Министерство науки и высшего образования Российской Федерации
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 № 7

Выполнил:
студент группы P3213
Нягин Михаил Алексеевич
Проверила:
Машина Екатерина
Алексеевна

г. Санкт-Петербург
2024 год

Цель работы:

Разработать программу, которая будет реализовать метод простых итераций:

• 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: $x1, x2, \dots, xn$

• Вывод количества итераций, за которое было найдено решение.

• Вывод вектора погрешностей: $|x_i^{(k)} - x_i^{(k-1)}|$

Описание работы:

Итерационные методы – это методы последовательных приближений. Задается некоторое начальное приближение. Далее с помощью определенного алгоритма проводится один цикл вычислений - итерация. В результате итерации

находят новое приближение. Итерации проводятся до получения решения с требуемой точностью.

Дана матрица вида

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

Для которой должно выполняться достаточное условие сходимости

Теорема. Достаточным условием сходимости *итерационного процесса* к решению системы при любом начальном векторе $x_i^{(0)}$ является выполнение условия преобладания диагональных элементов или доминирование диагонали:

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|, \qquad i = 1, 2, \ldots, n$$

Если достаточное условие выполняется, то мы выражаем неизвестные:

$$\begin{cases} x_1 = \frac{a_{12}}{a_{11}}x_2 + \frac{a_{13}}{a_{11}}x_3 + \cdots + \frac{a_{1n}}{a_{11}}x_n - \frac{b_1}{a_{11}} \\ x_2 = \frac{a_{21}}{a_{22}}x_1 + \frac{a_{23}}{a_{22}}x_3 + \cdots + \frac{a_{2n}}{a_{22}}x_n - \frac{b_2}{a_{22}} \\ \ldots\ldots \\ x_n = \frac{a_{n1}}{a_{nn}}x_1 + \frac{a_{n2}}{a_{nn}}x_2 + \cdots + \frac{a_{n-1n-1}}{a_{nn}}x_{n-1} - \frac{b_n}{a_{nn}} \end{cases} \qquad (6)$$

После чего выделяется матрица коэффициентов C и вектор свободных член: D

$$c_{ij} = \begin{cases} 0, & \text{при } i = j \\ -\frac{a_{ij}}{a_{ii}}, & \text{при } i \neq j \end{cases}$$

$$d_i = \frac{b_i}{a_{ii}} \quad i = 1, 2, \ldots, n$$

Для последующего приблежения для нормы матрицы(матрицы C) должно выполняться условие сходимости, то есть:

$||C|| < 1$

Далее вектор свободных членов принимается за начальное(нулевое) приближение, а все последующие приближения вычисляются следующим образом:

$$\overrightarrow{x^{(k)}} = c\,\vec{x}^{(k-1)} + \vec{d}$$

Также высчитывается критерий по абсолютным отклонением:

$$\max_{1 \le i \le n} \left| x_i^{(k)} - x_i^{(k-1)} \right| \le \varepsilon$$

Класс SimpleIteration

```
package Computational.math;

import Computational.math.Exceptions.DiagonalPredominanceException;
import Computational.math.utils.UtilsForSimpleIteration;

import static java.lang.System.exit;

public class SimpleIteration {
    private final Double[][] system;
    private final Double[] answers;
    private Double[][] norm;
    private Double[] startApproach;
    private Double[] lastApproach;
    private final double accuracy;
    private double absoluteDeviations = 1;
    private static int MAX_ITERATION = 10000;
    private static int iterationNumber = 0;


    public static int getIterationNumber() {
        return iterationNumber - 1;
    }


    public SimpleIteration(Double[][] system, Double[] answers, double accuracy)
{
        this.system = system;
        this.answers = answers;
        this.accuracy = accuracy;
    }

    public void swapRows(int positionFrom, int positionTo) {
        Double[] tmpSystemRow = this.system[positionTo];
        Double tmpAnswer = this.answers[positionTo];
        this.answers[positionTo] = this.answers[positionFrom];
        this.answers[positionFrom] = tmpAnswer;
        this.system[positionTo] = this.system[positionFrom];
        this.system[positionFrom] = tmpSystemRow;
    }
    /**
     * Приводит систему к условиям преобладания диагоналей или выбрасывает
исключения на случай если их нет
     * @throws DiagonalPredominanceException если невозможно привести данную
систему к условиям преобладания диагоналей
     * @see #isDiagonalPredominances() метод для проверки диагонального
преобладания
     */
    public void toDiagonalPredominance() throws DiagonalPredominanceException {
        Double[][] matrix = system;
        int rows = system.length;
        int cols = system[0].length;
        for (int i = 0; i < rows; i++) {
            double max = matrix[i][0];
            int maxIndex = 0;
            for (int j = 1; j < cols; j++) {
                if (Math.abs(matrix[i][j]) > Math.abs(max)) {
                    max = matrix[i][j];
                    maxIndex = j;
                }
            }
            if (maxIndex != i) {
                swapRows(maxIndex,i);
```

```
            }
        }
        if(!isDiagonalPredominances()){
            throw new DiagonalPredominanceException("Невозможно привести систему
к преобладанию диагональных элементов");
        }

    }

    /**
     * Метод для проверки диагонального вида матрицы
     * @return true если в матрице преобладает диагональ
     * @see #toDiagonalPredominance()
     */
    public boolean isDiagonalPredominances() {
        int dimension = this.system.length;
        double diagonal = 0;
        double notDiagonalSumAbs = 0;
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                if (i == j) {
                    diagonal = Math.abs(this.system[i][j]);
                } else {
                    notDiagonalSumAbs += Math.abs(this.system[i][j]);
                }
            }
            if (diagonal < notDiagonalSumAbs) {
                return false;
            }
            notDiagonalSumAbs = 0;
        }
        return true;
    }

    /**
     * Вспомогательный метод для построения нормы матрицы(матрицы C)
     * Делит все коэффициенты на коэффициент матрицы
     * @see #expressCoefficient() метод для получения нормы матрицы и начального
приближения
     */
    public void divideByDiagonalCoefficient() {
        for (int i = 0; i < system.length; i++) {
            double divider = system[i][i];
            answers[i] = answers[i] / divider;
            for (int j = 0; j < system[0].length; j++) {
                system[i][j] = system[i][j] / divider;
            }
        }
    }

    /**
     * Функция, позволяющая выделить коэфициенты x1,x2,...,xn, а также найти
начальное приближение(paramApproach)
     * @see #divideByDiagonalCoefficient() вспомогающий метод, выполняющий
деление коэффициентов
     */
    public void expressCoefficient() {
        int dimension = system.length;
        this.norm = new Double[dimension][dimension];
        this.startApproach = answers;
        this.lastApproach = startApproach;
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                norm[i][j] = i != j ? -system[i][j] : 0d;
```

```
            }
        }
    }

    /**
     * Проверка на условие сходимости: ||C|| = max(startApproach) < 1
     * @see #expressCoefficient()
     */
    public boolean convergenceCondition() {
        double sum = 0;
        for (Double[] row : norm) {
            for (int j = 0; j < norm.length; j++) {
                sum += Math.abs(row[j]);
            }
            if (sum >= 1) {
                return false;
            }
            sum = 0;
        }
        return true;
    }

    public Double[] countNewApproach(Double[] approachToCount) {
        Double[] newApproach = new Double[startApproach.length];
        for (int currentRow = 0; currentRow < newApproach.length; currentRow++)
{
            newApproach[currentRow] =
UtilsForSimpleIteration.roundDouble(approximationRow(currentRow,
approachToCount));
        }
        return newApproach;
    }


    public void approximations() {
        Double[] newApproach;
        newApproach = countNewApproach(this.lastApproach);
        if (iterationNumber == 0) {
            iterationNumber++;
            UtilsForSimpleIteration.printFinalTable(this.lastApproach, null);
            return;
        }
        double calculation = calculateAbsoluteDeviations(lastApproach,
newApproach);
        this.lastApproach = newApproach;
        iterationNumber++;
        UtilsForSimpleIteration.printFinalTable(lastApproach, calculation);
    }


    /**
     * Функция для вычисления критерия по абсолютным отклонениям
     */
    public double calculateAbsoluteDeviations(Double[] currentX, Double[]
previousX) {
        double[] tmp = new double[currentX.length];
        double max = 0d;
        for (int i = 0; i < currentX.length; i++) {
            tmp[i] = Math.abs(currentX[i] - previousX[i]);
            max = Math.max(tmp[i], max);
        }
        this.absoluteDeviations = max;
        return max;
    }
```

```
    /**
     * Данный метод является вспомогательным для вычисления приближения
     * @param currentRow номер ряда, для которого вычисляется приближение
     * @return сумма посчитанного приблежния
     * @see #approximations() основной метод
     */
    private Double approximationRow(int currentRow, Double[] approach) {
        double sumRow = 0d;
        for (int i = 0; i < approach.length; ++i) {
            sumRow += norm[currentRow][i] * approach[i];
        }
        return sumRow + startApproach[currentRow];
    }

    public void solve() {
        try {
            toDiagonalPredominance();
            divideByDiagonalCoefficient();
            expressCoefficient();
            if (!convergenceCondition()) {
                System.err.println("Условие сходимости не выполняется");
                exit(-1);
            }
            do {
                approximations();
                MAX_ITERATION--;
            }
            while (!(this.accuracy > this.absoluteDeviations) || MAX_ITERATION
== 0);
            iterationNumber = 0;
            MAX_ITERATION = 10000;
        }
        catch (DiagonalPredominanceException e) {
            System.err.println(e.getMessage());
            iterationNumber = 0;

        }
    }
}
```

Пример ввода с файла:

Вы хотите ввести данные файлом? да/нет

да

Введите название файла

input.txt

k = 0 | 0.6666666666666666 | 0.8888888888888888 | 0.25 | -

---

k = 1 | -0.00926 | 0.71296 | -0.55556 | 0.80556

---

k = 2 | 0.37655 | 0.95268 | -0.19213 | 0.38581

---

k = 3 | 0.09559 | 0.82656 | -0.48663 | 0.2945

---

k = 4 | 0.27784 | 0.92172 | -0.30245 | 0.18418

---

k = 5 | 0.153 | 0.86075 | -0.43027 | 0.12782

---

k = 6 | 0.23626 | 0.9027 | -0.34534 | 0.08493

---

k = 7 | 0.17998 | 0.87476 | -0.40278 | 0.05744

---

k = 8 | 0.21775 | 0.89365 | -0.36422 | 0.03856

---

k = 9 | 0.19231 | 0.88097 | -0.39019 | 0.02597

---

k = 10 | 0.20942 | 0.88951 | -0.37272 | 0.01747

---

k = 11 | 0.1979 | 0.88376 | -0.38448 | 0.01176

---

k = 12 | 0.20565 | 0.88763 | -0.37656 | 0.00792

Сам файл:

```
3
3 2 1
3 5 8
2 9 1

2 2 8
0.01
```

Пример ввода “руками”

Вы хотите ввести данные файлом? да/нет

нет

Введите размерность:

3

Введите матрицу:

2 2 10

10 1 1

2 10 1

Введите вектор ответов:

14 12 13

Введите значение epsilon:

0.01

k = 0 | 1.2 | 1.3 | 1.4 | -

---

k = 1 | 0.93 | 0.92 | 0.9 | 0.5

---

k = 2 | 1.018 | 1.024 | 1.03 | 0.13

---

k = 3 | 0.9946 | 0.9934 | 0.9916 | 0.0384

---

k = 4 | 1.0015 | 1.00192 | 1.0024 | 0.0108

---

k = 5 | 0.99957 | 0.99946 | 0.99932 | 0.00308

Вывод:

В результате данной лабораторной работы я написал программу, которая позволяет решать СЛАУ размера n*n при помощи метода простых итераций.