**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №6**

**по дисциплине «Вычислительная математика»**

**Тема: Исследование обусловленности задачи решения систем линейных уравнений.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1304 |  | Шаврин А.П. |
| Преподаватель |  | Попова Е.В. |

Санкт-Петербург

2022

**Цель работы.**

Изучение стандартной обусловленности задач решения систем линейных уравнений при различных вариантах неточных входных данных.

**Основные теоретические положения.**

Рассматривается система линейных уравнений n-го порядка с вещественными коэффициентами (1)



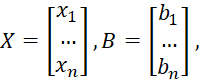
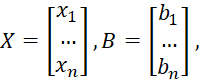




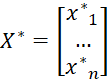
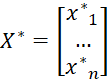


В матричной форме записи эта система принимает вид (2)



,  (2)

где  – квадратная матрица коэффициентов системы,  – вектор решений системы,  – вектор свободных членов. Матрица  – невырожденная, тогда решение системы (1) существует, единственно и устойчиво по входным данным. Это означает, что задача нахождения вектора  – корректна.

Пусть  – приближенное решение системы, тогда  называется вектором погрешности системы, необходимо стремиться к его уменьшению. Возможно рассматривать критерий малости вектора  который называется невязкой системы. Эти вектора связаны **.**

Удобной количественной характеристикой вектора является норма вектора. В вычислительной математике используются следующие три нормы (3)

   (3)

За норму матрицы принимают максимальную величину, на которую преобразование, описываемое матрицей, может растянуть любой ненулевой вектор в выбранной норме . Векторным нормам подчинены следующие нормы матрицы (4)







где  – собственные числа матрицы  Задача вычисления вектора  может быть плохо или хорошо обусловлена.

**Обусловленность задачи решения систем линейных алгебраических уравнений**

Рассмотрим случай, когда элементы матрицы  заданы точно, а вектор-столбец свободных членов – приближенно. Оценки для абсолютной и относительной погрешности (5)





где - абсолютное число обусловленности, а  - относительное число обусловленности (естественное число обусловленности). Максимальное естественное число обусловленности (6)

 (6)

называют стандартным числом обусловленности.



Если элементы матрицы  заданы приближенно и равны , а вектор-столбец свободных членов – точно, тогда оценка относительной погрешности (7)

 (7)

где  и .

Если с погрешностью заданы как коэффициенты матрицы, так и элементы вектора свободных членов, то справедливо неравенство (8)

 (8)

**Использование wxMaxima для подсчета обратной матрицы**

Матрица  – невырожденная, следовательно существует единственная обратная матрица . Для ее подсчета используется свободно распространяемый пакет системы компьютерной алгебры wxMaxima. Входная матрица задаётся с помощью выражения **matrix**(*стр1, стр2, ... стрN*), а обратная получается с помощью функции **invert**(*M*) (рисунок 1)

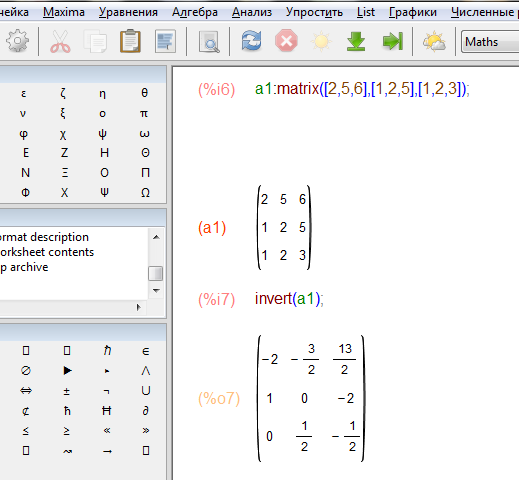


Рисунок 1 – Вычисление обратной матрицы с помощью функции *invert*

**Порядок выполнения работы.**

1. Составить подпрограмму для решения системы линейных уравнений методом Гаусса и методом обратной матрицы.
2. Решить систему, подсчитать стандартное число обусловленности, используя тип данных с двойной точностью. Подсчет обратной матрицы производить с помощью системы компьютерной алгебры wxMaxima.
3. С помощью встроенной функции – генератора случайных чисел, добавить ошибки в вектор свободных членов. Найти решение новой системы, стандартное число обусловленности (6) и оценку стандартного числа обусловленности (7).
4. С помощью встроенной функции – генератора случайных чисел, добавить ошибки в значения элементов матрицы. Найти решение новой системы, стандартное число обусловленности и оценку стандартного числа обусловленности.
5. Добавить ошибки в значения элементов матрицы и вектора свободных членов. Найти решение новой системы, стандартное число обусловленности и оценку стандартного числа обусловленности.
6. Сделать выводы по полученным значениям.

**Варианты заданий практической работы**

Варианты заданий соответствуют списку группы. Первая матрица получается из матрицы варианта путем добавления столбца и строки так, чтобы новая матрица размерности 4 на 4 была невырожденной.

Другая матрица получается из новой заменой 2 строк (для четных номеров) и 2 столбцов (для нечетных номеров) на соответствующие элементы матрицы Гильберта ().

**Выполнение работы:**

**Вариант 3.**

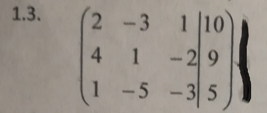


Рисунок 2. Исходная матрица.

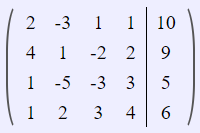
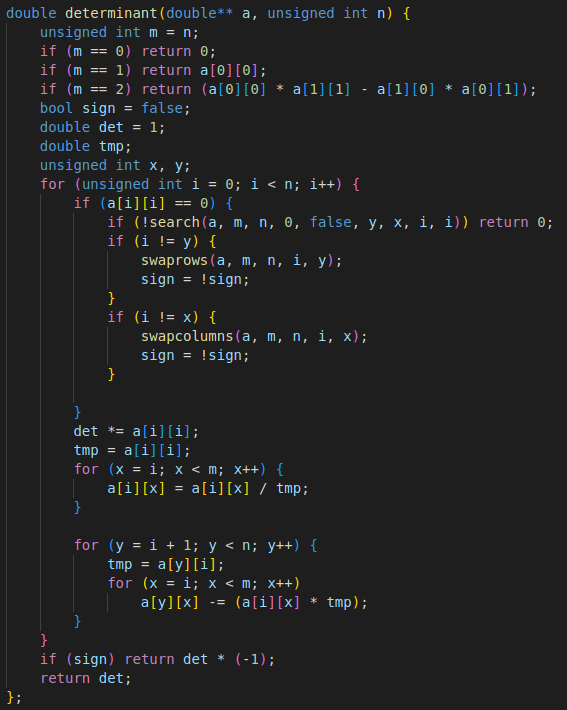


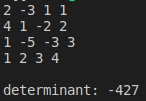
Рисунок 3. Матрица после добавления строки и столбца.

Определитель матрицы A не равен 0, следовательно матрица не вырожденная.

Программа так же считает определитель в функции determinant.



Результат -427.



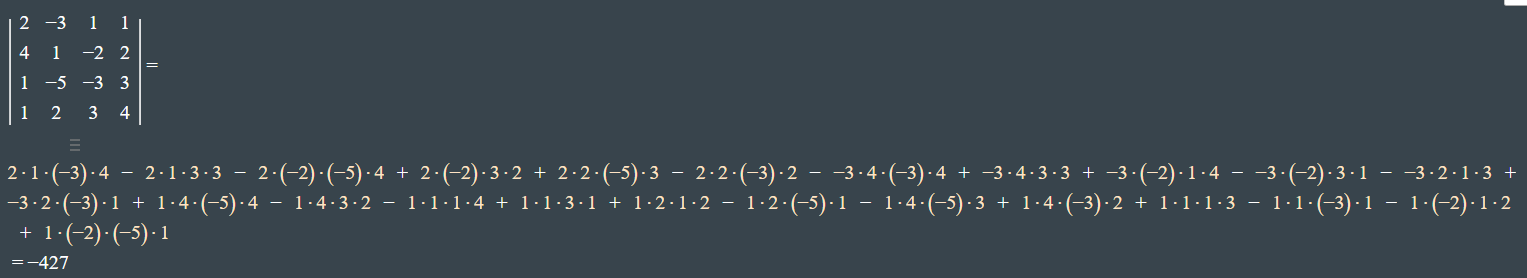


Рисунок 3.1 Определитель не 0 -> матрица невырожденная.

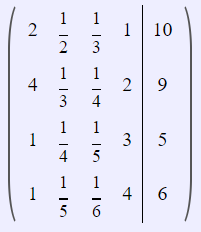


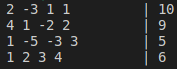
Рисунок 4. Матрица после замены 2 столбцов на соответствующие элементы матрицы Гильберта.

1. Функция Gauss() – принимает матрицу и вектор решений B (свободных членов), возвращает вектор значений x. Функция реализует метод Гаусса для поиска решения для системы линейных уравнений.

Функция ReverseMethod() - принимает матрицу, вектор решений и определитель, возвращает вектор значений x. Функция реализует метод обратной матрицы для поиска решения для системы линейных уравнений. Внутри функции вызывается метод ReverseMatr, который вычисляет обратную матрицу через миноры переданной матрицы.

Для матриц и векторов была выбрана норма 1.

1. Без внесения ошибок.



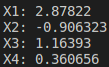


Рисунок 5. Решение системы.

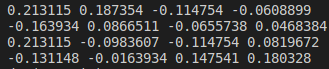


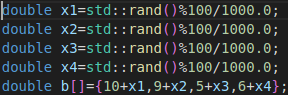
Рисунок 6. Обратная матрица.

Cond(A) = 7.93443

Абсолютное число обусловленности = 0.721311

Естественное число обусловленности = 4.07587

1. Ошибки свободных членов добавляются с помощью функции rand() из стандартной библиотеки, к каждому элементу прибавляется число в диапазоне от 0.0 до 0.1



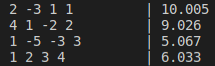


Рисунок 7. Добавление ошибок свободных членов

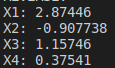


Рисунок 8. Новое решение системы.

Cond(A) = 7.93443

Абсолютная погрешность X = 1.86029

X\* = 0.350395

B\* = 0.672467

Абсолютное число обусловленности = 0.721311

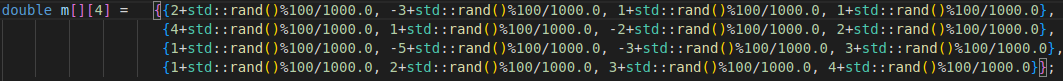
Естественное число обусловленности = 4.07032

Абсолютная погрешность абсолютного числа обусловленности = 0

Абсолютная погрешность естественного числа обусловленности = 0,00555

Вывод: Абсолютное число обусловленности не изменилось, поскольку в матрицу не вносились погрешности. Естественное число обусловленности уменьшилось, поскольку в вектор свободных членов вносились погрешности и в векторе решений они также присутствуют.

X\*<=B\*)\*Cond(A), следовательно обусловленность хорошая

1. Ошибки элементов матрицы добавляются с помощью функции rand() из стандартной библиотеки, к каждому элементу прибавляется число в диапазоне от 0.0 до 0.1

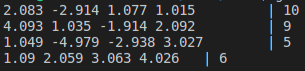


Рисунок 11. Добавление ошибок элементов матрицы



Рисунок 12. Решение системы.

Cond(A) = 7.93443

Абсолютная погрешность X = 1.94217

X\* = 0.365817

A\* = 0.998818

Абсолютное число обусловленности = 0.717235

Естественное число обусловленности = 4.16974

Абсолютная погрешность абсолютного числа обусловленности = 0,004065

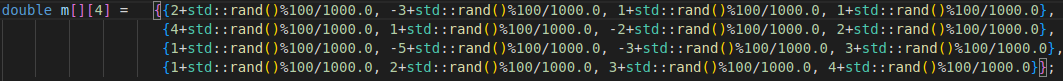
Абсолютная погрешность естественного числа обусловленности = 0,09394

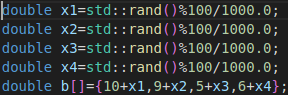
Вывод: Абсолютное число обусловленности уменьшилось, поскольку в матрицу вносились погрешности. Естественное число обусловленности увеличилось, поскольку в матрицу вносились погрешности и в векторе решений они также присутствуют.

X < A\*)\*Cond(A), следовательно обусловленность хорошая

оо

1. Ошибки свободных членов и элементов матрицы добавляются с помощью функции rand() из стандартной библиотеки, к каждому элементу прибавляется число в диапазоне от 0.0 до 0.1





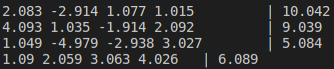


Рисунок 9. Добавление ошибок свободных членов и элементов матрицы.

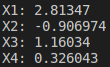


Рисунок 10. Решение системы.

Cond(A) = 7.88027

Абсолютная погрешность X = 1.91625

X\* = 0.360935

B\* = 0.674433

A\* = 0.998818

Абсолютное число обусловленности = 0.717235

Естественное число обусловленности = 4.16457

Абсолютная погрешность абсолютного числа обусловленности = 0,004076

Абсолютная погрешность естественного числа обусловленности = 0,0887

Вывод: Абсолютное число обусловленности уменьшилось, поскольку в матрицу вносились погрешности. Естественное число обусловленности увеличилось, поскольку в вектор свободных членов вносились погрешности и в векторе решений они также присутствуют.

X\*<=A\* + B\*)\*Cond(A), следовательно обусловленность хорошая

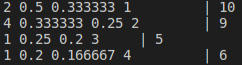
Вывод общий:

Наибольшее абсолютное число обусловленности встречается в матрицах, где изменяются элементы матрицы.

Наименьшее абсолютное число обусловленности встречается в матрице при внесении погрешности в вектор свободных значений.

Наименьшее естественное число обусловленности у матрицы, где изменяются элементы матрицы, а наибольшее у матрицы с “испорченным” вектором свободных значений.

1. Матрица со столбцами из матрицы гильберта.



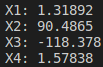


Рисунок 13. Решение системы.

Cond(A) = 1489.19

Абсолютная погрешность X = 25.9598

X\* = 39.9123

A\* = 0.909091

Абсолютное число обусловленности = 148.919

Естественное число обусловленности = 21.0971

Абсолютная погрешность абсолютного числа обусловленности = 148,19769

Абсолютная погрешность естественного числа обусловленности = 17,02123

Вывод: X\*>A\*)\*Cond(A), следовательно обусловленность плохая

**Выводы.**

Была изучена стандартная обусловленность задач решения систем линейных уравнений при различных вариантах неточных входных данных.

Наибольшее абсолютное число обусловленности встречается в матрице Гилберта, следовательно у данной матрицы самая большая погрешность.

Наименьшее абсолютное число обусловленности встречается в матрице при внесении погрешности в вектор свободных значений, следовательно у данной матрицы самая маленькая погрешность.

**ПРИЛОЖЕНИЕ А  
ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: lab6.cpp

#include <iostream>

#include <ctime>

#include <cmath>

#define SIZE 4

double\* Gauss(double\*\* matr,double\* b){

double \*x, max;

int k, index;

double eps = 0.00001;

x = new double[SIZE];

k = 0;

while (k < SIZE)

{

max = abs(matr[k][k]);

index = k;

for (int i = k + 1; i < SIZE; i++)

{

if (abs(matr[i][k]) > max)

{

max = abs(matr[i][k]);

index = i;

}

}

if (max < eps)

{

std::cout << "Решение получить невозможно из-за нулевого столбца ";

std::cout << index << " матрицы A\n";

return 0;

}

for (int j = 0; j < SIZE; j++)

{

double temp = matr[k][j];

matr[k][j] = matr[index][j];

matr[index][j] = temp;

}

double temp = b[k];

b[k] = b[index];

b[index] = temp;

for (int i = k; i < SIZE; i++)

{

double temp = matr[i][k];

if (abs(temp) < eps)

continue;

for (int j = 0; j < SIZE; j++)

matr[i][j] = matr[i][j] / temp;

b[i] = b[i] / temp;

if (i == k)

continue;

for (int j = 0; j < SIZE; j++)

matr[i][j] = matr[i][j] - matr[k][j];

b[i] = b[i] - b[k];

}

k++;

}

for (k = SIZE - 1; k >= 0; k--)

{

x[k] = b[k];

for (int i = 0; i < k; i++)

b[i] = b[i] - matr[i][k] \* x[k];

}

return x;

}

int search(double\*\* a, int m, int n, double what,

bool match, unsigned int& uI, unsigned int& uJ, unsigned int starti, unsigned int startj) {

if ((!m) || (!n)) return 0;

if ((starti >= n) || (startj >= m)) return 0;

for (unsigned int i = starti; i < n; i++)

for (unsigned int j = startj; j < m; j++) {

if (match == true) {

if (a[i][i] == what) {

uI = i; uJ = j; return 1;

}

}

else if (a[i][j] != what) {

uI = i; uJ = j; return 1;

}

}

return 0;

}

void swaprows(double\*\* a, int n, int m, unsigned int x1, unsigned int x2) {

if ((!n) || (!m)) return;

if ((x1 >= n) || (x2 >= n) || (x1 == x2)) return;

double tmp;

for (unsigned int x = 0; x < m; x++) {

tmp = a[x1][x];

a[x1][x] = a[x2][x];

a[x2][x] = tmp;

}

return;

};

void swapcolumns(double\*\* a, int n, int m, unsigned int x1, unsigned int x2) {

if ((!n) || (!m)) return;

if ((x1 >= m) || (x2 >= m) || (x1 == x2)) return;

double tmp;

for (unsigned int x = 0; x < n; x++) {

tmp = a[x][x1];

a[x][x1] = a[x][x2];

a[x][x2] = tmp;

}

return;

};

double determinant(double\*\* a, unsigned int n) {

unsigned int m = n;

if (m == 0) return 0;

if (m == 1) return a[0][0];

if (m == 2) return (a[0][0] \* a[1][1] - a[1][0] \* a[0][1]);

bool sign = false;

double det = 1;

double tmp;

unsigned int x, y;

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

if (a[i][i] == 0) {

if (!search(a, m, n, 0, false, y, x, i, i)) return 0;

if (i != y) {

swaprows(a, m, n, i, y);

sign = !sign;

}

if (i != x) {

swapcolumns(a, m, n, i, x);

sign = !sign;

}

}

det \*= a[i][i];

tmp = a[i][i];

for (x = i; x < m; x++) {

a[i][x] = a[i][x] / tmp;

}

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

tmp = a[y][i];

for (x = i; x < m; x++)

a[y][x] -= (a[i][x] \* tmp);

}

}

if (sign) return det \* (-1);

return det;

};

double minor(double \*\*matr,int fi,int fj){

double m=0.0;

double \*\*t\_matr = new double\*[SIZE];

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

t\_matr[i] = new double[SIZE];

}

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

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

t\_matr[j][i]=matr[i][j];

}

}

double \*\*A = new double\*[SIZE-1];

for(int i=0;i<SIZE-1;i++){

A[i] = new double[SIZE-1];

}

fi-=1,fj-=1;

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

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

if(i!=fi && j!=fj){

if(i>fi) {

if (j > fj)

A[i - 1][j - 1] = t\_matr[i][j];

else

A[i - 1][j] = t\_matr[i][j];

}else{

if (j > fj)

A[i][j - 1] = t\_matr[i][j];

else

A[i][j] = t\_matr[i][j];

}

}

}

}

m = (A[0][0]\*A[1][1]\*A[2][2] + A[0][1]\*A[1][2]\*A[2][0] + A[0][2]\*A[1][0]\*A[2][1]

-A[0][2]\*A[1][1]\*A[2][0] - A[0][1]\*A[1][0]\*A[2][2] - A[0][0]\*A[1][2]\*A[2][1] );

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

delete t\_matr[i];

}

delete[] t\_matr;

for(int i=0;i<SIZE-1;i++){

delete A[i];

}

delete[] A;

if((fi+fj)%2==0)

return m;

return (-1.0)\*m;

}

void ReverseMatr(double\*\* matr,double det){

double \*\*A = new double\*[SIZE];

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

A[i] = new double[SIZE];

}

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

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

A[i][j]=minor(matr,i+1,j+1)/det;

}

}

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

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

matr[i][j]=A[i][j];

}

}

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

delete A[i];

}

delete[] A;

}

double\* ReverseMethod(double\*\* rev\_matr,double\* b,double det){

double\* x= new double[SIZE];

ReverseMatr(rev\_matr,det);

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

double tmp=0;

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

tmp+=rev\_matr[i][j]\*b[j];

}

x[i]=tmp;

}

return x;

}

int main() {

double\* x;

double m[][4] = {{2+std::rand()%100/1000.0, -3+std::rand()%100/1000.0, 1+std::rand()%100/1000.0, 1+std::rand()%100/1000.0},

{4+std::rand()%100/1000.0, 1+std::rand()%100/1000.0, -2+std::rand()%100/1000.0, 2+std::rand()%100/1000.0},

{1+std::rand()%100/1000.0, -5+std::rand()%100/1000.0, -3+std::rand()%100/1000.0, 3+std::rand()%100/1000.0},

{1+std::rand()%100/1000.0, 2+std::rand()%100/1000.0, 3+std::rand()%100/1000.0, 4+std::rand()%100/1000.0}};

double mm[][4] = {{2, -3, 1, 1},

{4, 1, -2, 2},

{1, -5, -3, 3},

{1, 2, 3, 4}};

double bad[][4] = {{2, 1.0/2, 1.0/3, 1},

{4, 1.0/3, 1.0/4, 2},

{1, 1.0/4, 1.0/5, 3},

{1, 1.0/5, 1.0/6, 4}};

double \*\*matr = new double\*[SIZE];

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

matr[i] = new double[SIZE];

}

double \*\*dmatr = new double\*[SIZE];

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

dmatr[i] = new double[SIZE];

}

double \*\*rev\_matr = new double\*[SIZE];

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

rev\_matr[i] = new double[SIZE];

}

std::srand(std::time(nullptr));

// double x1=std::rand()%100/1000.0;

// double x2=std::rand()%100/1000.0;

// double x3=std::rand()%100/1000.0;

// double x4=std::rand()%100/1000.0;

// double b[]={10+x1,9+x2,5+x3,6+x4};

// std::cout<<"dB: "<<(fabs(10.0+b[0])+fabs(9.0-b[1])+fabs(5.0-b[2])+fabs(6.0-b[3]))/(30.0)<<"\n";

double b[]={10, 9, 5, 6};

double bb[4];

for(int i=0;i<SIZE;i++)

bb[i]=b[i];

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

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

matr[i][j] = bad[i][j];

dmatr[i][j] = bad[i][j];

rev\_matr[i][j] = bad[i][j];

std::cout<<matr[i][j]<<" ";

}

std::cout<<"\t | "<<b[i]<<'\n';

}

std::cout<<"\n";

/\*

x = Gauss(matr, b);

std::cout<<"GAUSS:\n";

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

std::cout<<"X"<<i+1<<": "<<x[i]<<"\n";

}

\*/

double det = determinant(dmatr,SIZE);

std::cout<<"determinant: "<<det<<'\n';

std::cout<<"\nREVERSE:\n";

x = ReverseMethod(rev\_matr, bb, det);

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

std::cout<<"X"<<i+1<<": "<<x[i]<<"\n";

}

for (int i = 0; i < 4; i++){

for (int j = 0; j < 4; j++){

std::cout<<rev\_matr[i][j]<<' ';

}

std::cout<<'\n';

}

double norm=0.0;

for(int j=0;j<SIZE;j++)

{

double temp=0.0;

for(int i=0;i<SIZE;i++)

temp+=(double)fabs(bad[i][j]);

//temp+=(double)fabs(mm[i][j] - m[i][j]);

//temp+=(double)fabs(mm[i][j] - bad[i][j]);

if(temp>norm)

norm=temp;

}

double norm2=0.0;

for(int j=0;j<SIZE;j++)

{

double temp=0.0;

for(int i=0;i<SIZE;i++)

temp+=(double)fabs(rev\_matr[i][j]);

if(temp>norm2)

norm2=temp;

}

// X1: 2.87822

// X2: -0.906323

// X3: 1.16393

// X4: 0.360656

//||X|| = 5,309129

//||A|| = 11

std::cout<<"dA (otnosit): "<<norm/11<<"\n";

std::cout<<"norm: "<<norm<<"\n";

std::cout<<"condA: "<<norm\*norm2<<"\n";

std::cout<<"norma rev\_matr (absolut nu): "<<norm2<<'\n';

std::cout<<"Delta X (absolut): "<<(fabs(2.87822-x[0])+fabs(0.906323-x[1])+fabs(1.16393-x[2])+fabs( 0.360656-x[3]))<<'\n';

std::cout<<"dX (otnosit): "<<(fabs(2.87822-x[0])+fabs(0.906323-x[1])+fabs(1.16393-x[2])+fabs( 0.360656-x[3]))/(2.87822+0.906323+1.16393+0.360656)<<"\n";

double norm\_x = 0;

double norm\_b = 0;

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

norm\_x += fabs(x[i]);

norm\_b += fabs(b[i]);

}

std::cout<<"Otnosit (esstestv) nu: "<<norm2 \* norm\_b / norm\_x << '\n';

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

delete matr[i];

delete rev\_matr[i];

delete dmatr[i];

}

delete[] matr;

delete [] rev\_matr;

delete [] dmatr;

return 0;

}