1. Use a suitable numerical integration scheme to evaluate the following integral:

$$\int_0^{\pi/2} \frac{1}{1+sinx} dx$$

```
#include<iostream>
#include<cmath>
using namespace std;
double f(double x){
return 1/(1+sin(x));
}
//trapezium rule
double sine_trap(double a,double b,double n){
double x=0.0,sum=0.0,h=0.0;
int i;
if(n==1){
return 0.5*(b-a)*(f(a)+f(b));
}
else {
h=(b-a)/n;
sum+=f(a);
x=a+h;
for(i=1;i<n;i++){
sum+=2.0*f(x);
x+=h;
}
sum+=f(b);
return (0.5*h*sum);
}
}
// simpson's 1/3 rule
double sine_sim13(double a,double b,int n){
int i;
double I=0.0,J=0.0;
double h=(b-a)/n;
for(i=1;i<n;i+=2){     //representing the odd numbers of the intervals i.e. 1,3,5.....,n-1; since the number of interval is always even for Simpson 1/3 rule
I+=f(a+i*h);
}
for(i=2;i<n;i+=2){     //representing the even numbers of the intervals i.e. 2,4,6.....,n-2; since the number of interval is always even for Simpson 1/3 rule
J+=f(a+i*h);
}
double A=(h/3)*(f(a)+(4*I)+(2*J)+f(b));
return A;
}
double sine_sim38(double a,double b,int n){
double h,I=0.0,J=0.0,K=0.0,L;
h=(b-a)/n;
int i;
for(i=1;i<n;i+=3){     //representing k=1,4,7,.....,N-2; since the number of interval is always an integer multiple of 3 for Simpson 3/8 rule
I+=f(a+i*h);
```

```
}
for(i=2;i<n;i+=3){      //representing k=2,5,8,.....,N-1; since the number of interval is always an integer multiple of 3 for Simpson 3/8 rule
J+=f(a+i*h);
}
for(i=3;i<n;i+=3){      //representing k=3,6,9,.....,N-3; since the number of interval is always an integer multiple of 3 for Simpson 3/8 rule
K+=f(a+i*h);
}
L=f(a)+(3*I)+(3*J)+(2*K)+f(b);
return (3*h*L)/8;
}
int main(){
double a=0.0,b=M_PI/2;
int n=120;   //number of interval is an integer multiple of 2 and 3 for simpson 1/3 rule and simpson 3/8 rule respectively
cout<<"The result by trapezium rule is: "<<sine_trap(a,b,n)<<endl;
cout<<"The result by simpson's 1/3 rule is:
"<<sine_sim13(a,b,n)<<endl;
cout<<"The result by simpson's 3/8 rule is:
"<<sine_sim38(a,b,n)<<endl;
return 0;
}
```

2. Use a suitable numerical integration scheme to evaluate the following integral:

$$\int_{-1}^{1} e^{\frac{-1}{1+x^2}}\,dx$$

```
#include<iostream>
#include<cmath>
using namespace std;
double f(double x){
return exp(-1/(1+(x*x)));
}
//trapezium rule
double sine_trap(double a,double b,double n){
double x=0.0,sum=0.0,h=0.0;
int i;
if(n==1){
return 0.5*(b-a)*(f(a)+f(b));
}
else {
h=(b-a)/n;
sum+=f(a);
x=a+h;
for(i=1;i<n;i++){
sum+=2.0*f(x);
x+=h;
}
sum+=f(b);
return (0.5*h*sum);
```

```
}
}
// simpson's 1/3 rule
double sine_sim1_3(double a,double b,int n){
int i;
double I=0.0,J=0.0;
double h=(b-a)/n;
for(i=1;i<n;i+=2){      //representing the odd numbers of the intervals i.e. 1,3,5.....,n-1; since the number of interval is always even for Simpson 1/3 rule
I+=f(a+i*h);
}
for(i=2;i<n;i+=2){      //representing the even numbers of the intervals i.e. 2,4,6.....,n-2; since the number of interval is always even for Simpson 1/3 rule
J+=f(a+i*h);
}
double A=(h/3)*(f(a)+(4*I)+(2*J)+f(b));
return A;
}
double sine_sim3_8(double a,double b,int n){
double h,I=0.0,J=0.0,K=0.0,L;
h=(b-a)/n;
int i;
for(i=1;i<n;i+=3){      //representing k=1,4,7,.....,N-2; since the number of interval is always an integer multiple of 3 for Simpson 3/8 rule
I+=f(a+i*h);
}
for(i=2;i<n;i+=3){      //representing k=2,5,8,.....,N-1; since the number of interval is always an integer multiple of 3 for Simpson 3/8 rule
J+=f(a+i*h);
}
for(i=3;i<n;i+=3){      //representing k=3,6,9,.....,N-3; since the number of interval is always an integer multiple of 3 for Simpson 3/8 rule
K+=f(a+i*h);
}
L=f(a)+(3*I)+(3*J)+(2*K)+f(b);
return (3*h*L)/8;
}
int main(){
double a=-1,b=1;
int n=120;// multiple of both 2 and 3
cout<<"The result by trapezium rule is: "<<sine_trap(a,b,n)<<endl;
cout<<"The result by simpson's 1/3 rule is:
"<<sine_sim1_3(a,b,n)<<endl;
cout<<"The result by simpson's 3/8 rule is:
"<<sine_sim3_8(a,b,n)<<endl;
return 0;
}
```

3. The time dependent temperature at a point of a long bar is given by

$$T(t) = 100\left(1 - \frac{2}{\sqrt{\pi}}\int_0^{8/\sqrt{t}} e^{-\tau^2}\, d\tau\right)$$

(a) Write a C++ function double temp(double t) that evaluates the temperature at a time t using any suitable numerical integration technique.

(b) Write a complete C++ program to save data for $T(t)$ in the range $t\epsilon[10,200]$ with an increment of 0.01. Plot the data Tvs. t using gnuplot and save the figure as temp.png.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double f(double t,double tau){
double a=2/sqrt(M_PI);
double b=exp(-(tau*tau));
double c=1-(a*b);
return 100*c;
}
// integration using simpson 1/3 rule
double temp(double t){
int i, n=100;
double a=0, b=8/sqrt(t);
double I=0.0,J=0.0;
double h=(b-a)/n;
for(i=1;i<n;i+=2){
I+=f(t,a+i*h);
}
for(i=2;i<n;i+=2){
J+=f(t,a+i*h);
}
double A=(h/3)*(f(t,a)+(4*I)+(2*J)+f(t,b));
return A;
}
int main(){
ofstream fout("temp.dat");
for(double t=10;t<=200;t+=0.01)
fout<<t<<"   "<<temp(t)<<endl;
return 0;}
```

4. The spherical Bessel's functions have the integral representation

$$j_n(z) = \frac{z^n}{2^{n+1}n!}\int_0^{\pi} cos(zcos\theta)(sin\theta)^{2n+1}d\theta$$

(a) (i) Write a C++ function double bessel (int n, double z), that evaluates the spherical Bessel's functions$j_n(z)$, using the above definition, at some point z (Use any suitable integration technique).

(ii) Plot the first four Bessel's function using gnuplot. Save the plot as sp_bessel.png.

(b) The Fresnel integral of the second kind may be defined in terms of the spherical Bessel's function by

$$S(x) = \frac{1}{2}\int_0^x j_0(z)\sqrt{(z)}dz$$

(i) Define a C++ function double fresnel (double x) using the above definition and the function bessel (n,z) in part (a) that will calculate $S(x)$.

(ii) Plot $S(x)$ for$x \in$ [0, 20] with an increment of 0.1 and by using gnuplot. Save your plot as fresnel.png.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double k=2*M_PI;
double factorial (int n){
double fact=1;
for(int i=1;i<=n;i++)
fact*=i;
return fact;
}
double f(int n,double z,double theta){
double a=pow(z,n)/(pow(2,(n+1))*factorial(n));
double b=cos(z*cos(theta))*pow(sin(theta),(2*n)+1);
return a*b;
}
double bessel(int n, double z){
int i;
double a=0.0,b=M_PI;
int N=100;
double I=0.0,J=0.0;
double h=(b-a)/N;
for(i=1;i<N;i+=2){
I+=f(n,z,a+i*h);
}
for(i=2;i<N;i+=2){
J+=f(n,z,a+i*h);
}
double A=(h/3)*(f(n,z,a)+(4*I)+(2*J)+f(n,z,b));
return A;
}
double ff(double z)
```

```
{
return 0.5*bessel(0,z)*sqrt(z);
}
double Fresnel(double x){
int i;
double a=0.0,b=x;
int N=1000;
double I=0.0,J=0.0;
double h=(b-a)/N;
for(i=1;i<N;i+=2){
I+=ff(a+i*h);
}
for(i=2;i<N;i+=2){
J+=ff(a+i*h);
}
double A=(h/3)*(ff(a)+(4*I)+(2*J)+ff(b));
return A;
}

int main(){
ofstream fout("bessel.dat");
ofstream file("fresnel.dat");
double z;
for(z=0.0;z<=25.0;z+=0.01){
fout<<z<<"  "<<bessel(0,z)<<"  "<<bessel(1,z)<<"  "<<bessel(2,z)<<"
"<<bessel(3,z)<<endl;
}
for(double x=0.0;x<=20;x+=0.1){
file<<x<<"  "<<Fresnel(x)<<endl;
}
return 0;}
```

5. The Fresnel sine and cosine integrals are defined respectively as

$S(x) = \int_0^x sin\left(\frac{\pi u^2}{2}\right) du$ and $C(x) = \int_0^x cos\left(\frac{\pi u^2}{2}\right) du$

(a) (i) Define two C++ functions double Fr_sin(double x) and Fr_cos(double x) that will calculate the above given integrals by using any suitable numerical technique.

(ii) Write the values of the integrals in different columns by using the defined functions in (a) in a file for $x\epsilon[0,5]$. Plot the Fresnel integrals and save the plot as fresnel.png.

(b) Consider the following diffraction pattern

$$I = 0.5I_0\{[C(u_0) + 0.5]^2 + [S(u_0) + 0.5]^2\}$$

Here $I_0$is the incident intensity, I is the diffracted intensity and u_0 is proportional to the distance away from the knife edge.

(i) Write a C++ function double diffraction (double u0, double I0) that calls the functions Fr_sin() and Fr_cos and returns the diffracted intensity

(ii) Calculate I/I_0 for u_0 varying from −1.0 to +4.0 in steps of 0.1 and write them in a file. Plot your results of I/I_0 vs u_0 from the datafile using gnuplot and save the file. [Check your answer: at u0=1,2,3 and 4 the values of I/I0 are respectively 1.25923,0.843997,1.10763 and 0.922073]

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double f1(double u){
return cos((M_PI*u*u)/2.0);
}
double f2(double u){
return sin((M_PI*u*u)/2.0);
}
double Fr_cos(double x){
int i, n=100;
double a=0, b=x;
double I=0.0,J=0.0;
double h=(b-a)/n;
for(i=1;i<n;i+=2){
I+=f1(a+i*h);
}
for(i=2;i<n;i+=2){
J+=f1(a+i*h);
}
double A=(h/3)*(f1(a)+(4*I)+(2*J)+f1(b));
return A;
}
double Fr_sin(double x){
int i,n=100;
double a=0,b=x;
double I=0.0,J=0.0;
double h=(b-a)/n;
for(i=1;i<n;i+=2){
I+=f2(a+i*h);
}
for(i=2;i<n;i+=2){
J+=f2(a+i*h);
}
double A=(h/3)*(f2(a)+(4*I)+(2*J)+f2(b));
return A;
}
double diffraction(double u0,double I0){
return 0.5*I0*(pow((Fr_cos(u0)+0.5),2)+pow((Fr_sin(u0)+0.5),2));
}
```

```
int main(){
ofstream fout("Fresnel.dat");
ofstream file("I.dat");
for(double x=0.0;x<=4;x+=0.001)
fout<<x<<"   "<<Fr_cos(x)<<"  "<<Fr_sin(x)<<endl;
double u0, I0=5.0;// choose any value of I0 excluding zero
for(u0=-1.0;u0<=4.0;u0+=0.01)
file<<u0<<"   "<<diffraction(u0,I0)/I0<<endl;
return 0;}
```

6. In the scattering of neutrons (A = 1) with a nucleus of mass number A > 1, the average of the cosine of the scattering angle ψ (i.e. $u = \langle cos\psi \rangle$) is given by

$$u(A) = \langle cos\psi \rangle = \frac{1}{2}\int_0^{\pi} \frac{Acos\theta + 1}{(A^2 + 2Acos\theta + 1)^2} sin\theta d\theta$$

(a) Write a C++ function double scatt_angle (double A) that evaluates the average $u(A)$ for a mass number A using any suitable numerical integration technique.
(7)

(b) Save data from your program for u (A) in the range $A\epsilon[2,20]$. Plot the data using gnuplot and save the plot as angle.png.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double f(double A,double theta){
double a=(A*cos(theta))+1;
double b=a*sin(theta);
double c=A*A+2*A*cos(theta)+1;
return (0.5*b)/(c*c);
}
double scat_angle(double A){
int i, n=100;
double a=0, b=M_PI;
double I=0.0,J=0.0;
double h=(b-a)/n;
for(i=1;i<n;i+=2){
I+=f(A,a+i*h);
}
for(i=2;i<n;i+=2){
J+=f(A,a+i*h);
}
double B=(h/3)*(f(A,a)+(4*I)+(2*J)+f(A,b));
return B;
}
int main(){
ofstream fout("angle.dat");
for(double A=2.0;A<=20.0;A+=0.01)
fout<<A<<"   "<<scat_angle(A)<<endl;
```

```
return 0;}
```

7. An integral representation of the Gauss' error function is given by: 06_G6

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

(a) Define a function double errint(double x) that evaluates the error function for the argument x usning any suitable numerical scheme.

(b) Plot your function errint(x) for $x\epsilon[-2,2]$. with an increment of 0.01 and save the plot as errf.png.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double f(double t){
return (2/sqrt(M_PI))*exp(-t*t);
}
double errint(double x){
double a=0.0,b=x;
double y=0.0,sum=0.0,h=0.0;
int i;
int n=100;
if(n==1){
return 0.5*(b-a)*(f(a)+f(b));
}
else {
h=(b-a)/n;
sum+=f(a);
y=a+h;
for(i=1;i<n;i++){
sum+=2.0*f(y);
y+=h;
}
sum+=f(b);
return (0.5*h*sum);
}
}
int main(){
ofstream fout("error.dat");
for(double x=-2.0;x<=2.0;x+=0.1)
{cout<<x<<"      "<<errint(x)<<endl;
fout<<x<<"      "<<errint(x)<<endl;
}
return 0;}
```

8. The Gaussian curve is defined by

$$f(x) = \frac{1}{\sqrt{2\pi}} exp\left(\frac{-x^2}{2}\right)$$

(a) Write a C++ function double mygaussian(double a), that evaluates, using any suitable numerical integration scheme, the integral

$$I(a) = \frac{1}{\sqrt{2\pi}} \int_{-a}^{a} exp\left(\frac{-x^2}{2}\right) dx$$

for a given value of the parameter a as the argument of the function.

(b) Write a C++ program that writes the values of a and I(a) in a file for $a\epsilon[-2,2]$ using the above function mygaussian(). Plot I(a) vs. a for $a\epsilon[-2,2]$ and save the plot as gauss.png. From the plot, find the value of a at which $I(a) = \frac{1}{2}$.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double f(double x){
return sqrt(1.0/(2*M_PI))*exp(-(x*x)/2.0);}
// By trapizeum rule
double mygaussian_trap(double a){
double x=0.0,sum=0.0,h=0.0;
int i;
double n=100;
if(n==1){
return 0.5*(a-(-a))*(f(-a)+f(a));
}
else {
h=(a-(-a))/n;
sum+=f(-a);
x=-a+h;
for(i=1;i<n;i++){
sum+=2.0*f(x);
x+=h;
}
sum+=f(a);
return (0.5*h*sum);
}
}
//By Simpson's 1/3 rule
double mygaussian_simp(double a){
int i,n=100;
double I=0.0,J=0.0;
double h=(a-(-a))/n;
for(i=1;i<n;i+=2){
I+=f(-a+i*h);
}
for(i=2;i<n;i+=2){
```

```
J+=f(-a+i*h);
}
double A=(h/3)*(f(-a)+(4*I)+(2*J)+f(a));
return A;
}
int main(){
ofstream fout("gaussian.dat");
for(double a=-4.0;a<=4.0;a+=0.001){
fout<<a<<"   "<<abs(mygaussian_simp(a))<<"
"<<abs(mygaussian_trap(a))<<"    "<<f(a)<<endl;
}
return 0;
}
```

9. The period of a simple pendulum for large angle amplitude (θ_M) is given as

$$T = 4\left(\frac{L}{g}\right)^{1/2} \int_0^{\frac{\pi}{2}} \left(1 - sin^2\left(\frac{\theta_M}{2}\right) sin^2\phi\right)^{-1/2} d\phi$$

Where $0 \leq \theta_M < \pi$.

(a) Write a C or C++ function double pendulum_T(double thetaM), that evaluates the period T for a given θ_M as the argument using the above definition. Choose the value of L/g such that, as $\theta_M \to 0$, T = 1 s.

Call the function pendulum_T() from the main() function with different values of θ_M as input.

(b) Using the above function, plot T vs.$\theta_M$for $\theta_M \epsilon [0, \pi/2]$.

Hint: Check values: $\theta_M = [10^0, 50^0, 90^0] \Rightarrow T = [1.00193, 1.05033, 1.18258]$.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double f(double x,double theta_M){
double s=pow((sin(theta_M/2.0)),2.0);
double t=s*pow((sin(x)),2.0);
double u=1-t;
return 4.0*sqrt(0.0253)*(1.0/sqrt(u));//(L/g)=0.0253
}
//Pendulum time period by using trapizeum rule
double pendulum_T(double theta_M){
double x=0.0,sum=0.0,h=0.0;
double a=0,b=M_PI/2.0;
int i,n=100;
if(n==1){
return 0.5*(b-a)*(f(a,theta_M)+f(b,theta_M));
}
else {
h=(b-a)/n;
```

```
sum+=f(a,theta_M);
x=a+h;
for(i=1;i<n;i++){
sum+=2.0*f(x,theta_M);
x+=h;
}
sum+=f(b,theta_M);
double T=(0.5*h*sum);
return T;
}
}
/*
//Pendulum time period by using Simpson 1/3 rule
double pendulum_T(double theta_M){
double a=0,b=M_PI/2.0;
int i,n=100;
double I=0.0,J=0.0;
double h=(b-a)/n;
for(i=1;i<n;i+=2){
I+=f(a+i*h,theta_M);
}
for(i=2;i<n;i+=2){
J+=f(a+i*h,theta_M);
}
double T=(h/3)*(f(a,theta_M)+(4*I)+(2*J)+f(b,theta_M));
return T;
}
*/
int main(){
ofstream fout("pendulum.dat");
for(double theta_M=0.0;theta_M<=(M_PI/2.0);theta_M+=M_PI/10000.0){
cout<<theta_M<<"   "<<pendulum_T(theta_M)<<endl;
fout<<theta_M<<"   "<<pendulum_T(theta_M)<<endl;
}
return 0;
}
```

10. The Bessel's function has the integral representation

$$J_n(x) = \frac{2}{\pi^{\frac{1}{2}}\left(n-\frac{1}{2}\right)!}\left(\frac{x}{2}\right)^n \int_0^{\pi/2} cos(xsin\theta)cos^{2n}\theta\, d\theta \quad for \quad n > \frac{-1}{2}$$

(a) (i) Write a C++ function double bessel(int n, double x), that evaluates the Bessel's functions Jn(x), using the above definition, at some point x. [3]

(ii) Write a complete C++ program to plot the first four Bessel's functions in a single plot and save the plot as bessel.png

(b) The fraction of light incident normally on a circular aperture that is transmitted is given

$$T = 1 - \frac{1}{2kq}\int_0^{2kq} J_0(x)dx$$

where a is the radius of the aperture and $k = 2\pi/\lambda$ is the wavenumber.

(i) Using the function bessel(n,x), define a C++ function double transmitted (double q) that evaluates the fraction above.

(ii)Plot transmitted(q) for ($k = 2\pi$) vs. q for $a\epsilon[0,4]$. Save the plot as transmitted.png.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double k=2*M_PI;
double factorial (double n){
double fact=1;
for(int i=1;i<=n;i++)
fact*=i;
return fact;
}
double f(int n,double x,double theta){
double a=(2*pow((x/2.0),n))/(sqrt(M_PI)*factorial(0.5*(2*n-1)));
double b=cos(x*sin(theta))*pow(cos(theta),(2*n));
return a*b;
}
double bessel(int n, double x){
int i;
double a=0.0,b=M_PI/2.0;
int N=100;
double I=0.0,J=0.0;
double h=(b-a)/100.0;
for(i=1;i<N;i+=2){
I+=f(n,x,a+i*h);
}
for(i=2;i<N;i+=2){
J+=f(n,x,a+i*h);
}
double A=(h/3)*(f(n,x,a)+(4*I)+(2*J)+f(n,x,b));
return A;
}
double ff(double q, double x)
{
return 1-bessel(0,x)/(2*k*q);
}
double transmitted(double q){
int i;
double a=0.0,b=2*k*q;
int N=1000;
double I=0.0,J=0.0;
double h=(b-a)/N;
```

```
for(i=1;i<N;i+=2){
I+=ff(q,a+i*h);
}
for(i=2;i<N;i+=2){
J+=ff(q,a+i*h);
}
double A=(h/3)*(ff(q,a)+(4*I)+(2*J)+ff(q,b));
return A;
}
int main(){
ofstream fout("bessel.dat");
ofstream file("transmited.dat");
double x;
for(x=0.0;x<=25.0;x+=0.01){
fout<<x<<"  "<<bessel(0,x)<<"  "<<bessel(1,x)<<"  "<<bessel(2,x)<<"
"<<bessel(3,x)<<endl;
}
for(double q=0.0;q<=4;q+=0.01){
file<<q<<"  "<<transmitted(q)<<endl;
}
return 0;}
```

11. The spherical Bessel's functions have the following integral representation

$$j_n(x) = \frac{x^n}{2^{n+1}n!}\int_0^{\pi} \cos(xcos\theta)\,(sin\theta)^{2n+1}d\theta$$

(a) Write a C++ function **double Bessel (double x, int n)**, that evaluates the Bessel's functions $j_n(x)$, using the above definition, at some point x by using any suitable integration technique. Then plot the first four Bessel's function for $x\epsilon[0,25]$ using gnuplot. Save the plot as **bessel.png**.

(b) An analysis of the antenna radiation pattern for a system of circular aperture involves the function

$$g(u) = \int_0^1 f(r)j_0(ur)rdr$$

where $f(r) = 1 - r^2$
Write a C++ function **double ant_radiation (double u)** from the above definition and plot $g(u)$ Vs. u for $u\epsilon[0,25]$. Save the plot as **radiation.png**.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double k=2*M_PI;
double factorial (int n){
double fact=1;
for(int i=1;i<=n;i++)
```

```
fact*=i;
return fact;
}
double f(int n,double z,double theta){
double a=pow(z,n)/(pow(2,(n+1))*factorial(n));
double b=cos(z*cos(theta))*pow(sin(theta),(2*n)+1);
return a*b;
}
double bessel(int n, double z){
int i;
double a=0.0,b=M_PI;
int N=100;
double I=0.0,J=0.0;
double h=(b-a)/N;
for(i=1;i<N;i+=2){
I+=f(n,z,a+i*h);
}
for(i=2;i<N;i+=2){
J+=f(n,z,a+i*h);
}
double A=(h/3)*(f(n,z,a)+(4*I)+(2*J)+f(n,z,b));
return A;}
double f1(double r){
return 1-(r*r);
}
double ff(double r,double u){
return f1(r)*bessel(0,u*r)*r;
}
double ant_radiation(double u){
int i;
double a=0.0,b=1.0;
int N=100;
double I=0.0,J=0.0;
double h=(b-a)/N;
for(i=1;i<N;i+=2){
I+=ff(a+i*h,u);
}
for(i=2;i<N;i+=2){
J+=ff(a+i*h,u);
}
double A=(h/3)*(ff(a,u)+(4*I)+(2*J)+ff(b,u));
return A;
}
int main(){
ofstream fout("bessel.dat");
ofstream file("radiation.dat");
double z;
for(z=0.0;z<=25.0;z+=0.01){
fout<<z<<"  "<<bessel(0,z)<<"  "<<bessel(1,z)<<"  "<<bessel(2,z)<<"
"<<bessel(3,z)<<endl;
}
for(double u=0.0;u<=25;u+=0.1){
```

```
file<<u<<"  "<<ant_radiation(u)<<endl;
}
return 0;}
```

12. The complete elliptic integral of the first and second kind are defined respectively as

$$K(m) = \int_0^{\frac{\pi}{2}} (1 - msin^2\theta)^{-\frac{1}{2}} d\theta \qquad \text{and} \qquad E(m) = \int_0^{\frac{\pi}{2}} (1 - msin^2\theta)^{\frac{1}{2}} d\theta$$

(a) Define two functions **double K(double m)** and **double E(double m)** to find the values of $K(m)$ and $E(m)$ by using any suitable numerical integration technique.

4

(b) Call the functions **K()** and **E()** into the **main()** in order to plot the complete elliptic integrals in a single plot for $m\epsilon[0.0,1.0]$ with an increment of 0.01. Save the plot as **elliptic.png**.

3

(c) The period of a simple pendulum for large angle amplitude (θ_M)is given in terms of the complete elliptic integral of the first kind $K(m)$ as

$$T = 4\left(\frac{L}{g}\right)^{\frac{1}{2}} K\left(\sin^2\left(\frac{\theta_M}{2}\right)\right) \quad where\ 0 \le \theta_M < \pi$$

(i) Write a C++ function **double penduT(double thetaM)** that will call the function **K()** and evaluate the time period T for a given θ_M using the above definition. Choose L/g such that, as $\theta_M \to 0, T = 1\ s$.

2

(ii) Call the function **penduT()** into the **main()** in order to plot T Vs. θ_M for $\theta_M\epsilon[0,\pi/2]$. Take the increment of θ_M as $\frac{\pi}{30}$. Save the plot as **period.png**.

2

Check values: $\theta_M = [10°, 50°, 90°] => T = [1.00193, 1.05033, 1.18258]$.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double f1(double m, double theta){
double a=1-m*pow(sin(theta),2);
return 1/sqrt(a);
}
double f2(double m, double theta){
double a=1-m*pow(sin(theta),2);
return sqrt(a);
}
double K(double m){
int i;
double a=0.0, b=M_PI/2,n=1000;
double I=0.0,J=0.0;
double h=(b-a)/n;
```

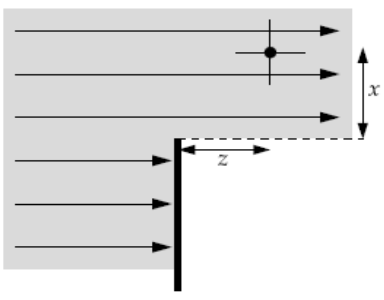
```
for(i=1;i<n;i+=2){
I+=f1(m,a+i*h);
}
for(i=2;i<n;i+=2){
J+=f1(m,a+i*h);
}
double A=(h/3)*(f1(m,a)+(4*I)+(2*J)+f1(m,b));
return A;
}
double E(double m){
int i;
double a=0.0, b=M_PI/2,n=1000;
double I=0.0,J=0.0;
double h=(b-a)/n;
for(i=1;i<n;i+=2){
I+=f2(m,a+i*h);
}
for(i=2;i<n;i+=2){
J+=f2(m,a+i*h);
}
double A=(h/3)*(f2(m,a)+(4*I)+(2*J)+f2(m,b));
return A;
}
double penduT(double thetaM){
return 4*sqrt(0.0253)*K(pow(sin(thetaM/2),2));          //(L/g)=0.0253


}
int main(){
ofstream fout("ellip.dat");
for(double m=0.0;m<=1.0;m+=0.01)
fout<<m<<"   "<<K(m)<<"   "<<E(m)<<endl;
ofstream file("time.dat");
for(double thetaM=0.0;thetaM<=M_PI/2;thetaM+=M_PI/30)
file<<thetaM<<"   "<<penduT(thetaM)<<endl;
return 0;}
```

13. Suppose a plane wave of wavelength λ such as light or a sound wave is blocked by an object with a straight edge, represented by the solid line at the bottom of this figure:
The wave will be diffracted at the edge and the resulting intensity at the position (x, z) marked by the dot is given by near-field diffraction theory to be

$$I = \frac{I_0}{8}([2C(u)+1]^2 + [2S(u)+1]^2)$$

Where I_0 is the intensity of the wave before diffraction and $u = x\sqrt{\frac{2}{\lambda z}}$ and the Fresnel integrals are defined as $C(u) = \int_0^u \cos(\frac{\pi t^2}{2})dt$ and $S(u) = \int_0^u \sin(\frac{\pi t^2}{2})dt$



(a) Define two functions **Fr_cos(double u)** and **double Fr_sin(double u)** to evaluate the above integrals by using any suitable numerical technique.

(b) Define another function **double diffraction(double x)** that will calculate I/I_0 by using the above definition and for a given value of x (as shown in the figure).

(c) Write a complete C++ program to calculate I/I_0 and make a plot of it as a function of x in the range $-5m$ to $5m$ for the case of a sound wave with wavelength $\lambda = 1m$, measured $z = 3m$ past the straight edge. Save the plot as **intensity.png**.

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double f1(double t){
return cos((M_PI*t*t)/2.0);
}
double f2(double t){
return sin((M_PI*t*t)/2.0);
}
double Fr_cos(double x){
int i, n=100;
double a=0, b=x;
double I=0.0,J=0.0;
double h=(b-a)/n;
for(i=1;i<n;i+=2){
I+=f1(a+i*h);
}
for(i=2;i<n;i+=2){
J+=f1(a+i*h);
}
double A=(h/3)*(f1(a)+(4*I)+(2*J)+f1(b));
return A;
}
double Fr_sin(double x){
int i,n=100;
double a=0,b=x;
double I=0.0,J=0.0;
double h=(b-a)/n;
for(i=1;i<n;i+=2){
I+=f2(a+i*h);
}
for(i=2;i<n;i+=2){
J+=f2(a+i*h);
}
double A=(h/3)*(f2(a)+(4*I)+(2*J)+f2(b));
return A;
}
double u(double x){
double lambda=1, z=3;
return x*sqrt(2/(lambda*z));
}
double diffraction(double x){
```

```
return (pow((2*Fr_cos(u(x)))+1,2)+pow((2*Fr_sin(u(x)))+1,2))/8;
}
int main(){
ofstream fout("Fresnel.dat");
ofstream file("si.dat");
for(double x=-6.0;x<=6;x+=0.01)
fout<<x<<"   "<<Fr_cos(x)<<"  "<<Fr_sin(x)<<endl;
double x;
for(x=-5.0;x<=5.0;x+=0.01)
file<<x<<"   "<<diffraction(x)<<endl;
return 0;}
```

14.

```
//A C++ Program To evaluate a Definite Integral by Gauss Quadrature
Formula
#include<iostream>
#include<cmath>
#include<iomanip>

using namespace std;

//The given Function of Integration
double f(double x)
{
    return x*x*x;
}

//Legendre's Polynomial Pn(x)
double Pn(double x, int n){
if(n==0) return 1;
else if (n==1) return x;
else return (((2*n-1)*x*Pn(x,n-1))-((n-1)*Pn(x,n-2)))/n;
}

//Derivative of Legendre's Polynomial Pn(x) i.e. d/dx (Pn(x))
double d_Pn(double x,int n){
return (n/((x*x)-1))*(x*Pn(x,n)-Pn(x,n-1));
}

//The Gaussian Quadrature rule
void gauss_qua(double a,double b,int n){
//First find the solution of the n-th Legendre polynomial  i.e. the n sample points xk (k = 1,2,...,n)
between the interval [-1,1]
if (n<=0)
cout<<"The number of sample points can not be negative or zero";
double x0[n],x[n],w[n],x_prime[n],w_prime[n],c,h;
int i;
for(i=0;i<n;i++){
x0[i]=cos((M_PI*((i+1)-0.25))/(n+0.5));
```

/*initial guess of the k-th root of the n-th Legendre polynomial by Tricomi's approximation $x(n,k) = cos(\pi * (k-0.25)/(n+0.5))$; where the values of k starts from 1 i.e. k=1,2,..,n. Thus to keep the correct values of k , which is represented by i in the loop , we add 1 to i, otherwise the values of k will be counted as k=0,1,....,n-1 which is not correct at all, but the array representation of the root needs to start i from 0;*/

```
//Imprived root by Newton-Raphson method
c=x0[i];
h=Pn(c,n)/d_Pn(c,n);
while (abs(h)>=0.00000000001){
c=c-h;
h=Pn(c,n)/d_Pn(c,n);
}
cout<<c<<endl;
x[i]=c;    // We store the root in an array
// Once the root is i.e. the sample point is found we calculate the weight
w[i]= (2/((1-c*c)*(d_Pn(c,n)*d_Pn(c,n))));
}
// Once the sample points (roots) and weights are calculated between [-1,1] (from the Legendre polynomial) we need to rescale the values for our given interval [a,b]
for(i=0;i<n;i++){
x_prime[i]=0.5*(((b-a)*x[i])+(b+a));
w_prime[i]=0.5*(b-a)*w[i];
}
//Now everything is ready and we have to perform the last step of the algorith i.e. use Gaussian Quadrature method defined in the equation
double result=0.0;
for(i=0;i<n;i++){
result+=(w_prime[i]*f(x_prime[i]));
}
// Now the desired result is shown below
cout<<"The value of integration is: "<<result<<endl;
/* If we use return type function then simply replace the cout line by return result
 return result;*/
}

int main(){
double a,b;
int n;
cout<<"Enter the lower limit of integration: ";
cin>>a;
cout<<"Enter the upper limit of integration: ";
cin>>b;
cout<<"Enter the order n of the legendre polynomial: ";
cin>>n;
gauss_qua(a,b,n);
/* When the gauss_qua(a.b.n) is return type write the following
cout<<"The value of integration is: "<<gauss_qua(a,b,n)<<endl;*/
return 0;}
```

15. Period of an anharmonic oscillator is given by

$$T = \sqrt{8m}\int_0^A \frac{dx}{\sqrt{V(A)-V(x)}}$$

where A is the amplitude of oscillation.

(a) Suppose the potential is $V(x) = x^4$ and the mass of the particle is $m = 1\,kg$. Write a C++ function **double tperiod(double A)** that evaluates the time period of the oscillator for the given amplitude **A** using Gaussian quadrature with 20 points.

[The required relations for Gaussian quadrature method are as follows

$$nP_n(x) = (2n-1)xP_{n-1}(x) - (n-1)P_{n-2}(x)$$

$$P_n'(x) = \frac{n}{x^2-1}\{xP_n(x) - P_{n-1}(x)\}$$

Where $P_n(x)$ the Legendre polynomial of degree n and $P_n'(x)$ is the derivative of $P_n(x)$]

(b) Use the function to make a graph of the period for the amplitudes ranging from $A = 0\ to\ A = 2$. You should find that the oscillator gets faster as the amplitude increases.

```
#include<iostream>
#include<cmath>
#include<iomanip>
#include<fstream>
using namespace std;
double V(double x){
return pow(x,4);
}
//The given Function of Integration
double f(double x, double A)
{double m=1.0;
double a=sqrt(8*m);
double b=sqrt(V(A)-V(x));
    return a/b;
}

//Legendre's Polynomial Pn(x)
double Pn(double x, int n){
if(n==0) return 1;
else if (n==1) return x;
else return (((2*n-1)*x*Pn(x,n-1))-((n-1)*Pn(x,n-2)))/n;
}

//Derivative of Legendre's Polynomial Pn(x)i.e.d/dx(Pn(x))
double d_Pn(double x,int n){
return (n/((x*x)-1))*(x*Pn(x,n)-Pn(x,n-1));
}

//The Gaussian Quadrature rule
double tperiod(double A){
```

```
//First find the solution of the n-th legendre polynomial i.e. the n sample points x_k (k=1,2,...,n) between the interval [-1,1]
double a=0,b=A;
int n=20;
double x0[n],x[n],w[n],x_prime[n],w_prime[n],c,h;
int i;
for(i=0;i<n;i++){
x0[i]=cos((M_PI*((i+1)-0.25))/(n+0.5));
/*initial guess of the k-th root of the n-th legendre polynomial by Tricomi's approximation x(n,k)=cos(PI*(k-0.25)/(n+0.5)); where the values of k starts from 1 i.e. k=1,2,..,n. Thus to keep the correct values of k , which is represented by i in the loop , we add 1 to i, otherwise the values of k will be counted as k=0,1,....,n-1 which is not correct at all, but the array representation of the root needs to start i from 0;*/
//Imprived root by Newton-Raphson method
c=x0[i];
h=Pn(c,n)/d_Pn(c,n);
while (abs(h)>=0.00000000001){
c=c-h;
h=Pn(c,n)/d_Pn(c,n);
}
x[i]=c;    // We store the root in an array
// Once the root is i.e. the sample point is found we calculate the weight
w[i]= (2/((1-c*c)*(d_Pn(c,n)*d_Pn(c,n))));
}
// Once the sample points (roots) and weights are calculated between [-1,1] (from the Legendre polynomial) we need to rescale the values for our given interval [a,b]
for(i=0;i<n;i++){
x_prime[i]=0.5*(((b-a)*x[i])+(b+a));
w_prime[i]=0.5*(b-a)*w[i];
}
//Now everything is ready and we have to perform the last step of the algorith i.e. use Gaussian Quadrature method defined in the equation
double result=0.0;
for(i=0;i<n;i++){
result+=(w_prime[i]*f(x_prime[i],A));
}
// Now the desired result is shown below
/*For void type function
cout<<"The value of integration is: "<<result<<endl;*/
//If we use return type function then simply replace the cout line by return result
 return result;
}

int main(){
ofstream fout("t.dat");
double A;
for(A=0;A<=2;A+=0.01)
fout<<A<<"     "<<tperiod(A)<<endl;
return 0;}
```

16. Debye's theory of solids gives the heat capacity of a solid at temperature T to be

$$C_v = 9V\rho K_B \left(\frac{T}{\theta_D}\right)^3 \int_0^{\theta_D/T} \frac{x^4 e^x}{(e^x-1)^2}\,dx$$

where V is the volume of the solid, ρ is the number density of atoms, k_B is Boltzmann's constant, and θ_D is the so-called Debye temperature, a property of solids that depends on their density and speed of sound.

a) Write a C++ function **double sp_heat(double T)** that calculates C_V for a given value of the temperature, for a sample consisting of 1000 cubic centimeters of solid aluminum, which has a number density of $\rho = 6.022 \times 10^{28}\ m^{-3}$ and a Debye temperature of $\theta_D = 428K$. Use Gaussian quadrature to evaluate the integral, with 50 sample points.

b) Use your function to make a graph of the heat capacity as a function of temperature from $T = 5K$ *to* $T = 500K$. Save the graph as **spheat.png**.

```
#include<iostream>
#include<cmath>
#include<fstream>
#define V pow(10,-3)
#define rho 6.022*pow(10,28)
#define KB 1.38*pow(10,-23)
#define theta_D 428
using namespace std;
double debye(double x,double T){
double a=exp(x);
double b=a*pow(x,4);
double c=pow((a-1),2);
double d=b/c;
double e=pow((T/theta_D),3.0);
double f=9.0*V*rho*KB;
return (f*e*d);
}
//Legendre's Polynomial Pn(x)
double Pn(double x, int n){
if(n==0) return 1;
else if (n==1) return x;
else return (((2*n-1)*x*Pn(x,n-1))-((n-1)*Pn(x,n-2)))/n;
}

//Derivative of Legendre's Polynomial Pn(x)i.e.d/dx (Pn(x))
double d_Pn(double x,int n){
return (n/((x*x)-1))*(x*Pn(x,n)-Pn(x,n-1));
}

//Calculation of specific heat by using the Gaussian Quadrature rule
double sp_heat(double T){
double a=0.0;          //The lower limit of integration
double b=theta_D/T;    //The upper limit of integration
```

```
int n=6;            //The order n of the Legendre polynomial
//First find the solution of the n-th legendre polynomila i.e. the n sample points x_k (k=1,2,...,n) between the interval [-1,1]
if (n<=0)
cout<<"The number of sample points can not be negative or zero";
double x0[n],x[n],w[n],x_prime[n],w_prime[n],c,h;
int i;
for(i=0;i<n;i++){
x0[i]=cos((M_PI*((i+1)-0.25))/(n+0.5));

//Imprived root by Newton-Raphson method
c=x0[i];
h=Pn(c,n)/d_Pn(c,n);
while (abs(h)>=0.00000000001){
c=c-h;
h=Pn(c,n)/d_Pn(c,n);
}
x[i]=c;

w[i]= (2/((1-c*c)*(d_Pn(c,n)*d_Pn(c,n))));
}

for(i=0;i<n;i++){
x_prime[i]=0.5*(((b-a)*x[i])+(b+a));
w_prime[i]=0.5*(b-a)*w[i];
}

double result=0.0;
for(i=0;i<n;i++){
result+=(w_prime[i]*debye(x_prime[i],T));
}
return result;
}

int main(){

ofstream fout("debye.dat");
for(double T= 5.0;T<=500;T+=0.005){
cout<<T<<"\t\t"<<sp_heat(T)<<endl;
fout<<T<<"\t\t"<<sp_heat(T)<<endl;
}
return 0;
}
```

17. Integrate $f(x) = e^{-x^2}$ over the range from zero to infinity

//let $z = x/(1+x)$ i.e. $x = z/(1-z)$; then this change of variable gives the same result but in the range 0 to 1 ;; see lecture notes

```
#include<iostream>
```

```
#include<cmath>
#include<fstream>
using namespace std;
double f(double z){
double a=(z*z)/((1-z)*(1-z));
double b=exp(-a)/((1-z)*(1-z));
return b;}
//Legendre's Polynomial P_n(x)
double Pn(double x, int n){
if(n==0) return 1;
else if (n==1) return x;
else return (((2*n-1)*x*Pn(x,n-1))-((n-1)*Pn(x,n-2)))/n;
}

//Derivative of Legendre's Polynomial P_n(x) i.e. d/dx (P_n(x))
double d_Pn(double x,int n){
return (n/((x*x)-1))*(x*Pn(x,n)-Pn(x,n-1));
}

//The Gaussian Quadrature rule
void gauss_qua(double a,double b,int n){
if (n<=0)
cout<<"The number of sample points can not be negative or zero";
double x0[n],x[n],w[n],x_prime[n],w_prime[n],c,h;
int i;
for(i=0;i<n;i++){
x0[i]=cos((M_PI*((i+1)-0.25))/(n+0.5));

//Imprived root by Newton-Raphson method
c=x0[i];
h=Pn(c,n)/d_Pn(c,n);
while (abs(h)>=0.00000000001){
c=c-h;
h=Pn(c,n)/d_Pn(c,n);
}
x[i]=c;  w[i]= (2/((1-c*c)*(d_Pn(c,n)*d_Pn(c,n))));
}
for(i=0;i<n;i++){
x_prime[i]=0.5*(((b-a)*x[i])+(b+a));
w_prime[i]=0.5*(b-a)*w[i];
}

double result=0.0;
for(i=0;i<n;i++){
result+=(w_prime[i]*f(x_prime[i]));
}
// Now the desired result is shown below
cout<<"The value of integration is: "<<result<<endl;
// If we use return type function then simply replace the cout line by return result
// return result;
}
```

```
int main(){
double a,b;
int n;
cout<<"Enter the lower limit of integration: ";
cin>>a;
cout<<"Enter the upper limit of integration: ";
cin>>b;
cout<<"Enter the order n of the legendre polynomial: ";
cin>>n;
gauss_qua(a,b,n);
// When the gauss_qua(a.b.n) is return type write the following
// cout<<"The value of integration is: "<<gauss_qua(a,b,n)<<endl;
return 0;}
```

// the answer is sqrt(M_PI)/2=0.886226925453. . .

18. The eigenfunctions of the one-dimensional simple harmonic oscillator are

$$u_n(x) = \frac{1}{\sqrt{2^n n!\sqrt{\pi}}} H_n(x)\, e^{-x^2/2}$$

where $H_n(x)$ is the Hermite polynomial of order n and we have used a system of units in which $\sqrt{\hbar/m\omega} = 1$.

(a) Define a function double hermite (double x, int N) that will evaluates the Hermite polynomial of order n at a point x by using the following recursion relation

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x)$$

(b) define a function double f (double x, int N) that will evaluates the eigenfunctions by using the above given definition. Plot the eigenfunctionds for n= 0,1,2,3 and for $x\epsilon[-5,5]$ with an increment of 0.001.

(C) Evaluate $\int_{-\infty}^{+\infty} u_n(x)u_m(x)dx$ for $n, m\epsilon[0,5]$. Can you interpret the result?

```
#include<iostream>
#include<cmath>
#include<fstream>
using namespace std;
double hermite(double x, int N){
if(N==0) return 1;
else if (N==1) return 2*x;
else return (2*x*hermite(x,N-1)-2*(N-1)*hermite(x,N-2));
}
int fact(int N){
if (N>1)
return N*fact(N-1);
```

```
else return 1;
}
double f(double x,int N){
double a=1/(sqrt(pow(2,N)*fact(N)*sqrt(M_PI)));
return a*hermite(x,N)*exp(-(x*x)/2);
}

double eigen(double z,int N){          //The change of variable is done to change the limit of integratio from [−∞,+∞] to [-1,+1]
return (f((z/(1-z*z)),N)*f((z/(1-z*z)),N)*(1+z*z))/((1-(z*z))*(1-(z*z)));
 }
//when we use both m and n in the calculation of the integration i.e. u_n(x) * u_m(x)
double d_eigen(double z,int N,int M){
return (f((z/(1-z*z)),N)*f((z/(1-z*z)),M)*(1+z*z))/((1-(z*z))*(1-(z*z)));
}

//Start of Gaussian_Quadrature process to solve the integration
//Legendre's Polynomial P_n(x)
double Pn(double x, int n){
if(n==0) return 1;
else if (n==1) return x;
else return (((2*n-1)*x*Pn(x,n-1))-((n-1)*Pn(x,n-2)))/n;
}

//Derivative of Legendre's Polynomial P_n(x) i.e. d/dx (P_n(x))
double d_Pn(double x,int n){
return (n/((x*x)-1))*(x*Pn(x,n)-Pn(x,n-1));
}

//The Gaussian Quadrature rule
double gauss_qua(int N,int M){    //The order of Hermite polynomial and the eigenfunction is represented by N which is given by small n in question.
//The degree to calculate Legendre polynomial and Guassian quadrature fomula is represented by n
double a=-1,b=1;          //limit of integration
int n=30;                 //order of Legendre polynomial i.e. the number of sample points for Gaussian quadrature rule

if (n<=0)
cout<<"The number of sample points can not be negative or zero";
double x0[n],x[n],w[n],x_prime[n],w_prime[n],c,h;
int i;
for(i=0;i<n;i++){
x0[i]=cos((M_PI*((i+1)-0.25))/(n+0.5));
//Imprived root by Newton-Raphson method
c=x0[i];
h=Pn(c,n)/d_Pn(c,n);
while (abs(h)>=0.00000000001){
c=c-h;
```

```
h=Pn(c,n)/d_Pn(c,n);
}
x[i]=c;  w[i]= (2/((1-c*c)*(d_Pn(c,n)*d_Pn(c,n))));
}
for(i=0;i<n;i++){
x_prime[i]=0.5*(((b-a)*x[i])+(b+a));
w_prime[i]=0.5*(b-a)*w[i];
}
double result=0.0;
for(i=0;i<n;i++){
//result+=(w_prime[i]*eigen(x_prime[i],N));
result+=(w_prime[i]*d_eigen(x_prime[i],N,M));//when we use both m and n in the calculation of the integration i.e. u_n(x) * u_m(x) then add the term int M
}
// Now the desired result is shown below
//cout<<"The value of integration is: "<<result<<endl;
// If we use return type function then simply replace the cout line by return result
 return result;
}
int main(){
ofstream fout("eigen.dat");
for(double x=-5;x<=5;x+=0.001)
{
fout<<x<<"   "<<f(x,0)<<"   "<<f(x,1)<<"   "<<f(x,2)<<"
"<<f(x,3)<<endl;}

int N,M    ;
for(N=0;N<=4;N++){
for(M=0;M<=4;M++){
cout<<N<<"  "<<M<<"  "<<gauss_qua(N,M)<<endl;
}
cout<<endl;
}

return 0;}
```