Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
"Национальный исследовательский университет ИТМО"

Факультет Программной Инженерии и Компьютерной Техники

Домашняя работа №1

по дисциплине

«Вычислительная метематика»

Вариант 7

Выполнил:
Студент группы P3210
Мальков Павел Александрович
Преподаватель:
Машина Екатерина Алексеевна

Санкт Петербург
2024

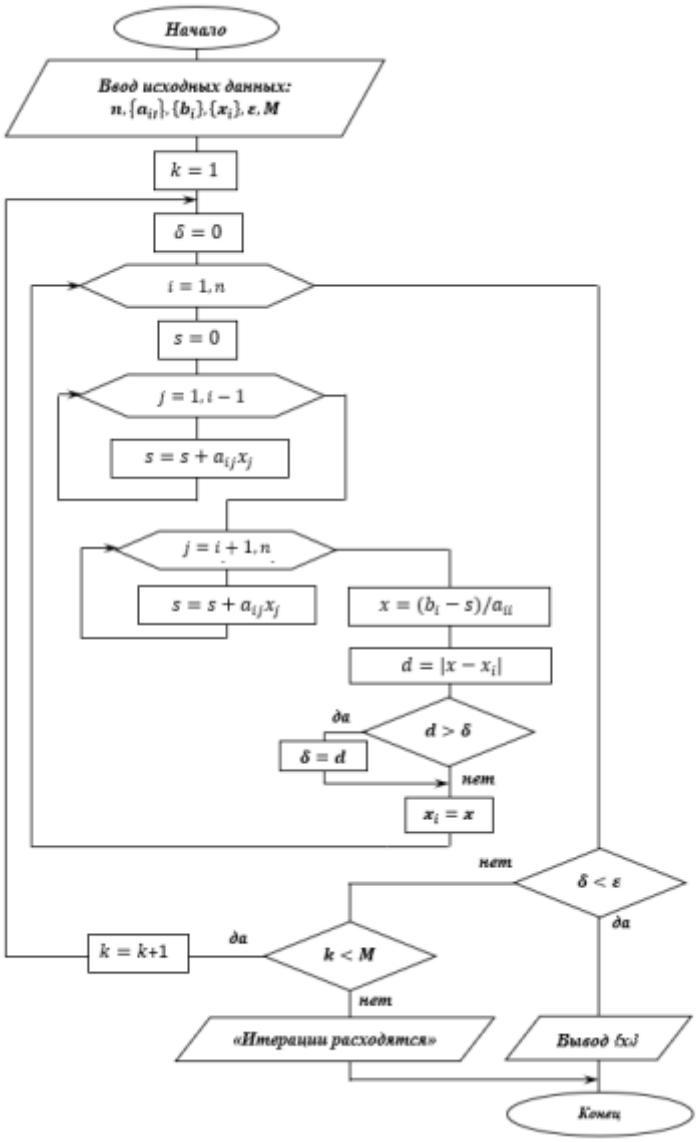
Метод Простых итераций (или метод простой итерации) - это численный метод для приближенного решения уравнений или систем уравнений. Он основан на идее преобразования исходного уравнения в эквивалентное уравнение, которое может быть решено итерационно. Основная идея метода заключается в следующем: пусть дано уравнение f(x) = 0, где f(x) - некоторая функция. Метод Простых итераций предлагает преобразовать это уравнение к виду x = g(x), где g(x) - другая функция. Затем начальное приближение x0 выбирается произвольным образом, и затем последовательно вычисляются значения x1 = g(x0), x2 = g(x1), x3 = g(x2), и так далее.

Таким образом, получается последовательность приближений x0, x1, x2, ... Условие сходимости метода Простых итераций обычно проверяется через анализ производной функции g(x). Для того чтобы метод сходился, необходимо, чтобы модуль производной g'(x) был меньше 1 на всей области сходимости. Если это условие выполняется, то последовательность x0, x1, x2, ... сходится к решению уравнения f(x) = 0. Выбор начального приближения x0 также оказывает влияние на сходимость метода. Неподходящее начальное приближение может привести к расходимости или замедлению сходимости. Поэтому важно выбирать начальное приближение близкое к истинному решению уравнения.

Метод Простых итераций может быть применен для решения различных задач, таких как нахождение корней уравнений, решение систем уравнений, решение интегральных уравнений и других задач. Он имеет простую структуру и легко реализуется на компьютере.

Однако метод Простых итераций может быть медленным и требовать большого числа итераций для достижения требуемой точности. Для улучшения сходимости метода можно применять различные модификации, такие как метод Ньютона или метод секущих.

Блок-схема численного метода



Код решения

```
package core;

import dataclasses.SLAEVector;
import exceptions.InvalidDiagonalException;

import java.util.*;

public class SLAESolver {
    private final double DEFAULT_MAX_MATRIX_LINE_VALUE = 0;

    private long iterationsCount = 0;

    private Stack<SLAEVector> errorsStack = new Stack<>();

    private int roundAccuracy;

    public record Solution(SLAEVector solutionVector, long iterationsCount,
Stack<SLAEVector> errorsStack) {
        @Override
        public String toString() {
            return String.format("Solution vector: %s\n"
                                 + "Iterations count: %s\n"
                                 + "Errors list:\n[%s]",
                    solutionVector,
                    iterationsCount,
                    String.join("\n",
errorsStack.stream().map(SLAEVector::toString).toArray(String[]::new)));
        }
    }

    public SLAESolver(int roundAccuracy) {
        this.roundAccuracy = roundAccuracy;
    }

    /**
     * Возвращает объект к исходному состоянию
     */
    private void clearFields() {
        iterationsCount = 0;
        errorsStack = new Stack<>();
    }

    /**
     * Округляет числа с заданной в конструкторе точностью
     * @param val
     * @return
     */
    protected double round(double val) {
        final double multiplier = Math.pow(10, roundAccuracy);
        val *= multiplier;
        val = Math.round(val);
        return val / Math.pow(10, roundAccuracy);
    }

    /**
     * Вычисляет текущую погрешность и сохраняет в стек
     * @param newVector
     * @param oldVector
     * @return максимальная погрешность
     */
    protected double calcError(SLAEVector newVector, SLAEVector oldVector) {
        double maxError = 0;
```

```
        SLAEVector errors = new SLAEVector(new
double[newVector.values().length]);
        for (int i = 0; i < newVector.values().length; i++) {
            double error = Math.abs(newVector.values()[i] -
oldVector.values()[i]);
            if (error > maxError) maxError = error;
            errors.values()[i] = round(error);
        }
        errorsStack.push(errors);
        return maxError;
    }

    /**
     * Рекурсивный процесс вычисления вектора-результата
     * @param matrix
     * @param accuracy
     * @param lastVector
     * @return
     */
    protected SLAEVector calculate(double[][] matrix, double accuracy,
SLAEVector lastVector) {
        SLAEVector currentVector = new SLAEVector(new double[matrix.length]);
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix.length; j++) {
                if (j < i) currentVector.values()[i] += matrix[i][j] *
lastVector.values()[j];
                if (j == matrix.length - 1) {
                    currentVector.values()[i] += matrix[i][j];
                    continue;
                }
                if (j >= i) currentVector.values()[i] += matrix[i][j] *
lastVector.values()[j + 1];
            }
            currentVector.values()[i] = round(currentVector.values()[i]);
        }

        double error = calcError(currentVector, lastVector);

        iterationsCount++;

        if (error < accuracy) return currentVector;
        return calculate(matrix, accuracy, currentVector);
    }

    /**
     * Получаем нулевой вектор вычислений
     * @param matrix
     * @return
     */
    protected SLAEVector getStartVector(double[][] matrix) {
        double[] dValues = new double[matrix.length];
        for (int i = 0; i < matrix.length; i++) {
            dValues[i] = matrix[i][matrix.length - 1];
        }
        return new SLAEVector(dValues);
    }

    /**
     * Выражает диагональные элементы через другие
     * @param matrix
     * @return
     */
    protected double[][] transformMatrix(double[][] matrix) {
        double[][] resultMatrix = new double[matrix.length][matrix.length];
```

```
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix.length + 1; j++) {
                if (j == i) continue;
                if (j < i) resultMatrix[i][j] = 0 - matrix[i][j] /
matrix[i][i];
                if (j > i) resultMatrix[i][j - 1] = 0 - matrix[i][j] /
matrix[i][i];
            }
        }
        return resultMatrix;
    }

    /**
     * Возвращает массив индексов-кандидатов для строк, создающих
диагональное преобладание
     * @param matrix
     * @return
     */
    protected List<List<Integer>> getIndexes(double[][] matrix) {
        List<List<Integer>> indexes = new ArrayList<>();
        for (int i = 0; i < matrix.length; i++) indexes.add(new
LinkedList<>());
        for (int i = 0; i < matrix.length; i++) {
            double max =
Arrays.stream(matrix[i]).map(Math::abs).limit(matrix.length).max().orElse(DEF
AULT_MAX_MATRIX_LINE_VALUE);
            for (int j = 0; j < matrix.length; j++) {
                if (Math.abs(matrix[i][j]) == max
                        &&
Arrays.stream(matrix[i]).limit(matrix.length).map(Math::abs).sum() - max <=
max)
                    indexes.get(j).add(i); //  Запоминаем строку, подходящую
для данного столбца
            }
        }
        return indexes;
    }

    /**
     * Возвращает массив отобранных индексов для строк, создающих
диагональное преобладание
     * @param matrix
     * @return
     * @throws InvalidDiagonalException
     */
    protected int[] getMatrixLinesIndexes(double[][] matrix) throws
InvalidDiagonalException {
        List<List<Integer>> indexes = getIndexes(matrix); //  Индексы,
удовлетворяющие для каждого места в диагонали
        int[] resultIndexes = new int[matrix.length]; //  Отобранные индексы
строк для диагонального преобладания
        boolean key = false; //  На случай, если нет ни одного строго
удовлетворения неравенство диагонального преобразования, ключ будет указывать
на это
        for (int i = 0; i < 2; i++) { //  Две итерации, так как число
кандидатов на одну позицию не может превысить 2 -> будут рассмотрены все
            for (int j = 0; j < indexes.size(); j++) { //  Проходимся по
всему списку индексов-кандидатов
                if (i == 0 && indexes.get(j).size() == 0)
                    throw new InvalidDiagonalException(); //  Если кандидат
на позицию изначально отсутствует -> диагонали нет
                if (indexes.get(j).size() == 1) {
                    key = true;
                    resultIndexes[j] = indexes.get(j).get(0);
```

```
                    for (var e : indexes)
                        e.remove(Integer.valueOf(resultIndexes[j])); //
Удаляем отобранный индекс из списка кандидатов
                }
            }
        }
        if (!key) throw new InvalidDiagonalException();
        return resultIndexes;
    }

    /**
     * Выполняет перестановки строк в матрицы
     * @param matrix
     * @return матрица с диагональным преобладанием
     * @throws InvalidDiagonalException
     */
    protected double[][] transformDiagonal(double[][] matrix) throws
InvalidDiagonalException {
        int[] resultIndexes = getMatrixLinesIndexes(matrix);
        double[][] resultMatrix = new double[matrix.length][matrix.length +
1];

        for (int i = 0; i < resultIndexes.length; i++) {
            resultMatrix[i] = matrix[resultIndexes[i]];
        }

        return resultMatrix;
    }

    /**
     * Приводим СЛАУ к однородному виду
     * @param matrix
     */
    protected void negLastMatrixColumn(double[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            matrix[i][matrix.length] *= (-1);
        }
    }

    /**
     * Запускает процесс вычисления вектора-решения
     * @param matrix
     * @param accuracy
     * @return Контейнер данных, содержащий основную информацию о решении
     * @throws InvalidDiagonalException
     */
    public Solution solve(double[][] matrix, double accuracy) throws
InvalidDiagonalException {
        negLastMatrixColumn(matrix);
        matrix = transformDiagonal(matrix);
        matrix = transformMatrix(matrix);
        SLAEVector startVector = getStartVector(matrix);
        SLAEVector solutionVector = calculate(matrix, accuracy, startVector);
        Solution solution = new Solution(solutionVector, iterationsCount,
errorsStack);
        clearFields();
        return solution;
    }

    public void setRoundAccuracy(int roundAccuracy) {
        this.roundAccuracy = roundAccuracy;
    }
}
```

Тестовые данные

SLAESolver.java data.txt

```
3
0.01
15 3 5 14
1 12 6 12
9 3 50 13
```

Результат выполнения

```
Select data source (file/console)
> file
Enter file path:
> data.txt
Solution vector: (0.7259; 0.8999; 0.0735)
Iterations count: 5
Errors list:
[(0.2866; 0.2078; 0.228)
(0.1175; 0.1379; 0.0641)
(0.0489; 0.0418; 0.0295)
(0.0182; 0.0188; 0.0113)
(0.0076; 0.0072; 0.0044)]
>
Process finished with exit code 0
```

Вывод

- Метод Простых итераций является итерационным численным методом для приближенного решения уравнений или систем уравнений.
- Основная идея метода заключается в преобразовании исходного уравнения в эквивалентное уравнение, которое может быть решено итерационно.
- Метод Простых итераций может быть применен для решения различных задач, таких как нахождение корней уравнений, решение систем уравнений, решение интегральных уравнений и других задач.
- Основным достоинством метода Простых итераций является его простота и универсальность. Однако он может быть медленным и требовать большого числа итераций для достижения требуемой точности.