Lab Manual for Numerical Analysis

Lab No. 5

INTERPOLATION - NEWTON'S FORWARD & BACKWARD INTERPOLATION FORMULA

BAHRIA UNIVERSITY KARACHI CAMPUS
Department of Software Engineering

NUMERICAL ANALYSIS

LAB EXPERIMENT # 5

Interpolation - Newton's Forward & Backward Interpolation Formula

OBJECTIVE:

This lab aims to introduce students to interpolation principles, with a focus on Newton's Forward and Backward Interpolation Formulas in order to equip them with practical skills for estimating intermediate values within scientific and engineering contexts.

Introduction

Interpolation stands as a fundamental and versatile mathematical and computational technique that finds significant application in numerous scientific and engineering disciplines. At its core, interpolation entails the art of fitting data points to construct a continuous representation of a function. This process serves a twofold purpose: firstly, it allows for the estimation of values within the confines of a **discrete data** set, and secondly, it seeks to establish a **function** that precisely traverses a given set of data points, typically represented as pairs of **(x, y)** coordinates.

Interpolation operates as a vital bridge between known **data points**, facilitating the precise prediction and computation of values for intermediate data points. Its practicality is evident across diverse domains; it plays a central role in forecasting variables like noise levels, rainfall patterns, elevation, and other phenomena where data collection occurs at discrete intervals. In the domains of science and engineering, interpolation's significance shines even brighter, especially when working with real-world experimental data.

Within the sphere of numerical analysis, interpolation takes on a pivotal role by addressing the unique challenges posed by discrete data. This technique empowers us to derive a simplified function that faithfully replicates the provided data points, thus enabling the precise estimation of values situated between these known data points. Moreover, interpolation offers the advantage of simplifying intricate functions by judiciously sampling data points, rendering it a versatile and indispensable tool for both data analysis and modeling.

For instance, consider an application within environmental science. If we have been recording specific atmospheric conditions at designated time intervals, interpolation equips us to accurately estimate environmental parameters for any intermediate time, significantly enhancing our capacity to make informed decisions and predictions.

Interpolation involves a range of methods and formulas essential for estimating values between known data points. These methods include **Lagrange Interpolation**, **Hermite Interpolation**, **Spline Interpolation**, **Barycentric Interpolation**, and more, each tailored to address specific interpolation scenarios.

In this lab, we place our primary emphasis on two specific interpolation techniques renowned for their versatility and practicality in the field of numerical analysis: **Newton's Forward** and **Backward Interpolation** Formulas. These methods have gained prominence, particularly when data points are uniformly spaced, and are known for their efficiency and effectiveness in estimating intermediate values within datasets.

Implementation of Newton's Forward and Backward Interpolation Formula

This section will provide a detailed discussion and implementation of Newton's Forward and Backward Interpolation Formulas, offering a comprehensive understanding of these interpolation techniques and their practical applications.

a. Newton's Forward Interpolation Formula

Newton's Forward Interpolation Formula is a powerful mathematical method employed for estimating values between known data points, with a particular focus on datasets where the independent variable values are uniformly spaced. This technique is instrumental in a variety of fields, including engineering, physics, computer science, and data analysis.

The foundation of Newton's Forward Interpolation lies in constructing a **polynomial** that precisely passes through the given data points. This polynomial interpolation method allows for the accurate estimation of values at points within the dataset, and its effectiveness is especially pronounced when dealing with evenly spaced data points.

By iteratively calculating finite differences between adjacent data points and building a polynomial based on these differences, Newton's Forward Interpolation provides a systematic and reliable approach to interpolate data.

The **formula** for Newton's Forward Interpolation is:

$$f(a+hu) = f(a) + u\Delta f(a) + \frac{u(u-1)}{2!}\Delta^2 f(a) + \cdots$$

$$+ \frac{u(u-1)(u-2)\ldots(u-n+1)}{n!}\Delta^n f(a)$$

Where:

$$u = \frac{x-a}{h}$$

Implementation in Python

```
def calculate_u(u, n):
    # Calculate the value of u raised to the power of n
    temp = u
    for i in range(1, n):
        temp = temp * (u - i)
    return temp


def factorial(n):
    # Calculate the factorial of a given number n
    f = 1
    for i in range(2, n + 1):
        f *= i
    return f

def display_forward_difference_table(x, y, n):
    # Display forward difference table for better visualization
    for i in range(n):
        print(x[i], end="\t")
        for j in range(n - i):
            print(y[i][j], end="\t")
        print("")


def interpolate(x, y, n, value):
    # Perform interpolation to estimate the value at a given point
    sum = y[0][0]
    u = (value - x[0]) / (x[1] - x[0])
    for i in range(1, n):
        sum = sum + (calculate_u(u, i) * y[0][i]) / factorial(i)
    return round(sum, 6)


n = int(input("Enter the number of data points: "))
x = []
y = []

# Input the data points
for i in range(n):
    x_val = float(input(f"Enter x[{i}]: "))
    y_val = float(input(f"Enter y[{i}]: "))
    x.append(x_val)
    y.append([0] * n)
    y[i][0] = y_val
```

```
Calculate the forward difference table
for i in range(1, n):
    for j in range(n - i):
        y[j][i] = y[j + 1][i - 1] - y[j][i - 1]

print("\nForward Difference Table:")
display_forward_difference_table(x, y, n)

value = float(input("\nEnter the value to interpolate at: "))
result = interpolate(x, y, n, value)
print(f"\nInterpolated value at {value} is approximately {result}")
```

b. Newton's Backward Interpolation Formula

Newton's backward interpolation formula is a pivotal numerical method in the realm of numerical analysis and mathematics. This method is employed to estimate a function's value at a specific point within a defined range based on a given set of data points. Its efficacy is particularly pronounced when the independent variable values are arranged in descending order. Newton's backward interpolation provides a valuable means of **approximating function** values within the data range, proving exceptionally useful in scenarios where precise function values are not readily accessible. At its core, this technique leverages polynomial interpolation principles. It furnishes a mathematical framework for constructing an interpolating polynomial—a polynomial function that precisely passes through a provided set of data points.

In the context of backward interpolation, the formula enables the estimation of a function's value at an intermediary point by working backward from the data point closest to and greater than the point of interest. The formula expresses the interpolated value as a sum of terms, with each term contingent on the differences between the data points and their corresponding function values. Newton's backward interpolation formula thus stands as a powerful tool for approximating function values within the given data range, contributing significantly to mathematical modeling and data analysis endeavors.

The formula for Newton's Backward Interpolation is:

$$f(a+nh+hu) = f(a+nh) + u \nabla f(a+nh) + \frac{u(u+1)}{2!} \nabla^2 f(a+nh) + \cdots$$

$$+ \frac{u(u-1)(u-2)\ldots(u-n+1)}{n!} \nabla^n f(a+nh)$$

Where again:

$$u = \frac{x-a}{h}$$

Implementation in Python

```
# Function to calculate 'u' as per the formula
def calculate u(u, n):
    result = u
    for i in range(1, n):
        result *= (u + i)
    return result

# Function to compute the factorial of 'n'
def factorial(n):
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result

# User input: Number of data points
n = int(input("Enter the number of data points: "))

# Initialize lists to store data
x = []
y = []


# Input data points
for i in range(n):
    x_val = float(input(f"Enter x[{i}]: "))
    y_val = float(input(f"Enter y[{i}]: "))
    x.append(x_val)
    y.append([0] * n)  # Initialize difference table
    y[i][0] = y_val

# Calculate the backward difference table
for i in range(1, n):
    for j in range(n - 1, i - 1, -1):
        y[j][i] = y[j][i - 1] - y[j - 1][i - 1]


# Display the backward difference table
for i in range(n):
    for j in range(i + 1):
        print(y[i][j], end="\t")
    print()

# User input: Value to interpolate at
value = float(input("Enter the value to interpole: "))
```

```
# Initialize 'u' and 'result'
result = y[n - 1][0]
u = (value - x[n - 1]) / (x[1] - x[0])

# Calculate the interpolated value using Newton's backward
interpolation
for i in range(1, n):
    result += (calculate_u(u, i) * y[n - 1][i]) / factorial(i)

# Display the interpolated value
print(f"\nThe interpolated value at year {value} is approximately
{result}")
```

Lab Tasks

1. Write a python program that can find the value of tan(0.12) using Newton's forward interpolation formula, with considering to the following data points:

x	0.10	0.15	0.20	0.25	0.30
y = tan(x)	0.1003	0.1511	0.2027	0.2553	0.3093

2. Write a python program implementing the backward interpolation formula that considers the following data points, and
    a. Find the value of y at x = 300
    b. Find the value of y at any given user input

x	50	100	150	200	250
y	618	724	805	906	1032