Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы численного анализа

**ОТЧЁТ**

к лабораторной работе

на тему

Численное решение систем линейных уравнений методом простых итераций и методом Зейделя

Выполнил: студент группы 153503

Кончик Денис Сергеевич

Проверил: Анисимов Владимир Яковлевич

Минск 2022

**CОДЕРЖАНИЕ**

1. ЦЕЛЬ3

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ4

3. ЗАДАНИЕ8

4. АЛГОРИТМ РЕШЕНИЯ СЛАУ9

5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ11

6. РЕЗУЛЬТАТЫ16

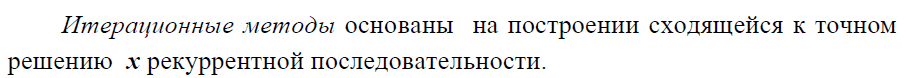
7. ОЦЕНКА21

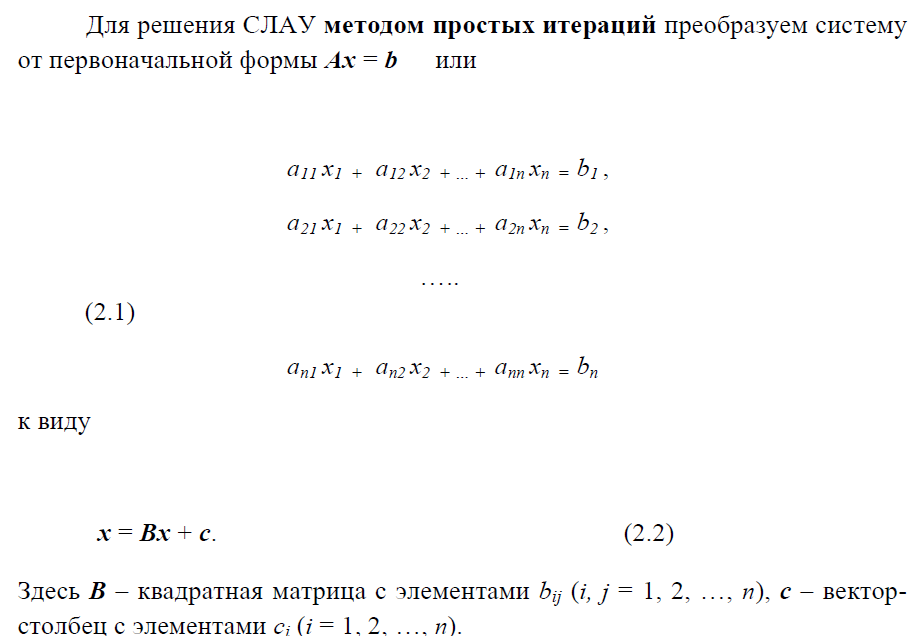
8. ВЫВОД22

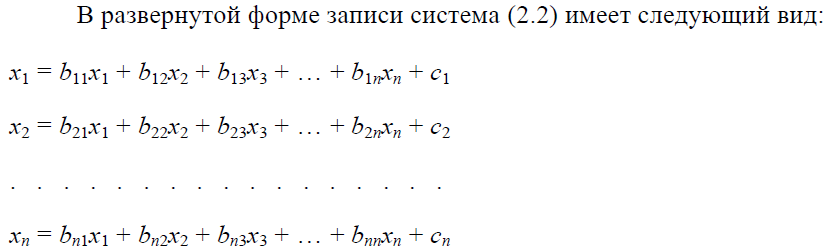
# **ЦЕЛЬ**

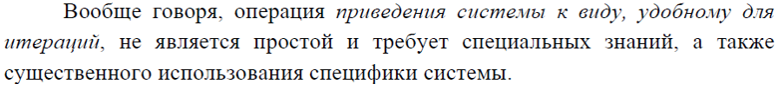
1. Изучить итерационные методы решения СЛАУ (метод простых итераций, метод Зейделя).
2. Составить алгоритм решения СЛАУ указанными методами, применимый для организации вычислений на ЭВМ.
3. Составить программу решения СЛАУ по разработанному алгоритму.
4. Численно решить тестовые примеры и проверить правильность работы программы. Сравнить трудоемкость решения методом простых итерацией и методом Зейделя.

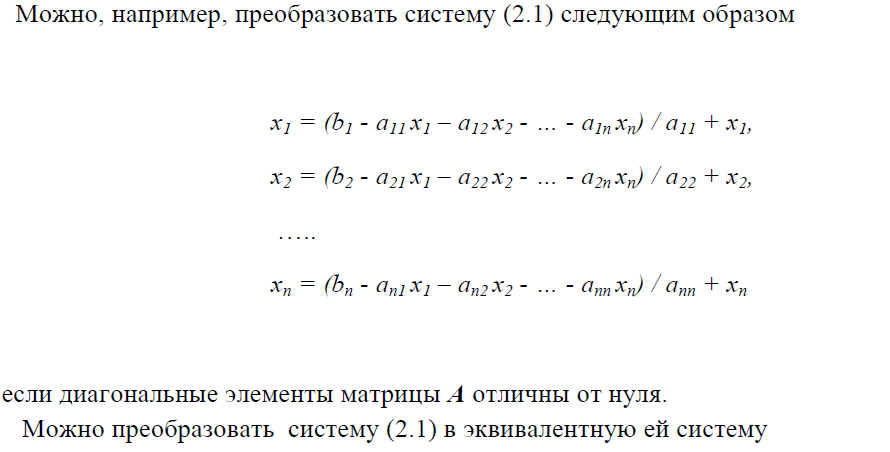
# **КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ**

****

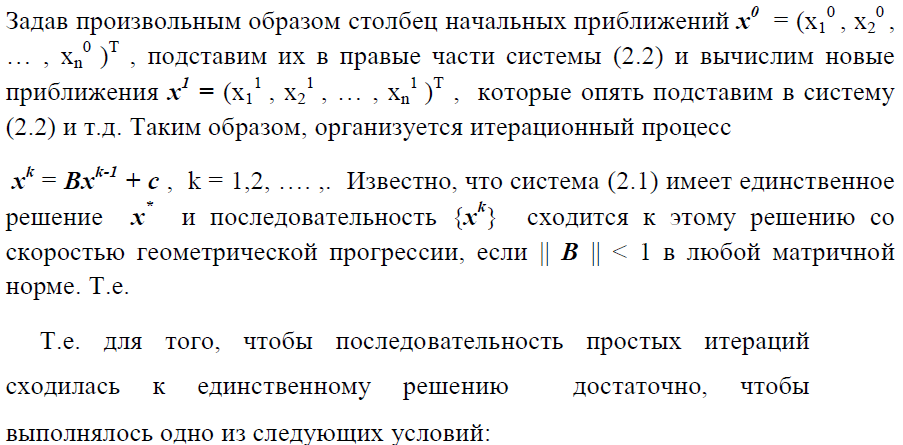
****

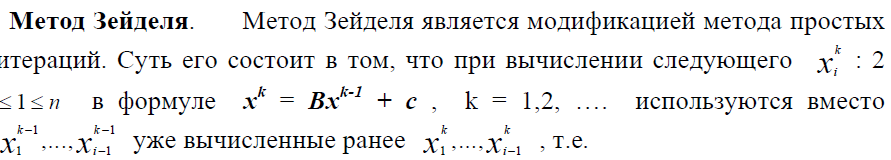
****

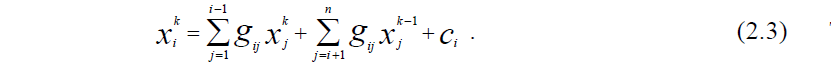
****

****

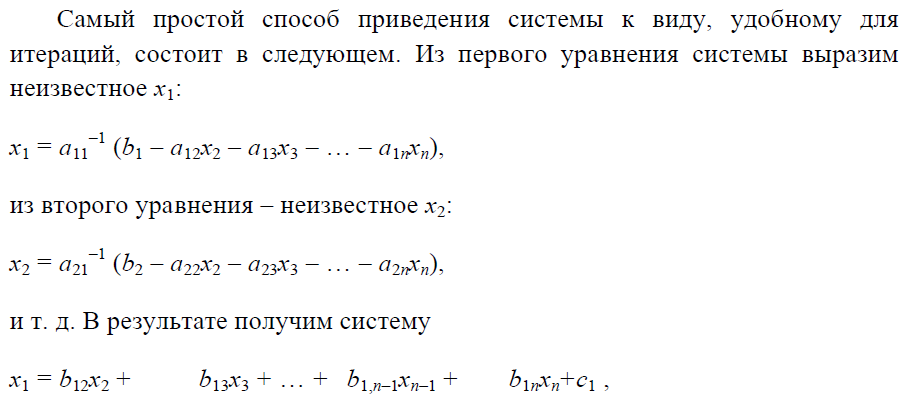
****

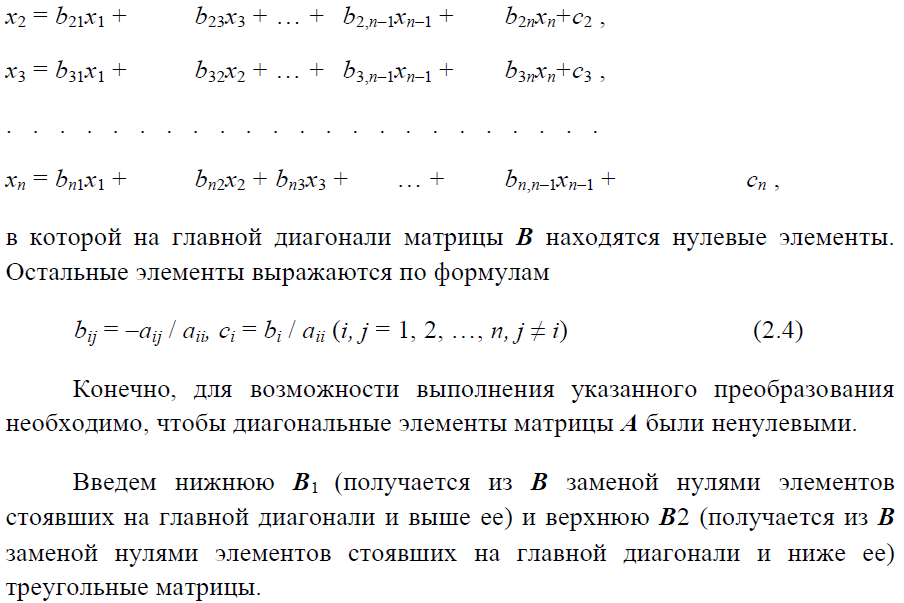
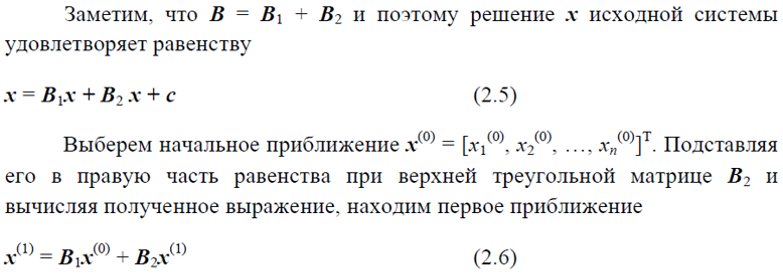
****

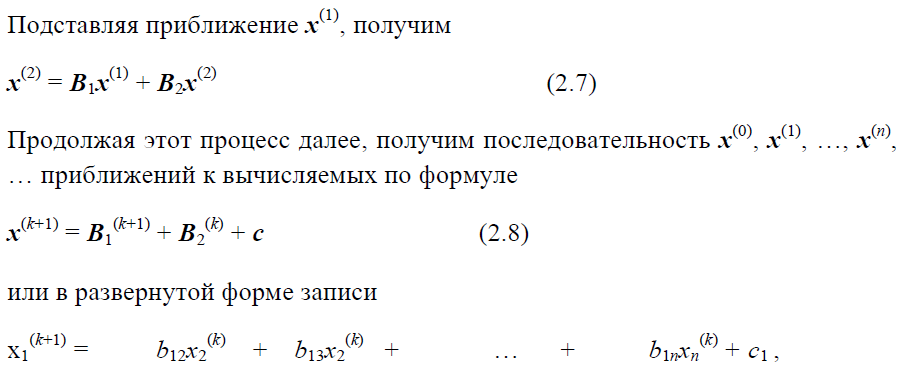
****

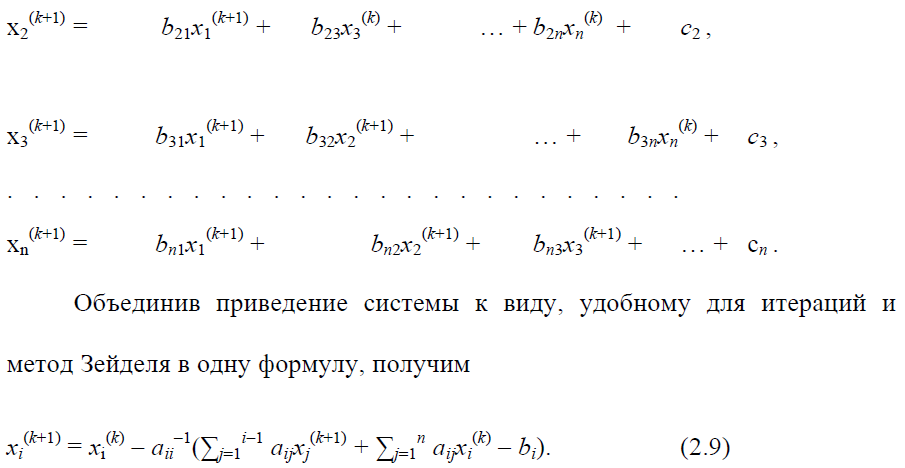
****

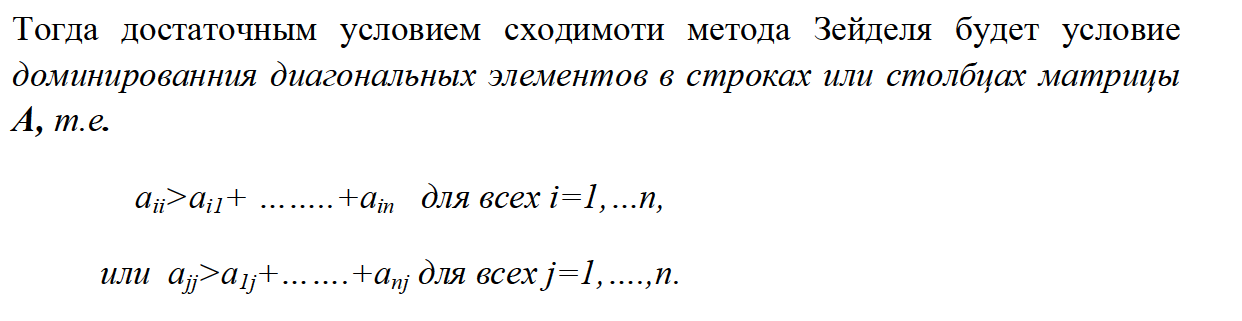
****

****

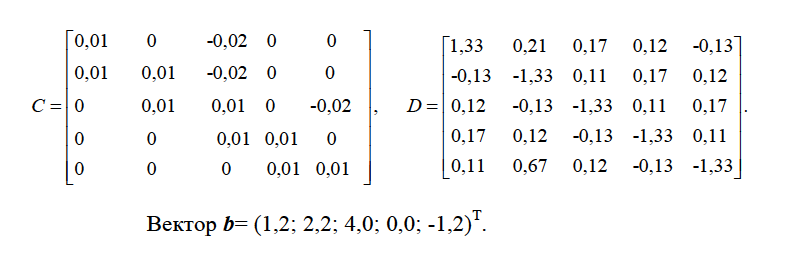
****

****

****



# **ЗАДАНИЕ**

 Вариант 11 (k = 11)

# **АЛГОРИТМ РЕШЕНИЯ СЛАУ**

**1. Метод простых итераций**



**2. Метод Зейделя**

# **ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

**1. Метод простых итераций**

#include "headers.h"

extern double precision;

vector<double> simpleIterations(Matrix<double> A, int& iterations) {

// Размер матрицы

int size = A.getRows();

// Неизвестные на предыдущей итерации

vector <double> prev\_X(size, 0.0);

// Пока не будет достигнута точность

bool stop = false;

while (!stop)

{

++iterations;

// Неизвестные на текущей итерации

vector <double> current\_X(size);

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

{

// x\_i = b\_i

current\_X[i] = A.at(i, size);

// Вычитаем сумму по всем отличным от i-ой неизвестным

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

{

// С прошой итерации

if (j != i)

current\_X[i] -= A.at(i, j) \* prev\_X[j];

}

// x\_i /= b\_i

current\_X[i] /= A.at(i, i);

}

// Максимальная погрешность

long double max\_error = 0.0;

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

double new\_max\_error = abs(current\_X[i] - prev\_X[i]);

max\_error = new\_max\_error > max\_error ? new\_max\_error : max\_error;

}

// Дотигнута ли точность

if (max\_error < precision)

stop = true;

// Переход к следующей итерации

prev\_X = current\_X;

}

return prev\_X;

}

**2. Метод Зейделя**

vector<double> seidelMethod(Matrix<double> A, int& iterations) {

// Размер матрицы

int size = A.getRows();

// Неизвестные на предыдущей итерации

vector <double> prev\_X(size, 0.0);

// Пока не будет достигнута точность

bool stop = false;

while (!stop)

{

++iterations;

// Неизвестные на текущей итерации

vector <double> current\_X(size);

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

{

// x\_i = b\_i

current\_X[i] = A.at(i,size);

// Вычитаем сумму по всем отличным от i-ой неизвестным

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

{

// С этой итерации

if (j < i)

current\_X[i] -= A.at(i, j) \* current\_X[j];

// С прошой итерации

if (j > i)

current\_X[i] -= A.at(i, j) \* prev\_X[j];

}

// x\_i /= b\_i

current\_X[i] /= A.at(i,i);

}

// Максимальная погрешность

long double max\_error = 0.0;

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

double new\_max\_error = abs(current\_X[i] - prev\_X[i]);

max\_error = new\_max\_error > max\_error ? new\_max\_error : max\_error;

}

// Дотигнута ли точность

if (max\_error < precision)

stop = true;

// Переход к следующей итерации

prev\_X = current\_X;

}

return prev\_X;

}

**3. Функция main**

#include <iostream>

#include <iomanip>

#include "Matrix.h"

#include "headers.h"

using namespace std;

int k = 11;

double precision = 0.0001;

int main()

{

setlocale(LC\_ALL, "Ru");

Matrix<double> C(create\_C(), 5, 5);

Matrix<double> D(create\_D(), 5, 5);

Matrix<double> A = k \* C + D;

vector<double> d = { 1.2, 2.2, 4.0, 0, -1.2 };

Matrix<double> A\_advanced = A + d;

A.print("Исходная матрица");

A\_advanced.print("Расширенная матрица");

cout << "\nРанг исходной матрицы: " << A.rank() <<

"\nРанг расширенной матрицы: " << A\_advanced.rank() << "\n";

if (A.rank() != A\_advanced.rank()) {

cout << "\nРанг исходной матрицы не равен рангу расширенной матрицы." <<

"\nСЛАУ не имеет решений.";

}

else if (A.rank() < A.getRows()) {

cout << "\nРанги матриц равны и меньше числа неизвестных системы." <<

"\nСЛАУ имеет бесконечное множество решений.";

}

else if (!A.isDiagonalDominance()) {

cout << "\nИсходная матрица не обладает свойством диагонального преобладания.\n"

<< "Решение методом [простых итераций / Зейделя] невозможно.";

}

else {

int iterations = 0;

vector<double> roots = simpleIterations(A\_advanced, iterations);

print\_roots(roots, "Корни (метод простых итераций):");

cout << "Количество итераций: " << iterations << "\n";

//check\_roots(A, roots, "Проверка. \nb1..b5 при подстановке корней:");

iterations = 0;

roots = seidelMethod(A\_advanced, iterations);

print\_roots(roots, "Корни (метод Зейделя):");

cout << "Количество итераций: " << iterations << "\n";

//check\_roots(A, roots, "Проверка. \nb1..b5 при подстановке корней:");

}

}

**4. Остальные функции**

double\*\* create\_C() {

double\*\* C = new double\* [5];

C[0] = new double[5]{ 0.0100, 0, -0.0200, 0, 0 };

C[1] = new double[5]{ 0.0100, 0.0100, -0.0200, 0, 0 };

C[2] = new double[5]{ 0, 0.0100, 0.0100, 0, -0.0200 };

C[3] = new double[5]{ 0, 0, 0.0100, 0.0100, 0 };

C[4] = new double[5]{ 0, 0, 0, 0.0100, 0.0100 };

return C;

}

double\*\* create\_D() {

double\*\* D = new double\* [5];

D[0] = new double[5]{ 1.3300, 0.2100, 0.1700, 0.1200, -0.1300 };

D[1] = new double[5]{ -0.1300, -1.3300, 0.1100, 0.1700, 0.1200 };

D[2] = new double[5]{ 0.1200, -0.1300, -1.3300, 0.1100, 0.1700 };

D[3] = new double[5]{ 0.1700, 0.1200, -0.1300, -1.3300, 0.1100 };

D[4] = new double[5]{ 0.1100, 0.6700, 0.1200, -0.1300, -1.3300 };

return D;

}

void print\_roots(const vector<double>& roots, string text, int w, int precision) {

if (text == "\n")

cout << "\n\n";

else

cout << "\n" << text << "\n";

for (const auto& i : roots)

cout << fixed << setw(w) << setprecision(precision) << i << "\n";

}

bool isDiagonalDominance() {

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

double sum = 0;

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

if (i != j) sum += fabs(this->at(i, j));

if (fabs(this->at(i, i)) < sum)

return false;

}

return true;

}

int rank() {

Matrix<double> matrix = \*this;

const double EPS = 1E-9;

int n = rows;

int m = columns;

int rank = max(n, m);

vector<char> line\_used(n);

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

int j;

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

if (!line\_used[j] && abs(matrix.at(j, i)) > EPS) {

break;

}

}

if (j == n) {

--rank;

}

else {

line\_used[j] = true;

for (int p = i + 1; p < m; ++p)

matrix.at(j,p) /= matrix.at(j,i);

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

if (k != j && abs(matrix.at(k,i)) > EPS)

for (int p = i + 1; p < m; ++p)

matrix.at(k,p)-=matrix.at(j,p)\*matrix.at(k,i);

}

}

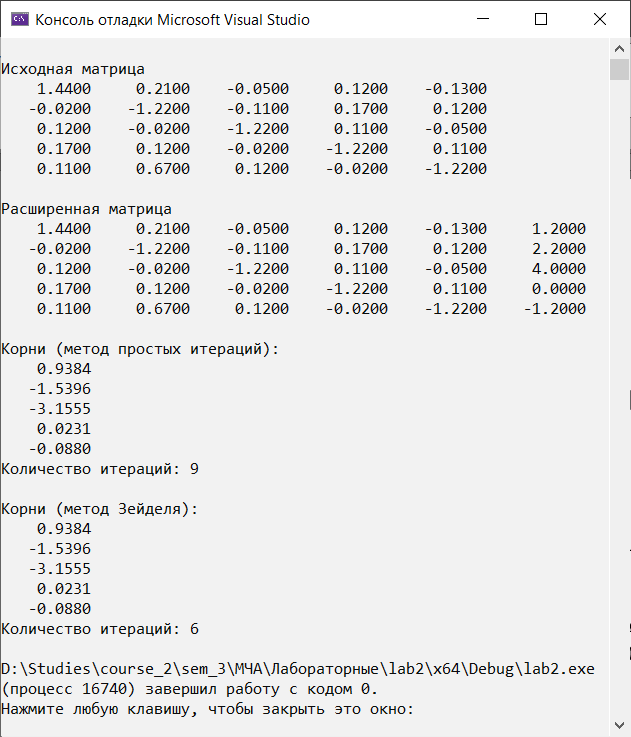
return rank;

}

# **ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

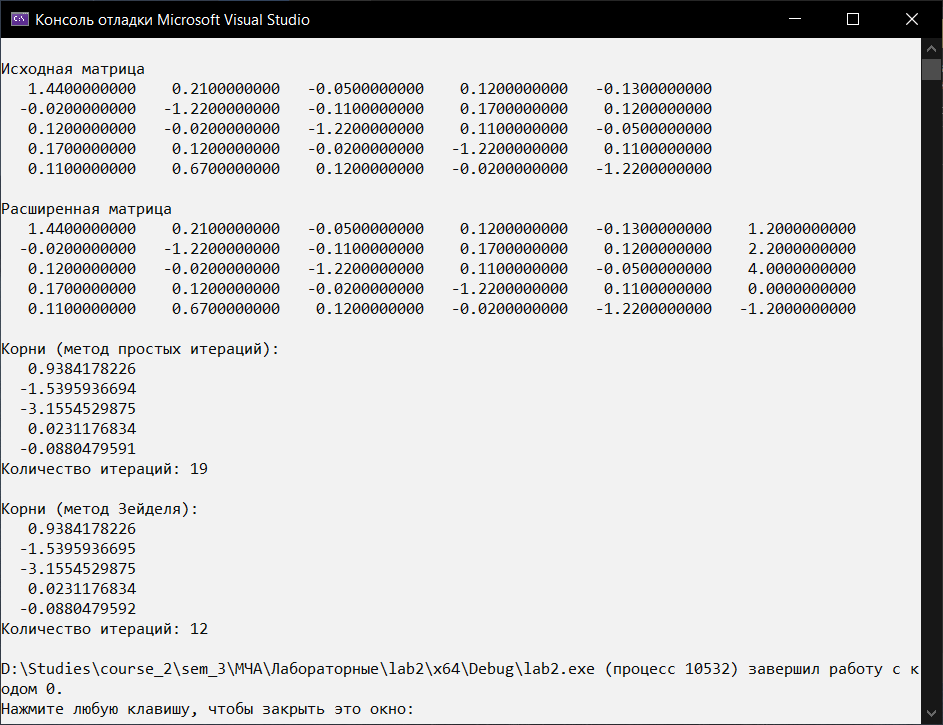
**Тестовый пример 1.**

Матрица А, полученная в результате вычисления A=11C+D и вектор свободных членов (по условию) b=(1.2; 2.2; 4.0; 0.0; -1.2)

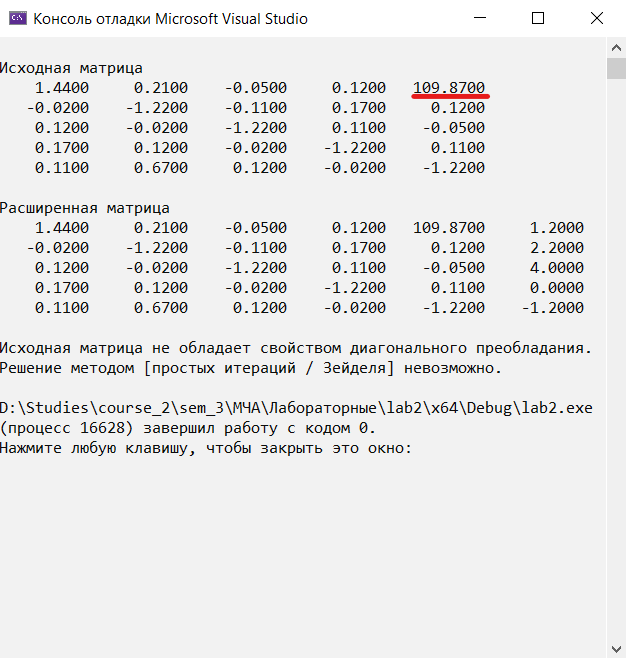


**Тестовый пример 2.**

Матрица А, полученная в результате вычисления A=11C+D и вектор свободных членов (по условию) b=(1.2; 2.2; 4.0; 0.0; -1.2).

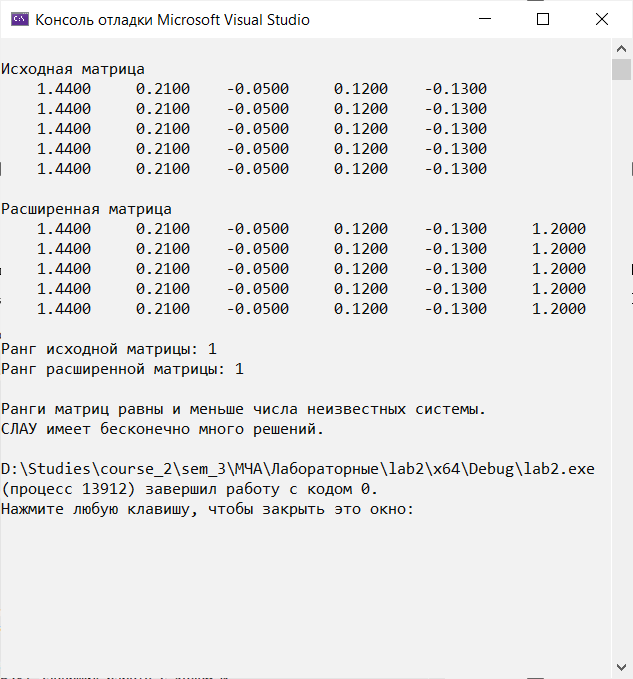
Точность eps=1e-10

**Тестовый пример 3.**

Исходная матрица не обладает диагональным преобладанием.

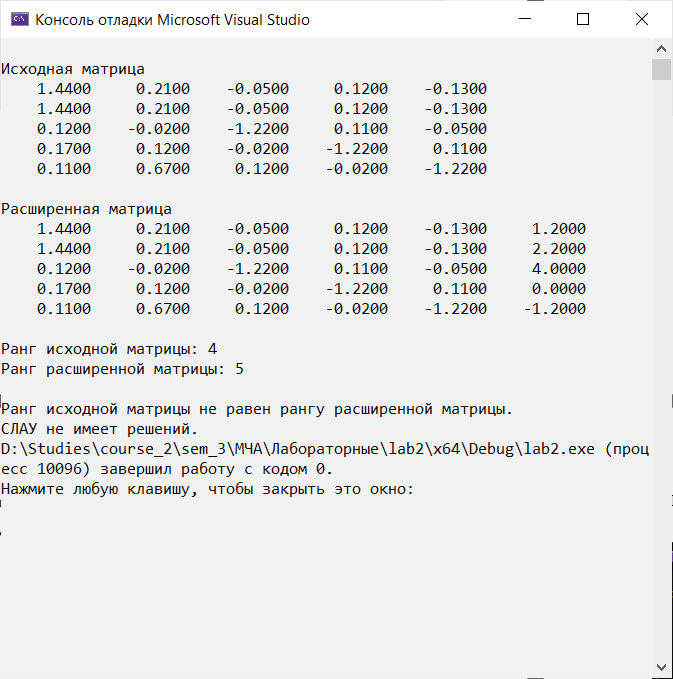
**Тестовый пример 4.**

СЛАУ имеет бесконечно много решений.



**Тестовый пример 5.**

СЛАУ не имеет решений.



# **ОЦЕНКА**

- значение, полученное по заданию с точностью .

– значение, полученное с точностью .

# **ВЫВОД**

В ходе лабораторной работы был применены итерационные методы решения СЛАУ в двух вариантах: метод простых итераций и метод Зейделя. Были разработаны алгоритмы решения СЛАУ указанными методами, составлена программа по разработанным алгоритмам, решены тестовые примеры.

На основании тестовых примеров можно сделать следующие выводы:

1. Метод простых итераций более ресурснозатратный из-за большего количества проводимых итераций.
2. Оба метода позволяют получить решение с заданной точностью, причем корни в обоих методах могут отличаться в пределах заданной погрешности.
3. Данные методы не всегда могут решить СЛАУ.