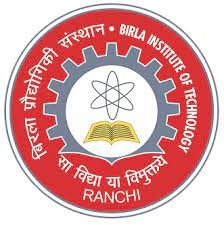
**Birla institute of technology Mesra,Ranchi**



**Numerical method**

Prepared by :Robin Roy

ROLL NO : B.Tech/15114/19

Faculty in charge

Dr. Prakash Chandra srivastava

Patna off campus

2021

**Write one application with example in each module (I-V) of Numerical method in EEE.**

**MODULE-1**

Bisection Method

The method is applicable in the Thermistor.

It measures temperature by measuring the electrical resistance with a

change in temperature. Bisection method can be applied to determine the resistance value

with the equation:

1/T=1.129241 x 10^3+2.341077 x 10^-3 In(R) + 8.775468 x 10^-8(In(R))^3

**MODULE-2**

Gauss Seidal Method

Gauss-Seidal Method is very simple and uses in digital computer for computing.

Gauss-Seidal Method is used to calculate the unknown value of voltage.

**MODULE-3**

Interpolation

It is used in rigid transformation of images by rigid transformation we

mean a linear transformation of the pixel coordinator.

Interpolation technique are also used for rooming digital images

**MODULE-4**

Simpson’s Rule

It is used for calculating static and dynamic reaction forces on areas and

volumes. Simpson's Rule is also used for calculating the average power over an Integral number of cycles of voltages and current.

Ex: Average Power-1/T/ Voltage(r)Current()dt

**MODULE-5**

Euler’s Method

This method is used to account for the behaviour of electric circuits using alternating current(AC).

It is used in the mon practical sense for working with radiative decay.

**1**.**Write syntax for breaking infinite loops by the use of break a statement**

**in the body of the while statement.**

while (test\_condition)

{

statement1;

if (condition )

break;

statement2;

}

**2.Write syntax for decision making instructions in C:**

* **The if statement**

if ( expression)

{

statements;

}

* **The if –else statement**

if (expression)

{

statement1;

}

else

{

statement2;

}

* **The switch statement**

switch(expression)

{

case exp1:

code block1;

break;

case exp2:

code block2;

break;

.......

default:

default code block;

}

**3.Write syntax for loop control structure in C:**

* **Using a for statement**

for ( statement1; statement2; statement3)

{

//body of for loop

}

* **Using a while statement**

while (condition)

{

//block of code to be executed

}

* **Using a do-while statement**

do

{

//block of code to be executed

} while (condition);

**4.Write the name of which method the following segment of algorithm is associated and making flow chart of its method**

**If (f(x2 ) not equal to 0) then**

**if f(x0)\*f(x2) < 0**

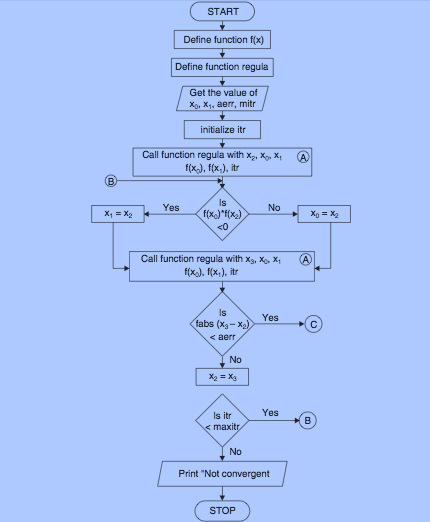
**x1 x2**

**else**

**x0  x2**

**This algorithm is associated with Regula falsi method.**

The flow chart of this method is given below-



**5. What shall be output by the following program segment :**

**sum = 0 ;**

**for(i=1; i<=5; i++)**

**{ sum = sum + i ; }**

**cout<<sum<<i ;**

**OUTPUT : 156**

**6.Find the syntax/logical errors in the following program segment of Gauss Elimination method :**

**for( p = 0 ; p<= 1;p++)**

**for(i=p+1 ;i<=2;i++)**

**for(j=0 ; j<=2; j++)**

**a[i][j] = a[i][j] - (a[i][p]\*a[j][p])/a[p][p] ;**

**Due to space between the last two statement the matrix operationis outside of the loop body.**

**for( p = 0 ; p<= 1;p++)**

**for(i=p+1 ;i<=2;i++)**

**for(j=0 ; j<=2; j++)**

**a[i][j] = a[i][j] - (a[i][p]\*a[j][p])/a[p][p] ;**

**7.Find the error in the following program segment :--**

**# define f(x) = (x\*x\*x -28)**

**void main()**

**{ float a , b ;**

**a = 2.5 ; b = 1.25 ;**

**c = f(a)/f(b) ;**

**}**

**Due to equal to sign in (# define f(x) = (x\*x\*x -28)) f(x) is not defined properly and data type of c is not define.**

# define f(x) x\*x\*x -28

void main()

{ float a , b ,c ;

a = 2.5 ; b = 1.25 ;

c = f(a)/f(b) ;

.....

.....................

}

**8.Write the name of which method the following segment of algorithm is associated and making flow chart of its method**

**If (f(x2 ) not equal to 0) then**

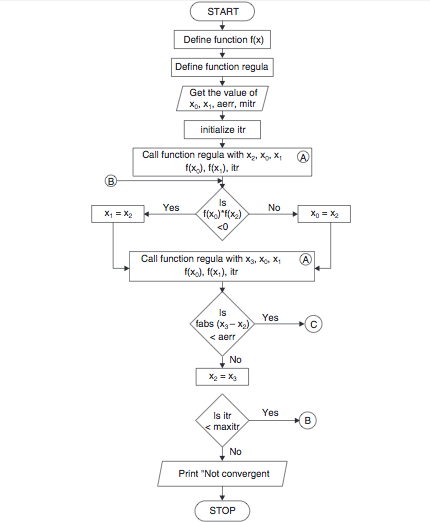
**if f(x0)\*f(x2) < 0**

**x1 x2**

**else**

**x0  x2**

**Regula Falsi method**



**Algorithm**

Initialize: x1, x2, E, n // E = convergence indicator

calculate f(x1),f(x2)

if(f(x1) \* f(x2) = E); //repeat the loop until the convergence

print 'x0' //value of the root

print 'n' //number of iteration

}

else

print "can not found a root in the given interval"

**9.Write the output using C or, C++ as evaluating using Simpson’s 1/3 rule (taking h = 0.1).**

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);

}

**Output:I =0.231050**

**10.Make the necessary correction/addition/deletion in the following back substitution algorithm—**

**Xn --an,n+1/an,n**

**For i= n-1 to 1 in steps of -1 do**

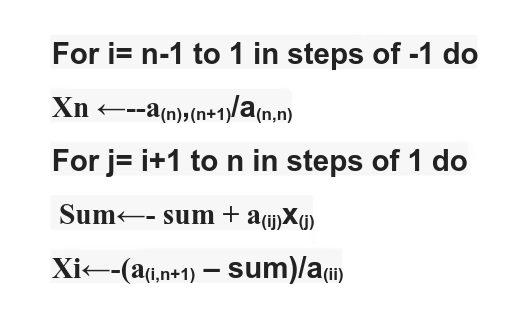
**For j= i+1 to n in steps of 1 do**

**Sum - sum + aijxj endfor**

**Xi -(ai,n+1 – sum)/aii**

**End for**

Ans-



**End for**

**End for**

**11. How many iterations , execution of the following loop shall terminate**

**e = 0.1 ;**

**do**

**{ x0  1 ; x1  3 ; x0  x1 ;**

**}**

**while(fabs(x0-x1)>= e )**

**one iterations (1)**

**12.Find the error in the following :--**

void main()

{ float a[3][3] ; int i,j ;

for(i=0; i<= 3 ; i++)

for(j=0; j<= 3 ; j++)

cin<< a[i][j] ; ........ ; }

void main()

{ float a[3][3] ; int i,j ;

for(i=0; i<= 3 ; i++)

for(j=0; j<= 3 ; j++)

cin>> a[i][j] ;

........ ;

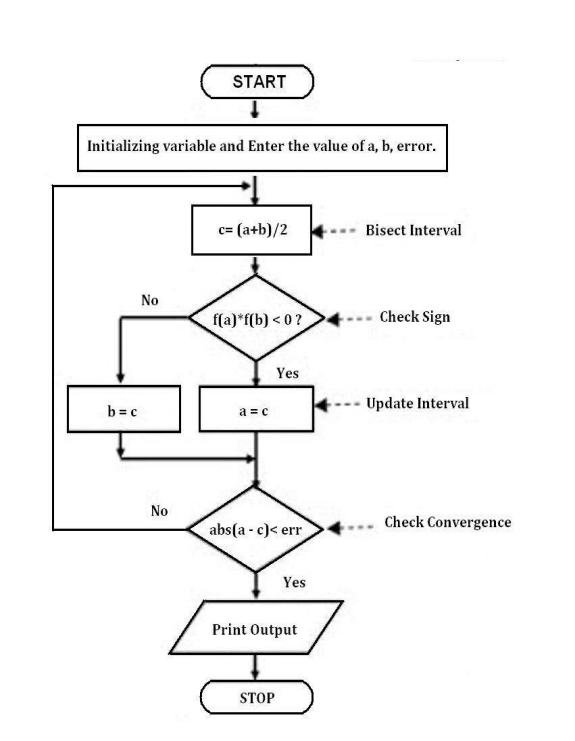
}

**Write any 10 Programs in C:**

BISECTION METHOD

ALGORITHM: -

* Start
* Define function F(x) and error e=0.001.
* Enter the initial guesses A and B.
* Calculate f1 & f2 where, f1=F(A) & f2=F(B).
* Now check if f1\*f2<0. If yes then go to step 5, else go to step 10.
* Calculate new approximated root 𝑥 0 as (𝐴+𝐵)/2 and its corresponding value of function as f3.
* Then check
* If f3\*f1 < 0 then B 0 =x
* If f3\*f1 > 0 then A 0 = x
* Repeat step 6 & 7 till (B-A) > e.
* Display 0 x as root.
* Stop



PROGRAM

Equation f (x) = x^3 + x – 1 = 0, using Bisection method.

#include <stdio.h>

#include<math.h>

#define e 0.001

#define F(x) x\*x\*x + x -1

int main()

{

int i;

float A,B,x0;

double f1,f2,f3;

printf("\nENTER THE VALUE OF A:");

scanf("%f",&A);printf("\nENTER THE VALUE OF B:");

scanf("%f",&B);

f1 = F(A);

f2 = F(B);

if (f1\*f2>0)

printf("REAL ROOT DOES NOT EXIST BETWEEN %f and %f",A,B);

else

{

printf("ITERATION NO. \tVALUE OF A \tVALUE OF B \tVALUE OF X

\tVALUE OF F(X)");

for(i=1;(B-A)>=e;i++)

{

x0 = (A+B)/2;

f3 = F(x0);

printf("\n\t%d \t %f \t %f \t %f \t %f",i,A,B,x0,F(x0));

if(f3\*f1<0)

{

B=x0;

}

else

{

A=x0;

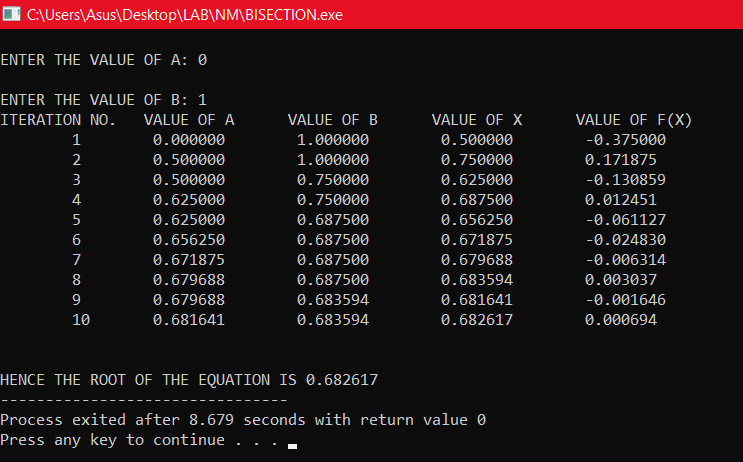
}

}

printf("\n\n\nHENCE THE ROOT OF THE EQUATION IS %f",x0);}

return 0;

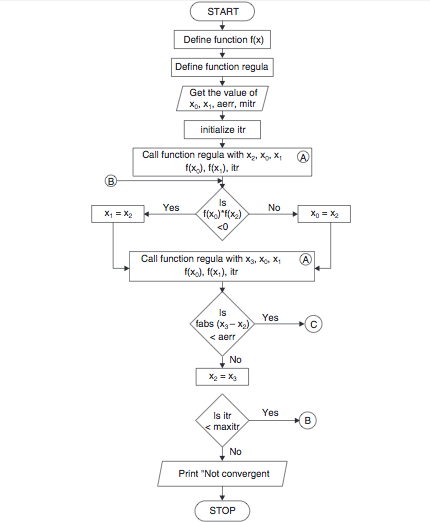
}



**REGULA FALSI METHOD**

ALGORITHM: -

* Start
* Define function F(x) and error e=0.001.
* Enter the initial guesses A and B. Also enter the maximum no. of iterations allowed.
* Calculate f1 & f2 where, f1=F(A) & f2=F(B).
* Now check if f1\*f2<0. If yes then go to step 5, else go to step 10.
* Calculate new approximated root 𝑥 0 as =(𝐴∗𝑓2−𝐵∗𝑓1)/(𝑓2−𝑓1) and its corresponding value
* of function as f3.
* Then check
* If f3\*f1 < 0 then B 0 = & x f2 = f3.
* If f3\*f1 > 0 then A 0 = & x f1 = f3.
* Repeat step 6 & 7 till the loop not reaches to maximum no. of iterations.
* Display 0 x as root.
* Stop

****

**PROGRAM -**

Equation f (x) = x^3 – 2x – 5 = 0 by method of False position.

#include<stdio.h>

#include<math.h>

#define F(x) x\*x\*x - 2\*x -5

#define e 0.001

int main()

{

float A, B, x0, f1, f2, f3;

int i,maxitr;

printf("\nENTER THE VALUE OF A:");

scanf("%f",&A);printf("\nENTER THE VALUE OF B:");

scanf("%f",&B);

printf("\nENTER THE Maximum no. of iterations allowed:");

scanf("%d",&maxitr);

f1 = F(A);

f2 = F(B);

if( f1\*f2 >0)

{

printf("REAL ROOT DOES NOT EXIST BETWEEN %f and %f",A,B);

}

else

{

printf("\nITERATION NO. \t \tVALUE OF A \tVALUE OF B \tVALUE OF X

\tVALUE OF F(X)");

for(i=1;i<=maxitr;i++)

{

x0 = ((A\*f2)- (B\*f1))/(f2-f1);

f3 = F(x0);

printf("\n\t%d\t\t%f\t%f\t%f\t%f\n",i, A, B, x0, f3);

if(f1\*f3 < 0)

{

B = x0;

f2=f3;

}

else

{

A = x0;f1=f3;

}

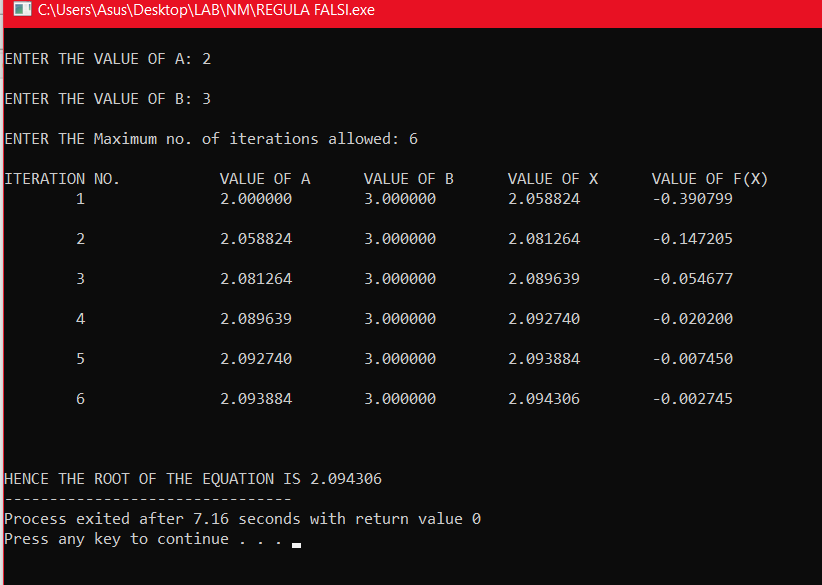
}

printf("\n\n\nHENCE THE ROOT OF THE EQUATION IS %f",x0);

}

return 0;

}



**GAUSS ELIMINATION METHOD**

ALGORITHM:

1. Start the program

2. Read the order of the matrix n

3. Read the elements of the augmented matrix

4. For j=1 to j<=n and for i=1 to i<=n; if (i>j) then c=a[i][j]/a[j][j]

5. For k=1 to k<=n+1 ,a[i][k]=a[i][k]-c\*a[j][k] ,k=k+1

6. Compute x[n]=a[n][n+1]/a[n][n]

7. For i=n-1 to i>=1 sum=0

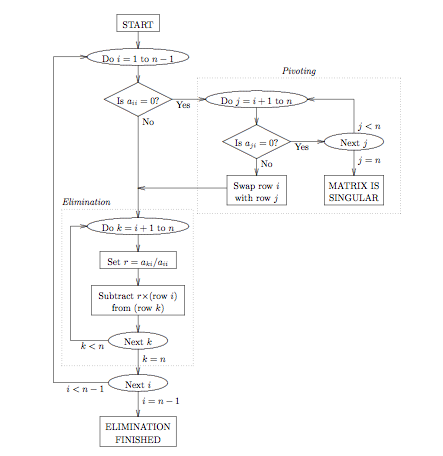
8. For j=i+1 to j<=n, sum=sum+a[i][j]\*x[j]

9. Compute x[i]=(a[i][n+1]-sum)/a[i][i]

10.For i=1 to i<=n

11.11 display the result x[i];i=i+1

12.stop



CODE:

#include<stdio.h>

#include<conio.h>

void main()

{

int i,j,k,n;

float a[20][20],c,x[10],sum=0.0;

clrscr();

printf("\n enter the order of matrix:");

scanf("%d",&n);

printf("\n Enter the elememts of augmented matrix row-wise:\n\n");

for(i=0;i<=n;i++)

{

for(j=1;j<=(n+1);j++)

{

printf("a[%d][%d]:",i,j);

scanf("%f",&a[i][j]);

}

}

for(j=1;j<=n;j++)

{

for(i=1;i<=n;i++)

{if(i>j)

{

c=a[i][j]/a[j][j];

for(k=1;k<=n+1;k++)

{

a[i][k]=a[i][k]-c\*a[j][k];

}

}

}

}

x[n]=a[n][n+1]/a[n][n];for(i=n-1;i>=1;i--)

{

sum=0;

for(j=i+1;j<=n;j++)

{

sum=sum+a[i][j]\*x[j];

}

x[i]=(a[i][n+1]-sum)/a[i][i];

}

printf("\n the solution is:\n");

for(i=1;i<=n;i++)

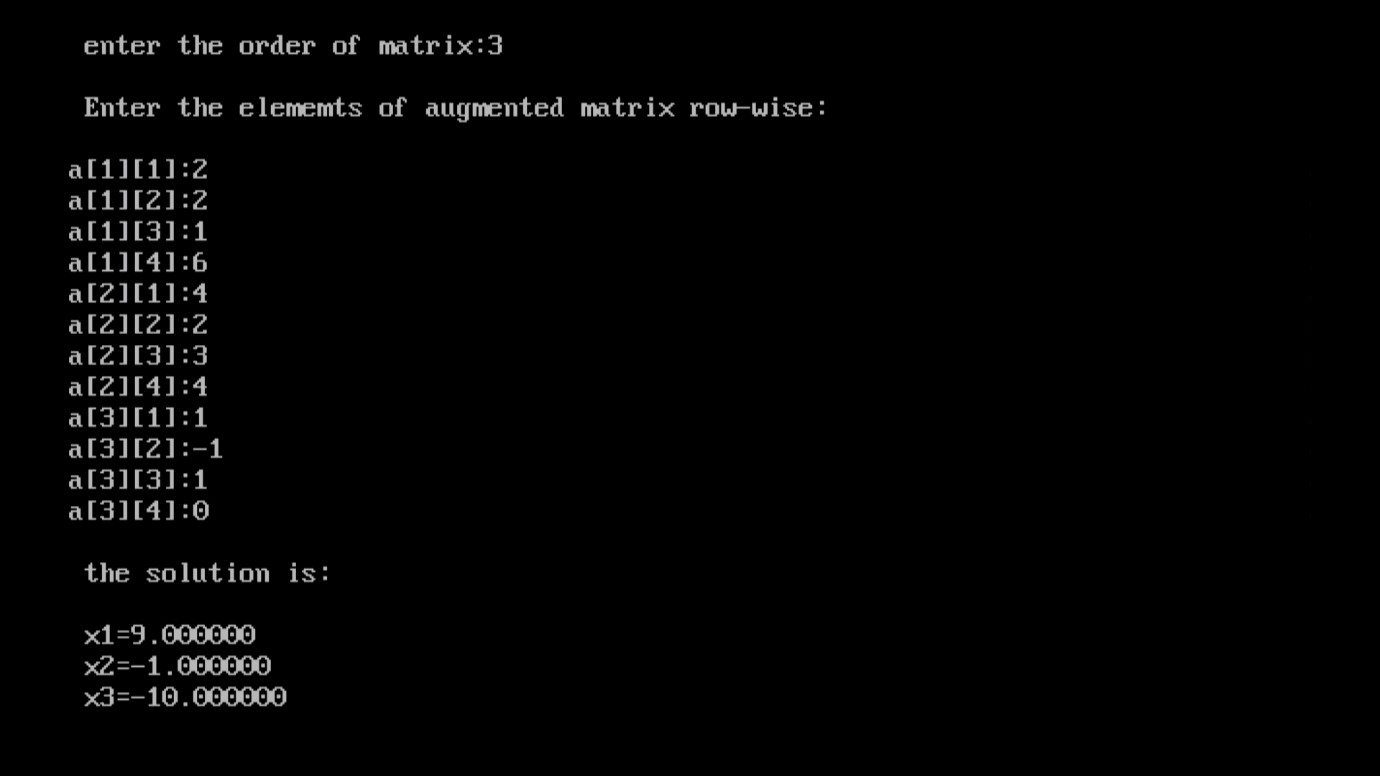
{

printf("\n x%d=%f\t",i,x[i]);

}

getch();

}



**GAUSS JORDAN METHOD**

ALGORITHM:

1. start

2. read the order pf matrix n

3. read the elements of augmented matrix

4. for j=1;j<=n and for i=1 to i<=n; repeat steps 5 to 10.

5. If(i!=j) then compute c=a[i][j]/a[i][i]

6. For k=1 to k<=n+1

7. Compute a[i][k]=a[i][k]-c\*a[j][k]

8. K=k+1

9. I=i+1

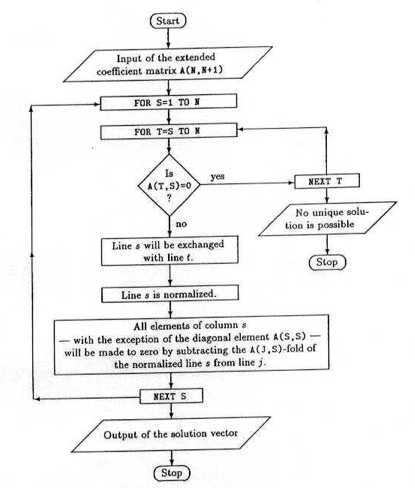
10.J=j+1

11.For i=1 to i<=n repeat step 12 and 13

12.Compute x[i]=a[i][n+1]/a[i][i]

13.Print the solution is x[i], i=i+2

14.Stop



CODE:

#include<stdio.h>

#include<conio.h>

void main()

{

int i,j,k,n;

float a[20][20],c,x[10];

clrscr();

printf("\n enter the size of matrix:");

scanf("%d",&n);

printf("\n enter the elements of augmented matrix:\n");

for(i=1;i<=n;i++)

{

for(j=1;j<=(n+1);j++)

{

printf("a[%d][%d]:",i,j);

scanf("%f",&a[i][j]);

}

}

for(j=1;j<=n;j++)

{

for(i=1;i<=n;i++)

{

if(i!=j)

{

c=a[i][j]/a[j][j];

for(k=1;k<=n+1;k++)

{

a[i][k]=a[i][k]-c\*a[j][k];}

}

}

printf("\n the solution is :\n");

for(i=1;i<=n;i++)

{

x[i]=a[i][n+1]/a[i][i];

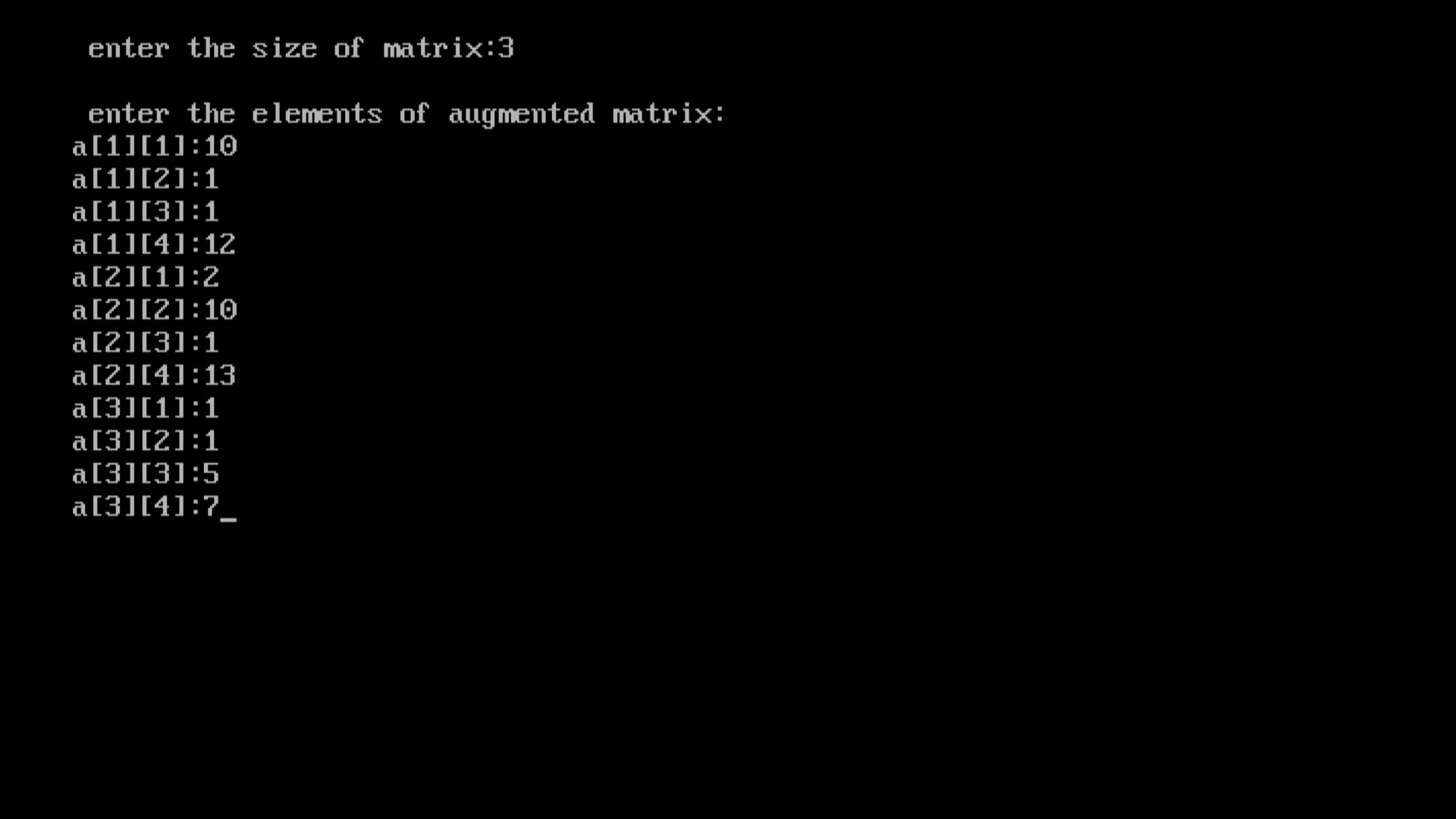
printf("\n x%d=%f\n",i,x[i]);

}

}

getch();

}



**Newton Forward Method**

**Algorithm:**

1. Start

2. Read number of data (n)

3. Read data points for x and y:

For i = 0 to n-1

Read Xi and Yi,0

Next i

4. Read calculation point where derivative is required (xp)

5. Set variable flag to 0

6. Check whether given point is valid data point or not. If it is valid point then get its position at variable index

For i = 0 to n-1

If |xp - Xi| < 0.0001

index = i

flag = 1

break from loop

End If

Next i

7. If given calculation point (xp) is not in x-data then terminate the process.

If flag = 0

Print "Invalid Calculation Point"

Exit

End If

8. Generate forward difference table

For i = 1 to n-1

For j = 0 to n-1-i

Yj,i = Yj+1,i-1 - Yj,i-1

Next j

Next i

9. Calculate finite difference: h = X1 - X0

10. Set sum = 0 and sign = 1

11. Calculate sum of different terms in formula to find derivatives using Newton's forward difference formula:

For i = 1 to n-1-index

term = (Yindex, i)i / i

sum = sum + sign \* term

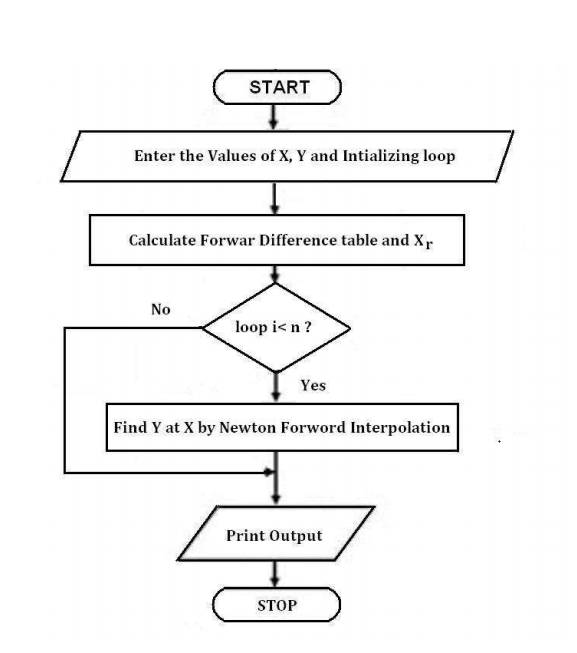
sign = -sign

Next i

12. Divide sum by finite difference (h) to get result first\_derivative = sum/h

13. Display value of first\_derivative

14. Stop



Code:

#include<stdio.h>

#include<conio.h>

#define MAXN 100

#define ORDER 4

main()

{

float ax[MAXN+1], ay [MAXN+1], diff[MAXN+1][ORDER+1], nr=1.0, dr=1.0,x,p,h,yp;

int n,i,j,k;

printf("\nEnter the value of n:\n");

scanf("%d",&n);

printf("\nEnter the values in form x,y:\n");

for (i=0;i<=n;i++)

scanf("%f %f",&ax[i],&ay[i]);

printf("\nEnter the value of x for which the value of y is wanted: \n");

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<=ORDER;j++)

for(i=0;i<=n-j;i++)

diff[i][j] = diff[i+1][j-1] - diff[i][j-1];

i=0;

while (!(ax[i]>x))

i++;

i--;

p = (x-ax[i])/h;

yp = ay[i];

for (k=1;k<=ORDER;k++)

{

nr \*=p-k+1;

dr \*=k;

yp +=(nr/dr)\*diff[i][k];

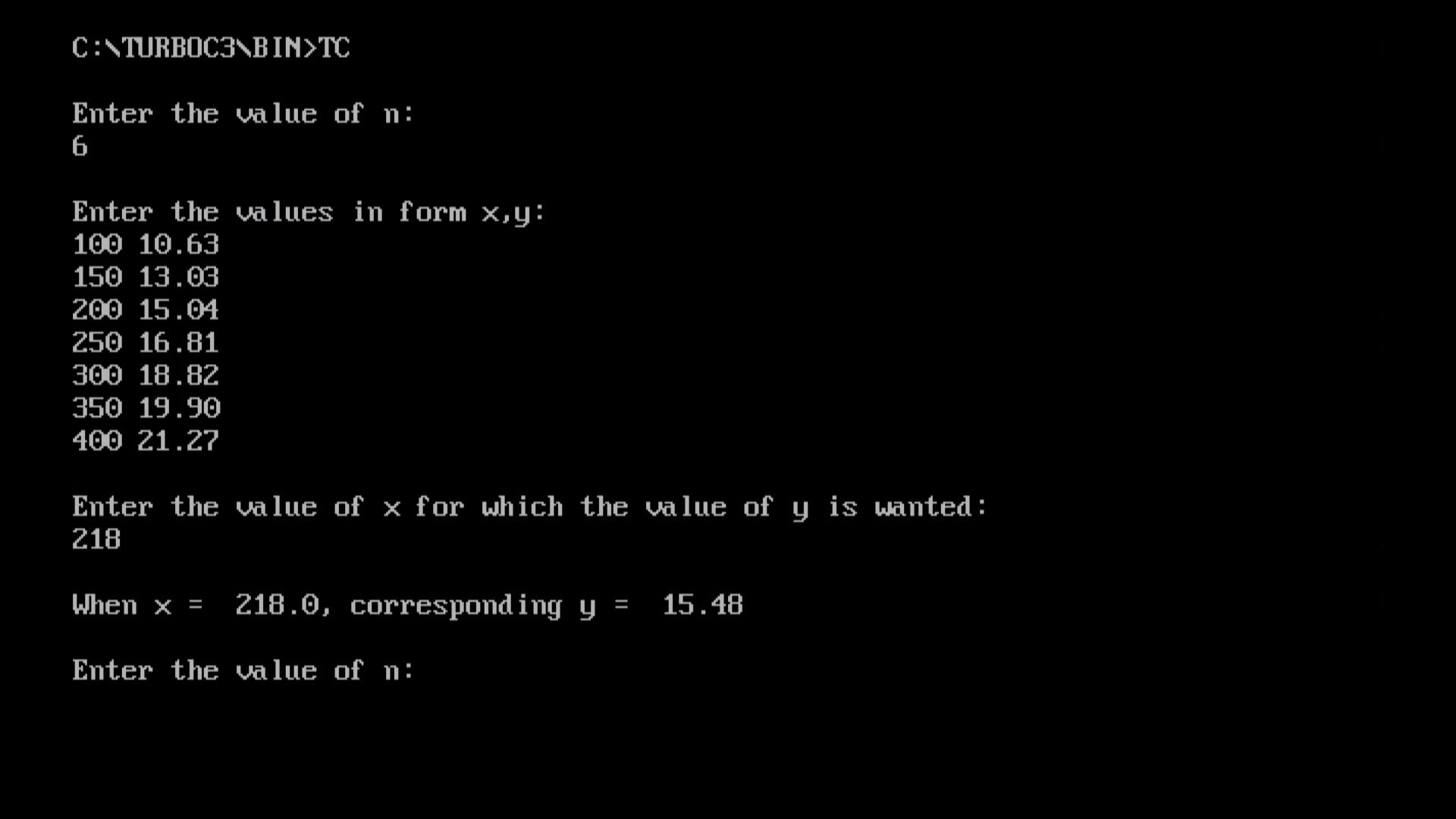
}

printf("\nWhen x = %6.1f, corresponding y = %6.2f\n",x,yp);

getch();

return 0;

}



**Newton Backward Method**

Algorithm:

1. Start

2. Read number of data (n)

3. Read data points for x and y:

For i = 0 to n-1

Read Xi and Yi,0

Next i

4. Read calculation point where derivative is required (xp)

5. Set variable flag to 0

6. Check whether given point is valid data point or not.

If it is valid point then get its position at variable index

For i = 0 to n-1

If |xp - Xi| < 0.0001

index = i

flag = 1

break from loop

End If

Next i

7. If given calculation point (xp) is not in

x-data then terminate the process.

If flag = 0

Print "Invalid Calculation Point"

Exit

End If

8. Generate backward difference table

For i = 1 to n-1

For j = n-1 to i (Step -1)

Yj,i = Yj,i-1 - Yj-1,i-1

Next j

Next i

9. Calculate finite difference: h = X1 - X0

10. Set sum = 0

11. Calculate sum of different terms in formula

to find derivatives using Newton's backward

difference formula:

For i = 1 to index

term = (Yindex, i)i / i

sum = sum + term

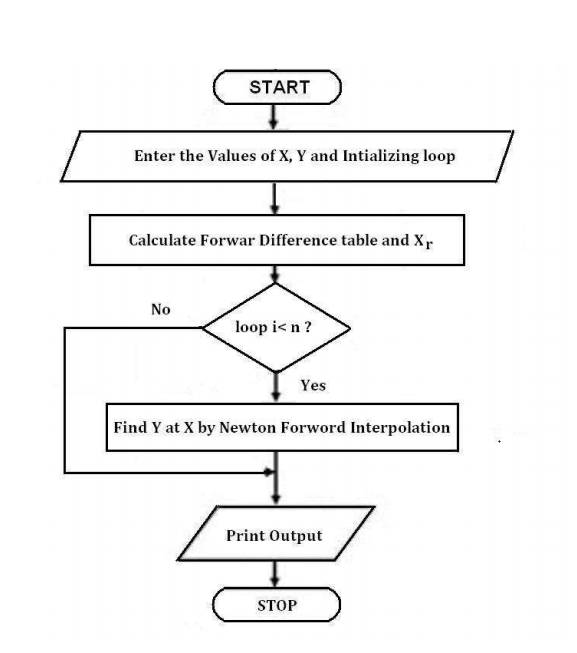
Next i

12. Divide sum by finite difference (h) to get result

first\_derivative = sum/h

13. Display value of first\_derivative

14. Stop



**Code:**

#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)

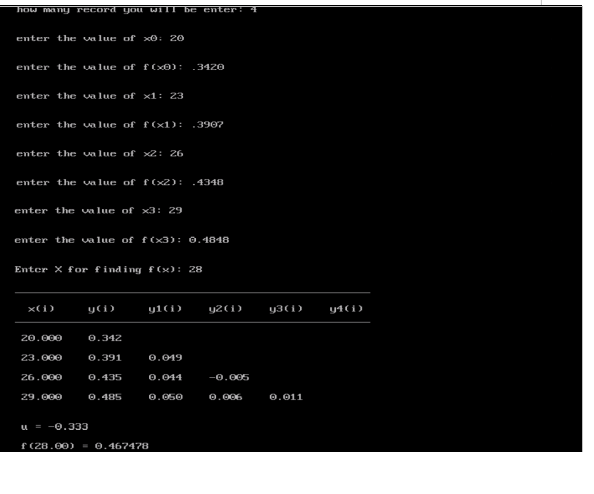
return (1);

else

fac = a \* fact(a-1);

return(fac);

}



**Simpson’s 1/3rd rule**

**Algorithm:**

1. Start.

2. Define an equation for f(x).

3. Define a method by the name of simpsonsRule().

4. Take the values of lower and upper limits of integration as well as the number of sub-intervals as inputs from the user.

5. Initialize a variable ifx with 0.

6. Find the value of h. [h = (b-a)/n]

7. Add the values of f(a) and f(b) to ifx.

8. Set the value of i = a+h.

9. Multiply the value of f(i) by 4 and add the result to ifx.

10. Add (2\*h) to the existing value of i.

11. Repeat steps 8, 9 and 10 till the value of i is less than b.

12. Set the value of j = a+(2\*h).

13. Multiply the value of f(j) by 2 and add the result to ifx.

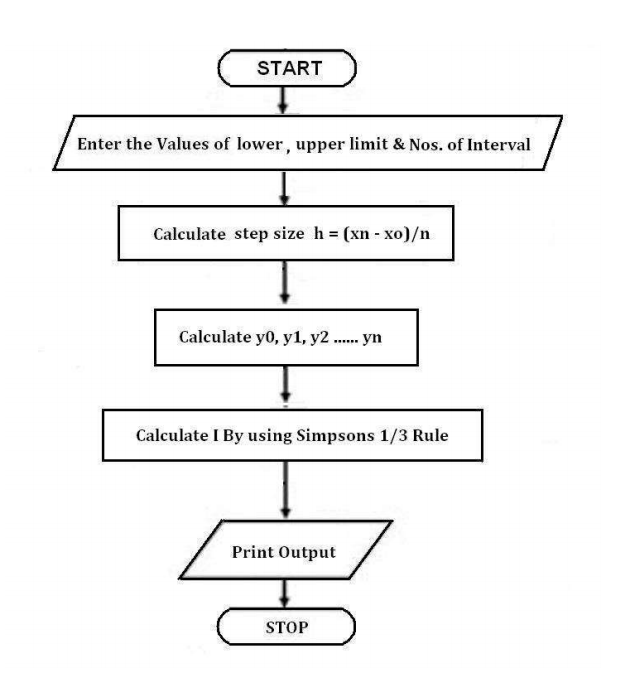
14. Add (2\*h) to the existing value of j.

15. Repeat steps 12, 13 and 14 till the value of j is less than b.

16. Multiply the final value of ifx with h and divide the result by 3 and store it back in ifx.

17. Print the value of integration ifx.

18. Stop.



**Program:**

#include<stdio.h>

#include<conio.h>

float y(float x)

{

return x\*x/(1+x\*x\*x);

}

void main()

{

float x0,xn,h,s,sum;

int i,n;

puts("\n Enter number of subdivision i.e n");

scanf("%d",&n);

puts("\n Enter lower limit of integrals i.e x0");

scanf("%f",&x0);

puts("\n Enter upper limit of integral i.e xn");

scanf("%f",&xn);

h = (xn-x0)/n;

s = y(x0)+ y(xn)+ 4\*y(x0+h);

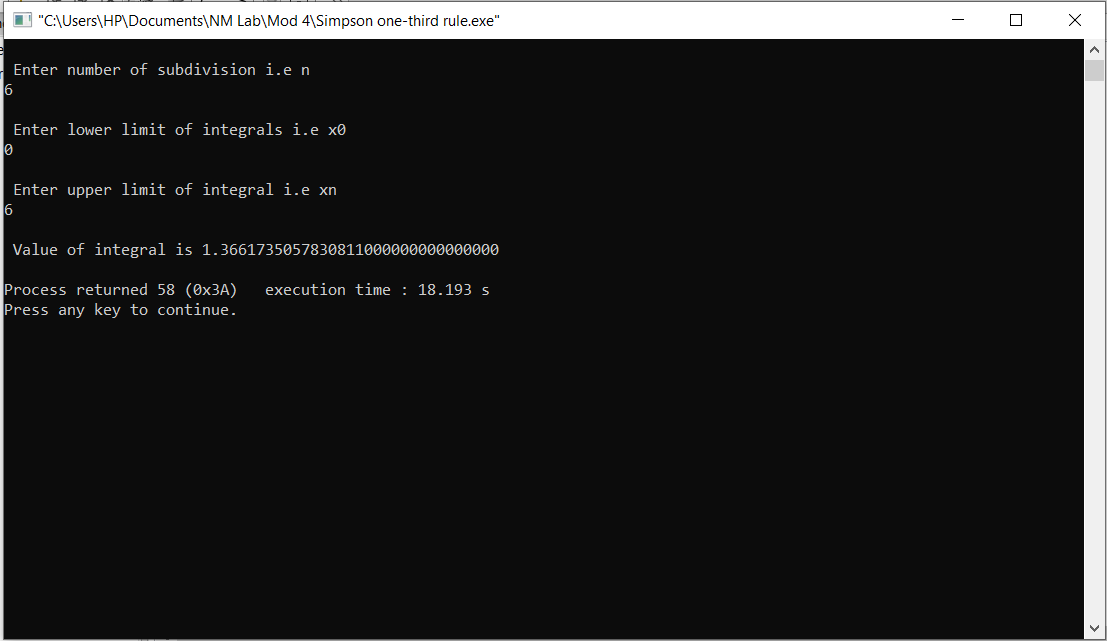
for(i=3;i<=n-1;i+=2)

s+=4\*y(x0+i\*h)+2\*y(x0+(i-1)\*h);

sum=s\*(h/3);

printf("\n Value of integral is %0.31f \n",sum);

}



**Trapezoidal rule**

**Algorithm:**

1. Start

2. Input Lower limit a

3. Input Upper Limit b

4. Input number of sub intervals n

5. h=(b-a)/n

6.sum=0

7.sum=fun(a)+fun(b)

8.fori=1; i<n; i++

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

10. End Loop i

11.result =sum\*h/2;

12. Print Output result

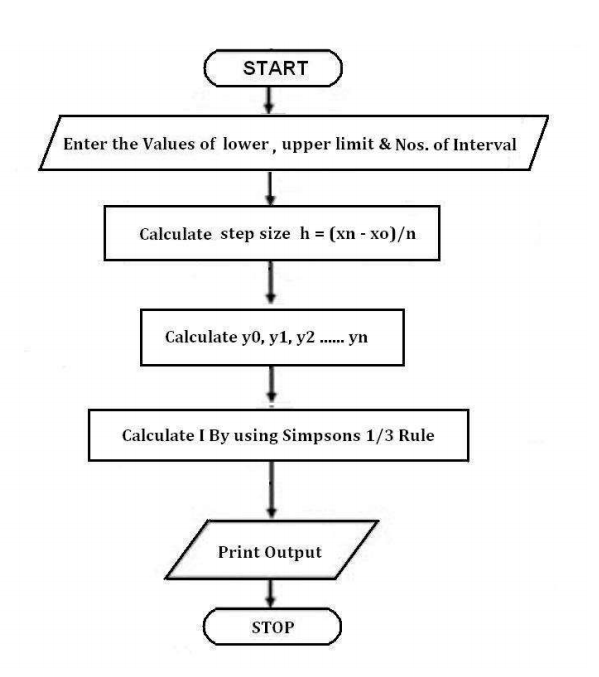
13. End of Program

14. Start of Section fun

15.temp = 1/(1+(x\*x))

16. Return temp

17. Stop



**code:**

#include<math.h>

#define f(x) 1/(1+pow(x,2))

int main()

{

float lower, upper, integration=0.0, stepSize, k;

int i, subInterval;

printf("Enter lower limit of integration: ");

scanf("%f", &lower);

printf("Enter upper limit of integration: ");

scanf("%f", &upper);

printf("Enter number of sub intervals: ");

scanf("%d", &subInterval);

stepSize = (upper - lower)/subInterval;

integration = f(lower) + f(upper);

for(i=1; i<= subInterval-1; i++)

{

k = lower + i\*stepSize;

integration = integration + 2 \* f(k);

printf("%f\t %f\n",k,integration);

}

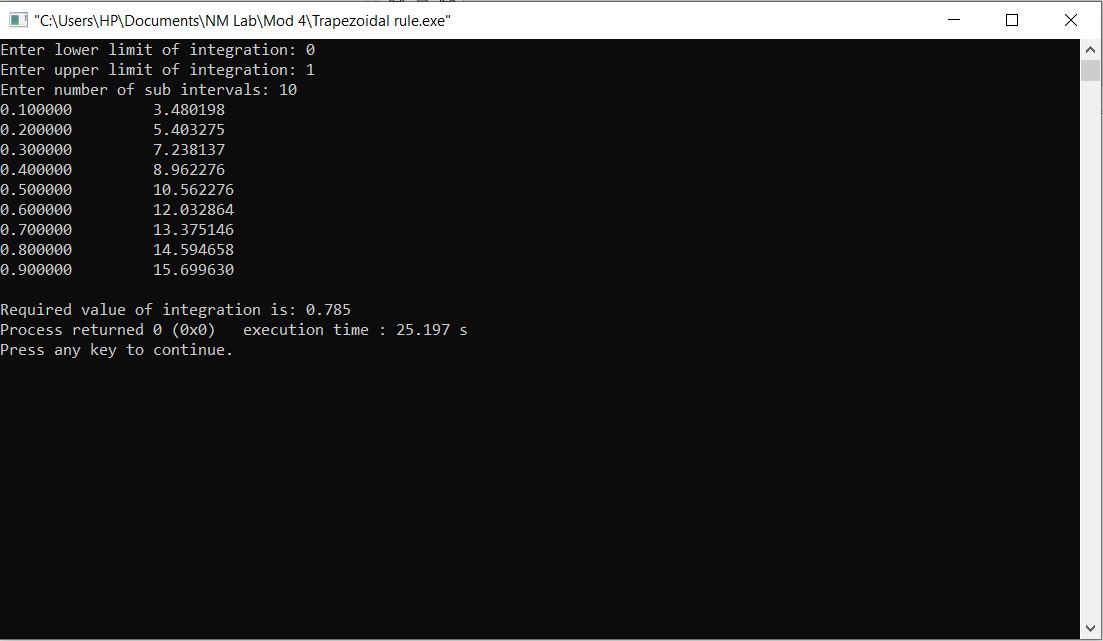
integration = integration \* stepSize/2;

printf("\nRequired value of integration is: %.3f", integration);

getch();

return 0;

}



**Euler’s Method**

**Algorithm:**

Step 1. Start;

Step 2. Input function f(x, y);

Step 3. Read x0, y0, xn, h;

/\* x0, y0 are initial values and xn is the last value of x

where the process will terminate, h is the step size \*/

Step 4. for x = x0 to xn step h do

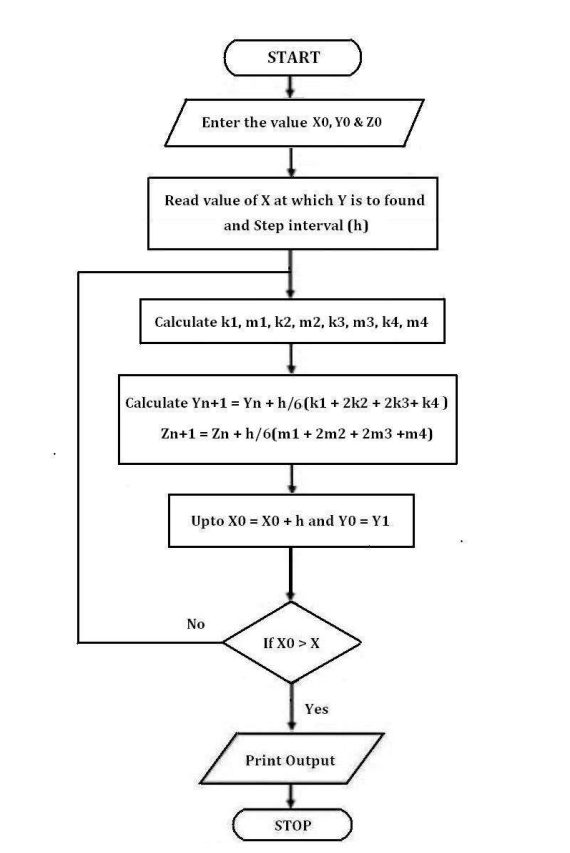
y = y0 + h \* f(x, y0);

Print x, y;

y0 = y;

end for loop;

Step 5. Stop;



**Code:**

include<stdio.h>

#include<conio.h>

float fun(float x,float y)

{

float f;

f=y\*y-x\*x;

return f;

}

main()

{

float a,b,x,y,h,t,k;

clrscr();

printf("\nEnter x0,y0,h,xn: ");

scanf("%f%f%f%f",&a,&b,&h,&t);

x=a;

y=b;

printf("\n x\t y\n");

while(x<=t)

{

k=h\*fun(x,y);

y=y+k;

x=x+h;

printf("%0.3f\t%0.3f\n",x,y);

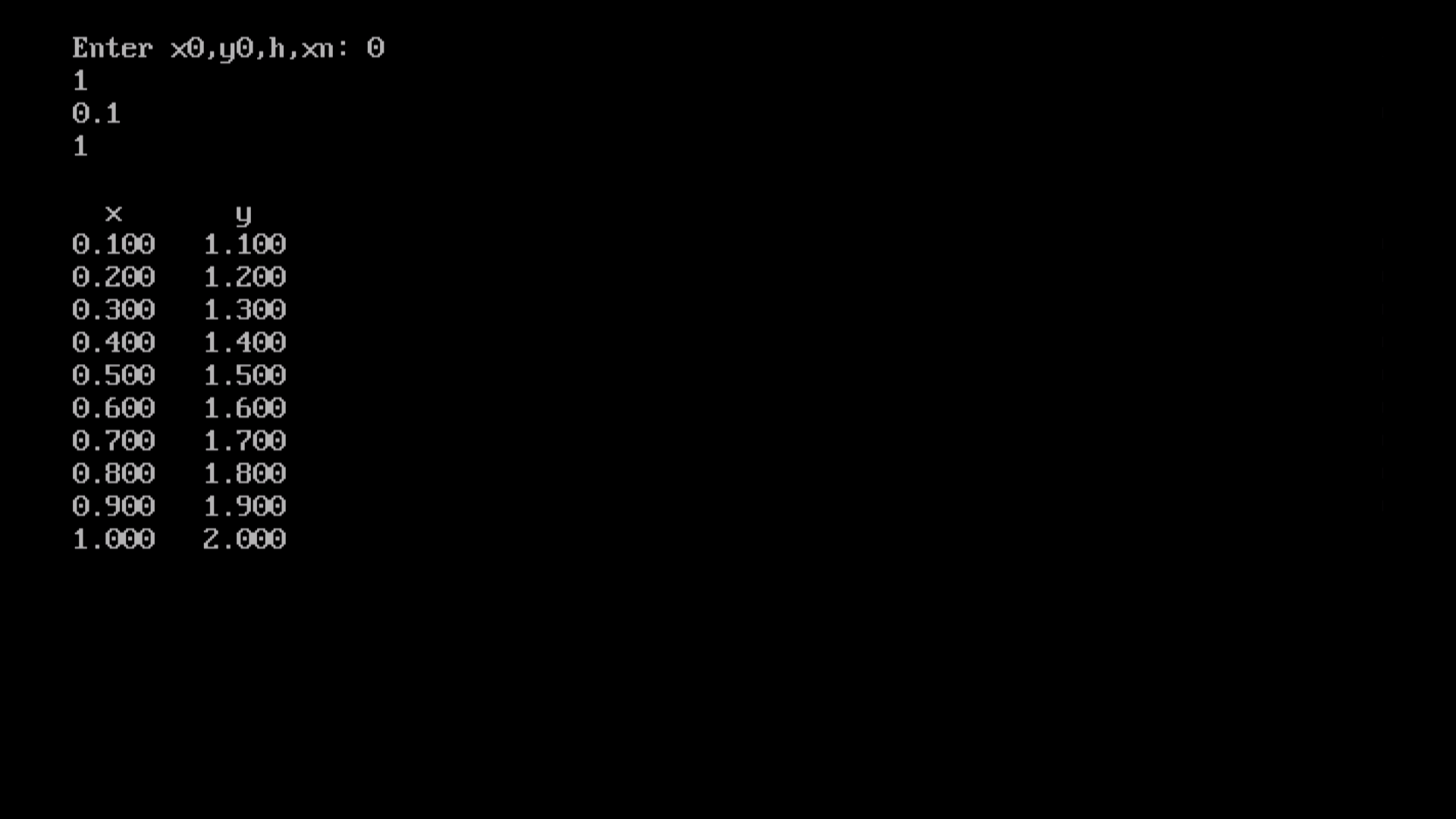
}

getch();

return 0;

}

**Output:**



**Runge-Kutta 4th order method**

**Algorithm:**

1. Start

2. Define function f(x,y)

3. Read values of initial condition(x0 and y0),

number of steps (n) and calculation point (xn)

4. Calculate step size (h) = (xn - x0)/n

5. Set i=0

6. Loop

k1 = h \* f(x0, y0)

k2 = h \* f(x0+h/2, y0+k1/2)

k3 = h \* f(x0+h/2, y0+k2/2)

k4 = h \* f(x0+h, y0+k3)

k = (k1+2\*k2+2\*k3+k4)/6

yn = y0 + k

i = i + 1

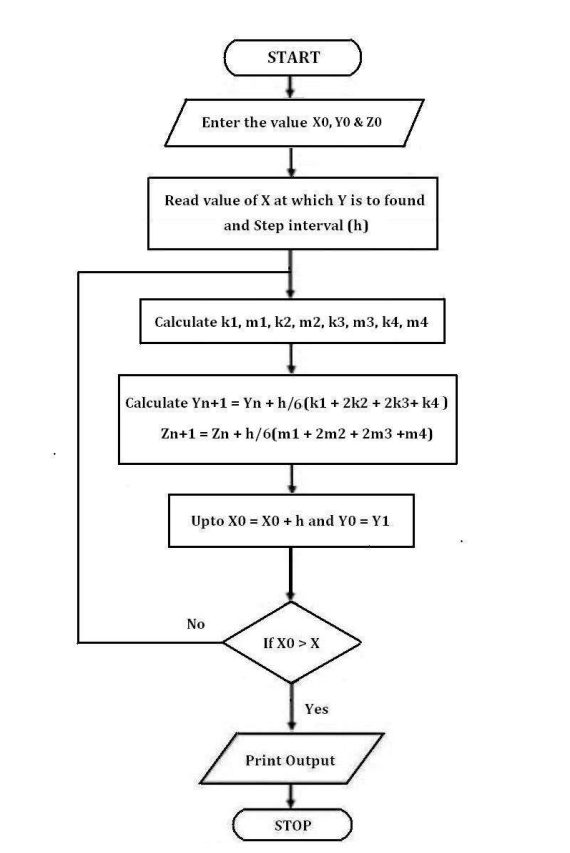
x0 = x0 + h

y0 = yn

While i < n

7. Display yn as result

8. Stop



**Code:**

#include<stdio.h>

#include <math.h>

#include<conio.h>

#define F(x,y) y-x

void main()

{

double y0,x0,y1,n,h,f,k1,k2,k3,k4;

clrscr();

printf("\nEnter the value of x0: ");

scanf("%lf",&x0);

printf("\nEnter the value of y0: ");

scanf("%lf",&y0);

printf("\nEnter the value of h: ");

scanf("%lf",&h);

printf("\nEnter the value of last point: ");

scanf("%lf",&n);

for(; x0<n; x0=x0+h)

{

f=F(x0,y0);

k1 = h \* f;

f = F(x0+h/2,y0+k1/2);

k2 = h \* f;

f = F(x0+h/2,y0+k2/2);

k3 = h \* f;

f = F(x0+h/2,y0+k2/2);

k4 = h \* f;

y1 = y0 + ( k1 + 2\*k2 + 2\*k3 + k4)/6;

printf("\n\n k1 = %.4lf ",k1);

printf("\n\n k2 = %.4lf ",k2);

printf("\n\n k3 = %.4lf ",k3);

printf("\n\n k4 = %.4lf ",k4);

printf("\n\n y(%.4lf) = %.3lf ",x0+h,y1);

y0=y1;

}

getch();

}

