Федеральное государственное автономное
образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский университет
ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ
ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА № 1

по дисциплине

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №14

Выполнила
Студентка группы P32151
Ярусова Анна Александровна
Преподаватель:
Машина Екатерина Алексеевна

Санкт-Петербург,
2023

Цель

Вычисление решения СЛАУ методом Гаусса-Зейделя.

Задание

1. № варианта определяется как номер в списке группы согласно ИСУ.
2. В программе численный метод должен быть реализован в виде отдельной подпрограммы или класса, в который входные/выходные данные передаются в качестве параметров.
3. Размерность матрицы n<=20 (задается из файла или с клавиатуры - по выбору конечного пользователя).
4. Должна быть реализована возможность ввода коэффициентов матрицы, как с клавиатуры, так и из файла (по выбору конечного пользователя)

   Для итерационных методов должно быть реализовано:

   • 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: $x_1,\ x_2,\ ...,\ x_n$

   • Вывод количества итераций, за которое было найдено решение.

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

Описание метода, расчетные формулы

Метод Гаусса-Зейделя является модификацией метода простой итерации для наиболее быстрого поиска решения за счёт быстрой сходимости. Идея метода: при вычислении компонента $x_i^{(k+1)}$ вектора неизвестных на (k+1)-й итерации используются $x_1^{(k+1)}$, $x_2^{(k+1)}$, … , $x_{i-1}^{(k+1)}$, уже вычисленные на (k+1)-й итерации. Значения остальных компонент $x_{i+1}^{(k+1)}, x_{i+2}^{(k+1)}, \ldots, x_{n}^{(k+1)}$ берутся из предыдущей итерации.

$$x_i^{(k+1)} = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{k+1} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^{k} \quad i = 1, 2, \ldots, n$$

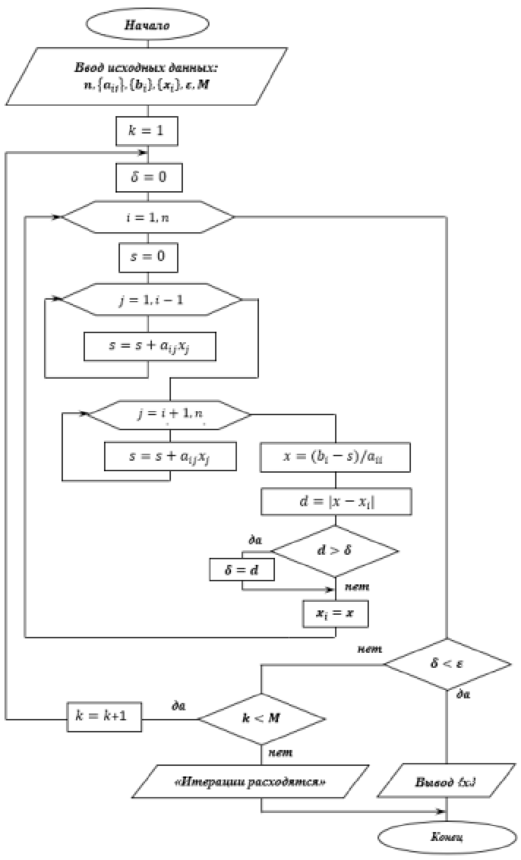
Итерационный процесс продолжается до тех пор, пока:

$$|x_1^{(k)} - x_1^{(k-1)}| \le \varepsilon, \quad |x_2^{(k)} - x_2^{(k-1)}| \le \varepsilon, \quad |x_3^{(k)} - x_3^{(k-1)}| \le \varepsilon$$

В начале, так же, как и в методе простой итерации, проверяется условие преобладания диагональных элементов, и если оно не выполнено, то строчки переставляются местами. Потом выражаются x и проверяется условие сходимости преобразованной матрицы, путём подсчёта нормы матрицы.

Блок-схема

n – порядок матрицы,
ε – погрешность вычислений,
a_{ij} ,b_i – коэффициенты и правые части уравнений системы,
x_i – начальные приближения,
M – максимально допустимое число итераций,
k – порядковый номер итерации;
i – номер уравнения, а также переменного, которое вычисляется в соответствующем цикле;
j – номер элемента вида $a_{ij}x_j^{(k)}$ или $a_{ij}x_j^{(k-1)}$ в правой части соотношения.
Итерационный процесс прекращается либо при выполнения условия:
$\max\limits_{1 \le i \le n} \left| x_i^{(k)} - x_i^{(k-1)} \right| < \boldsymbol{\varepsilon}$, либо при $k = M$, т.е. итерации не сходятся.



Листинг численного метода

```
public Solution solve(MethodData data) {
   EquationData equationData = data.getEquationData();
   Matrix matrix = equationData.getAMatrix();
   if (matrix.getRows() != matrix.getColumns()) {
       throw new IllegalArgumentException("Matrix must be
square");    }
   int n = matrix.getRows();
   double[] xVector;
   double[] xPrevVector = new double[n];
   double[] errorVector;
   int iterations = 0;

   if (!checkDiagonalDominance(matrix)) {
       if (!tryToMakeDiagonallyDominant(equationData)) {
           throw new IllegalArgumentException("Matrix must be
diagonally dominant");
       }
   }

   changeEquation(equationData);

   if (!checkMatrixNorm(equationData.getAMatrix())) {
       throw new IllegalArgumentException("Matrix must have
norm less than 1");
   }

private boolean tryToMakeDiagonallyDominant(EquationData data)
{
   double[][] m = data.getAMatrix().getMatrix();
   int n = data.getAMatrix().getRows();
   double[] b = data.getBVector();
   for (int i = 0; i < n; i++) {
       double sum = 0;
       for (int j = 0; j < n; j++) {
           if (i != j) {
               sum += Math.abs(m[i][j]);
           }
       }
```

```java
            if (Math.abs(m[i][i]) < sum) {
                int maxIndex = i;
                for (int j = i + 1; j < n; j++) {
                    double max = m[j][0];
                    int maxIndex2 = 0;
                    for (int k = 1; k < n; k++) {
                        if (Math.abs(m[j][k]) > Math.abs(max)) {
                            max = m[j][k];
                            maxIndex2 = k;
                        }
                    }
                    if (maxIndex2 == i) {
                        maxIndex = j;
                        break;
                    }
                }
                if (maxIndex == i) {
                    return false;
                }
                double[] temp = m[i];
                m[i] = m[maxIndex];
                m[maxIndex] = temp;
                double tempB = b[i];
                b[i] = b[maxIndex];
                b[maxIndex] = tempB;
            }
        }
        data.getAMatrix().setMatrix(m);
        data.setBVector(b);

        return checkDiagonalDominance(data.getAMatrix());
    }

    private void changeEquation(EquationData equationData) {
        Matrix matrix = equationData.getAMatrix();
        double[][] a = matrix.getMatrix();
        double[] b = equationData.getBVector();
        int n = matrix.getRows();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i != j) {
```

```
                a[i][j] = -a[i][j] / a[i][i];
            }
        }
        b[i] /= a[i][i];
        a[i][i] = 0;
    }
    equationData.getAMatrix().setMatrix(a);
    equationData.setBVector(b);
}

private boolean checkMatrixNorm(Matrix matrix) {
    double[][] a = matrix.getMatrix();
    int n = matrix.getRows();
    double norm = 0;
    for (int i = 0; i < n; i++) {
        double sum = 0;
        for (int j = 0; j < n; j++) {
            sum += Math.abs(a[i][j]);
        }
        if (sum > norm) {
            norm = sum;
        }
    }
    return norm < 1;
}

private boolean checkError(double[] errorVector, double
epsilon) {
    for (double error : errorVector) {
        if (error > epsilon) {
            return false;
        }
    }
    return true;
}

private double[] calculateError(double[] x, double[] xPrev) {
    double[] errorVector = new double[x.length];
    for (int i = 0; i < x.length; i++) {
        errorVector[i] = Math.abs(x[i] - xPrev[i]);
    }
    return errorVector;
```

```
}
```

Пример работы

```
x vector:
x1 = 1.00017808
x2 = 0.999936864
x3 = 0.9999770111999999
error vector:
e1 = 5.019200000000446E-4
e2 = 0.001992863999999983
e3 = 2.981887999999877E-4
iterations: 3
```

Вывод

В ходе выполнения лабораторной работы я узнала о различных численных методах решения СЛАУ, научилась решать СЛАУ методом Гаусса-Зейделя, написала для этого код на языке Java.