**Лабораторна** робота №5 **Оцінювання параметрів нелінійної стохастичної диференціальної системи узагальненим методом моменті**

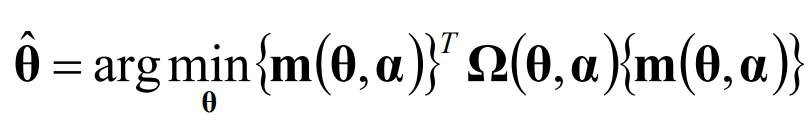
**Мета роботи**: отримати практичні навички оцінювання параметрів нелінійної стохастичної диференціальної системи узагальненим методом моментів із застосуванням методів оптимізації з використанням комп’ютера

**Завдання**

За реалізаціями випадкових процесів y(t) і y’(t), що отримані в роботі № 4 у разі, коли в якості моделі випадкового впливу x(t) використовується білий шум, виконати оцінювання параметрів нелінійної стохастичної диференціальної системи(СДС), поведінка якої описується СДР, узагальненим методом моментів. А саме: за узагальненим методом моментів знайти точкові оцінки b1 , b2 , c1 і c3 відповідних параметрів b1 , b2 , c1 і c3 . Представити графіки b1(t), b2(t), c1(t) і c3(t), а також точкові оцінки дисперсій відповідних параметрів b1, b2, c1 і c3

Хід роботи

Оцінювання невідомих параметрів нелінійної стохастичної диференціальної системи узагальненим методом моментів здійснюється в результаті вирішення наступної оптимізаційної задачі (задачі математичного програмування):



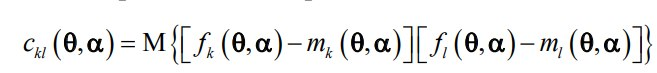
де **m(θ, α)** – вектор моментних умов; **θ** – вектор параметрів СДР, які потребують оцінювання; **α** – вектор статистичних моментів; **Ω(θ, α)** – додатна напіввизначена вагова матриця, **Ω-1 = C** ; **C** – кореляційна матриця. Фактично елементи вагової матриці **Ω(θ, α)** визначають ті вагові коефіцієнти, з якими беруться моментні умови при оцінюванні вектору **θ**. Для вирішення використовуватиметься метод градієнтного спуску.

Перші n компонент вектору **m(θ, α)** визначаються як:

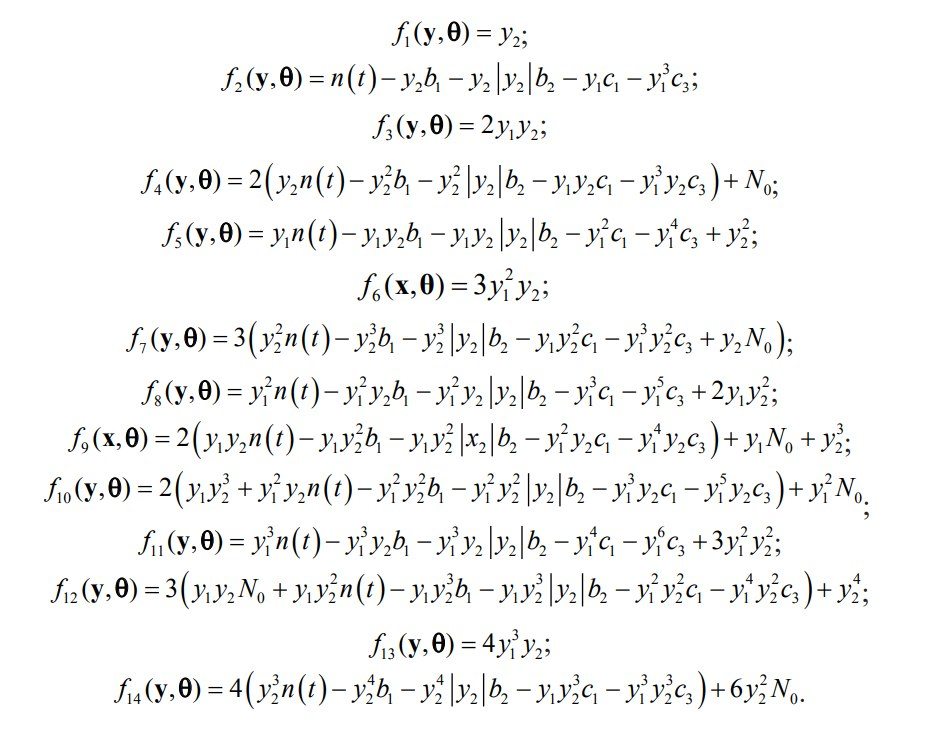


де M{ } – операція математичного сподівання.

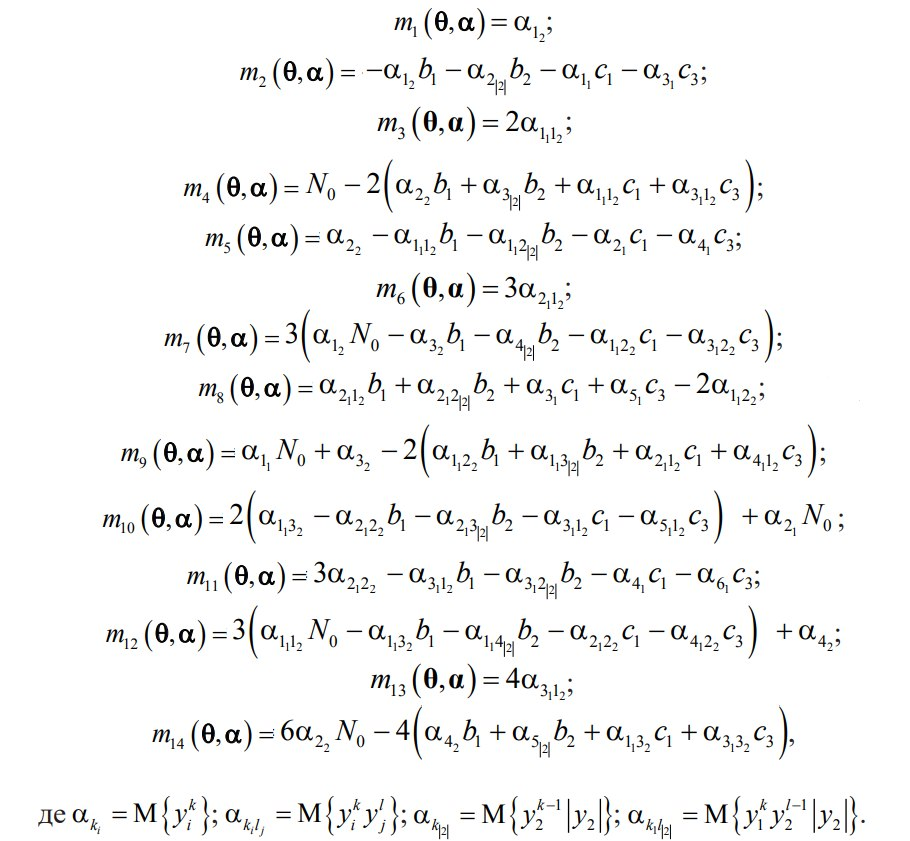
Елементи кореляційної матриці знаходять як:



де



Використовуючи вирази вище дозволяють записати компоненти вектору **m(θ, α)** наступним чином:



Перевага УММ в порівнянні з таким відомим методом як метод максимальної правдоподібності полягає у тому що для визначення **m(θ, α)** та **Ω(θ, α)** не потрібно знати аналітичний вираз функції щільності ймовірності в залежності від вектору θ та компонентів СДР.

Але недолік УММ пов’язаний з проблемою стрімкого ускладнення рішення задачі при збільшенні компонентів СДР.

Часова залежність b1(t) представлена на рисунку нижче:

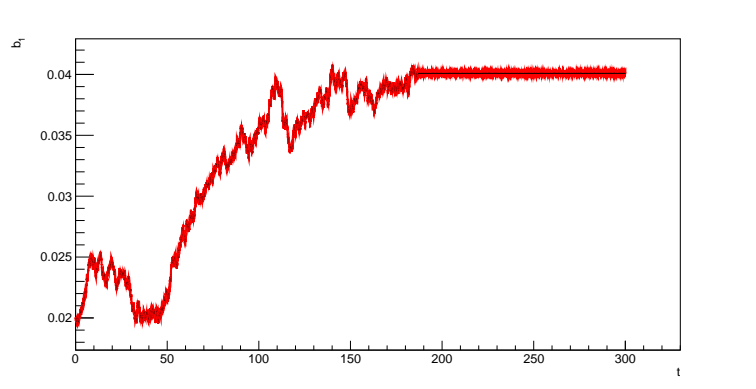


Рисунок 1 – Графік функції

Часова залежність b2(t) представлена на рисунку нижче:

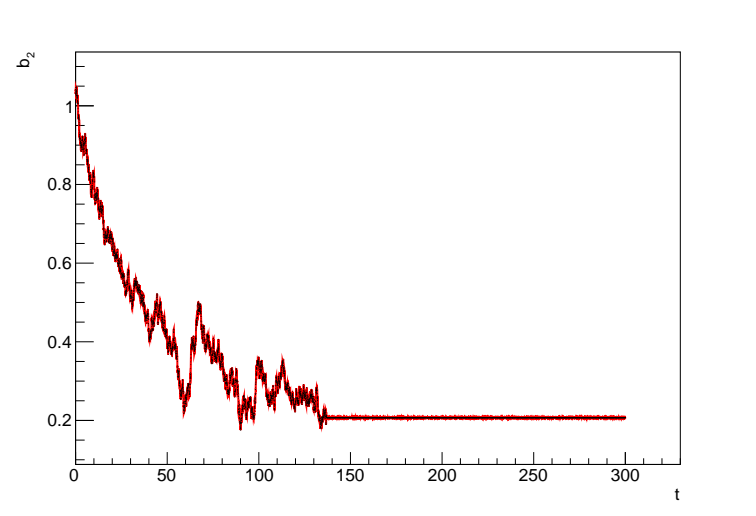
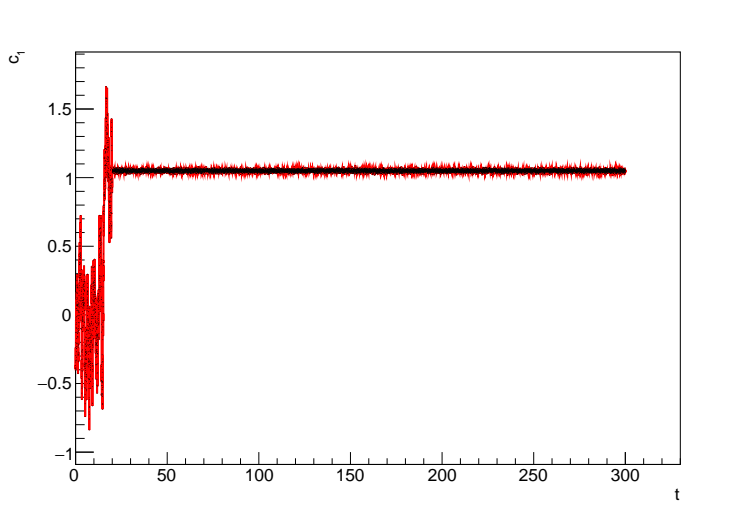
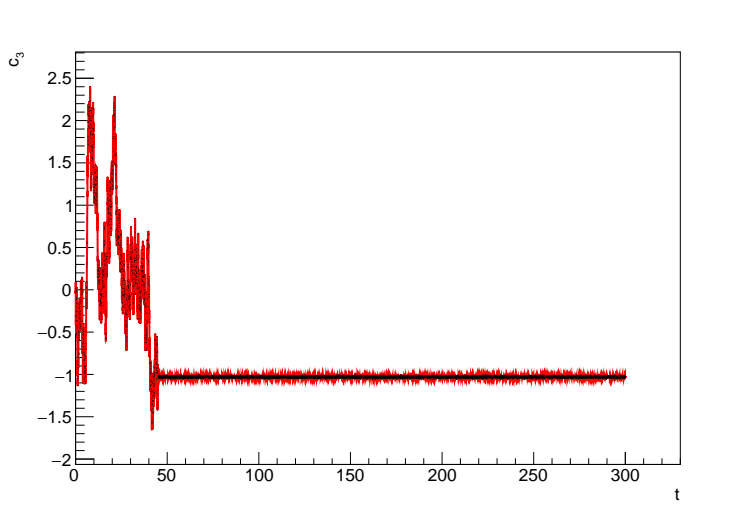


Рисунок 2 – Графік функції

Часова залежність c1(t) представлена на рисунку нижче:

Рисунок 3 – Графік функції

Часова залежність c3(t) представлена на рисунку нижче:

Рисунок 4 – Графік функції

**Висновки:** за результатами виконання лабораторної роботи було оцінено параметри нелінійної стохастичної диференціальної системи узагальненим методом моментів.

Побудовані графіки та отримані за ними графіки дозволяють зробити висновок про успішність оцінки параметрів нелінійної стохастичної системи узагальненим методом моментів.

#include <iostream>

#include <cmath>

#include <vector>

#include <random>

#include <fstream>

#include "inverse\_matrix.cpp"

#define N0 1

#define dt 0.0195

#define epsilon 0.1

std::default\_random\_engine generator;

std::normal\_distribution<double> distribution(0,1.0);

double gaussian(double x, double mu=0, double sig=1)

{

return 1/sqrt(2\*M\_PI\*sig\*sig)\*exp(-1\*(x-mu)\*(x-mu)/2/sig/sig);

}

double mean(vector<double> A)

{

if(A.size() != 0)

{

double mean=0;

for (double a : A)

{

mean+=a;

}

return mean=mean/A.size();

}

else

{

return 0;

}

}

double stdDev(vector<double> A)

{

if(A.size() != 0)

{

double stdDev=0;

double m = mean(A);

for (double a : A)

{

stdDev+=(a-m)\*(a-m);

}

return sqrt(stdDev/(A.size()-1));

}

else

{

return 0;

}

}

double specialMean(vector<double> A, vector<double> B = {}, int pow1=1, int pow2=1, int mode = 0)

{

if(A.size() != 0)

{

if(B.size()==0)

{

if(mode==1)

{

double mean=0;

for(int i=0;i<A.size();i++)

{

mean+=pow(A[i],pow1-1)\*fabs(A[i]);

}

return mean=mean/A.size();

}

else

{

double mean=0;

for (double a : A)

{

mean+=pow(a,pow1);

}

return mean=mean/A.size();

}

}

else

{

if(mode==0)

{

double mean=0;

for(int i=0;i<A.size();i++)

{

mean+=pow(A[i],pow1)\*pow(B[i],pow2);

}

return mean=mean/A.size();

}

else if(mode==2)

{

double mean=0;

for(int i=0;i<A.size();i++)

{

mean+=pow(A[i],pow1)\*pow(B[i],pow2-1)\*fabs(B[i]);

}

return mean=mean/A.size();

}

else

{

return 0;

}

}

}

else

{

return 0;

}

}

vector<double> F(double y1,double y2, vector<double> z) {

double nt = gaussian(distribution(generator),0,1)\*sqrt(N0/dt);

vector<double> f = {};

f.push\_back(y2);

f.push\_back(nt-y2\*z[0]-y2\*fabs(y2)\*z[1]-y1\*z[2]-y1\*y1\*y1\*z[3]);

f.push\_back(2\*y1\*y2);

f.push\_back(2\*(y2\*nt-y2\*y2\*z[0]-y2\*fabs(y2)\*z[1]-y1\*y2\*z[2]-y1\*y1\*y1\*y2\*z[3])+N0);

f.push\_back(y1\*nt-y1\*y2\*z[0]-y1\*y2\*fabs(y2)\*z[1]-y1\*y1\*z[2]-y1\*y1\*y1\*y1\*z[3]+y2\*y2);

f.push\_back(3\*y1\*y1\*y2);

f.push\_back(3\*(y2\*y2\*nt-y2\*y2\*z[0]-y2\*y2\*y2\*fabs(y2)\*z[1]-y1\*y2\*y2\*z[2]-y1\*y1\*y1\*y2\*y2\*z[3]+y2\*N0));

f.push\_back(y1\*y1\*nt-y1\*y1\*y2\*z[0]-y1\*y1\*y2\*fabs(y2)\*z[1]-y1\*y1\*y1\*z[2]-y1\*y1\*y1\*y1\*y1\*z[3]+2\*y1\*y2\*y2);

f.push\_back(2\*(y1\*y2\*nt-y1\*y2\*y2\*z[0]-y1\*y2\*y2\*fabs(y2)\*z[1]-y1\*y1\*y2\*z[2]-y1\*y1\*y1\*y1\*y2\*z[3])+y1\*N0+y2\*y2\*y2);

f.push\_back(2\*(y1\*y2\*y2\*y2+y1\*y1\*y2\*nt-y1\*y1\*y2\*y2\*z[0]-y1\*y1\*y2\*y2\*fabs(y2)\*z[1]-y1\*y1\*y1\*y2\*z[2]-y1\*y1\*y1\*y1\*y1\*y2\*z[3])+y1\*y1\*N0);

f.push\_back(y1\*y1\*y1\*nt-y1\*y1\*y1\*y2\*z[0]-y1\*y1\*y1\*y2\*fabs(y2)\*z[1]-y1\*y1\*y1\*y1\*z[2]-y1\*y1\*y1\*y1\*y1\*y1\*z[3]+3\*y1\*y1\*y2\*y2);

f.push\_back(3\*(y1\*y2\*N0+y1\*y2\*y2\*nt-y1\*y2\*y2\*y2\*z[0]-y1\*y2\*y2\*y2\*fabs(y2)\*z[1]-y1\*y1\*y2\*y2\*z[2]-y1\*y1\*y1\*y1\*y2\*y2\*z[3])+y2\*y2\*y2\*y2);

f.push\_back(4\*y1\*y1\*y1\*y2);

f.push\_back(4\*(y2\*y2\*y2\*nt-y2\*y2\*y2\*y2\*z[0]-y2\*y2\*y2\*y2\*fabs(y2)\*z[1]-y1\*y2\*y2\*y2\*z[2]-y1\*y1\*y1\*y2\*y2\*y2\*z[3])+6\*y2\*y2\*N0);

return f;

}

vector<double> M(vector<double> Y1,vector<double> Y2, vector<double> z) {

vector<double> m = {};

m.push\_back(specialMean(Y2));

m.push\_back(-z[0]\*specialMean(Y2)-specialMean(Y2,{},2,1,1)\*z[1]-specialMean(Y1)\*z[2]-specialMean(Y1,{},3)\*z[3]);

m.push\_back(2\*specialMean(Y1,Y2));

m.push\_back(N0-2\*(specialMean(Y2,{},2)\*z[0]+specialMean(Y2,{},3,0,1)\*z[1]+specialMean(Y1,Y2)\*z[2]+specialMean(Y1,Y2,3,1)\*z[3]));

m.push\_back(specialMean(Y2,{},2)-specialMean(Y1,Y2)\*z[0]-specialMean(Y1,Y2,1,2,2)\*z[1]-z[2]\*specialMean(Y1,{},4));

m.push\_back(3\*specialMean(Y1,Y2,2,1));

m.push\_back(3\*(specialMean(Y2)\*N0-specialMean(Y2,{},3)\*z[0]-z[1]\*specialMean(Y2,{},4,0,1)-z[2]\*specialMean(Y1,Y2,1,2)-z[3]\*specialMean(Y1,Y2,3,2)));

m.push\_back(specialMean(Y1,Y2,2,1)\*z[0]+specialMean(Y1,Y2,2,2,2)\*z[1]+specialMean(Y1,{},3)\*z[2]+z[3]\*specialMean(Y1,{},5)-2\*specialMean(Y1,Y2,1,2));

m.push\_back(specialMean(Y1)\*N0+specialMean(Y2,{},3)-2\*(specialMean(Y1,Y2,1,2)\*z[0]+specialMean(Y1,Y2,1,3,2)\*z[1]+z[2]\*specialMean(Y1,Y2,2,1)+z[3]\*specialMean(Y1,Y2,4,1)));

m.push\_back(specialMean(Y1,{},2)\*N0+2\*(specialMean(Y1,Y2,1,3)-specialMean(Y1,Y2,2,2)\*z[0]-specialMean(Y1,Y2,2,3,2)\*z[1]-specialMean(Y1,Y2,3,1)\*z[2]-specialMean(Y1,Y2,5,1)\*z[3]));

m.push\_back(3\*specialMean(Y1,Y2,2,2)-specialMean(Y1,Y2,3,1)\*z[0]-z[1]\*specialMean(Y1,Y2,3,2,2)-z[2]\*specialMean(Y1,{},4)-z[2]\*specialMean(Y1,{},6));

m.push\_back(specialMean(Y2,{},4)+3\*(specialMean(Y1,Y2,1,1)\*N0-specialMean(Y1,Y2,1,3)\*z[0]-z[1]\*specialMean(Y1,Y2,1,4,2)-specialMean(Y1,Y2,2,2)\*z[2]-z[3]\*specialMean(Y1,Y2,4,2)));

m.push\_back(4\*specialMean(Y1,Y2,3,1));

m.push\_back(6\*specialMean(Y2,{},2)\*N0-4\*(specialMean(Y2,{},4)\*z[0]+specialMean(Y2,{},5,0,1)\*z[1]+specialMean(Y1,Y2,1,3)\*z[2]+specialMean(Y1,Y2,3,3)\*z[2]));

return m;

}

vector<vector<double>> covarianceMatrix(vector<double> Y1,vector<double> Y2, vector<double> z){

vector<vector<double>> C;

vector<double> m = M(Y1,Y2,z);

for(int k=0; k<14;k++) {

for(int l=0; l<14;l++) {

for(int i=0; i<Y1.size();i++) {

C[k][l]+=(F(Y1[i],Y2[i],z)[k]-m[k])\*(F(Y1[i],Y2[i],z)[l]-m[l]);

}

if(Y1.size()!=0) {C[k][l]/=Y1.size();}

else {C[k][l]=0;}

}

}

return C;

}

vector<double> gradient(vector<double> Y1, vector<double> Y2, vector<double> z, double step=0.01) {

vector<double> grad = {};

vector<double> prevM = M(Y1,Y2,z);

vector<vector<double>> prevC = getInverse(covarianceMatrix(Y1,Y2,z));

vector<vector<double>> C0 = getInverse(covarianceMatrix(Y1,Y2,{z[0]+step,z[1],z[2],z[3]}));

vector<double> M0 = M(Y1,Y2,{z[0]+step,z[1],z[2],z[3]});

vector<vector<double>> C1 = getInverse(covarianceMatrix(Y1,Y2,{z[0],z[1]+step,z[2],z[3]}));

vector<double> M1 = M(Y1,Y2,{z[0],z[1]+step,z[2],z[3]});

vector<vector<double>> C2 = getInverse(covarianceMatrix(Y1,Y2,{z[0],z[1],z[2]+step,z[3]}));

vector<double> M2 = M(Y1,Y2,{z[0],z[1],z[2]+step,z[3]});

vector<vector<double>> C3 = getInverse(covarianceMatrix(Y1,Y2,{z[0],z[1],z[2],z[3]+step}));

vector<double> M3 = M(Y1,Y2,{z[0],z[1],z[2],z[3]+step});

double prev,nextZ0,nextZ1,nextZ2,nextZ3;

for(int k=0;k<14;k++) {

for(int l=0;l<14;l++) {

prev = prevM[k]\*prevC[k][l]\*prevM[l];

nextZ0 = M0[k]\*C0[k][l]\*M0[l];

nextZ1 = M1[k]\*C1[k][l]\*M1[l];

nextZ2 = M2[k]\*C2[k][l]\*M2[l];

nextZ3 = M3[k]\*C3[k][l]\*M3[l];

}

}

grad.push\_back((nextZ0-prev)/step);

grad.push\_back((nextZ1-prev)/step);

grad.push\_back((nextZ2-prev)/step);

grad.push\_back((nextZ3-prev)/step);

return grad;

}

void script() {

vector<double> T = {};

vector<double> b1 = {};

vector<double> b2 = {};

vector<double> c1 = {};

vector<double> c3 = {};

vector<double> Y1 = {};

vector<double> Y2 = {};

ifstream file("data.txt");

double l1,l2,l3;

T.clear();

Y1.clear();

Y2.clear();

while(!file.eof()) {

file>>l1>>l2>>l3;

T.push\_back(l1);

Y1.push\_back(l2);

Y2.push\_back(l3);

}

ofsteam output("output.txt");

for(int j=0;j<T.size();j++) {

vector<double> Zn = {0.02,0,0,0};

vector<double> ZnPrev;

bool run = true;

double gamma = 0.2;

do {

vector<double> gr = gradient(Y1,Y2,Zn);

ZnPrev = Zn;

int counter = 0;

for(int i=0; i<Zn.size();i++) {

Zn[i]-=gamma\*gr[i];

if(abs(Zn[i]-ZnPrev[i])<epsilon\*Zn[i]/4.){counter+=1;}

}

if(counter>3)

{

run=false;

b1.push\_back(Zn[0]);

b2.push\_back(Zn[1]);

b1.push\_back(Zn[2]);

b3.push\_back(Zn[3]);

}

}

while(run);

output<<b1.last()<<" "<< b2.last()<<" "<<c1.last()<<" "<<c3.last()<<endl;

}

}