Lab Manual for Numerical Analysis

Lab No. 6

INTERPOLATION - NEWTON’S DIVIDED DIFFERENCE FORMULA & LAGRANGE’S INTERPOLATION

BAHRIA UNIVERSITY KARACHI CAMPUS
Department of Software Engineering

NUMERICAL ANALYSIS

LAB EXPERIMENT # 6

Interpolation - Newton’s Forward & Backward Interpolation Formula & Lagrange’s Interpolation

OBJECTIVE:

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

Introduction

Interpolation, as discussed in previous lab a fundamental concept in mathematics and data analysis, holds a pivotal role in estimating values between established data points. It is essential for making informed decisions and predictions, and we've previously explored its core principles in our academic pursuits.

In our prior chapter, we have extensively delved into two key interpolation formulas that are integral to our understanding: Newton's Backward Formula and Newton's Forward Formula. These formulas prove to be invaluable tools for approximating intermediate values within datasets. Newton's Backward Formula is particularly advantageous when we encounter situations marked by decreasing intervals, while Newton's Forward Formula shines when we work with increasing intervals. Collectively, these formulas provide us with a versatile and robust toolkit for performing interpolation with precision and efficiency. Through these formulas, we gain the ability to make data-driven decisions and predictions, enabling us to navigate the intricate terrain of data analysis and mathematical modeling with confidence and accuracy.

In this lab, we will further enhance our understanding of interpolation by exploring two additional techniques: Newton’s Divided Difference Formula and Lagrange’s Interpolation

Implementation of Newton’s Divided Difference Formula and Lagrange’s Interpolation

This section will provide a detailed discussion and implementation of **Newton's Divided Difference Formula** and **Lagrange Interpolation**, offering a comprehensive understanding of these interpolation techniques and their practical applications.

a. Newton's Divided Difference Formula

Newton's divided difference interpolation formula, is a method for approximating a **polynomial** that passes through a set of given data points, particularly when the intervals between the data points are **not uniform**. This interpolation technique is named after Sir Isaac Newton and is widely used in numerical analysis and mathematics.

The formula allows for the estimation of function values at intermediate points between the data points. It is constructed using the divided difference coefficients, which are calculated recursively based on the function values at the data points. The polynomial created with these coefficients can be used for interpolation to predict function values at specific locations within the range of the data.

The formula for Newton's divided difference Interpolation is:

$$\boxed{\begin{gathered} f(x) = f(x_0) + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + \cdots \\ + (x - x_0)(x - x_1) \dots . (x - x_{k-1})f[x_0, x_1, x_2, \dots x_k] \end{gathered}}$$

Where;

- $f[x_o, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$
- $f[x_o, x_1, x_2] = \frac{f(x_1, x_2) - f(x_0, x_1)}{x_2 - x_0}$

and so on.

Implementation in Python

```python
# Function to calculate the product term
def calculate_product_term(i, value, x):
    product = 1
    for j in range(i):
        product = product * (value - x[j])
    return product

# Function for calculating the divided difference table
def calculate_divided_difference_table(x, y, n):
    for i in range(1, n):
        for j in range(n - i):
            y[j][i] = ((y[j][i-1] - y[j+1][i-1])/(x[j] - x[i+j]))
    return y

# Function for applying Newton's divided difference formula
def apply_newton_formula(value, x, y, n):
    result = y[0][0]
    for i in range(1, n):
        result=result+(calculate_product_term(i,value, x)*y[0][i])
    return result
```

```
# Function for displaying the divided difference table
def display_divided_difference_table(y, n):
    for i in range(n):
        for j in range(n - i):
            print(round(y[i][j], 4), "\t", end=" ")
        print("")

# User input: Number of inputs
n = int(input("Enter the number of inputs: "))
x = []
y = [[0 for i in range(n)] for j in range(n)]

# 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[i][0] = y_val

# Calculate the divided difference table
y = calculate_divided_difference_table(x, y, n)

# Display the divided difference table
print("\nDivided Difference Table:")
display_divided_difference_table(y, n)

# User input: Value to interpolate
value = float(input("\nEnter the value to interpolate: "))

# Calculate and display the interpolated value
interpolated_value = apply_newton_formula(value, x, y, n)
print("\nInterpolated value at", value, "is approximately",
round(interpolated_value, 4))
```

b. Lagrange Interpolation

Lagrange Interpolation stands as a valuable and versatile technique that plays a pivotal role in mathematical modeling, engineering, and data analysis. Its significance lies in its ability to estimate the value of a function at a specific point, even when the function's form remains unknown. This interpolation method leverages known data points, typically represented as pairs like **(x1, y1)** and **(x2, y2),** as its foundation.

By applying the Lagrange Interpolation Formula, it can effectively calculate the value of y at any arbitrary point, denoted as x = a. The core of this technique lies in the creation of a Lagrange Polynomial, an **nth-degree** expression of the underlying function f(x) where n i.e. degree of polynomial equals the total number of data points. This polynomial's flexibility and precision empower it to bridge the gaps between data points, enabling informed estimations.

Lagrange Interpolation proves its worth, particularly in scenarios where the function's explicit form remains elusive, offering a reliable means to navigate the intricate landscape of data analysis and mathematical modeling, making it an indispensable tool in various domains.

Lagrange First Order Interpolation Formula (for n = 1) is:

$$\boxed{f(x) = \frac{(x - x_1)}{(x_0 - x_1)} \times (y_0) + \frac{(x - x_0)}{(x_1 - x_0)} \times y_1}$$

and, **Lagrange First Order Interpolation** Formula (for n = 2) is:

$$\boxed{f(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} \times (y_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \times y_1 + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \times y_2}$$

Just like this, formula for Lagrange Interpolation at **nth order** can be defined as:

$$\boxed{\begin{aligned} f(x) = {} & \frac{(x - x_1)(x - x_2) \dots (x - x_n)}{(x_0 - x_1)(x_0 - x_2) \dots (x_0 - x_n)} \times (y_0) + \frac{(x - x_0)(x - x_2) \dots (x - x_n)}{(x_1 - x_0)(x_1 - x_2) \dots (x_1 - x_n)} \times y_1 + \cdots \\ & + \frac{(x - x_0)(x - x_1) \dots (x - x_n - 1)}{(x_n - x_0)(x_n - x_1) \dots (x_n - x_n - 1)} \times y_n \end{aligned}}$$

Implementation in Python

```
# Define a class to represent data points with x and y values
class DataPoint:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# Function for Lagrange Interpolation
def lagrange_interpolation(data_points, x_value):
    result = 0.0

    # Iterate through each data point
    for i in range(len(data_points)):
        term = data_points[i].y

        # Calculate the Lagrange term for the current data point
        for j in range(len(data_points)):
            if j != i:
                term *= (x_value - data_points[j].x) /
                        (data_points[i].x - data_points[j].x)
        # Add the term to the result
        result += term
    return result
```

```
if __name__ == "__main__":
    data_points = []

    # Input the number of known data points
    n = int(input("Enter the number of known data points: "))

    # Input x and y values for each data point
    for i in range(n):
        x = float(input(f"Enter x{i + 1}: "))
        y = float(input(f"Enter y{i + 1}: "))
        data_points.append(DataPoint(x, y))

    # Input the x value for interpolation
    x_value = float(input("Enter the x value for interpolation: "))

    # Calculate and print the interpolated value
    interpolated_value = lagrange_interpolation(data_points, x_value)
    print(f"Interpolated value at {x_value} is:{interpolated_value}")
```

Lab Tasks

1. Write a python program implementing the newton's difference formula that considers the following data points, and
   a. Find the value of y at x = 301
   b. Find the value of y at any given user input

x	300	304	305	307
y	2.4771	2.4829	2.4843	2.4871

2. Write a Python program that implements Newton's difference formula, reads the CSV file 'salary_data,' and considers 'years of experience' as x and 'salary' as y. Then, upon user input for a specific number of years of experience, it provides an output of the salary they would receive.

   (Note: the file has been uploaded in miscellaneous section)

3. Write a python program implementing the Lagrange interpolation formula that considers the following data points, and find the value of y at x = 7

x	1	2	3	5
y	10	4	4	7