Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ

по лабораторной работе №1
«Системы линейных алгебраических уравнений»

по дисциплине «**Вычислительная математика**»

Автор: Пряничников Кирилл Сергеевич

Факультет: ПИиКТ

Группа: P32202

Преподаватель: Перл Ольга Вячеславовна

Санкт-Петербург, 2023

Описание метода Гаусса-Зейделя

Метод Гаусса-Зейделя — численный метод для решения систем линейных алгебраических уравнений, который является улучшенной версией метода простой итерации.
Данный метод принимает на вход систему линейных уравнений вида $AX = B$, где A - матрица коэффициентов, X - вектор неизвестных и B - вектор правой части уравнения.
Для решения системы уравнений методом Гаусса-Зейделя используется итерационный процесс, в котором значения неизвестных на каждой итерации вычисляются последовательно. На каждой итерации новые значения неизвестных вычисляются с использованием уже известных значений в текущей итерации.

Итерационный процесс будет сходиться, если:

$$|a_{ii}| \geq \sum_{i \neq k} |a_{ik}| \ (i, k = 1, 2, \dots, n)$$

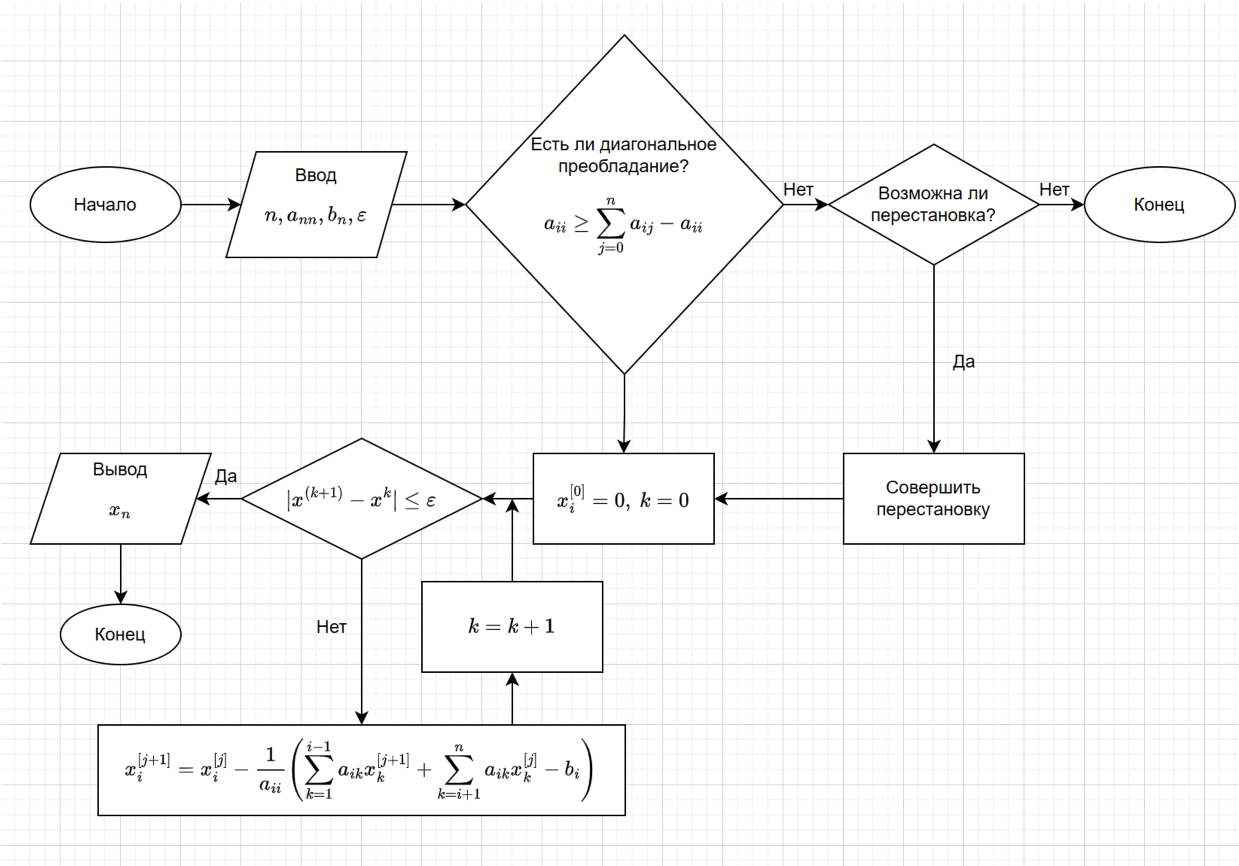
При этом хотя бы для одного уравнения должно соблюдаться условие:

$$|a_{ii}| > \sum_{i \neq k} |a_{ik}| \ (i, k = 1, 2, \dots, n)$$

В методе Гаусса-Зейделя на каждой итерации k значения неизвестных вычисляются следующим образом:

$$x_i^{[j+1]} = x_i^{[j]} - \frac{1}{a_{ii}} \left(\sum_{k=1}^{i-1} a_{ik} x_k^{[j+1]} + \sum_{k=i+1}^{n} a_{ik} x_k^{[j]} - b_i \right)$$

Блок-схема метода Гаусса-Зейделя



Функция, реализующая метод Гаусса-Зейделя на Java

```
import java.util.*;

public class GaussZeidelMethod {
    public static String errorMessage;
    public static boolean isMethodApplicable = true;
    public static int[][] matrixOfDominantElements;
    public static int numberOfIterations;
    public static List<Double> listOfDifferences = new ArrayList<>();

    public static List<Double> solveByGaussSeidel(int n, List<List<Double>> matrix, double epsilon) {
        if (hasDiagonalDominance(n, matrix)) {
            matrix = convertToDiagonalDominance(n, matrix);
            return solveEquation(n, matrix, epsilon);
        } else {
            isMethodApplicable = false;
            errorMessage = "Для решения матрицы методом Гаусса-Зейделя отсутствует диагональное преобладание";
            System.err.println(errorMessage);
            System.exit(0);
        }
        return null;
    }
    private static boolean hasDiagonalDominance(int n, List<List<Double>> matrix) {
        matrixOfDominantElements = getMatrixOfDominantElements(n, matrix);
        for (int row = 0, column = 0; row < n; row++, column++) {
            if ((getRowSum(n, row) != 1) || (getColumnSum(n, column) != 1)) return false;
        }
        return true;
    }
    private static int[][] getMatrixOfDominantElements(int n, List<List<Double>> matrix) {
        int[][] result = new int[n][n];
        for (int i = 0; i < n; i++) {
            double rowSum = 0.;
            for (int j = 0; j < n; j++) {
                rowSum += Math.abs(matrix.get(i).get(j));
            }
            for (int j = 0; j < n; j++) {
                if (2 * Math.abs(matrix.get(i).get(j)) > rowSum) {
                    result[i][j] = 1;
                } else {
                    result[i][j] = 0;
                }
            }
        }
        return result;
    }
    private static List<List<Double>> convertToDiagonalDominance(int n, List<List<Double>> matrix) {
        List<Double>[] result = new List[n];
        matrix.toArray(result);
        for (int i = 0; i < n; i++) {
            if (matrixOfDominantElements[i][i] == 1) continue;
            for (int j = 0; j < n; j++) {
                if (matrixOfDominantElements[i][j] == 1) {
                    List<Double> temp = result[i];
                    result[i] = result[j];
                    result[j] = temp;

                    int[] temp2 = matrixOfDominantElements[i];
                    matrixOfDominantElements[i] = matrixOfDominantElements[j];
                    matrixOfDominantElements[j] = temp2;
                }
            }
        }
        return List.of(result);
    }
    private static List<Double> solveEquation(int n, List<List<Double>> matrix, double epsilon) {
        List<Double> previousOccurence = getZeroOccurence(n, matrix);
        List<Double> currentOccurence = getNextOccurence(n, matrix, previousOccurence);
        numberOfIterations = 1;
        while (!differenceIsLessThanEpsilon(previousOccurence, currentOccurence, epsilon)) {
            previousOccurence = currentOccurence;
            currentOccurence = getNextOccurence(n, matrix, previousOccurence);
            numberOfIterations++;
        }
        return currentOccurence;
    }
    private static List<Double> getZeroOccurence(int n, List<List<Double>> matrix) {
        List<Double> result = new ArrayList<>();
        for (int row = 0; row < n; row++) {
            result.add(0.);
        }
        return result;
    }
    private static List<Double> getNextOccurence(int n, List<List<Double>> matrix, List<Double>
previousOccurence) {
        List<Double> result = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            Double currentResult = matrix.get(i).get(n);
```

```java
            for (int j = 0; j < n; j++) {
                try {
                    currentResult -= result.get(j) * matrix.get(i).get(j);
                } catch (IndexOutOfBoundsException exception) {
                    currentResult -= previousOccurence.get(j) * matrix.get(i).get(j);
                }
            }
            currentResult = currentResult / matrix.get(i).get(i) + previousOccurence.get(i);
            result.add(currentResult);
        }
        return result;
    }
    private static int getRowSum(int n, int row) {
        int result = 0;
        for (int column = 0; column < n; column++) {
            result += matrixOfDominantElements[row][column];
        }
        return result;
    }
    private static int getColumnSum(int n, int column) {
        int result = 0;
        for (int row = 0; row < n; row++) {
            result += matrixOfDominantElements[row][column];
        }
        return result;
    }
    private static boolean differenceIsLessThanEpsilon(List<Double> previousOccurence, List<Double>
currentOccurence, double epsilon) {
        listOfDifferences = new ArrayList<>();
        for (int i = 0; i < previousOccurence.size(); i++) {
            listOfDifferences.add(Math.abs(currentOccurence.get(i) - previousOccurence.get(i)));
            if (listOfDifferences.get(i) > epsilon) return false;
        }
        return true;
    }
}
```

Примеры работы программы

Пример 1:

```
Введите максимальную допустимую погрешность измерений: 0.0001

Ваша матрица:
    [5.0, 2.0, 1.0, -4.0]
    [1.0, 8.0, -2.0, 6.0]
    [6.0, 3.0, 10.0, -5.0]

Было выполнено 9 итераций
Столбец решений:
    x1 = -1,131952082
    x2 = 0,870961743
    x3 = -0,082117274
Столбец погрешностей:
    e1 = 0,000028158
    e2 = 0,000017004
    e3 = 0,000011794
```

Результат выполнения в онлайн-решателе:

Ответ:

$x_1 = -1{,}13196$

$x_2 = 0{,}87097$

$x_3 = -0{,}08211$

Пример 2:

```
Введите максимальную допустимую погрешность измерений: 0.0001

Ваша матрица:
    [8.6, 4.1, -3.2, 5.8]
    [3.5, -2.9, 9.8, 10.2]
    [6.1, 12.3, -5.2, 4.8]

Было выполнено 10 итераций
Столбец решений:
    x1 = 0,828259434
    x2 = 0,336534657
    x3 = 0,844596172
Столбец погрешностей:
    e1 = 0,000035161
    e2 = 0,000049369
    e3 = 0,000027167
```

Результат выполнения в онлайн-решателе:

Ответ:

x_1=0,8282

x_2=0,3366

x_3=0,8446

Пример 3:

```
Ваша матрица:
    [8.1, 16.2, -3.1, 1.4, 0.6, 1.7]
    [-1.2, 5.8, 20.8, -3.9, 4.5, 18.3]
    [15.3, -2.5, 0.7, -9.8, 1.5, -0.4]
    [5.8, 12.3, 6.5, -28.9, 2.3, 1.7]
    [1.9, 3.4, -6.1, 2.8, 15.9, -22.6]

Было выполнено 9 итераций
Столбец решений:
    x1 = 0,225646902
    x2 = 0,225717219
    x3 = 1,116317812
    x4 = 0,245145017
    x5 = -1,111511638
Столбец погрешностей:
    e1 = 0,000034863
    e2 = 0,000032064
    e3 = 0,000022668
    e4 = 0,000004819
    e5 = 0,000012236
```

Результат выполнения в онлайн-решателе

Ответ:

x_1=0,2257

x_2=0,2257

x_3=1,1163

x_4=0,2451

x_5=−1,1115

Вывод

В отличие от метода простой итерации, в котором каждая переменная на каждой итерации изменяется на основе предыдущих значений, метод Гаусса-Зейделя использует обновленные значения переменных во время вычисления следующих переменных. За счёт этого в методе Гаусса-Зейделя используется меньшее количество итераций.
Метод Гаусса-Зейделя имеет быстроту сходимости и может быть эффективен для решения систем линейных уравнений, если матрица А является диагонально преобладающей. А также метод может быть эффективным на больших матрицах, так как он решает систему уравнений постепенно, не храня всю матрицу целиком в памяти, в отличие от прямых методов. Главный недостаток метода относительно прямых методов заключается в том, что сложнее найти систему, которая будет соответствовать требованиям.
Алгоритмическая сложность метода Гаусса-Зейделя составляет $O(l \cdot n^2)$, где l – количество итераций, n - размерность матрицы А.
Метод Гаусса-Зейделя может привести к ошибкам округления при вычислении новых значений переменных. Однако, в целом, численная ошибка метода Зейделя не сильно отличается от численной ошибки метода простой итерации.
Подводя итог, метод Гаусса-Зейделя является эффективным методом для решения систем линейных уравнений.