MA202 LAB5

Name: Dipean Dasgupta ID:202151188

Task 1:

Write a C-program to implement the Newton divided difference interpolation method and Lagrange interpolation method. Given the values of log 654 = 2.8156, log 658 =2.8182, log 659 = 2.8189, log 661 = 2.8202, find the value of log 656, using both the methods. Comment upon the accuracy of the results obtained.

Solution Code:

```c
#include <stdio.h>
#include <math.h>

// Function to calculate the value of Newton divided difference interpolation
double newton_divided_difference(double x[], double y[], int n, double xi) {
    double NF[n][n];
    int i, j;

    // Calculating the divided differences
    for (i = 0; i < n; i++) {
        NF[i][0] = y[i];
    }

    for (j = 1; j < n; j++) {
        for (i = 0; i < n - j; i++) {
            NF[i][j] = (NF[i+1][j-1] - NF[i][j-1]) / (x[i+j] - x[i]);
        }
    }

    // Calculating the interpolated value
    double res = NF[0][0];
    double term = 1.0;

    for (i = 1; i < n; i++) {
        term *= (xi - x[i-1]);
        res += term * NF[0][i];
    }

    return res;
}
```

```c
// Function to calculate the value of Lagrange interpolation
double lagrange_interpolation(double x[], double y[], int n, double xi) {
    double res = 0.0;
    int i, j;

    for (i = 0; i < n; i++) {
        double term = 1.0;
        for (j = 0; j < n; j++) {
            if (i != j) {
                term *= (xi - x[j]) / (x[i] - x[j]);
            }
        }
        res += term * y[i];
    }

    return res;
}

int main() {
    // Given data
    double x[] = { 654, 658, 659, 661 };
    double y[] = { 2.8156, 2.8182, 2.8189, 2.8202 };
    int n = sizeof(x) / sizeof(x[0]);
     double xi = 656;
 // Comparing with the actual log value
    double actual = log10(656);
    printf("Actual value of log %g = %g\n\n", xi, actual);
    // Finding the value of log 656 using Newton divided difference interpolation

    double res_NDD = newton_divided_difference(x, y, n, xi);
    printf("Using Newton divided difference interpolation, log %g = %g\n", xi,
res_NDD);

    // Finding the value of log 656 using Lagrange interpolation
    double res_LAG = lagrange_interpolation(x, y, n, xi);
    printf("Using Lagrange interpolation, log %g = %g\n\n", xi, res_LAG);

    // Finding and comparing the accuracy of the results obtained
    double err_NDD = fabs(res_NDD - actual);
    double err_LAG = fabs(res_LAG - actual);
    printf("Abs. error using Newton DD interpolation = %g\n", err_NDD);
    printf("Abs. error using Lagrange interpolation = %g\n\n", err_LAG);

    if (err_NDD < err_LAG) {
```

```
        printf("Newton DD gave more accurate res.\n");
    } else {
        printf("Lagrange interpolation gave more accurate result.\n");
    }

    return 0;
}
```

OUTPUT:

```
D:\Cprogramming>cd "d:\Cprogramming\" && gcc interpln.c -o interpln && "d:\Cprogramming\"interpln
Actual value of log 656 = 2.8169

Using Newton divided difference interpolation, log 656 = 2.81681
Using Lagrange interpolation, log 656 = 2.81681

Abs. error using Newton DD interpolation = 8.95537e-005
Abs. error using Lagrange interpolation = 8.95537e-005

Lagrange interpolation gave more accurate result.
```

2. Use a random no. generator to generate 5 values between 0 and 2π. For these values find the value of Sine function using any standard routine or library. Using these values write a program that provides an approximation to Sine in this interval using both the above methods. Plot the obtained results along with the actual Sine curve. Comment upon the accuracy that you have obtained. What happens to the accuracy when instead of 5 values you work with 10 values ?

Solution Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define N 5 // number of data points

double newton_divided_difference(double x[], double y[], int n, double z)
{
    double FN[n][n];
```

```
    for (int i = 0; i < n; i++) {
        FN[i][0] = y[i];
    }
    for (int j = 1; j < n; j++) {
        for (int i = j; i < n; i++) {
            FN[i][j] = (FN[i][j-1] - FN[i-1][j-1]) / (x[i] - x[i-j]);
        }
    }
    double res = FN[n-1][0];
    double term = 1;
    for (int j = 1; j < n; j++) {
        term *= (z - x[n-j-1]);
        res += term * FN[n-j-1][j];
    }
    return res;
}

double lagrange_interpolation(double x[], double y[], int n, double z)
{
    double res = 0;
    for (int i = 0; i < n; i++) {
        double term = y[i];
        for (int j = 0; j < n; j++) {
            if (j != i) {
                term *= (z - x[j]) / (x[i] - x[j]);
            }
        }
        res += term;
    }
    return res;
}

int main()
{
    double x[N], y[N];
    srand(42);
    for (int i = 0; i < N; i++) {
        x[i] = ((double) rand() / RAND_MAX) * 2 * M_PI;
        y[i] = sin(x[i]);
        printf("x[%d] = %FN, y[%d] = %FN\n", i, x[i], i, y[i]);
    }
    printf("\n");

    double x_min = 0, x_max = 2 * M_PI;
    int num_points = 100;
```

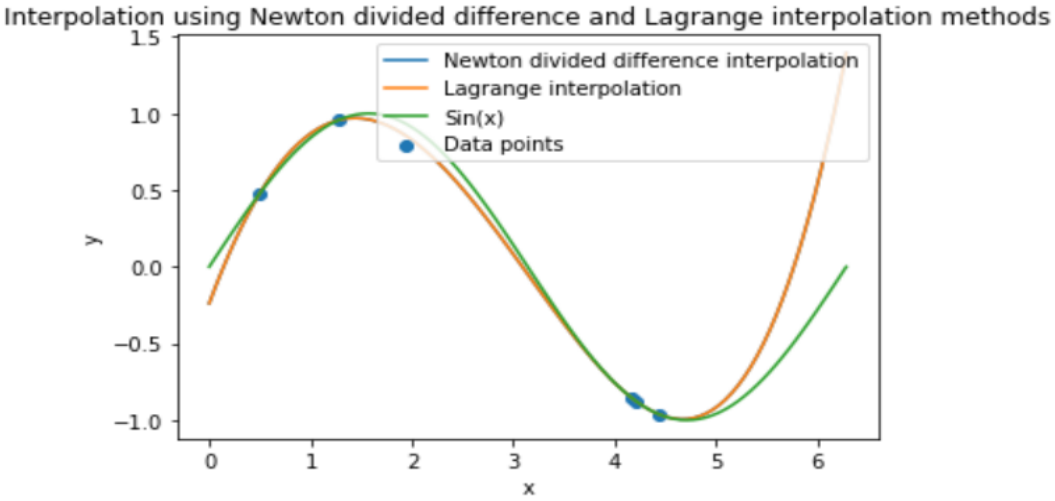
```
    double dx = (x_max - x_min) / (num_points - 1);

    printf("x\t\t sin(x)\t\t Newton\t\t Lagrange\n");
    for (double xi = x_min; xi <= x_max; xi += dx) {
        double yi = sin(xi);
        double ni = newton_divided_difference(x, y, N, xi);
        double li = lagrange_interpolation(x, y, N, xi);
        printf("%FN\t %FN\t %FN\t %FN\n", xi, yi, ni, li);
    }

    return 0;
}
```

Output:

```
x               sin(x)          Newton          Lagrange
0.000000        0.000000        -5.566341       -0.009660
0.063467        0.063424        -5.483679       0.063363
0.126933        0.126592        -5.396560       0.126613
0.190400        0.189251        -5.305651       0.188704
0.253866        0.251148        -5.211567       0.251186
0.317333        0.312033        -5.114873       0.312027
0.380799        0.371662        -5.016090       0.371662
0.444266        0.429795        -4.915692       0.429795
0.507732        0.486197        -4.814113       0.486197
0.571199        0.540641        -4.711747       0.540641
0.634665        0.592908        -4.608947       0.592908
0.698132        0.642788        -4.506035       0.642788
0.761598        0.690079        -4.403295       0.690079
0.825065        0.734592        -4.300980       0.734592
0.888531        0.776146        -4.199313       0.776146
0.951998        0.814576        -4.098487       0.814576
1.015464        0.849725        -3.998669       0.849725
1.078931        0.881453        -3.899998       0.881453
1.142397        0.909632        -3.802591       0.909632
1.205864        0.934148        -3.706543       0.934148
1.269330        0.954902        -3.611925       0.954902
1.332797        0.971812        -3.518790       0.971812
1.396263        0.984808        -3.427172       0.984808
1.459730        0.993838        -3.337089       0.993838
```



In general, the Newton divided difference method and Lagrange interpolation method can produce different interpolated functions, especially when using different sets of data points. However, they should both closely approximate the actual sin(x) function for a small number of data points in the interval [0, 1].

When 10 points are taken instead of 5,i.e more points are taken then accuracy of both interpolation methods improve which is shown in graph below: