**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования  
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ имени Н.Э.БАУМАНА  
(национальный исследовательский университет)»**

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

**Лабораторная работа № 1**

“Решение СЛАУ. Метод прогонки”

по дисциплине «Численные методы»

Работу выполнил

студент группы ИУ9-62Б

Сербин Денис

Москва, 2022

# **1. Цель работы**

Анализ метода прогонки и решение СЛАУ с трёхдиагональной матрицей.

**2. Постановка задачи**

Дано:

𝐴𝑥 = 𝑑, где 𝑥, 𝑑 ∈ R𝑛, 𝐴 - трехдиагональная матрица.

Найти: Решение СЛАУ с помощью метода прогонки.

Пример:

, .

Уравнение 𝐴𝑥 = 𝑑:

**3. Описание алгоритма**

Элементы матрицы a – элементы под диагональю, b – элементы на диагонали, c – элементы над диагональю:

Получим СЛАУ:

Заменим на

, где , , i = 2..n

Необходимое условие:

Достаточное условие:

1. i = 2..n-1

**4. Оценка погрешности для решения СЛАУ**

, получим: () = ().

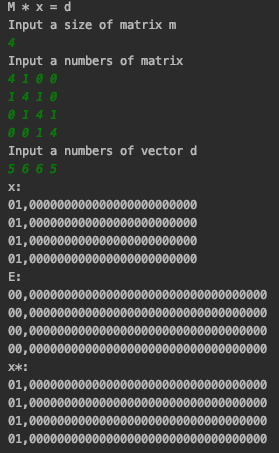
При векторе ошибки () и r = () получим: A. Исходный вектор -

**5. Реализация**

**public class** TriagMatrix {  
  
 **private static int** *size* = 0;  
  
 **private static** ArrayList<ArrayList<Double>> *matr* = **new** ArrayList<>();  
 **private static** ArrayList<Double> *a* = **new** ArrayList<>();  
 **private static** ArrayList<Double> *b* = **new** ArrayList<>();  
 **private static** ArrayList<Double> *c* = **new** ArrayList<>();  
  
 **private static** ArrayList<Double> *d* = **new** ArrayList<>();  
  
 **private static** ArrayList<Double> *alpha* = **new** ArrayList<>();  
 **private static** ArrayList<Double> *beta* = **new** ArrayList<>();  
  
 **private static** ArrayList<Double> *x* = **new** ArrayList<>();  
  
 **public void** initMatr(Scanner in) {  
 System.*out*.println("Input a numbers of matrix");  
 **for** (**int** i = 0; i < *size*; i++) {  
 ArrayList<Double> str = **new** ArrayList<>();  
 **for** (**int** j = 0; j < *size*; j++) {  
 **double** numberOfEq = in.nextDouble();  
 str.add(numberOfEq);  
 }  
 *matr*.add(str);  
 }  
  
 **for**(**int** i = 0; i < *size*; i++){  
 *b*.add(i, *matr*.get(i).get(i));  
 }  
  
 **for** (**int** i = 0; i < *size* - 1; i++){  
 *a*.add(i, *matr*.get(i + 1).get(i));  
 *c*.add(i, *matr*.get(i).get(i + 1));  
 }  
 }  
  
 **public void** initD(Scanner in){  
 System.*out*.println("Input a numbers of vector d");  
 ArrayList<Double> str = **new** ArrayList<>();  
 **for** (**int** i = 0; i < *size*; i++){  
 **double** numberOfEq = in.nextDouble();  
 *d*.add(numberOfEq);  
 }  
 }  
  
 **public void** initialize(){  
 Scanner in = **new** Scanner(System.*in*);  
 System.*out*.println("M \* x = d");  
 System.*out*.println("Input a size of matrix m");  
 *size* = in.nextInt();  
  
 initMatr(in);  
 initD(in);  
  
 **for** (**int** i = 0; i < *size*; i++) *x*.add((**double**) 0);  
 }  
  
 **public void** straightRun(ArrayList<Double> d){  
 *alpha*.add(- *c*.get(0) / *b*.get(0));  
 *beta*.add(d.get(0) / *b*.get(0));  
 **for**(**int** i = 1; i < *size*; i ++){  
 **double** divider = (*a*.get(i - 1) \* *alpha*.get(i - 1)) + *b*.get(i);  
 **double** betaDivisible = (d.get(i) - (*a*.get(i - 1) \* *beta*.get(i - 1))) / divider;  
 **if**(i != *size* - 1) {  
 **double** alphaDivisible = -*c*.get(i) / divider;  
 *alpha*.add(alphaDivisible);  
 }  
 *beta*.add(betaDivisible);  
 }  
 }  
  
 **public void** reverseRun(ArrayList<Double> x){  
 x.set(x.size() - 1, *beta*.get( *beta*.size() - 1 ));  
 **for** (**int** i = *size* - 2; i > -1; i--){  
 **double** xIter = *alpha*.get(i) \* x.get(i + 1) + *beta*.get(i);  
 x.set(i, xIter);  
 }  
 }  
  
 **public void** decideEq(ArrayList<Double> d, ArrayList<Double> x){  
 straightRun(d);  
 reverseRun(x);  
 }  
  
 **public void** printX(){  
 System.*out*.println("x: ");  
 **for** (**int** i = 0; i < *x*.size(); i++){  
 DecimalFormat dF = **new** DecimalFormat("00.000000000000000000000000");  
 //String str = String.format(" %f", x.get(i));  
 System.*out*.println(dF.format(*x*.get(i)));  
 }  
 }  
  
 **public void** findE(){  
 ArrayList<Double> dPoint = **new** ArrayList<>();  
 **for**(**int** i = 0; i < *size*; i++){  
 **double** sub = 0;  
 **for**(**int** j = 0; j < *x*.size(); j++){  
 sub += *x*.get(i) \* *matr*.get(i).get(j);  
 }  
 dPoint.add(sub);  
 }  
 ArrayList<Double> dE = **new** ArrayList<>();  
 **for** (**int** i = 0; i < *d*.size(); i++){  
 dE.add(*d*.get(i) - dPoint.get(i));  
 }  
 ArrayList<Double> xE = **new** ArrayList<>();  
 **for** (**int** i = 0; i < *size*; i++) xE.add((**double**) 0);  
 *alpha*.clear();  
 *beta*.clear();  
 decideEq(dE, xE);  
 System.*out*.println("E: ");  
 **for** (**int** i = 0; i < xE.size(); i++){  
 DecimalFormat dF = **new** DecimalFormat("00.0000000000000000000000000000000000");  
 System.*out*.println(dF.format(xE.get(i)));  
 }  
 **for** (**int** i = 0; i < *d*.size(); i++){  
 xE.set(i, *x*.get(i) - xE.get(i));  
 }  
 System.*out*.println("x\*: ");  
 **for** (**int** i = 0; i < xE.size(); i++){  
 DecimalFormat dF = **new** DecimalFormat("00.0000000000000000000000000000000000");  
 System.*out*.println(dF.format(xE.get(i)));  
 }  
  
 }  
  
 **public void** main() {  
 initialize();  
 decideEq(*d*, *x*);  
 printX();  
 findE();  
 }  
  
 **public int** size() {  
 **return** *size*;  
 }  
  
 **public double** getX(**int** i) {  
 **return** *x*.get(i);  
 }  
}

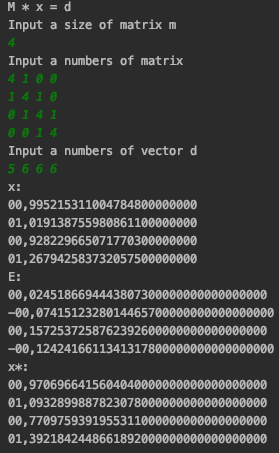
**6. Результаты**

При значениях из примера получаем:

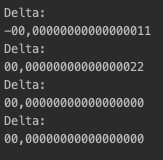
****

Вектор ошибки равен нулю и не содержит погрешность.

Протестируем программу при :



Погрешность между вычислением x методом прогонки и методом Гаусса:



**7. Вывод**

В ходе выполнения лабораторной работы был рассмотрен метод решения СЛАУ с трёхдиагональной матрицей: метод прогонки, на языке программирования Java.

Для метода прогонки можно отметить то, что отсутствует методологическая погрешность, но присутствует вычислительная погрешность в связи с использованием чисел с плавающей точкой, приводящая к накоплению вычислительной ошибки.