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

#define f(x) 1/(x)

int main(){
   float x0 ,xn, T[10][10],h,sm,sl,a;
   int i,k,c,r,m,p,q;

   printf("Enter Lower and Upper Limit\n");
   scanf("%f %f",&x0,&xn);
   printf("Enter p and q of required T(p,q) \n");
   scanf("%d %d",&p,&q);

   h = xn-x0;
   T[0][0] = h/2*((f(x0))+(f(xn)));

   for(i=1;i<=p;i++){
      sl = pow(2,i-1);
      sm = 0;
      for(k=1;k<=sl;k++){
         a = x0+(2*k-1)*h/pow(2,i);
         sm = sm+(f(a));
      }
      T[i][0] = T[i-1][0]/2+sm*h/pow(2,i);
   }
   for(c=1;c<=p;c++){
      for(k=1;k<=c&&k<=q;k++){
```

```
      m=c-k;
      T[m+k][k] = (pow(4,k)*T[m+k][k-1]-T[m+k-1][k-1])/(pow(4,k)-1);
    }
  }

  printf("\n Romberg Estimate of Integration = %f\n",T[p][q]);
  getchar();
 getchar();
  return 0;
}
```

Taylor series method

```
#include<stdio.h>

#include<math.h>

int fact(int n){
  if(n == 1)
    return 1;
  else
    return (n * fact(n - 1));
}

int main(){
  float x,x0,yx0,yx,fdy,sdy,tdy;

  printf("Enter initial values of x and y \n");
  scanf("%f %f",&x0,&yx0);
```

```
    printf("Enter x at which function to be evaluated\n");
    scanf("%f",&x);

    fdy = (x0)*(x0)+(yx0)*(yx0);
    sdy = 2*(x0) + 2*(yx0)*fdy;
    tdy = 2+2*yx0*sdy+2*fdy*fdy;

    yx = yx0+(x-x0)*fdy+(x-x0)*sdy/fact(2)+(x-x0)*(x-x0)*(x-x0)*tdy/fact(3);
    printf("Function Value at x = %f is %f\n",x,yx);
    getchar();
   getchar();
    return 0;
}
```

Picard method

```
#include<stdio.h>

#include<math.h>

#define y1(x) exp(x) + pow(x,2) - 3*x
#define y2(x) 1 + x + pow(x,2)/2 + pow(x,3)/6 + pow(x,2) - 3*x
#define y3(x) 1 + x + pow(x,2)/2 + pow(x,3)/6 + pow(x,4)/24 + pow(x,2) - 3*x

int main(){
    float x, x0, y0, y;

    printf("Enter initial values of x and y: \n");
    scanf("%f %f", &x0, &y0);

    printf("Enter x at which function is to be evaluated: \n");
    scanf("%f", &x);

    y = y0;
    y = y0 + y1(x);
    y = y0 + y2(x);
    y = y0 + y3(x);

    printf("Function Value at x = %f is %f\n", x, y);
    getchar();
   getchar();
    return 0;
}
```

Heun’s method

```
#include<stdio.h>
#include<math.h>
#define f(x,y) 2*y/x
int main(){
    float x,xp,x0,y0,y,h,m1,m2;

    printf("Enter initial value of x and y:\n");
    scanf("%f %f",&x0,&y0);

    printf("Enter x at which function to be evaluated: \n");
    scanf("%f",&xp);
    printf("Enter the step size: \n");
    scanf("%f",&h);

    y=y0;
    x=x0;

    for(x=x0;x<xp;x=x+h){
        m1 = f(x,y);
        m2=f(x+h,y+h*m1);
        y=y+h/2*(m1+m2);
    }
    printf("Function Value at x = %f is %f \n",xp,y);
    return 0;
}
```

RK method

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

#define f(x, y) (2 * (x) + (y))

int main() {
    float x, xp, x0, y, y0, h, m1, m2, m3, m4;

    printf("Enter initial values (x0, y0): ");
    scanf("%f %f", &x0, &y0);
    printf("Enter x at which function is to be evaluated (xp): ");
    scanf("%f", &xp);

    printf("Enter step size (h): ");
    scanf("%f", &h);

    y = y0;
    x = x0;

    for (x=x0 ; x < xp; x = x+h) {
        m1 = f(x, y);
        m2 = f(x + h / 2, y + (h / 2) * m1);
        m3 = f(x + h / 2, y + (h / 2) * m2);
        m4 = f(x + h, y + h * m3);

        y = y + (h / 6) * (m1 + 2 * m2 + 2 * m3 + m4);
    }

    printf("Function value at x = %.2f is y = %.4f\n", xp, y);
    return 0;
}
```

Shooting method

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

// Define functions f1 and f2
```

```
#define f1(x, y, z) (z)
#define f2(x, y, z) (6 * (x))

int main() {
    float xa, xb, ya, yb, z0, x, h, xp, sol, error, E;
    float V[3], g[3], ny, nz;
    int i;

    // Input boundary conditions
    printf("Enter Boundary Conditions (xa, ya, xb, yb): ");
    scanf("%f %f %f %f", &xa, &ya, &xb, &yb);

    // Input the x value where function is required
    printf("Enter x at which value is required (xp): ");
    scanf("%f", &xp);

    // Input step size
    printf("Enter the step size (h): ");
    scanf("%f", &h);

    // Input accuracy limit
    printf("Enter accuracy limit (E): ");
    scanf("%f", &E);

    x = xa;
    V[1] = ya;
    g[1] = (yb - ya) / (xb - xa);  // Initial guess
```

```
printf("g1 = %f\n", g[1]);

while (x < xb) {
    ny = V[1] + (f1(x, V[1], g[1])) * h;
    nz = g[1] + (f2(x, V[1], g[1])) * h;
    x = x + h;
    V[1] = ny;
    g[1] = nz;

    if (x == xp)
        sol = V[1];
}

V[1] = V[1];

if (V[1] < yb)
    g[2] = g[1] = 2 * g[1];
else
    g[2] = g[1] = (1.0 / 2.0) * g[1];

// Iterative shooting method loop
while (1) {
    x = xa;
    V[2] = ya;
    g[2] = g[1] - ((V[2] - yb) / (V[2] - V[1])) * (g[2] - g[1]);

    while (x < xb) {
        ny = V[2] + (f1(x, V[2], g[2])) * h;
```

```
            nz = g[2] + (f2(x, V[2], g[2])) * h;
            x = x + h;
            V[2] = ny;
            g[2] = nz;

            if (x == xp)
                sol = V[2];
        }

        error = fabs(V[2] - yb) / yb;
        V[1] = V[2];
        g[1] = g[2];

        if (error < E) {
            printf("y(%.2f) = %.4f\n", xp, sol);
            break;
        }
    }
    return 0;
}
```

Laplace equation

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

int main() {
    int n,i, j, k;
    float sum, error, E[10], a[10][10], b[10], new_x[10], old_x[10];
```

```
printf("Enter the dimension of plate (n,n): ");
scanf("%d", &n);

printf("Enter temperatures at left, right, top, and bottom: ");
float tL, tR, tT, tB;
scanf("%f %f %f %f", &tL, &tR, &tT, &tB);

// Initialize matrix
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-i)){
            a[i][j] = 1;
        }

    }
}
for(i=0;i<=n;i++){
    b[i] = 0;
    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) == 0){
            b[k] = b[k] - tR;
        }
        if((j-1) == 0){
            b[k] = b[k] - tB;
        }
        if((j+1) == n){
            b[k] = b[k] - tT;
        }
        k++;
    }
}

// Accuracy limit
printf("Enter accuracy limit: ");
scanf("%f", &error);

// Iterative calculation
while (1) {
    for (i = 0; i < n; i++) {
        sum = b[i];
        for (j = 0; j < n; j++)
            sum -= a[i][j] * old_x[j];
            new_x[i] = sum / a[i][i];
            E[i] = fabs(new_x[i] - old_x[i]) / fabs(new_x[i]);
        }

        // Check if error is within the limit
        for (i = 0; i < n; i++) {
            if (E[i] > error)
                break;
        }

        if (i == n)
            break;

        for (i = 0; i < n; i++)
            old_x[i] = new_x[i];
    }

    // Output solution
    printf("Solution:\n");
    for (i = 0; i < n; i++)
        printf("x[%d] = %.6f\n", i, new_x[i]);

        return 0;
}
```

POission equation

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

#define MAX 10  // Maximum grid size

int main() {
    int n, i, j, k;
    float sum, error, E[MAX], a[MAX][MAX], b[MAX], new_x[MAX], old_x[MAX];

    printf("Enter the dimension of plate (n,n): ");
    scanf("%d", &n);

    printf("Enter temperatures at left, right, top, and bottom: ");
    float tL, tR, tT, tB;
    scanf("%f %f %f %f", &tL, &tR, &tT, &tB);

    // Initialize coefficient matrix
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            a[i][j] = (i == j) ? -4 : 0;
            if (j == i - 1 || j == i + 1)
                a[i][j] = 1;
        }
    }

    // Initialize right-hand side vector
    for (i = 0; i < n; i++) {
        b[i] = 0;
    }

    // Apply boundary conditions
    k = 0;
    for (i = 1; i < n - 1; i++) {
        for (j = 1; j < n - 1; j++) {
            if (i - 1 == 0)
                b[k] -= tL;
            if (i + 1 == n - 1)
                b[k] -= tR;
            if (j - 1 == 0)
                b[k] -= tB;
            if (j + 1 == n - 1)
                b[k] -= tT;
            k++;
        }
    }

    // Accuracy limit
    printf("Enter accuracy limit: ");
    scanf("%f", &error);

    // Initialize guess values
    for (i = 0; i < n; i++) {
        old_x[i] = 0;
    }

    // Iterative calculation using Gauss-Seidel Method
```

```
while (1) {
    for (i = 0; i < n; i++) {
        sum = b[i];
        for (j = 0; j < n; j++) {
            if (i != j)
                sum -= a[i][j] * old_x[j];
        }
        new_x[i] = sum / a[i][i];
        E[i] = fabs(new_x[i] - old_x[i]) / fabs(new_x[i]);
    }

    // Check if error is within the limit
    int converged = 1;
    for (i = 0; i < n; i++) {
        if (E[i] > error) {
            converged = 0;
            break;
        }
    }

    if (converged)
        break;

    for (i = 0; i < n; i++)
        old_x[i] = new_x[i];
}

// Output solution
```

```
    printf("Solution:\n");
    for (i = 0; i < n; i++)
        printf("x[%d] = %.6f\n", i, new_x[i]);

    return 0;
}
```