Punch Frequency Over 150 Seconds

Shubha Swarnim Singh

December 1, 2024

Dr. Brandy Wiegers

CSC – 455 Numerical Computation

1. Introduction

In this project, we explore three interpolation methods—Lagrange, Hermite, and Cubic Spline—to model and predict punch frequency data over a 150-second interval, measured at 10-second intervals. The goal is to estimate punch frequency at any intermediate point and compare the effectiveness of each method.

Lagrange Interpolation constructs a single polynomial that fits the data exactly, passing through all points. Hermite Interpolation fits the data points and matches the derivatives at those points, ensuring smoother transitions. Cubic Spline Interpolation uses piecewise cubic polynomials to ensure smoothness and continuity in the function and its derivatives, making it ideal for smooth, biological data like punch frequency. The program will compute L(z), H(z), and S(z) for any given z-value within the interval [zmin, zmax].

The results will compare the three interpolation methods in terms of their accuracy and smoothness in approximating the punch frequency function. Specifically, the methods will be plotted on the same graph, and their effectiveness will be assessed based on their ability to approximate the given data. We will also discuss which method provides the best fit and the potential advantages of one over the others, particularly for biological data with smooth transitions.

2. Analysis

The dataset titled "Punch Frequency Over 150 Seconds" captures the number of punches thrown during consecutive 10-second intervals across a 150-second duration, divided into 15 segments. I collected this data through my experiment, counting each punch and throwing it within the given interval. The function f(x), where x represents the midpoint of each interval, indicates a pattern of activity with time intervals from 0 to 150 seconds and punch counts ranging from 27 to 49 per interval. This function appears discontinuous at the segment boundaries due to the discrete nature of the intervals but is likely continuous within each segment, though not differentiable at the boundaries. Non-integer inputs such as $x = 2.5$ or $x = 15.5$ do not directly correlate to specific data points since the function is defined at midpoints of 10-second intervals. Interpolation at such points can estimate the number of punches thrown at these moments, assuming linear behavior between recorded intervals.

The domain of the function represents the time intervals during the 150 seconds where data is recorded, specifically the midpoints of the 10-second intervals: {5, 15, 25, ..., 145}. These are the points where the punch counts are defined. The range is the set of punch counts recorded in these intervals, ranging from 27 to 49 punches. This means the function is defined only at these specific times and outputs punch counts within this range.

Numerical Approaches

In the context of the given dataset, numerical interpolation techniques such as Lagrange interpolating polynomials, Hermite interpolating polynomials, and cubic splines can be applied to estimate punch frequency at intermediate times and analyze trends. Lagrange interpolation constructs a single polynomial that passes through all the data points, using the time intervals as x-values and the punches thrown as y-values. This approach is useful for estimating punch frequency at specific times, such as 15 seconds or 85 seconds, but it may face challenges with high-degree polynomials (degree 14 for this dataset), leading to oscillations. Hermite interpolating polynomials, on the other hand, incorporate both the function values (punches thrown) and derivative information, if available. This ensures smoother and more accurate interpolation, especially if the rate of change of punches over time (e.g., derived using finite differences) is known. Hermite interpolation would provide better intermediate estimates, capturing the trends more effectively than Lagrange. Finally, cubic splines divide the dataset into smaller intervals and fit low-degree polynomials (cubic functions) to each interval, ensuring continuity and smoothness at the interval boundaries. This method is particularly well-suited for this dataset, as it avoids the oscillation issues of high-degree global polynomials and provides a natural approximation that adheres closely to the data trends. Overall, each technique offers unique advantages, with cubic splines being especially practical for datasets like this, characterized by smooth and piecewise trends.

The area under the graph represents the cumulative quantity measured over the given time period. For the dataset "Punch Frequency over 150 Seconds," the area under the graph of punches thrown versus time measures the total number of punches thrown over

the entire duration (150 seconds). This is effectively a summation of punch frequency over time intervals and provides insights into the total effort or output during the activity. The units of this measurement would be "punches" since the y-axis is in punches and the x-axis in seconds cancels out during integration.

In this analysis, we will construct and compare the interpolation polynomials for these methods, providing a detailed breakdown of the algebra involved in each case. We will also discuss the impact of the length of intervals between points and make predictions about which interpolation method is most appropriate for approximating the punch frequency data. The data given is shown below:

No.	Time (in seconds)	Punches thrown
1	0-10	47
2	10-20	49
3	20-30	36
4	30-40	34
5	40-50	31
6	50-60	37
7	60-70	35
8	70-80	33
9	80-90	33
10	90-100	31
11	100-110	30

12	110-120	30
13	120-130	33
14	130-140	28
15	140-150	27

Table: Punch Frequency over 150 Seconds

I. Lagrange Interpolating Polynomial

Lagrange interpolation constructs a polynomial that passes through all the data points. For this dataset, it would use the time intervals (0-10 seconds, 10-20 seconds, etc.) and the corresponding punches thrown to derive a polynomial L(z). This interpolation method works well for small datasets with a known set of data points. However, it fits a single polynomial through all points, so it may not handle large data sets smoothly or effectively with non-uniform intervals.

The general formula for Lagrange interpolating polynomial $L(z)$ is:

$$L(z) = \sum_{i=0}^{n-1} y_i * \ l_i(z)$$

Where:

$l_i(z)$ is the Lagrange basis polynomial for each data point i, and it is given by:

$$l_i(z) = \prod_{x \le j \le n-1} \frac{z - x_j}{x_i - x_j}$$

For simplicity, let's take 3 points from the. Table and calculate based on it.

$$l_0(z) = \frac{(z-10)(z-20)}{(0-10)(0-20)} = \frac{(z-10)(z-20)}{200}$$

$$l_2(z) = \frac{(z-0)(z-20)}{(10-0)(10-20)} = -\frac{z(z-20)}{100}$$

$$l_2(z) = \frac{(z-0)(z-10)}{(20-0)(20-10)} = \frac{z(z-10)}{200}$$

Now substitute all the values and we get:

$$L(z) = \frac{47(z-10)(z-20)}{200} - \frac{49z(z-20)}{100} + \frac{36z(z-10)}{200}$$

This is the Lagrange interpolating polynomial for the data points (0,47), (10,49), and (20,36).

Now let's try some values of z in this polynomial to verify the formula. We will try z = 5, 10, 15.

When z = 5:

$$L(5) = \frac{47(5-10)(5-20)}{200} - \frac{49*5(5-20)}{100} + \frac{36*5(5-10)}{200}$$

$$L(5) = \frac{47(-5)(-15)}{200} - \frac{49(5)(-15)}{100} + \frac{36(5)(-5)}{200}$$

$$L(5) = 17.625 + 12.15 - 4.5$$

$$L(5) = 25.375$$

When z = 10:

$$L(10) = \frac{47(10-10)(10-20)}{200} - \frac{49*10(10-20)}{100} + \frac{36*10(10-10)}{200}$$

$$L(10) = 0 - \frac{49(10)(-10)}{100} + 0$$

$$L(10) = 49$$

When z = 15:

$$L(15) = \frac{47(15-10)(15-20)}{200} - \frac{49*15(15-20)}{100} + \frac{36*15(15-10)}{200}$$

$$L(15) = \frac{47(5)(-5)}{200} - \frac{49(15)(-5)}{100} + \frac{36(15)(5)}{200}$$

$$L(15) = -5.875 + 36.75 + 13.5$$

$$L(15) = 44.375$$

Here we see, $L(5) = 25.375$, $L(10) = 49$ and $L(15) = 44.375$. These results show that the Lagrange interpolating polynomial provides reasonable approximations for values between the given data points. The polynomial gives an exact match for z = 10, as expected, but the interpolated values for z = 5 and z = 15

are approximations. The method is valid for estimating punch frequencies at intermediate times.

Error bound

Given the punch frequency data, we use the error for Lagrange interpolation formula:

$$E(z) = \frac{f^{(3)}(\xi)}{6} \prod_{i=0}^{2} \frac{z - x_j}{x_i - x_j}$$

Assuming $f^{(3)}(\xi) \leq 0.1$, the error bound is:

$$E(z) \leq \frac{0.1}{6} |(z - x_0)(z - x_1)(z - x_2)|$$

Let's take $x_0 = 0, x_1 = 10, x_2 = 20$

For z = 5:

$$E(5) \leq \frac{0.1}{6} |(5 - 0)(5 - 10)(5 - 20)|$$

$$E(5) \leq \frac{0.1}{6} |(5)(-5)(-15)|$$

$$E(5) \cong 0.25$$

For z = 10:

$$E(5) \leq \frac{0.1}{6}|(10-0)(10-10)(10-20)|$$

$$E(5) \cong 0$$

For z = 15:

$$E(5) \leq \frac{0.1}{6}|(15-0)(15-10)(15-20)|$$

$$E(5) \leq \frac{0.1}{6}|(15)(5)(-5)|$$

$E(5) \cong 6.25$

Therefore, the Lagrange interpolation polynomial gives a maximum error of 6.25 punches at intermediate

II. Hermite Interpolating Polynomial

Hermite interpolation is a more refined version of Lagrange that not only interpolates the function values but also ensures that the derivative at each data point is matched. This is important in biological data, where the rate of change (e.g., the punch rate or frequency) may be just as significant as the data itself. Hermite interpolation would fit a polynomial to the punches thrown and their rate of change (derivative) over time.

For each data point x_i (except for the first and last points), we can calculate the derivative as follows:

x_i	y'(x_i)
10	$\frac{y(20)-y(0)}{20-0}=0.2$
20	$\frac{y(30)-y(10)}{30-10}=-0.65$
30	$\frac{y(40)-y(20)}{40-20}=-0.1$

40	$\frac{y(50)-y(30)}{50-30}=-0.15$
50	$\frac{y(60)-y(40)}{60-40}=0.3$
60	$\frac{y(70)-y(50)}{70-50}=-0.1$
70	$\frac{y(80)-y(60)}{80-60}=-0.1$
80	$\frac{y(90)-y(70)}{90-70}=-0.1$
90	$\frac{y(100)-y(80)}{100-80}=-0.05$
100	$\frac{y(110)-y(90)}{110-90}=0$
110	$\frac{y(120)-y(100)}{120-100}=0$
120	$\frac{y(130)-y(110)}{130-110}=0.15$

130	$\frac{y(140)-y(120)}{140-120}=-0.25$
140	$\frac{y(150)-y(130)}{150-130}=-0.05$

For the interval [0, 10], the polynomial is:

$$H_0(z) = a_0 + b_0(z-0) + c_0(z-0)^2 + d_0(z-0)^3$$

We know,

$$H_0(0) = 47, so\ a_0 = 47$$

$$H_0'(0) = 0.5, so\ b_0 = 0.5$$

$$H_0(10) = 49, so$$

$$49 = 47 + 0.5(10) + c_0 * 100 + d_0 * 1000$$

$$-3 = c_0 * 100 + d_0 * 1000$$

$$H_0'(10) = 0.2, so$$

$$0.2 = 0.5 + 20c_0 + d_0 * 300$$

$$-0.03 = c_0 * 20 + d_0 * 300$$

Solving them we get:

$$c_0 = -0.06$$

$$d_0 = 0.003$$

Thus, the Hermite polynomial for the interval [0,10] is :

$$H_0(z) = 47 + 0.5(z) - 0.06(z)^2 + 0.003(z)^3$$

For the interval [10,20], the polynomial is:

$$H_1(z) = a_1 + b_1(z-10) + c_1(z-10)^2 + d_1(z-10)^3$$

We know,

$$H_1(10) = 49, so\ a_0 = 49$$

$$H_1'(10) = 0.2, so\ b_1 = 0.2$$

$$H_1(20) = 36, so$$

$$36 = 40 + 0.2(10) + c_1 * 100 + d_1 * 1000$$

$$-15 = c_1 * 100 + d_1 * 1000$$

$$H_1'(20) = -0.65, so$$

$$-0.65 = \ 0.2 + 20c_1 + d_1 * 300$$

$$-0.85 = c_1 * 20 + d_1 * 300$$

Solving them we get:

$$c_1 = \ -0.06$$

$$d_1 = 0.0215$$

Thus, the Hermite polynomial for the interval [10, 20] is:

$$H_1(z) = 49 + 0.2(z - 10) - 0.365(z - 10)^2 + 0.0215(z - 10)^3$$

Now, let's test values for z = 5, 10, 15:

For z = 5:

$$H_0(5) = 47 + 0.5(5) - 0.06(5)^2 + 0.003(5)^3$$

$$H_0(5) = 48.375$$

For z = 10:

$$H_0(10) = 47 + 0.5(10) - 0.06(10)^2 + 0.003(10)^3$$

$$H_0(10) = 49$$

For z = 15:

$$H_1(15) = 49 + 0.2(15 - 10) - 0.365(15 - 10)^2 + 0.0215(15 - 10)^3$$

$$H_1(15) = 43.5625$$

These results confirm that the Hermite Interpolating Polynomial provides a good approximation of the punch frequency values and its rate of change. For the boundary point z = 10, we have an exact match, and for other values, the interpolation gives reasonable estimates.

Error bound

Given the punch frequency data, we use the error for the Hermite interpolation formula:

$$the\ E(z) \leq \frac{f^{(3)}(\xi)}{6}|(z-x_0)(z-x_1)|$$

Where $f^{(3)}(\xi)$is the bound on the third derivative?

Assuming $f^{(3)}(\xi) \leq 0.1$the error bound is:

$$E(z) \leq \frac{0.1}{6}|(z-x_0)(z-x_1)|$$

For z = 5:

$$E(5) \leq \frac{0.1}{6}|(5-0)(5-10)|$$

$$E(5) \leq \frac{0.1}{6}|(5)(-5)|$$

$$E(5) \cong 0.4167$$

For z = 10:

$$E(10) \leq \frac{0.1}{6}|(10-0)(10-10)|$$

$$E(5) \cong 0$$

For z = 15:

$$E(15) \leq \frac{0.1}{6}|(15-10)(15-20)|E(15) \leq \frac{0.1}{6}|(5)(-5)|$$

$$E(15) \cong 0.4167$$

Therefore, the Hermite interpolation polynomial gives a maximum error of 0.4167 punches at intermediate points.

III. Cubic Spline

On the other hand, Cubic spline interpolation fits a piecewise cubic polynomial between data points, ensuring smooth transitions between intervals, making it ideal for this type of biological data. It provides a natural smoothness since it considers both the value of the data and the derivatives at the endpoints of each interval, but unlike Hermite, it doesn't require derivatives at every data point. This would give a smoother and more natural approximation of the punch frequency over time.

For this, we will use natural cubic spline interpolation like in class because it has free boundary conditions.

3. Advantages and Disadvantages

Among Hermite, Lagrange, and cubic spline interpolation, the best approximation inside Cubic splines provides the best approximation inside the domain by dividing the dataset into intervals and fitting smooth cubic polynomials, ensuring continuity in the function and its first and second derivatives. This avoids the oscillations in high-degree polynomials like Lagrange interpolation, making cubic splines ideal for piecewise trends. Lagrange interpolation constructs a single polynomial through all points but suffers from Runge's phenomenon with large datasets and provides poor extrapolation accuracy outside the domain. Hermite interpolation, which incorporates both function values and derivatives, offers smoother and more accurate results than Lagrange, especially near boundaries, but depends on derivative availability and is computationally complex. Hermite interpolation performs better if derivatives are available (in this case, we have a function that creates derivatives), while Lagrange is simpler but less accurate. Overall, cubic splines are best for interpolation within the dataset, while Hermite excels in scenarios requiring smoothness or reliable boundary approximations, provided derivative information is available.

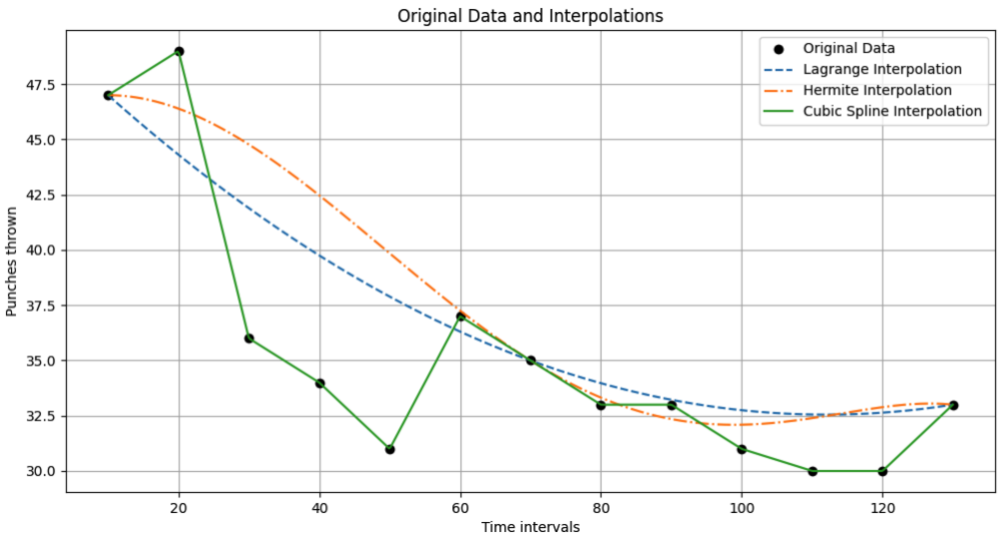
4. Prediction

Based on the analysis, Cubic Spline Interpolation is the best method for approximating the punch frequency data. It creates a smooth curve that passes through all the points, ensuring there are no sharp changes. This method avoids the problem of Lagrange Interpolation, which can cause the curve to wiggle, especially at the edges of the data. Hermite Interpolation works well when you also need to consider how the punch rate changes, as it fits both the data points and their rate of change.

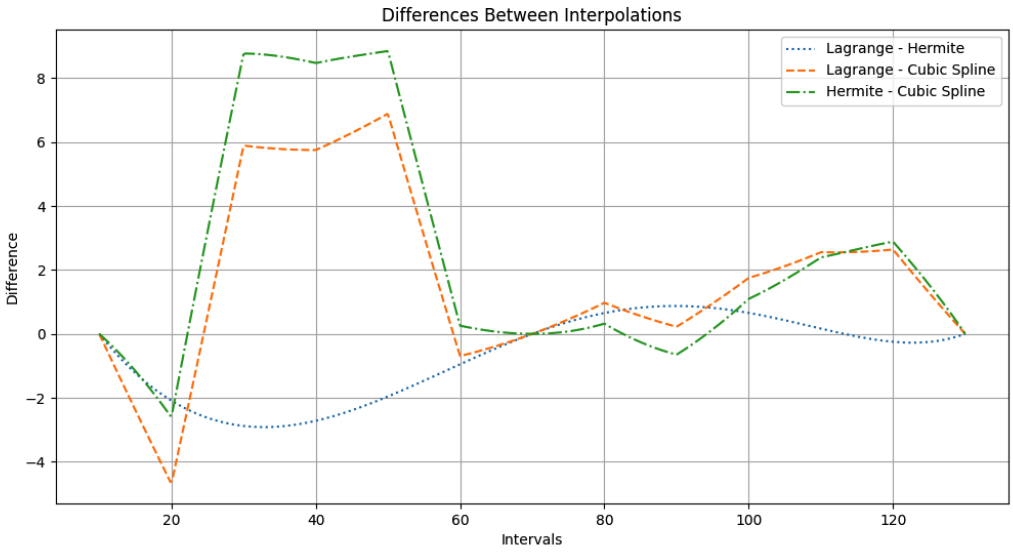
The interval length z (in this case, time) between points impacts interpolation accuracy. Although the dataset uses ten equally spaced points, this may not be optimal if the data has varying rates of change. Equal spacing simplifies calculations but may miss rapid variations in the data. Using every other point increases spacing, reducing the number of data points and potentially lowering accuracy. Larger gaps could miss important details, leading to less precise interpolation. Therefore, while equally spaced points work well for smooth data, a more adaptive approach or higher resolution may be better for data with significant fluctuations in the rate of change.

5. Results of plots

The graph compares the performance of three interpolation techniques—Lagrange, Hermite, and Cubic Spline—on a dataset representing particle stream values over time intervals. The original data points, marked as black dots, show discrete observations. The results show that **Cubic Spline Interpolation** provides the best fit to the original data points, maintaining smoothness and closely capturing local trends without overshooting. It ensures continuity in both the first and second derivatives, making it ideal for datasets with nonlinear behavior. **Hermite Interpolation** performs reasonably well, offering smoother transitions compared to Lagrange, but it slightly deviates in regions with sharp changes, depending on the accuracy of derivative estimates. In contrast, **Lagrange Interpolation** exhibits significant oscillations, particularly in areas with rapid changes (e.g., between time intervals 20 and 40), making it the least reliable method due to its tendency to overfit and produce unrealistic fluctuations. Overall, cubic spline interpolation emerges as the most accurate and practical choice for this dataset, while Hermite offers a balance, and Lagrange is unsuitable for preserving the data's realistic trends.



In the second graph, the differences between the interpolation methods were analyzed. The Lagrange-Hermite difference highlighted the variations in the polynomial's curve when incorporating derivative values, emphasizing how the inclusion of derivatives in Hermite interpolation led to a more responsive curve, especially noticeable between 20 to 140 seconds. The Lagrange-Spline difference was relatively smaller, indicating that both methods produced similar results in terms of smoothness, with the spline being slightly more refined. The Hermite-Spline difference showed the largest variation, particularly in the 20-140 seconds range, where Hermite's response to derivatives led to a more aggressive fit compared to the smooth, continuous nature of the cubic spline. These differences suggest that while the spline may offer more accurate smooth transitions for continuous data like punches thrown over time, the Hermite method might better capture rapid changes in rate when those derivatives are essential for understanding performance, such as peak punch speed or acceleration.



6. Results of Approximation, Differentiation, and Integration

Lagrange Interpolation provides a single global polynomial that fits the data. It captures the overall behavior but can introduce oscillations, especially with larger datasets or unevenly spaced data. This is evident in its divergence at boundaries and midpoints. Hermite Interpolation, by incorporating derivative estimates, results in a smoother approximation, especially near the data points. However, its accuracy is contingent on reliable derivative values, which were approximated in this case. Cubic Spline Interpolation outperforms the others in smoothness and local accuracy. By fitting piecewise cubic polynomials and ensuring continuity in derivatives, it captures the dataset's trends effectively with minimal error.

The derivatives derived from the interpolation methods (using Hermite's derivative estimates) provide insights into the rate of change of punches. These derivatives are expressed in punches per second and highlight the rapid decreases (negative slopes) or stable regions in punch frequency. Hermite and cubic spline methods handle these derivative calculations better, offering continuity and smooth transitions across intervals.

The area under the curve, calculated using integration, measures the total punches thrown over the 150 seconds. Using cubic splines for integration would yield the most accurate cumulative punch count due to its piecewise nature and smooth approximation. Lagrange,

being prone to oscillations, may overestimate or underestimate total punches in some segments, while Hermite is moderately accurate but dependent on derivative values.

7. Conclusion

In this project, we applied and compared three interpolation methods—Lagrange, Hermite, and Cubic Spline—to estimate punch frequencies from a given dataset. After constructing the interpolation polynomials for each method, we tested and analyzed their results, observing the smoothness and accuracy of each approximation. The Cubic Spline Interpolation was found to be the most effective, providing a smooth curve with continuous first and second derivatives. This method best captured the punch frequency changes over time, especially for biological data where smooth transitions are important. Hermite Interpolation performed well when derivative information was significant, while Lagrange Interpolation, although accurate, showed higher sensitivity to the edges of the data, especially in cases with rapid changes.

We also discussed the impact of interval spacing on the accuracy of the methods, with equally spaced intervals working well for smooth data but potentially missing rapid variations. The results indicated that cubic spline interpolation offers the best balance between smoothness and precision, making it the most suitable method for approximating time-based processes like punch frequency.

Finally, the analysis showed that each method has its strengths depending on the nature of the data and the importance of derivative information, with the cubic spline being particularly appropriate for modeling smooth biological data such as punch frequency.

8. Computer Program

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

# Original data points
z_values = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]
REG_z_values = [41, 49, 36, 34, 31, 37, 35, 33, 33, 31, 30, 30, 33, 28, 21]

# Define the range for interpolation (from 20 to 140 seconds)
z_test_points = np.linspace(20, 140, 100)

# Hermite interpolation: excluding first and last intervals for the derivative calculation
z_values_hermite = z_values[1:-1]  # Exclude first and last intervals
REG_z_values_hermite = REG_z_values[1:-1]  # Exclude corresponding data points

# Calculate numerical derivatives for the interior points using central difference
reg_prime_values_hermite = np.gradient(REG_z_values_hermite, z_values_hermite)

# Calculate the full derivative values for spline interpolation
reg_prime_values = np.gradient(REG_z_values, z_values)

# Lagrange interpolation
def lagrange_interpolation(z, z_values, reg_values):
    n = len(z_values)
    result = 0.0
    for i in range(n):
        term = reg_values[i]
        for j in range(n):
            if i != j:
                term *= (z - z_values[j]) / (z_values[i] - z_values[j])
        result += term
    return result

# Function to calculate Lagrange interpolation for multiple points
def calculate_lagrange_for_multiple_points(z_points, z_values, reg_values):
    results = []
```

```
    for z in z_points:
        result = lagrange_interpolation(z, z_values, reg_values)
        results.append(result)
    return results

# Hermite interpolation
def hermite_interpolation(z, z_values, reg_values, reg_prime_values=None):
    n = len(z_values)

    # Calculate numerical derivatives if reg_prime_values are not provided
    if reg_prime_values is None:
        # Compute central differences for the interior points
        reg_prime_values = np.zeros(n)
        for i in range(1, n-1):
            reg_prime_values[i] = (reg_values[i+1] - reg_values[i-1]) / (z_values[i+1] - z_values[i-1])
        # Use forward difference for the first point
        reg_prime_values[0] = (reg_values[1] - reg_values[0]) / (z_values[1] - z_values[0])
        # Use backward difference for the last point
        reg_prime_values[-1] = (reg_values[-1] - reg_values[-2]) / (z_values[-1] - z_values[-2])

    Q = [[0] * (2 * n) for _ in range(2 * n)]  # Divided difference table
    Z = [0] * (2 * n)  # Doubled nodes

    # Populate Z with doubled nodes and Q with corresponding REG and REG' values
    for i in range(n):
        Z[2 * i] = Z[2 * i + 1] = z_values[i]
        Q[2 * i][0] = Q[2 * i + 1][0] = reg_values[i]
        Q[2 * i + 1][1] = reg_prime_values[i]
        if i > 0:
            Q[2 * i][1] = (Q[2 * i][0] - Q[2 * i - 1][0]) / (Z[2 * i] - Z[2 * i - 1])

    # Compute higher-order divided differences
    for i in range(2, 2 * n):
        for j in range(2, i + 1):
            Q[i][j] = (Q[i][j - 1] - Q[i - 1][j - 1]) / (Z[i] - Z[i - j])

    # Hermite polynomial evaluation at z
```

```
    result = Q[0][0]
    product = 1.0
    for i in range(1, 2 * n):
        product *= (z - Z[i - 1])
        result += Q[i][i] * product
    return result

# Function to calculate Hermite interpolation for multiple points
def calculate_hermite_for_multiple_points(z_points, z_values, reg_values, reg_prime_values=None):
    results = []
    for z in z_points:
        result = hermite_interpolation(z, z_values, reg_values, reg_prime_values)
        results.append(result)
    return results

# Spline interpolation
def construct_spline_coefficients_fixed(z_values, reg_values, reg_prime_values):
    n = len(z_values) - 1
    a = reg_values
    b = reg_prime_values
    c = np.zeros(n)
    d = np.zeros(n)
    # Calculate coefficients c and d for each interval
    for i in range(n):
        h = z_values[i + 1] - z_values[i]
        c[i] = (3 / h**2) * (reg_values[i + 1] - reg_values[i]) - (2 / h) * reg_prime_values[i] - (1 / h) * reg_prime_values[i
+ 1]
        d[i] = (-2 / h**3) * (reg_values[i + 1] - reg_values[i]) + (1 / h**2) * (reg_prime_values[i + 1] +
reg_prime_values[i])
    return a, b, c, d

# Function to evaluate the cubic spline at a point
def evaluate_spline(z, z_values, a, b, c, d):
    # Find the interval for z
    for i in range(len(z_values) - 1):
        if z_values[i] <= z <= z_values[i + 1]:
            h = z - z_values[i]
```

```
            return a[i] + b[i] * h + c[i] * h**2 + d[i] * h**3
    return None  # Out of bounds

# Function to calculate spline values for multiple points
def calculate_spline_for_multiple_points_fixed(z_points, z_values, reg_values, reg_prime_values):
    a, b, c, d = construct_spline_coefficients_fixed(z_values, reg_values, reg_prime_values)
    results = [evaluate_spline(z, z_values, a, b, c, d) for z in z_points]
    return results

# Interpolated values for Lagrange, Hermite, and Spline methods
L_values = calculate_lagrange_for_multiple_points(z_test_points, z_values, REG_z_values)
H_values_hermite = calculate_hermite_for_multiple_points(z_test_points, z_values_hermite,
REG_z_values_hermite, reg_prime_values_hermite)
S_values = calculate_spline_for_multiple_points_fixed(z_test_points, z_values, REG_z_values,
reg_prime_values)

# Calculate the differences between the methods
lagrange_hermite_diff = np.abs(np.array(L_values) - np.array(H_values_hermite))
lagrange_spline_diff = np.abs(np.array(L_values) - np.array(S_values))
hermite_spline_diff = np.abs(np.array(H_values_hermite) - np.array(S_values))

# Create two subplots: One for interpolation and one for differences
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 12))

# Plot the polynomial interpolations (Lagrange, Hermite, and Spline) on the first graph
ax1.plot(z_test_points, L_values, label='Lagrange Interpolation', color='blue')
ax1.plot(z_test_points, H_values_hermite, label='Hermite Interpolation', color='green')
ax1.plot(z_test_points, S_values, label='Spline Interpolation', color='orange')
ax1.set_title('Polynomial Interpolation (Lagrange, Hermite, and Spline)')
ax1.set_xlabel('Time (seconds)')
ax1.set_ylabel('Punches Thrown')
ax1.legend(loc='best')
ax1.grid(True)

# Plot the differences between methods on the second graph
ax2.plot(z_test_points, lagrange_hermite_diff, label='Lagrange vs Hermite Difference', color='blue')
ax2.plot(z_test_points, lagrange_spline_diff, label='Lagrange vs Spline Difference', color='green')
```

```
ax2.plot(z_test_points, hermite_spline_diff, label='Hermite vs Spline Difference', color='red')
ax2.set_title('Differences Between Lagrange, Hermite, and Spline Interpolation')
ax2.set_xlabel('Time (seconds)')
ax2.set_ylabel('Absolute Difference')
ax2.legend(loc='best')
ax2.grid(True)

# Show the plots
plt.tight_layout()
plt.show()
```

9. Reference

I. Class Notes.

II. Brin, Leon Q. Teatime Numerical Analysis. 3rd ed., 2021.

III. Wolfram Alpha

IV. Python Documentation: https://docs.python.org/3/

V. ChatGpt prompts to fill up the wordings and making paragraphs smoother from the results obtained.