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 253504

Фроленко Кирилл Юрьевич

Проверил: Анисимов Владимир Яковлевич

Минск 2023

Оглавление

[**Цели выполнения задания** 3](#_30j0zll)

[**Краткие теоретические сведения** 4](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.1fob9te)

[**Задание** 8](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.3znysh7)

**Алгоритм решения**9

[**Программная реализация** 1](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.2et92p0)0

[**Полученные результаты** 1](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.tyjcwt)5

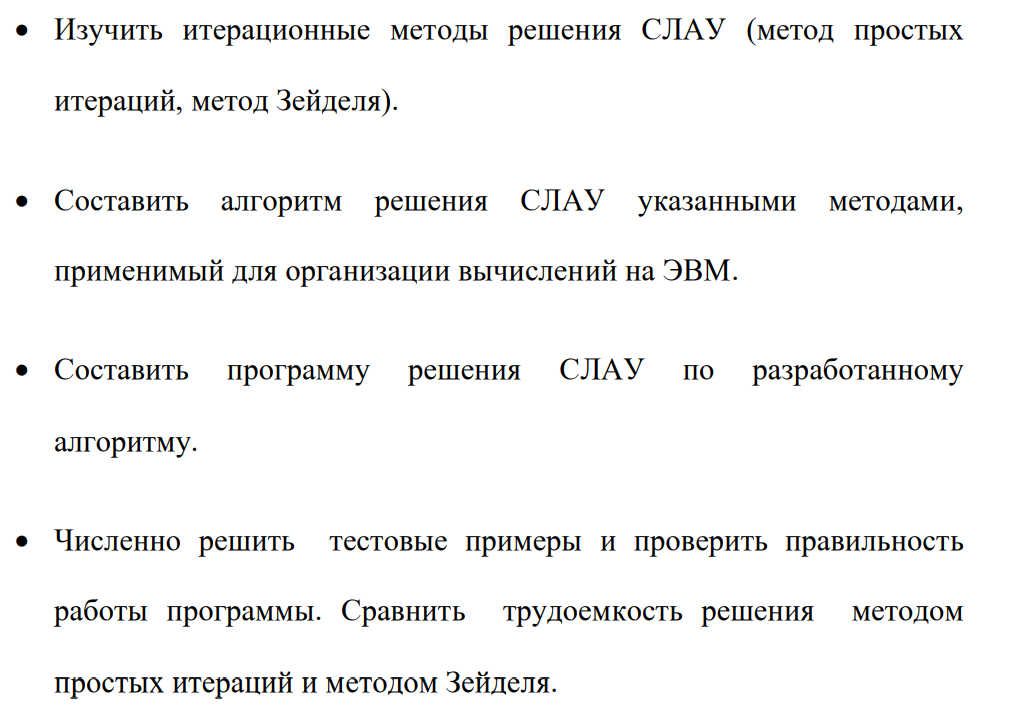
[**Оценка** 1](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.3dy6vkm)6

**Тесты** 17

[**Выводы** 1](https://docs.google.com/document/d/1EWdc2b-UMM5lmh1NDBJQCQ1GzPigp2mZ4cvigqo9fAY/edit#heading=h.1t3h5sf)8

Вариант 13

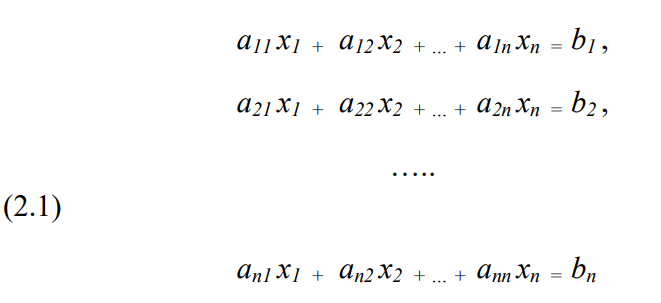
**Цель выполнения задания:**

****

**Краткие теоретические сведения**

*Итерационные методы* основаны на построении сходящейся к точному решению х рекуррентной последовательности.

Для решения СЛАУ **методом простых итераций** преобразуем систему от первоначальной формы **Ax = b** или

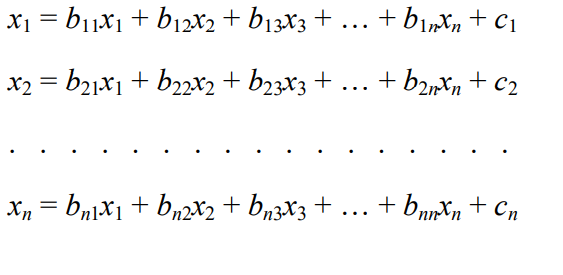


к виду

**x = Bx + c** (2.2)

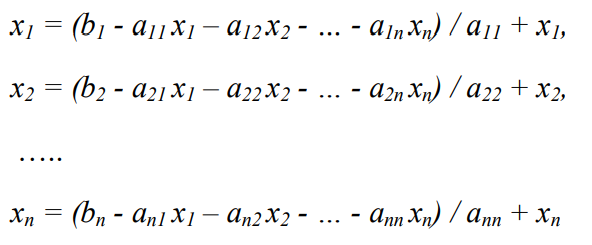
Здесь ***B*** - квадратная матрица с элементами bij (*i, j* = 1, 2. ..., *n*), ***с*** – вектор столбец с элементами сi (*i =* 1,2,..., *n*).

В развернутой форме записи система (2.2) имеет следующий вид:



Вообще говоря, операция *приведения системы* к виду, *удобному* *для итераций*, не является простой и требует специальных знаний, а также существенного использования специфики системы.

Можно, например, преобразовать систему (2.1) следующим образом



если диагональные элементы матрицы ***А*** отличны от нуля.

Можно преобразовать систему (2.1) в эквивалентную ей систему

***x = (E-A)x+b.***

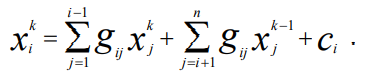
Задав произвольным образом столбец начальных приближений x0 = (x10, x20, ... . xn0 )Т . подставим их в правые части системы (2.2) и вычислим новые приближения x1 = (x11, x21, ... . xn1), которые опять подставим в систему (2.2) и т.д. Таким образом, организуется итерационный процесс xk = Bxk-1 + c, k = 1,2 ..., . Известно, что система (2.1) имеет единственное решение x\* и последовательность {xk} сходится к этому решению со скоростью геометрической прогрессии, если || В || < 1 в любой матричной норме.

Т.e. для того, чтобы последовательность простых итераций сходилась к единственному решению достаточно, чтобы выполнялось одно из следующих условий:

1. ) < 1;
2. ) < 1;

**Метод Зейделя**

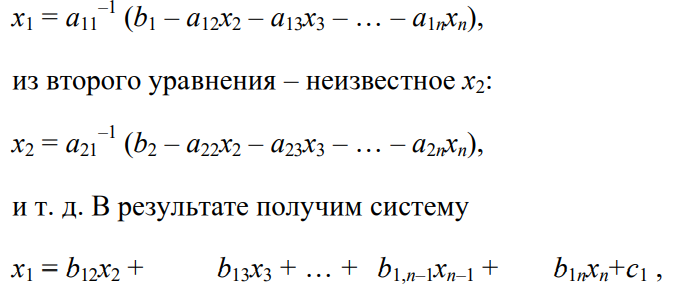
Метод Зейделя является модификацией метода простых итераций. Суть его состоит в том, что при вычислении следующего хik : 2 в формуле xk = Bxk-1 + c , k =1,2, ... используются вместо х1k-1 ,…, уже вычисленные ранее хik,..., т.e.

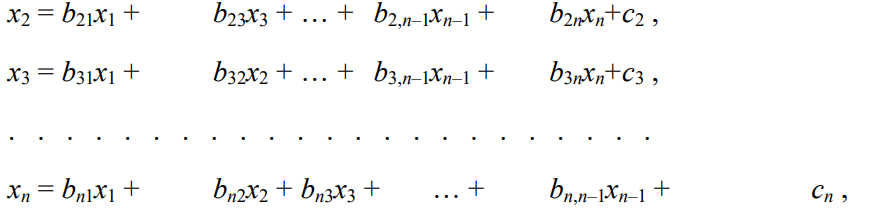
 (2.3)

Такое усовершенствование позволяет ускорить сходимость итераций почти в два раза. Кроме того, данный метод может быть реализован на ЭВМ без привлечения дополнительного массива, т.к. полученное новое сразу засыпается на место старого.

Схема алгоритма аналогична схеме метода простых итераций.

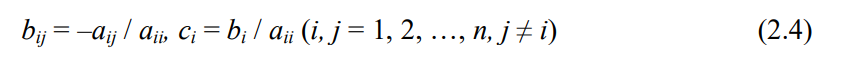
Самый простой способ приведения системы к виду, удобному для итераций, состоит в следующем. Из первого уравнения системы выразим неизвестное х1 :





в которой на главной диагонали матрицы ***B*** находятся нулевые элементы.

Остальные элементы выражаются по формулам



Конечно, для возможности выполнения указанного преобразования необходимо, чтобы диагональные элементы матрицы ***А*** были ненулевыми.

Введем нижнюю ***В1*** (получается из ***В*** заменой нулями элементов стоявших на главной диагонали и выше ее) и верхнюю ***В2*** (получается из ***В*** заменой нулями элементов стоявших на главной диагонали и ниже ее) треугольные матрицы.

Заметим, что ***B = B1 + B2*** и поэтому решение ***х*** исходной системы удовлетворяет равенству

***x = B1x + B2x+c***  (2.5)

Выберем начальное приближение x(0) = [x1(0), x2(0), … , xn(0)]T. Подставляя его в правую часть равенства при верхней треугольной матрице ***B2*** и вычисляя полученное выражение, находим первое приближение

***x(1) = B1x(0) + B2x(1)***  (2.6)

Подставляя приближение **x(**1), получим

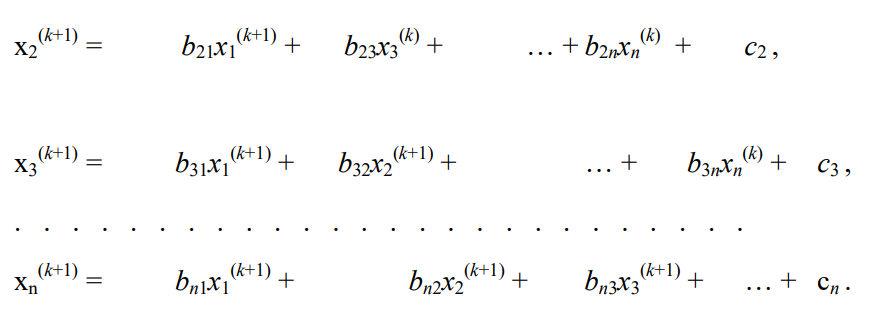
***X(2) = B1x(1) + B2x(2)***  (2.7)

Продолжая этот процесс далее, получим последовательность x(0) ,x(1), … , x(n) , … приближений к вычисляемых по формуле

***X(k+1) = B1(k+1) + B2(k) + c*** (2.8)

или в развернутой форме записи



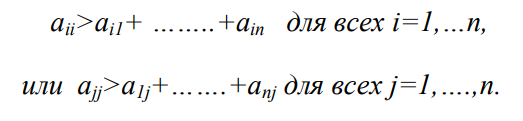


Объединив приведение системы к виду, удобному для итераций и метод Зейделя в одну формулу, получим



Тогда достаточным условием сходимости метода Зейделя будет условие *доминирования диагональных элементов в строках или столбцах матрицы*

***A, т.e.***



Методы простой итерации и Зейделя сходятся примерно так же, как геометрическая прогрессия со знаменателем || ***В*** || .

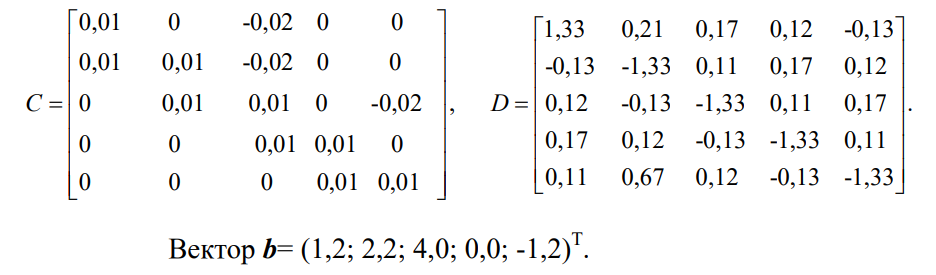
**ЗАДАНИЕ**

Методом простых итераций и методом Зейделя найти с

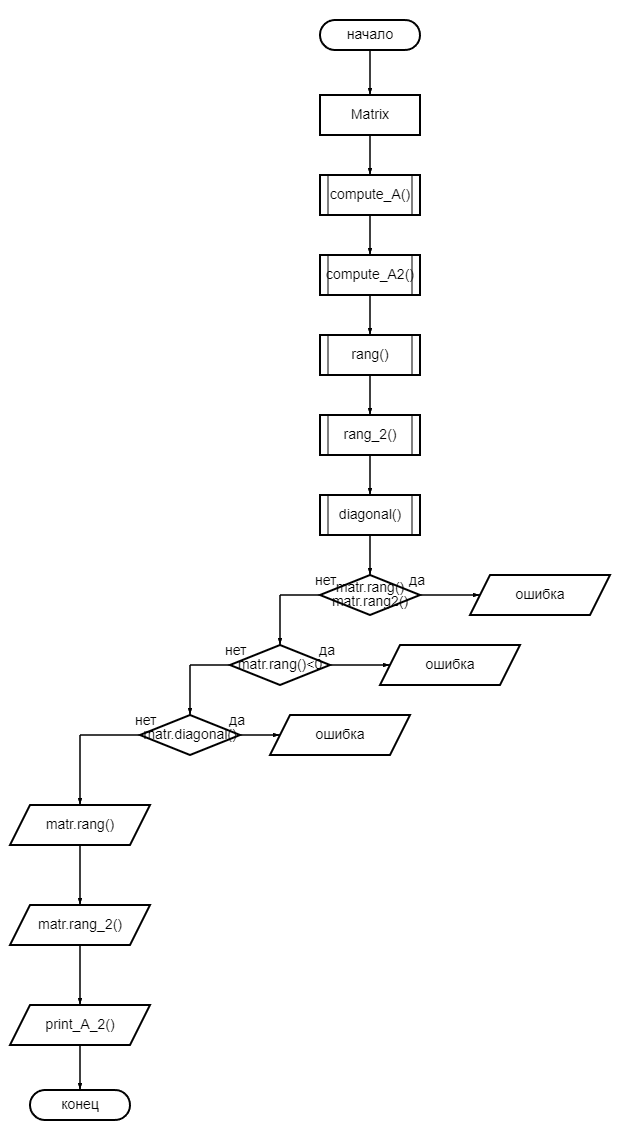
точностью 0,0001 численное решение системы ***Аx=b***,

где A = kC + D, A - исходная матрица для расчёта, k - номер варианта

**(0-15), матрицы C, D и вектор свободных членов ь задаются ниже.**

****

**Алгоритм выполнения задания**



**Программная реализация**

#include <bits/stdc++.h>

class Matrix {

private:

int V;

double A[5][5];

double A\_2[5][6];

double C[5][5]

{

{1.0 / 100, 0, -2.0 / 100, 0, 0},

{1.0 / 100, 1.0 / 100, -2.0 / 100, 0, 0},

{0, 1.0 / 100, 1.0 / 100, 0, -2.0 / 100},

{0, 0, 1.0 / 100, 1.0 / 100, 0},

{0, 0, 0, 1.0 / 100, 1.0 / 100}

};

double D[5][5]

{

{133.0 / 100, 21.0 / 100, 17.0 / 100, 12.0 / 100, -13.0 / 100},

{-13.0 / 100, -133.0 / 100, 11.0 / 100, 17.0 / 100, 12.0 / 100},

{12.0 / 100, -13.0 / 100, -133.0 / 100, 11.0 / 100, 17.0 / 100},

{17.0 / 100, 12.0 / 100, -13.0 / 100, -133.0 / 100, 11.0 / 100},

{11.0 / 100, 67.0 / 100, 12.0 / 100, -13.0 / 100, -133.0 / 100}

};

double b[5]{ 12.0 / 10, 22.0 / 10, 40.0 / 10, 0, -12.0 / 10 };

double roots[5]{ 0, 0, 0, 0, 0 };

public:

Matrix(const int V) : V(V) {

std::cout << std::fixed << std::setprecision(4);

compute\_A();

compute\_A\_2();

}

void compute\_A() {

for (int i = 0; i < 5; i++)

for (int j = 0; j < 5; j++)

A[i][j] = V \* C[i][j] + D[i][j];

}

void compute\_A\_2() {

for (int i = 0; i < 5; i++)

for (int j = 0; j < 5; j++)

A\_2[i][j] = A[i][j];

A\_2[0][5] = 12.0 / 10;

A\_2[1][5] = 22.0 / 10;

A\_2[2][5] = 40.0 / 10;

A\_2[3][5] = 0;

A\_2[4][5] = -12.0 / 10;

}

void print\_A() {

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++)

std::cout << std::setw(10) << A[i][j];

std::cout << std::setw(5) << '|' << std::setw(5) << b[i] << std::endl;

}

}

void print\_A\_2() {

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++)

std::cout << std::setw(10) << A\_2[i][j];

std::cout << std::setw(5) << '|' << std::setw(5) << A\_2[i][5] << std::endl;

}

}

std::vector<double> simpleIterations(int& iterations) {

// Размер матрицы

int size = 5;

// Неизвестные на предыдущей итерации

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

// Пока не будет достигнута точность

bool stop = false;

while (!stop) {

++iterations;

// Неизвестные на текущей итерации

std::vector <double> current\_X(size);

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

// x\_i = b\_i

current\_X[i] = A\_2[i][size];

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

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

// С прошлой итерации

if (j != i)

current\_X[i] -= A\_2[i][j] \* prev\_X[j];

// x\_i /= b\_i

current\_X[i] /= A\_2[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 < 0.0001)

stop = true;

// Переход к следующей итерации

prev\_X = current\_X;

}

return prev\_X;

}

std::vector<double> seidelMethod(int& iterations) {

// Размер матрицы

int size = 5;

// Неизвестные на предыдущей итерации

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

// Пока не будет достигнута точность

bool stop = false;

while (!stop) {

++iterations;

// Неизвестные на текущей итерации

std::vector <double> current\_X(size);

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

// x\_i = b\_i

current\_X[i] = A\_2[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\_2[i][j] \* current\_X[j];

// С прошой итерации

if (j > i)

current\_X[i] -= A\_2[i][j] \* prev\_X[j];

}

// x\_i /= b\_i

current\_X[i] /= A\_2[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 < 0.0001)

stop = true;

// Переход к следующей итерации

prev\_X = current\_X;

}

return prev\_X;

}

int rang() {

int rang = 5;

//std::cout << "\nf\_rang = " << rang << std::endl;

double sum = 0;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

sum += A[i][j];

//std::cout << "sum = " << sum << std::endl;

if (sum == 0)

rang = rang - 1;

//std::cout << "rang " << i << " = " << rang << std::endl;

sum = 0;

}

//std::cout << "end rang = " << rang << std::endl;

return rang;

}

int rang\_2() {

int rang = 6;

//std::cout << "\nf\_rang = " << rang << std::endl;

double sum = 0;

for (int i = 0; i < 6; i++)

{

for (int j = 0; j < 6; j++)

sum += A\_2[i][j];

//std::cout << "sum = " << sum << std::endl;

if (sum == 0)

rang = rang - 1;

//std::cout << "rang " << i << " = " << rang << std::endl;

sum = 0;

}

return rang;

}

bool diagonal() {

int i, j, k = 1;

double sum;

for (i = 0; i < 5; i++)

{

sum = 0;

for (j = 0; j < 5; j++)

sum += A[i][j];

sum -= A[i][i];

if (sum < A[i][i])

return 0;

}

return (k == 1);

}

};

int main()

{

system("chcp 65001");

Matrix matr(13);

if (matr.rang() != matr.rang\_2()) {

std::cout << "Ранг исходной матрицы не равен рангу расширенной матрицы.\nСЛАУ не имеет решений.";

return 0;

}

if (matr.rang() < 5) {

std::cout << "Ранги матрицы равны или меньше числа неизвестных системы.\nСЛАУ имеет бесконечное множество решений.";

return 0;

}

if (matr.diagonal()) {

std::cout << "Исходная матрица не обладает свойством диагонального преобладания.\nРешение методом итераций/Зейделя невозможен.";

return 0;

}

std::cout << matr.rang() << ", " << matr.rang\_2();

std::cout << "\t\t\tМетод итераций:" << std::endl << std::endl;

std::cout << "\tИсходная матрица:" << std::endl;

matr.print\_A\_2();

int iterations = 0;

std::vector<double> vect = matr.seidelMethod(iterations);

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

int t = 0;

for (auto i : vect) {

std::cout << ++t << ". " << i << "\n";

}

std::cout << "\n\n\t\t\tМетод Зейделя" << std::endl << std::endl;

std::cout << "\tИсходная матрица:" << std::endl;

matr.compute\_A();

matr.compute\_A\_2();

matr.print\_A\_2();

iterations = 0;

vect = matr.simpleIterations(iterations);

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

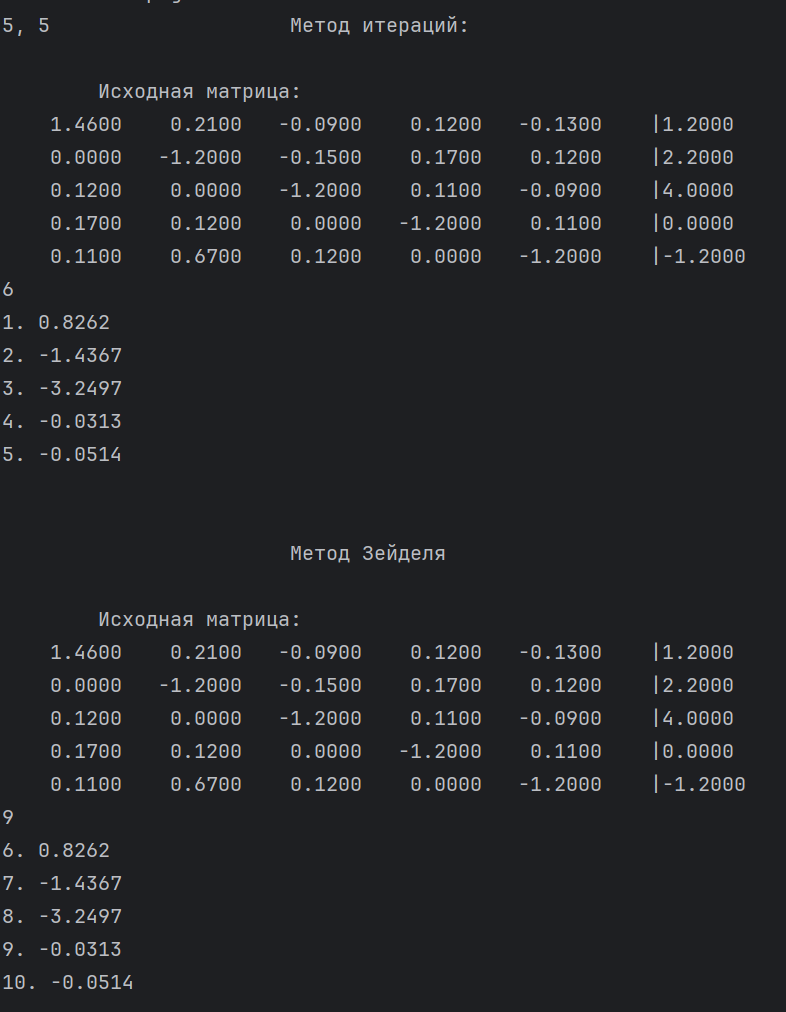
for (auto i : vect) {

std::cout << ++t << ". " << i << "\n";

}

}

**Полученные результаты программы**

****

**Оценка:**

Таким образом, мы видим, что результаты, полученные в онлайн калькуляторе, практически идентичны нашим. Различия лишь в погрешности вычисления нашей программы и онлайн-калькулятора

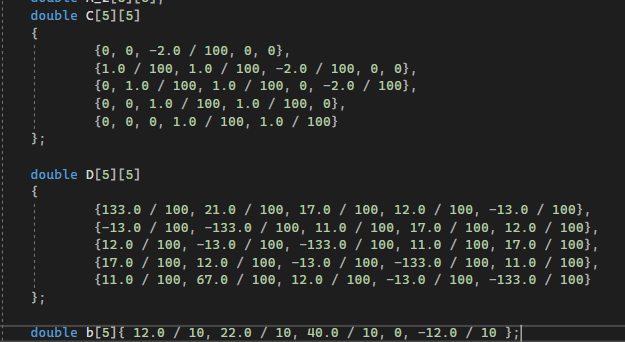
Рассчитаем относительную погрешность:

-3.2497 – (-3.24972487774) = 0,00002487774 0,0000315708

**Тесты**

**Тест 1:**

Матрицы С и D:

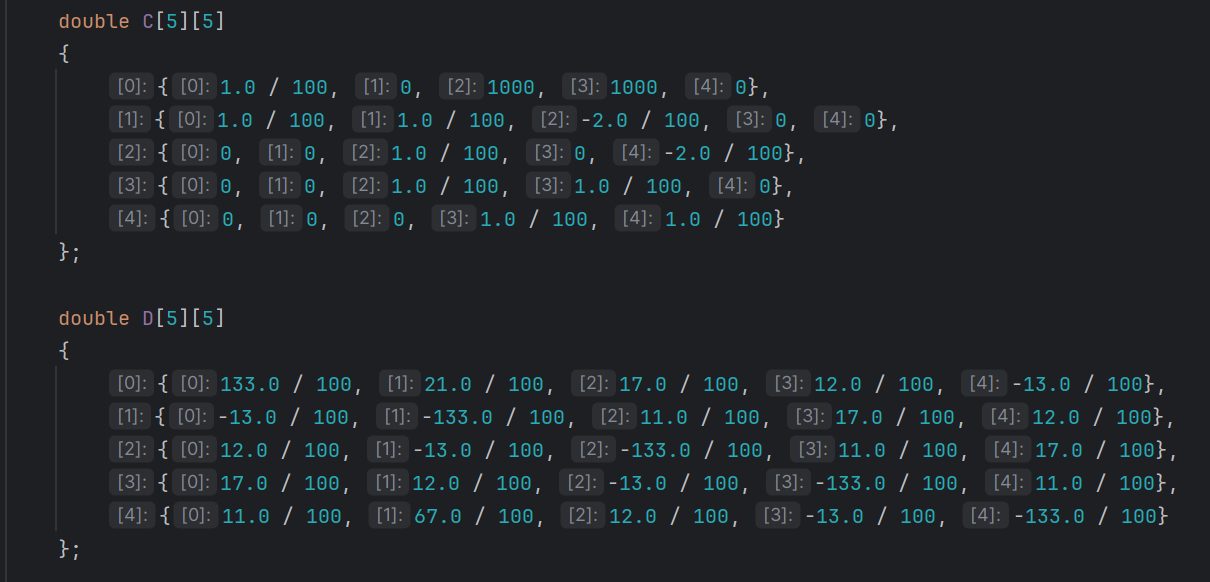
****

Результат:

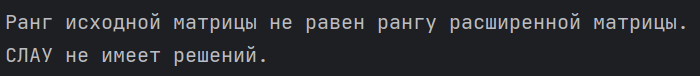


**Тест 2:**

Матрицы С и D:

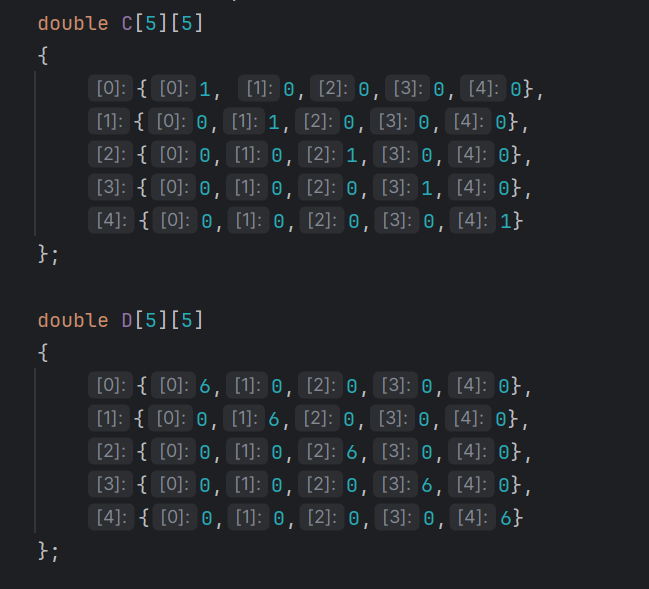


Результат:



**Тест 3:**

Матрицы С и D:



Результат:



**Выводы:**

В ходе выполнения данной лабораторной работы были изучены и применены итерационные методы решения СЛАУ, в частности метод простых итераций и метод Зейделя для решения системы линейных уравнений; были созданы соответствующие алгоритмы, применимые для организации вычислений на ЭВМ, и реализации программ на языке С++ для решения поставленной задачи; для проверки правильности работы программы были выполнены тестовые примеры и проведена оценка.

Сравнение трудоемкости метода простых итераций и метода Зейделя с точки зрения используемой памяти и процессорного времени может зависеть от ряда факторов, таких как размер системы уравнений, структура матрицы A, точность вычислений и специфика реализации алгоритмов на конкретном программном и аппаратном обеспечении. В целом, можно выделить следующие общие соображения:

1. **Память**:
   * **Метод простых итераций**: Требует дополнительной памяти для хранения временных переменных x^{(k+1)} и x^{(k)}. Это означает, что потребление памяти прямо пропорционально размеру вектора неизвестных x. Если система имеет большой размер, это может потребовать значительного объема дополнительной памяти.
   * **Метод Зейделя**: Обновление значений неизвестных происходит "на месте" внутри одного вектора, и поэтому требуется меньше дополнительной памяти, чем в методе простых итераций. В этом методе также потребление памяти прямо пропорционально размеру вектора неизвестных, но он может быть более эффективным с точки зрения использования памяти.
2. **Процессорное время**:
   * **Метод простых итераций**: Этот метод обычно требует большего числа итераций для сходимости, особенно если матрица A плохо обусловлена. Каждая итерация включает в себя умножение матрицы A на вектор x, что является вычислительно затратной операцией. Процессорное время может быть высоким, особенно для больших систем.
   * **Метод Зейделя**: Этот метод часто сходится быстрее, так как обновление значений неизвестных происходит более локально. Кроме того, в методе Зейделя можно использовать данные из самых последних итераций, что может ускорить сходимость. Это может привести к меньшему процессорному времени, особенно в случае систем с хорошей условной численностью.

В итоге, метод Зейделя обычно имеет небольшое преимущество с точки зрения использования памяти и может быть более эффективным с точки зрения процессорного времени, особенно для больших и плохо обусловленных систем уравнений.