**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по практической работе №6**

**по дисциплине «Вычислительная математика»**

**Тема: Исследование обусловленности задачи решения систем линейных уравнений**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Гайфутдинов А.Р. |
| Преподаватель |  | Попова Е.В. |

Санкт-Петербург

2025

**Цель работы:** Изучение стандартной обусловленности задач решения систем линейных уравнений при различных вариантах неточных входных данных.

**Основные теоретические положения.**

Рассматривается система линейных уравнений n-го порядка с вещественными коэффициентами (1)

В матричной форме записи эта система принимает вид (2)

, (2)

где – квадратная матрица коэффициентов системы, – вектор решений системы, – вектор свободных членов. Матрица – невырожденная, тогда решение системы (1) существует, единственно и устойчиво по входным данным. Это означает, что задача нахождения вектора – корректна.

Пусть – приближенное решение системы, тогда называется вектором погрешности системы, необходимо стремиться к его уменьшению. Возможно рассматривать критерий малости вектора который называется невязкой системы. Эти вектора связаны **.**

Удобной количественной характеристикой вектора является норма вектора. В вычислительной математике используются следующие три нормы (3)

(3)

За норму матрицы принимают максимальную величину, на которую преобразование, описываемое матрицей, может растянуть любой ненулевой вектор в выбранной норме . Векторным нормам подчинены следующие нормы матрицы (4)

где – собственные числа матрицы Задача вычисления вектора может быть плохо или хорошо обусловлена.

**Обусловленность задачи решения систем линейных алгебраических уравнений**

Рассмотрим случай, когда элементы матрицы заданы точно, а вектор-столбец свободных членов – приближенно. Оценки для абсолютной и относительной погрешности (5)

где - абсолютное число обусловленности, а - относительное число обусловленности (естественное число обусловленности). Максимальное естественное число обусловленности (6)

(6)

называют стандартным числом обусловленности.

Если элементы матрицы заданы приближенно и равны , а вектор-столбец свободных членов – точно, тогда оценка относительной погрешности (7)

(7)

где и .

Если с погрешностью заданы как коэффициенты матрицы, так и элементы вектора свободных членов, то справедливо неравенство (8)

**(8)**

**Порядок выполнения работы.**

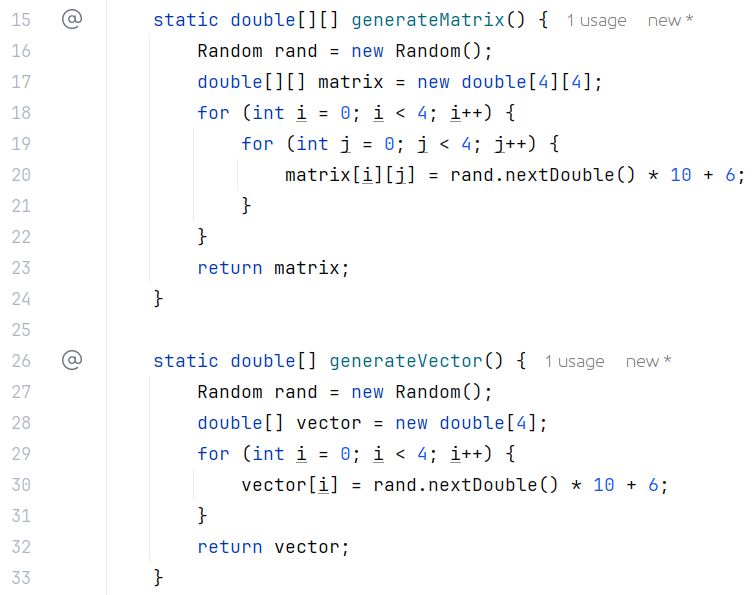
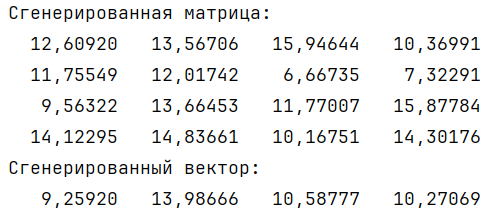
**Выучить теорию по данной теме.**

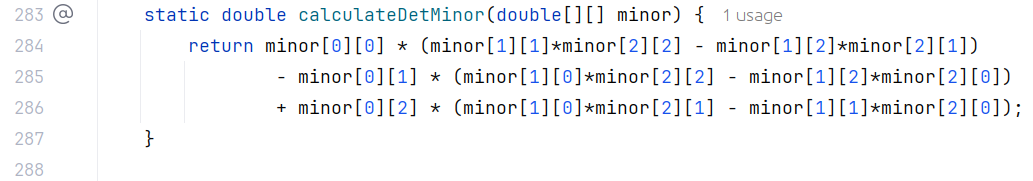
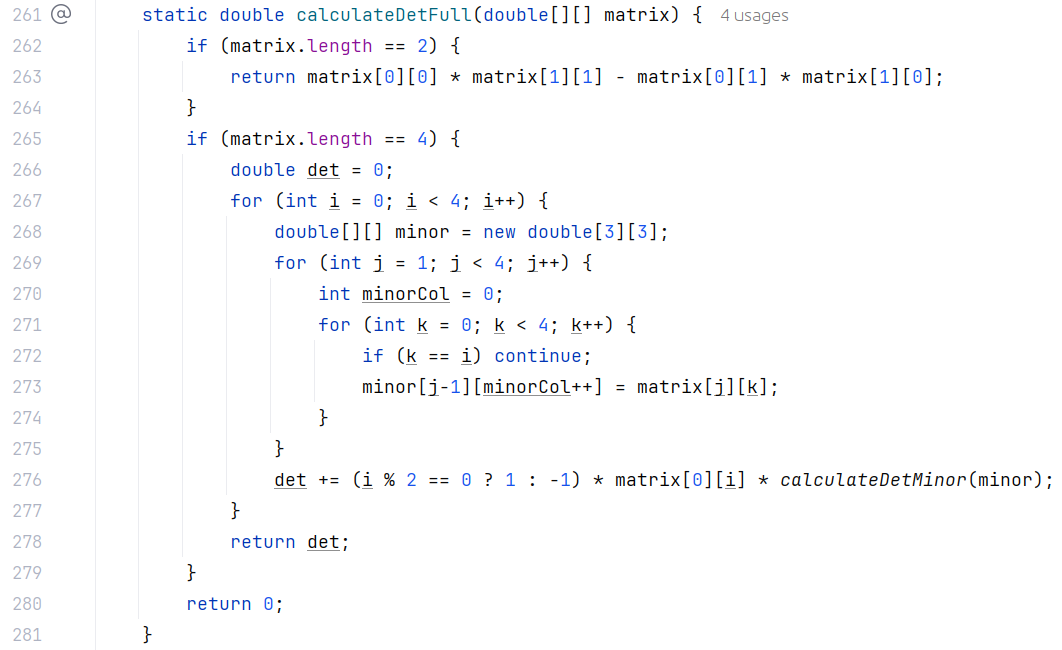
1. Сформировать расширенную входную матрицу, соответствующую 4 неизвестным и 4 уравнениям, с нецелыми коэффициентами, положительными и отрицательными. При генерации элементов матрицы, в методе random()языка java использовать в качестве слагаемого номер в списке. Проверить невырожденность матрицы.
2. Составить подпрограмму для решения системы линейных уравнений методом Крамера для четных номеров, использовать в работе нормы №1; и методом обратной матрицы для нечетных номеров, использовать в работе нормы . Объяснить метод в работе и сопоставить выкладки с полученным кодом.
3. Решить систему, подсчитать абсолютное, естественное, стандартное число обусловленности.
4. Добавить ошибки в вектор свободных членов. Найти решение новой системы, абсолютное, естественное, стандартное число обусловленности (6) и оценку стандартного числа обусловленности (7).
5. Добавить ошибки в значения элементов матрицы. Найти решение новой системы, абсолютное, естественное, стандартное число обусловленности и оценку стандартного числа обусловленности.
6. Добавить ошибки в значения элементов матрицы и вектора свободных членов. Найти решение новой системы, абсолютное, естественное, стандартное число обусловленности и оценку стандартного числа обусловленности.
7. Заменить элементы главной диагонали в сформированной в пункте 1 матрице на элементы матрицы Гильберта () для четных номеров, и элементы побочной диагонали для нечетных номеров. Проделать все 6 предыдущих пунктов для второй матрицы.
8. Сформировать итоговую таблицу: по столбцам 2 матрицы, по строкам результаты вычислений. Сделать выводы по полученным результатам.

**Выполнение работы.**

1. *Сформировать расширенную входную матрицу, соответствующую 4 неизвестным и 4 уравнениям, с нецелыми коэффициентами, положительными и отрицательными. При генерации элементов матрицы, в методе random()языка java использовать в качестве слагаемого номер в списке. Проверить невырожденность матрицы.*

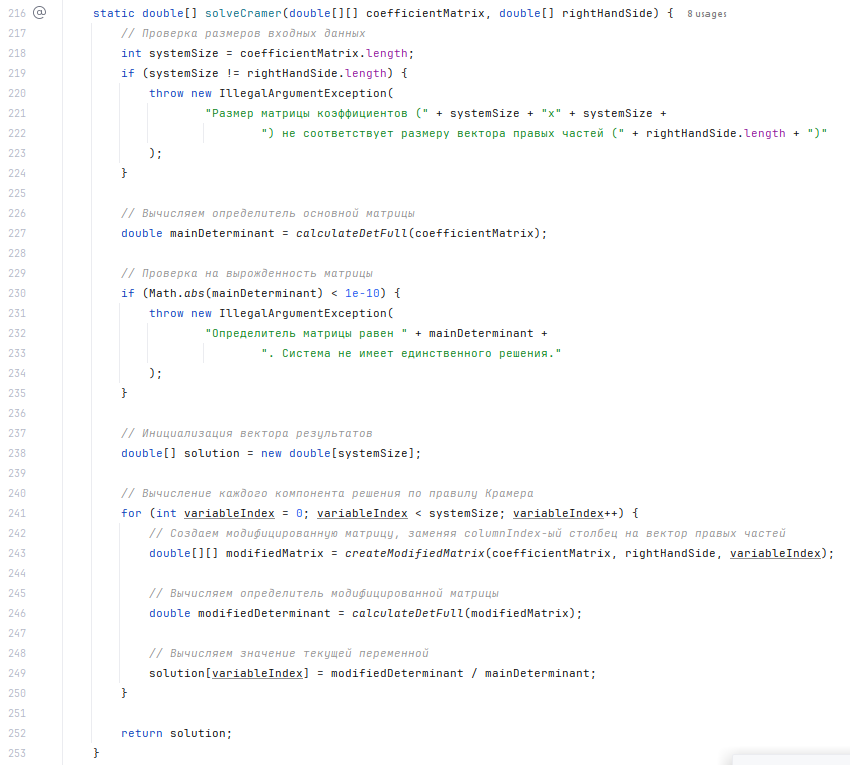
Для генерации матрицы были реализованы соответствующие методы:

Для проверки корректности сгенерированной матрицы для дальнейшей работы были реализованы методы по вычислению определителя матрицы:

Так как определитель матрицы != 0, матрица не является вырожденной, следовательно, подходит для дальнейшей обработки.

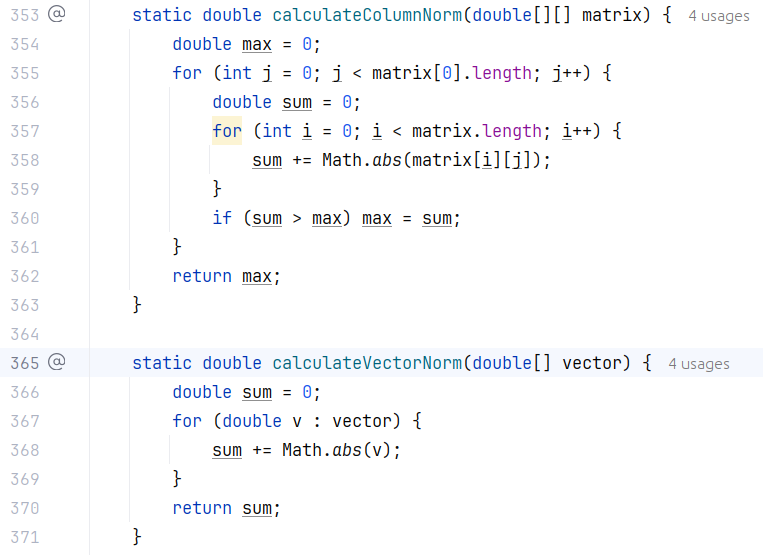
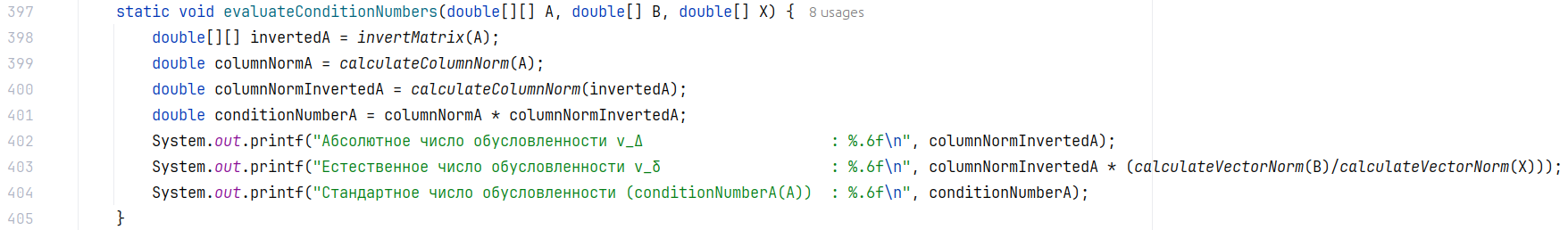
1. *Составить подпрограмму для решения системы линейных уравнений методом Крамера для четных номеров, использовать в работе нормы №1; и методом обратной матрицы для нечетных номеров, использовать в работе нормы . Объяснить метод в работе и сопоставить выкладки с полученным кодом.*

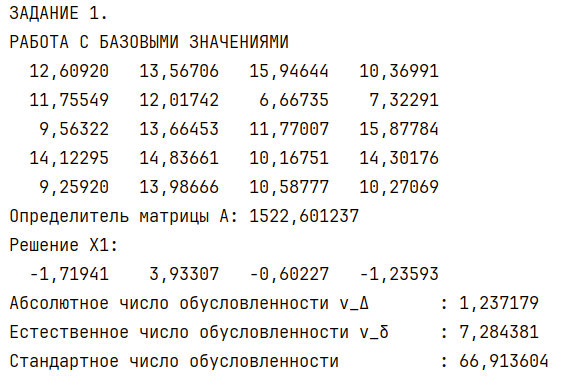
*Метод 1 решает систему линейных уравнений методом Крамера:*

*Метод 2 создает модифицированную матрицу для метода Крамера, заменяя указанный столбец на вектор правых частей:*

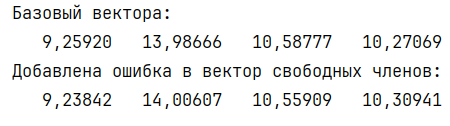
1. *Решить систему, подсчитать абсолютное, естественное, стандартное число обусловленности.*

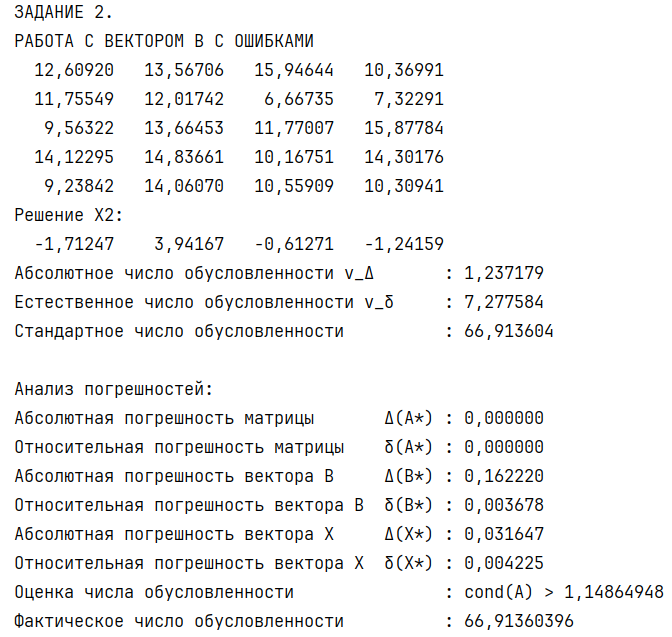
Система решена при помощи реализованного метода Крамера путем последовательной. Подсчет чисел обусловленности реализован в отдельном методе, как и вычисление норм вектора и матрицы:

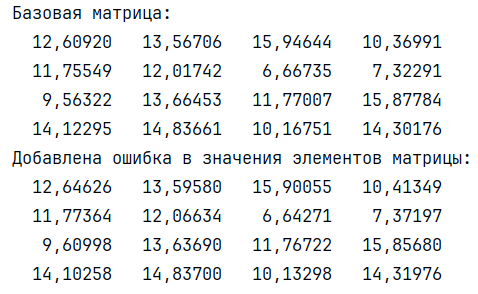


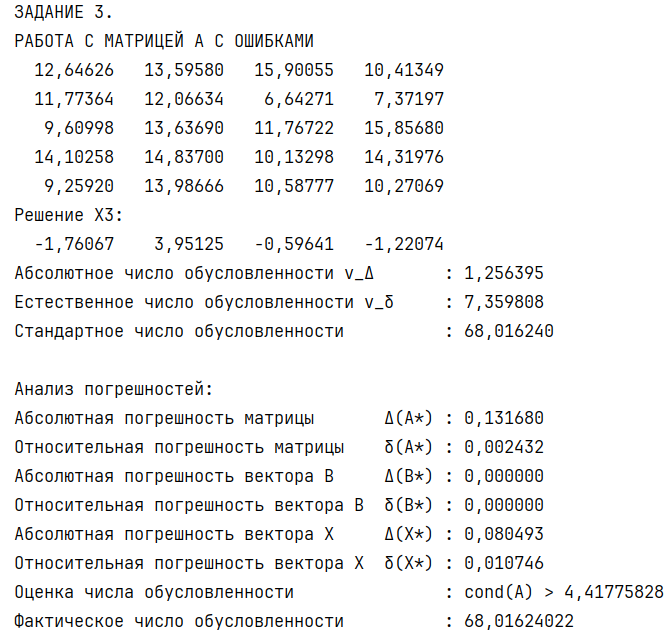
**

1. *Добавить ошибки в* ***вектор свободных членов****. Найти решение новой системы, абсолютное, естественное, стандартное число обусловленности (6) и оценку стандартного числа обусловленности (7).*

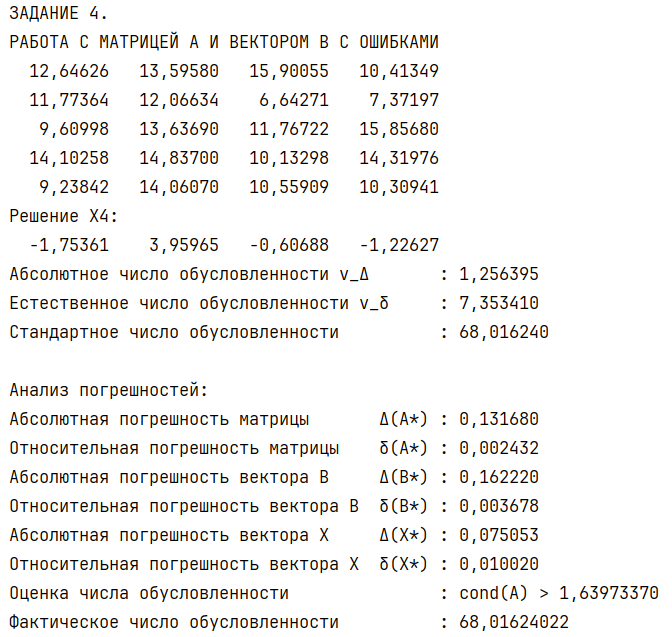
Для добавления ошибки в вектор свободных членов реализована отдельная функция. Ошибка находится в интервале [-0.05; 0.05].

1. *Добавить ошибки в* ***значения элементов матрицы****. Найти решение новой системы, абсолютное, естественное, стандартное число обусловленности и оценку стандартного числа обусловленности.*

Для добавления ошибки в вектор свободных членов реализована отдельная функция. Ошибка находится в интервале [-0.05; 0.05].

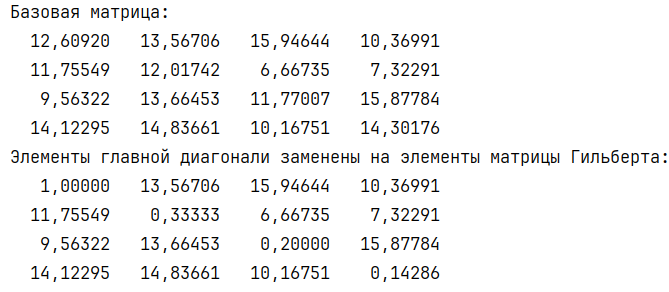
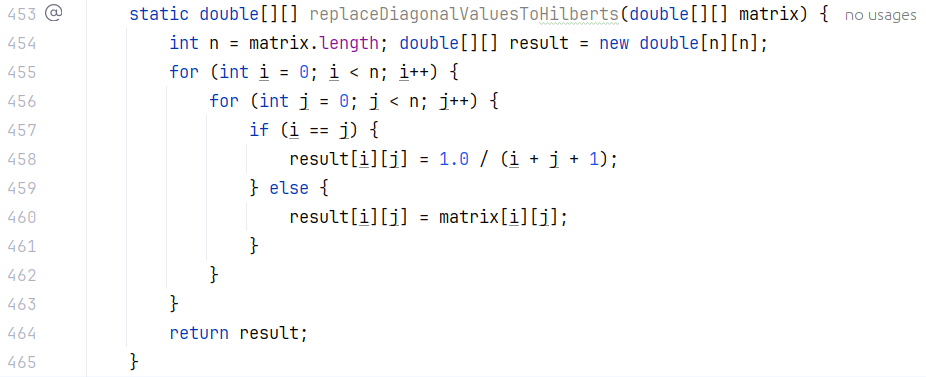
**

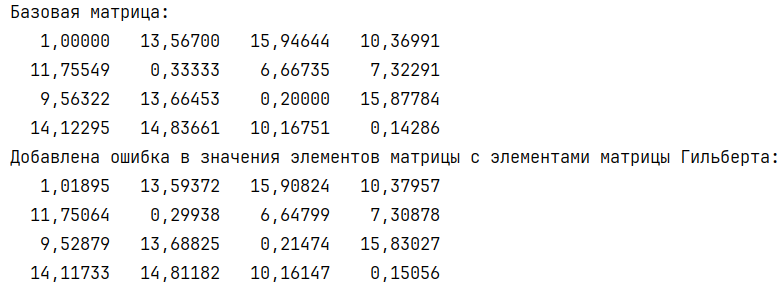
1. *Добавить ошибки в* ***значения элементов матрицы*** *и* ***вектора свободных членов****. Найти решение новой системы, абсолютное, естественное, стандартное число обусловленности и оценку стандартного числа обусловленности.*

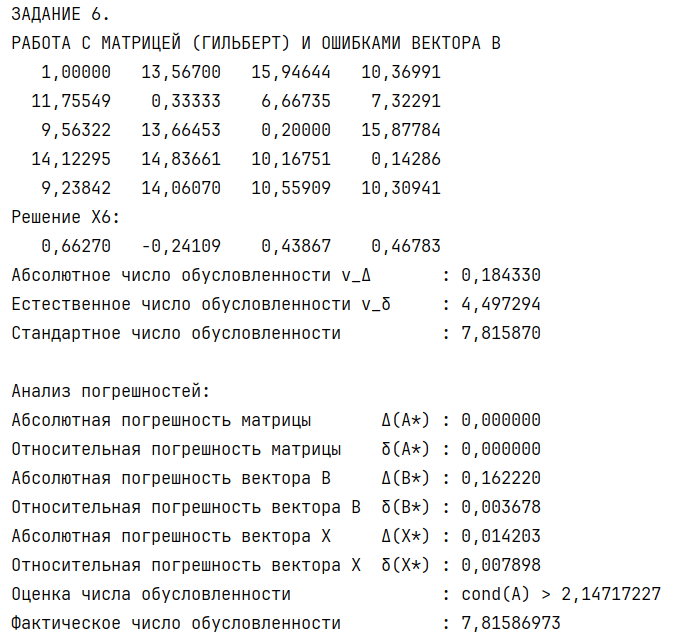
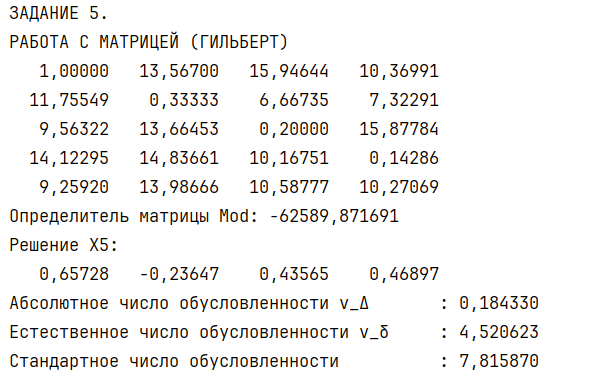
**

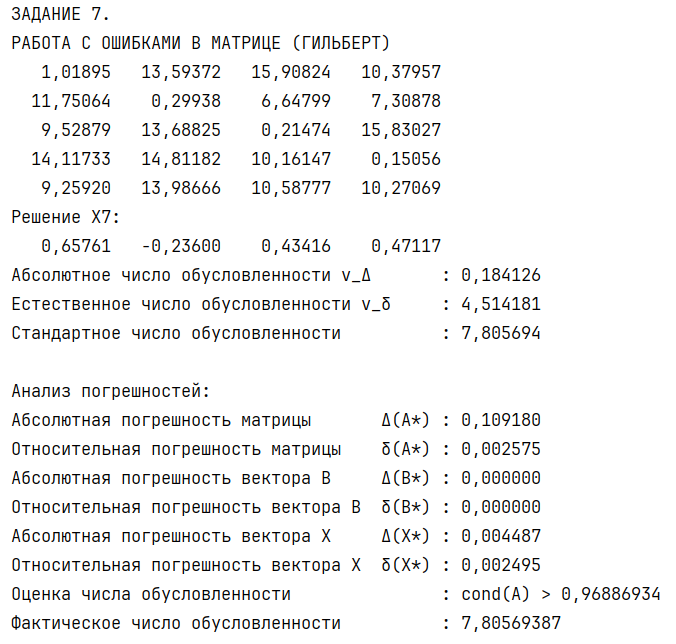
1. *Заменить элементы главной диагонали в сформированной в пункте 1 матрице на элементы матрицы Гильберта () для четных номеров, и элементы побочной диагонали для нечетных номеров. Проделать все 6 предыдущих пунктов для второй матрицы.*

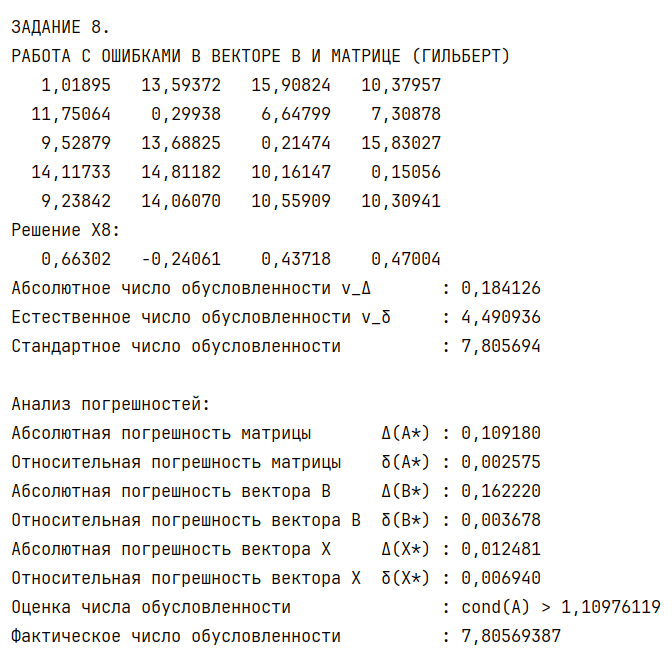
Для замены элементов главной диагонали матрицы на элементы матрицы Гильберта реализован следующий метод:

Добавлены ошибки в матрицу с элементами матрицы Гильберта:









1. Сформировать итоговую таблицу: по столбцам 2 матрицы, по строкам результаты вычислений. Сделать выводы по полученным результатам.

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Исходная матрица** | **Матрица с элементами Гильберта на диагонали** |
| Относительная погрешность матрицы A | B\*: 0.000000  А\*: 0,002432  A\* и B\*: 0,002432 | B\*: 0.000000  А\*: 0,002575  A\* и B\*: 0,002575 |
| Относительная погрешность вектора В | B\*: 0,003678  А\*: 0.000000  A\* и B\*: 0,003678 | B\*: 0,003678  А\*: 0.000000  A\* и B\*: 0,003678 |
| Относительная погрешность вектора Х | B\*: 0,004225  А\*: 0,010746  A\* и B\*: 0,010020 | B\*: 0,007898  А\*: 0,002495  A\* и B\*: 0,006940 |
| Абсолютное число обусловленности | B\*: 1,237179  А\*: 1,256395  A\* и B\*: 1,256395 | B\*: 0,184330  А\*: 0,184126  A\* и B\*: 0,184126 |
| Естественное число обусловленности | B\*: 7,277584  А\*: 7,359808  A\* и B\*: 7,353410 | B\*: 4,497294  А\*: 4,514181  A\* и B\*: 4,490936 |
| Стандартное число обусловленности  cond(A) | B\*: 66,913604  А\*: 68,016240  A\*и B\*: 68,016240 | B\*: 7,815870  А\*: 7,805694  A\* и B\*: 7,805694 |
| Оценка снизу | B\*: cond(A) > 1,14864948  А\*: cond(A) > 4,41775828  A\* и B\*: cond(A) > 1,63973370 | B\*: cond(A) > 2,14717227  А\*: cond(A) > 0,96886934  A\* и B\*: cond(A) > 1,10976119 |

Исходя из данных таблицы можно сделать несколько выводов:

1. Небольшие возмущения в векторе свободных членов и матрице по-разному сказываются на оценке относительной погрешности решения. Так, ошибки в векторе свободных членов приводят к большей относительной погрешности по сравнению с теми же ошибками в матрице коэффициентов.
2. Абсолютное число обусловленности оценивает, как ошибки в матрице системы и векторе свободных членов влияют на абсолютную погрешность решения. Чем больше данное значение, тем хуже обусловлена данная задача. В среднем, данное значение при работе с исзодной матрице равно 1.25, в случае же работы с модифицированной матрицей равно 0,18, что говорит о большей устойчивости системы с модифицированной матрицей к погрешностям.
3. Стандартное число обусловленности оценивает возможное усиление относительной погрешности при возмущении в данных. Данное значение в экспериментах с модифицированной матрицей принимает почти на порядок меньшее значение по сравнению с экспериментами в базовой матрице (7,8 против 68,1 по верхней оценке), что говорит о лучшей точности при работе с модифицированной матрицей. Позволяет оценить худший случай.
4. Естественное число обусловленности всегда меньше стандартного числа обусловленности, позволяет более точно оценить чувствительность задачи к возмущениям по сравнению со стандартным числом. Как и в случае со стандартным числом обусловленности, ествественное число меньше в экспериментах с модифицированной матрицей по сравнению с базовой (4.5 против 7.4), что так же показывает меньшую чувствительность системы, где элементы главной диагонали - элементы матрицы Гильберта.

**Анализируя выше указанные выводы, можно сделать следующее заключение:**

Система, в которой матрица коэффициентов представлена матрицей, где элементы главной диагонали - элементы оной у Гильбертовой матрицы, обладает лучшей устойчивостью к погрешностям в исходных данных как в худшем случае (согласно показателю стандартного числа обусловленности), так и в среднем (согласно показателю естественного числа обусловленности).

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Файл Util.java:

package com.github.chelovekkrokant;  
  
import java.util.Random;  
  
public class Util {  
  
 *// Исходная матрица А*  
public static double[][] *A* = {  
 {12.60920, 13.56706, 15.94644, 10.36991},  
 {11.75549, 12.01742, 6.66735, 7.32291},  
 {9.56322, 13.66453, 11.77007, 15.87784},  
 {14.12295, 14.83661, 10.16751, 14.30176}  
 };  
  
 *// Матрица А с ошибкой 0.05*  
public static double[][] *A\_WITH\_ERROR* = {  
 {12.64626, 13.59580, 15.90055, 10.41349},  
 {11.77364, 12.06634, 6.64271, 7.37197},  
 {9.60998, 13.63690, 11.76722, 15.85680},  
 {14.10258, 14.83700, 10.13298, 14.31976}  
 };  
  
 *// Исходная матрица А c элементами Матрицы Гильберта*  
public static double[][] *A\_HILBERT* = {  
 {1.00000, 13.5670, 15.94644, 10.36991},  
 {11.75549, 0.33333, 6.66735, 7.32291},  
 {9.56322, 13.66453, 0.20000, 15.87784},  
 {14.12295, 14.83661, 10.16751, 0.14286}  
 };  
  
 *// Исходная матрица А c элементами Матрицы Гильберта с ошибкой 0.05*  
public static double[][] *A\_HILBERT\_WITH\_ERROR* = {  
 {1.01895, 13.59372, 15.90824, 10.37957},  
 {11.75064, 0.29938, 6.64799, 7.30878},  
 {9.52879, 13.68825, 0.21474, 15.83027},  
 {14.11733, 14.81182, 10.16147, 0.15056}  
 };  
  
 *// Исходный вектор B*  
public static double[] *B* = {  
 9.25920, 13.98666, 10.58777, 10.27069  
 };  
  
 *// Вектор B с ошибкой 0.05*  
public static double[] *B\_WITH\_ERROR* = {  
 9.23842, 14.0607, 10.55909, 10.30941  
 };  
  
 public static void main(String[] args){  
 *addErrorHilbertMatrix*();  
 }  
  
 static void findDet(){  
 System.*out*.println("Определитель сгенерированной матрицы = " + *calculateDetFull*(*A*));  
 }  
  
 static void generate(){  
 System.*out*.println("Сгенерированная матрица:");  
 *printMatrix*(*generateMatrix*());  
  
 System.*out*.println("Сгенерированный вектор:");  
 *printVector*(*generateVector*());  
 }  
  
 static void addErrorVector(){  
 System.*out*.println("Базовый вектора:");  
 *printVector*(*B*);  
 System.*out*.println("Добавлена ошибка в вектор свободных членов:");  
 *printVector*(*addErrorToVector*(*B*));  
 }  
  
 static void addErrorMatrix(){  
 System.*out*.println("Базовая матрица:");  
 *printMatrix*(*A*);  
 System.*out*.println("Добавлена ошибка в значения элементов матрицы:");  
 *printMatrix*(*addErrorToMatrix*(*A*));  
 }  
  
 static void addErrorHilbertMatrix(){  
 System.*out*.println("Базовая матрица:");  
 *printMatrix*(*A\_HILBERT*);  
 System.*out*.println("Добавлена ошибка в значения элементов матрицы с элементами матрицы Гильберта:");  
 *printMatrix*(*addErrorToMatrix*(*A\_HILBERT*));  
 }  
  
 static void replaceDiagonalValues(){  
 System.*out*.println("Базовая матрица:");  
 *printMatrix*(*A*);  
 System.*out*.println("Элементы главной диагонали заменены на элементы матрицы Гильберта:");  
 *printMatrix*(*replaceDiagonalValuesToHilberts*(*A*));  
 }  
  
 static double[][] replaceDiagonalValuesToHilberts(double[][] matrix) {  
 int n = matrix.length; double[][] result = new double[n][n];  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 if (i == j) {  
 result[i][j] = 1.0 / (i + j + 1);  
 } else {  
 result[i][j] = matrix[i][j];  
 }  
 }  
 }  
 return result;  
 }  
  
 static double[][] addErrorToMatrix(double[][] matrix) {  
 Random rand = new Random();  
 double[][] result = *deepCopy*(matrix);  
 for (int i = 0; i < result.length; i++) {  
 for (int j = 0; j < result[0].length; j++) {  
 result[i][j] += 0.1 \* (rand.nextDouble() - 0.5);  
 }  
 }  
 return result;  
 }  
  
 static double[] addErrorToVector(double[] vector) {  
 Random rand = new Random();  
 double[] result = vector.clone();  
 for (int i = 0; i < result.length; i++) {  
 result[i] += 0.1 \* (rand.nextDouble() - 0.5);  
 }  
 return result;  
 }  
  
 static double[][] generateMatrix() {  
 Random rand = new Random();  
 double[][] matrix = new double[4][4];  
 for (int i = 0; i < 4; i++) {  
 for (int j = 0; j < 4; j++) {  
 matrix[i][j] = rand.nextDouble() \* 10 + 6;  
 }  
 }  
 return matrix;  
 }  
  
 static double[] generateVector() {  
 Random rand = new Random();  
 double[] vector = new double[4];  
 for (int i = 0; i < 4; i++) {  
 vector[i] = rand.nextDouble() \* 10 + 6;  
 }  
 return vector;  
 }  
  
 static void printMatrix(double[][] matrix) {  
 for (double[] row : matrix) {  
 for (double value : row) {  
 System.*out*.printf("%10.5f ", value);  
 }  
 System.*out*.println();  
 }  
 }  
  
 static void printVector(double[] vector) {  
 for (double v : vector) {  
 System.*out*.printf("%10.5f ", v);  
 }  
 System.*out*.println();  
 }  
  
 static double[][] deepCopy(double[][] matrix) {  
 double[][] copy = new double[matrix.length][matrix[0].length];  
 for (int i = 0; i < matrix.length; i++) {  
 System.*arraycopy*(matrix[i], 0, copy[i], 0, matrix[0].length);  
 }  
 return copy;  
 }  
  
 static double calculateDetFull(double[][] matrix) {  
 if (matrix.length == 2) {  
 return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0];  
 }  
 if (matrix.length == 4) {  
 double det = 0;  
 for (int i = 0; i < 4; i++) {  
 double[][] minor = new double[3][3];  
 for (int j = 1; j < 4; j++) {  
 int minorCol = 0;  
 for (int k = 0; k < 4; k++) {  
 if (k == i) continue;  
 minor[j-1][minorCol++] = matrix[j][k];  
 }  
 }  
 det += (i % 2 == 0 ? 1 : -1) \* matrix[0][i] \* *calculateDetMinor*(minor);  
 }  
 return det;  
 }  
 return 0;  
 }  
  
 static double calculateDetMinor(double[][] minor) {  
 return minor[0][0] \* (minor[1][1]\*minor[2][2] - minor[1][2]\*minor[2][1])  
 - minor[0][1] \* (minor[1][0]\*minor[2][2] - minor[1][2]\*minor[2][0])  
 + minor[0][2] \* (minor[1][0]\*minor[2][1] - minor[1][1]\*minor[2][0]);  
 }  
  
}

Файл SLAU.java:

package com.github.chelovekkrokant;  
  
import static com.github.chelovekkrokant.Util.\*;  
  
public class SLAU {  
   
 public static void main(String[] args) {  
 System.*out*.println("\n\nЗАДАНИЕ 1.\nРАБОТА С БАЗОВЫМИ ЗНАЧЕНИЯМИ");  
 *printMatrix*(*A*);  
 *printVector*(*B*);  
 *// Проверяем невырожденность*  
System.*out*.printf("Определитель матрицы A: %f\n", *calculateDetFull*(*A*));  
 if (Math.*abs*(*calculateDetFull*(*A*)) < 1e-8) {  
 System.*out*.println("Матрица вырождена. Решение невозможно.");  
 return;  
 }  
 *// Решение методом обратной матрицы*  
double[] X = *solveCramer*(*A*, *B*);  
 System.*out*.println("Решение X1:");  
 Util.*printVector*(X);  
 *// Оцениваем числа обусловленности*  
 *evaluateConditionNumbers*(*A*, *B*, X);  
 */// ///////////////////////////////////////////////////////////////////*  
  
  
System.*out*.println("\n\nЗАДАНИЕ 2.\nРАБОТА С ВЕКТОРОМ В С ОШИБКАМИ");  
 *printMatrix*(*A*);  
 *printVector*(*B\_WITH\_ERROR*);  
 double[] X\_star = *solveCramer*(*A*, *B\_WITH\_ERROR*);  
 System.*out*.println("Решение X2:");  
 Util.*printVector*(X\_star);  
 *evaluateConditionNumbers*(*A*, *B\_WITH\_ERROR*, X\_star);  
 *calculateAllErrorsAndCondition*(  
 *A*, *A*,  
 *B*, *B\_WITH\_ERROR*,  
 X, X\_star, 1);  
 */// ///////////////////////////////////////////////////////////////////*  
  
  
System.*out*.println("\n\nЗАДАНИЕ 3.\nРАБОТА С МАТРИЦЕЙ А С ОШИБКАМИ");  
 *printMatrix*(*A\_WITH\_ERROR*);  
 *printVector*(*B*);  
 X\_star = *solveCramer*(*A\_WITH\_ERROR*, *B*);  
 System.*out*.println("Решение X3:");  
 Util.*printVector*(X\_star);  
 *evaluateConditionNumbers*(*A\_WITH\_ERROR*, *B*, X\_star);  
 *calculateAllErrorsAndCondition*(  
 *A*, *A\_WITH\_ERROR*,  
 *B*, *B*,  
 X, X\_star, 2);  
 */// ///////////////////////////////////////////////////////////////////*  
  
  
System.*out*.println("\n\nЗАДАНИЕ 4.\nРАБОТА С МАТРИЦЕЙ А И ВЕКТОРОМ В С ОШИБКАМИ");  
 *printMatrix*(*A\_WITH\_ERROR*);  
 *printVector*(*B\_WITH\_ERROR*);  
 X\_star = *solveCramer*(*A\_WITH\_ERROR*, *B\_WITH\_ERROR*);  
 System.*out*.println("Решение X4:");  
 Util.*printVector*(X\_star);  
 *evaluateConditionNumbers*(*A\_WITH\_ERROR*, *B\_WITH\_ERROR*, X\_star);  
 *calculateAllErrorsAndCondition*(  
 *A*, *A\_WITH\_ERROR*,  
 *B*, *B\_WITH\_ERROR*,  
 X, X\_star, 3);  
 */// ///////////////////////////////////////////////////////////////////*  
  
  
System.*out*.println("\n\nЗАДАНИЕ 5.\nРАБОТА С МАТРИЦЕЙ (ГИЛЬБЕРТ)");  
 *printMatrix*(*A\_HILBERT*);  
 *printVector*(*B*);  
 System.*out*.printf("Определитель матрицы Mod: %f\n", *calculateDetFull*(*A\_HILBERT*));  
 if (Math.*abs*(*calculateDetFull*(*A\_HILBERT*)) < 1e-8) {  
 System.*out*.println("Модифицированная матрица вырождена. Решение невозможно.");  
 return;  
 }  
 X = *solveCramer*(*A\_HILBERT*, *B*);  
 System.*out*.println("Решение X5:");  
 Util.*printVector*(X);  
 *evaluateConditionNumbers*(*A\_HILBERT*, *B*, X);  
 */// ///////////////////////////////////////////////////////////////////*  
  
  
System.*out*.println("\n\nЗАДАНИЕ 6.\nРАБОТА С МАТРИЦЕЙ (ГИЛЬБЕРТ) И ОШИБКАМИ ВЕКТОРА В");  
 *printMatrix*(*A\_HILBERT*);  
 *printVector*(*B\_WITH\_ERROR*);  
 X\_star = *solveCramer*(*A\_HILBERT*, *B\_WITH\_ERROR*);  
 System.*out*.println("Решение X6:");  
 Util.*printVector*(X\_star);  
 *evaluateConditionNumbers*(*A\_HILBERT*, *B\_WITH\_ERROR*, X\_star);  
 *calculateAllErrorsAndCondition*(  
 *A\_HILBERT*, *A\_HILBERT*,  
 *B*, *B\_WITH\_ERROR*,  
 X, X\_star, 1);  
 */// ///////////////////////////////////////////////////////////////////*  
  
  
System.*out*.println("\n\nЗАДАНИЕ 7.\nРАБОТА С ОШИБКАМИ В МАТРИЦЕ (ГИЛЬБЕРТ)");  
 *printMatrix*(*A\_HILBERT\_WITH\_ERROR*);  
 *printVector*(*B*);  
 X\_star = *solveCramer*(*A\_HILBERT\_WITH\_ERROR*, *B*);  
 System.*out*.println("Решение X7:");  
 Util.*printVector*(X\_star);  
 *evaluateConditionNumbers*(*A\_HILBERT\_WITH\_ERROR*, *B*, X\_star);  
 *calculateAllErrorsAndCondition*(  
 *A\_HILBERT*, *A\_HILBERT\_WITH\_ERROR*,  
 *B*, *B*,  
 X, X\_star, 2);  
 */// ///////////////////////////////////////////////////////////////////*  
  
  
System.*out*.println("\n\nЗАДАНИЕ 8.\nРАБОТА С ОШИБКАМИ В ВЕКТОРЕ В И МАТРИЦЕ (ГИЛЬБЕРТ)");  
 *printMatrix*(*A\_HILBERT\_WITH\_ERROR*);  
 *printVector*(*B\_WITH\_ERROR*);  
 X\_star = *solveCramer*(*A\_HILBERT\_WITH\_ERROR*, *B\_WITH\_ERROR*);  
 System.*out*.println("Решение X8:");  
 Util.*printVector*(X\_star);  
 *evaluateConditionNumbers*(*A\_HILBERT\_WITH\_ERROR*, *B\_WITH\_ERROR*, X\_star);  
 *calculateAllErrorsAndCondition*(  
 *A\_HILBERT*, *A\_HILBERT\_WITH\_ERROR*,  
 *B*, *B\_WITH\_ERROR*,  
 X, X\_star, 3);  
 }  
  
 */\*\**  
 *\* Решает систему линейных уравнений методом Крамера*  
 *\**  
 *\* @param coefficientMatrix матрица коэффициентов системы (n x n)*  
 *\* @param rightHandSide вектор правых частей системы (длина n)*  
 *\* @return вектор решения системы*  
 *\* @throws IllegalArgumentException если матрица вырождена (определитель = 0)*  
 *\*/*  
static double[] solveCramer(double[][] coefficientMatrix, double[] rightHandSide) {  
 *// Проверка размеров входных данных*  
int systemSize = coefficientMatrix.length;  
 if (systemSize != rightHandSide.length) {  
 throw new IllegalArgumentException(  
 "Размер матрицы коэффициентов (" + systemSize + "x" + systemSize +  
 ") не соответствует размеру вектора правых частей (" + rightHandSide.length + ")"  
 );  
 }  
  
 *// Вычисляем определитель основной матрицы*  
double mainDeterminant = *calculateDetFull*(coefficientMatrix);  
  
 *// Проверка на вырожденность матрицы*  
if (Math.*abs*(mainDeterminant) < 1e-10) {  
 throw new IllegalArgumentException(  
 "Определитель матрицы равен " + mainDeterminant +  
 ". Система не имеет единственного решения."  
 );  
 }  
  
 *// Инициализация вектора результатов*  
double[] solution = new double[systemSize];  
  
 *// Вычисление каждого компонента решения по правилу Крамера*  
for (int variableIndex = 0; variableIndex < systemSize; variableIndex++) {  
 *// Создаем модифицированную матрицу, заменяя columnIndex-ый столбец на вектор правых частей*  
double[][] A\_HILBERT = *createModifiedMatrix*(coefficientMatrix, rightHandSide, variableIndex);  
  
 *// Вычисляем определитель модифицированной матрицы*  
double modifiedDeterminant = *calculateDetFull*(A\_HILBERT);  
  
 *// Вычисляем значение текущей переменной*  
solution[variableIndex] = modifiedDeterminant / mainDeterminant;  
 }  
  
 return solution;  
 }  
  
 */\*\**  
 *\* Создает модифицированную матрицу для метода Крамера, заменяя указанный столбец на вектор правых частей*  
 *\**  
 *\* @param originalMatrix исходная матрица коэффициентов*  
 *\* @param rightHandSide вектор правых частей*  
 *\* @param columnToReplace индекс столбца для замены*  
 *\* @return модифицированная матрица*  
 *\*/*  
private static double[][] createModifiedMatrix(double[][] originalMatrix, double[] rightHandSide, int columnToReplace) {  
 int size = originalMatrix.length;  
 double[][] A\_HILBERT = new double[size][size];  
  
 *// Копируем исходную матрицу, заменяя указанный столбец*  
for (int row = 0; row < size; row++) {  
 System.*arraycopy*(originalMatrix[row], 0, A\_HILBERT[row], 0, size);  
 A\_HILBERT[row][columnToReplace] = rightHandSide[row];  
 }  
  
 return A\_HILBERT;  
 }  
   
  
 static double[][] invertMatrix(double[][] matrix) {  
 int n = matrix.length;  
 double[][] a = Util.*deepCopy*(matrix);  
 double[][] inv = new double[n][n];  
  
 for (int i = 0; i < n; i++) inv[i][i] = 1.0;  
  
 for (int i = 0; i < n; i++) {  
 double diag = a[i][i];  
 for (int j = 0; j < n; j++) {  
 a[i][j] /= diag;  
 inv[i][j] /= diag;  
 }  
 for (int k = 0; k < n; k++) {  
 if (k != i) {  
 double factor = a[k][i];  
 for (int j = 0; j < n; j++) {  
 a[k][j] -= factor \* a[i][j];  
 inv[k][j] -= factor \* inv[i][j];  
 }  
 }  
 }  
 }  
 return inv;  
 }  
  
 static double calculateColumnNorm(double[][] matrix) {  
 double max = 0;  
 for (int j = 0; j < matrix[0].length; j++) {  
 double sum = 0;  
 for (int i = 0; i < matrix.length; i++) {  
 sum += Math.*abs*(matrix[i][j]);  
 }  
 if (sum > max) max = sum;  
 }  
 return max;  
 }  
  
 static double calculateVectorNorm(double[] vector) {  
 double sum = 0;  
 for (double v : vector) {  
 sum += Math.*abs*(v);  
 }  
 return sum;  
 }  
  
 static double matrixDifferenceNorm(double[][] A, double[][] A\_star) {  
 double max = 0;  
 for (int j = 0; j < A[0].length; j++) { *// идём по столбцам*  
double sum = 0;  
 for (int i = 0; i < A.length; i++) { *// суммируем по строкам*  
sum += Math.*abs*(A[i][j] - A\_star[i][j]);  
 }  
 if (sum > max) max = sum;  
 }  
 return max;  
 }  
  
 static double vectorDifferenceNorm(double[] B, double[] B\_star) {  
 double sum = 0;  
 for (int i = 0; i < B.length; i++) {  
 sum += Math.*abs*(B[i] - B\_star[i]);  
 }  
 return sum;  
 }  
  
 static void evaluateConditionNumbers(double[][] A, double[] B, double[] X) {  
 double[][] invertedA = *invertMatrix*(A);  
 double columnNormA = *calculateColumnNorm*(A);  
 double columnNormInvertedA = *calculateColumnNorm*(invertedA);  
 double conditionNumber = columnNormA \* columnNormInvertedA;  
 System.*out*.printf("Абсолютное число обусловленности ν\_Δ : %.6f\n", columnNormInvertedA);  
 System.*out*.printf("Естественное число обусловленности ν\_δ : %.6f\n", columnNormInvertedA \* (*calculateVectorNorm*(B)/*calculateVectorNorm*(X)));  
 System.*out*.printf("Стандартное число обусловленности : %.6f\n", conditionNumber);  
 }  
  
 public static double evaluateConditionNumber(double deltaX, double deltaA, double deltaB, int type) {  
 if (type == 3){  
 return deltaX / (deltaA + deltaB);  
 }  
 else if (type == 2) {  
 return deltaX / (deltaA);  
 } else {  
 return deltaX / (deltaB);  
 }  
 }  
  
 public static void calculateAllErrorsAndCondition(  
 double[][] A, double[][] A\_star,  
 double[] B, double[] B\_star,  
 double[] X, double[] X\_star, int type) {  
  
 double[][] invertedA = *invertMatrix*(A\_star);  
 double columnNormA = *calculateColumnNorm*(A\_star);  
 double columnNormInvertedA = *calculateColumnNorm*(invertedA);  
 double conditionNumber = columnNormA \* columnNormInvertedA;  
 double normB = *calculateVectorNorm*(B);  
 double normX = *calculateVectorNorm*(X);  
  
 double matrixAbsoluteError = *matrixDifferenceNorm*(A, A\_star);  
 double vectorAbsoluteError = *vectorDifferenceNorm*(B, B\_star);  
 double solutionAbsoluteError = *vectorDifferenceNorm*(X, X\_star);  
  
 double matrixRelativeError = matrixAbsoluteError / columnNormA;  
 double vectorRelativeError = vectorAbsoluteError / normB;  
 double solutionRelativeError = solutionAbsoluteError / normX;  
  
 double conditionEstimate = *evaluateConditionNumber*(solutionRelativeError, matrixRelativeError, vectorRelativeError, type);  
  
 System.*out*.println("\nАнализ погрешностей:");  
 System.*out*.printf("Абсолютная погрешность матрицы ∆(A\*) : %.6f\n", matrixAbsoluteError);  
 System.*out*.printf("Относительная погрешность матрицы δ(A\*) : %.6f\n", matrixRelativeError);  
 System.*out*.printf("Абсолютная погрешность вектора В ∆(B\*) : %.6f\n", vectorAbsoluteError);  
 System.*out*.printf("Относительная погрешность вектора В δ(B\*) : %.6f\n", vectorRelativeError);  
 System.*out*.printf("Абсолютная погрешность вектора Х ∆(X\*) : %.6f\n", solutionAbsoluteError);  
 System.*out*.printf("Относительная погрешность вектора Х δ(X\*) : %.6f\n", solutionRelativeError);  
 System.*out*.printf("Оценка числа обусловленности : cond(A) > %.8f\n", conditionEstimate);  
 System.*out*.printf("Фактическое число обусловленности : %.8f\n", conditionNumber);  
 }  
  
}