

Algorithm:

- ```

 if (f0 * f2 < 0)
 {
 x1 = x2;
 f1 = f2;

 }
 else
 {
 x0 = x2;
 f0 = f2;

 }
 step = step + 1;
 }while(fabs(f2)>e);
 printf("\nRoot is: %f", x2);
 getch();
}

```

**Algorithm:**

- 1.

**Algorithm:**

- ```
Program:
#include<stdio.h>
#include<conio.h>
#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);
    }
    integration = integration * stepSize/2;
    printf("\nRequired value of integration is: %.3f", integration);
    getch();
    return 0;
}
```

Algorithm:

- 3.

Algorithm:

- 5.

Laplace's Equation	
Algorithm:	
1.	Start
2.	Say n as dimension of plate
3.	Say tl, tr, tu, tb as temperature of left, right, upper and bottom part of plate
4.	Construct coefficient matrix as
5.	Construct RHS vector
6.	Use Gauss-Seidal/ jacobi iteration method to solve equation.
7.	Display the solution vector
8.	Stop

```

Program:
#include<stdio.h>
#include<math.h>
#define S 4
typedef float newvar[S+1][S+1];
void entrow(int i,newvar u)
{
    int j;
    printf("\n Enter the value of u[%d,j],j=1,%d\n",i,S);
    for(j=1;j<=S;j++)
        scanf("%f",&u[i][j]);
}

void entcol(int j, newvar u)
{
    int i;
    printf("Enter the value of u[i,%d],""i=2,%d\n",j,S-1);
    for(i=2;i<=S-1;i++)
        scanf("%f",&u[i][j]);
}

void oput(newvar u, int wd, int prsn)
{
    int i,j;
    for(i=1;i<=S;i++)
    {
        for(j=1;j<=S;j++)
            printf("%d,%d,%f",wd, prsn, u[i][j]);
        printf("\n");
    }
}

nt main()
{
    newvar u;
    float mer, ar, e, t;
    int i,j,itr,maxitr;
    for(i=1;i<=S;i++)
        for(j=1;j<=S;j++)
            u[i][j]=0;
    printf("\n Enter the Boundary Condition\n");
    entrow(1,u); entrow(S,u);
    entcol(1,u); entcol(S,u);
    printf(" Enter the allowed error and maximum number of iteration : ");
    scanf("%f%f",&ar,&maxitr);
    for(itr=1;itr<=maxitr;itr++)
    {
        mer=0;
        for(i=2;i<=S-1;i++)
        {
            for(j=2;j<=S-1;j++)
            {
                t=(u[i-1][j]+u[i+1][j]+u[i][j+1]+u[i][j-1])/4;
                e=fabs(u[i][j]-t);
                if(e>mer)
                    mer=e;
                u[i][j]=t;
            }
        }
        printf(" Iteration Number %d\n",itr);
        oput(u,9,2);
        if(mer<=ar)
        {
            printf(" After %d iteration \n The solution : \n",itr);
            oput(u,8,1);
            return 0;
        }
    }
    printf(" Sorry! The number of iteration is not sufficient");
    return 1;
}

```

```

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

for(i=n-1;i>=1;i--)
{
    x[i] = a[i][n+1];
    for(j=i+1;j<=n;j++)
    {
        x[i] = x[i] -
        a[i][j]*x[j];
    }
    x[i] = x[i]/a[i][i];
}
printf("\nSolution:\n");
for(i=1;i<=n;i++)
{
    printf("x[%d] = %0.3f\n",i, x[i]);
}
getch();
return(0);
}

```

Euler's Method	
Algorithm:	
1.	Start
2.	Define function f(x,y)
3.	Read value of initial condition (x0,y0),number of steps (n) and calculation point(xn)
4.	Calculate step size (h) = (xn- x0)/b
5.	Set i=0
6.	Loop yn=y0 + h * f(x0 + i*h,y0) y0=yn i = i +1 while i < n
7.	Display yn as result
8.	Stop

```

Program:
#include<stdio.h>
#include<conio.h>
#define f(x,y) x+y
int main()
{
    float x0, y0, xn, h, yn, slope;
    int i, n;
    printf("Enter Initial Condition\n");
    printf("x0 = ");
    scanf("%f", &x0);
    printf("y0 = ");
    scanf("%f", &y0);
    printf("Enter calculation point xn = ");
    scanf("%f", &xn);
    printf("Enter number of steps: ");
    scanf("%d", &n);
    h = (xn-x0)/n;
    printf("\nx0\ty0\tslope\tyn\n");
    printf("-----\n");
    for(i=0; i < n; i++)
    {
        slope = f(x0, y0);
        yn = y0 + h * slope;
        printf("%.4f\t%.4f\t%.4f\n",x0,y0,slope,yn);
        y0 = yn;
        x0 = x0+h;
    }
    printf("\nValue of y at x = %0.2f is %0.3f",xn, yn);
    getch();
    return 0;
}

```

```

Newton Rapson Method:
Program:
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
#define f(x) 3*x - cos(x) -1
#define g(x) 3 + sin(x)
int main()
{
    float x0, x1, f0, f1, g0, e;
    int step = 1, N;
    printf("\nEnter initial guess:\n");
    scanf("%f", &x0);
    printf("Enter tolerable error:\n");
    scanf("%f", &e);
    printf("Enter maximum iteration:\n");
    scanf("%d", &N);
    printf("\nStep\t\tx0\t\tf(x0)\t\tx1\t\tf(x1)\n");
    do
    {
        g0 = g(x0);
        f0 = f(x0);
        if(g0 == 0.0)
        {
            printf("Math Error.");
            exit(0);
        }
        x1 = x0 - f0/g0;

        printf("%d\t\t%f\t\t%f\t\t%f\n",step,x0,f0,x1,f1);
        x0 = x1;
        step = step+1;
        if(step > N)
        {
            printf("Not Convergent.");
            exit(0);
        }
        f1 = f(x1);
    }while(fabs(f1)>e);
    printf("\nRoot is: %f", x1);
    getch();
}

```

```

Lagrange Interpolation Method
Algorithm:
1. Start
2. Read number of data (n)
3. Read data Xi and Yi for i=1 to n
4. Read value of independent variables say xp whose corresponding value of dependent say yp is to be determined
5. Initialize yp = 0
6. For i=1 to n set p = 1 for j=1 to n if i ≠ j then calculate p = p *(xp-Xj)/(yp-Yj)
End if next j calculate yp = yp + p * Yi next i
7. Display value of yp as interpolated value
8. Stop

Program:
#include<stdio.h>
#include<conio.h>
void main()
{
    float x[100], y[100], xp, yp=0, p;
    int i,j,n;
    printf("Enter number of data: ");
    scanf("%d", &n);
    printf("Enter data:\n");
    for(i=1;i<=n;i++)
    {
        printf("x[%d] = ", i);
        scanf("%f", &x[i]);
        printf("y[%d] = ", i);
        scanf("%f", &y[i]);
    }
    printf("Enter interpolation point: ");
    scanf("%f", &xp);
    for(i=1;i<=n;i++)
    {
        p=1;
        for(j=1;j<=n;j++)
        {
            if(i!=j)
            {
                p = p* (xp -
                x[j])/(x[i] - x[j]);
            }
        }
        yp = yp + p * y[i];
    }
    printf("Interpolated value at %.3f is %.3f.", xp, yp);
    getch();
}

```

Poisson's Equations

Algorithm:

1.

Start
2.

Say n as dimensions of plate
3.

Say tl, tr, tu, tb as temperature of left, right, upper and bottom part of plate
4.

Define function g(x,y)
5.

Say h as dimension of square grid
6.

Construct coefficient matrix
7.

Construct RHS vector
8.

Set b[k]=h*h*g(l,j)
9.

Set k=0;
10.

Use Gauss-seidal /Jacobi iteration method
11.

Display solution vector.
12.

Stop

Program:

```
#include<stdio.h>
#include<math.h>
#define g(x,y) 2*(x)*(x)*(y)*(y)
nt main()
{
    int l,j,k,n;
    float
sum,error,E[10],a[10][10],b[10],new_x[10],old_x[10],tl,tr,tu,tb,h;
    printf("enter dimension of plate:\n ");
    scanf("%d",&n);
    printf("dimensions of grid:\n");
    scanf("%f",&h);
    printf("enter the temperatures at left, right, bottom and
upper part of plate:\n ");
    scanf("%f%f%f%f",&tl,&tr,&tb,&tu);
    for(i=0;i<=n;i++)
        a[i][i]= -4;
    for(i=0;i<=n;i++)
        a[i][n-i]=0;
    for(i=0;i<=n;i++)
        for(j=0;j<=n;j++)
        {
            if(i!=j && j!=n-1)
                a[i][j]=1;
            }
            k=0;
            for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                    b[k++]=h*h*g(l,j);
            k=0;
            for(i=1;i<=n;i++)
            {
                for(j=1;j<=n;j++)
                {
                    if(i-1==0)
                        b[k]=b[k]-tl;
                    if(i+1==n)
                        b[k]=b[k]-tr;
                    if(j-1==0)
                        b[k]=b[k]-tb;
                    if(j+1==n)
                        b[k]=b[k]-tu;
                    k++;
                }
            }
            printf("enter accuracy limit:\n");
            scanf("%f",&error);
            for(i=0;i<=n;i++)
            {
                new_x[i]=0;
            }
            while(1)
            {
                for(i=0;i<=n;i++)
                {
                    sum=b[i];
                    for(j=0;j<=n;j++)
                    {
                        if(i!=j)
                            sum=sum-
a[i][j]*new_x[j];
                    }
                    old_x[i]=new_x[i];
                    new_x[i]=sum/a[i][i];
                    E[i]=fabs(new_x[i]-
old_x[i])/fabs(new_x[i]);
                }
                for(i=0;i<=n;i++)
                {
                    if(E[i]>error)
                        break;
                }
            }
        }
    }
```

The Bisection Method

$c = \frac{a+b}{2}$

The Method of False Position (Regula Falsi)

$c = b - \frac{f(b) \cdot (b-a)}{f(b) - f(a)}$

Newton-Raphson Method

$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

Fixed Point Iteration

$x_{n+1} = g(x_n)$

Lagrange's Polynomials

$P(x) = \sum_{i=0}^n y_i \prod_{\substack{0 \leq j \leq n \\ j \neq i}} \frac{x-x_j}{x_i-x_j}$

Newton's Interpolation using Difference

$P(x) = y_0 + (x-x_0)\Delta y_0 + (x-x_0)(x-x_1)\Delta^2 y_0 + \dots$

Cubic Spline Interpolation

$S_i(x) = a_i + b_i(x-x_i) + c_i(x-x_i)^2 + d_i(x-x_i)^3$

Least Squares Method for Linear Data

$\hat{\beta} = (X^T X)^{-1} X^T y$

Newton's Differentiation Formulas

- Forward Difference Formula:

$f'(x) \approx \frac{f(x+h)-f(x)}{h}$

- Backward Difference Formula:

$f'(x) \approx \frac{f(x)-f(x-h)}{h}$

- Central Difference Formula:

$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$

- Trapezoidal Rule:

$\int_a^b f(x) dx \approx \frac{h}{2} [f(a) + 2 \sum_{i=1}^{n-1} f(x_i) + f(b)]$

Simpson's 1/3 Rule:

$\int_a^b f(x) dx \approx \frac{h}{3} [f(a) + 4 \sum_{\text{odd}} f(x_i) + 2 \sum_{\text{even}} f(x_i) + f(b)]$

Jacobi Iteration Method

$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$

Gauss-Seidel Iteration Method

$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right)$

Euler's Method

$y_{n+1} = y_n + hf(x_n, y_n)$

Runge-Kutta Method (4th Order)

$k_1 = hf(x_n, y_n)$

$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2})$

$k_3 = hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2})$

$k_4 = hf(x_n + h, y_n + k_3)$

$y_{n+1} = y_n + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)$

Error: Error refers to the difference between the exact value and the approximate value.

Taxonomy of Errors:

- **Absolute Error:** The difference between the true value and the approximated value.
- **Relative Error:** The ratio of the absolute error to the true value.
- **Round-off Error:** Errors due to rounding off numbers.
- **Truncation Error:** Errors that result from approximating an infinite process by a finite one.
- **Propagation of Error:** Errors that occur when approximate numbers are used in further calculations.

a) **Linear Interpretation:** Linear interpolation is a method to estimate values between two known values. If you have two points (x_0, y_0) and (x_1, y_1) , the interpolated value at x is:

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

b) **Boundary value problems:** Boundary value problems (BVPs) are differential equations where the solution is determined based on values at the boundaries of the domain. Unlike initial value problems, BVPs require conditions to be specified at more than one point.

c) **Partial Differential Equations:** Partial Differential Equations (PDEs) involve multiple independent variables and their partial derivatives. PDEs describe various phenomena such as heat conduction, wave propagation, and fluid dynamics.

```
if(i==(n+1))
break;
else
continue;
```

```
}
printf("solution:\n");
for(i=0;i<=n;i++)
printf("x%d=%f\n",i+1,new_x[i]);

return 0;
}
```

Newton's Forward Interpolation

Algorithms:

1. Start
2. Say n
3. Interpolated value needed say xp
4. Read n data points, say $x[i]$ and $f[x[i]]$
5. Set $h=x[1]-x[0]$ and $s=(xp-x[0])/h$
6. Calculate first forward difference
7. Calculate second to nth forward differences
8. Set $v=fd[0]$ and set $p=1$
9. Calculate interpolated value
10. Print the interpolated value v at x_p
11. Stop

Program:

```
#include<stdio.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 = %1f, corresponding y = %f\n",x,yp);
}
```