Bisection:

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
#define f(x) (pow(x,3)-x-1)
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
double a, b, c, ep;
m:
cout<<"Enter the initial guesses:\na";
cin>>a;
cout<<"\nb=";
cin>>b;
cout<<"Enter the desired accuracy"<<endl;
cin>>ep;
if(f(a)*f(b)>0)
{
cout<<"Please enter different
guesses:"<<endl;
goto m;
}
else
{
do
{
c=(a+b)/2.0;
if(f(c)==0)
{
cout<<"The root of the equation is"<<c;
```

```
break;
}
if(f(a)*f(c)>0)
{
a=c;
}
else
{
b=c;
}
}
while(fabs(a-b)>ep);
cout<<"The root of the equation
is"<<c<<endl;
}
return 0;
}
```

Iteration:

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
#define f(x) ((1+cos(x))/3)
#define df(x) (-sin(x)/3)
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
double x0, x, ep;
cout<<"Enter initial approximation:\n";
cin>>x0;
cout<<"Enter desired accuracy:\n";
cin>>ep;
if (fabs(df(x0))<1)
{
do
{
x = x0;
x0 = f(x);
}
while(fabs(x-x0)>ep);
cout<< endl<<"Root is: "<<x0<<endl;
}
else
{
cout<<"Initial approximation isn't convergent!
Please choose another approximation."<<endl;
}
return 0;
}
```

Newton Raphson (Single Equation):

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
#define f(x) (pow(x,3)-2*x-5)
#define df(x) (3*pow(x,2)-2)
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
double x, x0, fx, fx0, ep;
cout<<"Enter the initial guesses:\n";
m:
cin>>x0;
cout<<"Enter the desired accuracy:\n"<<endl;
cin>>ep;
do
{
x=x0;
if(df(x)!=0)
x0=x-(f(x)/df(x));
else
{
cout<<"Enter another guess:"<<endl;
goto m;
}
}
while(fabs(x-x0)>ep);
cout<<"The root of the equation is:
"<<x0<<endl;
return 0;
}
```

Newton Raphson (System of Equation):

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
#define f(x,y) (3*pow(x,2)*y-10*x+7)
#define g(x,y) (pow(y,2)-5*y+4)
#define dfx(x,y) (6*x*y-10)
#define dfy(x,y) (3*pow(x,2))
#define dgx(x,y) (0)
#define dgy(x,y) (2*y-5)
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
double x, x0, y, y0, fx, fx0, d, h, k, ep;
cout<<"Enter the initial guesses x0 and y0
respectively:\n";
m:
cin>>x0>>y0;
cout<<"Enter the desired accuracy:\n"<<endl;
cin>>ep;
do
{
x=x0,y=y0;
d=dfx(x,y)*dgy(x,y)-dgx(x,y)*dfy(x,y);
if (d==0)
{
cout<<"The system doesn't converge for this
initial guess. Please, give new guess:"<<endl;
goto m;
}
else
{
h=(g(x,y)*dfy(x,y)-f(x,y)*dgy(x,y))/d;
k=(f(x,y)*dgx(x,y)-g(x,y)*dfx(x,y))/d;
```

```
x0=x+h, y0=y+k;
}
}
while(fabs(x-x0)>ep || fabs(y-y0)>ep);
cout<<"Root of the equation: x= "<<x0<<" y=
"<<y0<<endl;
return 0;
}
```

Newton's Backward Interpolation:

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i, j, n;
cout<<"Enter the number of values to be
entered\n";
cin>>n;
double x[n], y[n][n];
cout<<"Enter the of values of x\n";
for(i=0; i<n; i++)
cin>>x[i];
cout<<"Enter the of values of y\n";
for(i=0; i<n; i++)
cin>>y[i][0];
//Calculate difference table
for(j=1; j<n; j++)
{
for(i=j; i<n; i++)
{
y[i][j]=y[i][j-1]-y[i-1][j-1];
}
}
//Print difference table
cout<<"\nThe backward difference table is as
follows:\n\n";
cout<<"x"<<setw(10)<<"y"<<setw(10);
for(i=1; i<n; i++)
cout<<"d"<<i<<"y"<<setw(10);
```

```
cout<<"\
n----------------------------------------
----------\n";
for(i=0; i<n; i++)
{
cout<<x[i]<<setw(10);
for(j=0; j<=i; j++)
{
cout<<y[i][j]<<setw(10);
}
cout<<"\n";
}
//Code of interpolation
double xn, h, p, sum=y[n-1][0], temp=1;
h=x[1]-x[0];
cout<<"Enter the values of x at which y to be
calculated\n";
cin>>xn;
p=(xn-x[n-1])/h;
for(j=1; j<n; j++)
{
temp=temp*(p+j-1)/j;
sum=sum+temp*y[n-1][j];
}
cout<<"The value of y at x= "<<xn<<" is:
"<<sum;
return 0;
}
```

Newton's Forward Interpolation:

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i, j, k, n;
cout<<"Enter the number of values to be
entered\n";
cin>>n;
double x[n], y[n][n];
cout<<"Enter the values of x\n";
for(i=0; i<n; i++)
cin>>x[i];
cout<<"Enter the values of y\n";
for(i=0; i<n; i++)
cin>>y[i][0];
//Calculate difference table
for(j=1; j<n; j++)
{
for(i=0; i<n-j; i++)
{
y[i][j]=y[i+1][j-1]-y[i][j-1];
}
}
//Print difference table
cout<<"\nThe forward difference table is as
follows:\n\n";
cout<<"x"<<setw(10)<<"y"<<setw(10);
for(i=1; i<n; i++)
```

```
cout<<"d"<<i<<"y"<<setw(10);
cout<<"\
n----------------------------------------
----------\n";
k=n;
for(i=0; i<n; i++)
{
cout<<x[i]<<setw(10);
for(j=0; j<k; j++)
{
cout<<y[i][j]<<setw(10);
}
cout<<"\n";
k--;
}
//Code of interpolation
double xn, h, p, sum=y[0][0], temp=1;
h=x[1]-x[0];
cout<<"Enter the values of x at which y to be
calculated\n";
cin>>xn;
p=(xn-x[0])/h;
for(j=1; j<n; j++)
{
temp=temp*(p-j+1)/j;
sum=sum+temp*y[0][j];
}
cout<<"The value of y at x="<<xn<<" is:
"<<sum<<endl;
return 0;
}
```

Lagrange Interpolation & Extrapolation:

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i, j, k, n;
cout<<"Enter the number of values to be
entered\n";
cin>>n;
double x[n], y[n];
cout<<"Enter the of values of x\n";
for(i=0; i<n; i++)
cin>>x[i];
cout<<"Enter the of values of y\n";
for(i=0; i<n; i++)
cin>>y[i];
//code of interpolation starts here
double xn, sum=0, temp;
cout<<"Enter the values of x at which y to be
calculated\n";
cin>>xn;
for(j=0; j<n; j++)
{
temp=1;
for(i=0; i<n; i++)
{
if(i==j)
continue;
```

```
else
temp=temp*(xn-x[i])/(x[j]-x[i]);
}
sum=sum+temp*y[j];
}
cout<<"The value of y at x="<<xn<<" is:
"<<sum<<endl;
return 0;
}
```

Newton's Divided Difference Interpolation & Extrapolation:

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i, j, k, n;
cout<<"Enter the number of values to be
entered\n";
cin>>n;
double x[n], y[n][n];
cout<<"Enter the of values of x\n";
for(i=0; i<n; i++)
cin>>x[i];
cout<<"Enter the of values of y\n";
for(i=0; i<n; i++)
cin>>y[i][0];
//Calculate difference table
for(j=1; j<n; j++)
{
for(i=0; i<n-j; i++)
{
y[i][j]=(y[i+1][j-1]-y[i][j-1])/(x[i+j]-x[i]);
}
}
//Print difference table
cout<<"\nThe Divided difference table is as
follows:\n\n";
cout<<"x"<<setw(10)<<"y"<<setw(10);
```

```
for(i=1; i<n; i++)
cout<<"d"<<i<<"y"<<setw(10);
cout<<"\
n----------------------------------------
----------\n";
k=n;
for(i=0; i<n; i++)
{
cout<<x[i]<<setw(10);
for(j=0; j<k; j++)
{
cout<<y[i][j]<<setw(10);
}
cout<<"\n";
k--;
}
//Code of interpolation
double xn, sum=y[0][0], temp=1.0;
cout<<"Enter the values of x at which y to be
calculated\n";
cin>>xn;
for(j=1; j<n; j++)
{
temp=temp*(xn-x[j-1]);
sum=sum+temp*y[0][j];
}
cout<<"The value of y at x="<<xn<<" is:
"<<sum<<endl;
return 0;
}
```

Gauss Elimination (3 variables):

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i, j, n, k;
n=3; //set number of equations
double a[n][n+1], x[n], temp, t;
cout<<"Enter equations in the form
ax+by+cz=d:"<<endl;
for (i = 0; i < n; i++)
{
for (j = 0; j <= n; j++)
{
cout<<"a["<<i<<","<<j<<"]: ";
cin>>a[i][j];
}
}
cout<<"\nThe matrix you have entered is \n";
for(i=0; i<n; i++)
{
for(j=0; j<=n; j++)
{
cout<<a[i][j]<<setw(10);
}
cout<<"\n";
}
//Do pivoting
for(k=0; k<n-1; k++)
```

```
{
for(i=k+1; i<n; i++)
{
if(fabs(a[k][k])<fabs(a[i][k]))
{
for(j=0; j<=n; j++)
{
temp=a[k][j];
a[k][j]= a[i][j];
a[i][j]=temp;
}
}
}
}
cout<<"\nThe matrix after pivoting is \n";
for(i=0; i<n; i++)
{
for(j=0; j<=n; j++)
{
cout<<a[i][j]<<setw(10);
}
cout<<"\n";
}
//Do the elementary row operation
for(k=0; k<n-1; k++)
{
for(i=k+1; i<n; i++)
{
t=a[i][k]/a[k][k];
for(j=0; j<=n; j++)
{
a[i][j]=a[i][j]-t*a[k][j];
}
}
}
```

```
cout<<"\nThe matrix after elementary row
operation is \n";
for(i=0; i<n; i++)
{
for(j=0; j<=n; j++)
{
cout<<a[i][j]<<setw(10);
}
cout<<"\n";
}
//Now Let's do the back substitution
for(i=n-1; i>=0; i--)
{
x[i]=a[i][n];
for(j=i+1; j<n; j++)
{
x[i]=x[i]-a[i][j]*x[j];
}
x[i]=x[i]/a[i][i];
}
cout<<"\nThe values of the variables are \n";
for (i = 0; i < n; i++)
{
cout << "\nx[" << i<<"]= "<<x[i];
}
return 0;
}
```

Jacobi Iteration (3 variables):

```
#include<iostream>
#include<cmath>
#include <iomanip>
using namespace std;
//Test convergency
bool convergency(double b[3][4])
{
int i, j;
bool converge=true;
for (i = 0; i < 3; i++)
{
int sum=0;
for (j = 0; j < 3; j++)
{
if(i==j)
continue;
sum=sum+fabs(b[i][j]);
}
if(sum>fabs(b[i][i]))
{
converge=false;
break;
}
}
return converge;
}
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i, j, k;
```

```
double a[3][4], b[3][4], x[3], x1[3], r[3],
ep, temp;
bool solvable=false;
cout<<"Enter equations in the form
ax+by+cz=d:"<<endl;
for (i = 0; i < 3; i++)
{
for (j = 0; j <= 3; j++)
{
cout<<"a["<<i<<","<<j<<"]: ";
cin>>a[i][j];
}
}
cout<<"\nThe matrix you have entered is \n";
for(i=0; i<3; i++)
{
for(j=0; j<=3; j++)
{
cout<<a[i][j]<<setw(10);
}
cout<<"\n";
}
//Rearrange the augmented matrix
for(k = 0; k < 3; k++)
{
for(i = 0; i < 3; i++)
{
for(j = 0; j <= 3; j++)
{
b[i][j]=a[((k+i)%3)][j];
}
}
if(convergency(b)==true)
{
solvable=true;
break;
```

```
}
for (j = 0; j <= 3; j++)
{
temp=b[1][j];
b[1][j]=b[2][j];
b[2][j]=temp;
}
if(convergency(b)==true)
{
solvable=true;
break;
}
}
//If any convergeable arrangement is found,
then try to solve
if(solvable==false)
{
cout<<"Given system of equations can't
converge to solution by this method."<<endl;
}
else
{
cout<<"\nThe matrix arrangement which will
converge:\n";
for(i=0; i<3; i++)
{
for(j=0; j<=3; j++)
{
cout<<b[i][j]<<setw(10);
}
cout<<"\n";
}
cout <<"\nEnter initial values of x\n";
for (i = 0; i < 3; i++)
{
cout << "x:[" << i<<"]=";
```

```
cin >> x[i];
}
cout << "\nEnter desired accuracy: ";
cin >> ep;
//calculate roots
do
{
for (i = 0; i < 3; i++)
{
x1[i]=x[i];
r[i] = (b[i][3] / b[i][i]);
for (j = 0; j < 3; j++)
{
if (j == i)
continue;
r[i] = r[i] - ((b[i][j] / b[i][i]) * x[j]);
}
}
for (i = 0; i < 3; i++)
{
x[i]=r[i];
}
}
while (fabs(x1[0]-x[0])>ep || fabs(x1[1]-
x[1])>ep || fabs(x1[2]-x[2])>ep);
for (i = 0; i < 3; i++)
{
cout << "\nx:[" << i+1<<"]= "<<x[i];
}
}
return 0;
}
```

Gauss Seidel (3 variables):

```
#include<iostream>
#include<cmath>
#include <iomanip>
using namespace std;
//Test convergency
bool convergency(double b[3][4])
{
int i, j;
bool converge=true;
for (i = 0; i < 3; i++)
{
int sum=0;
for (j = 0; j < 3; j++)
{
if(i==j)
continue;
sum=sum+fabs(b[i][j]);
}
if(sum>fabs(b[i][i]))
{
converge=false;
break;
}
}
return converge;
}
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i, j, k;
```

```
double a[3][4], b[3][4], x[3], x1[3], r[3],
ep, temp;
bool solvable;
cout<<"Enter equations in the form
ax+by+cz=d:"<<endl;
for (i = 0; i < 3; i++)
{
for (j = 0; j <= 3; j++)
{
cout<<"a["<<i<<","<<j<<"]: ";
cin>>a[i][j];
}
}
cout<<"\nThe matrix you have entered is \n";
for(i=0; i<3; i++)
{
for(j=0; j<=3; j++)
{
cout<<a[i][j]<<setw(10);
}
cout<<"\n";
}
//Rearrange the augmented matrix
for(k = 0; k < 3; k++)
{
for(i = 0; i < 3; i++)
{
for(j = 0; j <= 3; j++)
{
b[i][j]=a[((k+i)%3)][j];
}
}
if(convergency(b)==true)
{
solvable=true;
break;
```

```
}
for (j = 0; j <= 3; j++)
{
temp=b[1][j];
b[1][j]=b[2][j];
b[2][j]=temp;
}
if(convergency(b)==true)
{
solvable=true;
break;
}
}
//If any convergeable arrangement is found,
then try to solve
if(solvable==false)
{
cout<<"Given system of equations can't
converge to solution by this method."<<endl;
}
else
{
cout<<"\nThe matrix arrangement which will
converge: \n";
for(i=0; i<3; i++)
{
for(j=0; j<=3; j++)
{
cout<<b[i][j]<<setw(10);
}
cout<<"\n";
}
cout <<"\nEnter initial values of x\n";
for (i = 0; i < 3; i++)
{
cout << "x:[" << i<<"]=";
```

```
cin >> x[i];
}
cout << "\nEnter desired accuracy: ";
cin >> ep;
//calculate roots
do
{
for (i = 0; i < 3; i++)
{
x1[i]=x[i];
r[i] = (b[i][3] / b[i][i]);
for (j = 0; j < 3; j++)
{
if (j == i)
continue;
r[i] = r[i] - ((b[i][j] / b[i][i]) * x[j]);
}
x[i]=r[i];
}
}
while (fabs(x1[0]-x[0])>ep || fabs(x1[1]-
x[1])>ep || fabs(x1[2]-x[2])>ep);
for (i = 0; i < 3; i++)
{
cout << "\nx:[" << i<<"]= "<<x[i];
}
}
return 0;
}
```

Trapezoidal:

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
#define f(x) (1/(1+x))
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i, n;
double a, b, h, sum=0.0, integral;
cout<<"\nEnter the left limit of the
integration\n";
cin>>a;
cout<<"\nEnter the right limit of the
integration\n";
cin>>b;
cout<<"\nEnter the number of division\n";
cin>>n;
double x[n+1], y[n+1];
h=(b-a)/n;
for(i=0;i<=n;i++)
{
x[i]=a+i*h;
y[i]=f(x[i]);
}
for(i=1;i<=n-1;i++)
sum=sum+h*y[i];
integral=h/2.0*(y[0]+y[n])+sum;
cout<<"the value of the integral
is :"<<integral<<endl;
return 0;
```

```
}
```

Simpson's 1/3:

```
#include<iostream>
#include<cmath>
#include<iostream>
using namespace std;
#define f(x) (1/(1+x))
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i,n;
double a,b,h,sum=0,integral;
cout<<"Enter the left limit of the
integral :"<<endl;
cin>>a;
cout<<"Enter the right limit of the
integral :"<<endl;
cin>>b;
m:
cout<<"Enter the number of division (even
number):"<<endl;
cin>>n;
if(n%2==0)
{
double x[n+1],y[n+1];
h=(b-a)/n;
for(i=0; i<=n; i++)
{
x[i]=a+i*h;
y[i]=f(x[i]);
}
for(i=1; i<=n-1; i=i+2)
sum=sum+4*y[i];
```

```
for(i=2; i<=n-2; i=i+2)
sum=sum+2*y[i];
integral=h/3*(y[0]+y[n]+sum);
cout<<"the value of the integral
is :"<<integral<<endl;
}
else
{
goto m;
}
return 0;
}
```

Simpson's 3/8:

```
#include<iostream>
#include<cmath>
#include<iostream>
using namespace std;
#define f(x) (1/(1+x))
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i,n;
double a,b,h,sum=0,integral;
cout<<"Enter the left limit of the
integral :"<<endl;
cin>>a;
cout<<"Enter the right limit of the
integral :"<<endl;
cin>>b;
m:
cout<<"Enter the number of division (multiple
of 3):"<<endl;
cin>>n;
if(n%3==0)
{
double x[n+1],y[n+1];
h=(b-a)/n;
for(i=0; i<=n; i++)
{
x[i]=a+i*h;
y[i]=f(x[i]);
}
for(i=1; i<=n-1; i++)
{
```

```
if(i%3==0)
sum=sum+2*y[i];
else
sum=sum+3*y[i];
}
integral=3*h/8*(y[0]+y[n]+sum);
cout<<"the value of the integral
is :"<<integral<<endl;
}
else
{
goto m;
}
return 0;
}
```

Euler:

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;
#define df(x,y) (-y)
int main ()
{
cout.precision(6);
cout.setf(ios::fixed);
int i,n;
double x0,y0,h,xn;
cout<<"Enter the initial value of x\n";
cin>>x0;
cout<<"Enter the initial value of y\n";
cin>>y0;
cout<<"Enter the value of x up to which you
want to find the value of y\n";
cin>>xn;
cout<<"Enter the value of h\n";
cin>>h;
n=(xn-x0)/h;
double x[n+1],y[n+1];
x[0]=x0,y[0]=y0;
for(i=0; i<n; i++)
{
y[i+1]=y[i]+h*df(x[i],y[i]);
x[i+1]=x[i]+h;
}
for(i=0; i<=n;i++)
{
cout<<"y("<<x[i]<<"): "<<y[i]<<endl;
}
```

```
return 0;}
```

Modified Euler:

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;
#define df(x,y) (x+y)
int main ()
{
cout.precision(6);
cout.setf(ios::fixed);
int i,n;
double x0,y0,h,xn;
cout<<"Enter the initial value of x\n";
cin>>x0;
cout<<"Enter the initial value of y\n";
cin>>y0;
cout<<"Enter the value of x up to which you
want to find the value of y\n";
cin>>xn;
cout<<"Enter the value of h\n";
cin>>h;
n=(xn-x0)/h;
double x[n+1], y[n+1], y1[n+1];
x[0]=x0, y[0]=y0, y1[0]=y0;
for(i=0;i<n;i++)
{
y1[i+1]=y[i]+h*df(x[i],y[i]);
x[i+1]=x[i]+h;
y[i+1]=y[i]+h/2.0*(df(x[i],y[i])
+df(x[i+1],y1[i+1]));
}
for(i=0; i<=n; i++)
{
cout<<"y("<<x[i]<<"): "<<y[i]<<endl;
}
return 0;}
```

4th Order Runge Kutta (single equation):

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;
#define df(x,y) ((y*y-x*x)/(y*y+x*x))
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i,n;
double x0,y0,h,xn;
cout<<"Enter the initial value of x :\n";
cin>>x0;
cout<<"Enter the initial value of y :\n";
cin>>y0;
cout<<"Enter the value of x up to which you want to
find the value of y\n";
cin>>xn;
cout<<"Enter the value of h\n";
cin>>h;
n=(xn-x0)/h;
double x[n+1],y[n+1],k1,k2,k3,k4;
x[0]=x0, y[0]=y0;
for(i=0;i<n;i++)
{
k1=h*df(x[i],y[i]);
k2=h*df((x[i]+h/2.0),(y[i]+k1/2.0));
k3=h*df((x[i]+h/2.0),(y[i]+k2/2.0));
k4=h*df((x[i]+h),(y[i]+k3));
y[i+1]=y[i]+(1/6.0)*(k1+2*k2+2*k3+k4);
x[i+1]=x[i]+h;
}
for(i=0; i<=n; i++)
{
cout<<"y("<<x[i]<<"): "<<y[i]<<endl;
}
return 0;}
```

4th Order Runge Kutta (system of equations):

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;
#define dy_dx(x,y,z) (z)
#define dz_dx(x,y,z) (x*z*z-y*y)
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i,n;
double x0,y0,z0,h,xn;
cout<<"Enter the initial value of x:\n";
cin>>x0;
cout<<"Enter the initial value of y
corresponding to x:\n";
cin>>y0;
cout<<"Enter the initial value of z
corresponding to x:\n";
cin>>z0;
cout<<"Enter the value of x up to which you
want to find the value of y & z:\n";
cin>>xn;
cout<<"Enter the value of h\n";
cin>>h;
n=(xn-x0)/h;
double
x[n+1],y[n+1],z[n+1],k1,k2,k3,k4,l1,l2,l3,l4;
x[0]=x0, y[0]=y0, z[0]=z0;
for(i=0;i<n;i++)
```

```
{
k1=h*dy_dx(x[i],y[i],z[i]);
l1=h*dz_dx(x[i],y[i],z[i]);
k2=h*dy_dx((x[i]+h/2.0),(y[i]+k1/2.0),
(z[i]+l1/2.0));
l2=h*dz_dx((x[i]+h/2.0),(y[i]+k1/2.0),
(z[i]+l1/2.0));
k3=h*dy_dx((x[i]+h/2.0),(y[i]+k2/2.0),
(z[i]+l2/2.0));
l3=h*dz_dx((x[i]+h/2.0),(y[i]+k2/2.0),
(z[i]+l2/2.0));
k4=h*dy_dx((x[i]+h),(y[i]+k3),(z[i]+l3));
l4=h*dz_dx((x[i]+h),(y[i]+k3),(z[i]+l3));
y[i+1]=y[i]+(1/6.0)*(k1+2*k2+2*k3+k4);
z[i+1]=z[i]+(1/6.0)*(l1+2*l2+2*l3+l4);
x[i+1]=x[i]+h;
}
for(i=0; i<=n; i++)
{
cout<<"y("<<x[i]<<"): "<<y[i]<<endl;
cout<<"z("<<x[i]<<"): "<<z[i]<<endl;
}
return 0;
}
```

Parabolic Equation:

```
//Numerical solution to parabolic equation
du/dt = c^2 * d^2u/dx^2
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
#define u_x_0(x) (sin(3.1416*x))
//f(x)=(sin(3.1416*x)) assigned as initial
condition
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i,j,m,n, option;
double c,alpha,h,k,x,t,x_i,x_f,t_i,t_f,u_i,u_f;
cout<<"Numerical solution of heat equation
using finite difference formula"<<endl;
cout<<"Enter value of c"<<endl;
cin>>c;
cout<<"Enter initial x and corresponding u (1st
boundary condition)"<<endl;
cin>>x_i>>u_i; //1st boundary condition
u(x_i,t)=u_i
cout<<"Enter final x and corresponding u (2nd
boundary condition)"<<endl;
cin>>x_f>>u_f; //2nd boundary condition
u(x_f,t)=u_f
t_i=0; //initial simulation time assigned
cout<<"Enter final simulation time"<<endl;
cin>>t_f;
cout<<"Enter value of h and k"<<endl;
z:
cin>>h>>k;
alpha = (c*c*k)/(h*h); //calculated alpha
//check alpha limitation
if(alpha<=0 || alpha>0.5)
```

```
{
cout<<"please enter new value of h & k such
that 0 <= alpha <=0.5 where, alpha =
(c*c*k)/(h*h)"<<endl;
goto z;
}
//calculated division of x & t
m=(x_f-x_i)/h;
n=(t_f-t_i)/k;
//u defined as [(n+1) by (m+1)] dimensional
array
double u[n+1][m+1];
//calculated 1st & last column
for(i=0; i<=n; i++)
{
u[i][0]=u_i; //1st column
u[i][m]=u_f; //last column
}
//calculated 1st row
for( j=1; j<=m-1; j++)
{
x=x_i+j*h; //transforming j into x
u[0][j]= u_x_0(x);
}
//row wise calculated 2nd to last row
for(i=0; i<=n-1; i++)
{
for(j=1; j<=m-1; j++)
{
u[i+1][j]=alpha*(u[i][j-1]+u[i][j+1])+(1-
2*alpha)*u[i][j];
}
}
//printing value of u(x,t)
while(1)
{
cout<<"\nChoose option how you want
result:"<<endl;
```

```
cout<<"1. x, t both vary"<<endl;
cout<<"2. x vary, t constant"<<endl;
cout<<"3. x constant, t vary"<<endl;
cout<<"4. x, t both constant"<<endl;
cin>>option;
if(option==1)
{
for( i=0; i<=n; i++)
{
t=t_i+i*k; //transforming i into t
for( j=0; j<=m; j++)
{
x=x_i+j*h; //transforming j into x
cout<<"u("<<x<<","<<t<<")= "<<u[i][j]<<endl;
}
}
}
else if(option==2)
{
cout<<"Enter value of t where you want to find
the value of u:"<<endl;
cin>>t;
i=(t-t_i)/k; //transforming t into i
for( j=0; j<=m; j++)
{
x=x_i+j*h; //transforming j into x
cout<<"u("<<x<<","<<t<<")= "<<u[i][j]<<endl;
}
}
else if(option==3)
{
cout<<"Enter value of x where you want to find
the value of u:"<<endl;
cin>>x;
j=(x-x_i)/h; //transforming x into j
for( i=0; i<=n; i++)
{
```

```
t=t_i+i*k; //transforming i into t
cout<<"u("<<x<<","<<t<<")= "<<u[i][j]<<endl;
}
}
else if(option==4)
{
cout<<"Enter value of x where you want to find
the value of u:"<<endl;
cin>>x;
cout<<"Enter value of t where you want to find
the value of u:"<<endl;
cin>>t;
i=(t-t_i)/k; //transforming t into i
j=(x-x_i)/h; //transforming x into j
cout<<"u("<<x<<","<<t<<")= "<<u[i][j]<<endl;
}
}

return 0;
}
```

Hyperbolic Equation:

```
//Numerical solution to hyperbolic equation
d^2u/dt^2 = c^2 * d^2u/dx^2
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
#define u_x_0(x) (4*x-x*x) //f(x)=(4*x-x*x)
assigned as 1st initial condition
#define du_dt(x) (0) //g(x)=0 assigned as 2nd
initial condition
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i,j,m,n,option;
double c,alpha,h,k,x,t,x_i,x_f,t_i,t_f,u_i,u_f;
cout<<"Numerical solution of wave equation
using finite difference formula"<<endl;
cout<<"Enter value of c"<<endl;
cin>>c;
cout<<"Enter initial x and corresponding u (1st
boundary condition)"<<endl;
cin>>x_i>>u_i; //1st boundary condition
u(x_i,t)=u_i
cout<<"Enter final x and corresponding u (2nd
boundary condition)"<<endl;
cin>>x_f>>u_f; //2nd boundary condition
u(x_f,t)=u_f
t_i=0; //initial simulation time assigned
cout<<"Enter final simulation time"<<endl;
cin>>t_f;
cout<<"Enter value of h and k"<<endl;
z:
cin>>h>>k;
alpha = c*(k/h); //calculated alpha
//check alpha limitation
if(alpha>1)
```

```
{
cout<<"please enter new value of h & k such
that alpha<1 where, alpha=c*(k/h)"<<endl;
goto z;
}
//calculated division of x & t
m=(x_f-x_i)/h;
n=(t_f-t_i)/k;
```
//u defined as [(n+1) by (m+1)] dimensional array
```
ouble u[n+1][m+1];
//calculated 1st & last column
for(i=0; i<=n; i++)
{
u[i][0]=u_i; //1st column
u[i][m]=u_f; //last column
}
//calculated 1st row
for(j=1; j<=m-1; j++)
{
x=x_i+j*h; //transforming j into x
u[0][j]= u_x_0(x);
}
//calculated 2nd row
for(j=1; j<=m-1; j++)
{
x=x_i+j*h; //transforming j into x
u[1][j]=((pow(alpha,2))*(u[0][j+1]+u[0][j-
1]))/2.0+(1-pow(alpha,2))*u[0][j]+k*du_dt(x);
}
//row wise calculated 3rd to last row
for(i=1; i<=n-1; i++)
{
for(j=1; j<=m-1; j++)
{
x=x_i+j*h; //transforming j into x
u[i+1][j]=(pow(alpha,2))*(u[i][j-1]+u[i][j+1])
+2*(1-pow(alpha,2))*u[i][j]-u[i-1][j];
```

```
}
}
//printing value of u(x,t)
while(1)
{
cout<<"\nChoose option how you want
result:"<<endl;
cout<<"1. x, t both vary"<<endl;
cout<<"2. x vary, t constant"<<endl;
cout<<"3. x constant, t vary"<<endl;
cout<<"4. x, t both constant"<<endl;
cin>>option;
if(option==1)
{
for( i=0; i<=n; i++)
{
t=t_i+i*k; //transforming i into t
for( j=0; j<=m; j++)
{
x=x_i+j*h; //transforming j into x
cout<<"u("<<x<<","<<t<<")= "<<u[i][j]<<endl;
}
}
}
else if(option==2)
{
cout<<"Enter value of t where you want to find
the value of u:"<<endl;
cin>>t;
i=(t-t_i)/k; //transforming t into i
for( j=0; j<=m; j++)
{
x=x_i+j*h; //transforming j into x
cout<<"u("<<x<<","<<t<<")= "<<u[i][j]<<endl;
}
}
else if(option==3)
```

```
{
cout<<"Enter value of x where you want to find
the value of u:"<<endl;
cin>>x;
j=(x-x_i)/h; //transforming x into j
for( i=0; i<=n; i++)
{
t=t_i+i*k; //transforming i into t
cout<<"u("<<x<<","<<t<<")= "<<u[i][j]<<endl;
}
}
else if(option==4)
{
cout<<"Enter value of x where you want to find
the value of u:"<<endl;
cin>>x;
cout<<"Enter value of t where you want to find
the value of u:"<<endl;
cin>>t;
i=(t-t_i)/k; //transforming t into i
j=(x-x_i)/h; //transforming x into j
cout<<"u("<<x<<","<<t<<")= "<<u[i][j]<<endl;
}
}
return 0;
}
```

Curve Fitting (2nd degree polynomial):

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i,j,k,n,m;
cout<<"\nEnter the number of points to be
entered\n";
cin>>m;
double p[m], q[m], temp, t;
cout<<"\nEnter the x values\n";
for(i=0; i<m; i++)
{
cin>>p[i];
}
cout<<"\nEnter the y values\n";
for(i=0; i<m; i++)
{
cin>>q[i];
}
n=2; //Set degree of polynomial
double a[n+1][n+2],x[n+1];
for(i=0; i<=n; i++)
{
//Calculate different element of the
augmented matrix in a row
for(j=0; j<=n; j++)
{
a[i][j]=0;
```

```
for(k=0;k<m;k++)
{
a[i][j]=a[i][j]+pow(p[k],(i+j));
}
}
//Calculate last element of the augmented
matrix in a row
a[i][n+1]=0;
for(k=0;k<m;k++)
{
a[i][n+1]=a[i][n+1]+pow(p[k],i)*q[k];
}
}
n=n+1;
//Do pivoting
for(k=0; k<n-1; k++)
{
for(i=k+1; i<n; i++)
{
if(fabs(a[k][k])<fabs(a[i][k]))
{
for(j=0; j<=n; j++)
{
temp=a[k][j];
a[k][j]= a[i][j];
a[i][j]=temp;
}
}
}
}
//Do the elementary row operation
for(k=0; k<n-1; k++)
{
```

```
for(i=k+1; i<n; i++)
{
t=a[i][k]/a[k][k];
for(j=0; j<=n; j++)
{
a[i][j]=a[i][j]-t*a[k][j];
}
}
}
//Back substitution
for(i=n-1; i>=0; i--)
{
x[i]=a[i][n];
for(j=i+1; j<n; j++)
{
if(j!=i)
x[i]=x[i]-a[i][j]*x[j];
}
x[i]=x[i]/a[i][i];
}
cout<<"\nThe values of the coefficients
are\n";
for(i=0; i<n; i++)
{
cout<<x[i]<<endl;
}
cout<<"\nThe required polynomial is\n";
cout<<"y="<<x[0]<<"+"<<x[1]<<"x+"<<x[2]<<
"x^2"<<endl;
return 0;
}
```

Curve Fitting (3rd degree polynomial):

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
int main()
{
cout.precision(6);
cout.setf(ios::fixed);
int i,j,k,n,m;
cout<<"\nEnter the number of points to be
entered\n";
cin>>m;
double p[m], q[m], temp, t;
cout<<"\nEnter the x values\n";
for(i=0; i<m; i++)
{
cin>>p[i];
}
cout<<"\nEnter the y values\n";
for(i=0; i<m; i++)
{
cin>>q[i];
}
n=3; //Set degree of polynomial
double a[n+1][n+2],x[n+1];
for(i=0; i<=n; i++)
{
//Calculate different element of the
augmented matrix in a row
for(j=0; j<=n; j++)
{
a[i][j]=0;
```

```
for(k=0;k<m;k++)
{
a[i][j]=a[i][j]+pow(p[k],(i+j));
}
}
//Calculate last element of the augmented
matrix in a row
a[i][n+1]=0;
for(k=0;k<m;k++)
{
a[i][n+1]=a[i][n+1]+pow(p[k],i)*q[k];
}
}
n=n+1;
//Do pivoting
for(k=0; k<n-1; k++)
{
for(i=k+1; i<n; i++)
{
if(fabs(a[k][k])<fabs(a[i][k]))
{
for(j=0; j<=n; j++)
{
temp=a[k][j];
a[k][j]= a[i][j];
a[i][j]=temp;
}
}
}
}
//Do the elementary row operation
for(k=0; k<n-1; k++)
{
```

```
for(i=k+1; i<n; i++)
{
t=a[i][k]/a[k][k];
for(j=0; j<=n; j++)
{
a[i][j]=a[i][j]-t*a[k][j];
}
}
}
//Back substitution
for(i=n-1; i>=0; i--)
{
x[i]=a[i][n];
for(j=i+1; j<n; j++)
{
if(j!=i)
x[i]=x[i]-a[i][j]*x[j];
}
x[i]=x[i]/a[i][i];
}
cout<<"\nThe values of the coefficients
are\n";
for(i=0; i<n; i++)
{
cout<<x[i]<<endl;
}
cout<<"\nThe required polynomial is\n";
cout<<"y="<<x[0]<<"+"<<x[1]<<"x+"<<x[2]<<
"x^2+"<<x[3]<<"x^3"<<endl;
return 0;
}
```