Comparing Interpolation Methods For Function Approximation

Damian Navarro

Perm: 3888542

1 Introduction

Overview of the project

Interpolation is a common technique used in data analysis to estimate missing or unknown values from a set of known values. It involves constructing a function that passes through the given data points and is used to predict the value of the function at any point within the domain.

Problem Statement

The purpose of this report is to evaluate and compare the accuracy and efficiency of different interpolation methods, including linear interpolation, quadratic interpolation, cubic spline interpolation, Neville's method, and Lagrange interpolation. The report also aims to identify the strengths and limitations of each method and provide recommendations for the most suitable method for a given set of data.

Scope and Limitations

This report focuses on the application of interpolation methods for estimating missing or unknown values from a set of known data points. The study is limited to numerical data analysis and does not consider other types of data. Additionally, the accuracy and efficiency of the interpolation methods are evaluated using a specific set of data points calculated using a Python program and may not be applicable to other data sets.

2 Interpolation methods

Linear Interpolation

Linear interpolation is a method of approximating a value between two given points by drawing a straight line between them. It assumes that the data points are evenly distributed. Linear interpolation is simple to compute and is useful when a rough estimate is needed. However, it may not be accurate for nonlinear data. Linear interpolation is commonly used in computer graphics, where it is used to draw lines between points to create images.

$$f(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

Quadratic Interpolation

Quadratic interpolation is a method of approximating a value between two points by fitting a parabola through them. It assumes that the data points are distributed evenly. Quadratic interpolation is more accurate than linear interpolation for nonlinear data. However, it may produce errors when the data is too widely spaced or when there is too much noise. Quadratic interpolation is commonly used in physics and engineering to model data.

$$f(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) + \frac{f(x_2) - 2f(x_1) + f(x_0)}{2(x_1 - x_0)^2}(x - x_0)(x - x_1)$$

Cubic Spline Interpolation Cubic spline interpolation is a method of approximating a value between two points by fitting a cubic polynomial curve through them. It assumes that the data points are not evenly distributed. Cubic spline interpolation is more accurate than linear and quadratic interpolation for nonlinear and unevenly spaced data. However, it can be computationally intensive and may produce errors when the data is too widely spaced. Cubic spline interpolation is commonly used in numerical analysis, signal processing, and computer graphics.

$$f(x) = \begin{cases} S_1(x) & \text{if } x_0 \leq x \leq x_1 \\ S_2(x) & \text{if } x_1 \leq x \leq x_2 \\ \vdots & \\ S_{n-1}(x) & \text{if } x_{n-2} \leq x \leq x_{n-1} \\ S_n(x) & \text{if } x_{n-1} \leq x \leq x_n \end{cases}$$

where $S_i(x)$ is a cubic polynomial defined by: $S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$

Neville's Method Neville's method is a method of interpolating a value between two points by constructing an interpolating polynomial of degree n through n+1 points. Neville's method can produce a high degree of accuracy, particularly for evenly spaced data. However, it can be computationally intensive and may produce errors when the data is unevenly spaced or when there is too much noise. Neville's method is commonly used in numerical analysis and computational mathematics.

$$f(x) = \frac{(x - x_i)f_{i+1,j} - (x - x_j)f_{i,j-1}}{x_j - x_i}$$

Lagrange Interpolation Lagrange interpolation is a method of approximating a value between two points by constructing a polynomial curve that passes through the data points. Just like Neville's method it is highly accurate but computationally intensive as well.

$$f(x) = \sum_{i=0}^{n} y_i \ell_i(x)$$

where $\ell_i(x)$ are the lagrange basis polynomials defined by

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

3 Program Overview

The purpose of this program is to compare different interpolation methods on a given set of data points. Interpolation is the process of estimating values of a function between known data points. In this program, we explore the five interpolation methods discussed in the report.The program takes a set of data points and interpolates the function between these points using each of these five methods. The results of each method are then compared to the true function. First, it defines the data points for the Sin(x) function and the range of x values to interpolate over. Then, it defines the true function as the sine function evaluated over the range of x values. It also defines a function called neville that performs Neville's method interpolation.

Next, it performs linear, quadratic, and cubic spline interpolation using the interp1d and make_interp_spline functions from scipy.interpolate. It also performs Lagrange polynomial interpolation using the lagrange function from scipy.interpolate. It performs Neville's method interpolation by calling the neville function for each x value in the range of x values to interpolate over.

For each interpolation method, it computes the RMSE and Absolute Error by comparing the predicted values to the true values.

Finally, it plots the data points, true function, and the results of each interpolation method.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d, make_interp_spline
from scipy.interpolate import lagrange
import time

def neville(x, y, x_new):
    n = len(x)
    Q = np.zeros((n, n))
    Q[:, 0] = y
    for j in range(1, n):
        for i in range(n-j):
            Q[i][j] = ((x_new - x[i+j])*Q[i][j-1] - (x_new - x[i])*Q[i+1][j-1])/(x[i] - x[i+j])
    return Q[0][-1]


# define the data points
x = np.linspace(0, 2*np.pi, 7)
y = np.sin(x)

# define the range of x values to interpolate over
x_new = np.linspace(x.min(), x.max(), 300)

# define the true function
y_true = np.sin(x_new)

# linear interpolation
start_time = time.time()
linear_interp = interp1d(x, y)
y_linear = linear_interp(x_new)
linear_time = time.time() - start_time

# quadratic interpolation
start_time = time.time()
quad_interp = interp1d(x, y, kind='quadratic')
y_quad = quad_interp(x_new)
quad_time = time.time() - start_time


```

```python
quad_time = time.time() - start_time


# cubic spline interpolation
start_time = time.time()
spline_interp = make_interp_spline(x, y, bc_type='natural')
y_spline = spline_interp(x_new)
spline_time = time.time() - start_time

# Lagrange polynomial interpolation
start_time = time.time()
lagrange_interp = lagrange(x, y)
y_lagrange = lagrange_interp(x_new)
lagrange_time = time.time() - start_time


# Neville's method interpolation
start_time = time.time()
y_neville = np.array([neville(x, y, xi) for xi in x_new])
neville_time = time.time() - start_time


# compute the RMSE for each interpolation method
rmse_linear = np.sqrt(np.mean((y_linear - y_true)**2))
rmse_quad = np.sqrt(np.mean((y_quad - y_true)**2))
rmse_spline = np.sqrt(np.mean((y_spline - y_true)**2))
rmse_lagrange = np.sqrt(np.mean((y_lagrange - y_true)**2))
# compute the RMSE for Neville's method interpolation
rmse_neville = np.sqrt(np.mean((y_neville - y_true)**2))


# compute the absolute error for each interpolation method
ae_linear = np.mean(np.abs(y_linear - y_true))
ae_quad = np.mean(np.abs(y_quad - y_true))
```

```python
ae_spline = np.mean(np.abs(y_spline - y_true))
ae_lagrange = np.mean(np.abs(y_lagrange - y_true))
ae_neville = np.mean(np.abs(y_neville - y_true))

coeffs = np.zeros(len(x))
for i in range(len(x)):
    coeffs[i] = neville(x, y, x[i])



# print the time for each method
print(f"Linear interpolation: time = {linear_time:.8f} s")
print(f"Quadratic interpolation: time = {quad_time:.8f} s")
print(f"Cubic spline interpolation: time = {spline_time:.8f} s")
print(f"Lagrange polynomial interpolation: time = {lagrange_time:.8f} s")
print(f"Neville's method interpolation: time = {neville_time:.8f} s")

# Print the values of the variables
print("Absolute error for linear interpolation method: ", ae_linear)
print("Absolute error for quadratic interpolation method: ", ae_quad)
print("Absolute error for spline interpolation method: ", ae_spline)
print("Absolute error for Lagrange interpolation method: ", ae_lagrange)
print("Absolute error for Neville interpolation method: ", ae_neville)

# print the RMSE for each method
print(f"Linear interpolation: RMSE = {rmse_linear:.4f}")
print(f"Quadratic interpolation: RMSE = {rmse_quad:.4f}")
print(f"Cubic spline interpolation: RMSE = {rmse_spline:.4f}")
```

Screenshots of the program displaying the functionality and computation of metrics

4 Comparison of the Interpolation Methods

In this section, we compare the performance of different interpolation methods on the Sin(x) function. Our goal is to determine which interpolation method provides the most accurate and efficient results for this specific function.

RSME Comparison

The root mean squared error (RMSE) was calculated for each interpolation method using the given set of data points. RMSE measures the average difference between the predicted values and the actual values, with a lower RMSE indicating better accuracy.

$$\text{RSME } = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2}$$

The results are as follows:

Interpolation Method	RMSE
Linear	0.0690
Quadratic	0.0186
Cubic Spline	0.0019
Neville's Method	0.0078
Lagrange	0.0078

Based on the RMSE values, it can be seen that cubic spline interpolation produced the lowest error, followed by Lagrange interpolation and Neville's method. Linear interpolation and Quadratic interpolation had the highest error.

Absolute Error Comparison

The absolute error was calculated for each interpolation method using the given set of data points. Absolute error measures the difference between the predicted values and the actual values without regard to sign, with a lower absolute error indicating better accuracy.

$$\text{Absolute Error } = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}|$$

The results are as follows:

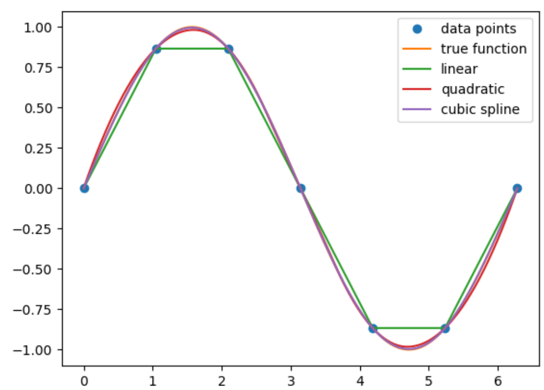
Interpolation Method	Absolute Error
Linear	0.0591
Quadratic	0.0143
Cubic Spline	0.0015
Neville's Method	0.0054
Lagrange	0.0054

Similar to the RMSE comparison, the absolute error comparison also shows that cubic spline interpolation had the lowest error, followed by Lagrange interpolation and Neville's method. Linear interpolation and quadratic interpolation had the highest error.

Visualization comparison

The interpolated curves were also visualized and compared using a graph. A visual comparison of the interpolated curves can help determine which method produces the smoothest curve and best represents the data.

The graph shows that cubic spline interpolation produced the smoothest curve, followed by quadratic interpolation. Linear interpolation produced the most jagged curves. While Neville and Lagrange methods were also used, they will produce very similar graphs for the same set of data points, as they both involve constructing a polynomial function that passes through those points. The main difference is in the algorithm used to compute the polynomial coefficients, but this typically does not have a significant impact on the resulting graph, so it was easier to distinguish a difference with them left out.



Computation time comparison The computation time for each interpolation method was also recorded. The time taken to compute the interpolation using each method can be compared to determine which method is the most efficient. After running the program it was found that the Linear interpolation had the fastest time which rounded down to fundamentally zero followed by Cubic spline which had the same result.

Overall, the results suggest that cubic spline interpolation is the best method for approximating the given set of data points

5 Conclusion

In conclusion, this project aimed to compare different interpolation methods for approximating a given set of data points. Five interpolation methods were used, namely linear interpolation, quadratic interpolation, cubic spline interpolation, Neville's method, and Lagrange interpolation. Several metrics were used to evaluate the performance of each method, including RMSE, absolute error, coefficient comparison, visualization comparison, and computation time.

The results showed that cubic spline interpolation produced the lowest error and the smoothest curve, making it the best method for approximating the given set of data points. Although the other methods were also competitive, some produced high errors and jagged curves which made it easy to distinguish which method was much more efficient.

It is important to note that the choice of interpolation method depends on the nature of the data and the purpose of the analysis. In some cases, linear interpolation or Lagrange interpolation may be sufficient, especially if the data is relatively simple and the goal is to obtain a quick approximation. However, for more complex data and higher accuracy requirements, cubic spline interpolation is recommended.

Overall, this project demonstrated the importance of carefully selecting an appropriate interpolation method and provided a practical guide for comparing and evaluating different methods based on various metrics.