
IMS Engineering College, Ghaziabad - Delhi NCR

A NAAC ACCREDITED INSTITUTION

**[Session 2016-17 ]**

# LAB FILE

**NT LAB**

**[NCS-353]**

**Faculty Name :**

**Mr. Aditya k Saxena**

**Mr. Bhupesh Gupta**

## LIST OF EXPERIMENTS

| Experiment No | Name /Objectives | Outcomes | Co-relation with POs | Co-relation with PSOs |
|---|---|---|---|---|
| 1 | Aim: To deduce error involved in polynomial equation. | Outcome: To understand the concepts of roots of polynomial equations. | 1,2,3,4,12 | 1,2 |
| 2 | Aim: To Find out the root of the Algebraic and Transcendental equations using Bisection method. | Outcome: To understand concepts of finding roots using bisection method. | 1,2,3,4,12 | 1,2 |
| 3 | Aim: To Find out the root of the Algebraic and Transcendental equations using Regula-Falsi method. | Outcome : to understand the concepts of finding roots using regula falsi method | 1,2,3,4,12 | 1,2 |
| 4 | Aim: To Find out the root of the Algebraic and Transcendental equations using Newton-Raphson method. | Outcome : to understand the concepts of finding roots using newton raphson method | 1,2,3,4,12 | 1,2 |
| 5 | Aim: To Find out the root of the Algebraic and Transcendental equations using Iterative method. | Outcome : To understand the concept finding roots using iterative method. | 1,2,3,4,12 | 1,2 |
| 6 | Aim : To implement Numerical Integration using Trapezoidal rule. | Outcome : To understand the concepts of Integration using Trapezoidal rule. | 1,2,3,4,12 | 1,2 |
| 7 | Aim: To implement Numerical Integration using Simpson 1/3 rule | Outcome : to understand the concepts of integration using Simpson 1/3 rule. | 1,2,3,4,12 | 1,2 |
| 8 | Aim: To implement Numerical Integration Simpson 3/8 rule. | Outcome : to understand the concept of integration using Simpson 3/8 rule. | 1,2,3,4,12 | 1,2 |
| 9 | Aim: To implement Newton's Forward Interpolation formula. | Outcome : To understand the concept of interpolation formula. | 1,2,3,4,12 | 1,2 |
| 10 | Aim: Write a program to implement Newton backward | Outcome: To understand the concept of newton backword | 1,2,3,4,12 | 1,2 |

| | | | | |
|---|---|---|---|---|
| | method of interpolation | method of interpolation. | | |
| 11 | Aim: Write a program to implement GAUSS' FORWARD interpolation formula | Outcome : To understand the concept of gauss forward interpolation formula. | 1,2,3,4,12 | 1,2 |
| 12 | Aim: Write a program to implement Gauss's Backward interpolation formula | Outcome: To understand the concept of gauss backward interpolation formula. | 1,2,3,4,12 | 1,2 |
| 13 | Aim: Write a program to implement Lagaranges interpolation formula. | Outcome: To understand the concept of lagaranges interpolation formula. | 1,2,3,4,12 | 1,2 |

**Guidelines to Students**

- Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.
- Students are required to carry their observation / programs book with completed exercises while entering the lab.
- Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.
- Lab can be used in free time / lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- Lab records need to be submitted on or before date of submission.
- Students are not supposed to use disks.

## Program 1 :

## Objective : Algorithm to deduce error involved in polynomial equation.

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Theory -** Absolute and relative error are two types of error with which every experimental scientist should be familiar. The differences are important.

**Absolute Error**: Absolute error is the amount of physical error in a measurement, period. Let's say a meter stick is used to measure a given distance. The error is rather hastily made, but it is good to 1*mm*. This is the absolute error of the measurement. That is,

absolute error = 1mm (0.001m).

In terms common to Error Propagation

absolute error=del x

where *x* is any variable.

**Relative Error**: Relative error gives an indication of how good a measurement is relative to the size of the thing being measured. Let's say that two students measure two objects with a meter stick. One student measures the height of a room and gets a value of 3.215 meters (0.001m). Another student measures the height of a small cylinder and measures 0.075 meters (0.001m). Clearly, the overall accuracy of the ceiling height is much better than that of the 7.5 cm cylinder. The comparative accuracy of these measurements can be determined by looking at their relative errors.

$$\text{relative error} = \frac{\text{absolute error}}{\text{Value of quantity being measured}}$$

**Algorithm:**

**Step-1.** Start of the program.

**Step-2.** Input the variable t_val, a_value.

**Step-3.** Calculate absolute error as

abs_err=|t_val-a_val|

**Step-4.** Calculate relative error as

rel_err=abs_err/t_val

**Step-5.** Calculate percentage relative error as

p_rel_err=rel_err*100

**Step-6.** PRINT abs_err, rel_err and p_rel_err

**Step-7.** STOP

**Program**

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

void main()
{
double abs_err, rel_err, p_rel_err, t_val, a_val;

printf("\n INPUT TRUE ALUE:");
scanf("%lf", &t_val);

printf("\n INPUT APPROXIMATE ALUE:");
 scanf("%lf", &a_val);
```

```
abs_err=fabs(t_val-_val);
rel_err=abs_err/t_val;

p_rel_err=rel_err*100;

printf("\nABSOLUTE ERROR= %lf", abs_err); printf("\nRELATIVE
ERROR= %lf", rel_err); printf("\nPERCENTAGE RELATIVE ERROR=
%lf", p_rel_err);

getch();
}
```

**Output**- INPUT TRUE VALUE
23
INPUT APPROXIMATE VALUE
23.5
ABSOLUTE ERROR= 0.5
RELATIVE ERROR= 0.0217
PERCENTAGE RELATIVE ERROR= 2.17

**APPLICATIONS: Learning of the concepts of roots of polynomial equations and implementation details in c programming language.**

## Program2 :

**Objective : To Find out the root of the Algebraic and Transcendental equations using Bisection method.**

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Related Theory : Theory: -** The **bisection method** in mathematics is a root-finding **method** that repeatedly bisects an interval and then selects a subinterval in which a root must lie for further processing. It is a very simple and robust **method**, but it is also relatively slow.

Bisection method also known as **Bolzano method** is one of the simplest method. To start with two initial approximations, say x1 and x2 such that f(x1)*f(x2)<0 which ensures that root lies between x1 and x2 are taken. The next x value say x3 as mid point of the interval [x1, x2] is computed.

There are three possibilities that can arise:
i) If f(x3), then we have a root at x3=0
ii) If f(x1) and f(x3) are of opposite sign, then root lies in the interval (x1, x3). Thus x2 is replaced by x3, and the new interval which is half of the current interval, is again bisected.
iii)If f(x1) and f(x3) are of same sign, then root lies in the interval(x3,x2). Thus x1 is replaced by x3, and the new interval is again bisected.

Therefore , by repeating this interval bisection procedure we keep enclosing the root in a new search interval, which is halved in each iteration.. This iterative cycle is terminated when the search interval becomes smaller than the prescribed tolerance i.e **epsilon.**

**Algorithm:-**

**Step-1.** Start of the program.

**Step-2.** Input the variable x1, x2 for the task.

**Step-3.** Check f(x1)*f(x2)<0

**Step-4.** If yes proceed

**Step-5.** If no exit and print error message

**Step-6.** Repeat 7-11 if condition not satisfied

**Step-7.** X0=(x1+x2)/2

**Step-8.** If f(x0)*f(x1)<0

**Step-9.** X2=x0

**Step-10.** Else

**Step-11.** X1=x0

**Step-12.** Condition:

**Step-13.** if | (x1-x2)/x1) | < maximum possible error or f(x0)=0

**Step-14.** Print output

**Step-15.** End of program.

**Program Code :**

```
#include<stdio.h>
#include <math.h>
#include<conio.h>
#define ESP 0.001
#define F(x) (x)*(x)*(x) + (x)*(x) + (x) + 7
void main()
{
  int i = 1;
  float x0,x1,x2;
  double f1,f2,f0,t;
  clrscr();
  printf("\nEnter the value of x0: ");
  scanf("%f",&x0);

  printf("\nEnter the value of x1: ");
  scanf("%f",&x1);
  printf("\n_____\n");
  printf("\niteration\t x0\t       x1\t x2\t   f0\t   f1\t   f2");
  printf("\n_____\n");
  do
  {
  x2=(x0+x1)/2;
  f0=F(x0);
  f1=F(x1);
  f2=F(x2);
  printf("\n%d %f %f %f %lf %lf %lf", i, x0,x1,x2,f0,f1,f2);
  if(f0*f2<0)
   {
    x1=x2;
   }
   else
   {
    x0=x2;
   }
   i++;
  }while(fabs(f2)>ESP);
printf("\n_____\n");
printf("\n\nApp.root = %f",x2);
getch();
}
```

**OUT PUT**

```
Enter the value of x0: -2
Enter the value of x0: -1
Enter the value of x1: -2
```

| x0 | x1 | x2 | f0 | f1 | f2 |
|---|---|---|---|---|---|
| -1.000000 | -2.000000 | -1.500000 | -5.000000 | 2.000000 | -1.750000 |
| -1.500000 | -2.000000 | -1.750000 | -1.750000 | 2.000000 | 0.062500 |
| -1.500000 | -1.750000 | -1.625000 | -1.750000 | 0.062500 | -0.859375 |
| -1.625000 | -1.750000 | -1.687500 | -0.859375 | 0.062500 | -0.402344 |
| -1.687500 | -1.750000 | -1.718750 | -0.402344 | 0.062500 | -0.170898 |
| -1.718750 | -1.750000 | -1.734375 | -0.170898 | 0.062500 | -0.054443 |
| -1.734375 | -1.750000 | -1.742188 | -0.054443 | 0.062500 | 0.003967 |
| -1.734375 | -1.742188 | -1.738281 | -0.054443 | 0.003967 | -0.025253 |
| -1.738281 | -1.742188 | -1.740234 | -0.025253 | 0.003967 | -0.010647 |
| -1.740234 | -1.742188 | -1.741211 | -0.010647 | 0.003967 | -0.003341 |
| -1.741211 | -1.742188 | -1.741699 | -0.003341 | 0.003967 | 0.000313 |

App.root = -1.741699

**Applications: Students will be able to understand the concepts of bisection method and will be able to implement the details in c programming language.**

**Program3 :**

**Objective : Program of FALSE POSITION or REGULA-FALSI METHOD.**

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Algorithm**:-

**Step-1.** Start of the program.

**Step-2.** Input the variable x0, x1,e, n for the task.

**Step-3.** f0=f(x0)

**Step-4.** f2=f(x2)

**Step-5.** for i=1 and repeat if i<=n

**Step-6.** x2 = (x1.f1-xo.f1)/(f1-f0)

**Step-7.** f2 = x2

**Step-8.** if | f2 | <=e

**Step-9.** print "convergent ", x2, f2

**Step-10.** if sign (f2)!=sign(f0)

**Step-11.** x1=x2 & f1 = f2

**Step-12.** else

**Step-13.** X0 = x2 & f0 = f2

**Step-14.** End loop

**Step-15.** Print output

**Step-16.** End the program.

**PROGRAM: FALSE POSITION or REGULA-FALSI METHOD**.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define ESP 0.0001
#define F(x) 3*(x) - 1 - cos(x)
void main()
{
  float x0,x1,x2,f1,f2,f0;
  int count=0;
  clrscr();
  do
```

```
 {
 printf("\nEnter the value of x0: ");
 scanf("%f",&x0);
 }while(F(x0) > 0);
 do
 {
 printf("\nEnter the value of x1: ");
 scanf("%f",&x1);

 }while(F(x1) < 0);
 printf("\n_____\n");
 printf("\n    x0\t       x1\t x2\t   f0\t   f1\t   f2");
 printf("\n_____\n");
 do
 {
 f0=F(x0);
 f1=F(x1);
 x2=x0-((f0*(x1-x0))/(f1-f0));
 f2=F(x2);
 printf("\n%f %f %f %f %f",x0,x1,x2,f0,f1,f2);
 if(f0*f2<0)
  {
   x1=x2;
  }
  else
  {
   x0 = x2;
  }
 }while(fabs(f2)>ESP);

printf("\n_____\n");
printf("\n\nApp.root = %f",x2);
getch();
}
```

**OUT PUT**

```
Enter the value of x0: -1
Enter the value of x1: 1
_____________________________________________________________

   x0        x1        x2        f0       f1       f2
_____________________________________________________________

-1.000000 1.000000 0.513434 -4.540302 1.459698 -0.330761
0.513434  1.000000 0.603320 -0.330761 1.459698 -0.013497
0.603320  1.000000 0.606954 -0.013497 1.459698 -0.000527
0.606954  1.000000 0.607096 -0.000527 1.459698 -0.000021
_____________________________________________________________
App.root = 0.607096
```

**APPLICATIONS: Students will be able to understand the regula-falsi method and will be able to implement the concepts in c programming language.**

**Program4 :**

**Objective : Write a program of NEWTON-RAPHSON METHOD**

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Related Theory :**

- Newton raphson method also known as Newton's method of tangents, is one of the fast iterative methods. .This method begins with one initial approximation . Here one have to take due care while selecting the initial approximation. , as it is very sensitive to the initial approximation. Once proper choice is made for initial approximation, it converges faster than false position method and the secant method**.**

**Algorithm:-**

**Step-1.** Start of the program.

**Step-2.** input the variables x0, n for the task.

**Step-3.** input Epsilon & delta

**Step-4.** for i= 1 and repeat if i <= n

**Step-5.** f0 = f(x0)

**Step-6.** dfo = df(x1)

**Step-7.** if | dfo | <= delta

- **a.** Print slope too small
- **b.** Print x0, f0, df0, i
- **c.** End of program

**Step-8.** x1 = x0 –(f0/df0)

**Step-9.** if | (x1-x0/x1) | < epsilon

- **a.** Print convergent
- **b.** Print x1, f(x1), i
- **c.** End of program

**Step-10.** x0 = x1

**Step-11.** End loop.

**PROGRAM:**

```
#include<conio.h>
#include<stdio.h>
```

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

int user_power,i=0,cnt=0,flag=0;
int coef[10]={0};
float x1=0,x2=0,t=0;
float fx1=0,fdx1=0;

void main()
{

   clrscr();


   printf("\n\n\t\t\t PROGRAM FOR NEWTON RAPHSON GENERAL");

   printf("\n\n\n\tENTER THE TOTAL NO. OF POWER:::: ");
   scanf("%d",&user_power);

   for(i=0;i<=user_power;i++)
   {
      printf("\n\t x^%d::",i);
      scanf("%d",&coef[i]);
   }

   printf("\n");

   printf("\n\t THE POLYNOMIAL IS ::: ");
   for(i=user_power;i>=0;i--)//printing coeff.
   {
      printf(" %dx^%d",coef[i],i);
   }

   printf("\n\tINTIAL X1---->");
   scanf("%f",&x1);

   printf("\n ***********************************************************");
   printf("\n ITERATION    X1    FX1    F'X1  ");
   printf("\n ***********************************************************");

   do
   {
         cnt++;
         fx1=fdx1=0;
         for(i=user_power;i>=1;i--)
         {
            fx1+=coef[i] * (pow(x1,i)) ;
         }
         fx1+=coef[0];
         for(i=user_power;i>=0;i--)
         {
            fdx1+=coef[i]* (i*pow(x1,(i-1)));
         }
         t=x2;
```

```
        x2=(x1-(fx1/fdx1));

        x1=x2;

        printf("\n %d        %.3f  %.3f  %.3f ",cnt,x2,fx1,fdx1);

   }while((fabs(t - x1))>=0.0001);
   printf("\n\t THE ROOT OF EQUATION IS %f",x2);
   getch();
}
```

**OUTPUT :**

```
   ENTER THE TOTAL NO. OF POWER:::: 3

    x^0::-3

    x^1::-1

    x^2::0

    x^3::1


    THE POLYNOMIAL IS :::  1x^3 0x^2 -1x^1 -3x^0

    INTIAL X1---->3

*************************************
ITERATION    X1    FX1    F'X1
*************************************
1       2.192  21.000 26.000
2       1.794  5.344  13.419
3       1.681  0.980  8.656
4       1.672  0.068  7.475
5       1.672  0.000  7.384
*************************************

    THE ROOT OF EQUATION IS 1.671700
```

**APPLICATION: Student will learn about Newton rapson method and will implements the details in c programming language.**

**Program5 :**

**Objective : - Write a program to find roots using iteration method.**

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Related Theory:**

In certain case the iterative methods are preferred over direct methods particularly coefficient matrix is sparse i.e have many zeros. These methods are more rapid. They are more economical in memory requirements of a computer. Sometimes these methods are used to reduce round –off error in solutions computed by direct methods. They can also used to find the solution of the system of non-linear equations. These methods are

also called methods of successive approximations. The value of unknown is improved by substituting directly the previous value of unknowns in the equation. The necessary and sufficient condition for use of these methods is that the diagonal elements of coefficient matrix must be dominant. i.e the element a11, a22, a33,… ann must be sufficiently dominant. Such a system of linear equations is known as a diagonal system. Note that it is very difficult to define the exact minimum margin by which these coefficients must dominate the other coefficients to ensure convergence and it is even more difficult to predict the rate of convergence for some combinations of coefficient value when convergence is assured.

**Algorithm (Iteration Method)**

**Step-1.** Define function *f*(*x*)

**Step-2.** Define function *df*(*x*)

**Step-3.** Get the value of *a*, max_err.

**Step-4.** Initialize *j*

**Step-5.** If *df*(*a*) < 1 then *b* = 1, *a* = *f*(*a*)

**Step-6.** Print root after *j*, iteration is *f*(*a*)

**Step-7.** If *fabs*(*b* – *a*) > max_err then

**Step-8.** *j*++, goto (5)

**Step-9.** End if

**Step-10.** Else print root doesn‟t exist

**Step-11.** End.

**Program Code :**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define ESP 0.0001
#define X1(x2,x3) ((17 - 20*(x2) + 2*(x3))/20)
#define X2(x1,x3) ((-18 - 3*(x1) + (x3))/20)
```

```
#define X3(x1,x2) ((25 - 2*(x1) + 3*(x2))/20)

void main()
{
  double x1=0,x2=0,x3=0,y1,y2,y3;
  int i=0;
  clrscr();
  printf("\n_____\n");
  printf("\n   x1\t\t   x2\t\t   x3\n");
  printf("\n_____\n");
  printf("\n%f\t%f\t%f",x1,x2,x3);
  do
  {
   y1=X1(x2,x3);
   y2=X2(x1,x3);
   y3=X3(x1,x2);
   if(fabs(y1-x1)<ESP && fabs(y2-x2)<ESP && fabs(y3-x3)<ESP )
   {
     printf("\n_____\n");


     printf("\n\nx1 = %.3lf",y1);
     printf("\n\nx2 = %.3lf",y2);
     printf("\n\nx3 = %.3lf",y3);
     i = 1;
   }
   else
   {
     x1 = y1;
     x2 = y2;
     x3 = y3;
     printf("\n%f\t%f\t%f",x1,x2,x3);
   }
  }while(i != 1);
getch();
}
```

**OUT PUT**

```
_____
   x1            x2            x3
_____

0.000000       0.000000      0.000000
0.850000      -0.900000      1.250000
1.875000      -0.965000      1.030000
1.918000      -1.129750      0.917750
2.071525      -1.141812      0.888737
2.080686      -1.166292      0.871576
2.103449      -1.168524      0.866988
2.105223      -1.172168      0.864376
2.108606      -1.172565      0.863653
2.108930      -1.173108      0.863255
2.109434      -1.173177      0.863141
x1 = 2.109
```

x2 = -1.173
x3 = 0.863

**APPLICATIONS: Students will be able to learn about the concepts of finding roots using iteration method and be able to implement the details in C programming language.**

**Program6 :**

**Objective : Write a program to implement trapezoidal rule.**

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Related Theory:**

Trapezoidal method is basically a numerical integration method. Numerical method is basically a process of computing the value of a definite integral from a set of numerical values of the function referred to as integrand. When applied to the integration of a function of a single variable, the process is sometimes called Numerical Quadrature. When applied to compute double integral of a function of two independent variables, the process is called numerical cubature. If the function is defined as a mathematical expression , then its integral is usually determined using the technique of calculus.

The Trapezoidal rule approximates the area under a curve by connecting successive points on the curve to form trepezoids of uniform width, and then summing the area under these trapezoids to obtain the approximate area under the curve.

**ALGORITHM:-**

**Step-1.** Start of the program.

**Step-2.** Input Lower limit a

**Step-3.** Input Upper Limit b

**Step-4.** Input number of sub intervals n

**Step-5.** h=(b-a)/n

**Step-6.** sum=0

**Step-7.** sum=fun(a)+fun(b)

**Step-8.** for i=1; i<n; i++

**Step-9.** sum +=2*fun(a+i)

**Step-10.** End Loop i

**Step-11.** result =sum*h/2;

**Step-12.** Print Output result

**Step-13.** End of Program

**Step-14.** Start of Section fun

**Step-15.** temp = 1/(1+(x*x))

**Step-16.** Return temp

**Step-17.** End of Section fun.

**PROGRAM: TRAPEZOIDAL METHOD.**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
  float x[10],y[10],sum=0,h,temp;
  int i,n,j,k=0;
  float fact(int);
  clrscr();
  printf("\nhow many record you will be enter: ");
  scanf("%d",&n);
  for(i=0; i<n; i++)
  {
   printf("\n\nenter the value of x%d: ",i);
   scanf("%f",&x[i]);
   printf("\n\nenter the value of f(x%d): ",i);
   scanf("%f",&y[i]);
  }
  h=x[1]-x[0];
  n=n-1;
  for(i=0;i<n;i++)
  {
    if(k==0)
    {
     sum = sum + y[i];
     k=1;
    }
    else
     sum = sum + 2 * y[i];
   }
   sum = sum + y[i];
   sum = sum * (h/2);
   printf("\n\n  I = %f  ",sum);
getch();
}
```

**OUT PUT**

```
how many record you will be enter: 6
enter the value of x0: 7.47
enter the value of f(x0): 1.93
enter the value of x1: 7.48
enter the value of f(x1): 1.95
enter the value of x2: 7.49
enter the value of f(x2): 1.98
enter the value of x3: 7.50
enter the value of f(x3): 2.01
enter the value of x4: 7.51
enter the value of f(x4): 2.03
enter the value of x5: 7.52
enter the value of f(x5): 2.06
  I = 0.099652
```

**APPLICATIONS: Student will be able to learn about trapezoidal rule and will implement the concepts in c programming .**

**Program7 :**

**Objective : -Write a program to implement simpson 1/3 rd rule**.

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Related Theory :**

Simpson 1/3 method is numerical integration method. Simpson's 1/3 rule gives more accurate approximation of the integral value since it connects three points on the curve by second by second order parabolas and then sum the areas under the parabolas to obtain the approximate area under the curve.

**ALGORITHM :-**

**Step-1.** Start of the program.

**Step-2.** Input Lower limit a

**Step-3.** Input Upper limit b

**Step-4.** Input number of subintervals n

**Step-5.** h=(b–a)/n

**Step-6.** sum=0

**Step-7.** sum=fun(a)+4*fun(a+h)+fun(b)

**Step-8.** for i=3; i<n; i + = 2

**Step-9.** sum + = 2*fun(a+(i – 1)*h) + 4*fun(a+i*h)

**Step-10.** End of loop i

**Step-11.** result=sum*h/3

**Step-12.** Print Output result

**Step-13.** End of Program

**Step-14.** Start of Section fun

**Step-15.** temp = 1/(1+(x*x))

**Step-16.** Return temp

**Step-17.** End of Section fun

**PROGRAM: SIMPSON'S 1/3$^{rd}$ METHOD**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
```

```
{
  float x[10],y[10],sum=0,h,temp;
  int i,n,j,k=0;
  float fact(int);
  clrscr();
  printf("\nhow many record you will be enter: ");
  scanf("%d",&n);
  for(i=0; i<n; i++)
  {

   printf("\n\nenter the value of x%d: ",i);
   scanf("%f",&x[i]);
   printf("\n\nenter the value of f(x%d): ",i);
   scanf("%f",&y[i]);
  }
  h=x[1]-x[0];
  n=n-1;
  sum = sum + y[0];
  for(i=1;i<n;i++)
  {
    if(k==0)
    {
     sum = sum + 4 * y[i];
     k=1;
    }
    else
    {
     sum = sum + 2 * y[i];
     k=0;
    }
   }
   sum = sum + y[i];
   sum = sum * (h/3);
   printf("\n\n  I = %f  ",sum);
getch();}
```

**OUT PUT**

how many record you will be enter: 5
enter the value of x0: 0
enter the value of f(x0): 1
enter the value of x1: 0.25
enter the value of f(x1): 0.8
enter the value of x2: 0.5
enter the value of f(x2): 0.6667
enter the value of x3: 0.75
enter the value of f(x3): 0.5714
enter the value of x4: 1
enter the value of f(x4): 0.5

I = 0.693250

**APPLICATIONS: Students will be able to implement the concepts of simpsons 1/3 rule and be able to implement the details in c programming language.**

**Program8 :**

**Objective : Write a program to implement simpson 3/8 rule.**

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**ALGORITHM**

**Step-1.** Start of the program.

**Step-2.** Input Lower limit a

**Step-3.** Input Upper limit b

**Step-4.** Input number of sub intervals n

**Step-5.** h = (b – a)/n

**Step-6.** sum = 0

**Step-7.** sum = fun(a) + fun (b)

**Step-8.** for i = 1; i < n; i++

**Step-9.** if i%3=0:

**Step-10.** sum + = 2*fun(a + i*h)

**Step-11.** else:

**Step-12.** sum + = 3*fun(a+(i)*h)

**Step-13.** End of loop i

**Step-14.** result = sum*3*h/8

**Step-15.** Print Output result

**Step-16.** End of Program

**Step-17.** Start of Section fun

**Step-18.** temp = 1/(1+(x*x))

**Step-19.** Return temp

**Step-20.** End of section fun

**PROGRAM: SIMPSON'S 3/8$^{th}$ METHOD OF NUMERICAL INTEGRATION**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
  float x[10],y[10],sum=0,h,temp;
```

```
  int i,n,j,k=0,l=0;
  float fact(int);
  clrscr();
  printf("\nhow many record you will be enter: ");
  scanf("%d",&n);
  for(i=0; i<n; i++)
  {
   printf("\n\nenter the value of x%d: ",i);
   scanf("%f",&x[i]);
   printf("\n\nenter the value of f(x%d): ",i);
   scanf("%f",&y[i]);
  }
  h=x[1]-x[0];
  n=n-1;
  sum = sum + y[0];
  for(i=1;i<n;i++)
  {
    if(k==0 || l==0)
    {
     sum = sum + 3 * y[i];
     if(k==1)
      {
       l=1;
      }
     k=1;
    }
    else
    {
     sum = sum + 2 * y[i];
     k=0;
     l=0;
    }
   }
   sum = sum + y[i];
   sum = sum * (3*h/8);
   printf("\n\n  I = %f  ",sum);
getch();
}
```

**OUT PUT**

```
how many record you will be enter: 10
enter the value of x0: 0.1
enter the value of f(x0): 1.001
enter the value of x1: 0.2
enter the value of f(x1): 1.008
enter the value of x2: 0.3
enter the value of f(x2): 1.027
enter the value of x3: 0.4
enter the value of f(x3): 1.064
enter the value of x4: 0.5
enter the value of f(x4): 1.125
enter the value of x5: 0.6
enter the value of f(x5): 1.216
enter the value of x6: 0.7
enter the value of f(x6): 1.343
```

enter the value of x7: 0.8
enter the value of f(x7): 1.512
enter the value of x8: 0.9
enter the value of f(x8): 1.729
enter the value of x9: 1.0
enter the value of f(x9): 2
I = 1.149975

**APPLICATIONS: Students will learn about the concepts of simpsons 3/8 rule and will implement the details in c programming language.**

**Program9 :**

**Objective : - Write a program to implement Newton forward method of interpolation.**

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Theory:** - Suppose x and y are two variable and their relations can be expressed as y=f(x) x1<=x<=xn. Then we say that x is an independent variable & y is a dependent variable. When the form of f(x)is known , then the value of y can be computed directly corresponding to any value of x in the range x1<=x<=xn.

However , if the form of f(x) is not known and only a set of values (x1,y1), (x2,y2), (x3,y3)……..(xn, yn) satisfying the relation y=f(x) are known, then the process of estimating (approximating)the value of dependent variable y for a given value of independent variable x in the range x1<=x<=xn is known as interpolation.

Forward Differences:- If y1, y2,y3 and yn denote the value of the function of type y=f(x) at x=x1, x2, x3, ……..xn, then y2-y1, y3-y2, y4-y3, ……yn-yn-1 are called the forward differences of y. These differences are denoted as del y1, del y2, del y3, del yn-1.

**Algorithm**

**Step-1.** Start of the program

**Step-2.** Input number of terms n

**Step-3.** Input the array ax

**Step-4.** Input the array ay

**Step-5.** h=ax[1] – ax[0]

**Step-6.** for i=0; i<n-1; i++

**Step-7.** diff[i] [1]=ay[i + 1] – ay[i]

**Step-8.** End Loop i

**Step-9.** for j=2; j<=4; j++

**Step-10.** for i = 0; i <n – j; i++

**Step-11.** diff[i][j]=diff [i + 1] [j – 1]-diff [i][j – 1]

**Step-12.** End Loop i

**Step-13.** End Loop j

**Step-14.** i=0

**Step-15.** Repeat Step 16 until ax[i]<x

**Step-16.** i=i + 1

**Step-17.** i=i – 1;

**Step-18.** p=(x – ax [i])/h

**Step-19.** y1=p*diff[i – 1][1]

**Step-20.** y2=p*(p+1)*diff [i – 1][2]/2

**Step-21.** y3=(p+1)*p*(p-1)*diff[i –2 ][3]/6

**Step-22.** y4=(p+2)*(p+1)*p*(p – 1)*diff[i – 3][4]/24

**Step-23.** y=ay[i]+y1+y2+y3+y4

**Step-24.** Print output x, y

**Step-25.** End of program.

**PROGRAM: NEWTON'S FORWORD METHOD OF INTERPOLATION**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
  float x[10],y[10][10],sum,p,u,temp;
  int i,n,j,k=0,f,m;
  float fact(int);
  clrscr();
  printf("\nhow many record you will be enter: ");
  scanf("%d",&n);
  for(i=0; i<n; i++)
  {
   printf("\n\nenter the value of x%d: ",i);
   scanf("%f",&x[i]);
   printf("\n\nenter the value of f(x%d): ",i);
   scanf("%f",&y[k][i]);
  }

  printf("\n\nEnter X for finding f(x): ");
  scanf("%f",&p);

  for(i=1;i<n;i++)
  {
    for(j=0;j<n-i;j++)
    {
     y[i][j]=y[i-1][j+1]-y[i-1][j];
    }
  }
  printf("\n_____\n");
  printf("\n  x(i)\t   y(i)\t    y1(i)    y2(i)    y3(i)    y4(i)");
  printf("\n_____\n");
  for(i=0;i<n;i++)

{
```

```
    printf("\n %.3f",x[i]);
    for(j=0;j<n-i;j++)
    {
     printf("    ");
     printf(" %.3f",y[j][i]);
    }
   printf("\n");
  }
  i=0;
  do
  {
   if(x[i]<p && p<x[i+1])
    k=1;
   else
    i++;
  }while(k != 1);
  f=i;
  u=(p-x[f])/(x[f+1]-x[f]);
  printf("\n\n u = %.3f ",u);

  n=n-i+1;
  sum=0;
  for(i=0;i<n-1;i++)
  {

   temp=1;
   for(j=0;j<i;j++)
   {
    temp = temp * (u - j);
   }
    m=fact(i);
    sum = sum + temp*(y[i][f]/m);
  }
  printf("\n\n f(%.2f) = %f ",p,sum);
  getch();
}

float fact(int a)
{
  float fac = 1;

  if (a == 0)
   return (1);
  else
   fac = a * fact(a-1);

  return(fac);}
```

**OUT PUT**

```
how many record you will be enter: 5
enter the value of x0: 2
enter the value of f(x0): 9
enter the value of x1: 2.25
enter the value of f(x1): 10.06
```

enter the value of x2: 2.5
enter the value of f(x2): 11.25
enter the value of x3: 2.75
enter the value of f(x3): 12.56
enter the value of x4: 3
enter the value of f(x4): 14
Enter X for finding f(x): 2.35

| x(i) | y(i) | y1(i) | y2(i) | y3(i) | y4(i) |
|---|---|---|---|---|---|
| 2.000 | 9.000 | 1.060 | 0.130 | -0.010 | 0.020 |
| 2.250 | 10.060 | 1.190 | 0.120 | 0.010 | |
| 2.500 | 11.250 | 1.310 | 0.130 | | |
| 2.750 | 12.560 | 1.440 | | | |
| 3.000 | 14.000 | | | | |

u = 0.400
f(2.35) = 10.522240

**APPLICATION: Students will learn about the concepts of Newton forward method of interpolation and will implement the details in c programming language.**

**Program10 :**

**Objective : -Write a program to implement Newton backward method of interpolation**.

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Theory:-** Suppose x and y are two variable and their relations can be expressed as y=f(x) x1<=x<=xn. Then we say that x is an independent variable & y is a dependent variable. When the form of f(x)is known , then the value of y can be computed directly corresponding to any value of x in the range x1<=x<=xn.

However, if the form of f(x) is not known and only a set of values (x1, y1), (x2, y2), (x3,y3)……..(xn, yn) satisfying the relation y=f(x) are known, then the process of estimating (approximating)the value of dependent variable y for a given value of independent variable x in the range x1<=x<=xn is known as interpolation.

Backward Difference- If y1,y2,y3……yn denote t he value of the function of type y = f(x) at x1, x2, x3, ……xn, then y2-y1, y3-y2, y4-y3, ……..yn- yn-1 are called the backward differences of y. These differences are denoted as dely2, dl y3, dely4, ……..del yn.

**Algorithm**:-

**Step-1.** Start of the program.

**Step-2.** Input number of terms n

**Step-3.** Input the array ax

**Step-4.** Input the array ay

**Step-5.** h=ax[1]-ax[0]

**Step-6.** for i=0; i<n–1; i++

**Step-7.** diff[i][1]=ay[i+1]–ay[i]

**Step-8.** End Loop i

**Step-9.** for j = 2; j < = 4; j + +

**Step-10.** for i=0; i<n–j; i++

**Step-11.** diff[i][j]=diff[i+1][j–1]–diff [i][j–1]

**Step-12.** End Loop i

**Step-13.** End Loop j

**Step-14.** i=0

**Step-15.** Repeat Step 16 until (!ax[i]<x)

**Step-16.** i=i+1

**Step-17.** x0=mx[i]

**Step-18.** sum=0

**Step-19.** y0=my[i]

**Step-20.** fun=1

**Step-21.** p=(x–x0)/h

**Step-22.** sum=y0

**Step-23.** for k=1; k<=4; k++

**Step-24.** fun=(fun*(p–(k–1)))/k

**Step-25.** sum=sum+fun*diff[i][k]

**Step-26.** End loop k

**Step-27.** Print Output x,sum

**Step-28.** End of Program

**PROGRAM: NEWOTN'S BACKWARD METHOD OF INTERPOLATION**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
 float x[10],y[10][10],sum,p,u,temp;
 int i,n,j,k=0,f,m;
 float fact(int);

 clrscr();
 printf("\nhow many record you will be enter: ");
 scanf("%d",&n);
 for(i=0; i<n; i++)
 {
  printf("\n\nenter the value of x%d: ",i);
  scanf("%f",&x[i]);
  printf("\n\nenter the value of f(x%d): ",i);
  scanf("%f",&y[k][i]);
 }
 printf("\n\nEnter X for finding f(x): ");
 scanf("%f",&p);

 for(i=1;i<n;i++)
 {
   for(j=i;j<n;j++)
   {
    y[i][j]=y[i-1][j]-y[i-1][j-1];
   }
 }


 printf("\n_____\n");


 printf("\n  x(i)\t   y(i)\t    y1(i)    y2(i)    y3(i)    y4(i)");
 printf("\n_____\n");
```

```
 for(i=0;i<n;i++)
 {
  printf("\n %.3f",x[i]);
  for(j=0;j<=i;j++)
  {
   printf("    ");
   printf(" %.3f",y[j][i]);
  }
 printf("\n");
 }

 i=0;
 do
 {
  if(x[i]<p && p<x[i+1])
   k=1;
  else
   i++;

 }while(k != 1);
 f=i+1;
 u=(p-x[f])/(x[f]-x[f-1]);
 printf("\n\n u = %.3f ",u);

 n=n-i+1;
 sum=0;
 for(i=0;i<n;i++)
 {
  temp=1;
  for(j=0;j<i;j++)
  {
   temp = temp * (u + j);
  }
   m=fact(i);
   sum = sum + temp*(y[i][f]/m);
 }
 printf("\n\n f(%.2f) = %f ",p,sum);
 getch();
}

float fact(int a)
{
 float fac = 1;

 if (a == 0)
  return (1);
 else
  fac = a * fact(a-1);

 return(fac);
}
```

**OUT PUT**

how many record you will be enter: 5
enter the value of x0: 2.5
enter the value of f(x0): 9.75
enter the value of x1: 3
enter the value of f(x1): 12.45
enter the value of x2: 3.5
enter the value of f(x2): 15.70
enter the value of x3: 4
enter the value of f(x3): 19.52
enter the value of x4: 4.5
enter the value of f(x4): 23.75
Enter X for finding f(x): 4.25

| x(i) | y(i) | y1(i) | y2(i) | y3(i) | y4(i) |
|---|---|---|---|---|---|
| 2.500 | 9.750 | | | | |
| 3.000 | 12.450 | 2.700 | | | |
| 3.500 | 15.700 | 3.250 | 0.550 | | |
| 4.000 | 19.520 | 3.820 | 0.570 | 0.020 | |
| 4.500 | 23.750 | 4.230 | 0.410 | -0.160 | -0.180 |

u = -0.500
f(4.25) = 21.583750

**APPLICATION: Students will learn about the concepts of Newton backward method of interpolation and will implement the details in c programming language.**

**Program11 :**

**Objective:** Write a program to implement gauss forward interpolation formula

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Theory: -** Interpolation refers to the process of creating new data points given within the given set of data. The Gaussian interpolation comes under the Central Difference Interpolation Formulae which differs from Newton's Forward interpolation formula.
Suppose we are given the following value of y=f(x) for set values of x:

X: x0 x1 x2 ……. xn
Y: y0 y1 y2 …… yn
Then the process of finding the value of y corresponding to any value of x=xi between x0 and xn is called interpolation. Thus interpolation is the technique of estimating the value of a function for any intermediate value of the independent variable while the process of computing the value of the function outside the given range is called extrapolation.

Suppose that the function y=f(x) is tabulated for the equally spaced values x = x0, x = x0+h , x = x0+2h ,…… x = x0+nh, giving y= y0 , y1, y2,……, yn. . To determine the value of f(x) or f'(x) for some intermediate values of x, the following three types of differences are found useful.
1. Forward Difference
2. Backward difference
3. Central Difference

The common Newton's forward formula belongs to the Forward difference category. However , the Gaussian forward formula belongs to the central difference method.
Gauss forward formula is derived from Newton's forward formula .

**Algorithm:-**

**Step-1.** Start of the program.

**Step-2.** Input number of terms n

**Step-3.** Input the array ax

**Step-4.** Input the array ay

**Step-5.** h=ax[1]-ax[0]

**Step-6.** for i=0;i<n–1;i++

**Step-7.** diff[i][1]=ay[i+1]-ay[i]

**Step-8.** End Loop i

**Step-9.** for j=2;j<=4;j++

**Step-10.** for i=0;i<n–j;i++

**Step-11.** diff[i][j]=diff[i+1][j–1]–diff[i][j–1]

**Step-12.** End Loop i

**Step-13.** End Loop j

**Step-14.** i=0

**Step-15.** Repeat Step 16 until ax[i]<x

**Step-16.** i=i+1

**Step-17.** i=i–1;

**Step-18.** p=(x–ax[i])/h

**Step-19.** y1=p*diff[i][1]

**Step-20.** y2=p*(p–1)*diff[i–1][2]/2

**Step-21.** y3=(p+1)*p*(p-1)*diff[i–2][3]/6

**Step-22.** y4=(p+1)*p*(p–1)*(p–2)*diff[i–3][4]/24

**Step-23.** y=ay[i]+y1+y2+y3+y4

**Step-24.** Print Output x,y

**Step-25.** End of Program

**PROGRAM: GAUSS'S FORWORD METHOD OF INTERPOLATION**

```
# include <stdio.h>
# include <conio.h>
# include <math.h>
# include <process.h>
# include <string.h>
void main()
{
int n;
int i,j;
float ax[10];
float ay[10];
float x;
float nr,dr;
float y=0; float h;
float p;
float diff[20][20];
float y1,y2,y3,y4;
clrscr();
printf(" Enter the number of terms -      ");
scanf("%d",&n);

printf("\n Enter the value in the form of x - ");
for (i=0;i<n;i++)
{

printf(" Enter the value of x%d -  ",i+1);
scanf("%f",&ax[i]);
}
printf(" Enter the value in the form of y -      ");
for(i=0;i<n;i++)
```

```
{
printf("Enter the value of y%d  -  ",i+1);
scanf("%f",&ay[i]);
}
printf("\nEnter the value of x for  -    ");
printf("\nwhich you want the value of y -     ");
scanf ("%f",&x);
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];
}
}
i=0; do { i++;
}
while(ax[i]<x); i--; p=(x-vx[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;
printf("\nwhen x=%6.4f,y=%6.8f ",x,y); getch();
}
```

**APPLICATION: Students will learn about the concepts of gauss forward method of interpolation and will learn the way of coding it in c programming language.**

**Program12 :**

**Objective: Program to implement Gauss's Backward interpolation formula**

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Theory**:- - Interpolation refers to the process of creating new data points given within the given set of data. The gaussian interpolation comes under the Central Difference Interpolation Formulae which differs from Newton's Forward interpolation formula.
Suppose we are given the following value of y=f(x) for set values of x:

X: x0 x1 x2 ……. xn
Y: y0 y1 y2 …… yn
Then the process of finding the value of y corresponding to any value of x=xi between x0 and xn is called interpolation. Thus interpolation is the technique of estimating the value of a function for any intermediate value of the independent variable while the process of computing the value of the function outside the given range is called extrapolation.

Suppose that the function y=f(x) is tabulated for the equally spaced values x = x0, x = x0+h , x = x0+2h ,…… x = x0+nh, giving y= y0 , y1, y2,……, yn. . To determine the value of f(x) or f'(x) for some intermediate values of x, the following three types of differences are found useful.
1. Forward Difference
2. Backward difference
3. Central Difference
The common Newton's forward formula belongs to the Forward difference category. However , the Gaussian forward formula belongs to the central difference method.
Gauss forward formula is derived from Newton's forward formula .

**Algorithm**

**Step-1.** Start of the program.

**Step-2.** Input number of terms n

**Step-3.** Input the array ax

**Step-4.** Input the array ay

**Step-5.** h=ax[1]-ax[0]

**Step-6.** for i=0;i<n-l;i++

**Step-7.** diff[i][1]=ay[i+1]-ay[i]

**Step-8.** End Loop i

**Step-9.** for j=2;j<=4;j++

**Step-10.** for i=0;i<n–j;i++

**Step-11.** diff[i][j]=diff[i+1][j–1]–diff[i][j–1]

**Step-12.** End Loop i

**Step-13.** End Loop j

**Step-14.** i=0

**Step-15.** Repeat Step 16 until ax[i]<x

**Step-16.** i=i+1

**Step-17.** i=i–1;

**Step-18.** p=(x–ax[i])/h

**Step-19.** y1=p*diff[i-1][1]

**Step-20.** y2=p*(p+1)*diff[i–1][2]/2

**Step-21.** y3=(p+1)*p*(p-1)*diff[i–2][3]/6

**Step-22.** y4=(p+2)*(p+1)*p*(p–1)*diff[i–3][4]/24

**Step-23.** y=ay[i]+y1+y2+y3+y4

**Step-24.** Print Output x,y

**Step-25.** End of Program

**PROGRAM: GAUSS'S BACKWARD METHOD OF INTERPOLATION.**

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

# include <process.h>
# include <string.h>
void main()
{
int n;
int i,j; float ax[10];
 float ay[10];
float x;
float y=0;
float h;
float p;
float diff[20][20];
float y1,y2,y3,y4;
clrscr();
printf("\n Enter the number of terms -      ");
scanf("%d",&n);
printf("\n Enter the value in the form of x  -      ");
for (i=0;i<n;i++)
{
printf("\n\n Enter the value of x%d     - ",i+1);
scanf("%f",&ax[i]);
}
printf("\n\n Enter the value in the form of y  -  ");
for(i=0;i<n;i++)
{
printf("\n Enter the value of y%d    -   ",i+1);
```

```
scanf("%f",&ay[i]);
}
printf("\nEnter the value of x for    -  ");
printf("\nwhich you want the value of y     -  ");
scanf("%f",&x);
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++)
{

diff[i][j]=diff[i+1][j-1]-diff[i][j-1];
}
}
i=0; do { i++;
}
while (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+2)*(p+1)*p*(p-1)*diff[i-3][4]/24;
y=ay[i]+y1+y2+y3+y4;
printf("\nwhen x=%6.1f,y=%6.4f ",x,y);
getch();
}
```

**APPLICATIONS: Students will learn the concepts of Gauss backward method of interpolation and will be able to implement the details in c programming language.**

**Program 13**

**Objective : LAGRANGE'S INTERPOLATION FORMULA.**

| PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 2 | | | | | | | | 2 |

| PSO1 | PSO2 | PSO3 | PSO4 |
|---|---|---|---|
| 2 | 2 | | |

**Theory:-** The main uses of Langrangian polynomial are-

(1) To find any value of a function when the tabulated value of the function are not equidistant.

(2) To find the value of the independent variable corresponding to a given value of a function.

**Algorithm:-**

**Step-1.** Start of the program

**Step-2.** Input number of terms n

**Step-3.** Input the array ax

**Step-4.** Input the array ay

**Step-5.** for i=0; i<n; i++

**Step-6.** nr=1

**Step-7.** dr=1

**Step-8.** for j=0; j<n; j++

**Step-9.** if j !=i

**a.** nr=nr*(x-ax[j])

**Step-10.** b.dr*(ax[i]-ax[j])

**Step-11.** End Loop j

**Step-12.** y+=(nr/dr)*ay[i]

**Step-13.** End Loop i

**Step-14.** Print Output x, y

**Step-15.** End of Program

**PROGRAM: LAGRANGE'S INTERPOLATION FORMULA.**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
  float x[10],y[10],temp=1,f[10],sum,p;
  int i,n,j,k=0,c;
  clrscr();
```

```
 printf("\nhow many record you will be enter: ");
 scanf("%d",&n);
 for(i=0; i<n; i++)
 {
  printf("\n\nenter the value of x%d: ",i);
  scanf("%f",&x[i]);

  printf("\n\nenter the value of f(x%d): ",i);
  scanf("%f",&y[i]);
 }
 printf("\n\nEnter X for finding f(x): ");
 scanf("%f",&p);

 for(i=0;i<n;i++)
 {
   temp = 1;
   k = i;
   for(j=0;j<n;j++)
   {
     if(k==j)
     {
       continue;
     }
     else
     {
       temp = temp * ((p-x[j])/(x[k]-x[j]));
     }
   }
   f[i]=y[i]*temp;
 }

 for(i=0;i<n;i++)
 {
    sum = sum + f[i];
 }
 printf("\n\n f(%.1f) = %f ",p,sum);
 getch();
}
```

**OUT PUT**

```
how many record you will be enter: 4
enter the value of x0: 0
enter the value of f(x0): 0
enter the value of x1: 1
enter the value of f(x1): 2
enter the value of x2: 2
enter the value of f(x2): 8
enter the value of x3: 3
enter the value of f(x3): 27
Enter X for finding f(x): 2.5
 f(2.5) = 15.312500
```

**APPLICATIONS: Students will learn the concepts of Lagrange's interpolation formula and will implement the details in c programming language.**