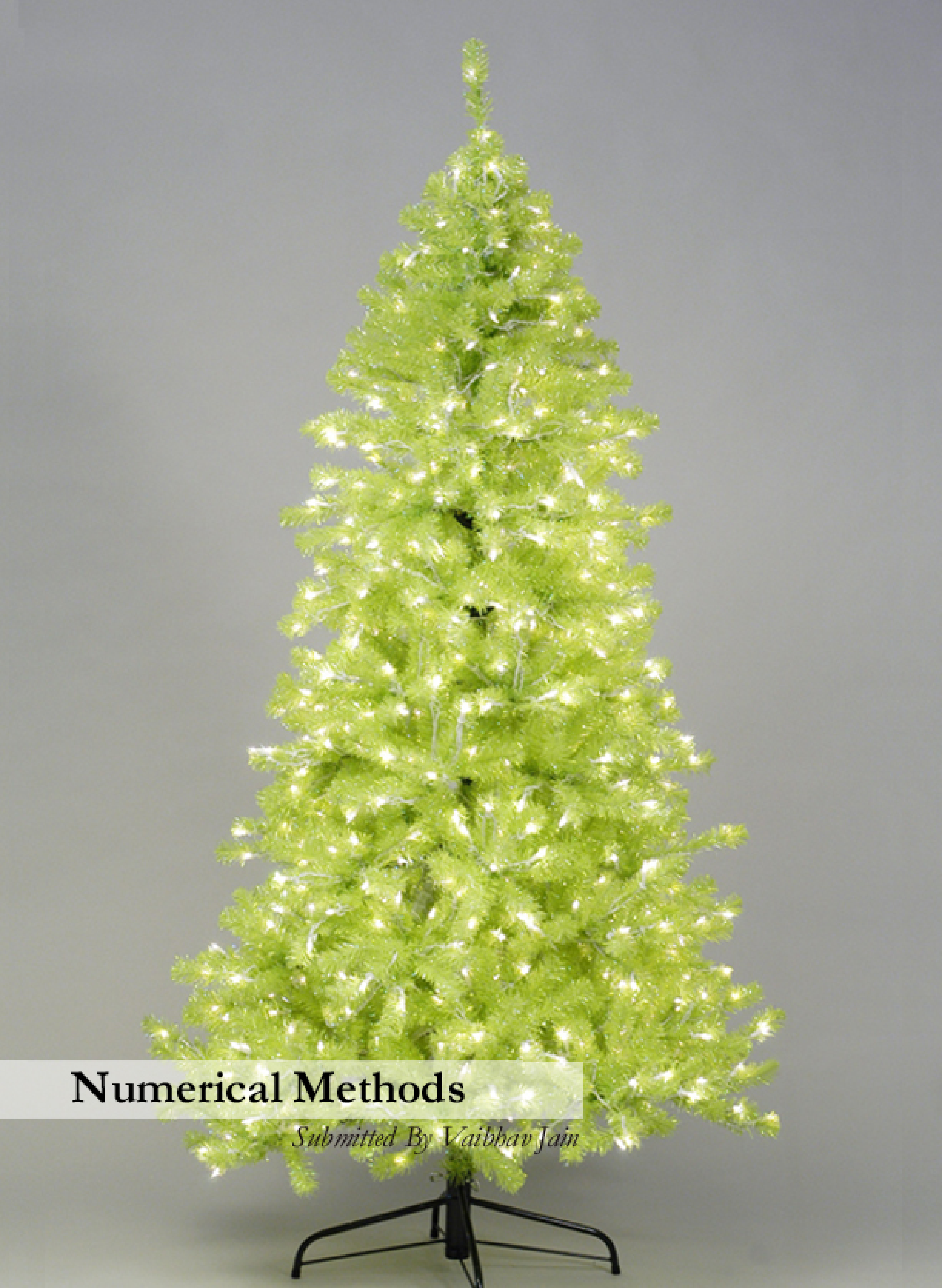


Numerical Methods

Submitted By Vaibhav Jain

PRACTICAL IN COMPUTER ORIENTED NUMERICAL METHODS USING 'C'

Submitted By

Vaibhav Jain

Student

Bachelor of Computer Applications

Swati Jain Institute Of Management Studies.

2001-2004.

Head Office	**Institute Campus**
182, Jaora Compound, **Indore.**	**33, Sampat Farms,** **Bicholi Mardana,** **Indore**
SJA Swati Jain Academy (SJA)	SJIMS Swati Jain Institute of Management Studies (SJIMS)

Acknowledgments

No man is born complete and I am no exception. When the times were tensed and it seemed like I should kick the whole bunch of meaningless symbols and code into the recycle bin, pour some water on my keyboard and throw the book away for good, all that could sustain me was the support of teachers, friends and elders. I was lucky enough to be surrounded by such a people who were helpful and supportive. Without their help this Project File would have probably completed on my 75^{th} birthday.

In am greatly thankful to Mr. Suyash Jain, our Teacher with out whose dedicated guidance and support this project would have being non existent.

I am also exceedingly thankful to the member faculty at Swati Jain Institute Of Management Studies as well as Swati Jain Madam who were there when their support and that wonderful sense of humor was badly and eagerly needed. Thanks, I can never forget you all.

And, finally a word of gratitude to my Parents and brother Ronak, who were always there with their support and encouragement, even though I'm a little crazy at times.

Vaibhav Jain

vaibhav@genesisconvent.com
57, Shiv Shahkti Nagar,
Kanadia Road,
Indore.
May 2004.

CERTIFICATE

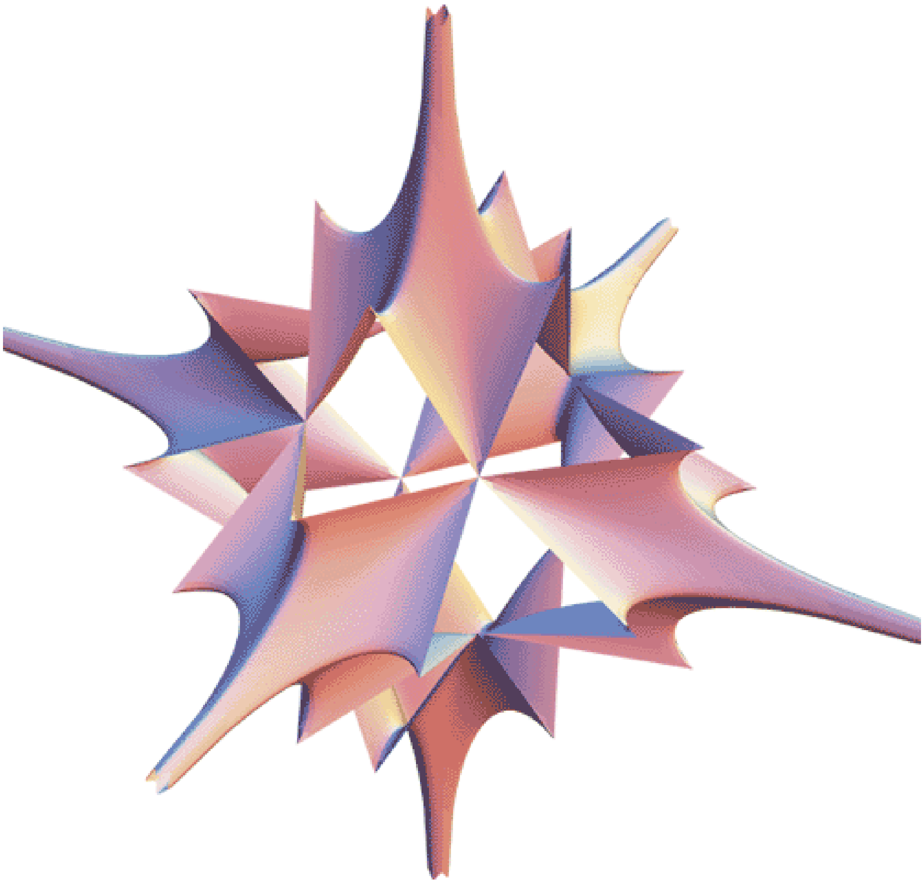
This is to certify that **Vaibhav Jain** an enrollee of Bachelor of Computer Application and a student **Swati Jain Institute of Management Studies** has worked on the project **"Practical in Computer Oriented Numerical Methods".** He has put sincere effort in the project and has performed tasks related to the project in the Computer Lab of **Swati Jain Institute of Management Studies**. This project may be considered as a partial fulfillment for the examinations conducted by **Devi Ahilya Vishva Vidyalaya, Indore**.

Date:

Mr. Suyash Jain	External

Index.

Sl #	Topic	Page no.
1.	The Bisection Method	1
2.	The False Position Method	2
3.	Newton-Raphlson Method	3
4.	The Secant Method	4
5.	Binary Addition & Subtraction	5
6.	Gauss Elimination Method..	6
7.	Gauss-Jordan Elimination Method..	8
8.	Gauss-Seidal Iterative Method..	10
9.	Jacobi’s Iterative Method..	12
10.	Polynomial Curve Fitting Method..	14
11.	Exponential Curve Fitting Procedure	16
12.	Interpolation With Langrage’s Polynomial	17
13.	Newton’s Dividend Interpolation Formulae	18
14.	Newton’s Forward Interpolation Formulae	19
15.	Newton's Backward Interpolation Formulae	20
16.	Piece-Wise Linear Fit Method	21
17.	Integration With Trapezoidal Rule	22
18.	Integration With Simpson’s 1/3 Rule	23
19.	Integration With Simpson's 3/8 Rule	24
20.	Euler’s Method For Differential Equations	25
21.	Second Order Runge-Kutta Method	26
22.	Fourth Order Runge-Kutta Method	27



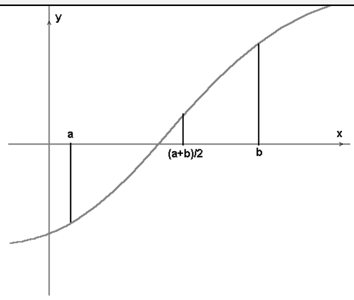
Cup To Round: Courtesy Mathworld.Wolfarm.com

Numerical Methods

The Bisection Method

Objective

To find the root of the equation F(x)=X^2-9 using the bisection method



Theory

Given a continuous function F(x) whose root are to be determined and let there be two points a and b such that

F(a)>0 and F(b)<0

Or F(a)<0 and F(b)>0

Than X'= (a+b)/2

Code (Bisect.cpp)

```
#include <math.h>
#include <stdio.h>
#include <conio.h>
#define SIGN(x) ((x<0.0)?-1:1)
#define APPROX 0.00001
double fx(double x)
{return (x*x-9);}

int getpointwithsign(int sign)
{int i=0;
for(i=0;i<=1000;i++)
 {if(SIGN(fx(i))==sign) return i;
  if(SIGN(fx(-i))==sign) return -i;
 }
return 0;
}

int main()
{double hi=0,lo,fhi,x,y;
 unsigned iterations=0;
 clrscr();
 hi=0,fhi=fx(0.0);
 lo=getpointwithsign(SIGN(fhi)*-1);
 if(lo==0) {puts("\nInvalid Function");return 1;}
 while(SIGN(hi-lo)*(hi-lo)>APPROX)
        {x=(hi+lo)/2;
         y=fx(x);
         if(y==0.0) break;
         else
         if(SIGN(y)==SIGN(fhi)) hi=x,fhi=y;
         else lo=x;
         iterations++;
         printf("\nIteration %d: y=%f",iterations,x);
        }
printf("\nThe Root of the Equation is :%f\nTotal
Iterations=%u",x,iterations);
getch();
return 0;}
//***code concludes*****
```

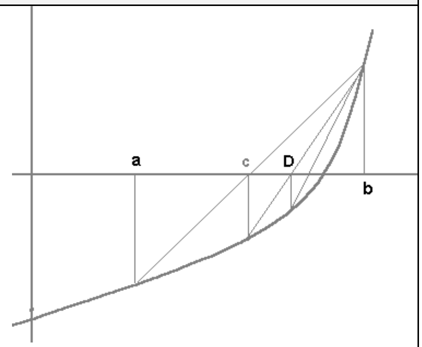
Output

Iteration 1: y=1.500000
Iteration 2: y=2.250000
Iteration 3: y=2.625000
Iteration 4: y=2.812500
Iteration 5: y=2.906250
Iteration 6: y=2.953125
Iteration 7: y=2.976562
Iteration 8: y=2.988281
Iteration 9: y=2.994141
Iteration 10: y=2.997070
Iteration 11: y=2.998535
Iteration 12: y=2.999268
Iteration 13: y=2.999634
Iteration 14: y=2.999817
Iteration 15: y=2.999908
Iteration 16: y=2.999954
Iteration 17: y=2.999977
Iteration 18: y=2.999989
Iteration 19: y=2.999994
The Root of the Equation is :2.999994
Total Iterations=19

The False Position Method

Objective

To find the root of following function correct up to 4 decimal places using the False Position Method. $F(x)=X^2-25$.



Theory

Let F(x) be a function whose roots are to be found and points x0 & x1 are two approximations to its root than the next approximation to the root is given by:

$x_2= [x_0\ f(x_1) – x_1\ f(x_0)] / (f(x_1)-f(x_0))$

Code (Flspos.cpp)

```
// root of a equation with false position method;
#include <math.h>
#include <stdio.h>
#include <conio.h>
#define SIGN(x) ((x<0.0)?-1:1)
#define APPROX 0.00003
double fx(double x)
{return x*x-25;}

int main()
{double x1=7,y1,x2=4,y2;
 double x,y;
 unsigned iterations=0;
 y1=fx(x1);
 y2=fx(x2);
 clrscr();
 if(SIGN(y2)==SIGN(y1))
        {fprintf(stderr,"\n!!Error:Invalid Domain Provided");
        return 1;}

 while(fabs(y1)>APPROX)
        {x=-y1*(x2-x1)/(y2-y1)+x1;
         y1=fx(x);
         x1=x;
         iterations++;
         printf("\nIteration %d: y=%f",iterations,x);
        }
printf("\nThe Root of the Equation is :%f\nTotal
Iterations=%u"   ,x,iterations);
getch();
return 0;
}
//***code concludes*****
```

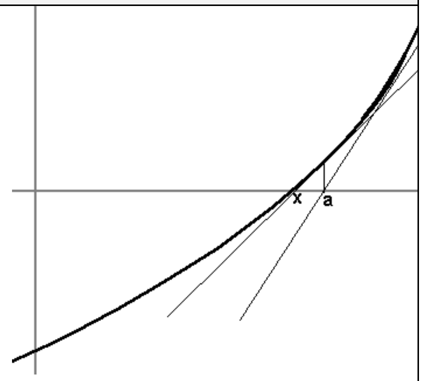
Output

Iteration 1: y=4.818182
Iteration 2: y=5.020619
Iteration 3: y=4.997714
Iteration 4: y=5.000254
Iteration 5: y=4.999972
Iteration 6: y=5.000003
Iteration 7: y=5.000000
The Root of the Equation is :5.000000
Total Iterations=7

Newton-Raphlson Method

Objective

To find the root of following function correct up to 2 decimal places using the Newton Raphlson Method. $F(x)=X^2-4X+4$



Theory

This method provides the fastest convergence to the root of the given equation F(X). Let Point F(x1),x1 be any point on this curve and F'(x1) be the slope of tangent on this point than the next approximation to the root is given by

$x_2 = x_1 - F(x_1)/ F'(x_1);$

Code (Newtraph.cpp)

```
//CONM
// root of a equation with newton raphalson method;
//06/02/03 (c) Vaibhav Jain
#include <math.h>
#include <stdio.h>
#include <conio.h>
#define SIGN(x) ((x<0.0)?-1:1)
#define APPROX 0.00003
double fx(double x)
        {return (x*x-4*x+4);}

double dfx(double x)
        {return (2*x-4);}
int main()
{double x=10,y,dy;
unsigned iterations=0;
clrscr();
y=fx(x); dy=dfx(x);
while(fabs(y)>APPROX)
{x=x-y/dy;
y=fx(x);
dy=dfx(x);
iterations++;
printf("Iteration%d: x=%f\n",iterations,x); }
printf("\nThe Root of the Equation is :%f\nTotal
Iterations=%u",x,iterations);
getch();
return 0;
}
//***code concludes*****
```

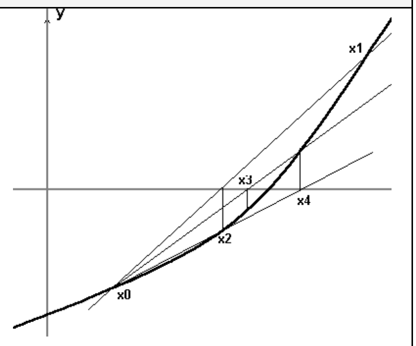
Output

```
Iteration1: x=6.000000
Iteration2: x=4.000000
Iteration3: x=3.000000
Iteration4: x=2.500000
Iteration5: x=2.250000
Iteration6: x=2.125000
Iteration7: x=2.062500
Iteration8: x=2.031250
Iteration9: x=2.015625
Iteration10: x=2.007812
Iteration11: x=2.003906

The Root of the Equation is :2.003906
Total Iterations=11
```

The Secant Method

Objective

Using the Secant method find the root of the Equation F(x)=X2-4X+4



Theory

This method is an approximation of Newton Raphlson method but the calculations required are much less than it. The slope of the curve at a point is approximated. The next approximate to the root is given by :

$x_2 = x_1 - F(x_1) \, . \, (x_2 - x_1) / [F(x_2) - F(x_1)];$

Code (Secant.cpp)

```
// root of a equation with secant method;
//06/02/2004 (c) Vaibhav Jain
#include <math.h>
#include <stdio.h>
#include <conio.h>
#define SIGN(x) ((x<0.0)?-1:1)
#define APPROX 0.0000001

double fx(double x)
        {return (x*x-4*x+4);}

int main()
{double x1=5,x2,y1,y2,x,y=APPROX;
 unsigned iterations=0;
 y1=fx(x1);x2=x1-1;y2=fx(x2);clrscr();
 clrscr();
 while(fabs(y)>=APPROX)
        {x=x1-y1/(y1-y2)*(x1-x2);
         y=fx(x);
         x2=x1,y2=y1;
         x1=x,y1=y;
         iterations++;
         printf("Iteration%d: x=%f\n",iterations,x);
        }
printf("\nThe Root of the Equation is :\
        %f\nTotal Iterations=%u",x,iterations);
getch();

return 0;
}
//***code concludes*****
```

Output

```
Iteration1: x=3.200000
Iteration2: x=2.857143
Iteration3: x=2.500000
Iteration4: x=2.315789
Iteration5: x=2.193548
Iteration6: x=2.120000
Iteration7: x=2.074074
Iteration8: x=2.045802
Iteration9: x=2.028302
Iteration10: x=2.017493
Iteration11: x=2.010811
Iteration12: x=2.006682
Iteration13: x=2.004129
Iteration14: x=2.002552
Iteration15: x=2.001577
Iteration16: x=2.000975
Iteration17: x=2.000602
Iteration18: x=2.000372
Iteration19: x=2.000230

The Root of the Equation is :   2.000230
Total Iterations=19
```

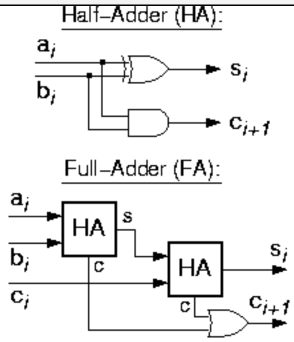
Binary Addition & Subtraction

Objective

To implement the algorithm of binary addition in C++ and to add two numbers and store there addition result in third.

Theory

Binary addition is performed by doing a bit scan from right to the left of the operands and doing a Bitwise XOR of individual bits with the carry bit. The result is pushed into the result and the carry bit is set if any two of the three operand bits were 1.



Code (Binadd.cpp)

```
//conm
//binary addition/subtraction of two integers
//with out using addition operator
//08/02/2004 (c) Vaibhav Jain
#include <stdio.h>
#include <conio.h>
void main()
{unsigned a=7,b=2,c=0;
 unsigned char carry=0,sum=0;
 unsigned count;

//do the loop 16 times;

for(count=~0;count;count>>=1)
 {  sum=((a&1)^(b&1)^carry);
    c|=sum<<(sizeof(int)*8-1);
    carry= (a&carry&&!(b&1))||
           (b&carry&&!(a&1))||
            ((a&b&1)&&
           !(carry))||
           ((a&b&1) &&
           carry);
    if(count==1) continue;
    a>>=1;b>>=1;c>>=1;
 }
printf("\nThe Sum is %d",c);

if(carry)printf(" With Overflow");

getch();
}
//***code concludes*****
```

Output

The sum is 9

Gauss Elimination Method..

Objective

To create an interactive program to accept an augment matrix from the user and solve it using Gauss Elimination method.

$$\begin{bmatrix} A11 & A12 & A13 & A14 & A15 \\ 0 & A22 & A23 & A24 & A25 \\ 0 & 0 & A33 & A34 & A35 \\ 0 & 0 & 0 & A44 & A45 \end{bmatrix}$$

Theory

The Gaussian elimination algorithm aims at converting a given augment matrix into its echelonic form and than performing back substitution to get the values of individual variables. In each iteration a pivot variable row is selected and it is eliminated from remaining rows through multiplication and subtraction. The resulting matrix is a triangular form of the original matrix.

Code (Gaussele.cpp)

```
//conm
//Solving a Linear Equation using
//Gauss Elemination Method
#include <conio.h>
#include <stdio.h>
float matrix[4][5];
int rows=0,cols=0;
int partialpivot(int row)// perform pivot on row with a non
zero row
{  for(int i=row;i<rows;i++)
    if(matrix[i][row]!=0) break;
  if(i>=rows) return 0; // pivot not possible
  for(int j=row;j<cols;j++)
  {float temp=matrix[row][j];
   matrix[row][j]=matrix[i][j];
   matrix[i][j]=temp;
  }
 return 1;
}
void main()
{ int cvars=0;
  clrscr();
  printf("*********************Gauss Elemination
Method*****************\n");
  printf("\nEnter Number of variables: ");
  scanf("%d",&cvars);
  printf("\nNow Enter the Augment Matrix:");
  for(int i=0;i<cvars;i++)
   {printf("\nEnter Row[%d]->",i);
   for(int j=0;j<cvars+1;j++)
   scanf("%f",&matrix[i][j]);
   }
  rows=cvars,cols=cvars+1;
  printf("\nMatrix::");
   for(i=0;i<rows;i++)
   {printf("\n");
     for(int j=0;j<cols;j++)
      printf("%.2f | ",matrix[i][j]);
   }
//***code continues*****
```

Output

```
******Gauss Elemination Method*****

Enter Number of variables: 3

Now Enter the Augment Matrix:
Enter Row[0]->1 1 1 3

Enter Row[1]->2 3 1 6

Enter Row[2]->1 -1 -1 -3

Matrix::
1.00 | 1.00 | 1.00 | 3.00 |
2.00 | 3.00 | 1.00 | 6.00 |
1.00 | -1.00 | -1.00 | -3.00 |

Solutions:-> |0.00 | 1.50 | 1.50 |
```

Gauss Elimination Method.

Code (Gaussele.cpp)

```
for(i=0;i<rows;i++)
{ if(matrix[i][i]==0 && partialpivot(i)==0)
   {// the first element is zero and pivoting is also not possible
    printf("\nThe System in Inconsistent or reductant");
    getch();
    return;
   }

 for(int j=cols-1;j>=0;j--)
            matrix[i][j]/=matrix[i][i]; //divide each element in row by first element
  for(j=i+1;j<rows;j++)//iterate through next rows
    for(int k=cols-1;k>=i;k--)//iterate through each element in this row
      matrix[j][k]-=matrix[i][k]*matrix[j][i];//subtract this element with
                                                //product of rows first element
                                                //by pivoting rows corrosponding
                                                //element
}
//upper trangulization is complete now perform back substitution
for(i=rows-1;i>=0;i--) // iterate through each row backward
{float sum=0;            //intialize subtractant to zero
 for(int j=i+1;j<cols-1;j++)// iterate through each non unity element
    sum+= matrix[i][j]*matrix[j][cols-1];// add to subtractend product of
                                          // of the element & its corrosponding
                                          //augument matrix element;
 matrix[i][cols-1]-=sum; // subtract subtractend from the current augument element
}
printf("\n\nSolutions:-> |");
 for(i=0;i<rows;i++)
    printf("%.2f | ",matrix[i][cols-1]);
getch();
}
//***code concludes*****
```

Gauss-Jordan Elimination Method..

Objective

To create an interactive program to accept an augment matrix from the user and solve it using Gauss Jordan Elimination method.

$$\begin{bmatrix} A11 & 0 & 0 & 0 & A15 \\ 0 & A22 & 0 & 0 & A25 \\ 0 & 0 & A33 & 0 & A35 \\ 0 & 0 & 0 & A44 & A45 \end{bmatrix}$$

Theory

This method is similar to the Gauss Elimination Method. However here the pivot variable is eliminated from all the other rows including the previous rows. The resulted matrix is an identity matrix and so there is no need for performing a back substitution on the individual rows of the matrix.

Code (Gaussjor.cpp)

```
//conm
//Solving a Linear Equation using
//Gauss Jordan Elemination Method
#include <conio.h>
#include <stdio.h>
float matrix[3][4];
int rows=0,cols=0;
int partialpivot(int row)// perform pivot on row with a non
zero row
{  for(int i=row;i<rows;i++)
    if(matrix[i][row]!=0) break;
  if(i>=rows) return 0; // pivot not possible
  for(int j=row;j<cols;j++)
  {float temp=matrix[row][j];
   matrix[row][j]=matrix[i][j];
   matrix[i][j]=temp;
  }
 return 1;
}
void main()
{ int cvars=0;
  clrscr();
  printf("*********************Gauss-Jordan Elemination
Method*****************\n");
  printf("\nEnter Number of variables: ");
  scanf("%d",&cvars);
  printf("\nNow Enter the Augment Matrix:");
  for(int i=0;i<cvars;i++)
   {printf("\nEnter Row[%d]->",i);
   for(int j=0;j<cvars+1;j++)
   scanf("%f",&matrix[i][j]);
   }
rows=cvars,cols=cvars+1;
  printf("\nAugument Matrix::");
   for(i=0;i<rows;i++)  {printf("\n");
     for(int j=0;j<cols;j++)
      printf("%.2f | ",matrix[i][j]);
   }
//***code continues*****
```

Output

```
****Gauss-Jordan Elemination Method**

Enter Number of variables: 3

Now Enter the Augment Matrix:
Enter Row[0]->2 4 2 15

Enter Row[1]->2 1 2 -5

Enter Row[2]->4 1 -2 0

Augument Matrix::
2.00 | 4.00 | 2.00 | 15.00 |
2.00 | 1.00 | 2.00 | -5.00 |
4.00 | 1.00 | -2.00 | 0.00 |

Solutions:-> |-3.06 | 6.67 | -2.78 |
```

Gauss-Jordan Elimination Method.

Code (Gaussjor.cpp)

```
for(i=0;i<rows;i++)
{ if(matrix[i][i]==0 && partialpivot(i)==0)
   {// the first element is zero and pivoting is also not possible
    printf("\nThe System in Inconsistent or reductant");
    getch();
    return;
   }
  for(int j=cols-1;j>=0;j--)
           matrix[i][j]/=matrix[i][i]; //divide each element in row by first element

  for(j=(i+1)%rows;j!=i;j=(j+1)%rows)//iterate through other rows
    for(int k=cols-1;k>=i;k--)//iterate through each element in this row
      matrix[j][k]-=matrix[i][k]*matrix[j][i];//subtract this element with
                                                    //product of rows first element
                                                    //by pivoting rows corresponding
                                                    //element
}

printf("\n\nSolutions:-> |");
 for(i=0;i<rows;i++)
    printf("%.2f | ",matrix[i][cols-1]);
getch();
}

//***code concludes*****
```

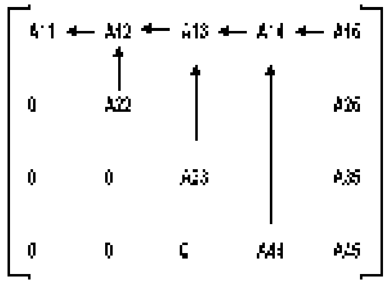
Gauss-Seidal Iterative Method..

Objective

To create an interactive program to accept an augment matrix from the user and solve it using Gauss Seidal Iterative method.

Theory

Gauss Seidal Iterative method is similar to the Jacobie's method. However in this method the new value of a variable is immediately employed to solve other equations. Thus a faster convergence to the root is made possible than the Jacobi's method.



Code (Gausssed.cpp)

```
//conm
//Solving a Linear Equation using
//Gauss Seidal Iterative Method
#include <conio.h>
#include <stdio.h>
float matrix[3][4];
int rows=0,cols=0;
int maxiterations;
void main()
{ int cvars=0;
  clrscr();
  printf("****Gauss Seidal Iterative Method****\n");
  printf("\nEnter Number of variables: ");
  scanf("%d",&cvars);
  printf("\nNow Enter the Augment Matrix:");
  for(int i=0;i<cvars;i++)
   {printf("\nEnter Row[%d]->",i);
   for(int j=0;j<cvars+1;j++)
   scanf("%f",&matrix[i][j]);
   }
  rows=cvars,cols=cvars+1;
  printf("\nMax Number of Iterations: ");
  scanf("%d",&maxiterations);
  printf("\nMatrix::");
   for(i=0;i<rows;i++)
   {printf("\n");
     for(int j=0;j<cols;j++)
      printf("%.2f | ",matrix[i][j]);
   }
//check if any cooficient is zero
for(i=0;i<rows;i++) if(matrix[i][i]==0)
            {for(int j=0;j<rows;j++)
    if(matrix[j][i]!=0 && matrix[i][j]!=0) break;
if(j>=rows)
   {printf("Invalid Matrix:one or more Cooficients are zero");
    return;}
 //perform rows interchange
 for(int k=0;k<cols;k++)
//***code continues*****
```

Output

```
Enter Number of variables: 3

Now Enter the Augment Matrix:
Enter Row[0]->9 2 4 20

Enter Row[1]->1 10 4 6

Enter Row[2]->2 -4 10 -15

Max Number of Iterations: 8

Matrix::
9.00 | 2.00 | 4.00 | 20.00 |
1.00 | 10.00 | 4.00 | 6.00 |
2.00 | -4.00 | 10.00 | -15.00 |

7:Solutions:-> |1.556 | 0.044 | -1.793 |
6:Solutions:-> |3.009 | 1.016 | -1.695 |
5:Solutions:-> |2.750 | 1.003 | -1.649 |
4:Solutions:-> |2.732 | 0.986 | -1.652 |
3:Solutions:-> |2.737 | 0.987 | -1.653 |
2:Solutions:-> |2.737 | 0.987 | -1.653 |
1:Solutions:-> |2.737 | 0.987 | -1.653 |
0:Solutions:-> |2.737 | 0.987 | -1.653 |

Solutions:-> |2.737 | 0.987 | -1.653 |
```

Gauss-Seidal Iterative Method.

Code (Gausssed.cpp)

```
              {float temp=matrix[i][k];
               matrix[i][k]=matrix[j][k];
               matrix[j][k]=temp;
              }
             }
for(i=0;i<rows;i++)  //make coffienent of one variabe in each
   { for(int j=0,coff=matrix[i][i];j<cols;j++)   //each equation =1
              matrix[i][j]/=coff;
     matrix[i][i]=0;
    }
//the initial guess of each variable =0
//now lets roll!!!
  while(maxiterations-->0)
  { for(int i=0;i<rows;i++)
     {float sum=0;
      matrix[i][i]=0; // let the new value intially be zero
      for(int j=0;j<cols-1;j++)
       sum+= matrix[i][j]*matrix[j][j];// multiply each coffiecent with its varible
       matrix[i][i]=matrix[i][cols-1]-sum;//new value= constant-(vars*coffs);
    }
  printf("\n\n%d:Solutions:-> |",maxiterations);
  for(int k=0;k<rows;k++)
    printf("%.3f | ",matrix[k][k]);
  getch();
 }
printf("\n\nSolutions:-> |");
 for(i=0;i<rows;i++)
    printf("%.3f | ",matrix[i][i]);
getch();
}

//***code concludes*****
```

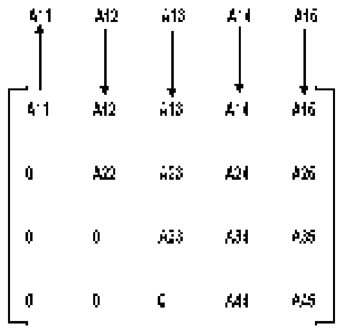
Jacobi's Iterative Method..

Objective

To create an interactive program to accept an augment matrix from the user and solve it using Gauss Seidal Iterative method.

Theory

Jordan's Iterative Method starts with an initial guess of all variables and then sequentially solved the whole set of variables. Each iterations gives new approximation of the variables. As the iterations proceed the approximation approaches the values of the variables.



Code (Jacobi.cpp)

```
//conm
//Solving a Linear Equation using
//Jacobi Iterative Method
#include <conio.h>
#include <stdio.h>
//maximum variable in an equation
#define VARSMAX 2
float matrix[VARSMAX][VARSMAX+1];
float vars[VARSMAX];
int rows=0,cols=0;
int maxiterations;
void main()
{ int cvars=0;
  clrscr();
  printf("****Gauss Seidal Iterative Method****\n");
  printf("\nEnter Number of variables: ");
  scanf("%d",&cvars);
  printf("\nNow Enter the Augment Matrix:");
  for(int i=0;i<cvars;i++)
   {printf("\nEnter Row[%d]->",i);
   for(int j=0;j<cvars+1;j++)
   scanf("%f",&matrix[i][j]);
   }
  rows=cvars,cols=cvars+1;
  printf("\nMax Number of Iterations: ");
  scanf("%d",&maxiterations);
  printf("\nMatrix::");
   for(i=0;i<rows;i++)
   {printf("\n");
     for(int j=0;j<cols;j++)
      printf("%.2f | ",matrix[i][j]);   }
//check if any cooficient is zero
for(i=0;i<rows;i++)
if(matrix[i][i]==0)
            {for(int j=0;j<rows;j++)
                 if(matrix[j][i]!=0 && matrix[i][j]!=0) break;
if(j>=rows)
{printf("Invalid Matrix:one or more Cooficients are zero");
//***code continues*****
```

Output

```
****Jordan Iterative Method****

Enter Number of variables: 3

Now Enter the Augment Matrix:
Enter Row[0]->9 2 4 20
Enter Row[1]->1 10 4 6
Enter Row[2]->2 -4 10 -15

Max Number of Iterations: 8

Matrix::
9.00 | 2.00 | 4.00 | 20.00 |
1.00 | 10.00 | 4.00 | 6.00 |
2.00 | -4.00 | 10.00 | -15.00 |
7:Solutions:-> |0.000 | 0.000 | 0.000 |
6:Solutions:-> |2.222 | 0.600 | -1.500 |
5:Solutions:-> |2.756 | 0.978 | -1.704 |
4:Solutions:-> |2.762 | 1.006 | -1.660 |
3:Solutions:-> |2.736 | 0.988 | -1.650 |
2:Solutions:-> |2.736 | 0.986 | -1.652 |
1:Solutions:-> |2.737 | 0.987 | -1.653 |
0:Solutions:-> |2.737 | 0.987 | -1.653 |
```

Jacobi's Iterative Method.

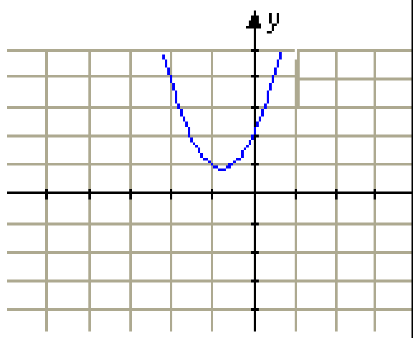
Code (Jacobi.cpp)

```
              return;}
            //perform rows interchange
            for(int k=0;k<cols;k++)
            {float temp=matrix[i][k];
             matrix[i][k]=matrix[j][k];
             matrix[j][k]=temp;
            }
           }
for(i=0;i<rows;i++)  //make coffienent of one variabe in each
   { for(int j=0,coff=matrix[i][i];j<cols;j++)   //each equation =1
            matrix[i][j]/=coff;
     matrix[i][i]=0;
    }
//the initial guess of each variable =0
//now lets roll!!!
  while(maxiterations-->0)
  { for(int i=0;i<rows;i++)
     {float sum=0;
      matrix[i][i]=0; // let the new value intially be zero
      for(int j=0;j<cols-1;j++)
       sum+= matrix[i][j]*vars[j];// multiply each coffiecent with its varible
       matrix[i][i]=matrix[i][cols-1]-sum;//new value= constant-(vars*coffs);
    }
  printf("\n%d:Solutions:-> |",maxiterations);
  for(int k=0;k<rows;k++)
    {printf("%.3f | ",vars[k]);
    vars[k]=matrix[k][k];//save new values
    }
  getch();
 }
}

//***code concludes*****
```

Polynomial Curve Fitting Method..

Objective

To fit a linear polynomial of second degree to a given set of data using the curve fitting method.



Theory

Given a set of tuples (X_i,Y_i) then the best fitting curve of degree N can be found by solving following set of equation for variables (a_0…a_N).

$\Rightarrow N.A_0+a_1.\Sigma x_i+….an.\ \Sigma x_i^n=\Sigma y_i$

$\Rightarrow A_0.\Sigma x+a_1.\Sigma x_i^2+….an.\ \Sigma x_i^{(n+1)}=\Sigma x_i.y_i$

….

$\Rightarrow A_0.\Sigma x^n+a_1.\Sigma x_i^{(n+1)}+….a_n.\ \Sigma x_i^{2n}=\Sigma x_i^n.y_i$

Code (Curvefit.cpp)

```
//conm
//least square curve fitting method
//fits a linear polynomial to a given set of data
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
float sumx[7]={0},sumy[3]={0};
float matrix[3][4];
int degree=0;
void GaussElemination(float **matrix,int cvars)
{ int rows=cvars,cols=cvars+1;
for(int i=0;i<rows;i++)
{ if(matrix[i][i]==0) exit(1);
  for(int j=cols-1;j>=0;j--)
            matrix[i][j]/=matrix[i][i]; //divide each element in row
by first element
  for(j=i+1;j<rows;j++)//iterate through next rows
    for(int k=cols-1;k>=i;k--)//iterate through each element in
this row
      matrix[j][k]-=matrix[i][k]*matrix[j][i];//subtract this element
with
}
for(i=rows-1;i>=0;i--) // iterate through each row backward
{float sum=0;           //intialize subtractant to zero
 for(int j=i+1;j<cols-1;j++)// iterate through each non unity
element
    sum+= matrix[i][j]*matrix[j][cols-1];// add to subtractend
product of
 matrix[i][cols-1]-=sum; // subtract subtractend from the
current augument element
}
}
void main()
{ printf("*************Curve Fitting Method*************\n");
  printf("\nEnter Degree of curve to fit: ");
  scanf("%d",&degree);
  printf("\nNow Enter the data set:\n");
 float x,y;char buffer[20];
//***code continues*****
```

Output

```
*****Curve Fitting Method*******

Enter Degree of curve to fit: 2

Now Enter the data set:
Enter Pair as x<space>y: -4 21
Enter Pair as x<space>y: -3 12
Enter Pair as x<space>y: -2 4
Enter Pair as x<space>y: -1 1
Enter Pair as x<space>y: 0 2
Enter Pair as x<space>y: 1 7
Enter Pair as x<space>y: 2 15
Enter Pair as x<space>y: 3 30
Enter Pair as x<space>y: 4 45
Enter Pair as x<space>y: 5 67
Enter Pair as x<space>y:

Solutions:->| 2.03 | 3.00 | 1.98 |
```

Polynomial Curve Fitting Method.

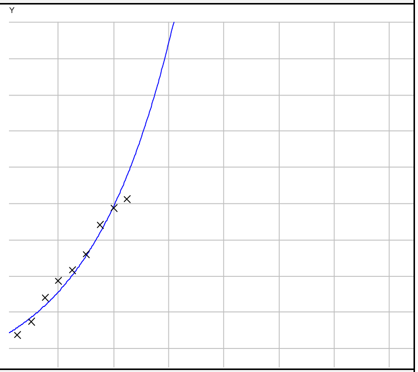
Code (Curvefit.cpp)

```
do
 {fflush(stdin);
  printf("Enter Pair as x<space>y: ");
  scanf("%[0-9e. -]s",buffer);
  sscanf(buffer,"%f %f",&x,&y);
  if(!*buffer) continue;
  float xp=1;
  for(int i=0;i<=2*degree;i++) // calculating the cooficient terms
  {sumx[i]+=xp;
   if(i<=degree) sumy[i]+=y*xp;
   xp*=x;
  }
 }while(*buffer);
for(int i=0;i<=degree;i++)  // forming the system matrix
{  for(int j=0;j<=degree;j++)
   matrix[i][j]=sumx[i+j];
   matrix[i][j]=sumy[i];
}
//solving the matrix using gaussian elemination
GaussElemination((float**)matrix,degree+1);
printf("\nSolutions:->| ");
for(i=0;i<=degree;i++)
printf("%.2f | ",matrix[i][degree+1]);
getch();
}

//***code concludes*****
```

Exponential Curve Fitting Procedure

Objective

To fit an exponential curve of form $y=ae^{-bx}$ on a given set of data using the curve fitting method.



Theory

Given a set of tuples (X_i,Y_i) then the best fitting curve of form $y=ae^{-bx}$ can be found by solving following set of equation for variables a_0,a_1).

$\Rightarrow N.a_0+a_1.\Sigma x_i=\Sigma \log y_i$

$\Rightarrow a_0.\Sigma x +a_1.\Sigma x_i^2 =\Sigma x_i.\log y_i$

thus, $a=e^{a_0}$ & $b=-a_1$

Code (Curvfit2.cpp)

```
//least square curve fitting method
//fits a exponential function of form y=ae^(-bx)
//to a given set of data
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>
float sumx[3]={0},sumy[2]={0};float matrix[2][3];
void GaussElemination(float **,int cvars)
{ int rows=cvars,cols=cvars+1;
for(int i=0;i<rows;i++)
{ if(matrix[i][i]==0) exit(1);
  for(int j=cols-1;j>=0;j--) matrix[i][j]/=matrix[i][i];
  for(j=i+1;j<rows;j++)    for(int k=cols-1;k>=i;k--)
      matrix[j][k]-=matrix[i][k]*matrix[j][i];}
for(i=rows-1;i>=0;i--) {float sum=0;
 for(int j=i+1;j<cols-1;j++)
 sum+= matrix[i][j]*matrix[j][cols-1]; matrix[i][cols-1]-=sum;
}}
void main()
{ printf("***Exponential Curve Fitting Method***\n");
  printf("\nEnter the data set:\n");
 float x,y;char buffer[20];
 do {fflush(stdin);
  printf("Enter Pair as x<space>y: ");
  scanf("%[0-9e. -]s",buffer);
  sscanf(buffer,"%f %f",&x,&y);
  if(!*buffer) continue;  y=log(y);
  sumx[0]++; sumx[1]+=x;  sumx[2]+=x*x;
sumy[0]+=y;  sumy[1]+=x*y;         }while(*buffer);
matrix[0][0]=sumx[0];  matrix[0][1]=sumx[1]; //sum(x)
matrix[0][2]=sumy[0]; matrix[1][0]=sumx[1]; //Sum(x)
matrix[1][1]=sumx[2]; matrix[1][2]=sumy[1]; //sum(x*y)
//solving the matrix using gaussian elemination
GaussElemination((float**)matrix,2);
printf("\nSolutions:-> ");
printf("a=%.2f , ",exp(matrix[0][2]));
printf("b=%.2f  ",-matrix[1][2]);getch();}
//***code concludes*****
```

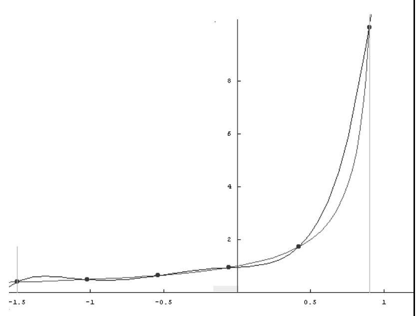
Output

```
***Exponential Curve Fitting Method***

Enter the data set:
Enter Pair as x<space>y: 1 5.5
Enter Pair as x<space>y: 2 7
Enter Pair as x<space>y: 3 9.6
Enter Pair as x<space>y: 4 11.5
Enter Pair as x<space>y: 5 12.6
Enter Pair as x<space>y: 6 14.4
Enter Pair as x<space>y: 7 17.6
Enter Pair as x<space>y: 8 19.5
Enter Pair as x<space>y: 9 20.5
Enter Pair as x<space>y:

Solutions:-> a=5.34 , b=-0.16
```

Interpolation With Langrage's Polynomial

Objective

To interpolate a given set of tabulated values to a given point using the Langrage's Interpolation Polynomial.



Theory

This is an explicit form of the polynomial *p* of degree *n* which interpolates a given function *f* at the points $x_0, x_1, x_2 .. xn$, and is given by :

$$p(x) = \sum_{i=0}^{n} \ell_i(x) f(x_i), \; \ell_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^{n} \left(\frac{x - x_k}{x_i - x_k} \right).$$

Code (Langrage.cpp)

```
//interpolation using lagrange`s method
#include <stdio.h>
#include <conio.h>
float x[10];
float fx[10];
int count=0;
float num,result;
float adder;
void main()
{int i=0,j;char buffer[20];
 clrscr();
 do
 {printf("Enter Pair as x<space>f(x): ");
  scanf("%[0-9e. -]s",buffer);
  fflush(stdin);
  sscanf(buffer,"%f %f",&x[i],&fx[i]);
  i++;
 }while(*buffer &&i<10);
count=i-1;

printf("\n\nEnter the value to be interpolated: ");
scanf("%f",&num);


for(result=i=0;i<count;i++)
{adder=fx[i];
 for(j=0;j<count;j++)
  if(j!=i)
        adder*=(num-x[j])/(x[i]-x[j]);
 result+=adder;
}
printf("\n Interpolation result for %f = %f",num,result);
getch();
}
//***code concludes*****
```

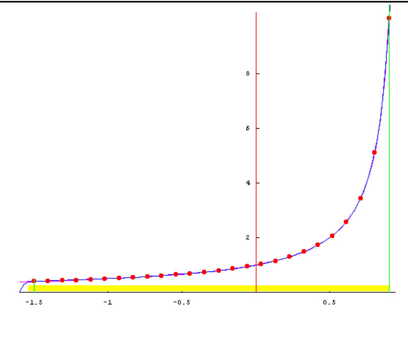
Output

```
Enter Pair as x<space>f(x): .4 -0.916
Enter Pair as x<space>f(x): .5 -.693
Enter Pair as x<space>f(x): .7 -.357
Enter Pair as x<space>f(x): .8 -.223
Enter Pair as x<space>f(x):


Enter the value to be interpolated: .6

Interpolation result for 0.600000 = -0.510167
```

Newton's Dividend Interpolation Formulae

Objective

To implement a program that accepts a set of values and than finds the interpolated value using Newton's Dividend Interpolation Formulae.



Theory

Let, $\pi_n(x) \equiv \prod_{k=1}^{n}(x - x_k),$ Then by Newton Dividend Interpolation Formulae:

$$f(x) = f_0 + \sum_{k=1}^{n} \pi_{k-1}(x)[x_0, x_1, \ldots, x_k] + R_n,$$

Code (Difftable.cpp)

```
//conm
//Diffrence table interpolation method
//10/02/04 (c) Vaibhav Jain
#include <stdio.h>
#include <conio.h>
float x[10];
float fx[10];
int count=0;
float num,result=0;
float adder;
void main()
{int i=0,j;char buffer[20];
 clrscr();
 do
 {printf("Enter Pair as x<space>f(x): ");scanf("%[0-9e. -
]s",buffer);
  fflush(stdin);
  sscanf(buffer,"%f %f",&x[i],&fx[i]);
  i++;
 }while(*buffer &&i<10);
count=i-1;

printf("\n\nEnter the value to be interpolated: ");
scanf("%f",&num);

for(i=0;i<count;i++)
{ adder=fx[0];
 for(j=0;j<i;j++)
  adder*=(num-x[j]);
 result+=adder;
 for(j=0;j<(count-1)-i;j++)
   fx[j]=(fx[j+1]-fx[j])/(float)(x[j+i+1]-x[j]);
}
printf("\n Interpolation result for %f = %f",num,result);
getch();
}
//***code concludes*****
```

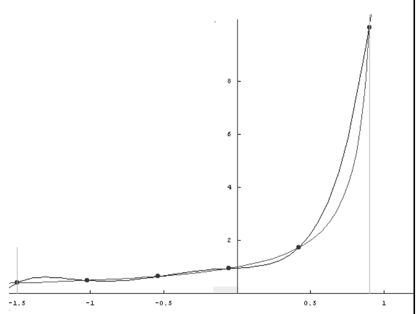
Output

```
Enter Pair as x<space>f(x): -3 -30
Enter Pair as x<space>f(x): -1 -22
Enter Pair as x<space>f(x): 0 -12
Enter Pair as x<space>f(x): 3 330
Enter Pair as x<space>f(x): 5 3458
Enter Pair as x<space>f(x):


Enter the value to be interpolated: 2.5

Interpolation result for 2.5000 = 102.6875
```

Newton's Forward Interpolation Formulae

Objective

To implement a program that accepts a set of values and than finds the interpolated value using Newton's Forward Interpolation Formulae.



Theory

Given a set of values that tabulated at equally spaced intervals than at point x Newton's forward interpolation formulae is:

$$f_a = f_0 + a\Delta + \frac{1}{2!}a(a-1)\Delta^2 + \frac{1}{3!}a(a-1)(a-2)\Delta^3 + \dots$$

where : a=(x-x_1)/h

Code

```
//forward interpolation using newton gregory method
//12/02/04 (c) Vaibhav Jain

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

float fx[10],diff,start;
unsigned int count=0;
float num,result;
void main()
{int i,j,fac;
 clrscr();
 printf("Enter Number of observations:\t");
 scanf("%u",&count);
 printf("Enter Common Diffrence:\t\t");
 scanf("%f",&diff);
 printf("Enter First Term:\t\t");
 scanf("%f",&start);
 for(printf("\n\n"),i=0;i<count;i++)
	{printf("Enter f(%f) -->",start+diff*i);
	  scanf("%f",fx+i);
	 }
 printf("\n\nEnter interpolation value:\t");
 scanf("%f",&num);

num-=start;
num/=diff;
for(result=0,fac=1,i=0;i<count;fac*=++i)
{float adder=fx[0];
 for(j=0;j<i;j++) adder*=(num-j);
 result+=adder/(float)fac;

 for(j=0;j<(count-1)-i;j++)
   fx[j]=(fx[j+1]-fx[j]);
}

printf("\nInterpolation result for %f = %f",num,result);
getch();}
//***code concludes*****
```

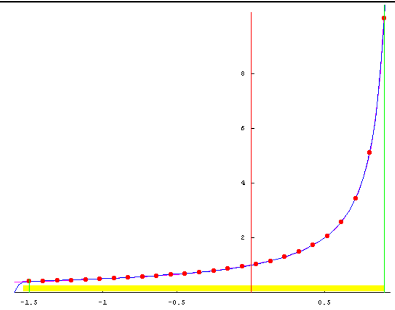
Output

```
Enter Number of observations:   7
Enter Common Diffrence:         10
Enter First Term:               1921


Enter f(1921.0000) -->35
Enter f(1931.0000) -->42
Enter f(1941.0000) -->58
Enter f(1951.0000) -->84
Enter f(1961.0000) -->120
Enter f(1971.0000) -->165
Enter f(1981.0000) -->220


Enter interpolation value:      1925

Interpolation result for 0.40000 = 36.7567
```

Newton's Backward Interpolation Formulae

Objective

To implement a program that accepts a set of values and than finds the interpolated value using Newton's Forward Interpolation Formulae.



Theory

Given values of a function f(x) at a finite number of discrete points, and letting $p = [(x - x_i) / (x_i - x_{i-1})]$, then Newton's backward interpolation formula can be expressed:

$$f_p = f_0 + p\nabla f_0 + \frac{p(p+1)}{2!}\nabla^2 f_0 + \frac{p(p+1)(p+2)}{3!}\nabla^3 f_0 + \cdots$$

Code (Nwetback.cpp)

```
//forward interpolation using newton backward formulae
//12/02/04 (c) Vaibhav Jain
#include <stdio.h>
#include <conio.h>
float fx[10],diff,start;
unsigned int count=0;
float num,result;
void main()
{int i,j,fac;
 clrscr();
 printf("Enter Number of observations:\t");
 scanf("%u",&count);
 printf("Enter Common Diffrence:\t\t");
 scanf("%f",&diff);
 printf("Enter First Term:\t\t");
 scanf("%f",&start);
 for(printf("\n\n"),i=0;i<count;i++)
            {printf("Enter f(%f) -->",start+diff*i);
             scanf("%f",fx+i);
            }
 printf("\n\nEnter interpolation value:\t");
 scanf("%f",&num);
//reverse the array
for(i=0;i<count/2;i++)
{float temp=fx[i];
 fx[i]=fx[count-i-1];
 fx[count-i-1]=temp;
}
num-=start+diff*(count-1);
num/=diff;
for(result=0,fac=1,i=0;i<count;fac*=++i)
{float adder=fx[0];
 for(j=0;j<i;j++) adder*=(num-j);
 result+=adder/(float)fac;
 for(j=0;j<(count-1)-i;j++)
   fx[j]=(fx[j+1]-fx[j]);
}
printf("\nInterpolation result for %f = %f",num,result);
getch();}
//***code concludes*****
```

Output

```
Enter Number of observations:    4
Enter Common Difference:         .1
Enter First Term:                .1


Enter f(0.100000) -->1.005
Enter f(0.200000) -->1.020
Enter f(0.300000) -->1.045
Enter f(0.400000) -->1.081


Enter interpolation value:     .35

Interpolation result for 0.350000 = 1.103438
```

Piece-Wise Linear Fit Method

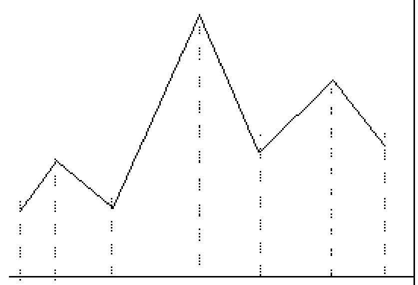
Objective

To implement a program that accepts a set of values and than finds the interpolated value using Piece-Wise Linear Fit Method.

Theory

This method takes a local view instead of global view. The given point is interpolated on the line connecting the two neighborhood points. The interpolation at a point x is given by :

$$f(x) = \frac{f(x_{i+1})\,(x - x_i) - f(x_i)\,(x - x_{i+1})}{(x_{i+1} - x_i)}$$



Code (Picefit.cpp)

```
//interpolation using piece wise fit method
//12/02/04 (c) Vaibhav Jain
#include <stdio.h>
#include <conio.h>

float x[10],fx[10];
float num,result=0;
void main()
{int i,min,max,count;
 char buffer[20];
 clrscr();

 for(count=max=min=0;*buffer&&count<10;count++)
 {printf("Enter Pair as x<space>f(x): ");
  scanf("%[0-9e. -]s",buffer);
  fflush(stdin);
  sscanf(buffer,"%f %f",&x[count],&fx[count]);
  if(x[count]>x[max]&&*buffer)max=count;
  if(x[count]<x[min]&&*buffer)min=count;
 }
count--;

printf("\n\nEnter the value to be interpolated: ");
scanf("%f",&num);

for(i=0;i<count;i++)
{ if(x[i]<=num && x[i]>x[min]) min=i;
  if(x[i]>num &&x[i]<x[max]) max=i;
}

result=fx[max]*(num-x[min])-fx[min]*(num-x[max]);
result/=(x[max]-x[min]);

printf("\nInterpolation result for %f = %f",num,result);
getch();
}
//***code concludes*****
```

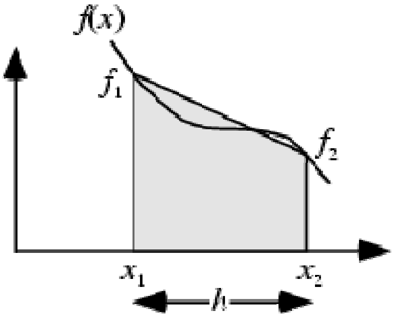
Output

```
Enter Pair as x<space>f(x): 3 4
Enter Pair as x<space>f(x): 7 9
Enter Pair as x<space>f(x): 10 12
Enter Pair as x<space>f(x): 9 72
Enter Pair as x<space>f(x): 10 12
Enter Pair as x<space>f(x):


Enter the value to be interpolated: 9.8

Interpolation result for 9.8000 = 23.999989
```

Integration With Trapezoidal Rule

Objective

To find the integral of a given tabulated function using the Trapezoidal Method of numerical Integration.



Theory

Given a Function F(x) than area under its curve from point x_1 to x_1+h is given by:

$$\int_{x_1}^{x_2} f(x)\,dx = \tfrac{1}{2}h(f_1+f_2) - \tfrac{1}{12}h^3 f''(\xi).$$

Code (Trapozoi.cpp)

```
//conm
//integration of a tabluted function using
//trapozoidal rule
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>

float points[10][2],result=0;
int count=0;

void main()
{ float x,y;char buffer[20];
  printf("**************Trapoziodal Rule**************\n");
  printf("\nNow Enter the Data Table:\n");
  do{fflush(stdin);
  printf("Enter Pair as x<space>y: ");
  scanf("%[0-9e. -]s",buffer);
  sscanf(buffer,"%f %f",&x,&y);

  if(!*buffer) continue;

  points[count][0]=x;
  points[count++][1]=y;
  }while(*buffer);

  for(int i=0;i<(count-1);i++)
        result+=1.0/2*(points[i+1][0]-points[i][0])*
        (points[i][1]+points[i+1][1]);

printf("\nIntegration Result:->%f",result);

getch();
}
//***code concludes*****
```

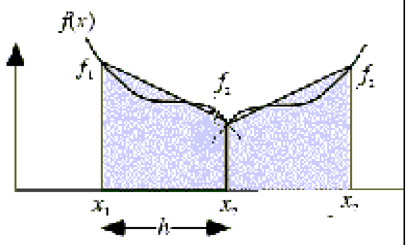
Output

```
*******Trapoziodal Rule*******

Now Enter the Data Table:
Enter Pair as x<space>y: 1 1
Enter Pair as x<space>y: 2 4
Enter Pair as x<space>y: 3 9
Enter Pair as x<space>y: 4 16
Enter Pair as x<space>y: 5 25
Enter Pair as x<space>y: 6 36
Enter Pair as x<space>y: 7 49
Enter Pair as x<space>y:

Integration Result:->115.000000
```

Integration With Simpson's 1/3 Rule

Objective

To find the integral of a given tabulated function using Simpson's 1/3 Rule.



Theory

Given a Function F(x) than area under its curve from point x_1 to x_1+h to is x_1+2h given by:

$$\int_{x_1}^{x_3} f(x)\,dx = \tfrac{1}{3}h(f_1+4f_2+f_3) - \tfrac{1}{90}h^5 f^{(4)}(\xi)$$

Code (Simpsn3.cpp)

```
//conm
//integration of a tabluted function using
//simpson's 1/3 rule
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>

float points[10][2],result=0;
int count=0;

void main()
{ float x,y;char buffer[20];
  printf("\n****Simpson's 1/3 Rule****\n");
  printf("\nNow Enter the Data Table:\n");
  do{fflush(stdin);
  printf("Enter Pair as x<space>y: ");
  scanf("%[0-9e. -]s",buffer);
  sscanf(buffer,"%f %f",&x,&y);
  if(!*buffer) continue;
  points[count][0]=x;
  points[count++][1]=y;
  }while(*buffer);

  int i=0;
  if(count%2==0)//total points are even solve first point with
trapoziodal
  result= 1.0/2*(points[i+1][0]-points[i][0])/
             (points[i+1][1]+points[i][1]);
  for(;i<(count-1);i+=2)
    {  result+=1.0/3*(points[i+1][0]-points[i][0])*
          (points[i][1]+4*points[i+1][1]+points[i+2][1]);
    }

 //applying simpsons 1/3 rule
printf("\nIntegration Result:->%f",result);
getch();
}
//***code concludes*****
```

Output

```
****Simpson's 1/3 Rule****

Now Enter the Data Table:
Enter Pair as x<space>y: 1 1
Enter Pair as x<space>y: 2 1.41
Enter Pair as x<space>y: 3 1.71
Enter Pair as x<space>y: 4 2
Enter Pair as x<space>y: 5 2.23
Enter Pair as x<space>y: 6 2.44
Enter Pair as x<space>y:

Integration Result:->10.967469
```

Integration With Simpson's 3/8 Rule

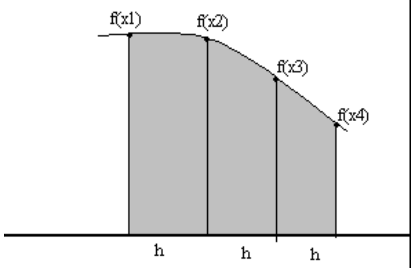
Objective

To find the integral of a given tabulated function using Simpson's 3/8 Rule.

Theory

Given a Function F(x) than area under its curve from point x_1 to x_1+h to is x_1+2h to x_1+3h given by:

$$\int_{x_1}^{x_4} f(x)\,dx = \tfrac{3}{8}h(f_1 + 3f_2 + 3f_3 + f_4) - \tfrac{3}{80}h^5 f^{(4)}(\xi)$$



Code (Simpsn8.cpp)

```
//conm
//integration of a tabluted function using
//simpson's 3/8 rule
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
float points[10][2],result=0;
int count=0;
void main()
{ float x,y;char buffer[20];
  printf("\n****Simpson's 3/8 Rule****\n");
  printf("\nNow Enter the Data Table:\n");
  do{fflush(stdin);
  printf("Enter Pair as x<space>y: ");
  scanf("%[0-9e. -]s",buffer);
  sscanf(buffer,"%f %f",&x,&y);
  if(!*buffer) continue;
  points[count][0]=x;
  points[count++][1]=y;
  }while(*buffer || count<2);
  int i=0;
  //simpsons 3/8 rule can only be applied to 4 closed points
  //extra points are evaluated with trapoziod rule
  int extras=count%4;
  for(i=0;i<extras;i++)
  result+=1.0/2*(points[i+1][0]-points[i][0])*
             (points[i+1][1]+points[i][1]);

  for(;i<(count-1);i+=3)
    {  result+=1.0*3/8*(points[i+1][0]-points[i][0])*
           (points[i][1]+3*points[i+1][1]+3*points[i+2][1]
           +points[i+3][1]);
    }
 //applying simpsons 3/8 rule
printf("\nIntegration Result:->%f",result);
getch();
}
//***code concludes*****
```

Output

```
****Simpson's 3/8 Rule****

Now Enter the Data Table:
Enter Pair as x<space>y: 1 1
Enter Pair as x<space>y: 2 8
Enter Pair as x<space>y: 3 1.4
Enter Pair as x<space>y: 4 3.21
Enter Pair as x<space>y: 5 31
Enter Pair as x<space>y:

Integration Result:->24.311251
```

Euler's Method For Differential Equations

Objective

To find the numerical solution to the differential equation f'(x)=x2+y2 with initial condition as y(0)=1 for 1<x<0

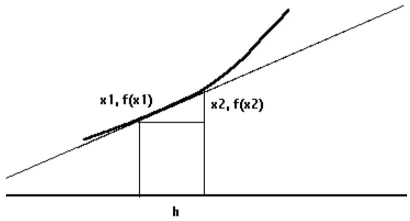
Theory

Consider the following differential equation: y'=f(x,y) , with $y(x_0) = K$,dividing interval [a,b] into n sub intervals of size h than a numerical approximation to the differential equation is:

$y_{i+1} = y_i + h.k$ for i=0,1,...,n-1

where

$y_0 = K$ (starting value) ; $k = h .f(x_i,y_i)$



Code (Difreule.cpp)

```
//conm
//solving ordinary diffrential equations
//using Eulers Method of form dy/dx=f(x,y)

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

float initx=0,inity=1;//boundary conditions
float tx,step;

//gives the value of f(x,y)
float fx(float x,float y) {return x*x+y*y;}

void main()
{printf("\nEnter Target X: ");//destination x
 scanf("%f",&tx);
 printf("Step Size?:");scanf("%f",&step);
 float x=initx,y=inity;
 //calculate no. of steps required
 long steps=(tx-initx)/step+1;
 while(steps--)
 {y=y+step*fx(x,y);//apply eulers formulae
  x+=step;
  printf("\nF(%f)=%f",x,y);
  getch();
 }
printf("\nY(%f)=%f",x,y);
getch();
}
//***code concludes*****
```

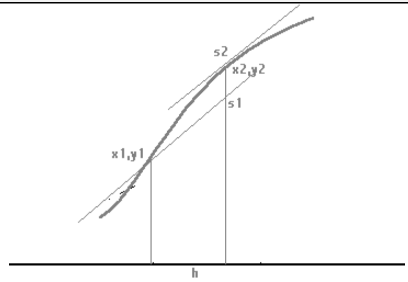
Output

```
Enter Target X: 1
Step Size?:.1

F(0.100000)=1.100000
F(0.200000)=1.222000
F(0.300000)=1.375328
F(0.400000)=1.573481
F(0.500000)=1.837065
F(0.600000)=2.199546
F(0.700000)=2.719347
F(0.800000)=3.507832
F(0.900000)=4.802320
F(1.000000)=7.189548
Y(1.000000)=7.189548
```

Second Order Runge-Kutta Method

Objective

To find the numerical solution to the differential equation f'(x)=x2+y2 with initial condition as y(0)=1 for 1<x<0



Theory

Consider the following differential equation: y'=f(x,y) , with $y(x_0) = K$,dividing interval [a,b] into n sub intervals of size h than a numerical approximation to the differential equation is:

$y_{i+1} = y_i + (1/2)\ [k1 + k2\ ]$ for i=0,1,...,n-1

where

$y_0 = K$ (starting value) ; $k1 = h\ .f(x_i,y_i)$; $k2 = h.f(x_i+h,y_i+k)$,

Code (Rukutta.cpp)

```
//conm
//solving ordinary diffrential equations
//using Runge-Kutta 2nd Order Formule
#include <conio.h>
#include <stdio.h>

float initx=0,inity=1;//boundary conditions
float tx,step;

//gives the value of f(x,y)

float fx(float x,float y)
        {return x*x+y*y;}

void main()
{printf("\nEnter Target X: ");//destination x
 scanf("%f",&tx);
 printf("Step Size?:");scanf("%f",&step);
 float x=initx,y=inity;
 //calculate no. of steps required
 long steps=(tx-initx)/step+1;
 while(steps--)
 {float s1=fx(x,y);
  float s2=fx(x+step,y+s1*step);
  y=y+step*(s1+s2)/2;
  x+=step;
 printf("\nF(%f)=%f",x,y);
 getch();
 }
printf("\nY(%f)=%f",x,y);
getch();
}
//***code concludes*****
```

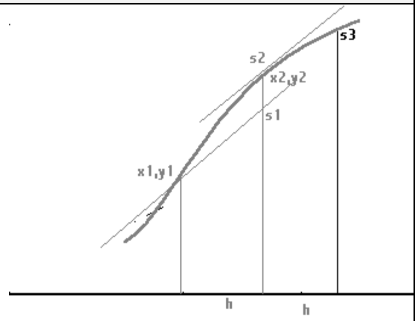
Output

```
Enter Target X: 1
Step Size?:.1

F(0.100000)=1.111000
F(0.200000)=1.251531
F(0.300000)=1.436058
F(0.400000)=1.688008
F(0.500000)=2.048771
F(0.600000)=2.600026
F(0.700000)=3.529012
F(0.800000)=5.371471
F(0.900000)=10.348342
F(1.000000)=38.134342
Y(1.000000)=38.134342
```

Fourth Order Runge-Kutta Method

Objective

To find the numerical solution to the differential equation f'(x)=x2+y2 with initial condition as y(0)=1 for 1<x<0



Theory

Consider the following differential equation: y'=f(x,y) , with $y(x_0)$ = K ,dividing interval [a,b] into n sub intervals of size h than a numerical approximation to the differential equation is:

$y_{i+1} = y_i + (1/6)\,[k1 + 2k2 + 2k3 + k4]$ for i=0,1,...,n-1

where

y_0 = K (starting value) ; $k1 = hf(x_i,y_i)$; $k2 = hf(x_i+h/2,y_i+k1/2)$,
$k3 = hf(x_i+h/2,y_i+k2/2)$; $k4 = hf(x_i+h,y_i+k3)$

Code (Rukutta4.cpp)

```
//solving ordinary diffrential equations
//using Runge-Kutta 4th Order Formule

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

float initx=0,inity=1;//boundary conditions
float tx,step;

//gives the value of f(x,y)
float fx(float x,float y)
        {return x*x+y*y;}

void main()
{printf("\nEnter Target X: ");//destination x
 scanf("%f",&tx);
 printf("Step Size?:");scanf("%f",&step);
 float x=initx,y=inity;
 //calculate no. of steps required
 long steps=(tx-initx)/step+1;
 while(steps--)
 {float s1=fx(x,y);
  float s2=fx(x+step/2,y+s1*step/2);
  float s3=fx(x+step/2,y+s2*step/2);
  float s4=fx(x+step,y+s3*step);
  y=y+step*(s1+2*s2+2*s3+s4)/6;
  x+=step;
  printf("\nF(%f)=%f",x,y);
  getch();
 }
printf("\nY(%f)=%f",x,y);
getch();
}
//***code concludes*****
```

Output

```
Enter Target X: 1
Step Size?:.1

F(0.100000)=1.111463
F(0.200000)=1.253015
F(0.300000)=1.439666
F(0.400000)=1.696098
F(0.500000)=2.066961
F(0.600000)=2.643860
F(0.700000)=3.652201
F(0.800000)=5.842014
F(0.900000)=14.021826
F(1.000000)=735.101379
Y(1.000000)=735.101379
```