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

**Численные методы решения СЛАУ с матрицами специального вида**

Отчёт по лабораторной работе

Вариант 12

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)

[Вычислительные эксперименты 7](#_Toc151595660)

[Приложение 10](#_Toc151595664)

FirstLabApplication.java [10](#_Toc151595665)

[Vector.java 1](#_Toc151595666)1

CrackedMatrix[.java 14](#_Toc151595667)

[MatrixOperationUnit.java 1](#_Toc151595668)8

FirstLabOperationTests.java22

# Постановка задачи

1. Реализовать классы для работы с векторами и матрицами специального вида в соответствии с вариантом.
2. Разработать и реализовать алгоритм вычисления решения матриц специального вида.
3. Провести вычислительные эксперименты.
4. Проанализировать полученные результаты.

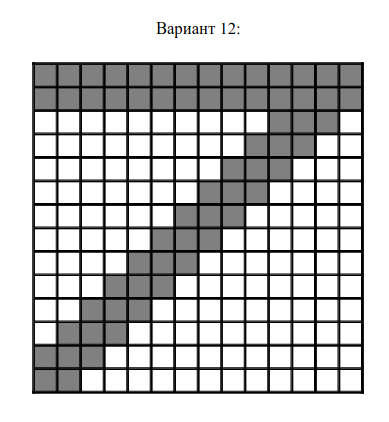


Рис. 1 — Вид специальной матрицы

# Анализ задачи

Систему уравнений, в данном варианте, задают шесть векторов: a, b, c, f, p, q ∈ Rn:

* a, b, c – векторы элементов матрицы А, расположенных на верхней побочной кодиагонали, на побочной диагонали и на нижней кодиагонали (в программе представлены как lSideDiagonal, sideDiagonal, rSideDiagonal).
* p – вектор элементов 1-ой строки матрицы А (в программе — uCrackedLine).
* q – вектор элементов 2-ой строки матрицы А (в программе — dCrackedLine).
* f – вектор правой части системы уравнений.

**Этапы решения**

1. Обратный ход: от последней строки до нижней «испорченной» выполняется цикл:

1.1) Значения всей строки делятся на число, стоящее на побочной диагонали, нормируя её.

1.2) С помощью гауссовых преобразований обнуляются все значения столбца, кроме нижней кодиагонали. Таким образом, для верхней кодиагонали и обеих «испорченных» строк нормированная строка умножается на значение в обнуляемой строке и вычитается из этой строки.

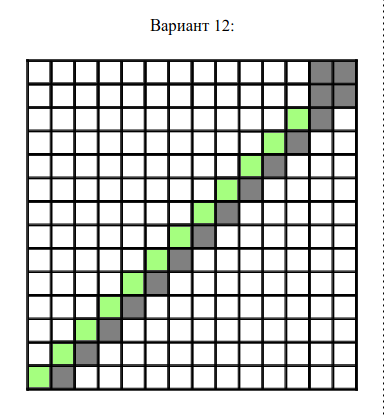


Рис. 2 – Матрица после выполнения шага 1

(зелёные клетки — 1, серые — непустые значения, остальные — 0)

2) Далее нормируются оставшиеся 2 значения побочной диагонали: сначала нормируется нижняя «испорченная строка» (делением на элемент (size - 1, 2)), а далее с помощью гауссовых преобразований обнуляется элемент (size - 1, 1). Затем нормируется оставшееся значение на верхней «испорченной» строке. Так, матрица принимает вид, показанный на рисунке 3. Отметим, что после этого шага необходимо соотнести значения векторов p, q, b и c. При этом, приоритет принадлежит векторам p и q, т. к. в преобразованиях участвовали именно они.

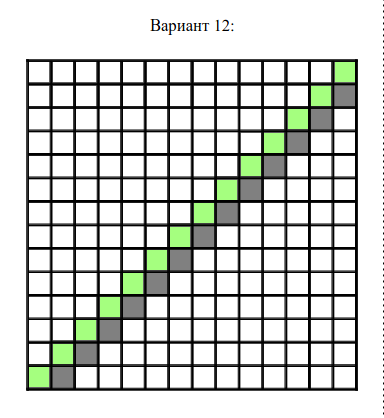


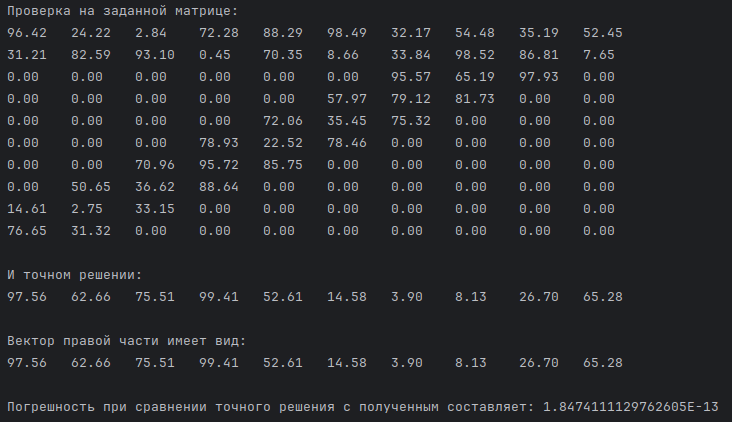
Рис. 3 – Матрица после выполнения шага 2

(зелёные клетки — 1, серые — непустые значения, остальные — 0)

3) Теперь матрица имеет вид, в котором легко находятся решения: сначала вычисляется x[size] — оно, очевидно, равно значению f[0]. Далее значения вектора х считаются рекуррентно: x[i] = f[size-i] - x[i-1] \* c[size-i].

# Вычислительные эксперименты

1. В ходе данного вычислительного эксперимента по случайно заполненной матрице специального вида crackedMatrix и заранее известного точного решения xPrecise был найден соответствующий вектор правой части vector. Затем, после вычисления данной системы было найдено решение x. На рисунке 3 отображены результаты теста, а погрешность стремится к нулю.

Рис. 4 – Простой вычислительный эксперимент

2. В ходе следующего вычислительного эксперимента по заранее заданной в файле матрице специального вида a и точному решению xPrecise был найден соответствующий вектор правой части vector. Затем используется разработанный алгоритм и на каждом шаге проверяется значение погрешности как норма разности полученного вектора и точного решения. Результаты представлены ниже. По ним видно, что максимальная погрешность была достигнута ещё на первом шаге, и далее она не увеличивалась.

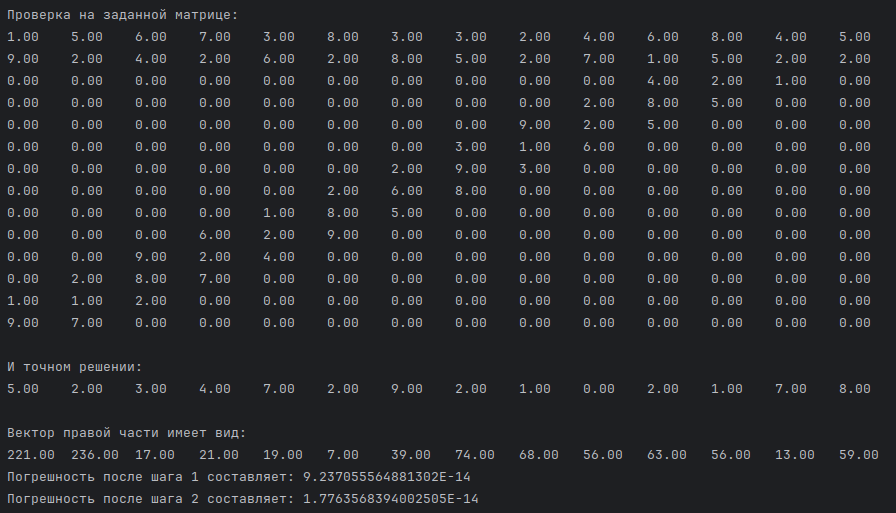


Рис. 5 – Поэтапный простой вычислительный эксперимент

3. Далее будем исследовать влияние на погрешность таких факторов, как размерность матрицы, диапазон её значений и её обусловленность. Для этого будем фиксировать 2 значения или приводить их к аналогичному виду и изменять третий параметр.

3.1. Рассмотрим влияние размерности матрицы на погрешность. Результаты отображены на рисунке 6. По ним мы видим, что при логарифмическом шаге в 8 единиц погрешность сильно ухудшается (примерно в 100-200 раз).

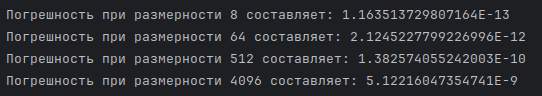
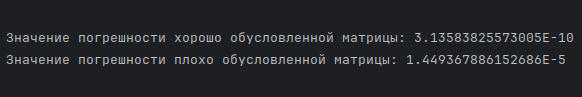
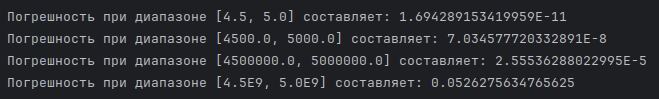


Рис. 6 – Зависимость погрешности от размера матрицы

3.2. Перейдём к рассмотрению влияния обусловленности матрицы на погрешность. Для этого внедрим в одну из матриц зависимости в генерации строк. Результаты показаны на рисунке 7. По ним мы видим, что погрешность существенно отличается (примерно в 10^5 раз).

Рис. 7 – Зависимость погрешности от обусловленности матрицы

3.3. Рассмотрим влияние диапазона значений матрицы на погрешность. По рисунку 8 можно сделать вывод о том, что диапазон значений существенно влияет на погрешность вычислений.

 Рис. 8 – Зависимость погрешности от диапазона значений элементов матрицы

Из вышепроведённых экспериментов можно сделать вывод, что обусловленность матрицы влияет на погрешность значительно сильнее, чем остальные зависимости. Это связано с тем, что обусловленность матрицы характеризует силу колебаний значений матрицы от изменения её входных аргументов. В данном случае им является вектор правой части, в случае плохой обусловленности увеличивающий погрешность.

# **Приложение**

**FirstLabApplication,java**

package com.k4r3l1ns;  
  
  
import com.k4r3l1ns.models.CrackedMatrix;  
import com.k4r3l1ns.models.Vector;  
import com.k4r3l1ns.service.MatrixOperationUnit;  
  
import java.io.\*;  
  
public class FirstLabApplication {  
  
 private static final String *FILES\_DIRECTORY* = "/home/k4r3l1ns/Desktop/NM\_labs/first\_lab/src/main/resources/";  
  
 private static final File *MATRIX\_FILE* = new File(*FILES\_DIRECTORY* + "matrix.txt");  
 private static final File *VECTOR\_FILE* = new File(*FILES\_DIRECTORY* + "vector.txt");  
 private static final File *DESTINATION\_FILE* = new File(*FILES\_DIRECTORY* + "result.txt");  
  
 public static void main(String[] args) {  
  
 try {  
  
 InputStream matrixFile = new FileInputStream(*MATRIX\_FILE*);  
 InputStream vectorFile = new FileInputStream(*VECTOR\_FILE*);  
 Writer writer = new FileWriter(*DESTINATION\_FILE*, false);  
  
 var crackedMatrix = CrackedMatrix.*read*(matrixFile);  
 var vector = Vector.*read*(vectorFile);  
  
 var result = MatrixOperationUnit.*solveEquation*(crackedMatrix, vector);  
 result.write(writer);  
  
 } catch (FileNotFoundException ex) {  
 System.*out*.println("Файл не найден");  
 } catch (SecurityException ex) {  
 System.*out*.println("Файл для записи обладает недоступными настройками доступа");  
 } catch (IOException ex) {  
 throw new RuntimeException(ex);  
 }  
 }  
}

**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;  
 }  
}

**CrackedMatrix****.****java**

package com.k4r3l1ns.models;  
  
import lombok.Getter;  
import lombok.Setter;  
  
import java.io.\*;  
import java.util.Scanner;  
  
@Getter  
@Setter  
public class CrackedMatrix {  
  
 private int size;  
  
 */\*\**  
 *\* Побочная диагональ матрицы*  
 *\*/*  
private Vector sideDiagonal;  
  
 */\*\**  
 *\* Диагональ слева от побочной*  
 *\*/*  
private Vector lSideDiagonal;  
  
 */\*\**  
 *\* Диагональ справа от побочной*  
 *\*/*  
private Vector rSideDiagonal;  
  
 */\*\**  
 *\* Верхняя "испорченная" строка*  
 *\*/*  
private Vector uCrackedLine;  
  
 */\*\**  
 *\* Нижняя "испорченная" строка*  
 *\*/*  
private Vector dCrackedLine;  
  
 public CrackedMatrix() {  
 size = 0;  
 sideDiagonal = new Vector(0);  
 lSideDiagonal = new Vector(0);  
 rSideDiagonal = new Vector(0);  
 uCrackedLine = new Vector(0);  
 dCrackedLine = new Vector(0);  
 }  
  
 public CrackedMatrix(int size) {  
  
 if (size < 0) {  
 throw new RuntimeException("Некорректная размерность");  
 }  
  
 this.size = size;  
 sideDiagonal = new Vector(size);  
  
 lSideDiagonal = new Vector(size);  
 lSideDiagonal.setValueAt(lSideDiagonal.getSize() - 1, 0.0);  
  
 rSideDiagonal = new Vector(size);  
 rSideDiagonal.setValueAt(0, 0.0);  
  
 uCrackedLine = new Vector(size);  
 dCrackedLine = new Vector(size);  
 }  
  
 public CrackedMatrix add(CrackedMatrix crackedMatrix) {  
  
 if (crackedMatrix.size != size) {  
 throw new RuntimeException("Несовместимые размерности");  
 }  
  
 sideDiagonal = sideDiagonal.add(crackedMatrix.sideDiagonal);  
 lSideDiagonal = lSideDiagonal.add(crackedMatrix.lSideDiagonal);  
 rSideDiagonal = rSideDiagonal.add(crackedMatrix.rSideDiagonal);  
 uCrackedLine = uCrackedLine.add(crackedMatrix.uCrackedLine);  
 dCrackedLine = dCrackedLine.add(crackedMatrix.dCrackedLine);  
  
 return this;  
 }  
  
 public CrackedMatrix subtract(CrackedMatrix crackedMatrix) {  
  
 crackedMatrix = crackedMatrix.negate();  
 return this.add(crackedMatrix);  
 }  
  
 public CrackedMatrix multiplyByScalar(double scalar) {  
  
 sideDiagonal = sideDiagonal.multiplyByScalar(scalar);  
 lSideDiagonal = lSideDiagonal.multiplyByScalar(scalar);  
 rSideDiagonal = rSideDiagonal.multiplyByScalar(scalar);  
 uCrackedLine = uCrackedLine.multiplyByScalar(scalar);  
 dCrackedLine = dCrackedLine.multiplyByScalar(scalar);  
  
 return this;  
 }  
  
 public CrackedMatrix negate() {  
 return this.multiplyByScalar(-1);  
 }  
  
 public static CrackedMatrix read(InputStream inputStream) {  
  
 Scanner scanner = new Scanner(inputStream);  
  
 int lineIndex = 0;  
 CrackedMatrix result = new CrackedMatrix();  
 result.size = -1;  
  
 while (  
 scanner.hasNextLine() &&  
 (result.size == -1 || lineIndex < result.size)  
 ) {  
 String line = scanner.nextLine();  
  
 Vector row =  
 Vector.*read*(  
 new ByteArrayInputStream(  
 line.getBytes()  
 )  
 );  
  
 if (result.size == -1) {  
 result = new CrackedMatrix(row.getSize());  
 } else if (result.size != row.getSize()) {  
 throw new RuntimeException("Входные данные не являются матрицей");  
 }  
  
 if (lineIndex == 0) {  
 result.uCrackedLine = row;  
 } else if (lineIndex == 1) {  
 result.dCrackedLine = row;  
 }  
  
 result.sideDiagonal.setValueAt(  
 lineIndex,  
 row.getValueAt(row.getSize() - 1 - lineIndex)  
 );  
  
 if (lineIndex > 0) {  
 result.rSideDiagonal.setValueAt(  
 lineIndex,  
 row.getValueAt(row.getSize() - lineIndex)  
 );  
 }  
  
 if (lineIndex < result.size - 1) {  
 result.lSideDiagonal.setValueAt(  
 lineIndex,  
 row.getValueAt(row.getSize() - 2 - lineIndex)  
 );  
 }  
  
 ++lineIndex;  
 }  
  
 return result;  
 }  
  
 public void write(Writer writer) {  
  
 for (int i = 0; i < size; ++i) {  
 rowAt(i).write(writer);  
 }  
  
 try {  
 writer.flush();  
 } catch (IOException ex) {  
 System.*out*.println("Ошибка ввода в файл");  
 }  
 }  
  
 public void print() {  
  
 for (int i = 0; i < size; ++i) {  
 rowAt(i).print();  
 }  
 }  
  
 public Vector rowAt(int pos) {  
 Vector row;  
 if (pos == 0) {  
 row = uCrackedLine;  
 } else if (pos == 1) {  
 row = dCrackedLine;  
 } else if (pos < size - 1) {  
 row = new Vector(size);  
 row.setValueAt(size - pos - 2, lSideDiagonal.getValueAt(pos));  
 row.setValueAt(size - pos - 1, sideDiagonal.getValueAt(pos));  
 row.setValueAt(size - pos, rSideDiagonal.getValueAt(pos));  
 } else {  
 row = new Vector(size);  
 row.setValueAt(0, sideDiagonal.getValueAt(size - 1));  
 row.setValueAt(1, rSideDiagonal.getValueAt(size - 1));  
 }  
  
 return row;  
 }  
  
 public static CrackedMatrix copyOf(CrackedMatrix crackedMatrix) {  
  
 CrackedMatrix copy = new CrackedMatrix(crackedMatrix.size);  
  
 copy.size = crackedMatrix.size;  
 System.*arraycopy*(crackedMatrix.rSideDiagonal.getValues(), 0, copy.rSideDiagonal.getValues(), 0, copy.size);  
 System.*arraycopy*(crackedMatrix.lSideDiagonal.getValues(), 0, copy.lSideDiagonal.getValues(), 0, copy.size);  
 System.*arraycopy*(crackedMatrix.sideDiagonal.getValues(), 0, copy.sideDiagonal.getValues(), 0, copy.size);  
 System.*arraycopy*(crackedMatrix.uCrackedLine.getValues(), 0, copy.uCrackedLine.getValues(), 0, copy.size);  
 System.*arraycopy*(crackedMatrix.dCrackedLine.getValues(), 0, copy.dCrackedLine.getValues(), 0, copy.size);  
  
 return copy;  
 }  
  
 public static CrackedMatrix poorConditionedMatrix(int size, double min, double max) {  
  
 var poorConditionedMatrix = new CrackedMatrix(size);  
  
 poorConditionedMatrix.getLSideDiagonal().fillWithRandomValues(min, max);  
 poorConditionedMatrix.getRSideDiagonal().fillWithRandomValues(min, max);  
  
 poorConditionedMatrix.getSideDiagonal().fillWithDependentValues(min, max);  
 poorConditionedMatrix.getUCrackedLine().fillWithDependentValues(min, max);  
 poorConditionedMatrix.getDCrackedLine().fillWithDependentValues(min, max);  
  
 return poorConditionedMatrix;  
 }  
  
 public static CrackedMatrix wellConditionedMatrix(int size, double min, double max) {  
  
 var wellConditionedMatrix = new CrackedMatrix(size);  
  
 wellConditionedMatrix.getSideDiagonal().fillWithRandomValues(min, max);  
 wellConditionedMatrix.getLSideDiagonal().fillWithRandomValues(min, max);  
 wellConditionedMatrix.getRSideDiagonal().fillWithRandomValues(min, max);  
 wellConditionedMatrix.getUCrackedLine().fillWithRandomValues(min, max);  
 wellConditionedMatrix.getDCrackedLine().fillWithRandomValues(min, max);  
  
 return wellConditionedMatrix;  
 }  
}

**MatrixOperationUnit****.****java**

package com.k4r3l1ns.service;  
  
import com.k4r3l1ns.models.CrackedMatrix;  
import com.k4r3l1ns.models.Vector;  
  
public class MatrixOperationUnit {  
  
 public static Vector solveEquation(CrackedMatrix a, Vector f) {  
  
 if (a.getSize() != f.getSize()) {  
 throw new RuntimeException("Размерности матрицы и вектора несовместимы");  
 }  
  
 a = CrackedMatrix.*copyOf*(a);  
 f = Vector.*copyOf*(f);  
  
 int size = a.getSize();  
  
 // Верхняя кодиагональ  
 var lSideDiagonal = a.getLSideDiagonal();  
  
 // Побочная диагональ  
 var sideDiagonal = a.getSideDiagonal();  
  
 // Нижняя кодиагональ  
 var rSideDiagonal = a.getRSideDiagonal();  
  
 var uCrackedLine = a.getUCrackedLine();  
 var dCrackedLine = a.getDCrackedLine();  
  
 // Шаг 2 (шаг 1 пропускается, т.к. первая "испорченная" диагональ на верхней строке матрицы)  
 for (int i = size - 1; i > 1; --i) {  
  
 var r = 1 / sideDiagonal.getValueAt(i);  
 sideDiagonal.setValueAt(i, 1.0);  
 rSideDiagonal.multiplyValueAt(i, r);  
 f.multiplyValueAt(i, r);  
  
 if (i > 2) {  
 r = lSideDiagonal.getValueAt(i - 1);  
 lSideDiagonal.setValueAt(i - 1, 0.0);  
 sideDiagonal.subtractValueAt(i - 1, r \* rSideDiagonal.getValueAt(i));  
 f.subtractValueAt(i - 1, r \* f.getValueAt(i));  
 }  
  
 r = dCrackedLine.getValueAt(size - 1 - i);  
 dCrackedLine.setValueAt(size - 1 - i, 0.0);  
 dCrackedLine.subtractValueAt(size - i, r \* rSideDiagonal.getValueAt(i));  
 f.subtractValueAt(1, r \* f.getValueAt(i));  
  
 r = uCrackedLine.getValueAt(size - 1 - i);  
 uCrackedLine.setValueAt(size - 1 - i, 0.0);  
 uCrackedLine.subtractValueAt(size - i, r \* rSideDiagonal.getValueAt(i));  
 f.subtractValueAt(0, r \* f.getValueAt(i));  
 }  
  
 var r = 1 / dCrackedLine.getValueAt(size - 2);  
 dCrackedLine.setValueAt(size - 2, 1.0);  
 dCrackedLine.multiplyValueAt(size - 1, r);  
 f.multiplyValueAt(1, r);  
  
 r = uCrackedLine.getValueAt(size - 2);  
 uCrackedLine.setValueAt(size - 2, 0.0);  
 uCrackedLine.subtractValueAt(size - 1, r \* dCrackedLine.getValueAt(size - 1));  
 f.subtractValueAt(0, r \* f.getValueAt(1));  
  
 r = 1 / uCrackedLine.getValueAt(size - 1);  
 uCrackedLine.setValueAt(size - 1, 1.0);  
 f.multiplyValueAt(0, r);  
  
 // Значения "cracked lines" верны, осталось подогнать под них диагонали 0 и 1 рядов  
 lSideDiagonal.setValueAt(0, 0.0);  
 sideDiagonal.setValueAt(0, uCrackedLine.getValueAt(size - 1));  
 sideDiagonal.setValueAt(1, dCrackedLine.getValueAt(size - 2));  
 rSideDiagonal.setValueAt(1, dCrackedLine.getValueAt(size - 1));  
  
 var x = new Vector(size);  
  
 x.setValueAt(size - 1, f.getValueAt(0));  
 for (int i = size - 2; i >= 0; --i) {  
 x.setValueAt(  
 i,  
 f.getValueAt(size - 1 - i) - rSideDiagonal.getValueAt(size - 1 - i) \* x.getValueAt(i + 1)  
 );  
 }  
  
 return x;  
 }  
  
 public static Vector multiply(CrackedMatrix crackedMatrix, Vector vector) {  
  
 if (crackedMatrix.getSize() != crackedMatrix.getSize()) {  
 throw new RuntimeException("Размерности матрицы и вектора несовместимы");  
 }  
  
 int size = vector.getSize();  
  
 var uCrackedLine = crackedMatrix.getUCrackedLine();  
 var dCrackedLine = crackedMatrix.getDCrackedLine();  
  
 Vector result = new Vector(size);  
  
 double uRowSum = 0.0;  
 double dRowSum = 0.0;  
 for (int i = 0; i < size; ++i) {  
 uRowSum += uCrackedLine.getValueAt(i) \* vector.getValueAt(i);  
 dRowSum += dCrackedLine.getValueAt(i) \* vector.getValueAt(i);  
 }  
 result.setValueAt(0, uRowSum);  
 result.setValueAt(1, dRowSum);  
  
 // Верхняя кодиагональ  
 var lSideDiagonal = crackedMatrix.getLSideDiagonal();  
  
 // Побочная диагональ  
 var sideDiagonal = crackedMatrix.getSideDiagonal();  
  
 // Нижняя кодиагональ  
 var rSideDiagonal = crackedMatrix.getRSideDiagonal();  
  
 for (int i = 2; i < size - 1; ++i) {  
 result.setValueAt(  
 i,  
 lSideDiagonal.getValueAt(i) \* vector.getValueAt(size - 2 - i) +  
 sideDiagonal.getValueAt(i) \* vector.getValueAt(size - 1 - i) +  
 rSideDiagonal.getValueAt(i) \* vector.getValueAt(size - i)  
 );  
 }  
  
 result.setValueAt(  
 size - 1,  
 sideDiagonal.getValueAt(size - 1) \* vector.getValueAt(0) +  
 rSideDiagonal.getValueAt(size - 1) \* vector.getValueAt(1)  
 );  
  
 return result;  
 }  
}

**FirstLabApplicationTests****.****java**

import com.k4r3l1ns.models.CrackedMatrix;  
import com.k4r3l1ns.models.Vector;  
import com.k4r3l1ns.service.MatrixOperationUnit;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.params.ParameterizedTest;  
import org.junit.jupiter.params.provider.ValueSource;  
  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
  
public class FirstLabApplicationTests {  
  
 private static final String *FILES\_DIRECTORY* = "/home/k4r3l1ns/Desktop/NM\_labs/first\_lab/src/main/resources/";  
  
 private static final File *MATRIX\_FILE* = new File(*FILES\_DIRECTORY* + "matrix.txt");  
 private static final File *X\_PRECISE\_FILE* = new File(*FILES\_DIRECTORY* + "precise.txt");  
  
 @Test  
 public void testIncorrectnessByReverseChecking() {  
  
 int size = 1000;  
  
 var crackedMatrix = CrackedMatrix.*wellConditionedMatrix*(size, 0.0, 100.0);  
 var xPrecise = new Vector(size).fillWithRandomValues(0.0, 100.0);  
  
 var vector = MatrixOperationUnit.*multiply*(crackedMatrix, xPrecise);  
 var result = MatrixOperationUnit.*solveEquation*(crackedMatrix, vector);  
  
 System.*out*.println("\nПогрешность при сравнении точного решения с полученным составляет: " +  
 result.subtract(xPrecise).findNorm() + "\n\n");  
 }  
  
 @ParameterizedTest  
 @ValueSource(ints = {5, 50, 500, 5000})  
 public void testIncorrectnessByDimension(int size) {  
  
 CrackedMatrix crackedMatrix = CrackedMatrix.*wellConditionedMatrix*(size, 0.0, 100.0);  
  
 var xPrecise = new Vector(size).fillWithRandomValues(0.0, 100.0);  
  
 var vector = MatrixOperationUnit.*multiply*(crackedMatrix, xPrecise);  
 var result = MatrixOperationUnit.*solveEquation*(crackedMatrix, vector);  
  
 System.*out*.println("Погрешность при размерности " + size +  
 " составляет: " + result.subtract(xPrecise).findNorm());  
 }  
  
 @Test  
 public void testPoorConditionedMatrix() {  
  
 final int size = 1000;  
  
 CrackedMatrix poorConditionedMatrix = CrackedMatrix.*poorConditionedMatrix*(size, 0.0, 100.0);  
 CrackedMatrix wellConditionedMatrix = CrackedMatrix.*wellConditionedMatrix*(size, 0.0, 100.0);  
  
 var xPrecise = new Vector(size).fillWithRandomValues(0.0, 100.0);  
  
 var poorVector = MatrixOperationUnit.*multiply*(poorConditionedMatrix, xPrecise);  
 var goodVector = MatrixOperationUnit.*multiply*(wellConditionedMatrix, xPrecise);  
  
 var poorResult = MatrixOperationUnit.*solveEquation*(poorConditionedMatrix, poorVector);  
 var goodResult = MatrixOperationUnit.*solveEquation*(wellConditionedMatrix, goodVector);  
  
 System.*out*.println("\nЗначение погрешности хорошо обусловленной матрицы: "  
 + goodResult.subtract(xPrecise).findNorm());  
 System.*out*.println("Значение погрешности плохо обусловленной матрицы: "  
 + poorResult.subtract(xPrecise).findNorm() + "\n");  
 }  
  
 @ParameterizedTest  
 @ValueSource(doubles = {5.0, 5000.0, 5000000.0, 5000000000.0})  
 public void testIncorrectnessByDoubles(double maxDouble) {  
  
 final int size = 1000;  
  
 CrackedMatrix crackedMatrix = CrackedMatrix.*wellConditionedMatrix*(size, 0.9 \* maxDouble, maxDouble);  
 var xPrecise = new Vector(size).fillWithRandomValues(0.9 \* maxDouble, maxDouble);  
  
 var vector = MatrixOperationUnit.*multiply*(crackedMatrix, xPrecise);  
 var result = MatrixOperationUnit.*solveEquation*(crackedMatrix, vector);  
  
 System.*out*.println("Погрешность при диапазоне [" + 0.9 \* maxDouble + ", " + maxDouble +  
 "] составляет: " + result.subtract(xPrecise).findNorm());  
 }  
  
 @Test  
 public void testProcessingCorrectness()  
 throws FileNotFoundException {  
  
 var a = CrackedMatrix.*read*(new FileInputStream(*MATRIX\_FILE*));  
 var xPrecise = Vector.*read*(new FileInputStream(*X\_PRECISE\_FILE*));  
  
 var f = MatrixOperationUnit.*multiply*(a, xPrecise);  
  
 if (a.getSize() != f.getSize()) {  
 throw new RuntimeException("Размерности матрицы и вектора несовместимы");  
 }  
  
 a = CrackedMatrix.*copyOf*(a);  
  
 int size = a.getSize();  
  
 // Верхняя кодиагональ  
 var lSideDiagonal = a.getLSideDiagonal();  
  
 // Побочная диагональ  
 var sideDiagonal = a.getSideDiagonal();  
  
 // Нижняя кодиагональ  
 var rSideDiagonal = a.getRSideDiagonal();  
  
 var uCrackedLine = a.getUCrackedLine();  
 var dCrackedLine = a.getDCrackedLine();  
  
 System.*out*.println("Проверка на заданной матрице: ");  
 a.print();  
 System.*out*.println("\nИ точном решении: ");  
 xPrecise.print();  
 System.*out*.println("\nВектор правой части имеет вид: ");  
 f.print();  
 System.*out*.println("\nПогрешность в начале вычислений составляет: " +  
 MatrixOperationUnit.*solveEquation*(a, Vector.*copyOf*(f)).subtract(xPrecise).findNorm());  
  
 // Шаг 2 (шаг 1 пропускается, т.к. первая "испорченная" диагональ на верхней строке матрицы)  
 for (int i = size - 1; i > 1; --i) {  
  
 var r = 1 / sideDiagonal.getValueAt(i);  
 sideDiagonal.setValueAt(i, 1.0);  
 rSideDiagonal.multiplyValueAt(i, r);  
 f.multiplyValueAt(i, r);  
  
 if (i > 2) {  
 r = lSideDiagonal.getValueAt(i - 1);  
 lSideDiagonal.setValueAt(i - 1, 0.0);  
 sideDiagonal.subtractValueAt(i - 1, r \* rSideDiagonal.getValueAt(i));  
 f.subtractValueAt(i - 1, r \* f.getValueAt(i));  
 }  
  
 r = dCrackedLine.getValueAt(size - 1 - i);  
 dCrackedLine.setValueAt(size - 1 - i, 0.0);  
 dCrackedLine.subtractValueAt(size - i, r \* rSideDiagonal.getValueAt(i));  
 f.subtractValueAt(1, r \* f.getValueAt(i));  
  
 r = uCrackedLine.getValueAt(size - 1 - i);  
 uCrackedLine.setValueAt(size - 1 - i, 0.0);  
 uCrackedLine.subtractValueAt(size - i, r \* rSideDiagonal.getValueAt(i));  
 f.subtractValueAt(0, r \* f.getValueAt(i));  
  
 System.*out*.println("Погрешность на этапе " + (size - i) + " составляет: " +  
 MatrixOperationUnit.*solveEquation*(a, Vector.*copyOf*(f)).subtract(xPrecise).findNorm());  
 }  
  
 var r = 1 / dCrackedLine.getValueAt(size - 2);  
 dCrackedLine.setValueAt(size - 2, 1.0);  
 dCrackedLine.multiplyValueAt(size - 1, r);  
 f.multiplyValueAt(1, r);  
  
 r = uCrackedLine.getValueAt(size - 2);  
 uCrackedLine.setValueAt(size - 2, 0.0);  
 uCrackedLine.subtractValueAt(size - 1, r \* dCrackedLine.getValueAt(size - 1));  
 f.subtractValueAt(0, r \* f.getValueAt(1));  
  
 r = 1 / uCrackedLine.getValueAt(size - 1);  
 uCrackedLine.setValueAt(size - 1, 1.0);  
 f.multiplyValueAt(0, r);  
  
 // Значения "cracked lines" верны, осталось подогнать под них диагонали 0 и 1 рядов  
 lSideDiagonal.setValueAt(0, 0.0);  
 sideDiagonal.setValueAt(0, uCrackedLine.getValueAt(size - 1));  
 sideDiagonal.setValueAt(1, dCrackedLine.getValueAt(size - 2));  
 rSideDiagonal.setValueAt(1, dCrackedLine.getValueAt(size - 1));  
  
 System.*out*.println("Погрешность на последнем этапе составляет: " +  
 MatrixOperationUnit.*solveEquation*(a, Vector.*copyOf*(f)).subtract(xPrecise).findNorm());  
  
 var x = new Vector(size);  
  
 x.setValueAt(size - 1, f.getValueAt(0));  
 for (int i = size - 2; i >= 0; --i) {  
 x.setValueAt(  
 i,  
 f.getValueAt(size - 1 - i) - rSideDiagonal.getValueAt(size - 1 - i) \* x.getValueAt(i + 1)  
 );  
 }  
  
 System.*out*.println("\nОкончательная погрешность составляет: " + x.subtract(xPrecise).findNorm() + "\n");  
 }  
}