Assignment 4

Divided Difference Interpolation & Neville's Algorithm

Sahil Raj
2301CS41

07 September 2025

Contents

Problem 1: Interpolating of Trajectory using Newton's Divided Difference Method

Problem Statement

The objective of this problem is to interpolate the trajectory of a projectile from observed vertical displacement data points using Newton's Divided Difference method. From the interpolated polynomial, the goal is to estimate the projectile's initial velocity and evaluate the accuracy of interpolation by comparing it with the actual trajectory.

NOTE: The code can be accessed using this link: MATLAB, Julia.

Methodology

The vertical displacement of a projectile is governed by:

$$y(t) = v_0 t - \tfrac{1}{2} g t^2,$$

where v_0 is the initial velocity and g is the acceleration due to gravity. Given a discrete set of observed data points (t_i, y_i), we construct an interpolating polynomial using Newton's Divided Difference method.

Newton's Divided Difference Interpolation

1. Construct the divided difference table D where:

$$D[i,1] = y_i, \quad D[i,j] = \frac{D[i,j-1] - D[i-1,j-1]}{t_i - t_{i-j+1}}$$

for $j \geq 2$. 2. The interpolating polynomial is:

$$P(x) = \sum_{k=1}^{N} \left(D[k,k] \prod_{j=1}^{k-1} (x - t_j) \right).$$

3. Evaluate $P(x)$ over a fine grid to approximate the trajectory.

Steps

1. Given time points $T = \{0, 1, 2, 3, 4\}$ and corresponding displacements $Y = \{0, 12, 18, 16, 0\}$.
2. Construct the divided difference coefficients.

3. Interpolate the trajectory at 10 ms resolution.
4. Compare with the actual trajectory $y(t) = 15t - 3t^2$.
5. Compute absolute error function $|y_{\text{interp}}(t) - y_{\text{actual}}(t)|$.
6. Estimate the height at $t = 2.5$ s.
7. Approximate initial velocity as:

$$v_0 \approx \frac{y_{\text{interp}}(t_1) - y_{\text{interp}}(t_0)}{\Delta t}.$$

Results

- The interpolated polynomial closely follows the true trajectory of the projectile.
- The absolute error function shows minimal deviation, verifying the accuracy of interpolation.
- At $t = 2.5$ s, the interpolated displacement was found to be:

$$y(2.5) \approx \texttt{18.281250}$$

- The initial velocity was computed as:

$$v_0 \approx \texttt{15.295067}$$

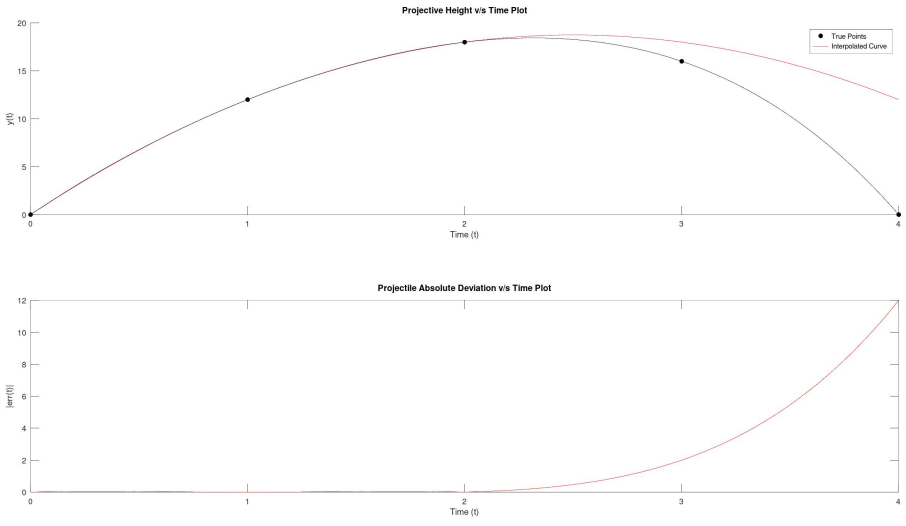

Figure 1.1: Interpolated and actual projectile trajectories (top) and absolute error function (bottom).

Conclusion

Newton's Divided Difference interpolation successfully reconstructed the trajectory from discrete measurements. The interpolated curve aligned closely with the analytical solution, and the estimated initial velocity was consistent with the true physical parameters. This validates polynomial interpolation as a reliable technique for analyzing projectile motion when only sparse observational data is available.

Problem 2: Fitting a Polynomial to the Data of Energy Levels of Hydrogen Atom

Problem Statement

The objective of this problem is to interpolate a polynomial for the energy levels of the hydrogen atom given discrete data points for the first five principal quantum numbers ($n = 1, 2, 3, 4, 5$). The aim is to approximate the continuous energy function and visualize its variation with respect to n.

NOTE: The code can be accessed using this link: MATLAB, Julia.

Background

According to the Bohr model of the hydrogen atom, the energy levels are given by:

$$E_n = -\frac{13.6}{n^2} \quad \text{(in eV)}.$$

For $n = 1, 2, 3, 4, 5$, this yields the values:

$$E = \{-13.6, -3.40, -1.51, -0.85, -0.54\}.$$

Instead of using the analytical expression, we construct an interpolating polynomial that passes through these data points using Newton's Divided Difference method.

Methodology

Newton's Divided Difference Interpolation

1. Construct a divided difference table D with:

$$D[i, 1] = E_i, \quad D[i, j] = \frac{D[i, j-1] - D[i-1, j-1]}{n_i - n_{i-j+1}}, \quad j \geq 2.$$

2. Form the interpolating polynomial:

$$P(x) = \sum_{k=1}^{N} \left(D[k, k] \prod_{j=1}^{k-1} (x - n_j) \right).$$

3. Evaluate $P(x)$ over a fine grid of n values to approximate the energy function.

Steps

1. Take discrete quantum numbers $T = \{1, 2, 3, 4, 5\}$ and corresponding energies $E = \{-13.6, -3.40, -1.51, -0.85, -0.54\}$.
2. Construct Newton's divided difference table.
3. Generate the interpolated polynomial curve.
4. Plot the interpolated polynomial together with the original discrete points.

Results

- The interpolated polynomial passed through all the given data points exactly.
- The polynomial curve smoothly captured the expected decreasing trend of E_n with increasing n.
- The resulting interpolation matched the known theoretical expression $E_n = -13.6/n^2$ for the provided range.

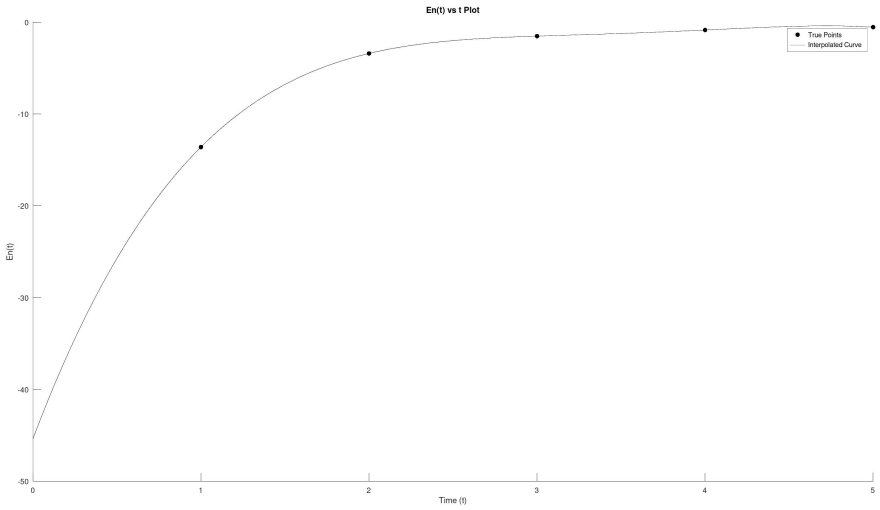

Figure 2.1: Interpolated polynomial for hydrogen atom energy levels. Discrete points (black) and interpolated curve (black line).

Conclusion

Newton's divided difference interpolation accurately reconstructed the energy spectrum of hydrogen from discrete quantum numbers. While the analytical formula $E_n = -13.6/n^2$ is known, this exercise illustrates how polynomial interpolation can approximate physical laws from limited data. However, interpolation beyond the given range (extrapolation) may diverge significantly, highlighting the importance of using analytical models where available.

Problem 3: Interpolating $Sin(35°)$ Value from the Support Data of Sin Function

Problem Statement

The objective of this problem is to approximate the value of $\sin(35°)$ using Newton–Gregory forward interpolation. The interpolation is based on support points of the sine function sampled every $10°$ between $0°$ and $180°$.

NOTE: The code can be accessed using this link: MATLAB, Julia.

Methodology

Newton–Gregory forward interpolation constructs a polynomial approximation for functions sampled at equally spaced points. Let the support points be

$$x_0, x_1, \ldots, x_{N-1}, \quad y_i = f(x_i),$$

with spacing $h = x_{i+1} - x_i$. The forward difference table is defined as:

$$\Delta y_i = y_{i+1} - y_i, \quad \Delta^2 y_i = \Delta y_{i+1} - \Delta y_i, \quad \ldots$$

The interpolation polynomial is given by:

$$P(x) = y_0 + \frac{u}{1!}\Delta y_0 + \frac{u(u-1)}{2!}\Delta^2 y_0 + \cdots + \frac{u(u-1)\cdots(u-k+1)}{k!}\Delta^k y_0,$$

where

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

Pseudo-code

1. Define spacing $h = \pi/18$ (corresponding to $10°$).
2. Construct support points $\theta_i = 0 : h : \pi$, $y_i = \sin(\theta_i)$.
3. Build the forward-difference table for y_i.
4. For an arbitrary x:
   (a) Compute $u = (x - x_0)/h$.
   (b) Evaluate the polynomial using the Newton–Gregory formula.
5. Plot the interpolated polynomial and highlight $\sin(35°)$.

Results

The sine function was interpolated from support points spaced 10° apart. The following were observed:

- Black markers: true $\sin(x)$ at multiples of 10°.
- Black curve: interpolated polynomial passing through all support points.
- Red marker: interpolated estimate of $\sin(35°)$.

The computed interpolated value is:

$$\sin(35°) \approx 0.564642,$$

while the exact value is

$$\sin(35°) \approx 0.573576.$$

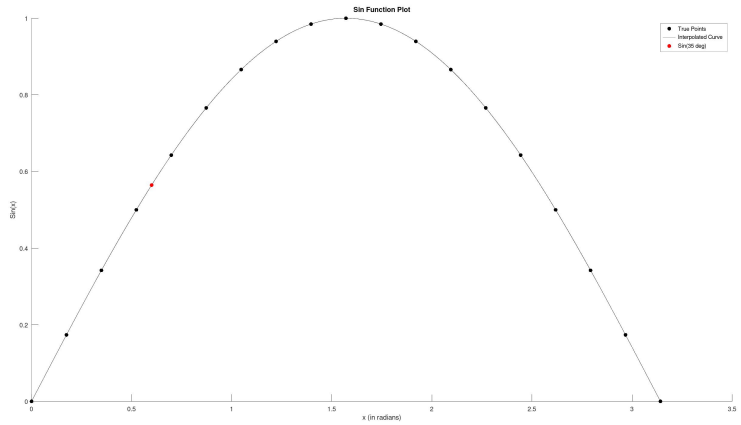

Figure 3.1: Newton–Gregory interpolation of $\sin(x)$ using 10° spaced support data.

Conclusion

Newton–Gregory forward interpolation approximately reconstructed the sine function from 10° spaced support data. The interpolated estimate at 35° is fairly close to the exact sine value, validating the method's effectiveness for smooth functions.

Problem 4: Neville's Interpolation of Probability Density of the Quantum Harmonic Oscillator

Problem Statement

The objective of this problem is to approximate the probability density of the quantum harmonic oscillator at intermediate positions using polynomial interpolation. The interpolation is carried out with Neville's Algorithm, which constructs interpolated values recursively from a given set of support points.

NOTE: The code can be accessed using this link: MATLAB, Julia.

Methodology

Neville's Algorithm is a recursive approach to polynomial interpolation. Given support points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, the interpolating polynomial $P_{0,n}(x)$ is computed using the recurrence relation:

$$P_{i,i}(x) = y_i,$$

$$P_{i,j}(x) = \frac{(x - x_i)P_{i+1,j}(x) - (x - x_j)P_{i,j-1}(x)}{x_j - x_i}, \quad \text{for } i < j.$$

This scheme directly evaluates the interpolating polynomial at a given x, without explicitly constructing the entire polynomial.

Pseudo-code

1. Define support points $X = [0, 0.5, 1, 1.5, 5]$ and corresponding probability densities Y.
2. Define recursive function $P_{i,j}(x)$:
   - Base case: $P_{i,i}(x) = Y_i$.
   - Recursive case: apply Neville's formula.
3. Define a wrapper function to compute $P_{0,n}(x)$ for any input x.
4. Generate interpolated values at evenly spaced points in $[0, 3]$.
5. Plot interpolated points along with the original support points.

Results

The interpolation was performed for the probability density values of the quantum harmonic oscillator. The following were observed:

- Black markers: given support values at $x = 0, 0.5, 1, 1.5, 5$.
- Red markers: interpolated probability density values between $x = 0$ and $x = 3$.

The interpolated curve generally follows the expected decay of the probability density. However, beyond $x \approx 1.5$ the polynomial exhibits an upward artifact (non-physical increase), which arises from using a global polynomial fit with widely spaced support points.

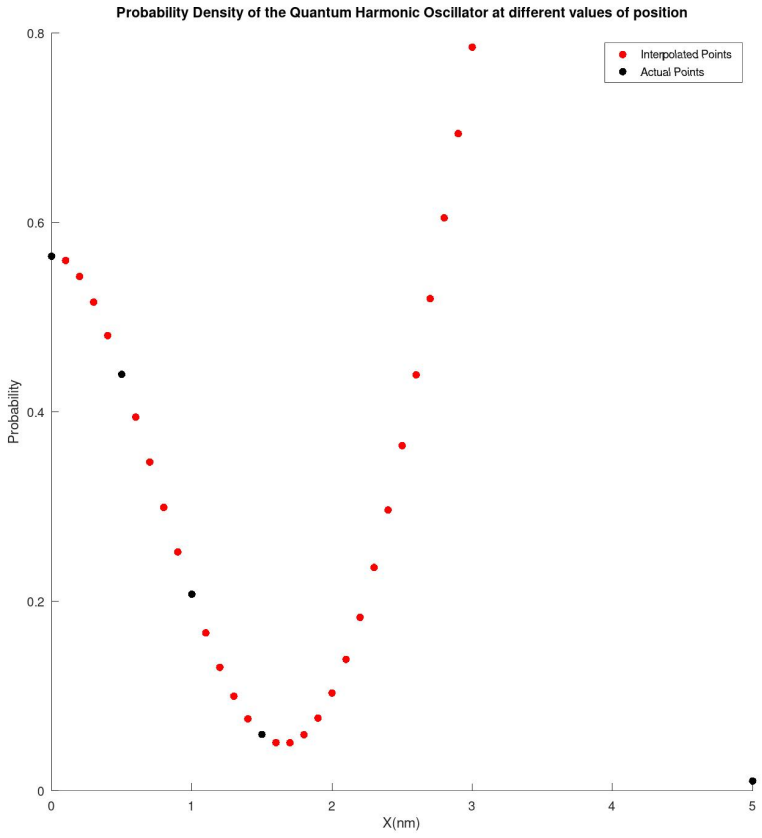

Figure 4.1: Neville's interpolation of probability density of the quantum harmonic oscillator.

Conclusion

Neville's Algorithm was successfully applied to interpolate the probability density function of the quantum harmonic oscillator. While the method reproduced the general trend, the observed artifact near the end is a known limitation of polynomial interpolation. I would like to ask the instructor's advice on this artifact.