МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «ВГУ»)**

Факультет прикладной математики, информатики и механики

Кафедра математического обеспечения ЭВМ

**Применение точных методов решения**

**систем линейных алгебраических уравнений**

Отчёт по лабораторной работе

Вариант 6

Студент 3 курса                                                 \_\_\_\_\_\_\_ М.О. Курченков

Преподаватель                                                   \_\_\_\_\_\_\_     О.А. Махинова

Воронеж 2023

**Содержание**

[Постановка задачи 3](#_Toc151595658)

[Анализ задачи 4](#_Toc151595659)

Этапы решения [5](#_Toc151595659)

Тестирование алгоритма [7](#_Toc151595660)

[Вычислительные эксперименты](#_Toc151595660) 8

[Приложение](#_Toc151595664) 10

SecondLabApplication.java10

[Vector.java 1](#_Toc151595666)1

TapeMatrix[.java 1](#_Toc151595667)4

Tape[MatrixOperationUnit.java 1](#_Toc151595668)7

SecondPartTests.java20

# Постановка задачи

1. Реализовать классы для работы с векторами и матрицами ленточного вида в соответствии с вариантом.
2. Реализовать метод Халецкого решения СЛАУ с ленточными матрицами.
3. Провести вычислительные эксперименты.
4. Проанализировать полученные результаты.

# Анализ задачи

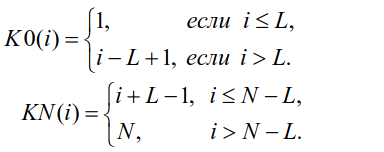
Систему уравнений, в данном варианте, задают 2 вектора: a, f ∈ Rn и целочисленные значения N, L:

* N, L – размерность системы и половина ширины ленты матрицы;
* А – одномерный массив размерности N(2L-1) − содержащий ленту матрицы по строкам;
* f – вектор правой части системы размерности N.

**Этапы решения**

1. LU-разложение:

Для формирования матриц B и C, таких, что A = B x C, используются следующие формулы:



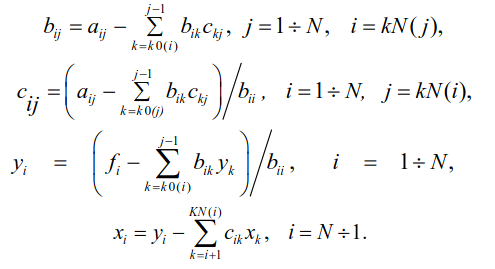


Рис. 1-2 – Формулы, используемые для LU-разложения по Халецкому

Отметим, что в данной реализации нумерация начинается с 0, поэтому в формулах индексается смещается на 1 «назад».

2) Решение уравнения BCx = f относительно Сx = y:

Полученная матрица B является нижнетреугольной с главной диагональю, заполненной единицами. В связи с этим решение выполняется «сверху вниз». На первом шаге значение y[0] = f[0]. Далее для вычисления y[i] из f[i] вычитаются значения y[j]\*b[i,j], где j = {0, 1, … , i-1}. Отметим, что для решения используются только одномерные векторы, поэтому для перехода к двоичной индексации используется hash-функция.

3) Решение уравнения Cx = y:

Матрица C является верхнетреугольной. Так, решение выполняется «снизу вверх». На первом шаге значение x[N-1] = y[N-1] / c[N-1][N-1]. Далее для вычисления x[i] из y[i] вычитаются значения y[j]\*c[i,j], где j = {N-1, N-2, … , i+1}, а полученное значение делится на c[i,i].

# Тестирование алгоритма

Используем функцию testLU() для проверки правильности LU-разложения (в программном коде — BC-разложение). Для этого выполним LU-разложение случайно сгенерированной матрицы A размерности 7 и шириной ленты 3 и перемножим матрицы B и C, сверив результат с исходной матрицей A. Как мы видим по рис. 3, представляющему результаты тестирования, вычисления проводятся верно:

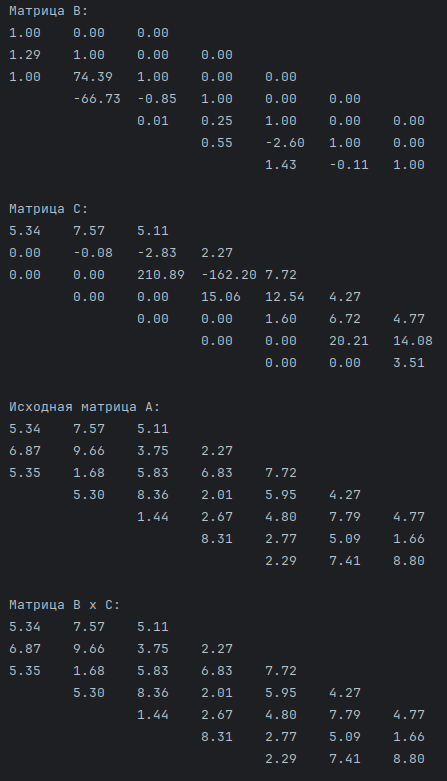


Рис. 3 – Результаты тестирования LU-разложения

# Вычислительные эксперименты

1. В ходе данного вычислительного эксперимента формируется хорошо обусловленная матрица ленточного вида wellConditionedMatrix и заранее известного точного решения xPrecise был найден соответствующий вектор правой части goodVector. Затем, после вычисления данной системы было найдено точное решение и его погрешность. Эксперимент выполняется для различных размерностей системы и соотношения параметров L/N на диапазоне [0.1, 10]. На рисунке 4 отображены результаты теста. По ним можно сделать вывод, что при увеличении одного из параметров погрешность растёт.

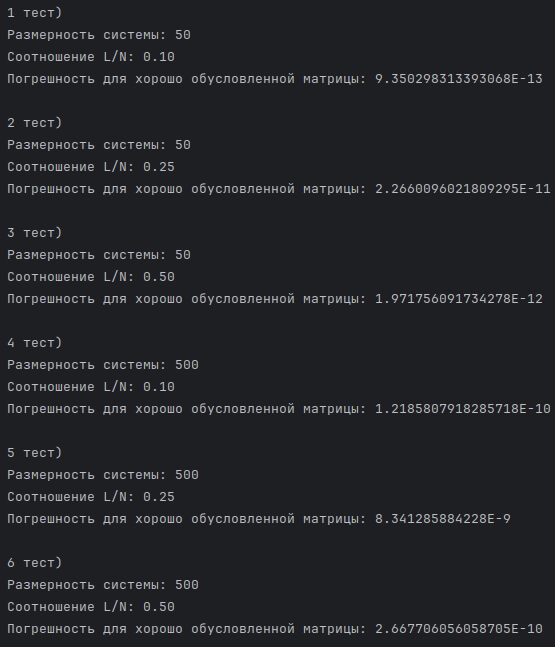


Рис. 4 – Вычислительный эксперимент для хорошо обусловленной матрицы

2. В ходе данного вычислительного эксперимента формируется плохо обусловленная матрица ленточного вида poorConditionedMatrix и заранее известного точного решения xPrecise был найден соответствующий вектор правой части poorVector. Затем, после вычисления данной системы было найдено точное решение и его погрешность. Эксперимент выполняется для различных размерностей системы и параметров обусловленности (делителя элементов на главной диагонали хорошо обусловленной матрицы для получения «плохой») на диапазоне [0.1, 10]. На рисунке 5 отображены результаты теста. По ним можно сделать вывод, что при увеличении любого из параметров погрешность растёт.

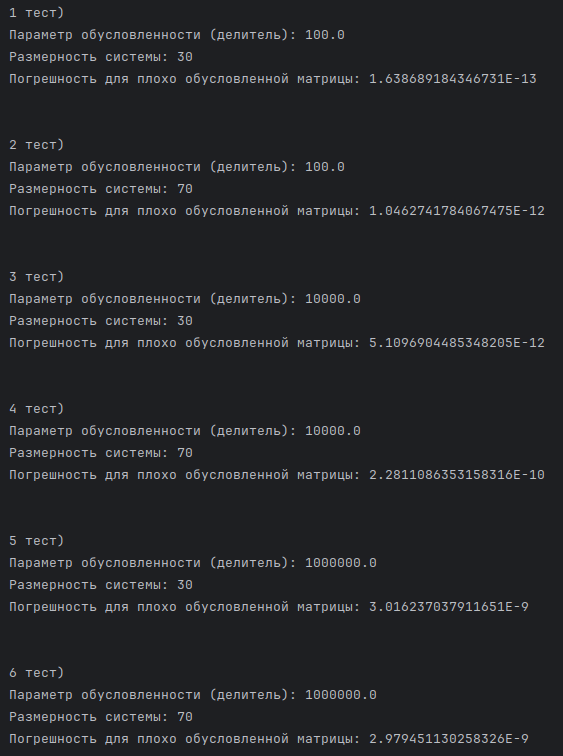


Рис. 5 – Вычислительный эксперимент для хорошо обусловленной матрицы

# **Приложение**

**SecondLabApplication,java**

package com.k4r3l1ns;  
  
import com.k4r3l1ns.models.CrackedMatrix;  
import com.k4r3l1ns.models.TapeMatrix;  
import com.k4r3l1ns.models.Vector;  
import com.k4r3l1ns.service.CrackedMatrixOperationUnit;  
import com.k4r3l1ns.service.TapeMatrixOperationUnit;  
  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.InputStream;  
  
public class SecondLabApplication {  
  
 public static final int *SIZE* = 7;  
  
 public static final int *TAPE\_WIDTH* = 3;  
  
 public static final double[] *RANGE* = {1.0, 10.0};  
  
 public static void main(String[] args) {  
  
 TapeMatrix tapeMatrix = TapeMatrix.*wellConditionedMatrix*(*SIZE*, *TAPE\_WIDTH*, *RANGE*);  
 Vector xPrecise = new Vector(*SIZE*).fillWithRandomValues(*RANGE*[0], *RANGE*[1]);  
  
 var vector = TapeMatrixOperationUnit.*multiply*(tapeMatrix, xPrecise);  
 var result = TapeMatrixOperationUnit.*solveEquation*(tapeMatrix, vector);  
  
 xPrecise.print();  
 result.print();  
  
 System.*out*.println(Vector.*copyOf*(result).subtract(xPrecise).findNorm());  
  
 }  
}

**Vector.****java**

package com.k4r3l1ns.models;  
  
import lombok.Getter;  
import lombok.Setter;  
  
import java.io.\*;  
import java.util.\*;  
import java.util.concurrent.ThreadLocalRandom;  
  
@Getter  
@Setter  
public class Vector {  
  
 private double[] values;  
 private int size;  
  
 public Vector() {  
 size = 0;  
 values = new double[0];  
 }  
  
 public Vector(int size) {  
  
 if (size < 0) {  
 throw new RuntimeException("Некорректная размерность");  
 }  
  
 this.size = size;  
 values = new double[size];  
 }  
  
 public void fill(double[] values) {  
  
 if (values.length != size) {  
 throw new RuntimeException("Размеры не совпадают");  
 }  
  
 System.*arraycopy*(values, 0, this.values, 0, size);  
 }  
  
 public Vector add(Vector vector) {  
  
 if (size != vector.size) {  
 throw new RuntimeException("Размеры не совпадают");  
 }  
  
 for (int i = 0; i < size; ++i) {  
 values[i] += vector.values[i];  
 }  
  
 return this;  
 }  
  
 public Vector subtract(Vector vector) {  
  
 vector = *copyOf*(vector).negate();  
 return this.add(vector);  
 }  
  
 private Vector negate() {  
 return this.multiplyByScalar(-1);  
 }  
  
 public Vector multiplyByScalar(double scalar) {  
  
 for (int i = 0; i < size; ++i) {  
 values[i] \*= scalar;  
 }  
  
 return this;  
 }  
  
 */\*\**  
 *\* Вычисление нормы как максимального элемента вектора*  
 *\**  
 *\* @return Норма*  
 *\*/*  
public double findNorm() {  
 return Arrays.*stream*(values).map(Math::*abs*).max().orElseThrow();  
 }  
  
 public Vector fillWithRandomValues(double min, double max) {  
  
 for (int i = 0; i < size; ++i) {  
 values[i] = ThreadLocalRandom.*current*().nextDouble(min, max);  
 }  
 return this;  
 }  
  
 public Vector fillWithDependentValues(double min, double max) {  
  
 double coefficient1 = ThreadLocalRandom.*current*().nextDouble(min, max);  
 double coefficient2 = ThreadLocalRandom.*current*().nextDouble(min, max);  
  
 for (int i = 0; i < size; ++i) {  
 values[i] = i \* coefficient1 + coefficient2;  
 }  
  
 return this;  
 }  
  
 public static Vector read(InputStream inputStream) {  
  
 Scanner scanner = new Scanner(inputStream);  
 List<Double> values = new ArrayList<>();  
 while (scanner.hasNext()) {  
 values.add(scanner.nextDouble());  
 }  
  
 Vector result = new Vector(values.size());  
 result.values = values.stream().mapToDouble(x -> x).toArray();  
  
 return result;  
 }  
  
 public void write(Writer writer) {  
 try {  
 for (var value : values) {  
 writer.write(String.*format*("%.2f\t", value));  
 }  
 writer.write("\n");  
 writer.flush();  
 } catch (IOException ex) {  
 throw new RuntimeException(ex);  
 }  
 }  
  
 public void print() {  
 for (var value : values) {  
 System.*out*.printf("%.2f\t", value);  
 }  
 System.*out*.println();  
 }  
  
 public double getValueAt(int pos) {  
 return values[pos];  
 }  
  
 public void setValueAt(int pos, double value) {  
 values[pos] = value;  
 }  
  
 public void multiplyValueAt(int pos, double scalar) {  
 values[pos] \*= scalar;  
 }  
  
 public void addValueAt(int pos, double value) {  
 values[pos] += value;  
 }  
  
 public void subtractValueAt(int pos, double value) {  
 values[pos] -= value;  
 }  
  
 public void setValues(double[] values) {  
 this.values = values;  
 this.size = values.length;  
 }  
  
 public static Vector copyOf(Vector vector) {  
 Vector result = new Vector(vector.size);  
 System.*arraycopy*(vector.getValues(), 0, result.getValues(), 0, vector.size);  
 return result;  
 }  
}

**TapeMatrix****.****java**

package com.k4r3l1ns.models;  
  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@NoArgsConstructor  
public class TapeMatrix {  
  
 int size;  
 int tapeWidth;  
  
 Vector values;  
  
 public TapeMatrix(int size, int tapeWidth) {  
  
 if (size <= 0 || tapeWidth <= 0) {  
 throw new RuntimeException("Размерность матрицы не может быть неположительной");  
 }  
  
 this.size = size;  
 this.tapeWidth = Math.*min*(tapeWidth, size);  
  
 values = new Vector(this.size \* (2 \* this.tapeWidth - 1));  
 }  
  
 public static TapeMatrix wellConditionedMatrix(  
 int size,  
 int tapeWidth,  
 double[] range  
 ) {  
 if (range.length != 2 || range[0] > range[1]) {  
 throw new RuntimeException("Некорретный диапазон");  
 }  
  
 TapeMatrix wellConditionedMatrix = new TapeMatrix(size, tapeWidth);  
 wellConditionedMatrix.values.fillWithRandomValues(range[0], range[1]);  
  
 return wellConditionedMatrix;  
 }  
  
 public static TapeMatrix poorConditionedMatrix(  
 int size,  
 int tapeWidth,  
 double[] range,  
 double divisor  
 ) {  
 TapeMatrix poorConditionedMatrix = *wellConditionedMatrix*(size, tapeWidth, range);  
  
 var values = poorConditionedMatrix.getValues();  
 for (int i = 0; i < size; ++i) {  
 values.multiplyValueAt(  
 tapeWidth - 1 + i \* (2 \* tapeWidth - 1),  
 1 / divisor  
 );  
 }  
  
 return poorConditionedMatrix;  
 }  
  
 public static TapeMatrix copyOf(TapeMatrix a) {  
  
 TapeMatrix copy = new TapeMatrix(a.size, a.tapeWidth);  
 copy.values = Vector.*copyOf*(a.values);  
  
 return copy;  
 }  
  
 public int k0(int index) {  
 return index < tapeWidth ? 0 : index - tapeWidth + 1;  
 }  
  
 public int kN(int index) {  
 return index >= size - tapeWidth ? size - 1 : index + tapeWidth - 1;  
 }  
  
 public int hash(int i, int j) {  
  
 if (Math.*abs*(i - j) >= tapeWidth ||  
 i < 0 || i >= size ||  
 j < 0 || j >= size  
 ) {  
 return -1;  
 }  
  
 return i \* (2 \* tapeWidth - 2) + tapeWidth - 1 + j;  
 }  
  
 public double valueAt(int i, int j) {  
  
 int index = hash(i, j);  
 return index == -1 ? 0.0 : values.getValueAt(index);  
 }  
  
 public void setValueAt(int i, int j, double value) {  
  
 int index = hash(i, j);  
 if (index != -1) {  
 values.setValueAt(index, value);  
 }  
 }  
  
 public void print() {  
 for (int i = 0; i < size; ++i) {  
 for (int j = 0; j < size; ++j) {  
 int index = hash(i, j);  
 if (index == -1) {  
 System.*out*.print("\t\t");  
 } else {  
 System.*out*.printf("%.2f\t", values.getValueAt(index));  
 }  
 }  
 System.*out*.println();  
 }  
 }  
}

**TapeMatrixOperationUnit****.****java**

package com.k4r3l1ns.service;  
  
import com.k4r3l1ns.models.TapeMatrix;  
import com.k4r3l1ns.models.Vector;  
  
public class TapeMatrixOperationUnit {  
  
 public static Vector solveEquation(TapeMatrix tapeMatrix, Vector vector) {  
  
 var a = TapeMatrix.*copyOf*(tapeMatrix);  
 var f = Vector.*copyOf*(vector);  
  
 if (a.getSize() != f.getSize()) {  
 throw new RuntimeException("Размерности матрицы и вектора несовместимы");  
 }  
  
 int size = a.getSize();  
 int tapeWidth = a.getTapeWidth();  
  
 TapeMatrix b = new TapeMatrix(size, tapeWidth);  
 TapeMatrix c = new TapeMatrix(size, tapeWidth);  
  
 for (int i = 0; i < size; ++i) {  
 b.setValueAt(i, i, 1.0);  
 }  
  
 for (int i = 0; i < size; ++i) {  
 for (int j = 0; j < size; ++j) {  
  
 double valueToSubtract = 0.0;  
 double aValue = a.valueAt(i, j);  
  
 if (i <= j) {  
 for (int k = a.k0(j); k <= i; ++k) {  
 valueToSubtract += b.valueAt(i, k) \* c.valueAt(k, j);  
 }  
 c.setValueAt(i, j, aValue - valueToSubtract);  
 } else {  
 for (int k = a.k0(i); k <= j; ++k) {  
 valueToSubtract += b.valueAt(i, k) \* c.valueAt(k, j);  
 }  
 b.setValueAt(i, j, (aValue - valueToSubtract) / c.valueAt(j, j));  
 }  
 }  
 }  
  
 // Получили уравнение BCx = f. Решаем отн-но Cx  
 var y = new Vector(size);  
 y.setValueAt(0, f.getValueAt(0));  
 for (int i = 1; i < size; ++i) {  
 double valueToSubtract = 0.0;  
 for (int j = 0; j < i; ++j) {  
 valueToSubtract += y.getValueAt(j) \* b.valueAt(i, j);  
 }  
 y.setValueAt(i, f.getValueAt(i) - valueToSubtract);  
 }  
  
 // Получили уравнение Cx = y. Решаем отн-но x  
 var result = new Vector(size);  
 result.setValueAt(  
 size - 1,  
 y.getValueAt(size - 1) / c.valueAt(size - 1, size - 1)  
 );  
 for (int i = size - 2; i >= 0; --i) {  
 double valueToSubtract = 0.0;  
 for (int j = size - 1; j > i; --j) {  
 valueToSubtract += result.getValueAt(j) \* c.valueAt(i, j);  
 }  
 result.setValueAt(i, (y.getValueAt(i) - valueToSubtract) / c.valueAt(i, i));  
 }  
  
 return result;  
 }  
  
 public static Vector multiply(TapeMatrix tapeMatrix, Vector vector) {  
  
 var a = TapeMatrix.*copyOf*(tapeMatrix);  
 var x = Vector.*copyOf*(vector);  
  
 if (a.getSize() != x.getSize()) {  
 throw new RuntimeException("Размерности матрицы и вектора несовместимы");  
 }  
  
 int size = a.getSize();  
 int tapeWidth = a.getTapeWidth();  
  
 var result = new Vector(size);  
  
 for (int i = 0; i < size; ++i) {  
 double rowSum = 0.0;  
 for (int j = i - tapeWidth + 1; j < i + tapeWidth; ++j) {  
 if (j >= 0 && j < size) {  
 rowSum += a.valueAt(i, j) \* x.getValueAt(j);  
 }  
 }  
 result.setValueAt(i, rowSum);  
 }  
  
 return result;  
 }  
  
 public static TapeMatrix multiply(TapeMatrix matrix1, TapeMatrix matrix2) {  
  
 var b = TapeMatrix.*copyOf*(matrix1);  
 var c = TapeMatrix.*copyOf*(matrix2);  
  
 if (b.getSize() != c.getSize() || b.getTapeWidth() != c.getTapeWidth()) {  
 throw new RuntimeException("Размерности матриц несовместимы");  
 }  
  
 int size = b.getSize();  
 int tapeWidth = b.getTapeWidth();  
  
 var result = new TapeMatrix(size, tapeWidth);  
  
 for (int i = 0; i < size; ++i) {  
 for (int j = 0; j < size; ++j) {  
 double sum = 0.0;  
 for (int k = 0; k < size; ++k) {  
 sum += b.valueAt(i, k) \* c.valueAt(k, j);  
 }  
 result.setValueAt(i, j, sum);  
 }  
 }  
  
 return result;  
 }  
}

**SecondPartTests****.****java**

import com.k4r3l1ns.models.TapeMatrix;  
import com.k4r3l1ns.models.Vector;  
import com.k4r3l1ns.service.TapeMatrixOperationUnit;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.params.ParameterizedTest;  
import org.junit.jupiter.params.provider.ValueSource;  
  
public class SecondPartTests {  
  
 public static final int *SIZE* = 7;  
  
 public static final int *TAPE\_WIDTH* = 3;  
  
 public static final double[] *RANGE* = {1.0, 10.0};  
  
 public static int *testNum* = 1;  
  
 @ParameterizedTest  
 @ValueSource(  
 ints = { 50, 500 }  
 )  
 public void testWellConditionedMatrix(int size) {  
  
 double[] coefficientLToN = { 0.1, 0.25, 0.5 };  
  
 for (double coefficient : coefficientLToN) {  
  
 int tapeWidth = (int) (size / coefficient);  
  
 TapeMatrix wellConditionedMatrix = TapeMatrix.*wellConditionedMatrix*(size, tapeWidth, *RANGE*);  
 Vector xPrecise = new Vector(size).fillWithRandomValues(*RANGE*[0], *RANGE*[1]);  
  
 Vector goodVector = TapeMatrixOperationUnit.*multiply*(wellConditionedMatrix, xPrecise);  
  
 System.*out*.println(*testNum*++ + " тест)");  
 System.*out*.println("Размерность системы: " + size);  
 System.*out*.format("Соотношение L/N: %.2f\n", coefficient);  
 System.*out*.println(  
 "Погрешность для хорошо обусловленной матрицы: " +  
 TapeMatrixOperationUnit.*solveEquation*(wellConditionedMatrix, goodVector)  
 .subtract(xPrecise).findNorm() + "\n"  
 );  
 }  
 }  
  
 @ParameterizedTest  
 @ValueSource(  
 doubles = { 100, 10000, 1000000 }  
 )  
 public void testPoorConditionedMatrix(double divisor) {  
  
 int[] sizes = { 30, 70 };  
  
 for (int size : sizes) {  
  
 int tapeWidth = size / 10;  
  
 TapeMatrix poorConditionedMatrix = TapeMatrix.*poorConditionedMatrix*(size, tapeWidth, *RANGE*, divisor);  
 Vector xPrecise = new Vector(size).fillWithRandomValues(*RANGE*[0], *RANGE*[1]);  
  
 Vector poorVector = TapeMatrixOperationUnit.*multiply*(poorConditionedMatrix, xPrecise);  
  
 System.*out*.println(*testNum*++ + " тест)");  
 System.*out*.println("Параметр обусловленности (делитель): " + divisor);  
 System.*out*.println("Размерность системы: " + size);  
 System.*out*.println(  
 "Погрешность для плохо обусловленной матрицы: " +  
 TapeMatrixOperationUnit.*solveEquation*(poorConditionedMatrix, poorVector)  
 .subtract(xPrecise).findNorm() + "\n\n"  
 );  
 }  
 }  
  
 @Test  
 public void testLU() {  
 TapeMatrix tapeMatrix = TapeMatrix.*wellConditionedMatrix*(*SIZE*, *TAPE\_WIDTH*, *RANGE*);  
 Vector xPrecise = new Vector(*SIZE*).fillWithRandomValues(*RANGE*[0], *RANGE*[1]);  
  
 var vector = TapeMatrixOperationUnit.*multiply*(tapeMatrix, xPrecise);  
  
 var a = TapeMatrix.*copyOf*(tapeMatrix);  
 var f = Vector.*copyOf*(vector);  
  
 if (a.getSize() != f.getSize()) {  
 throw new RuntimeException("Размерности матрицы и вектора несовместимы");  
 }  
  
 int size = a.getSize();  
 int tapeWidth = a.getTapeWidth();  
  
 TapeMatrix b = new TapeMatrix(size, tapeWidth);  
 TapeMatrix c = new TapeMatrix(size, tapeWidth);  
  
 for (int i = 0; i < size; ++i) {  
 b.setValueAt(i, i, 1.0);  
 }  
  
 for (int i = 0; i < size; ++i) {  
 for (int j = 0; j < size; ++j) {  
  
 double valueToSubtract = 0.0;  
 double aValue = a.valueAt(i, j);  
  
 if (i <= j) {  
 for (int k = a.k0(j); k <= i; ++k) {  
 valueToSubtract += b.valueAt(i, k) \* c.valueAt(k, j);  
 }  
 c.setValueAt(i, j, aValue - valueToSubtract);  
 } else {  
 for (int k = a.k0(i); k <= j; ++k) {  
 valueToSubtract += b.valueAt(i, k) \* c.valueAt(k, j);  
 }  
 b.setValueAt(i, j, (aValue - valueToSubtract) / c.valueAt(j, j));  
 }  
 }  
 }  
  
 System.*out*.println("Матрица B:");  
 b.print();  
 System.*out*.println("\nМатрица C:");  
 c.print();  
 System.*out*.println("\nИсходная матрица A:");  
 a.print();  
 System.*out*.println("\nМатрица B x C:");  
 TapeMatrixOperationUnit.*multiply*(b, c).print();  
 System.*out*.println();  
 }  
}