**Национальный исследовательский университет ИТМО**

**Факультет ПиИКТ**

****

**Вычислительная математика**

**Лабораторная работа № 1**

Методы решения СЛАУ

Вариант № **6 (Метод простых итераций)**

Работу выполнил: Манжиков Никита Сергеевич

Преподаватель: Рыбаков Степан Дмитриевич

**Санкт-Петербург**

**2023 г.**

**Цель работы**

Используя известные методы вычислительной математики, написать программный код, осуществляющий решение СЛАУ. Проанализировать полученные результаты, оценить погрешность.

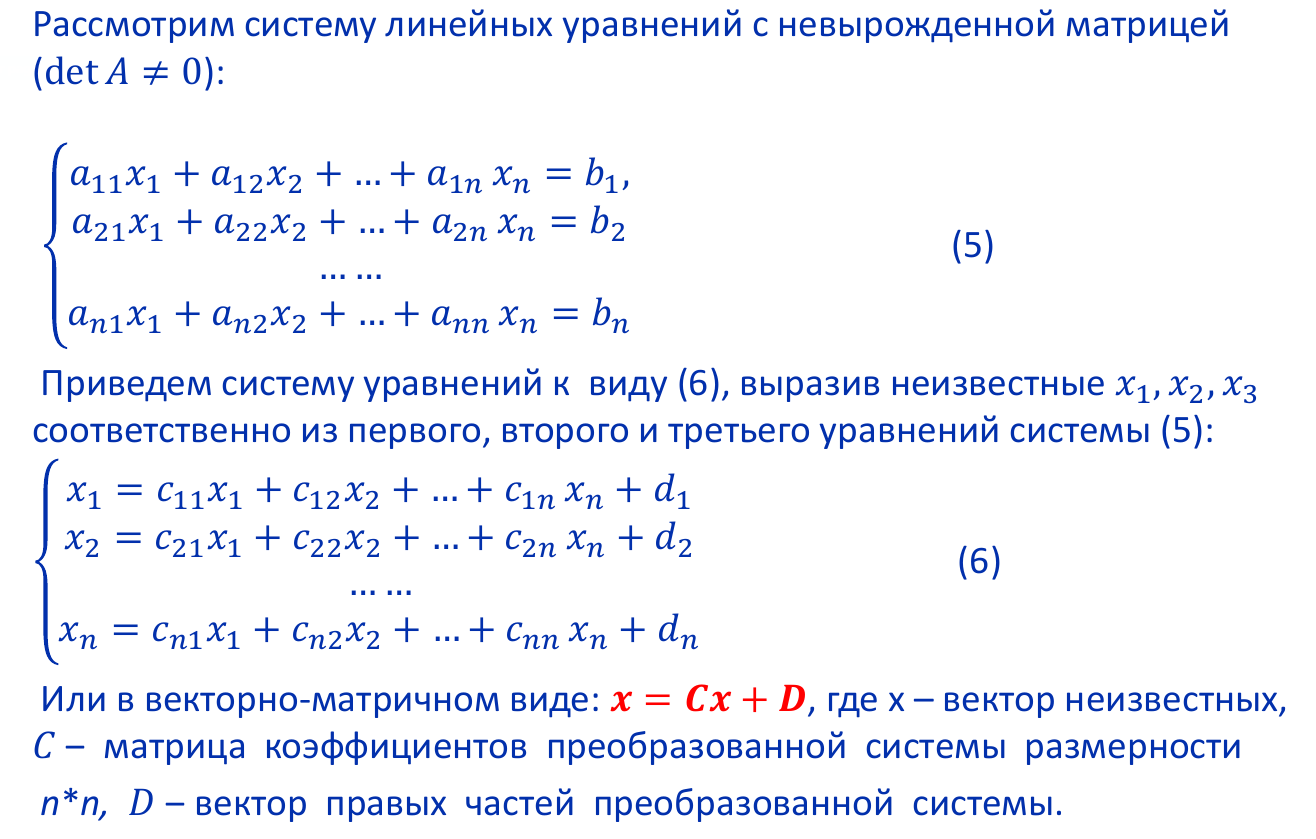
**Задание лабораторной работы**

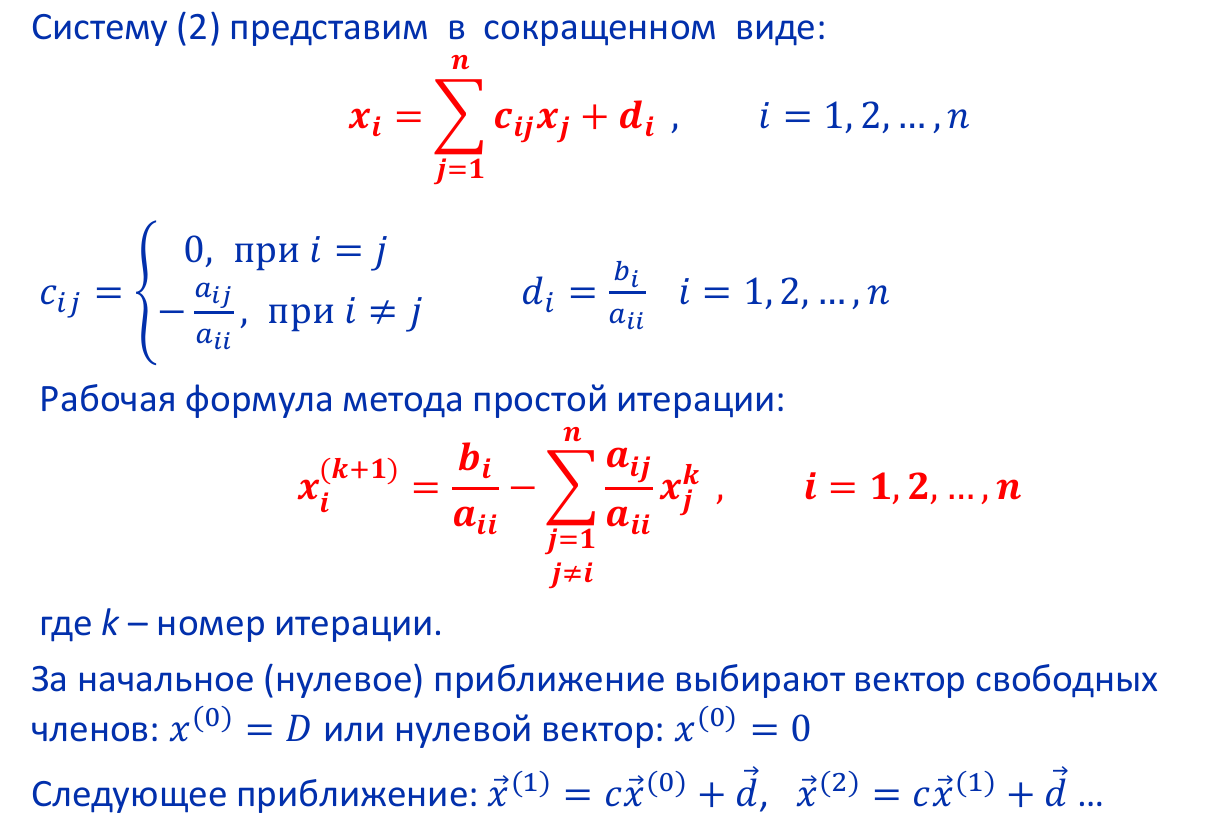
1. № варианта определяется как номер в списке группы согласно ИСУ.
2. В программе численный метод должен быть реализован в виде отдельной подпрограммы или класса, в который входные/выходные данные передаются в качестве параметров.
3. Размерность матрицы n<=20 (задается из файла или с клавиатуры - по выбору конечного пользователя).
4. Должна быть реализована возможность ввода коэффициентов матрицы, как с клавиатуры, так и из файла (по выбору конечного пользователя).

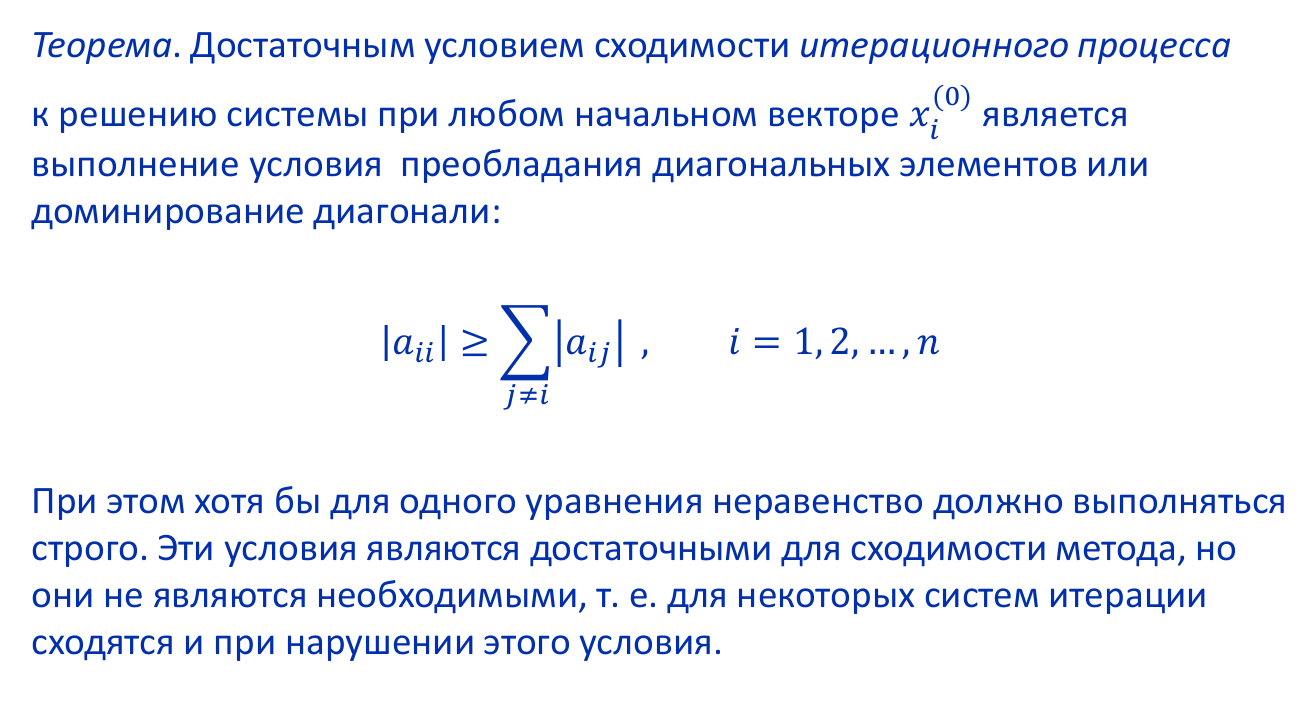
**Для итерационных методов должно быть реализовано:**

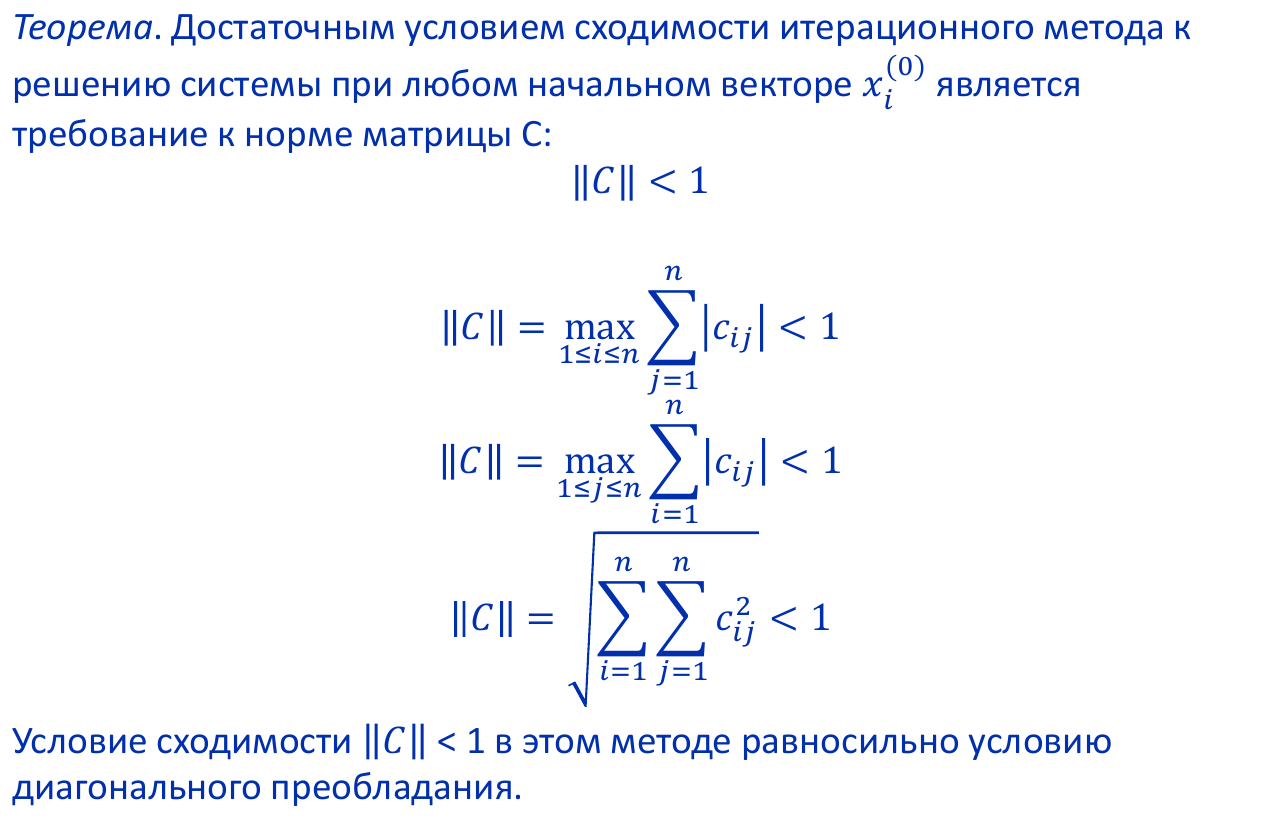
* Точность задается с клавиатуры/файла
* Проверка диагонального преобладания (в случае, если диагональное преобладание в исходной матрице отсутствует, сделать перестановку строк/столбцов до тех пор, пока преобладание не будет достигнуто). В случае невозможности достижения диагонального преобладания - выводить соответствующее сообщение.
* Вывод вектора неизвестных:
* Вывод количества итераций, за которое было найдено решение.
* Вывод вектора погрешностей:

**Описание метода, расчетные формулы**

****

****

****

****

**Листинг программы**

Процедура, приводящая матрицу к виду (6)

private void modifyMatrix() {  
 for (int i = 0; i < matrix.length; i++) {  
 int currentIndex = i;  
 matrix[i] = Arrays.*stream*(matrix[i])  
 .map(x -> -x / matrix[currentIndex][currentIndex])  
 .toArray();  
 matrix[i][i] = 0;  
 matrix[i][n] \*= -1;  
 }  
}

Метод итераций

private int iterate() {  
 int currentIter = 0;  
 double maxEps = Double.*MAX\_VALUE*;  
 error = new double[solution.length];  
 while (currentIter < *MAX\_ITERATION* && maxEps >= eps) {  
 maxEps = performIteration();  
 currentIter++;  
 }  
 return currentIter;  
}

private double performIteration() {  
 double currentMaxEps = 0;  
 double[] currentSolution = Arrays.*copyOf*(solution, solution.length);  
 for (int i = 0; i < currentSolution.length; i++) {  
 double newValue = 0;  
 for (int j = 0; j < currentSolution.length; j++) {  
 if (i == j) continue;  
 newValue += matrix[i][j] \* currentSolution[j];  
 }  
 newValue += matrix[i][matrix.length];  
 error[i] = Math.*abs*(newValue - solution[i]);  
 if (error[i] > currentMaxEps) currentMaxEps = error[i];  
 solution[i] = newValue;  
 }  
 return currentMaxEps;  
}

Вся программа:  
package ru.rosroble;  
  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.stream.IntStream;  
  
public class SolverRunner {  
 public static void main(String[] args) throws IOException {  
 Solver solver = new Solver();  
 solver.run();  
 }  
}  
  
class Solver {  
 public static final int *MAX\_ITERATION* = 1000000;  
 private double eps;  
 private BufferedReader reader;  
 private int n;  
 private double[][] matrix;  
 private double[] solution;  
 private double[] error;  
  
 public void run() throws IOException {  
 init();  
 readAndPrintMatrix();  
 attemptDiagonalDominance();  
  
 if (!diagonalDominanceAchieved()) {  
 System.*out*.println("Нельзя достигнуть диагонального преобладания. Завершение работы.");  
 return;  
 }  
  
 modifyMatrix();  
 solution = initSolution(matrix);  
 int iterations = iterate();  
  
 if (iterations == *MAX\_ITERATION*) {  
 System.*out*.println("Не удалось достичь требуемой сходимости за допустимое число итераций.");  
 return;  
 }  
  
 printResults(iterations);  
 }  
  
 private void init() throws IOException {  
 reader = new BufferedReader(new InputStreamReader(System.*in*));  
 getInputFileOrKeyboard();  
 readEpsilon();  
 readNumberOfUnknowns();  
 }  
  
 private void getInputFileOrKeyboard() throws IOException {  
 System.*out*.println("Введите имя файла или введите 0 чтобы считать с клавиатуры: ");  
 String input = reader.readLine();  
  
 if (!input.equals("0")) {  
 reader = getFileReader(input);  
 }  
 }  
  
 private BufferedReader getFileReader(String input) throws IOException {  
 BufferedReader fileReader = null;  
 while (fileReader == null) {  
 try {  
 fileReader = new BufferedReader(new FileReader(input));  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Файл с указанным именем не найден. Повторите ввод.");  
 }  
 }  
 return fileReader;  
 }  
  
 private void readEpsilon() throws IOException {  
 System.*out*.println("Введите погрешность: ");  
 eps = Double.*parseDouble*(reader.readLine());  
 }  
  
 private void readNumberOfUnknowns() throws IOException {  
 do {  
 System.*out*.println("Введите количество неизвестных системы (<= 20): ");  
 n = Integer.*parseInt*(reader.readLine());  
 } while (n <= 0 || n > 20);  
 }  
  
 private void readAndPrintMatrix() throws IOException {  
 System.*out*.println("Введите матрицу: ");  
 matrix = readMatrix(n);  
 System.*out*.println("Считана матрица: ");  
 printMatrix();  
 }  
  
 private double[][] readMatrix(int n) throws IOException {  
 double[][] matrix = new double[n][n + 1];  
 for (int i = 0; i < matrix.length; i++) {  
 String[] line = reader.readLine().split(" ");  
 matrix[i] = Arrays.*stream*(line).mapToDouble(Double::*parseDouble*).toArray();  
 }  
  
 return matrix;  
 }  
  
 private void printMatrix() {  
 for (int i = 0; i < matrix.length; i++) {  
 for (int j = 0; j < matrix[0].length; j++) {  
 System.*out*.print(matrix[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
 }  
  
 private void attemptDiagonalDominance() {  
 System.*out*.println("Попытка достичь диагонального преобладания.");  
 Kuhn kuhn = new Kuhn(matrix);  
 matrix = kuhn.getMatrixWithDiagonalDominance();  
 }  
  
 private boolean diagonalDominanceAchieved() {  
 for (double[] row : matrix) {  
 if (row == null) {  
 return false;  
 }  
 }  
 System.*out*.println("Диагональное преобладание достигнуто. Модифицированная матрица: ");  
 printMatrix();  
 return true;  
 }  
  
 private void modifyMatrix() {  
 for (int i = 0; i < matrix.length; i++) {  
 int currentIndex = i;  
 matrix[i] = Arrays.*stream*(matrix[i])  
 .map(x -> -x / matrix[currentIndex][currentIndex])  
 .toArray();  
 matrix[i][i] = 0;  
 matrix[i][n] \*= -1;  
 }  
 }  
  
 private double[] initSolution(double[][] matrix) {  
 double[] solution = new double[matrix.length];  
 for (int i = 0; i < matrix.length; i++) {  
 solution[i] = matrix[i][matrix.length];  
 }  
 return solution;  
 }  
  
 private int iterate() {  
 int currentIter = 0;  
 double maxEps = Double.*MAX\_VALUE*;  
 error = new double[solution.length];  
 while (currentIter < *MAX\_ITERATION* && maxEps >= eps) {  
 maxEps = performIteration();  
 currentIter++;  
 }  
 return currentIter;  
 }  
  
 private double performIteration() {  
 double currentMaxEps = 0;  
 double[] currentSolution = Arrays.*copyOf*(solution, solution.length);  
 for (int i = 0; i < currentSolution.length; i++) {  
 double newValue = 0;  
 for (int j = 0; j < currentSolution.length; j++) {  
 if (i == j) continue;  
 newValue += matrix[i][j] \* currentSolution[j];  
 }  
 newValue += matrix[i][matrix.length];  
 error[i] = Math.*abs*(newValue - solution[i]);  
 if (error[i] > currentMaxEps) currentMaxEps = error[i];  
 solution[i] = newValue;  
 }  
 return currentMaxEps;  
 }  
  
 private void printResults(int iterations) {  
 System.*out*.println("Вектор решения: " + Arrays.*toString*(solution));  
 System.*out*.println("Вектор погрешностей: " + Arrays.*toString*(error));  
 System.*out*.println("Количество итераций: " + iterations);  
 }  
  
 // внутренний класс реализующий алгоритм Куна по нахождению максимального паросочетания в двудольном графе  
 // используется для достижения диагонального преобладания в матрице  
 private static class Kuhn {  
 private double[][] matrix;  
 private List<List<Integer>> graph;  
 private int[] mt;  
 private boolean[] used;  
  
 public Kuhn(double[][] matrix) {  
 this.matrix = matrix;  
 graph = new ArrayList<>();  
 mt = new int[matrix.length];  
 used = new boolean[matrix.length];  
 Arrays.*fill*(mt, -1);  
 }  
  
 public double[][] getMatrixWithDiagonalDominance() {  
 buildGraph();  
 findMaximumMatching();  
 return buildMatrixWithDiagonalDominance();  
 }  
  
 private void buildGraph() {  
 for (int i = 0; i < matrix.length; i++) {  
 List<Integer> possibleIndexes = new ArrayList<>();  
 double sum = Arrays.*stream*(matrix[i]).map(Math::*abs*).sum() - Math.*abs*(matrix[i][matrix.length]);  
 for (int j = 0; j < matrix.length; j++) {  
 if (sum - 2 \* Math.*abs*(matrix[i][j]) <= 0) possibleIndexes.add(j);  
 }  
 graph.add(possibleIndexes);  
 }  
 }  
  
 private void findMaximumMatching() {  
 for (int v = 0; v < matrix.length; v++) {  
 Arrays.*fill*(used, false);  
 tryKuhn(v);  
 }  
 }  
  
 private boolean tryKuhn(int v) {  
 if (used[v]) return false;  
 used[v] = true;  
 for (int i = 0; i < graph.get(v).size(); ++i) {  
 int to = graph.get(v).get(i);  
 if (mt[to] == -1 || tryKuhn(mt[to])) {  
 mt[to] = v;  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private double[][] buildMatrixWithDiagonalDominance() {  
 double[][] matrixCopy = Arrays.*copyOf*(matrix, matrix.length);  
 for (int i = 0; i < matrix.length; i++) {  
 if (mt[i] == -1) {  
 matrix[i] = null;  
 } else {  
 matrix[i] = matrixCopy[mt[i]];  
 }  
 }  
 return matrix;  
 }  
 }  
}

package ru.rosroble;  
  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.stream.IntStream;  
  
public class SolverRunner {  
 public static void main(String[] args) throws IOException {  
 Solver solver = new Solver();  
 solver.run();  
 }  
}  
  
class Solver {  
 public static final int *MAX\_ITERATION* = 1000000;  
 private double eps;  
 private BufferedReader reader;  
 private int n;  
 private double[][] matrix;  
 private double[] solution;  
 private double[] error;  
  
 public void run() throws IOException {  
 init();  
 readAndPrintMatrix();  
 attemptDiagonalDominance();  
  
 if (!diagonalDominanceAchieved()) {  
 System.*out*.println("Нельзя достигнуть диагонального преобладания. Завершение работы.");  
 return;  
 }  
  
 modifyMatrix();  
 solution = initSolution(matrix);  
 int iterations = iterate();  
  
 if (iterations == *MAX\_ITERATION*) {  
 System.*out*.println("Не удалось достичь требуемой сходимости за допустимое число итераций.");  
 return;  
 }  
  
 printResults(iterations);  
 }  
  
 private void init() throws IOException {  
 reader = new BufferedReader(new InputStreamReader(System.*in*));  
 getInputFileOrKeyboard();  
 readEpsilon();  
 readNumberOfUnknowns();  
 }  
  
 private void getInputFileOrKeyboard() throws IOException {  
 System.*out*.println("Введите имя файла или введите 0 чтобы считать с клавиатуры: ");  
 String input = reader.readLine();  
  
 if (!input.equals("0")) {  
 reader = getFileReader(input);  
 }  
 }  
  
 private BufferedReader getFileReader(String input) throws IOException {  
 BufferedReader fileReader = null;  
 while (fileReader == null) {  
 try {  
 fileReader = new BufferedReader(new FileReader(input));  
 } catch (FileNotFoundException e) {  
 System.*out*.println("Файл с указанным именем не найден. Повторите ввод.");  
 }  
 }  
 return fileReader;  
 }  
  
 private void readEpsilon() throws IOException {  
 System.*out*.println("Введите погрешность: ");  
 eps = Double.*parseDouble*(reader.readLine());  
 }  
  
 private void readNumberOfUnknowns() throws IOException {  
 do {  
 System.*out*.println("Введите количество неизвестных системы (<= 20): ");  
 n = Integer.*parseInt*(reader.readLine());  
 } while (n <= 0 || n > 20);  
 }  
  
 private void readAndPrintMatrix() throws IOException {  
 System.*out*.println("Введите матрицу: ");  
 matrix = readMatrix(n);  
 System.*out*.println("Считана матрица: ");  
 printMatrix();  
 }  
  
 private double[][] readMatrix(int n) throws IOException {  
 double[][] matrix = new double[n][n + 1];  
 for (int i = 0; i < matrix.length; i++) {  
 String[] line = reader.readLine().split(" ");  
 matrix[i] = Arrays.*stream*(line).mapToDouble(Double::*parseDouble*).toArray();  
 }  
  
 return matrix;  
 }  
  
 private void printMatrix() {  
 for (int i = 0; i < matrix.length; i++) {  
 for (int j = 0; j < matrix[0].length; j++) {  
 System.*out*.print(matrix[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
 }  
  
 private void attemptDiagonalDominance() {  
 System.*out*.println("Попытка достичь диагонального преобладания.");  
 Kuhn kuhn = new Kuhn(matrix);  
 matrix = kuhn.getMatrixWithDiagonalDominance();  
 }  
  
 private boolean diagonalDominanceAchieved() {  
 for (double[] row : matrix) {  
 if (row == null) {  
 return false;  
 }  
 }  
 System.*out*.println("Диагональное преобладание достигнуто. Модифицированная матрица: ");  
 printMatrix();  
 return true;  
 }  
  
 private void modifyMatrix() {  
 for (int i = 0; i < matrix.length; i++) {  
 int currentIndex = i;  
 matrix[i] = Arrays.*stream*(matrix[i])  
 .map(x -> -x / matrix[currentIndex][currentIndex])  
 .toArray();  
 matrix[i][i] = 0;  
 matrix[i][n] \*= -1;  
 }  
 }  
  
 private double[] initSolution(double[][] matrix) {  
 double[] solution = new double[matrix.length];  
 for (int i = 0; i < matrix.length; i++) {  
 solution[i] = matrix[i][matrix.length];  
 }  
 return solution;  
 }  
  
 private int iterate() {  
 int currentIter = 0;  
 double maxEps = Double.*MAX\_VALUE*;  
 error = new double[solution.length];  
 while (currentIter < *MAX\_ITERATION* && maxEps >= eps) {  
 maxEps = performIteration();  
 currentIter++;  
 }  
 return currentIter;  
 }  
  
 private double performIteration() {  
 double currentMaxEps = 0;  
 double[] currentSolution = Arrays.*copyOf*(solution, solution.length);  
 for (int i = 0; i < currentSolution.length; i++) {  
 double newValue = 0;  
 for (int j = 0; j < currentSolution.length; j++) {  
 if (i == j) continue;  
 newValue += matrix[i][j] \* currentSolution[j];  
 }  
 newValue += matrix[i][matrix.length];  
 error[i] = Math.*abs*(newValue - solution[i]);  
 if (error[i] > currentMaxEps) currentMaxEps = error[i];  
 solution[i] = newValue;  
 }  
 return currentMaxEps;  
 }  
  
 private void printResults(int iterations) {  
 System.*out*.println("Вектор решения: " + Arrays.*toString*(solution));  
 System.*out*.println("Вектор погрешностей: " + Arrays.*toString*(error));  
 System.*out*.println("Количество итераций: " + iterations);  
 }  
  
 // внутренний класс реализующий алгоритм Куна по нахождению максимального паросочетания в двудольном графе  
 // используется для достижения диагонального преобладания в матрице  
 private static class Kuhn {  
 private double[][] matrix;  
 private List<List<Integer>> graph;  
 private int[] mt;  
 private boolean[] used;  
  
 public Kuhn(double[][] matrix) {  
 this.matrix = matrix;  
 graph = new ArrayList<>();  
 mt = new int[matrix.length];  
 used = new boolean[matrix.length];  
 Arrays.*fill*(mt, -1);  
 }  
  
 public double[][] getMatrixWithDiagonalDominance() {  
 buildGraph();  
 findMaximumMatching();  
 return buildMatrixWithDiagonalDominance();  
 }  
  
 private void buildGraph() {  
 for (int i = 0; i < matrix.length; i++) {  
 List<Integer> possibleIndexes = new ArrayList<>();  
 double sum = Arrays.*stream*(matrix[i]).map(Math::*abs*).sum() - Math.*abs*(matrix[i][matrix.length]);  
 for (int j = 0; j < matrix.length; j++) {  
 if (sum - 2 \* Math.*abs*(matrix[i][j]) <= 0) possibleIndexes.add(j);  
 }  
 graph.add(possibleIndexes);  
 }  
 }  
  
 private void findMaximumMatching() {  
 for (int v = 0; v < matrix.length; v++) {  
 Arrays.*fill*(used, false);  
 tryKuhn(v);  
 }  
 }  
  
 private boolean tryKuhn(int v) {  
 if (used[v]) return false;  
 used[v] = true;  
 for (int i = 0; i < graph.get(v).size(); ++i) {  
 int to = graph.get(v).get(i);  
 if (mt[to] == -1 || tryKuhn(mt[to])) {  
 mt[to] = v;  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private double[][] buildMatrixWithDiagonalDominance() {  
 double[][] matrixCopy = Arrays.*copyOf*(matrix, matrix.length);  
 for (int i = 0; i < matrix.length; i++) {  
 if (mt[i] == -1) {  
 matrix[i] = null;  
 } else {  
 matrix[i] = matrixCopy[mt[i]];  
 }  
 }  
 return matrix;  
 }  
 }  
}

**Примеры и результаты работы программы**

Проверим работу программы на СЛАУ:

Изображение выглядит как календарь

Автоматически созданное описание

На вход программе подаем файл test.txt:

|  |
| --- |
| 0.0001  3  10 2 -1 5  -2 -6 -1 24.52  1 -3 12 36 |

В первой строчке вводится погрешность, во второй количество неизвестных n.

Затем ожидается n строчек - СЛАУ в матричном виде.

Заметим, что диагональное преобладание не выполнено.

Результат работы программы:

Изображение выглядит как текст

Автоматически созданное описание

Результат равен ожидаемому с точностью до указанной погрешности.

**Выводы по работе**

В ходе работы реализован метод простых итераций, позволяющий решать СЛАУ с высокой точностью. Метод несложен в программном изложении и довольно быстро справляется с небольшими по размеру матрицами.