**Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»**

**Факультет программной инженерии и компьютерной техники**

**Вычислительная математика**

**Лабораторная работа №1**

**Вариант 8**

**Студент: Коляда Анастасия, Тажибаева Ева**

**P3266**

**Преподаватель: Машина Екатерина Алексеевна**

**Оценка: \_\_\_\_\_\_\_\_\_\_**

**Подпись преподавателя: \_\_\_\_\_\_\_\_\_\_**

1. **Цели работы**

Написать программу, которая должна решить систему алгебраических уравнений методом Гаусса.

1. **Описание метода, расчётные формулы**

Метод Гаусса основан на приведении матрицы системы к треугольному виду так, чтобы ниже ее главной диагонали находились только нулевые элементы.

Прямой ход метода Гаусса состоит в последовательном исключении неизвестных из уравнений системы. Сначала с помощью первого уравнения исключается 𝑥1 из всех последующих уравнений системы. Затем с помощью второго уравнения исключается 𝑥2 из третьего и всех последующих уравнений и т.д. Этот процесс продолжается до тех пор, пока в левой части последнего (n-го) уравнения не останется лишь один член с неизвестным 𝑥𝑛 , т. е. матрица системы будет приведена к треугольному виду.

Обратный ход метода Гаусса состоит в последовательном вычислении искомых неизвестных: решая последнее уравнение, находим единственное в этом уравнении неизвестное 𝑥𝑛 . Далее, используя это значение, из предыдущего уравнения вычисляем 𝑥𝑛−1 и т. д. Последним найдем 𝑥1 из первого уравнения. Метод имеет много различных вычислительных схем, но в каждой из них основным требованием является det A ≠ 0 .

**Листинг программы**

Код написан на языке java в редакторе InteliJIIDEA. Код программы:

public class Matrix {  
 */\*\*  
 \* место, для хранения матрицы и манипуляций с ней  
 \*/* private double[][] matrix;  
 */\*\*  
 \* копия матрицы, здесь всегда оригинальная матрица, как до расчетов  
 \*/* private double[][] matrixBackup;  
 */\*\*  
 \* вектор значений B  
 \*/* private double[] b\_vector;  
 */\*\*  
 \* копия вектора значений B  
 \*/* private double[] b\_vectorBackup;  
 */\*\*  
 \* Размерность матрицы  
 \*/* private int n\_length;  
  
 */\*\*  
 \* Конструктор матрицы. Создает матрицу и вектор B  
 \*   
 \* @param n - размер матрицы  
 \*/* public Matrix(int n) {  
 matrix = new double[n][n];  
 b\_vector = new double[n];  
 n\_length = n;  
 }  
  
 */\*\*  
 \* Метод производит необходимые расчеты и выводит результат  
 \*/* public void calculateAll() {  
 try {  
 // Сохранеят копию матрицы, так как все манипуляции проихводятся с основными  
 // элементами  
 createBackup();  
 printMatrix();  
 System.*out*.printf("Определитель матрицы равен %.5f\n", calculateDeterminant());  
 // Расчет СЛАУ методом Гаусса  
 calcGauss();  
 } catch (MatrixException e) {  
 System.*out*.printf("Ошибка при расчете матрицы: %s\n", e.getMessage());  
 }  
 }  
  
 */\*\*  
 \* Производит расчет методом Гаусса  
 \*   
 \* @throws MatrixException - если матрицу невозможно посчитать.  
 \*/* private void calcGauss() throws MatrixException {  
 System.*out*.println("Расчет СЛАУ методом Гаусса.");  
 runForward();  
 System.*out*.println("Треугольная матрица:");  
 printMatrix();  
 System.*out*.println("Вектор результатов x:");  
 double[] res = runBackward();  
 *printVector*(res);  
 System.*out*.println("Вектор невязок r:");  
 res = calcInconsistencesOnBackup(res);  
 *printVector*(res);  
 }  
  
 */\*\*  
 \* Расчет невязок  
 \*   
 \* @param xs - вектор результатов x  
 \* @return - вектор невязок  
 \*/* private double[] calcInconsistencesOnBackup(double[] xs) {  
 double[] inc = new double[n\_length];  
 for (int row = 0; row < n\_length; row++) {  
 double sum = 0;  
 for (int col = 0; col < n\_length; col++) {  
 sum += matrixBackup[row][col] \* xs[col];  
 }  
 inc[row] = sum - b\_vectorBackup[row];  
 }  
 return inc;  
 }  
  
 */\*\*  
 \* Сохраняет копию матрицы  
 \*/* private void createBackup() {  
 matrixBackup = new double[n\_length][n\_length];  
 b\_vectorBackup = new double[n\_length];  
 for (int row = 0; row < n\_length; row++) {  
 System.*arraycopy*(matrix[row],0,matrixBackup[row],0,n\_length);  
 b\_vectorBackup[row] = b\_vector[row];  
 }  
 }  
  
 */\*\*  
 \* Расчет определителя методом Саррюса  
 \*   
 \* @return - рассчитанный определитель  
 \*/* private double calculateDeterminant() {  
  
 double plus = 0;  
 double minus = 0;  
 for (int shift = 0; shift < n\_length; shift++) {  
 double dp = 1;  
 // Расчет по основной диагонали  
 for (int row = 0; row < n\_length; row++) {  
 int col = row + shift;  
 if (col >= n\_length)  
 col -= n\_length;  
 dp \*= matrix[row][col];  
 }  
 plus += dp;  
 // Расчет по побочной диагонали  
 dp = 1;  
 int pos = 0;  
 for (int row = n\_length - 1; row >= 0; row--) {  
 int col = pos + shift;  
 if (col >= n\_length)  
 col -= n\_length;  
 dp \*= matrix[row][col];  
 pos++;  
 }  
 minus += dp;  
 }  
 return plus - minus;  
  
 }  
  
 */\*\*  
 \* Прямой проход по методу Гаусса  
 \*  
 \* @throws MatrixException  
 \*/* private void runForward() throws MatrixException {  
 int lMargin = 0;  
 // Для каждой строки матрицы  
 for (int l = 0; l < n\_length; l++) {  
 // Если в текущей позиции строки стоит 0, то поищем замену  
 if (matrix[l][lMargin] == 0) {  
 swapLine(l, lMargin);  
 }  
 // Производим исключение позиции lMargin из следующих строк  
 // Для каждой следующей строки  
 for (int l2 = l + 1; l2 < n\_length; l2++) {  
 // Определяем коэффициент для исключения позиции lMargin  
 double factor = 0 - matrix[l2][lMargin] / matrix[l][lMargin];  
 // Получаем результат умножения текущей верхней строки на коэффициент  
 double[] toAdd = multVc(l, factor);  
 // Добавляем его к текущей строке  
 addToLine(l2, toAdd);  
 b\_vector[l2] += b\_vector[l] \* factor;  
 }  
 lMargin++;  
 }  
 }  
  
 */\*\*  
 \* Обратный проход по методу Гаусса  
 \*   
 \* @return - вектор решений x  
 \*/* private double[] runBackward() {  
 double[] result = new double[n\_length];  
 for (int row = n\_length - 1; row >= 0; row--) {  
 result[row] = (b\_vector[row] - sumOfRightPart(row, result)) / matrix[row][row];  
 }  
 return result;  
 }  
  
 */\*\*  
 \* Вспомогательный метод - рассчитывает при обратном ходе сумму элементов с  
 \* известными значениями  
 \*   
 \* @param row - строка, для которой производится расчет  
 \* @param result - вектор решений СЛАУ  
 \* @return - искомая сумма  
 \*/* private double sumOfRightPart(int row, double[] result) {  
 double sum = 0;  
 for (int c = row + 1; c < n\_length; c++) {  
 sum += result[c] \* matrix[row][c];  
 }  
 return sum;  
 }  
  
 */\*\*  
 \* Вспомогательный метод добавляет к строке матрицы значения из расчетного  
 \* массива  
 \*  
 \* @param line  
 \* @param toAdd  
 \*/* private void addToLine(int line, double[] toAdd) {  
 for(int i = 0;i<n\_length;i++){  
 matrix[line][i] += toAdd[i];  
 }  
 }  
  
 */\*\*  
 \* Метод умножает строку матрицы на указанный коэффициент  
 \*   
 \* @param line - индекс строки матрицы  
 \* @param factor - коэффициент  
 \* @return - вектор результат умножения каждого элемента строки на коэффициент  
 \*/* private double[] multVc(int line, double factor) {  
 double[] result = new double[n\_length];  
 for (int i = 0; i < n\_length; i++) {  
 result[i] = matrix[line][i] == 0 ? 0 : matrix[line][i] \* factor;  
 }  
 return result;  
 }  
  
 */\*\*  
 \* Ищет строку с ненулевой величиной в позиции col и меняет эту строку местами  
 \* со строкой line. Если такой строки нет, то выбрасывается ошибка  
 \*   
 \* @param line - номе строки  
 \* @param col - номер колонки  
 \* @throws MatrixException  
 \*/* private void swapLine(int line, int col) throws MatrixException {  
 for (int l = line + 1; l < n\_length; l++) {  
 if (matrix[l][col] != 0) {  
 double[] temp = matrix[line];  
 matrix[line] = matrix[l];  
 matrix[l] = temp;  
  
 double tb = b\_vector[line];  
 b\_vector[line] = b\_vector[l];  
 b\_vector[l] = tb;  
 return;  
 }  
 }  
 throw new MatrixException("Нет строки с ненулевым элементом для перемены строк");  
 }  
  
 */\*\*  
 \* сохраняет текстовую строку в матрице  
 \*   
 \* @param lineNumber - номер строки  
 \* @param s - строка из массива чисел в текстовом представлении  
 \* @return - true, если строка сохранена  
 \* @throws NumberFormatException - в результате ошиюбки при преобразования чисел  
 \* из текста в double  
 \*/* public boolean saveString(int lineNumber, String[] s) throws NumberFormatException {  
 if (lineNumber < 0 || lineNumber >= n\_length)  
 return false;  
 int i;  
 for (i = 0; i < n\_length; i++) {  
 matrix[lineNumber][i] = Double.*parseDouble*(s[i]);  
 }  
 b\_vector[lineNumber] = Double.*parseDouble*(s[i]);  
 return true;  
 }  
  
 */\*\*  
 \* Выводит текущую матрицу на консоль  
 \*/* private void printMatrix() {  
 *printMatrix*(matrix, b\_vector);  
 }  
  
 */\*\*  
 \* Выводит на консоль указанный вектор  
 \*   
 \* @param line - вектор чисел для вывода  
 \*/* public static void printVector(double[] line) {  
 System.*out*.print("(");  
 for (double cell : line) {  
 System.*out*.printf("%8.8s", String.*format*("%.3f", cell));  
 }  
 System.*out*.print(")\n");  
 System.*out*.println("Конец результата");  
 }  
  
 */\*\*  
 \* Выводит на консоль указанную СЛАУ  
 \*   
 \* @param m - матрица  
 \* @param b - вектор B  
 \*/* private static void printMatrix(double[][] m, double[] b) {  
 int bc = 0;  
 for (double[] line : m) {  
 for (double cell : line) {  
 System.*out*.printf("%8.8s", String.*format*("%.3f", cell));  
 }  
 System.*out*.printf(" | %f\n", b[bc++]);  
 }  
 System.*out*.println("Конец матрицы.");  
 }  
}

public class Gauss {  
 public static void main(String[] args) {  
  
 //Создаем ридер для чтения с консоли, такой метод чтения работает и из консоли IDE  
 BufferedReader bufferReader = new BufferedReader(new InputStreamReader(System.*in*));  
 System.*out*.println("Решение СЛАУ методом ГАУССА.");  
 System.*out*.println("Студент Анастасия Коляда");  
  
 int inputType = -1;  
  
 // Цикл считывания действия пользователя по запросу  
 do {  
 System.*out*.println(  
 """  
 Для продолжения введите:  
 0 - для прекращения работы;  
 1 - если Вы хотите продолжить и ввести размерность матрицы и матрицу с консоли;  
 2 - если Вы хотите указать текстовый файл, который модержиит размерность матрицы, матрицу и вектор B.  
 """);  
  
 try {  
 inputType = Integer.*parseInt*(bufferReader.readLine());  
 } catch (NumberFormatException | IOException e) {  
 System.*out*.printf("Ошибка чтения %s\n", e.getMessage());  
 }  
 // До тех пор, пока пользоветль не введет ожидаемые значения  
 } while (inputType < 0 || inputType > 2);  
  
 //прекращение работы по запросу  
 if (inputType == 0)  
 return;  
  
 //Создаем подходящий для запроса ридер  
 IGetMatrix reader = inputType == 1 ? new ReadFromConsole() : new ReadFromFile();  
 try {  
 // Получаем у ридера матрицу  
 Matrix matrix = reader.GetMatrix();  
 if (matrix == null)  
 return;  
 //Запускаем требуемые расчеты  
 matrix.calculateAll();  
 } catch (IOException e) {  
 System.*out*.printf("Ошибка чтения матрицы %s\n", e.getMessage());  
 }  
 }  
}

1. **Примеры и результаты работы программы**
2. Ввод вручную

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание**

1. **Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

   Автоматически созданное описание**Ввод из файла

1. **Вывод**

В ходе работы мы повторно изучили работу метода Гаусса при решении системы уравнений**.** Написали программу, которая способная решать СЛАУ размерностью до 20. Данные могут подаваться с клавиатуры и из .txt файла.