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**

Преподаватель:
Машина Е. А.

Выполнил:
Вальц Мартин
Группа: P3210

Санкт-Петербург, 2024 г

Цель работы

Изучить численные методы решения систем линейных алгебраических уравнений и реализовать один из них средствами программирования.

Описание метода

Итерационные методы дают возможность для системы (1) построить последовательность векторов $x^{(0)}, x^{(1)}, \dots, x^{(k)}$, пределом которой должно быть точное решение $x^{(*)}$: $x^{(*)} = \lim_{k \to \infty} x^{(k)}$ Построение последовательности заканчивается, как только достигается желаемая точность.

Приведем систему уравнений, выразив неизвестные $x1, x2, \dots, xn$ соответственно из первого, второго и т.д. уравнений системы.

Листинг программы

Jacobi.java

```
package lab1.algebra;

public class Jacobi {
    private static final int MAX_PERMUTATIONS = 300;

    private double[][] matrix;
    private final double[][] diagDomMatrix;
    private final int size;

    private double[] rootsSolution;
    private double[] errorMargins;
    private int iterations;

    public Jacobi(double[][] matrix, int size) {
        this.matrix = matrix;
        this.diagDomMatrix = matrix;
        this.size = size;
        iterations = 1;
    }

    public boolean makeDominant() {
        int permutations = 0;
        if (!hasSolution()) return false;
        while (MAX_PERMUTATIONS > permutations) {
            if (isDiagonallyDominant()) {
                matrix = diagDomMatrix;
                return true;
            } else {
                permuteRows();
                permutations++;
            }
        }
        return false;
    }
```

```
public boolean isDiagonallyDominant() {
    for (int i = 0; i < size; i++) {
        double sumRow = 0;
        for (int j = 0; j < size; j++) sumRow += Math.abs(diagDomMatrix[i][j]);

        // удаление диагонального элемента
        sumRow -= Math.abs(diagDomMatrix[i][i]);

        // проверка на то, что диагональный элемент меньше не диагональных
        if (Math.abs(diagDomMatrix[i][i]) < sumRow)
            return false;
    }
    return true;
}

public void permuteRows() {
    int maxItemIndex = 0;
    int rowMaxItem = 0;
    for (int i = 0; i < size; i++) {
        maxItemIndex = 0;
        for (int j = 0; j < size; j++) {
            // проходимся по ряду. ищем индекс максимального в ряду
            if (Math.abs(diagDomMatrix[i][maxItemIndex]) < Math.abs(diagDomMatrix[i][j])) {
                maxItemIndex = j;
            }
        }
        // если найденный элемент не стоит на диагонали, то обновляем значение ряда
        if (diagDomMatrix[i][maxItemIndex] != diagDomMatrix[i][i]) {
            rowMaxItem = i;
            break;
        }
    }
    //перестановка элементов в соответствии с найденным элементом
    for (int i = 0; i < diagDomMatrix[rowMaxItem].length; i++) {
        double temp = diagDomMatrix[rowMaxItem][i];
        diagDomMatrix[rowMaxItem][i] = diagDomMatrix[maxItemIndex][i];
        diagDomMatrix[maxItemIndex][i] = temp;
    }
}

public boolean hasSolution() {
    for (int i = 0; i < size; i++) {
        double e = 0;
        for (int j = 0; j < size; j++) {
            if (get(i,j) != 0) e = get(i, j);
        }
        if (e == 0) return false;
    }
    return true;
}


public double get(int row, int column) {
```

```
        return matrix[row][column];
    }

    public double getFreeMember(int row) {
        return matrix[row][size];
    }

    @Override
    public String toString() {
        StringBuilder s = new StringBuilder();
        s.append("Решение:\n");
        s.append("+————————————————+————————————————————————————————+
————————————————————————————————+\n")
         .append("|переменная| значение             | отклонение          |\n")
         .append("+————————————————+————————————————————————————————+
————————————————————————————————+\n");
        for (int i = 0; i < rootsSolution.length; i++) {
            s.append(String.format("|   x_%02d   | %-20.16f | % -20.16f |", i+1, rootsSolution[i],
errorMargins[i])).append("\n");
        }
        s.append("+————————————————+————————————————————————————————+
————————————————————————————————+\n");
        s.append("Всего ").append(iterations).append(" итераций сделано.\n");
        return s.toString();
    }

    public String printSystem() {
        StringBuilder s = new StringBuilder();
        s.append("Система уравнений:\n");
        for (double[] row : matrix) {
            for (int j = 0; j < row.length; j++) {
                if (j != row.length - 1)
                    s.append(row[j]).append(" * ").append("x_").append(j+1).append((j == row.length-2) ? " = " : "
+ ");
                else
                    s.append(row[j]);
            }
            s.append("\n");
        }
        return s.toString();
    }

    public int getSize() {
        return size;
    }
    public void setIterations(int iterations) {
        this.iterations = iterations;
    }
    public void setErrorMargins(double[] errorMargins) {
        this.errorMargins = errorMargins;
    }
    public void setRootsSolution(double[] rootsSolution) {
        this.rootsSolution = rootsSolution;
```

```
    }
}
```

SystemSolver.java

```
package lab1.algebra;

import java.util.Arrays;

public class SystemSolver {

    private final int MAX_ITERATIONS = 100;

    public boolean solveWithJacoby(Jacobi jacobi, double accuracy) {
        int iterations = 0;
        int n = jacobi.getSize();
        double[] actual = new double[n];
        double[] prev = new double[n];
        Arrays.fill(actual, 0);
        Arrays.fill(prev, 0);
        do {
            prev = actual.clone();
            // пробегаемся по рядам
            for (int i = 0; i < n; i++) {
                double sum = jacobi.getFreeMember(i);
                // пробегаемся по элементам
                for (int j = 0; j < n; j++) {
                    // если элемент не диагональный, то мы из суммы (недобудущего элемента), вычитаем предыдущий * коэфф
                    if (j != i)
                        sum -= jacobi.get(i, j) * prev[j];
                }
                // иначе мы присваиваем новый элемент в текущий вектор
                actual[i] = 1 / jacobi.get(i,i) * sum;
            }
            iterations++;
            if (iterations > MAX_ITERATIONS) {
                return false;
            }
            // проверка на достижение нужной погрешности
        } while (getMaxAbsValue(subtractFromVector(actual, prev)) > accuracy);

        jacobi.setIterations(iterations);
        jacobi.setRootsSolution(actual);
        jacobi.setErrorMargins(subtractFromVector(actual, prev));
        return true;
    }


    public double[] subtractFromVector(double[] v1, double[] v2) {
        double[] r = new double[v1.length];
        for (int i = 0; i < r.length; i++)
```

```
            r[i] = v1[i] - v2[i];
        return r;
    }

    public double getMaxAbsValue(double[] v1) {
        return Arrays.stream(v1).map(Math::abs).max().getAsDouble();
    }
}
```

Вывод:

В результате выполнения данной лабораторной работой я познакомился с численными методами решения математических задач на примере систем алгебраических уравнений, реализовав на языке программирования Java метод простых итераций.