Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Лабораторная работа № 1

### по дисциплине “ Вычислительная математика”

Вариант 2 (метод простых итераций)

Выполнил:

Студент группы P3214

Арсеньев Денис Егорович

Преподаватель: Малышева Татьяна Алексеевна

г. Санкт-Петербург

2024

**Цель работы:**

Воспользоваться численным методом простых итераций для решения СЛАУ с помощью программного кода. Вычислить вектор приближенного решения и погрешности.

**Описание метода и расчётные формулы:**

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

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Листинг программы (класс с методами для итераций):**

package arden.simple\_iterations.algorithm;  
  
import arden.simple\_iterations.model.Matrix;  
import arden.simple\_iterations.utils.PrettyPrinter;  
  
import java.util.OptionalInt;  
import java.util.stream.IntStream;  
  
public class SimpleIterator {  
 private final Matrix matrix;  
 private final double[][] elements;  
 private final int size;  
  
 public SimpleIterator(Matrix matrix) {  
 this.matrix = matrix;  
 elements = matrix.matrix();  
 size = matrix.size();  
 }  
  
 public double[] iterate() {  
 if (checkAndMakeDiagonalDomination()) {  
 System.*out*.println("\nМатрица с диагональным преобладанием:");  
 PrettyPrinter.*printMatrix*(matrix);  
 PrettyPrinter.*printTableHeader*(size);  
  
 getDiagonalElements();  
  
 double[] approximation = new double[size];  
 for (int i = 0; i < size; i++) {  
 approximation[i] = elements[i][size];  
 }  
  
 double[] newApproximation = new double[size];  
 double inaccuracy = 0;  
  
 int cnt = 0;  
  
 PrettyPrinter.*printRow*(cnt++, approximation, inaccuracy);  
 while (true) {  
 for (int i = 0; i < size; i++) {  
 double sum = 0;  
 for (int j = 0; j < size; j++) {  
 sum += elements[i][j] \* approximation[j];  
 }  
  
 sum += elements[i][size];  
 newApproximation[i] = sum;  
 }  
  
 inaccuracy = countInaccuracy(approximation, newApproximation);  
  
 PrettyPrinter.*printRow*(cnt++, newApproximation, inaccuracy);  
  
 if (inaccuracy > matrix.accuracy()) {  
 System.*arraycopy*(newApproximation, 0, approximation, 0, size);  
 } else break;  
 }  
  
 PrettyPrinter.*printResult*(cnt - 1, approximation, newApproximation);  
 return newApproximation;  
 } else {  
 return null;  
 }  
 }  
  
 private double countInaccuracy(double[] approximation, double[] newApproximation) {  
 double inaccuracy = 0, diff = 0;  
 for (int i = 0; i < size; i++) {  
 inaccuracy = Math.*max*(diff, Math.*abs*(approximation[i] - newApproximation[i]));  
 diff = Math.*abs*(approximation[i] - newApproximation[i]);  
 }  
 return inaccuracy;  
 }  
  
 private boolean checkAndMakeDiagonalDomination() {  
 int row = 0, prev = -1;  
 int strict = 0;  
 while (row < size) {  
 int tmp = row;  
 OptionalInt maxIndex = IntStream.*range*(0, size)  
 .reduce((k, l) -> Math.*abs*(elements[tmp][k]) > Math.*abs*(elements[tmp][l]) ? k : l);  
 if (maxIndex.isPresent()) {  
 int index = maxIndex.getAsInt();  
 double sum = sumWithoutDiagonal(elements, row, index);  
 if (Math.*abs*(elements[row][index]) > sum) {  
 strict++;  
 }  
 if (index == prev || Math.*abs*(elements[row][index]) < sum) {  
 return false;  
 } else if (index != row) {  
 permutation(elements, row, index);  
 prev = index;  
 } else {  
 prev = index;  
 row++;  
 }  
 } else {  
 return false;  
 }  
 }  
 return strict != 0;  
 }  
  
 private double sumWithoutDiagonal(double[][] elements, int row, int index) {  
 double sum = 0;  
 for (int i = 0; i < size; i++) {  
 if (i != index) {  
 sum += Math.*abs*(elements[row][i]);  
 }  
 }  
 return sum;  
 }  
 private void permutation(double[][] elements, int row, int index) {  
 for (int i = 0; i < size + 1; i++) {  
 double tmp = elements[row][i];  
 elements[row][i] = elements[index][i];  
 elements[index][i] = tmp;  
 }  
 }  
  
 private void getDiagonalElements() {  
 for (int i = 0; i < size; i++) {  
 double diagCoefficient = elements[i][i];  
 elements[i][i] = 0;  
 for (int j = 0; j < size; j++) {  
 if (i != j) {  
 elements[i][j] /= -diagCoefficient;  
 }  
 }  
 elements[i][size] /= diagCoefficient;  
 }  
 }  
}

Полный код на GitHub: <https://github.com/Arden30/CompMathLab1>

**Примеры и результаты работы программы:**

***Первый пример (возможность свести к диагональному преобладанию):***

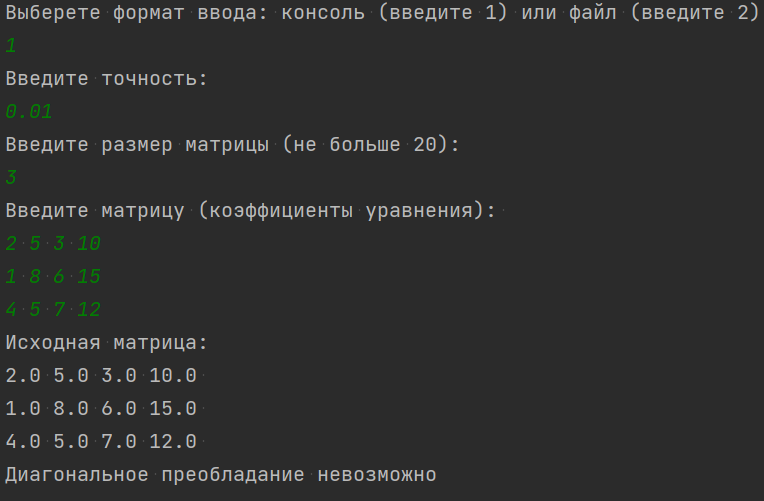
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

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

Автоматически созданное описание***

***Второй пример (нельзя свести к диагональному преобладанию):***



***Третий пример (чтение из файла):***

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

**Вывод:**

В ходе выполнения лабораторной работы я познакомился с решением СЛАУ методом простой итерации, реализовав его в программном виде на языке Java.