Lab Manual for Numerical Analysis

Lab No. 7

DIFFERENTIATION- NEWTON'S FORWARD & NEWTON'S BACKWARD FORMULA

BAHRIA UNIVERSITY KARACHI CAMPUS
Department of Software Engineering

NUMERICAL ANALYSIS

LAB EXPERIMENT # 7

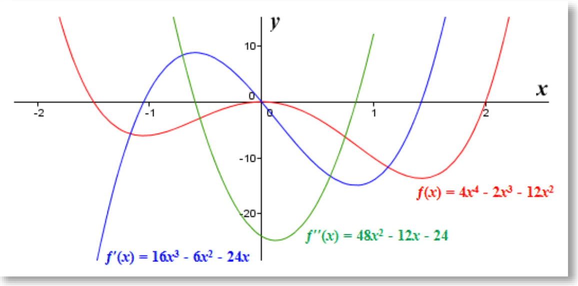
Differentiation- Newton’s Forward and Newton’s Backward Formula

OBJECTIVE:

This lab aims to introduce students to the principles of numerical differentiation using Newton's Forward and Newton's Backward Formulas. The focus will be on equipping students with practical skills for estimating derivatives of functions within scientific and engineering contexts.

Introduction

Differentiation is a fundamental mathematical concept that involves finding the rate of change or slope of a function concerning one of its independent variables. It is commonly used to understand how a function behaves and how it changes over a specific interval. Differentiation helps us answer questions about the steepness of curves, the velocity of moving objects, and the rate of growth or decay in various real-world applications. In essence, it provides a means to quantify and analyze how a function varies, making it a vital tool in mathematics, science, engineering, and many other fields.



Numerical differentiation is the process of computing the value of the derivative of an explicitly unknown function, with given discrete set of points (x_i , y_i) , i = 0, 1, 2, 3, …., n. To differentiate a function numerically, we first determine an interpolating polynomial and then compute the approximate derivative at the given point.

If x_i’s are equispaced:

- Newton's forward interpolation formula is used to find derivative near beginning of table.
- Newton's backward interpolation formula is used to compute derivation near end of table.
- Stirling’s formula is used to estimate the derivative near the center of the table.

If x_i's are not equispaced, we may find using Newton's divided difference method or Lagrange's interpolation formula and then differentiate it as many times as required.

In numerical analysis, differentiation approximates derivatives for complex functions, aiding real-world understanding. It's vital in interpolation for data fitting, making data analysis and modeling precise. Additionally, for solving dynamic systems through differential equations, accurate derivative approximations are key for numerical solutions.

In this chapter, we will thoroughly explore and implement the Newton's Backward and Newton's Forward Differentiation techniques. These methods are vital tools in numerical analysis for approximating derivatives and understanding the rate of change in functions.

Implementation of Newton's Forward & Newton's Backward Differentiation Formula

This section will thoroughly cover the concepts and Python implementation of Newton's Forward and Backward Differentiation Formulas, highlighting their significance in numerical analysis.

a. Newton's Forward Differentiation Formula

Newton's Forward Differentiation is a numerical technique employed to estimate the derivative of a function at a specific point, especially when dealing with tabulated data or discrete points. It is a valuable tool for approximating slopes or rates of change when the function values are known only at distinct data points. This method proves particularly useful in scientific and engineering applications when dealing with real-world data, as it allows us to approximate how a function changes at specific points.

Newton's Forward Differentiation works by taking two closely spaced data points, typically a point of interest and the one just after it, and calculating the difference in function values between these two points. The derivative at the specific point is then approximated by dividing this difference by the step size between the data points. This approach provides a reasonable estimate of the slope or derivative at the chosen point, making it an essential tool for approximating derivatives when dealing with discrete data or tabulated information.

The formula for finding the first order derivative is following:

$$\boxed{\left.\frac{dy}{dx}\right]_{x=x0} = \frac{1}{h}\left[\Delta y_0 - \frac{\Delta^2 y_0}{2} + \frac{\Delta^3 y_0}{3} - \frac{\Delta^4 y_0}{4} + \cdots\right]}$$

Implementation in Python

```
import math

def main():
    x = [0.0] * 20
    y = [[0.0] * 20 for _ in range(20)]
    sum_value = 0.0
    index = 0
    flag = 0
    sign = 1
```

```
   # Read the number of data points
    n = int(input("Enter the number of data points: "))

    # Read the actual data for x and y
    print("Enter data:")
    for i in range(n):
        x[i] = float(input(f"x[{i}] = "))
        y[i][0] = float(input(f"y[{i}] = "))


    # Read the calculation point
    xp = float(input("Enter the value of x where you want to
calculate the derivative: "))


    # Check if given point (xp) is a valid point in the x data
    for i in range(n):
        if abs(xp - x[i]) < 0.0001:
            index = i
            flag = 1
            break


    # If the flag is still 0, (xp) is not in list of x data
    if flag == 0:
        print("Invalid calculation point. Exiting program...")
        exit(0)

    # Generate 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]

    # Calculate the finite difference (step size)
    h = x[1] - x[0]


    # Apply formula to calculate sum of terms for derivatives
    for i in range(1, n - index):
        term = (y[index][i] ** i) / i
        sum_value += sign * term
        sign = -sign


    # Divide by h to get the first derivative
    first_derivative = sum_value / h


    # Display the final result
    print(f"The first derivative at x = {xp:.2f} is
{first_derivative:.2f}")

if __name__ == "__main__":
    main()
```

b. Newton’s Backward Differentiation Formula

Newton's Backward Differentiation Formula, like Newton's Forward Differentiation, is a numerical technique used to estimate the derivative of a function at a specific point, particularly when dealing with tabulated data or discrete points. It is a valuable tool for approximating slopes or rates of change when the function values are known only at distinct data points. This method is also commonly employed in scientific and engineering applications to analyze real-world data and understand how a function changes at specific points.

Newton's Backward Differentiation Formula works by taking two closely spaced data points, typically a point of interest and the one just before it, and calculating the difference in function values between these two points. Unlike the forward differentiation formula, which estimates the derivative using the next data point, the backward differentiation formula estimates it using the previous data point. This approach provides a reasonable estimate of the slope or derivative at the chosen point, making it an essential tool for approximating derivatives when dealing with discrete data or tabulated information.

The formula for finding the first order derivative is following:

$$\boxed{\left.\frac{dy}{dx}\right]_{x=xn} = \frac{1}{h}\left[\nabla y_n + \frac{\nabla^2 y_n}{2} + \frac{\nabla^3 y_n}{3} + \frac{\nabla^4 y_n}{4} + \cdots\right]}$$

Implementation in Python

```
import math

def main():
    x = [0.0] * 20
    y = [[0.0] * 20 for _ in range(20)]
    sum_value = 0.0
    index = 0
    flag = 0

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

    # Read the actual data for x and y
    print("Enter data:")
    for i in range(n):
        x[i] = float(input(f"x[{i}] = "))
        y[i][0] = float(input(f"y[{i}] = "))

    # Read the calculation point
    xp = float(input("Enter the value of x where you want to
calculate the derivative: "))

    # Check if given point (xp) is a valid point in the x data
    for i in range(n):
        if abs(xp - x[i]) < 0.0001:
            # Index of the calculation point
            index = i
            flag = 1
            break
    # If the flag is still 0, the given point (xp) is not in
the list of x data
    if flag == 0:
        print("Invalid calculation point. Exiting the
program...")
        exit(0)

    # Generate 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]

    # Calculate the finite difference (step size)
    h = x[1] - x[0]

    # Apply the formula to calculate the sum of terms for
finding derivatives using the backward difference formula
    for i in range(1, index + 1):
        term = (y[index][i] ** i) / i
        sum_value += term

    # Divide by h to get the first derivative
    first_derivative = sum_value / h

    # Display the final result
    print(f"The first derivative at x = {xp:.2f} is
{first_derivative:.2f}")

if __name__ == "__main__":
    main()
```

Lab Tasks

1. Write a python program implementing the Newton's forward/backward differentiation method to find first order derivative, when
   a. x = 1.1
   b. x = 1.6

x	1.0	1.1	1.2	1.3	1.4	1.5	1.6
y	7.989	8.403	8.781	9.129	9.451	9.750	10.031