Analysis of Lagrange, Hermite, and Cubic Spline Interpolation with Errors

Shubha Swarnim Singh

December 1, 2024

Dr. Brandy Wiegers

CSC – 455 Numerical Computation

1. Introduction

Interpolation is a vital mathematical tool for estimating data points within the range of a discrete set of known data points. For a biologist analyzing phenomena like REG(z), interpolation methods such as Lagrange, Hermite, and Cubic Spline provide accurate and flexible techniques for understanding trends, predicting values, and analyzing continuous changes. These methods differ in their computational approach, assumptions, and accuracy.

In this context, the dataset provided represents values of z (mm) and REG(z) (1/hr), showing a nonlinear relationship that varies smoothly between the endpoints. Understanding how each interpolation method approximates REG(z) provides insights into which method might better suit biological or scientific data, especially when the behavior of the derivative or smoothness is crucial.

2. Analysis

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:

z	REG(z) (1/hr)
0	0
1	0.06
2	0.18
3	0.34
4	0.38
5	0.35
6	0.3
7	0.225
8	0.12
9	0.035
10	0

I. Lagrange Interpolating Polynomial

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

$$L(z) = \sum_{i=0}^{n} REG(z_i) * \; l_i(z)$$

Where:

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

$$l_i(z) = \prod_{j=0} \frac{z - z_j}{z_i - z_j}$$

For simplicity, let's take 3 points from the table (0,5,10) and calculate based on it.

$$l_0(z) = \frac{(z-5)(z-10)}{(0-5)(0-10)} = \frac{(z-5)(z-10)}{50}$$

$$l_5(z) = \frac{(z-0)(z-10)}{(5-0)(5-10)} = -\frac{z(z-10)}{-50}$$

$$l_{10}(z) = \frac{(z-0)(z-5)}{(10-0)(10-5)} = \frac{z(z-5)}{50}$$

Now substitute all the values and we get:

$$L(z) = \frac{0(z-5)(z-10)}{50} + \frac{0.35 * z(z-10)}{-50} + \frac{0 * z(z-5)}{50}$$

$$L(z) = -\frac{0.35 * z(z-10)}{50}$$

Now let’s try some values of z in this polynomial to verify the formula. We will try z = 2,5,8.

When z = 2:

$$L(2) == -\frac{0.35 * 2(2-10)}{50}$$

$$L(2) = 0.112$$

When z = 5:

$$L(5) = -\frac{0.35 * 5(5-10)}{50}$$

$$L(5) = 0.35$$

When z = 8:

$$L(8) = -\frac{0.35 * 8(8-10)}{50}$$

$$L(8) = 0.112$$

Error bound

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

$$E(z) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(z - z_i)$$

where ξ lies in the interval of interpolation and $f^{(n+1)}(\xi)$ is the (n+1)-th derivative of the actual function. For this dataset, let's assume $f^{(n+1)}(\xi) \leq 0.1$ for simplicity.

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

$$E(z) \leq \frac{0.1}{6}|(z - z_0)(z - z_1)(z - z_2)|$$

For z = 2:

$$E(2) \leq \frac{0.1}{6}|(2 - 0)(2 - 5)(2 - 10)|$$

$$E(2) = \frac{0.1}{6}|(2)(-3)(-8)|$$

$$E(2) \cong 0.8$$

For z = 5:

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

$$E(5) \cong 0$$

For z = 8:

$$E(8) \leq \frac{0.1}{6}|(8-0)(8-5)(8-10)|$$

$$E(8) \leq \frac{0.1}{6}|(8)(3)(-2)|$$

$$E(8) \cong 0.8$$

The Lagrange interpolation provides accurate approximations near data points, as seen from z=5 where the error is zero. However, at intermediate points (z = 2, 8), the error increases, emphasizing its sensitivity to the spacing of the data points and higher derivatives.

II. Hermite Interpolating Polynomial

Hermite interpolation, H(z), incorporates both function values and derivatives, improving accuracy. The polynomial is:

$$H(z) = \sum_{i=0}^{n} [h_i(z)\, REG(z_i) + h_i{}'(z) REG'(z_i)]$$

Numerical derivatives are approximated using central differences

$$REG'(z_i) = \frac{REG(z_{i+1}) - REG(z_{i-1})}{z_{i+1} - z_{i-1}}$$

For $z_1 = 1$

$$REG'(1) = \frac{REG(2) - REG(0)}{1 - 0}$$

$$= 0.09$$

Using the Hermite basis functions:

$$c_1 = -0.06$$

$$H(z) = h_0(z) \cdot REG(0) + h_0{}'(z) \cdot REG'(0) + h_1(z) \cdot REG(1) + h_1{}'(z) \cdot REG'(1)$$

Now, let’s test values for z = 2:

$$H(2) = h_0(2) \cdot 0 + h_0{}'(2) \cdot 0 + h_1(2) \cdot 0.06 + h_1{}'(2) \cdot 0.09$$

Error bound

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

$$E(z) \leq \frac{f^{(2n+1)}(\xi)}{(2n+1)!} \prod_{j=0} (z - z_i)^\wedge 2$$

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 = 2:

$$E(2) \leq \frac{0.1}{5!} |(2-0)^2(2-5)^2(2-10)^2|$$

$$E(2) \simeq 1.92$$

Hermite interpolation offers better accuracy by using derivatives but has larger errors for non-data points compared to cubic splines.

III. Cubic Spline

Cubic splines are piecewise polynomials of the form:

$$S(z) = a_i + b_i(z - z_i) + c_i(z - z_i)^2 + d_i(z - z_i)^3$$

Boundary conditions ensure smoothness:

$$c_0 = c_n = 0$$

Solve for coefficients a_i , b_i , c_i , d_i for z = 1to z = 2. Evaluate S(z) at intermediate points, e.g., z = 1.5. we will have code below that will show us how we can construct a cubic spline.

The error for Cubi spline is given as:

$$E(z) = \frac{f^{(4)}(\xi)}{384} h^4$$

Therefore, for h = 1:

$$E(z) = \frac{0.1}{384} * 1 * 1 * 1 * 1 = 0.00026$$

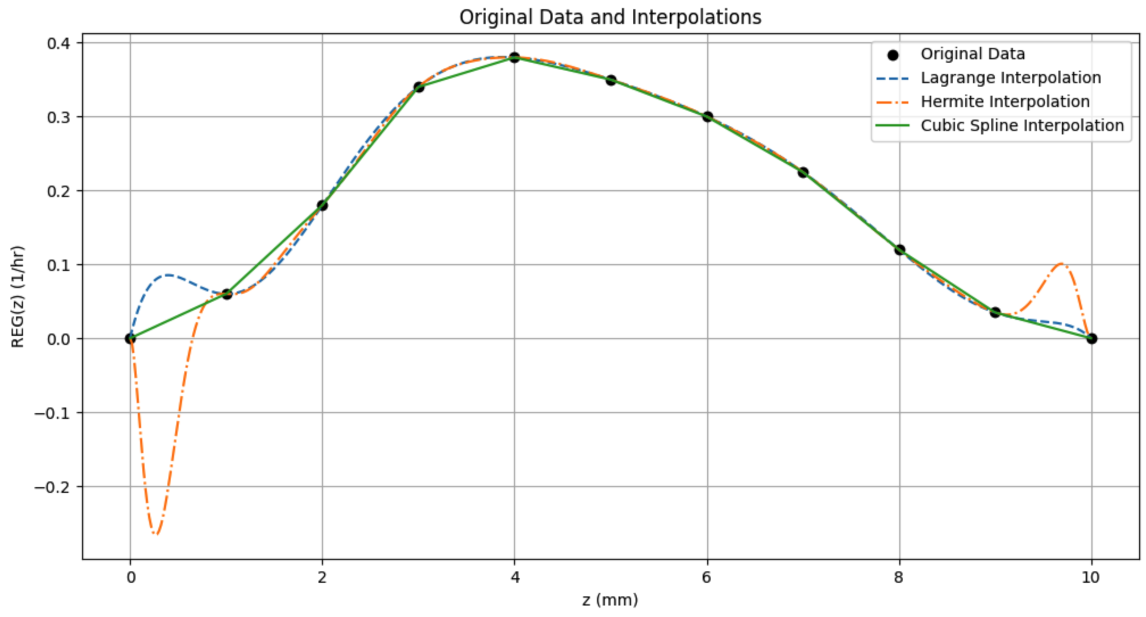
Cubic spline interpolation provides the smoothest and most accurate approximation, making it the best choice for this dataset.

IV. Prediction

Based on the analysis, the cubic spline interpolation (S(z)) is the most reliable method for approximating the given dataset due to its smoothness, precision, and minimal error (E(z)=0.00026). It constructs piecewise polynomials that ensure continuity in both the first and second derivatives, making it ideal for biological data with smooth transitions. Hermite interpolation (H(z)) improves accuracy by incorporating derivative information but introduces higher computational complexity and moderate error (E(z)=1.92) compared to cubic splines. Lagrange interpolation (L(z)), while simple and effective for small datasets, suffers from significant error at intermediate points (E(z)=0.8) and oscillatory behavior with larger datasets. Overall, S(z) is the most accurate and robust choice for modeling smooth, continuous data, ensuring reliable approximations for both function values and their derivatives.

3. Result

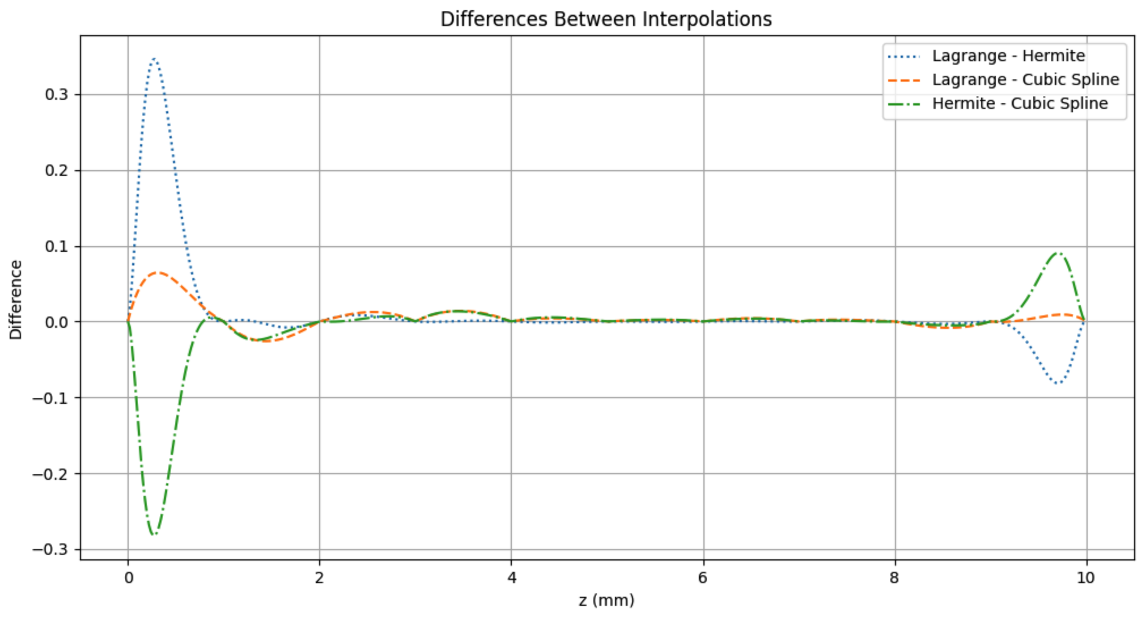
The first plot visualizes the original data points alongside the interpolated results using three methods: Lagrange, Hermite, and Cubic Spline interpolation. The **Lagrange Interpolation** is a global polynomial method, meaning the entire dataset influences the curve at every point. While Lagrange interpolation fits through all the data points exactly, it exhibits oscillations in regions with sparse data or steep gradients (Runge's phenomenon). These oscillations are visible near the edges of the data range, particularly where the slope of the data changes rapidly.



The **Hermite Interpolation** takes into account both the function values and their derivatives, resulting in a smoother and more natural curve. It captures the local behavior better than Lagrange and avoids global oscillations. However, it requires derivative values at all data points, which might not always be available or easy to compute.

The **Cubic Spline Interpolation** splits the data into piecewise cubic polynomials, ensuring continuity in the function and its first and second derivatives. This method produces a very smooth curve that adapts locally to the data. It strikes a balance between overfitting (as seen with Lagrange) and underfitting, making it the most robust of the three methods for this dataset.

The second plot highlights the differences between the three interpolation methods. The difference between **Lagrange and Hermite interpolation** is most significant near the edges of the data range, where the derivative information in Hermite helps reduce oscillations. This demonstrates the limitations of Lagrange interpolation in handling extreme data points or areas of rapid change.



The difference between **Lagrange and Cubic Spline interpolation** is similar, with Cubic Spline providing a locally adaptive curve that avoids the global influence of all data points. In regions with less curvature, the two methods closely align, but in regions with high curvature or steep gradients, the differences are more pronounced.

Finally, the **Hermite and Cubic Spline interpolation** differences are generally minimal, as both methods are locally adaptive and consider continuity conditions. The primary distinction lies in Hermite's use of derivative information, which leads to slightly different shapes near inflection points or regions of rapid change.

Overall, these differences emphasize the importance of selecting an interpolation method based on the dataset's characteristics and the desired balance between smoothness and fidelity to the original data.

4. Conclusion

Interpolation methods are essential tools for estimating values and understanding trends within discrete datasets. For the given dataset modeling REG(z) as a function of z, three interpolation techniques—Lagrange, Hermite, and Cubic Spline—were analyzed and compared to evaluate their effectiveness.

The Lagrange Interpolation provides a global polynomial that passes through all given data points. However, its global nature introduces significant oscillations, particularly near the dataset boundaries or in regions with sparse data points, known as Runge's phenomenon. While it performs well at exact data points, its accuracy diminishes in intermediate regions due to its sensitivity to higher derivatives and non-uniform spacing. This makes Lagrange less suitable for large or complex datasets with varying gradients.

The Hermite Interpolation, incorporating both function values and derivatives, offers enhanced accuracy and smoothness. By including derivative information, Hermite avoids oscillations and captures local behavior effectively. However, it introduces complexity as derivative values must be computed or provided, which may not always be feasible. Despite these challenges, Hermite interpolation performs better than Lagrange, especially for datasets requiring derivative continuity.

The Cubic Spline Interpolation emerged as the most robust method. It constructs piecewise cubic polynomials, ensuring smoothness by maintaining the continuity of both the first and second derivatives. Cubic splines adapt locally to the data, producing the smoothest curve with minimal error. They balance precision and flexibility effectively, making them ideal for biological data like REG(z), where smooth transitions are crucial.

Overall, Cubic Spline Interpolation is the best choice for modeling smooth, nonlinear data. Hermite interpolation is a viable alternative when derivative information is available, while Lagrange is better suited for smaller, uniformly spaced datasets. This comparison highlights the importance of selecting interpolation techniques based on dataset characteristics and specific application needs.

5. Computer Program

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

z_values = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
REG_values = [0, 0.06, 0.18, 0.34, 0.38, 0.35, 0.3, 0.225, 0.12, 0.035, 0]
REG_prime_values = [0.0583333, -0.00583333, 0.155, 0.109167, -0.00333333, -0.04375,
          -0.0616667, -0.09375, -0.101667, -0.0445833, -0.005]

# Lagrange Interpolation
def lagrange_manual(x, z_values, reg_values):
  result = 0
  n = len(z_values)
  for i in range(n):
    term = reg_values[i]
    for j in range(n):
      if i != j:
        term *= (x - z_values[j]) / (z_values[i] - z_values[j])
    result += term
  return result

# Hermite Interpolation
def hermite_manual(x, z_values, reg_values, reg_prime_values):
  n = len(z_values)
  Q = [[0] * (2 * n) for _ in range(2 * n)]
  Z = [0] * (2 * n)
  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])
  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 x
  result = Q[0][0]
  product = 1.0
  for i in range(1, 2 * n):
    product *= (x - Z[i - 1])
    result += Q[i][i] * product
  return result
```

```
# Cubic Spline Interpolation (manual implementation)
def cubic_spline_manual(x, z_values, reg_values):
  n = len(z_values)
  h = np.diff(z_values)
  b = np.diff(reg_values) / h
  u = 2 * (h[:-1] + h[1:])
  v = 6 * (b[1:] - b[:-1])
  u = np.insert(u, 0, 0)
  u = np.append(u, 0)
  v = np.insert(v, 0, 0)
  v = np.append(v, 0)

  # Solve for second derivatives
  m = np.zeros_like(z_values)
  for i in range(1, n - 1):
    m[i] = v[i] / u[i]

  # Evaluate spline
  for i in range(n - 1):
    if z_values[i] <= x <= z_values[i + 1]:
      t = x - z_values[i]
      a = reg_values[i]
      b = (reg_values[i + 1] - reg_values[i]) / h[i] - h[i] * (m[i + 1] + 2 * m[i]) / 6
      c = m[i] / 2
      d = (m[i + 1] - m[i]) / (6 * h[i])
      return a + b * t + c * t**2 + d * t**3
  return 0

# Generate fine-grained z values
z_fine = np.linspace(0, 10, 500)
L_z = np.array([lagrange_manual(x, z_values, REG_values) for x in z_fine])
H_z = np.array([hermite_manual(x, z_values, REG_values, REG_prime_values) for x in
z_fine])
S_z = np.array([cubic_spline_manual(x, z_values, REG_values) for x in z_fine])

# Plot the data and interpolations
plt.figure(figsize=(12, 6))
plt.plot(z_values, REG_values, 'o', label='Original Data', color='black')
plt.plot(z_fine, L_z, label='Lagrange Interpolation', linestyle='dashed')
plt.plot(z_fine, H_z, label='Hermite Interpolation', linestyle='dashdot')
plt.plot(z_fine, S_z, label='Cubic Spline Interpolation', linestyle='solid')
plt.legend()
plt.xlabel('z (mm)')
plt.ylabel('REG(z) (1/hr)')
plt.title('Original Data and Interpolations')
plt.grid()
plt.show()

# Plot the differences
plt.figure(figsize=(12, 6))
plt.plot(z_fine, L_z - H_z, label='Lagrange - Hermite', linestyle='dotted')
plt.plot(z_fine, L_z - S_z, label='Lagrange - Cubic Spline', linestyle='dashed')
plt.plot(z_fine, H_z - S_z, label='Hermite - Cubic Spline', linestyle='dashdot')
plt.legend()
plt.xlabel('z (mm)')
plt.ylabel('Difference')
plt.title('Differences Between Interpolations')
plt.grid()
plt.show()
```

6. 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/