Prosjekt 3 - kode

# 3a Legendre

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <fstream>

#include <iomanip>

#include <string>

#include <armadillo>

#include "time.h"

#include "lib.h"

using namespace std;

using namespace arma;

double **int\_function**(double x1, double y1, double z1, double x2, double y2, double z2);

void **gauleg**(double x1, double x2, double x[], double w[], int N);

int **main**(int argc, char\* argv[])

{

int N = atoi(argv[1]);

double \*x = new double [N];

double \*w = new double [N];

double a = -3;

double b = 3;

double pi = 3.1415926535;

double sixteen = 16\*16;

double nevner = 1/sixteen;

double answer = 5\*pi\*pi\*nevner;

gauleg(a,b,x,w, N);

double int\_gauss = 0.;

// six-double loops

clock\_t start, finish;

start = clock();

for (int i=0;i<N;i++){

for (int j = 0;j<N;j++){

for (int k = 0;k<N;k++){

for (int l = 0;l<N;l++){

for (int m = 0;m<N;m++){

for (int n = 0;n<N;n++){

int\_gauss+=w[i]\*w[j]\*w[k]\*w[l]\*w[m]\*w[n]\*int\_function(x[i],x[j],x[k],x[l],x[m],x[n]);

}}}}}

}

finish = clock();

double timeused = (double) (finish - start)/(CLOCKS\_PER\_SEC );

cout << "Running program with N value = " << N << endl;

cout << setiosflags(ios::showpoint | ios::uppercase);

cout << setprecision(10) << "Time used = " << timeused << " seconds." << endl;

cout << setprecision(3) << "The analytical solution is " << int\_gauss << endl;

cout << setprecision(3) << "The closed form answer is " << answer << endl;

}

double **int\_function**(double x1, double y1, double z1, double x2, double y2, double z2){

// this function defines the function to integrate

double alpha = 2.;

// evaluate the different terms of the exponential

double exp1=-2\*alpha\*sqrt(x1\*x1+y1\*y1+z1\*z1);

double exp2=-2\*alpha\*sqrt(x2\*x2+y2\*y2+z2\*z2);

double deno=sqrt(pow((x1-x2),2)+pow((y1-y2),2)+pow((z1-z2),2));

if (deno < 1e-4){

return 0;

}

else{

return exp(exp1+exp2)/deno;

}

} // end of function to evaluate

void **gauleg**(double x1, double x2, double x[], double w[], int N)

{

int m,j,i;

double z1,z,xm,xl,pp,p3,p2,p1;

double const pi = 3.14159265359;

double \*x\_low, \*x\_high, \*w\_low, \*w\_high;

m = (N + 1)/2; // roots are symmetric in the interval

xm = 0.5 \* (x2 + x1);

xl = 0.5 \* (x2 - x1);

x\_low = x; // pointer initialization

x\_high = x + N - 1;

w\_low = w;

w\_high = w + N - 1;

for(i = 1; i <= m; i++) { // loops over desired roots

z = cos(pi \* (i - 0.25)/(N + 0.5));

/\*

\*\* Starting with the above approximation to the ith root

\*\* we enter the mani loop of refinement bt Newtons method.

\*/

do {

p1 =1.0;

p2 =0.0;

/\*

\*\* loop up recurrence relation to get the

\*\* Legendre polynomial evaluated at x

\*/

for(j = 1; j <= N; j++) {

p3 = p2;

p2 = p1;

p1 = ((2.0 \* j - 1.0) \* z \* p2 - (j - 1.0) \* p3)/j;

}

/\*

\*\* p1 is now the desired Legrendre polynomial. Next compute

\*\* ppp its derivative by standard relation involving also p2,

\*\* polynomial of one lower order.

\*/

pp = N \* (z \* p1 - p2)/(z \* z - 1.0);

z1 = z;

z = z1 - p1/pp; // Newton's method

} while(fabs(z - z1) > ZERO);

/\*

\*\* Scale the root to the desired interval and put in its symmetric

\*\* counterpart. Compute the weight and its symmetric counterpart

\*/

\*(x\_low++) = xm - xl \* z;

\*(x\_high--) = xm + xl \* z;

\*w\_low = 2.0 \* xl/((1.0 - z \* z) \* pp \* pp);

\*(w\_high--) = \*(w\_low++);

}

} // End\_ function gauleg()

# 3b – Laguerre

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <fstream>

#include <iomanip>

#include <string>

#include <armadillo>

#include "time.h"

#include "lib.h"

#include <random>

using namespace std;

using namespace arma;

void **gauss\_laguerre**(double \*x, double \*w, int n, double alf);

double **int\_func**(double r1, double r2, double theta1, double theta2, double phi1, double phi2);

double **gammln**( double xx);

int **main**(int argc, char\* argv[])

{

//Definerer Closed form løsning 5\*Pi^2/16^2

double pi = 3.1415;

double sixteen = 16\*16;

double nevner = 1/sixteen;

double answer = 5\*pi\*pi\*nevner;

//Definerer alle variabler

int N = atoi(argv[1]);

double \*x = new double [N];

double \*w = new double [N];

double a = -3.1;

double b = -a;

double alf = 1.0;

double \*xgl1 = new double [N+1];

double \*wgl1 = new double [N+1];

double \*xgl2 = new double [N+1];

double \*wgl2 = new double [N+1];

double \*xgl3 = new double [N+1];

double \*wgl3 = new double [N+1];

double \*r = new double [N];

double \*s = new double [N];

// set up the mesh points and weights

gauss\_laguerre(xgl1,wgl1, N, alf);

gauleg(0,pi, xgl2, wgl2, N);

gauleg(0,2\*pi, xgl3, wgl3, N);

double int\_gauss = 0.;

clock\_t start, finish;

start = clock();

for (int i=0;i<N;i++){

for (int j = 0;j<N;j++){

for (int k = 0;k<N;k++){

for (int l = 0;l<N;l++){

for (int m = 0;m<N;m++){

for (int n = 0;n<N;n++){

int\_gauss+=wgl1[i]\*wgl1[j]\*wgl2[k]\*wgl2[l]\*wgl3[m]\*wgl3[n]\*int\_func(xgl1[i],xgl1[j],xgl2[k],xgl2[l],xgl3[m],xgl3[n]);

}}}}}

}

double **gammln**( double xx);

finish = clock();

double timeused = (double) (finish - start)/(CLOCKS\_PER\_SEC );

cout << "Running program with N value = " << N << endl;

cout << setiosflags(ios::showpoint | ios::uppercase);

cout << setprecision(10) << "Time used = " << timeused << endl;

cout << setprecision(3) << "The analytical solution is " << int\_gauss << endl;

cout << setprecision(3) << "The closed form answer is " << answer << endl;

}

double **gammln**( double xx)

{

double x,y,tmp,ser;

static double cof[6]={76.18009172947146,-86.50532032941677,

24.01409824083091,-1.231739572450155,

0.1208650973866179e-2,-0.5395239384953e-5};

int j;

y=x=xx;

tmp=x+5.5;

tmp -= (x+0.5)\*log(tmp);

ser=1.000000000190015;

for (j=0;j<=5;j++) ser += cof[j]/++y;

return -tmp+log(2.5066282746310005\*ser/x);

}

void **gauss\_laguerre**(double \*x, double \*w, int n, double alf)

{

int i,its,j;

double ai;

double p1,p2,p3,pp,z,z1;

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

if (i == 1) {

z=(1.0+alf)\*(3.0+0.92\*alf)/(1.0+2.4\*n+1.8\*alf);

} else if (i == 2) {

z += (15.0+6.25\*alf)/(1.0+0.9\*alf+2.5\*n);

} else {

ai=i-2;

z += ((1.0+2.55\*ai)/(1.9\*ai)+1.26\*ai\*alf/

(1.0+3.5\*ai))\*(z-x[i-2])/(1.0+0.3\*alf);

}

for (its=1;its<=10;its++) {

p1=1.0;

p2=0.0;

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

p3=p2;

p2=p1;

p1=((2\*j-1+alf-z)\*p2-(j-1+alf)\*p3)/j;

}

pp=(n\*p1-(n+alf)\*p2)/z;

z1=z;

z=z1-p1/pp;

if (fabs(z-z1) <= 1e-4) break;

}

if (its > 10) cout << "too many iterations in gaulag" << endl;

x[i]=z;

w[i] = -exp(gammln(alf+n)-gammln((double)n))/(pp\*n\*p2);

}

}

double **int\_func**(double r1, double r2, double theta1, double theta2, double phi1, double phi2){

double cosb = cos(theta1)\*cos(theta2) + sin(theta1)\*sin(theta2)\*cos(phi1-phi2);

double deno = r1\*r1 + r2\*r2 -2\*r1\*r2\*cosb;

if(deno < 1E-8){

return 0;

}

else{

return (r1\*r2\*sin(theta1)\*sin(theta2))/(1024.0\*sqrt(double(deno)));

}

}

void **gauleg**(double x1, double x2, double x[], double w[], int n)

{

int m,j,i;

double z1,z,xm,xl,pp,p3,p2,p1;

double const pi = 3.14159265359;

double \*x\_low, \*x\_high, \*w\_low, \*w\_high;

m = (n + 1)/2; // roots are symmetric in the interval

xm = 0.5 \* (x2 + x1);

xl = 0.5 \* (x2 - x1);

x\_low = x; // pointer initialization

x\_high = x + n - 1;

w\_low = w;

w\_high = w + n - 1;

for(i = 1; i <= m; i++) { // loops over desired roots

z = cos(pi \* (i - 0.25)/(n + 0.5));

/\*

\*\* Starting with the above approximation to the ith root

\*\* we enter the mani loop of refinement bt Newtons method.

\*/

do {

p1 =1.0;

p2 =0.0;

/\*

\*\* loop up recurrence relation to get the

\*\* Legendre polynomial evaluated at x

\*/

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

p3 = p2;

p2 = p1;

p1 = ((2.0 \* j - 1.0) \* z \* p2 - (j - 1.0) \* p3)/j;

}

/\*

\*\* p1 is now the desired Legrendre polynomial. Next compute

\*\* ppp its derivative by standard relation involving also p2,

\*\* polynomial of one lower order.

\*/

pp = n \* (z \* p1 - p2)/(z \* z - 1.0);

z1 = z;

z = z1 - p1/pp; // Newton's method

} while(fabs(z - z1) > ZERO);

/\*

\*\* Scale the root to the desired interval and put in its symmetric

\*\* counterpart. Compute the weight and its symmetric counterpart

\*/

\*(x\_low++) = xm - xl \* z;

\*(x\_high--) = xm + xl \* z;

\*w\_low = 2.0 \* xl/((1.0 - z \* z) \* pp \* pp);

\*(w\_high--) = \*(w\_low++);

}

} // End\_ function gauleg()

# 3c – Monte Carlo

FEIL::

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <fstream>

#include <iomanip>

#include <string>

#include <armadillo>

#include "time.h"

#include "lib.h"

#include <random>

using namespace std;

using namespace arma;

void **gauss\_laguerre**(double \*x, double \*w, int n, double alf);

double **int\_func**(double r1, double r2, double theta1, double theta2, double phi1, double phi2);

double **gammln**( double xx);

void **Brute\_MonteCarlo**(int n, double a, double b, double &integral, double &std);

int **main**(int argc, char\* argv[])

{

//Definerer Closed form løsning 5\*Pi^2/16^2

double pi = 3.1415;

double sixteen = 16\*16;

double nevner = 1/sixteen;

double answer = 5\*pi\*pi\*nevner;

//Definerer alle variabler

int N = atoi(argv[1]);

double \*x = new double [N];

double \*w = new double [N];

double a = -3.1;

double b = -a;

double alf = 1.0;

double \*xgl1 = new double [N+1];

double \*wgl1 = new double [N+1];

double \*xgl2 = new double [N+1];

double \*wgl2 = new double [N+1];

double \*xgl3 = new double [N+1];

double \*wgl3 = new double [N+1];

double \*r = new double [N];

double \*s = new double [N];

int n = 10000;

double integral;

double std;

// set up the mesh points and weights

gauss\_laguerre(xgl1,wgl1, N, alf);

gauleg(0,pi, xgl2, wgl2, N);

gauleg(0,2\*pi, xgl3, wgl3, N);

double **gammln**( double xx);

double int\_gauss = 0.;

clock\_t start, finish;

start = clock();

for (int i=0;i<N;i++){

for (int j = 0;j<N;j++){

for (int k = 0;k<N;k++){

for (int l = 0;l<N;l++){

for (int m = 0;m<N;m++){

for (int n = 0;n<N;n++){

int\_gauss+=wgl1[i]\*wgl1[j]\*wgl2[k]\*wgl2[l]\*wgl3[m]\*wgl3[n]\*int\_func(xgl1[i],xgl1[j],xgl2[k],xgl2[l],xgl3[m],xgl3[n]);

}}}}}

}

double **gammln**( double xx);

Brute\_MonteCarlo(n,a,b,*integral*, *std*);

finish = clock();

double timeused = (double) (finish - start)/(CLOCKS\_PER\_SEC );

cout << "Running program with N value = " << N << endl;

cout << setiosflags(ios::showpoint | ios::uppercase);

cout << setprecision(10) << "Time used = " << timeused << endl;

cout << setprecision(3) << "The analytical solution with improved Gauss legandre quadrature is " << int\_gauss << endl;

cout << setprecision(3) << "The closed form answer is " << answer << endl;

cout << "The answer using the Monte Carlo method looping over n equal to "<< n << ", is:" <<integral << endl;

}

double **gammln**( double xx)

{

double x,y,tmp,ser;

static double cof[6]={76.18009172947146,-86.50532032941677,

24.01409824083091,-1.231739572450155,

0.1208650973866179e-2,-0.5395239384953e-5};

int j;

y=x=xx;

tmp=x+5.5;

tmp -= (x+0.5)\*log(tmp);

ser=1.000000000190015;

for (j=0;j<=5;j++) ser += cof[j]/++y;

return -tmp+log(2.5066282746310005\*ser/x);

}

void **gauss\_laguerre**(double \*x, double \*w, int n, double alf)

{

int i,its,j;

double ai;

double p1,p2,p3,pp,z,z1;

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

if (i == 1) {

z=(1.0+alf)\*(3.0+0.92\*alf)/(1.0+2.4\*n+1.8\*alf);

} else if (i == 2) {

z += (15.0+6.25\*alf)/(1.0+0.9\*alf+2.5\*n);

} else {

ai=i-2;

z += ((1.0+2.55\*ai)/(1.9\*ai)+1.26\*ai\*alf/

(1.0+3.5\*ai))\*(z-x[i-2])/(1.0+0.3\*alf);

}

for (its=1;its<=10;its++) {

p1=1.0;

p2=0.0;

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

p3=p2;

p2=p1;

p1=((2\*j-1+alf-z)\*p2-(j-1+alf)\*p3)/j;

}

pp=(n\*p1-(n+alf)\*p2)/z;

z1=z;

z=z1-p1/pp;

if (fabs(z-z1) <= 1e-4) break;

}

if (its > 10) cout << "too many iterations in gaulag" << endl;

x[i]=z;

w[i] = -exp(gammln(alf+n)-gammln((double)n))/(pp\*n\*p2);

}

}

double **int\_func**(double r1, double r2, double theta1, double theta2, double phi1, double phi2){

double cosb = cos(theta1)\*cos(theta2) + sin(theta1)\*sin(theta2)\*cos(phi1-phi2);

double deno = r1\*r1 + r2\*r2 -2\*r1\*r2\*cosb;

if(deno < 1E-8){

return 0;

}

else{

return (r1\*r2\*sin(theta1)\*sin(theta2))/(1024.0\*sqrt(double(deno)));

}

}

void **gauleg**(double x1, double x2, double x[], double w[], int n)

{

int m,j,i;

double z1,z,xm,xl,pp,p3,p2,p1;

double const pi = 3.14159265359;

double \*x\_low, \*x\_high, \*w\_low, \*w\_high;

m = (n + 1)/2; // roots are symmetric in the interval

xm = 0.5 \* (x2 + x1);

xl = 0.5 \* (x2 - x1);

x\_low = x; // pointer initialization

x\_high = x + n - 1;

w\_low = w;

w\_high = w + n - 1;

for(i = 1; i <= m; i++) { // loops over desired roots

z = cos(pi \* (i - 0.25)/(n + 0.5));

/\*

\*\* Starting with the above approximation to the ith root

\*\* we enter the mani loop of refinement bt Newtons method.

\*/

do {

p1 =1.0;

p2 =0.0;

/\*

\*\* loop up recurrence relation to get the

\*\* Legendre polynomial evaluated at x

\*/

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

p3 = p2;

p2 = p1;

p1 = ((2.0 \* j - 1.0) \* z \* p2 - (j - 1.0) \* p3)/j;

}

/\*

\*\* p1 is now the desired Legrendre polynomial. Next compute

\*\* ppp its derivative by standard relation involving also p2,

\*\* polynomial of one lower order.

\*/

pp = n \* (z \* p1 - p2)/(z \* z - 1.0);

z1 = z;

z = z1 - p1/pp; // Newton's method

} while(fabs(z - z1) > ZERO);

/\*

\*\* Scale the root to the desired interval and put in its symmetric

\*\* counterpart. Compute the weight and its symmetric counterpart

\*/

\*(x\_low++) = xm - xl \* z;

\*(x\_high--) = xm + xl \* z;

\*w\_low = 2.0 \* xl/((1.0 - z \* z) \* pp \* pp);

\*(w\_high--) = \*(w\_low++);

}

} // End\_ function gauleg()

void **Brute\_MonteCarlo**(int n, double a, double b, double &integral, double &std){

random\_device rd;

mt19937\_64 gen(rd());

uniform\_real\_distribution<double> RandomNumberGenerator(0.0,1.0);

double \* x = new double [n];

double x1, x2, y1, y2, z1, z2, f;

double mc = 0.0;

double sigma = 0.0;

int i;

double jacob = pow((b-a),6);

#pragma omp parallel for reduction(+:mc) private (i, x1, x2, y1, y2, z1, z2, f)

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

x1=RandomNumberGenerator(gen)\*(b-a)+a;

x2=RandomNumberGenerator(gen)\*(b-a)+a;

y1=RandomNumberGenerator(gen)\*(b-a)+a;

y2=RandomNumberGenerator(gen)\*(b-a)+a;

z1=RandomNumberGenerator(gen)\*(b-a)+a;

z2=RandomNumberGenerator(gen)\*(b-a)+a;

f=int\_func(x1, x2, y1, y2, z1, z2);

mc += f;

x[i] = f;

}

mc = mc/((double) n );

#pragma omp parallel for reduction(+:sigma) private (i)

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

sigma += (x[i] - mc)\*(x[i] - mc);

}

sigma = sigma\*jacob/((double) n );

std = sqrt(sigma)/sqrt(n);

integral = mc\*jacob;

delete [] x;

}

# MPI Settings

QMAKE\_CXX = mpicxx

QMAKE\_CXX\_RELEASE = $$QMAKE\_CXX

QMAKE\_CXX\_DEBUG = $$QMAKE\_CXX

QMAKE\_LINK = $$QMAKE\_CXX

QMAKE\_CC = mpicc

QMAKE\_CFLAGS += $$system(mpicc --showme:compile)

QMAKE\_LFLAGS += $$system(mpicxx --showme:link)

QMAKE\_CXXFLAGS += $$system(mpicxx --showme:compile) -DMPICH\_IGNORE\_CXX\_SEEK

QMAKE\_CXXFLAGS\_RELEASE += $$system(mpicxx --showme:compile) -DMPICH\_IGNORE\_CXX\_SEEK