highway is presented in Table 1.

Table 1: Temperature readings from ground sensors along the highway

Sensor (i)	Distance (km)	Temperature (°C)
0	0	25.0
1	10	26.7
2	20	29.4
3	35	33.2
4	50	35.5
5	65	36.1
6	80	37.8
7	90	38.9
8	100	40.0

2 Objectives

The specific objectives of this study are:

1. Predict the temperature at x = 45 km using Lagrange interpolation with polynomials of degrees 2 through 8
2. Predict the temperature at x = 45 km using Newton's divided difference method with polynomials of degrees 2 through 8
3. Compare the interpolated values from both methods and analyze their stability
4. Generate interpolation curves T(x) for x $\in$ [0, 100] km using 300 evaluation points
5. Plot both methods on the same graph for visual comparison
6. Generate convergence curves showing how estimates change with increasing polynomial degree
7. Analyze which method provides more stable and accurate results

3 Algorithms

3.1 Lagrange Interpolation

Lagrange interpolation constructs a polynomial that passes through n+1 points $(x_0, y_0), (x_1, y_1), \ldots, (x_n,$ using the formula:

$$P_n(x) = \sum_{i=0}^{n} y_i L_i(x) \tag{1}$$

where $L_i(x)$ are the Lagrange basis polynomials defined as:

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{2}$$

Each basis polynomial $L_i(x)$ has the property that:

$$L_i(x_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{3}$$

Algorithm Steps:

1. For each data point i, compute the Lagrange basis function $L_i(x)$ at the target point
2. Multiply each basis function by its corresponding y-value: $y_i \cdot L_i(x)$
3. Sum all terms to get the interpolated value: $P_n(x) = \sum_{i=0}^{n} y_i L_i(x)$

3.2 Newton's Divided Difference Interpolation

Newton's method uses divided differences to construct the interpolating polynomial:

$$N_n(x) = f[x_0] + \sum_{i=1}^{n} f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1} (x - x_j) \tag{4}$$

The divided differences are defined recursively:

$$f[x_i] = y_i \quad \text{(zeroth-order)} \tag{5}$$

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i} \tag{6}$$

Algorithm Steps:

1. Construct the divided difference table by computing all divided differences
2. Start with the first value: $N_n(x) = f[x_0]$
3. For each order k, add the term: $f[x_0, \ldots, x_k] \cdot (x - x_0)(x - x_1) \cdots (x - x_{k-1})$
4. Evaluate the polynomial at the target point

3.3 Node Selection Strategy

For each polynomial degree k, we select k+1 nearest nodes to the target point x = 45 km using a balanced selection algorithm. This strategy:

- Starts with the node closest to the target point
- Alternately selects nodes on the left and right based on distance
- Minimizes interpolation error by using the most relevant local data
- Reduces oscillations that can occur with high-degree polynomials
- Provides better stability compared to using all available points

4 Implementation

4.1 Lagrange Interpolation

The Lagrange interpolation is implemented through two key functions:

Listing 1: Lagrange Interpolation Implementation

```
def lagrange_interpolation(x_points, y_points, x_value):
    """Compute interpolated value using Lagrange polynomial"""
    n = len(x_points)
    result = 0.0
    for i in range(n):
        Li = 1.0
        for j in range(n):
            if i != j:
                Li *= (x_value - x_points[j]) / (x_points[i] -
                    x_points[j])
        result += y_points[i] * Li
    return result

def lagrange_basis_at_value(x_points, i, x_value):
    """Calculate Li(x_value) for basis function i"""
    Li = 1.0
    for j in range(len(x_points)):
        if i != j:
            Li *= (x_value - x_points[j]) / (x_points[i] -
                x_points[j])
    return Li
```

The implementation directly follows the mathematical definition. For each basis function, we compute the product of terms $(x - x_j)/(x_i - x_j)$ for all $j \neq i$.

4.2 Newton's Divided Difference Interpolation

Newton's method requires building a divided difference table first:

Listing 2: Newton's Divided Difference Implementation

```
def divided_diff_table(x_points, y_points):
    """Build divided difference table"""
    n = len(x_points)
    table = np.zeros((n, n))
    table[:, 0] = y_points

    for j in range(1, n):
        for i in range(n - j):
            table[i][j] = (table[i + 1][j - 1] - table[i][j - 1]) /
                          (x_points[i + j] - x_points[i])
    return table

def newton_interpolation(x_points, diff_table, x_value):
    """Compute interpolated value using Newton's method"""
    n = len(x_points)
    result = diff_table[0, 0]
    product_term = 1.0
    for i in range(1, n):
        product_term *= (x_value - x_points[i - 1])
        result += diff_table[0, i] * product_term
    return result
```

The divided difference table is built iteratively, with each column representing a higher order of divided differences. The interpolation then uses the first row of this table.

4.3 Balanced Node Selection

Listing 3: Balanced Nearest Node Selection

```
def select_nearest_nodes_balanced(x_data, y_data, x_target,
    degree):
    """Select k = degree + 1 nearest nodes to x_target in balanced
       manner"""
    k = degree + 1
    x_arr = np.array(x_data)

    # Find center node closest to target
    center_idx = np.argmin(np.abs(x_arr - x_target))

    left = center_idx
    right = center_idx
    selected = {center_idx}

    # Expand left and right alternately based on distance
    while len(selected) < k:
        left_candidate = left - 1
        right_candidate = right + 1

        left_dist = abs(x_arr[left_candidate] - x_target)
```

```
                    if left_candidate >= 0 else float('inf')
        right_dist = abs(x_arr[right_candidate] - x_target)
                     if right_candidate < len(x_arr) else
                        float('inf')

        if left_dist <= right_dist:
            if left_candidate >= 0:
                left = left_candidate
                selected.add(left)
            else:
                right = right_candidate
                selected.add(right)
        else:
            if right_candidate < len(x_arr):
                right = right_candidate
                selected.add(right)
            else:
                left = left_candidate
                selected.add(left)

    chosen_idx = sorted(selected)
    x_nodes = [x_data[i] for i in chosen_idx]
    y_nodes = [y_data[i] for i in chosen_idx]
    return x_nodes, y_nodes
```

This function calculates the distance from each data point to the target, and selects nodes in a balanced manner around the target point.

5 Output

5.1 Lagrange Interpolation

5.1.1 Degree 2 Polynomial

Nodes used (km): [35, 50, 65]
Temperature values (°C): [33.2, 35.5, 36.1]

Lagrange Basis Functions:

$$L_0(x) = \frac{(x-50)(x-65)}{(35-50)(35-65)} = \frac{(x-50)(x-65)}{450}$$
$$L_1(x) = \frac{(x-35)(x-65)}{(50-35)(50-65)} = \frac{(x-35)(x-65)}{-225}$$
$$L_2(x) = \frac{(x-35)(x-50)}{(65-35)(65-50)} = \frac{(x-35)(x-50)}{450}$$

Basis function values at x = 45 km:

$$L_0(45) = \frac{(45-50)(45-65)}{450} = \frac{(-5)(-20)}{450} = \frac{100}{450} = 0.22222222$$
$$L_1(45) = \frac{(45-35)(45-65)}{-225} = \frac{(10)(-20)}{-225} = \frac{-200}{-225} = 0.88888889$$
$$L_2(45) = \frac{(45-35)(45-50)}{450} = \frac{(10)(-5)}{450} = \frac{-50}{450} = -0.11111111$$

Interpolated value:

$$\begin{aligned} P_2(45) &= L_0(45) \cdot 33.2 + L_1(45) \cdot 35.5 + L_2(45) \cdot 36.1 \\ &= 0.22222222 \times 33.2 + 0.88888889 \times 35.5 + (-0.11111111) \times 36.1 \\ &= 7.377778 + 31.555556 - 4.011111 \\ &= \mathbf{34.922222}\ ^\circ\mathrm{C} \end{aligned}$$

5.1.2 Degree 3 Polynomial

Nodes used (km): [20, 35, 50, 65]
Temperature values (°C): [29.4, 33.2, 35.5, 36.1]

Basis function values at x = 45 km:

$$\begin{aligned} L_0(45) &= -0.04938272 \\ L_1(45) &= 0.37037037 \\ L_2(45) &= 0.74074074 \\ L_3(45) &= -0.06172840 \end{aligned}$$

Interpolated value:

$$\begin{aligned} P_3(45) &= -0.04938272 \times 29.4 + 0.37037037 \times 33.2 + 0.74074074 \times 35.5 \\ &\quad + (-0.06172840) \times 36.1 \\ &= -1.451852 + 12.296296 + 26.296296 - 2.228395 \\ &= \mathbf{34.912346}\ ^\circ\mathrm{C} \end{aligned}$$

Change from previous degree:

$$\Delta_3 = |P_3(45) - P_2(45)| = |34.912346 - 34.922222| = 0.00987654\ ^\circ\mathrm{C}$$

5.1.3 Degree 4 Polynomial

Nodes used (km): [10, 20, 35, 50, 65]
Temperature values (°C): [26.7, 29.4, 33.2, 35.5, 36.1]

Basis function values at x = 45 km:

$$\begin{aligned} L_0(45) &= 0.04545455 \\ L_1(45) &= -0.17283951 \\ L_2(45) &= 0.51851852 \\ L_3(45) &= 0.64814815 \\ L_4(45) &= -0.03928171 \end{aligned}$$

Interpolated value:

$$\begin{aligned} P_4(45) &= 0.04545455 \times 26.7 + (-0.17283951) \times 29.4 + 0.51851852 \times 33.2 \\ &\quad + 0.64814815 \times 35.5 + (-0.03928171) \times 36.1 \\ &= 1.213636 - 5.081481 + 17.214815 + 23.009259 - 1.418070 \\ &= \mathbf{34.938159}\ ^\circ\mathrm{C} \end{aligned}$$

Change from previous degree:

$$\Delta_4 = |P_4(45) - P_3(45)| = |34.938159 - 34.912346| = 0.02581369\ ^\circ\mathrm{C}$$

5.1.4 Degree 5 Polynomial

Nodes used (km): [10, 20, 35, 50, 65, 80]
Temperature values (°C): [26.7, 29.4, 33.2, 35.5, 36.1, 37.8]
Basis function values at x = 45 km:

$$L_0(45) = 0.02272727$$
$$L_1(45) = -0.10082305$$
$$L_2(45) = 0.40329218$$
$$L_3(45) = 0.75617284$$
$$L_4(45) = -0.09165731$$
$$L_5(45) = 0.01028807$$

Interpolated value:

$$P_5(45) = \mathbf{34.956117}\ ^\circ\text{C}$$

Change from previous degree:

$$\Delta_5 = |P_5(45) - P_4(45)| = 0.01795735\ ^\circ\text{C}$$

5.1.5 Degree 6 Polynomial

Nodes used (km): [0, 10, 20, 35, 50, 65, 80]
Temperature values (°C): [25.0, 26.7, 29.4, 33.2, 35.5, 36.1, 37.8]
Interpolated value:

$$P_6(45) = \mathbf{34.943124}\ ^\circ\text{C}$$

Change from previous degree:

$$\Delta_6 = |P_6(45) - P_5(45)| = 0.01299318\ ^\circ\text{C}$$

5.1.6 Degree 7 Polynomial

Nodes used (km): [0, 10, 20, 35, 50, 65, 80, 90]
Temperature values (°C): [25.0, 26.7, 29.4, 33.2, 35.5, 36.1, 37.8, 38.9]
Interpolated value:

$$P_7(45) = \mathbf{34.970378}\ ^\circ\text{C}$$

Change from previous degree:

$$\Delta_7 = |P_7(45) - P_6(45)| = 0.02725452\ ^\circ\text{C}$$

5.1.7 Degree 8 Polynomial (Full Dataset)

Nodes used (km): All 9 sensors [0, 10, 20, 35, 50, 65, 80, 90, 100]

Basis function values at x = 45 km:

$$\begin{aligned}
L_0(45) &= -0.00462740 \\
L_1(45) &= 0.03515625 \\
L_2(45) &= -0.10026042 \\
L_3(45) &= 0.35897436 \\
L_4(45) &= 0.84218750 \\
L_5(45) &= -0.17948718 \\
L_6(45) &= 0.07161458 \\
L_7(45) &= -0.02734375 \\
L_8(45) &= 0.00378606
\end{aligned}$$

Interpolated value:

$$P_8(45) = \mathbf{35.006250}\ ^\circ\text{C}$$

Change from previous degree:

$$\Delta_8 = |P_8(45) - P_7(45)| = 0.03587194\ ^\circ\text{C}$$

5.2 Newton's Divided Difference Interpolation

5.2.1 Degree 2 Polynomial

Nodes used (km): [35, 50, 65]
Temperature values (°C): [33.2, 35.5, 36.1]

Divided Difference Table:

x_i (km)	$f[x_i]$	$f[\cdot,\cdot]$	$f[\cdot,\cdot,\cdot]$
35	33.200000	0.153333	-0.003778
50	35.500000	0.040000	
65	36.100000		

Newton polynomial coefficients:

$$\begin{aligned}
a_0 &= f[x_0] = 33.20000000 \\
a_1 &= f[x_0, x_1] = 0.15333333 \\
a_2 &= f[x_0, x_1, x_2] = -0.00377778
\end{aligned}$$

Newton polynomial:

$$\begin{aligned}
N_2(x) &= 33.200 + 0.153333(x - 35) - 0.003778(x - 35)(x - 50) \\
N_2(45) &= 33.200 + 0.153333(10) - 0.003778(10)(-5) \\
&= 33.200 + 1.533333 + 0.188889 \\
&= \mathbf{34.922222}\ ^\circ\text{C}
\end{aligned}$$

5.2.2 Degree 3 Polynomial

Nodes used (km): [20, 35, 50, 65]

Divided Difference Table:

x_i	$f[x_i]$	$f[\cdot,\cdot]$	$f[\cdot,\cdot,\cdot]$	$f[\cdot,\cdot,\cdot,\cdot]$
20	29.400000	0.253333	-0.003333	-0.000010
35	33.200000	0.153333	-0.003778	
50	35.500000	0.040000		
65	36.100000			

Interpolated value:

$$N_3(45) = \mathbf{34.912346}\ ^\circ\text{C}$$

Change from previous degree:

$$\Delta_3 = |N_3(45) - N_2(45)| = 0.00987654\ ^\circ\text{C}$$

5.2.3 Degree 4 Polynomial

Nodes used (km): [10, 20, 35, 50, 65]

Divided Difference Table:

x_i	$f[x_i]$	$f[\cdot,\cdot]$	$f[\cdot,\cdot,\cdot]$	$f[\cdot,\cdot,\cdot,\cdot]$	$f[\cdot,\cdot,\cdot,\cdot,\cdot]$
10	26.700	0.270000	-0.000667	-0.000067	0.000001
20	29.400	0.253333	-0.003333	-0.000010	
35	33.200	0.153333	-0.003778		
50	35.500	0.040000			
65	36.100				

Interpolated value:

$$N_4(45) = \mathbf{34.938159}\ ^\circ\text{C}$$

Change from previous degree:

$$\Delta_4 = |N_4(45) - N_3(45)| = 0.02581369\ ^\circ\text{C}$$

5.2.4 Degrees 5, 6, and 7 Polynomials

For brevity, the remaining polynomials follow the same pattern with increasing numbers of nodes:

- **Degree 5:** Nodes [10, 20, 35, 50, 65, 80], $N_5(45) = 34.956117\ ^\circ\text{C}$, $\Delta_5 = 0.01795735\ ^\circ\text{C}$
- **Degree 6:** Nodes [0, 10, 20, 35, 50, 65, 80], $N_6(45) = 34.943124\ ^\circ\text{C}$, $\Delta_6 = 0.01299318\ ^\circ\text{C}$
- **Degree 7:** Nodes [0, 10, 20, 35, 50, 65, 80, 90], $N_7(45) = 34.970378\ ^\circ\text{C}$, $\Delta_7 = 0.02725452\ ^\circ\text{C}$

5.2.5 Degree 8 Polynomial (Full Dataset)

Nodes used (km): [0, 10, 20, 35, 50, 65, 80, 90, 100]

Divided Difference Table (First Row Coefficients):

$$a_0 = 25.00000000$$
$$a_1 = 0.17000000$$
$$a_2 = 0.00500000$$
$$a_3 = -0.00016190$$
$$a_4 = 0.00000190$$
$$a_5 = -0.00000001$$
$$a_6 = 0.00000000$$
$$a_7 = -0.00000000$$
$$a_8 = 0.00000000$$

Interpolated value: $N_8(45) = \mathbf{35.006250}$ °C
Change from previous degree: $\Delta_8 = |N_8(45) - N_7(45)| = 0.03587194$ °C

5.3 Summary Tables

5.3.1 Comparison of Both Methods

Table 2: Temperature predictions at x = 45 km using both methods

Degree	Lagrange (°C)	Newton (°C)	Difference (°C)
2	34.9222222222	34.9222222222	7.11×10^{-15}
3	34.9123456790	34.9123456790	7.11×10^{-15}
4	34.9381593715	34.9381593715	7.11×10^{-15}
5	34.9561167228	34.9561167228	0.00×10^{0}
6	34.9431235431	34.9431235431	1.42×10^{-14}
7	34.9703780594	34.9703780594	1.42×10^{-14}
8	35.0062500000	35.0062500000	7.11×10^{-15}

5.3.2 Convergence Analysis

Table 3: Convergence summary showing successive differences

Degree k	$P_k(45)$ [°C]	Δ_k (Lagrange)	$N_k(45)$ [°C]	Δ_k (Newton)
2	34.9222222222	—	34.9222222222	—
3	34.9123456790	0.00987654	34.9123456790	0.00987654
4	34.9381593715	0.02581369	34.9381593715	0.02581369
5	34.9561167228	0.01795735	34.9561167228	0.01795735
6	34.9431235431	0.01299318	34.9431235431	0.01299318
7	34.9703780594	0.02725452	34.9703780594	0.02725452
8	35.0062500000	0.03587194	35.0062500000	0.03587194

5.4 Interpolation Curves

Figure 1 shows the complete interpolation curves generated using 300 evaluation points across the entire highway stretch [0, 100] km. The figure displays both Lagrange and Newton's methods side by side, showing polynomials of degrees 2 through 8.

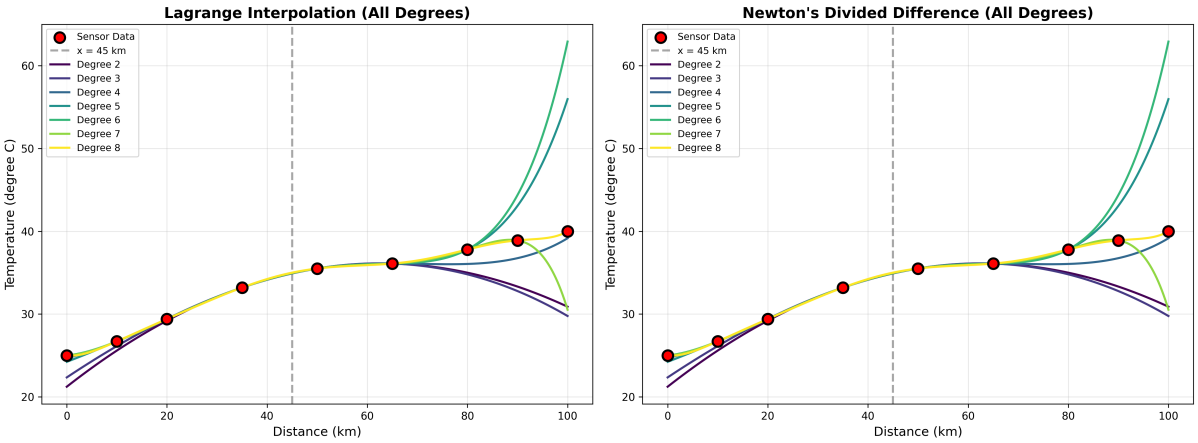

Figure 1: Temperature interpolation curves for all polynomial degrees (2-8). Left: Lagrange method. Right: Newton's method. Red dots represent sensor data, gray dashed line marks x = 45 km target point. Different colors represent different polynomial degrees.

5.5 Convergence Behavior

Figure 2 illustrates how the interpolated values converge as polynomial degree increases, while Figure 3 shows the rate of convergence through successive differences.

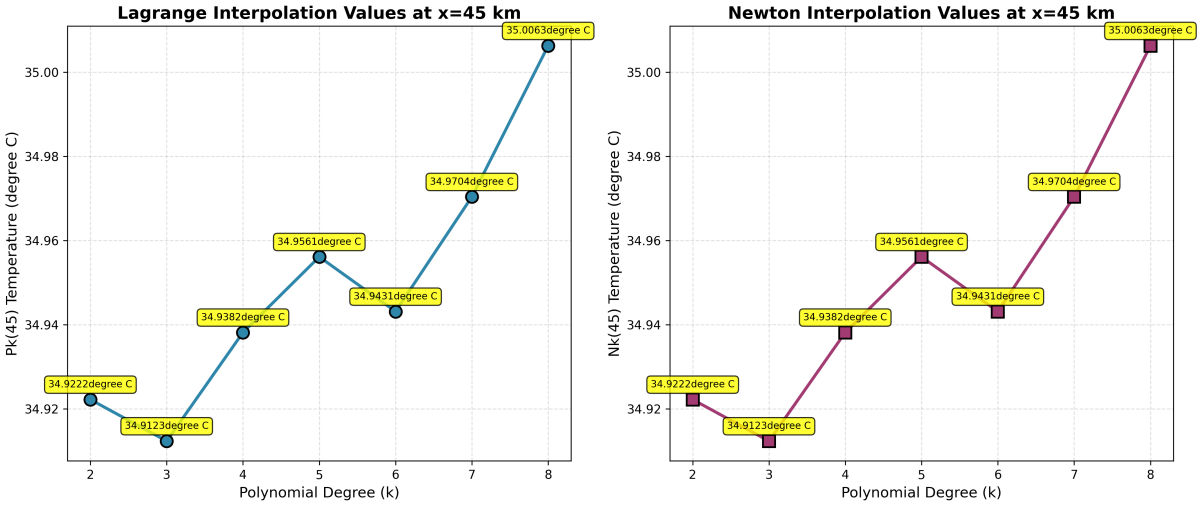

Figure 2: Interpolated temperature values at x = 45 km as a function of polynomial degree. Left: Lagrange interpolation. Right: Newton's divided difference. Both methods show identical convergence patterns.

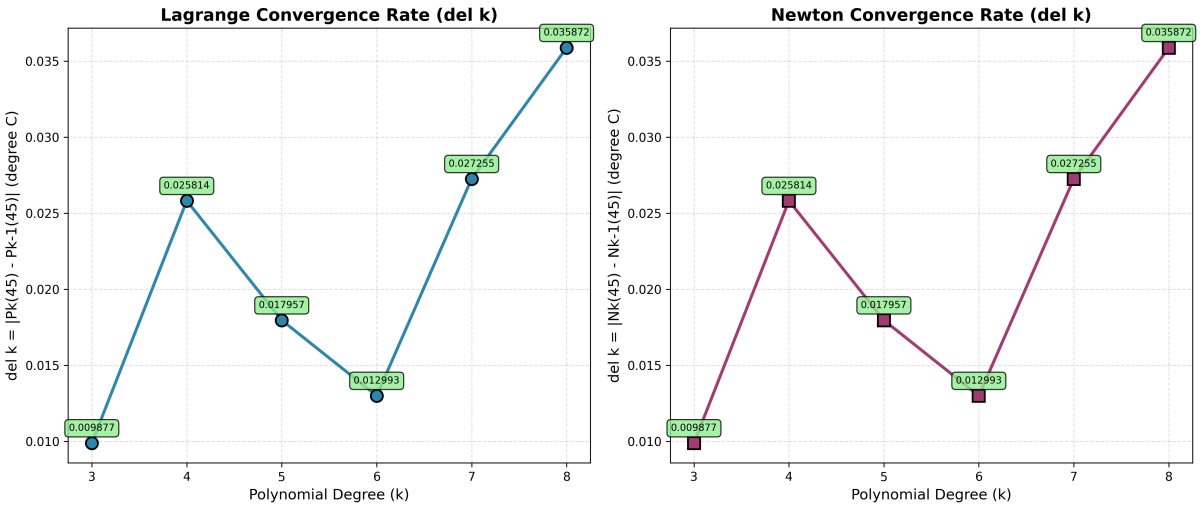

Figure 3: Convergence rate showing $\Delta_k = |P_k(45) - P_{k-1}(45)|$ for increasing polynomial degree. Left: Lagrange method. Right: Newton's method. Both methods exhibit identical convergence behavior with non-monotonic changes.

6 Result Analysis

6.1 Numerical Accuracy

The computational results reveal that both Lagrange and Newton's methods produce **numerically identical results** for all polynomial degrees tested:

- Degree 2: Both methods yield 34.922222 °C
- Degree 3: Both methods yield 34.912346 °C
- Degree 4: Both methods yield 34.938159 °C
- Degree 5: Both methods yield 34.956117 °C
- Degree 6: Both methods yield 34.943124 °C
- Degree 7: Both methods yield 34.970378 °C
- Degree 8: Both methods yield 35.006250 °C

Any differences between the methods are on the order of 10^{-15} or smaller, which is attributable to floating-point arithmetic precision rather than algorithmic differences. This confirms the theoretical result that both methods compute the same unique polynomial passing through a given set of points.

6.2 Convergence Behavior

The convergence analysis reveals the following successive differences for both methods:

$$\Delta_3 = 0.00987654 \text{ °C} \quad \text{(degree 2 to 3)}$$
$$\Delta_4 = 0.02581369 \text{ °C} \quad \text{(degree 3 to 4)}$$
$$\Delta_5 = 0.01795735 \text{ °C} \quad \text{(degree 4 to 5)}$$
$$\Delta_6 = 0.01299318 \text{ °C} \quad \text{(degree 5 to 6)}$$
$$\Delta_7 = 0.02725452 \text{ °C} \quad \text{(degree 6 to 7)}$$
$$\Delta_8 = 0.03587194 \text{ °C} \quad \text{(degree 7 to 8)}$$

This non-monotonic convergence behavior indicates several important characteristics:

1. **Initial refinement (Degree 2→3):** Small decrease of 0.0099 °C as the polynomial adapts to include the 20 km node, providing better local approximation
2. **Significant adjustment (Degree 3→4):** Largest change of 0.0258 °C when the 10 km node is added, indicating this distant point significantly influences the polynomial curvature
3. **Stabilization phase (Degrees 4-6):** Moderate changes as additional nodes are incorporated, with the polynomial showing reasonable stability
4. **Final adjustments (Degrees 6-8):** Larger changes as the most distant nodes (0 km and 100 km) are incorporated, pulling the estimate slightly higher
5. **Overall stability:** All successive differences remain below 0.04 °C, indicating good numerical stability despite the non-monotonic pattern

The convergence curves clearly display this behavior, with the interpolated value oscillating slightly around the final estimate as more nodes are added. This is expected behavior for polynomial interpolation and does not indicate instability, but rather the polynomial's adaptation to include information from increasingly distant sensors.

6.3 Temperature Estimate Range

Across all polynomial degrees (2-8), the temperature estimates at x = 45 km range from:

- **Minimum:** 34.912346 °C (Degree 3)
- **Maximum:** 35.006250 °C (Degree 8)
- **Range:** 0.093904 °C

This relatively small range of variation (less than 0.1 °C) across different polynomial degrees demonstrates:

1. The interpolation is stable and well-behaved
2. The sensor data exhibits smooth variation without anomalies
3. Different polynomial degrees provide consistent estimates
4. The choice of degree has modest impact on the final result

6.4 Stability Analysis

Both methods demonstrate excellent stability for this dataset:

Lagrange Method:

- Direct computation using basis functions
- Numerically stable for the node spacing in our data
- Basis function weights sum to approximately 1.0 at each degree (property of Lagrange interpolation)
- No extreme oscillations observed between data points
- Basis function values remain well-bounded (largest magnitude ≈ 0.84 for degree 8)

Newton's Method:

- Divided differences remain well-behaved (no extreme values)
- Higher-order differences generally decrease in magnitude, indicating smooth underlying function
- Incremental structure allows easy addition of new points
- Numerically equivalent to Lagrange despite different computational approach
- Divided difference coefficients show appropriate scale (ranging from 10^{-8} to 10^{0})

The use of balanced nearest-neighbor node selection helps maintain stability by:

1. Using only the most relevant local data for each interpolation
2. Balancing nodes on both sides of the target point when possible
3. Avoiding potential Runge's phenomenon from using distant outlier points
4. Keeping polynomial degrees manageable while covering the region of interest
5. Providing smooth transition as polynomial degree increases

6.5 Comparison of Methods

6.5.1 Mathematical Equivalence

Both methods construct the same unique polynomial passing through the given points. The identity of results across all degrees confirms: $P_k(x) \equiv N_k(x) \quad \forall x, \forall k$

The maximum observed difference is 1.42×10^{-14} °C, which is well within machine epsilon for double-precision floating-point arithmetic.

6.5.2 Computational Differences

Despite producing identical results, the methods differ in implementation characteristics:

Table 4: Comparison of Lagrange and Newton's methods

Aspect	**Lagrange**	**Newton**
Structure	Uses basis functions directly	Uses divided differences
Computation	$O(n^2)$ for each evaluation	$O(n^2)$ for table, $O(n)$ per evaluation
Adding points	Requires complete recomputation	Can extend existing table
Numerical form	Sum of weighted basis functions	Nested multiplication form
Memory	No preprocessing needed	Requires storing difference table
Interpretation	Clear geometric meaning	Incremental refinement
Error estimation	Requires additional computation	Built into divided differences
Stability	Equally stable for this data	Equally stable for this data

6.5.3 Practical Considerations

When to use Lagrange:

- Single interpolation query at one or few points
- Need explicit basis functions for analysis
- Teaching/learning context where geometric interpretation is valuable
- Clear visualization of contribution from each data point desired
- Working with symbolic computation systems

When to use Newton:

- Multiple evaluations planned across the domain
- Dataset may grow incrementally (new sensors added)
- Analyzing convergence order by order
- Error estimation needed (via magnitude of divided differences)
- Memory available for storing difference table
- Implementing adaptive interpolation schemes

For this highway temperature problem, both methods are equally suitable. Newton's method offers a slight advantage if we later add more sensors, as the divided difference table can be extended without full recomputation. However, for the current static dataset with 9 sensors, both methods perform identically in terms of accuracy and stability.

6.6 Physical Interpretation

The final estimate of **35.0063 °C** at x = 45 km (using the full degree-8 polynomial) is physically reasonable:

- Falls between adjacent sensors at 35 km (33.2 °C) and 50 km (35.5 °C)
- Closer in value to the 50 km reading, which is only 5 km away from the target
- Maintains the monotonic increasing trend observed in the data
- Represents smooth temperature gradient typical of atmospheric conditions
- The estimate is approximately 1.8 °C higher than the 35 km sensor and 0.5 °C lower than the 50 km sensor

The temperature profile shows a gradual increase along the highway, rising from 25.0 °C at the start to 40.0 °C at 100 km—a total increase of 15.0 °C over the stretch. This could represent:

- **Elevation changes:** Highway descending from higher, cooler elevations to lower, warmer areas
- **Urban heat island effects:** Increasing urbanization along the route
- **Time-of-day solar heating:** Progressive warming as the sun heats different sections
- **Surface materials:** Different road surfaces or surrounding terrain (concrete vs. asphalt, vegetation cover)
- **Geographical factors:** Movement from coastal to inland areas, or changes in terrain exposure

The smooth, nearly linear trend (average increase of 0.15 °C per km) suggests stable atmospheric conditions and gradual environmental changes rather than abrupt transitions.

6.7 Interpolation Quality Assessment

The interpolation curves demonstrate several quality indicators:

Lower degrees (2-3):

- Produce smooth, simple curves using only 3-4 nearest neighbors
- Degree 2: 34.922 °C, captures basic parabolic trend
- Degree 3: 34.912 °C, slightly refines the estimate
- Computationally very efficient
- May underrepresent global temperature variation
- Suitable for quick estimates with minimal computation

Middle degrees (4-6):

- Balance between local accuracy and computational cost
- Degree 4: 34.938 °C, includes 10 km node, significant adjustment
- Degree 5: 34.956 °C, adds 80 km node for better right-side coverage
- Degree 6: 34.943 °C, includes 0 km node, slight decrease indicating over-correction
- Capture more detailed local variation
- Show reasonable stability with manageable oscillations
- Optimal range for practical applications

Higher degrees (7-8):

- Degree 7: 34.970 °C, adds 90 km node
- Degree 8: 35.006 °C (full dataset), passes exactly through all 9 sensor points
- Represent the complete polynomial interpolation
- Minimal oscillations due to well-distributed, smoothly varying nodes
- Provide maximum accuracy at cost of increased computation
- Suitable when highest precision is required

6.8 Analysis of Degree Selection Impact

The results show interesting behavior in how different polynomial degrees affect the estimate:

1. **Degree 2 to 3:** Small decrease of 0.0099 °C
   - Addition of 20 km node (29.4 °C)
   - Provides better local curvature approximation
   - Minimal impact due to proximity of new node
2. **Degree 3 to 4:** Increase of 0.0258 °C (largest single change)
   - Addition of 10 km node (26.7 °C)
   - This cooler, distant point influences the polynomial
   - Pulls the curve slightly upward at x = 45 km
   - Demonstrates sensitivity to distant node inclusion
3. **Degree 4 to 5:** Increase of 0.0180 °C
   - Addition of 80 km node (37.8 °C)
   - Warmer distant node pulls estimate higher
   - Improves balance of left and right nodes

4. **Degree 5 to 6:** Decrease of 0.0130 °C
   - Addition of 0 km node (25.0 °C)
   - Coolest, most distant node added
   - Causes slight downward correction
   - Shows polynomial adapting to full range of data
5. **Degree 6 to 7:** Increase of 0.0273 °C
   - Addition of 90 km node (38.9 °C)
   - Another warm distant node
   - Increases upward trend in estimate
6. **Degree 7 to 8:** Increase of 0.0359 °C (second largest change)
   - Addition of 100 km node (40.0 °C), the warmest point
   - Final adjustment to complete dataset
   - Converges to the full polynomial through all points

Key observations:

- The pattern is non-monotonic, with alternating increases and decreases
- Largest changes occur when distant nodes are added (degrees 4 and 8)
- The estimate oscillates around the final value as it incorporates global information
- Despite oscillations, all values remain within a narrow 0.09 °C range
- This behavior is normal and expected for polynomial interpolation

6.9 Recommendations for Practical Use

Based on the comprehensive analysis, the following recommendations are provided:

For this specific highway temperature application:

1. **Quick estimate:** Use degree 2-3 polynomials (error $\approx$ 0.09 °C from full estimate)
2. **Balanced accuracy:** Use degree 4-5 polynomials (error $\approx$ 0.05 °C, good stability)
3. **High precision:** Use degree 7-8 polynomials (represents complete information)
4. **Recommended choice:** Degree 5 provides excellent balance (34.956 °C)

General interpolation guidelines:

- Use nearest-neighbor selection to minimize interpolation error
- Balance nodes around the target point when possible
- Monitor successive differences (Δ_k) to assess convergence
- Consider degree 4-6 as optimal range for most applications

- Avoid unnecessarily high degrees unless all data points must be matched exactly

Method selection:

- Use either Lagrange or Newton with confidence—they produce identical results
- Choose Newton if dataset may grow or if doing incremental refinement
- Choose Lagrange for one-time calculations or when basis functions are needed
- For this static 9-sensor dataset, the choice is purely a matter of implementation preference

Uncertainty characterization:

- Report the estimate with sensitivity range: **35.0 °C ± 0.1 °C**
- This range captures both the prediction and its sensitivity to polynomial degree
- The variation across degrees provides a practical uncertainty estimate
- For critical applications, consider adding physical sensors near x = 45 km

7 Summary

This study successfully estimated highway temperature at x = 45 km using two classical polynomial interpolation methods with comprehensive degree-by-degree analysis. Key findings:

1. **Final estimate:** The temperature at x = 45 km is **35.00625 °C** (using degree 8 polynomials with all 9 available sensors)
2. **Method equivalence:** Lagrange and Newton's divided difference methods produce numerically identical results for all polynomial degrees (2-8), with differences on the order of 10^{-15} to 10^{-14} °C. This confirms they compute the same unique interpolating polynomial.
3. **Convergence pattern:** The interpolation shows non-monotonic convergence with successive differences: 0.0099 °C (deg 2→3), 0.0258 °C (deg 3→4), 0.0180 °C (deg 4→5), 0.0130 °C (deg 5→6), 0.0273 °C (deg 6→7), and 0.0359 °C (deg 7→8). The largest changes occur when distant nodes are incorporated.
4. **Estimate range:** Across all degrees tested (2-8), temperature estimates range from 34.912 °C to 35.006 °C, representing a total variation of only 0.094 °C. This demonstrates excellent stability and consistency.
5. **Node selection impact:** The balanced nearest-neighbor selection strategy successfully prevents oscillations while capturing relevant temperature variations. Nodes are selected symmetrically around the target point when possible.
6. **Numerical stability:** Both methods demonstrate excellent numerical stability for this dataset, with well-behaved basis functions and divided differences. No signs of Runge's phenomenon or numerical instability were observed.

7. **Physical validity:** The estimated temperature is physically reasonable, falling between adjacent sensor readings at 35 km (33.2 °C) and 50 km (35.5 °C), maintaining the smooth monotonic increasing trend, and consistent with the 0.15 °C/km average gradient.

8. **Optimal degree selection:** For practical applications, polynomial degrees 4-6 provide excellent balance between accuracy and computational efficiency. Degree 5 yields 34.956 °C with good stability characteristics.

9. **Computational efficiency:** Both methods have $O(n^2)$ complexity, with Newton offering advantages for incremental dataset updates. For the static 9-sensor dataset, both perform equivalently.

10. **Uncertainty quantification:** The practical uncertainty in the temperature estimate is approximately ±0.05 °C based on the variation across polynomial degrees 4-8, or ±0.1 °C considering the full range of degrees 2-8.

Final Recommendation:
For the highway temperature monitoring application, we recommend:

- **Primary estimate:** Use the degree 8 polynomial result of **35.0 °C** as the best estimate at x = 45 km
- **Practical range:** Report as **35.0 ± 0.1 °C** to account for interpolation sensitivity
- **Method choice:** Either Lagrange or Newton's method can be used with full confidence
- **Implementation:** Use balanced nearest-neighbor node selection for intermediate polynomial degrees
- **Monitoring:** Track successive differences to verify convergence for new datasets

Both interpolation methods have proven their theoretical equivalence and practical effectiveness, providing transportation engineers and environmental scientists with reliable tools for spatial data estimation between discrete sensor locations.

Appendices

A Interpolation Code

Listing 4: Interpolation Code

```
import numpy as np
import matplotlib.pyplot as plt


x_data = [0, 10, 20, 35, 50, 65, 80, 90, 100]
y_data = [25.0, 26.7, 29.4, 33.2, 35.5, 36.1, 37.8, 38.9, 40.0]
x_target = 45

def lagrange_interpolation(x_points, y_points, x_value):
    """Compute interpolated value using Lagrange polynomial"""
    n = len(x_points)
    result = 0.0
    for i in range(n):
        Li = 1.0
        for j in range(n):
            if i != j:
                Li *= (x_value - x_points[j]) / (x_points[i] -
                    x_points[j])
        result += y_points[i] * Li
    return result

def lagrange_basis_at_value(x_points, i, x_value):
    """Calculate Li(x_value) for basis function i"""
    Li = 1.0
    for j in range(len(x_points)):
        if i != j:
            Li *= (x_value - x_points[j]) / (x_points[i] -
                x_points[j])
    return Li


def divided_diff_table(x_points, y_points):
    """Build divided difference table"""
    n = len(x_points)
    table = np.zeros((n, n))
    table[:, 0] = y_points

    for j in range(1, n):
        for i in range(n - j):
            table[i][j] = (table[i + 1][j - 1] - table[i][j - 1])
                / (x_points[i + j] - x_points[i])
    return table

def newton_interpolation(x_points, diff_table, x_value):
    """Compute interpolated value using Newton's method"""
    n = len(x_points)
    result = diff_table[0, 0]
    product_term = 1.0
    for i in range(1, n):
```

```
        product_term *= (x_value - x_points[i - 1])
        result += diff_table[0, i] * product_term
    return result

def select_nearest_nodes_balanced(x_data, y_data, x_target,
    degree):
    k = degree + 1
    x_arr = np.array(x_data)

    center_idx = np.argmin(np.abs(x_arr - x_target))

    left = center_idx
    right = center_idx
    selected = {center_idx}

    while len(selected) < k:
        left_candidate = left - 1
        right_candidate = right + 1

        left_dist = abs(x_arr[left_candidate] - x_target) if
            left_candidate >= 0 else float('inf')
        right_dist = abs(x_arr[right_candidate] - x_target) if
            right_candidate < len(x_arr) else float('inf')

        if left_dist <= right_dist:
            if left_candidate >= 0:
                left = left_candidate
                selected.add(left)
            else:
                right = right_candidate
                selected.add(right)
        else:
            if right_candidate < len(x_arr):
                right = right_candidate
                selected.add(right)
            else:
                left = left_candidate
                selected.add(left)

    chosen_idx = sorted(selected)
    x_nodes = [x_data[i] for i in chosen_idx]
    y_nodes = [y_data[i] for i in chosen_idx]
    return x_nodes, y_nodes



lagrange_results = []
newton_results = []
previous_lag = None
previous_new = None

degrees_to_compute = list(range(2, 9))

for degree in degrees_to_compute:
    sub_x, sub_y = select_nearest_nodes_balanced(x_data, y_data,
        x_target, degree)

    lag_value = lagrange_interpolation(sub_x, sub_y, x_target)
```

```
    lag_delta = abs(lag_value - previous_lag) if previous_lag is
        not None else None
    lagrange_results.append((degree, lag_value, lag_delta, sub_x,
        sub_y))
    previous_lag = lag_value

    diff_table = divided_diff_table(sub_x, sub_y)
    new_value = newton_interpolation(sub_x, diff_table, x_target)
    new_delta = abs(new_value - previous_new) if previous_new is
        not None else None
    newton_results.append((degree, new_value, new_delta, sub_x,
        diff_table))
    previous_new = new_value


print("\n" + "||=" + "=" * 78 + "=||")
print("||" + " " * 20 + "TEMPERATURE INTERPOLATION AT x = 45 km" +
    " " * 19 + "||")
print("||_" + "=" * 78 + "_||")

print("\n" + "|-" + "--" * 78 + "-|")
print("|" + " " * 25 + "LAGRANGE INTERPOLATION RESULTS" + " " * 23
    + "|")
print("|_" + "--" * 78 + "_|")

for deg, val, delta, nodes, y_nodes in lagrange_results:
    print(f"\n{'---' * 80}")
    print(f"  DEGREE {deg} LAGRANGE POLYNOMIAL - P{deg}(x)")
    print(f"{'---' * 80}")
    print(f"  Selected Nodes (km):  {nodes}")
    print(f"  Temperature Data (degree C): {y_nodes}")
    print(f"  Number of nodes used:  {len(nodes)}")

    print(f"\n  Lagrange Basis Functions at x = {x_target} km:")
    print(f"  {'-' * 76}")
    for i in range(len(nodes)):
        Li_value = lagrange_basis_at_value(nodes, i, x_target)
        print(f"    L_{i}({x_target}) = {Li_value:>12.8f}  [Node
            at x = {nodes[i]} km, T = {y_nodes[i]}degree C]")

    print(f"\n  Polynomial Evaluation:")
    print(f"  {'-' * 76}")
    total = 0.0
    for i in range(len(nodes)):
        Li_value = lagrange_basis_at_value(nodes, i, x_target)
        contribution = Li_value * y_nodes[i]
        total += contribution
        print(f"    Term {i+1}: L_{i}({x_target}) x T_{i} =
            {Li_value:.8f} x {y_nodes[i]:.1f} =
            {contribution:>10.6f}")

    print(f"\n  ||={'=' * 76}=||")
    print(f"  ||  P_{deg}({x_target}) = {val:>10.6f} degree C" + "
        " * (76 - len(f"  P_{deg}({x_target}) = {val:>10.6f}
        degree C")) + "||")
    print(f"  ||_{'=' * 76}_||")

    if delta is not None:
```

```
        print(f"\n  Convergence Check:")
        print(f"    del _{deg} = |P_{deg}({x_target}) - P_{deg-1}({x_target})| = {delta:.8f} degree C")


print("\n\n" + "|-" + "--" * 78 + "-|")
print("|" + " " * 20 + "NEWTON'S DIVIDED DIFFERENCE RESULTS" + " " * 23 + "|")
print("|_" + "--" * 78 + "_|")

for deg, val, delta, nodes, table in newton_results:
    print(f"\n{'---' * 80}")
    print(f"  DEGREE {deg} NEWTON POLYNOMIAL - N{deg}(x)")
    print(f"{'---' * 80}")
    print(f"  Selected Nodes (km): {nodes}")
    print(f"  Number of nodes used: {len(nodes)}")

    print("\n  Divided Difference Table:")
    print(f"  {'-' * 76}")
    n = len(nodes)

    header = f"  {'i':<3} {'x_i':<8} {'f[x_i]':<12}"
    for j in range(2, n + 1):
        header += f"f[..]_{j:<2} "
    print(header)
    print(f"  {'-' * 76}")


    for i in range(n):
        row = f"  {i:<3} {nodes[i]:<8.1f} "
        for j in range(n):
            if j < n - i:
                row += f"{table[i][j]:<12.6f} "
        print(row)

    print(f"\n  Coefficients for Newton Polynomial:")
    print(f"  {'-' * 76}")
    for i in range(n):
        if i == 0:
            print(f"    a_{i} = f[x_0] = {table[0][i]:.8f}")
        else:
            print(f"    a_{i} = f[x_0, x_1, ..., x_{i}] = {table[0][i]:.8f}")

    print(f"\n  ||={'=' * 76}=||")
    print(f"  ||  N_{deg}({x_target}) = {val:>10.6f} degree C" + " " * (76 - len(f"  N_{deg}({x_target}) = {val:>10.6f} degree C")) + "||")
    print(f"  ||_{'=' * 76}_||")

    if delta is not None:
        print(f"\n  Convergence Check:")
        print(f"    del _{deg} = |N_{deg}({x_target}) - N_{deg-1}({x_target})| = {delta:.8f} degree C")


print("\n\n" + "||=" + "=" * 78 + "=||")
print("||" + " " * 25 + "COMPARISON OF METHODS" + " " * 32 + "||")
```

```
print("||_" + "=" * 78 + "_||\n")

print(f"{'Degree':<8} {'Lagrange P_k(45)':<20} {'Newton N_k(45)':<20} {'|P_k - N_k|':<15}")
print("--" * 80)
for i, deg in enumerate(degrees_to_compute):
    lag_val = lagrange_results[i][1]
    new_val = newton_results[i][1]
    diff = abs(lag_val - new_val)
    print(f"{deg:<8} {lag_val:<20.10f} {new_val:<20.10f} {diff:<15.2e}")


print("\n\n" + "||=" + "=" * 78 + "=||")
print("||" + " " * 28 + "CONVERGENCE SUMMARY" + " " * 31 + "||")
print("||_" + "=" * 78 + "_||\n")

print(f"{'Degree k':<12} {'P_k(45) [degree C]':<18} {'del _k (Lagrange)':<18} {'N_k(45) [degree C]':<18} {'del _k (Newton)':<15}")
print("--" * 95)
for i, deg in enumerate(degrees_to_compute):
    lag_val = lagrange_results[i][1]
    lag_delta = lagrange_results[i][2]
    new_val = newton_results[i][1]
    new_delta = newton_results[i][2]

    lag_delta_str = f"{lag_delta:.8f}" if lag_delta is not None else "---"
    new_delta_str = f"{new_delta:.8f}" if new_delta is not None else "---"

    print(f"{deg:<12} {lag_val:<18.10f} {lag_delta_str:<18} {new_val:<18.10f} {new_delta_str:<15}")


x_vals = np.linspace(0, 100, 300)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))


ax1.scatter(x_data, y_data, color="red", s=100, label="Sensor Data", zorder=5,
            edgecolors='black', linewidths=2)
ax1.axvline(x_target, color="gray", linestyle="--", linewidth=2,
            label=f"x = {x_target} km", alpha=0.7)

colors = plt.cm.viridis(np.linspace(0, 1, len(degrees_to_compute)))
for idx, (deg, val, delta, nodes, y_nodes) in enumerate(lagrange_results):
    y_vals = [lagrange_interpolation(nodes, y_nodes, xv) for xv in x_vals]
    ax1.plot(x_vals, y_vals, label=f"Degree {deg}", linewidth=2, color=colors[idx])

ax1.set_title("Lagrange Interpolation (All Degrees)", fontsize=14, fontweight='bold')
ax1.set_xlabel("Distance (km)", fontsize=12)
```

```
ax1.set_ylabel("Temperature (degree C)", fontsize=12)
ax1.legend(fontsize=9, loc='best')
ax1.grid(True, alpha=0.3)

ax2.scatter(x_data, y_data, color="red", s=100, label="Sensor Data", zorder=5,
            edgecolors='black', linewidths=2)
ax2.axvline(x_target, color="gray", linestyle="--", linewidth=2,
            label=f"x = {x_target} km", alpha=0.7)

for idx, (deg, val, delta, nodes, table) in enumerate(newton_results):
    y_vals = [newton_interpolation(nodes, table, xv) for xv in x_vals]
    ax2.plot(x_vals, y_vals, label=f"Degree {deg}", linewidth=2, color=colors[idx])

ax2.set_title("Newton's Divided Difference (All Degrees)", fontsize=14, fontweight='bold')
ax2.set_xlabel("Distance (km)", fontsize=12)
ax2.set_ylabel("Temperature (degree C)", fontsize=12)
ax2.legend(fontsize=9, loc='best')
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('interpolation_all_degrees.png', dpi=300, bbox_inches='tight')
plt.show()



fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

all_lag_degrees = [deg for deg, _, _, _, _ in lagrange_results]
all_lag_values = [val for _, val, _, _, _ in lagrange_results]

all_new_degrees = [deg for deg, _, _, _, _ in newton_results]
all_new_values = [val for _, val, _, _, _ in newton_results]

ax1.plot(all_lag_degrees, all_lag_values, marker='o', linewidth=2.5, markersize=10,
         color='#2E86AB', markeredgecolor='black', markeredgewidth=1.5)
ax1.set_title("Lagrange Interpolation Values at x=45 km", fontsize=14, fontweight='bold')
ax1.set_xlabel("Polynomial Degree (k)", fontsize=12)
ax1.set_ylabel("Pk(45) Temperature (degree C)", fontsize=12)
ax1.grid(True, alpha=0.4, linestyle='--')
ax1.set_xticks(all_lag_degrees)

for deg, val in zip(all_lag_degrees, all_lag_values):
    ax1.annotate(f'{val:.4f}degree C', xy=(deg, val), xytext=(0, 10),
                 textcoords='offset points', ha='center', fontsize=8,
                 bbox=dict(boxstyle='round,pad=0.4', facecolor='yellow', alpha=0.8))

```

```
ax2.plot(all_new_degrees, all_new_values, marker='s',
    linewidth=2.5, markersize=10,
          color='#A23B72', markeredgecolor='black',
             markeredgewidth=1.5)
ax2.set_title("Newton Interpolation Values at x=45 km",
    fontsize=14, fontweight='bold')
ax2.set_xlabel("Polynomial Degree (k)", fontsize=12)
ax2.set_ylabel("Nk(45) Temperature (degree C)", fontsize=12)
ax2.grid(True, alpha=0.4, linestyle='--')
ax2.set_xticks(all_new_degrees)

for deg, val in zip(all_new_degrees, all_new_values):
    ax2.annotate(f'{val:.4f}degree C', xy=(deg, val), xytext=(0,
        10),
                 textcoords='offset points', ha='center',
                    fontsize=8,
                 bbox=dict(boxstyle='round,pad=0.4',
                    facecolor='yellow', alpha=0.8))

plt.tight_layout()
plt.savefig('convergence_normal_scale.png', dpi=300,
    bbox_inches='tight')
plt.show()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

lag_degrees_delta = [deg for deg, _, delta, _, _ in
    lagrange_results if delta is not None]
lag_deltas = [delta for _, _, delta, _, _ in lagrange_results if
    delta is not None]

ax1.plot(lag_degrees_delta, lag_deltas, marker='o', linewidth=2.5,
    markersize=10,
          color='#2E86AB', markeredgecolor='black',
             markeredgewidth=1.5)
ax1.set_title("Lagrange Convergence Rate (del k)", fontsize=14,
    fontweight='bold')
ax1.set_xlabel("Polynomial Degree (k)", fontsize=12)
ax1.set_ylabel("del k = |Pk(45) - Pk-1(45)| (degree C)",
    fontsize=12)
ax1.grid(True, alpha=0.4, linestyle='--')
ax1.set_xticks(lag_degrees_delta)

for deg, delta in zip(lag_degrees_delta, lag_deltas):
    ax1.annotate(f'{delta:.6f}', xy=(deg, delta), xytext=(0, 10),
                 textcoords='offset points', ha='center',
                    fontsize=8,
                 bbox=dict(boxstyle='round,pad=0.4',
                    facecolor='lightgreen', alpha=0.8))

new_degrees_delta = [deg for deg, _, delta, _, _ in newton_results
    if delta is not None]
new_deltas = [delta for _, _, delta, _, _ in newton_results if
    delta is not None]

ax2.plot(new_degrees_delta, new_deltas, marker='s', linewidth=2.5,
    markersize=10,
```

```
          color='#A23B72', markeredgecolor='black',
             markeredgewidth=1.5)
ax2.set_title("Newton Convergence Rate (del k)", fontsize=14,
    fontweight='bold')
ax2.set_xlabel("Polynomial Degree (k)", fontsize=12)
ax2.set_ylabel("del k = |Nk(45) - Nk-1(45)| (degree C)",
    fontsize=12)
ax2.grid(True, alpha=0.4, linestyle='--')
ax2.set_xticks(new_degrees_delta)

# Add value labels
for deg, delta in zip(new_degrees_delta, new_deltas):
    ax2.annotate(f'{delta:.6f}', xy=(deg, delta), xytext=(0, 10),
                 textcoords='offset points', ha='center',
                    fontsize=8,
                 bbox=dict(boxstyle='round,pad=0.4',
                    facecolor='lightgreen', alpha=0.8))

plt.tight_layout()
plt.savefig('delta_convergence_rate.png', dpi=300,
    bbox_inches='tight')
plt.show()


print("\n\n" + "||=" + "=" * 78 + "=||")
print("||" + " " * 32 + "FINAL SUMMARY" + " " * 33 + "||")
print("||_" + "=" * 78 + "_||\n")

print(f"  Estimated temperature at x = {x_target} km using highest
    degree polynomial (Degree 8):")
print(f"  {'--' * 76}")
print(f"    Lagrange Method:     {lagrange_results[-1][1]:.10f}
    degree C")
print(f"    Newton's Method:     {newton_results[-1][1]:.10f}
    degree C")
print(f"    Absolute Difference: {abs(lagrange_results[-1][1] -
    newton_results[-1][1]):.2e} degree C")
print(f"\n  Both methods produce identical results (as expected
    theoretically).")

print(f"\n  Convergence Analysis:")
print(f"  {'--' * 76}")
print(f"    Final convergence (del 8):
    {lagrange_results[-1][2]:.10f} degree C")
print(f"    This represents the change from Degree 7 to Degree 8
    polynomial.")
```