Numerical Analysis
Lab 3 : Interpolation

Objective

The goal of this lab is to understand the principles of polynomial interpolation and to implement two classic methods: Lagrange and Newton interpolation. Using MATLAB, students will code these methods, apply them to various datasets, and compare their performance and accuracy. They will also visualize the interpolating polynomials to better grasp the differences between the approaches and how well they approximate the original data.

1 Introduction to Interpolation

Interpolation is a method used to estimate the value of a function based on a set of known data points. Polynomial interpolation consists of constructing a **unique polynomial** that passes exactly through these points. In other words, we seek a polynomial $P(x)$ such that:

$$P(x_i) = y_i \quad \text{for all } i = 0, 1, \ldots, n$$

This polynomial is then used to approximate the actual function between the known data points.

2 Lagrange Method

The Lagrange interpolation method constructs a polynomial $P(x)$ of degree at most n passing through the points $(x_0, y_0), \ldots, (x_n, y_n)$ using the following formula:

$$P(x) = \sum_{i=0}^{n} L_i(x) \cdot f(x_i),$$

where,

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

is the Lagrange basis polynomial associated with the point x_i.

Exercise 2.1 – Implementation

Write a MATLAB function `Lagrange(x, y, x_eval)` that takes as input:

- `x`: vector of interpolation points' x-coordinates,
- `y`: vector of corresponding y-values,
- `x_eval`: vector of points where the polynomial should be evaluated.

The function should return the value of the interpolating polynomial evaluated at the points `x_eval`.

Exercise 2.2 – Visualization

Plot the resulting polynomial together with the interpolation points to visualize the curve.

Test the function with the following dataset:

$$x = [1, 2, 3], \quad y = [2, 3, 5]$$

and evaluate the polynomial at 100 points evenly spaced between 1 and 3.

$$x_{\text{eval}} = \texttt{linspace}(1, 3, 100).$$

3 Newton Method

$$P(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)\cdots(x - x_{n-1})$$

where a_i are the divided difference coefficients.

Exercise 3.1 – Calculation of divided differences

Write a function `Newton_coeff(x, y)` that returns the table of coefficients a_i.

Exercise 3.2 – Evaluation of the Newton polynomial

Write a function `newton_eval(x, ai, x_eval)` that uses the coefficients a_i obtained and the recursive formula to evaluate the polynomial.
Test with the same dataset as for the Lagrange method.

4 Comparison of Lagrange and Newton Methods

- Compare the two implementations in terms of:
  - Code simplicity and readability.
  - Computation time (try with $n = 20$, $n = 100$, $n = 500$).
- Plot the interpolating polynomials obtained using both methods on the same graph for visual comparison.

- Discuss whether the methods give identical results and where numerical differences might appear.

Exercise 4.1 – Practical Comparison
Given $f(x) = \sin(x)$ with points at $x = 0, \pi/2, \pi$:

1. Compute Lagrange polynomial analytically
2. Implement Newton's divided differences in MATLAB
3. Use `plot` to visualize the results.

5 Real-life Application: Sensor Calibration

Context: In industrial applications, sensor outputs may not perfectly follow the theoretical response curve. Interpolation can help build a calibration curve from measured data.

Exercise 5.1 – Sensor Calibration Curve

- Suppose a temperature sensor gives the following readings:

Sensor Output (V)	True Temperature (°C)
0.50	0
1.10	10
1.75	20
2.50	30
3.40	40

- Use Newton interpolation to create a calibration curve (temperature as a function of voltage).
- Evaluate the temperature corresponding to a new sensor output, e.g., $V = 2.0$ V.
- Plot the calibration curve and compare it with the measured points.