RAJSHAHI UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Lab report: 05

Date of Experiment: 07.03.2018
Date of Submission: 24.03.2018

Submitted to:
Shyla Afroge
Assistant Professor,
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

Submitted by:
Riyad Morshed Shoeb
Roll No: 1603013
Section: A
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

Name of the Experiment: Implementation of Gauss' Central Difference Formulae

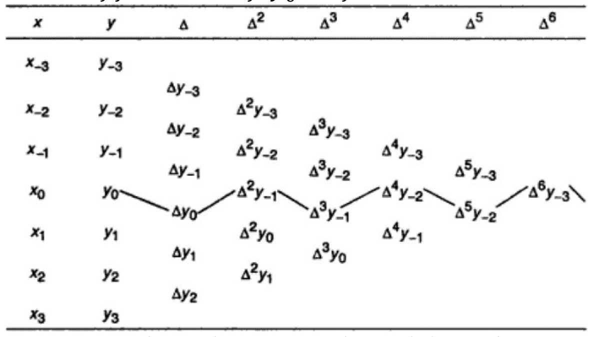
Theory:

Gauss' forward formula

Let's consider the following difference table where the central ordinate is taken for the convenience as y_0 corresponding to $x=x_0$.
The differences used in this formula lie on the line shown on the table. The formula is, therefore, of the form,

$$y_p = y_0 + G_1\Delta y_0 + G_2\Delta^2 y_{-1} + G_3\Delta^3 y_{-1} + G_4\Delta^4 y_{-2} + \;\dots\dots\dots (1)$$

where $G_1, G_2, \dots\dots$ *have to be determined. The* y_p *on the left side can be expressed in the term of* $y_0, \Delta y_0$ *and higher order differences of* y_0*, as follows:*



x	y	Δ	Δ^2	Δ^3	Δ^4	Δ^5	Δ^6
x_{-3}	y_{-3}						
		Δy_{-3}					
x_{-2}	y_{-2}		$\Delta^2 y_{-3}$				
		Δy_{-2}		$\Delta^3 y_{-3}$			
x_{-1}	y_{-1}		$\Delta^2 y_{-2}$		$\Delta^4 y_{-3}$		
		Δy_{-1}		$\Delta^3 y_{-2}$		$\Delta^5 y_{-3}$	
x_0	y_0		$\Delta^2 y_{-1}$		$\Delta^4 y_{-2}$		$\Delta^6 y_{-3}$
		Δy_0		$\Delta^3 y_{-1}$		$\Delta^5 y_{-2}$	
x_1	y_1		$\Delta^2 y_0$		$\Delta^4 y_{-1}$		
		Δy_1		$\Delta^3 y_0$			
x_2	y_2		$\Delta^2 y_1$				
		Δy_2					
x_3	y_3						

$$y_p = y_0 + p\Delta y_0 + \frac{p(p-1)}{2!}\Delta^2 y_0 + \frac{p(p-1)(p-2)}{3!}\Delta^3 y_0 + \;\dots\dots\dots$$

Similarly,

$$\Delta^2 y_{-1} = \Delta^2 y_0 - \Delta^3 y_0 + \Delta^4 y_0 - \Delta^5 y_0 + \;\dots\dots\dots$$
$$\Delta^3 y_{-1} = \Delta^3 y_0 - \Delta^4 y_0 + \Delta^5 y_0 - \Delta^6 y_0 + \;\dots\dots\dots$$
$$\Delta^4 y_{-2} = \Delta^4 y_0 - 2\Delta^5 y_0 + 3\Delta^6 y_0 - 4\Delta^7 y_0 + \;\dots\dots\dots$$

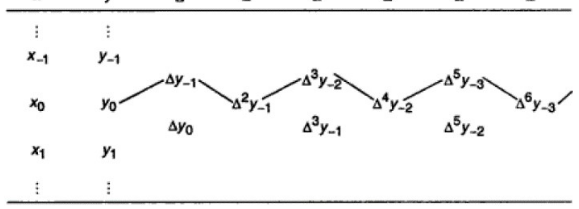
Hence, equation (1) gives the identity,

$$y_0 + p\Delta y_0 + \frac{p(p-1)}{2!}\Delta^2 y_0 + \frac{p(p-1)(p-2)}{3!}\Delta^3 y_0 + \frac{p(p-1)(p-2)(p-3)}{4!}\Delta^4 y_0 + \;\dots\dots =$$
$$y_0 + G_1\Delta y_0 + G_2(\Delta^2 y_0 - \Delta^3 y_0 + \Delta^4 y_0 - \Delta^5 y_0 + \;\dots) + G_3(\Delta^3 y_0 - \Delta^4 y_0 + \Delta^5 y_0 - \Delta^6 y_0 + \;\dots) +$$
$$G_4(\Delta^4 y_0 - 2\Delta^5 y_0 + 3\Delta^6 y_0 - 4\Delta^7 y_0 + \;\dots) + \;\dots\dots\dots$$

Equating the co-efficient from both sides of the equation, we obtain,

$$G_1 = p$$
$$G_2 = \frac{p(p-1)}{2!}$$
$$G_3 = \frac{(p+1)p(p-1)}{3!}$$
$$G_4 = \frac{(p+1)p(p-1)(p-2)}{4!}$$

Gauss' backward formula



x	y	Δ	Δ^2	Δ^3	Δ^4	Δ^5	Δ^6
$\vdots$	$\vdots$						
x_{-1}	y_{-1}						
		Δy_{-1}		$\Delta^3 y_{-2}$		$\Delta^5 y_{-3}$	
x_0	y_0		$\Delta^2 y_{-1}$		$\Delta^4 y_{-2}$		$\Delta^6 y_{-3}$
		Δy_0		$\Delta^3 y_{-1}$		$\Delta^5 y_{-2}$	
x_1	y_1						
$\vdots$	$\vdots$						

Gauss' backward formula can therefore be assumed to be of the form,

$$y_p = y_0 + G_1'\Delta y_{-1} + G_2'\Delta^2 y_{-1} + G_3'\Delta^3 y_{-2} + G_4'\Delta^4 y_{-2} + \;\dots\dots\dots$$

Following the same procedure in Gauss' forward formula, we obtain,

$$G_1' = p$$

$$G_2' = \frac{p(p+1)}{2!}$$

$$G_3' = \frac{(p+1)p(p-1)}{3!}$$

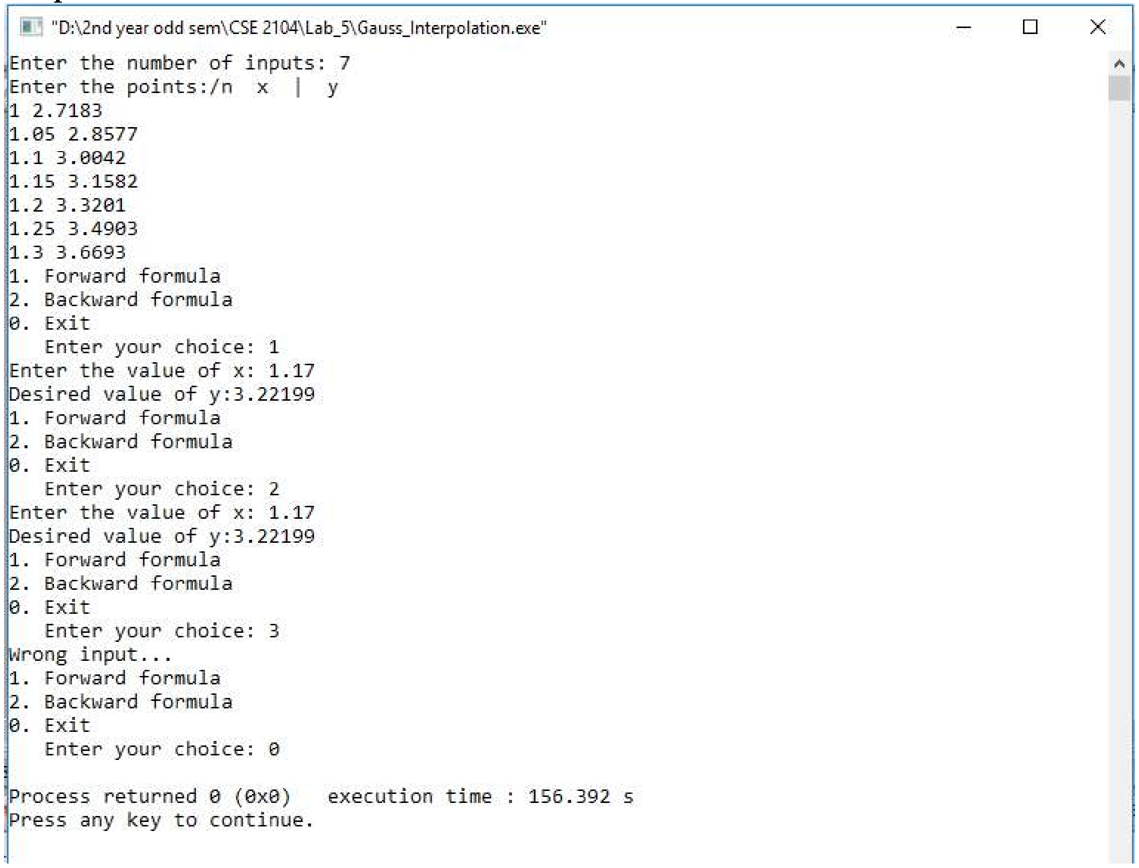
$$G_4' = \frac{(p+2)(p+1)p(p-1)}{4!}$$

Code:

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cmath>
using namespace std;
int main(void)
{
    int n,i,j,checker;
    double ax[200],ay[200],diff[200][200];
    double x,p,h,nr,dr,y=0,y1,y2,y3,y4,y5,y6;
    printf("Enter the number of inputs: ");
    cin>>n;
    printf("Enter the points:/n  x  |  y\n");
    for(i=0;i<n;i++)
        cin>>ax[i]>>ay[i];
    h=ax[1]-ax[0];
    for(i=0;i<n-1;i++)
        diff[i][1]=ay[i+1]-ay[i];
    for(j=2;j<=4;j++)
        for(i=0;i<n-j;i++)
            diff[i][j]=diff[i+1][j-1]-diff[i][j-1];
    while(1)
    {
        printf("1. Forward formula\n2. Backward formula\n0. Exit\n
Enter your choice: ");
        cin>>checker;
        switch(checker)
        {
        case 0:
            return 0;
        case 1:
            printf("Enter the value of x: ");
            cin>>x;
            for(i=0;ax[i]<x;i++);
            p=(x-ax[i])/h;
            y1=p*diff[i][1];
            y2=p*(p-1)*diff[i-1][2]/2;
            y3=(p+1)*p*(p-1)*diff[i-2][3]/6;
            y4=(p+1)*p*(p-1)*(p-2)*diff[i-3][4]/24;
            y=ay[i]+y1+y2+y3+y4;
            cout<<"Desired value of y:"<<y<<endl;
            break;
        case 2:
            printf("Enter the value of x: ");
            cin>>x;
```

```
                for(i=0;ax[i]<x;i++);
                p=(x-ax[i])/h;
                y1=p*diff[i-1][1];
                y2=p*(p+1)*diff[i-1][2]/2;
                y3=(p+1)*p*(p-1)*diff[i-2][3]/6;
                y4=(p+1)*p*(p-1)*(p+2)*diff[i-3][4]/24;
                y=ay[i]+y1+y2+y3+y4;
                cout<<"Desired value of y:"<<y<<endl;
                break;
            default:
                printf("Wrong input...\n");
            }
        }
}
```

Output:



```
"D:\2nd year odd sem\CSE 2104\Lab_5\Gauss_Interpolation.exe"
Enter the number of inputs: 7
Enter the points:/n  x  |  y
1 2.7183
1.05 2.8577
1.1 3.0042
1.15 3.1582
1.2 3.3201
1.25 3.4903
1.3 3.6693
1. Forward formula
2. Backward formula
0. Exit
   Enter your choice: 1
Enter the value of x: 1.17
Desired value of y:3.22199
1. Forward formula
2. Backward formula
0. Exit
   Enter your choice: 2
Enter the value of x: 1.17
Desired value of y:3.22199
1. Forward formula
2. Backward formula
0. Exit
   Enter your choice: 3
Wrong input...
1. Forward formula
2. Backward formula
0. Exit
   Enter your choice: 0

Process returned 0 (0x0)   execution time : 156.392 s
Press any key to continue.
```

Discussion:

The advantage of Gauss' interpolation formulas consists in the fact that this selection of interpolation nodes ensures the best approximation of the residual term of all possible choices, while the ordering of the nodes by their distances from the interpolation point reduces the numerical error in the interpolation.

Name of the Experiment: Implementation of Lagrange’s Interpolation Formula

Theory:

$Let\ y(x) be\ continuous\ and\ differentiable\ (n+1)\ times\ in\ the\ interval\ (a,b). Given\ the\ (n+1)\ points\ (x_0,y_0),(x_1,y_1),\dots\dots,(x_n,y_n)\ where\ the\ values\ of\ x\ need\ not\ necessarily\ be\ eq-ually\ spaced, we\ wish\ to\ find\ a\ polynomial\ of\ degree\ n, say\ L_n(x). Let's\ say, the\ equation\ of\ a\ straight\ line\ passing\ through\ two\ points\ (x_0,y_0)\ and\ (x_1,y_1), then,$

$$L_1(x)=\frac{x-x_1}{x_0-x_1}y_0+\frac{x-x_0}{x_1-x_0}y_1$$

This is the Lagrange polynomial of degree one passing through two points. In a similar way, Lagrange polynomial of degree two passing through three points will be,

$$L_2(x)=\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0+\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1+\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2$$

So, the general formula will be,

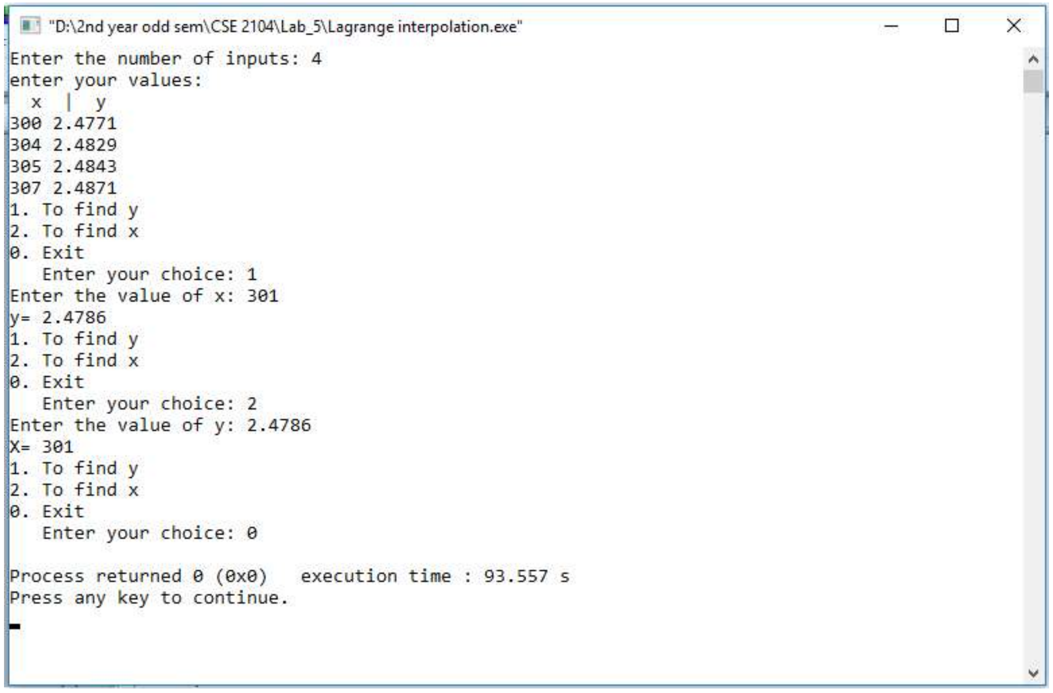
$$L_n(x)=a_0y_0+a_1y_1+\cdots+a_ny_n$$

$$where,\qquad a_i=\frac{(x-x_0)(x-x_1)\dots\dots(x-x_{i-1})(x-x_{i+1})\dots\dots(x-x_n)}{(x_i-x_0)(x_i-x_1)\dots\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots\dots(x_i-x_n)}$$

Code:

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cmath>
using namespace std;

int main(void)
{
    double x[100];
    double y[100];
    double sum=0;
    double yy,xx;
    int n,i,j;
    int checker;
    double r,s;

    printf("Enter the number of inputs: ");
    cin>>n;
    printf("enter your values:\n");
    printf("  x  |  y\n");
    for(i=0;i<n;i++)
    {
       cin>>x[i];
       cin>>y[i];
    }

    while(1)
    {
        printf("1. To find y\n2. To find x\n0. Exit\n    Enter your
choice: ");
        cin>>checker;
        switch(checker)
        {
        case 0:
            return 0;

        case 1:
            printf("Enter the value of x: ");
            cin>>xx;
            for(i=0;i<n;i++)
            {
                 r=1;
                 s=1;
                 for(j=0;j<n;j++)
                 {
                     if(j!=i)
                     {
                         r=r*(xx-x[j]);
                         s=s*(x[i]-x[j]);
                     }
                 }
                 sum+=(r/s)*y[i];
            }
            cout<<"y= "<<sum<<endl;
            sum=0;
            break;

        case 2:
            printf("Enter the value of y: ");
            cin>>yy;
            for(i=0;i<n;i++)
            {
                 r=1;
                 s=1;
                 for(j=0;j<n;j++)
                 {
                     if(j!=i)
                     {
                         r=r*(yy-y[j]);
                         s=s*(y[i]-y[j]);
                     }
                 }
                 sum+=(r/s)*x[i];
            }
            cout<<"X= "<<sum<<endl;
            sum=0;
            break;

        default:
            printf("Wrong choice...\n");
        }
    }
}
```

Output:



"D:\2nd year odd sem\CSE 2104\Lab_5\Lagrange interpolation.exe"

```
Enter the number of inputs: 4
enter your values:
  x  |  y
300 2.4771
304 2.4829
305 2.4843
307 2.4871
1. To find y
2. To find x
0. Exit
   Enter your choice: 1
Enter the value of x: 301
y= 2.4786
1. To find y
2. To find x
0. Exit
   Enter your choice: 2
Enter the value of y: 2.4786
X= 301
1. To find y
2. To find x
0. Exit
   Enter your choice: 0

Process returned 0 (0x0)   execution time : 93.557 s
Press any key to continue.
```

Discussion:

The Lagrange form of the interpolation polynomial shows the linear character of polynomial interpolation and the uniqueness of the interpolation polynomial. Therefore, it is preferred in proofs and theoretical arguments. Uniqueness can also be seen from the invertibility of the Vandermonde matrix, due to the non-vanishing of the Vandermonde determinant.