Haven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

Lab Report

Course Code:	CSE 2204
Course Title:	Numerical Methods Sessional
Experiment No:	03
Experiment Name:	Exploring Functional Values through Newton's Forward and Backward Interpolation Formula.

Date:

December 15, 2023

Submitted By:	Submitted To:
Name: Md. Abdullah Al Mamun	Shyla Afroge
Section: A	Assistant Professor
Roll No: 2003028	Computer Science & Engineering
Year: 2nd Year Odd Semester	Rajshahi University of Engineering & Technology

Experiment No: 03

Experiment Name: **Exploring Functional Values through Newton's Forward and Backward Interpolation Formula.**

Theory:

Newton's Forward Interpolation Formula:

Given a set of data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ with a common difference $h = x_1 - x_0$, Newton's Forward Interpolation Formula estimates y at a point x within the range (x_0, x_n) as follows:

$$y = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_0 + \ldots + \frac{t(t-1)\ldots(t-n+1)}{n!}\Delta^n y_0$$

where $t = \frac{x-x_0}{h}$ and $\Delta y_i = y_{i+1} - y_i$ are the finite differences.

Algorithm:

1. **Calculate Finite Differences:**
   - Calculate first-order differences: $\Delta y_i = y_{i+1} - y_i$ for $i = 0, 1, \ldots, n-1$.
   - Calculate second-order differences: $\Delta^2 y_i = \Delta y_{i+1} - \Delta y_i$ for $i = 0, 1, \ldots, n-2$.
   - Continue this process up to the n-th order differences.
2. **Calculate Coefficients:**
   - Initialize coefficients $a_0 = y_0$.
   - For $k = 1, 2, \ldots, n$, calculate $a_k = \frac{\Delta^k y_0}{k!}$.
3. **Interpolation Formula:**
   - The interpolated value at a point x is given by the formula:

     $$P(x) = a_0 + a_1 t + a_2 t(t-1) + \ldots + a_n t(t-1)\ldots(t-n+1)$$

     where $t = \frac{x-x_0}{h}$.

Newton's Backward Interpolation Formula:

When the data points are given in equally spaced intervals but in reverse order, Newton's Backward Interpolation Formula estimates y at a point x within the range (x_0, x_n) as follows:

$$y = y_n + t\Delta y_n + \frac{t(t+1)}{2!}\Delta^2 y_n + \ldots + \frac{t(t+1)\ldots(t+n-1)}{n!}\Delta^n y_n$$

where $t = \frac{x-x_n}{h}$ and $\Delta y_i = y_i - y_{i-1}$. These interpolation formulas are valuable tools when evenly spaced data points are available, and the choice between forward and backward interpolation depends on the arrangement of the given data points.

Algorithm:

1. **Calculate Finite Differences:**
   - Calculate first-order differences: $\Delta y_i = y_i - y_{i-1}$ for $i = 1, 2, \ldots, n$.
   - Calculate second-order differences: $\Delta^2 y_i = \Delta y_i - \Delta y_{i-1}$ for $i = 2, 3, \ldots, n$.
   - Continue this process up to the n-th order differences.
2. **Calculate Coefficients:**
   - Initialize coefficients $a_n = y_n$.
   - For $k = 1, 2, \ldots, n$, calculate $a_{n-k} = \frac{\Delta^k y_n}{k!}$.
3. **Interpolation Formula:**
   - The interpolated value at a point x is given by the formula:

$$P(x) = a_n + a_{n-1}t + a_{n-2}t(t+1) + \ldots + a_0 t(t+1)\ldots(t+n-1)$$

   where $t = \frac{x - x_n}{h}$.

Program:

Listing 1: Newton's Forward Interpolation

```
double NewtonForward(double x[], double y[], int n, double p)
{
    double dely[n][n], a = x[0], h = x[1] - x[0], u, y0;

    for (int i = 0; i < n; i++)
        dely[i][0] = y[i];

    for (int i = 1; i < n; i++)
        for (int j = 0; j < n - i; j++)
            dely[j][i] = dely[j + 1][i - 1] - dely[j][i - 1];

    int k = 0;
    while (a - p > h)
    {
        a = x[k];
        k++;
    }
    u = (p - a) / h;
    y0 = dely[0][0];
    double tempU = u;

    for (int i = 1; i < (n - k); i++)
    {
        if (i != 1)
            tempU = tempU * (u - i + 1);

        tempU = tempU / (double)i;
        y0 += tempU * dely[k][i];
    }
    return y0;
}
```

Listing 2: Newton's Backward Interpolation

```
double NewtonBackward(double x[], double y[], int n, double p)
{
    double dely[n][n], a = x[n - 1], h = x[1] - x[0], u, y0;

```

```
        for (int i = 0; i < n; i++)
            dely[i][0] = y[i];

        for (int i = 1; i < n; i++)
            for (int j = 0; j < n - i; j++)
                dely[j][i] = dely[j + 1][i - 1] - dely[j][i - 1];

        int k = 0;
        while (a - p > h)
        {
            a = x[k];
            k++;
        }
        u = (p - a) / h;
        y0 = dely[n - 1][0];
        double tempU = u;

        for (int i = 1; i < (n - k); i++)
        {
            if (i != 1)
                tempU = tempU * (u + i - 1);

            tempU = tempU / (double)i;
            y0 += tempU * dely[n - k - i - 1][i];
        }
        return y0;
    }
```

Listing 3: Main Program

```
    #include <iostream>
    using namespace std;

    void takeInput(double *x, double *y, int &n, double &x0)
    {
        cout << "Enter the number of data points: ";
        cin >> n;
        cout << "Enter the values of x: ";
        for (int i = 0; i < n; i++)
            cin >> x[i];

        cout << "Enter the values of y: ";
        for (int i = 0; i < n; i++)
            cin >> y[i];
        cout << endl;

        cout << "Enter the value of x for which y is to be found: ";
        cin >> x0;
    }

    int main()
    {
        int input = 0;
        while (input != 3)
        {
            // take all the inputs
            int n = 0;
            double x[10000], y[10000], x0;

            // take the choice
            cout << "1. Newton Forward" << endl;
            cout << "2. Newton Backward" << endl;
            cout << "3. Exit" << endl
                 << endl;
            cout << "Note: All the data must be equally spaced" << endl;
            cout << "Enter your choice: ";
            cin >> input;
            cout << endl;

            switch (input)
            {
            case 1:
```

```
            {
                takeInput(x, y, n, x0);
                double y0 = NewtonForward(x, y, n, x0);
                cout << "The value of y at x = " << x0 << " is " << y0 <<
                    endl
                      << endl
                      << endl;
                break;
            }
            case 2:
            {
                takeInput(x, y, n, x0);
                double y0 = NewtonBackward(x, y, n, x0);
                cout << "The value of y at x = " << x0 << " is " << y0 <<
                    endl
                      << endl
                      << endl;
                break;
            }
            case 3:
            {
                return 0;
            }
            default:
            {
                cout << "Invalid choice" << endl;
                break;
            }
                cout << endl
                      << endl;
            }
        }
    }
```

Result:

Newton's Forward and Backward Interpolation Formulas are methods for estimating values between known data points through polynomial interpolation. Forward Interpolation assumes equally spaced data from left to right, using forward finite differences. Backward Interpolation is for right-to-left data with backward finite differences. Both involve systematic finite difference and coefficient calculations, providing efficient interpolation. The choice depends on data arrangement, with forward for left-to-right and backward for right-to-left data. Despite directional distinctions, both share the same principles and are valuable for accurately estimating values within a given data range.

```
┌──(shohag㉿Shohag-Ubuntu)-[~/Documents/LABs/CSE-2204]
└─$ cd "/home/shohag/Documents/LABs/CSE-2204/" && g++ lab03.cpp -o lab03 && "/home/shohag/Documents/LABs/CSE-2204/"lab03
1. Newton Forward
2. Newton Backward
3. Exit

Note: All the data must be equally spaced
Enter your choice: 1

Enter the number of data points: 6
Enter the values of x: 15 20 25 30 35 40
Enter the values of y: 0.2588190 0.3420201 0.4226183 0.5 0.5735764 0.6427876

Enter the value of x for which y is to be found: 17
The value of y at x = 17 is 0.292372


1. Newton Forward
2. Newton Backward
3. Exit

Note: All the data must be equally spaced
Enter your choice: 2

Enter the number of data points: 6
Enter the values of x: 15 20 25 30 35 40
Enter the values of y: 0.2588190 0.3420201 0.4226183 0.5 0.5735764 0.6427876

Enter the value of x for which y is to be found: 38
The value of y at x = 38 is 0.615661
```

Figure 1: Output of the Program