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 – “Метод Гаусса – Зейделя”

Группа: P32312

Выполнил: Обляшевский С.А.

Преподаватель:

Перл О. В.

Санкт-Петербург, 2023

Описание метода, расчетные формулы:

Данный метод предназначен для решения СЛАУ. Он является итерационным, что означает, что ответ будет дан с какой-то погрешностью. Метод подходит для решения СЛАУ с большим кол-вом неизвестных, ведь не надо держать в памяти всю матрицу целиком. Но данный метод не способен решать все системы. Главное условие сходимости, при несоблюдении которого метод не работает, представляет собой наличие диагонального преобладания у матрицы. Диагональные преобладание – свойство матрицы, когда на главной диагонали находятся такие элементы, которые по модулю будут больше, чем сумма модулей оставшихся элементов данной строки.

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}|, (i \in \{1, 2, ..., n\})$$

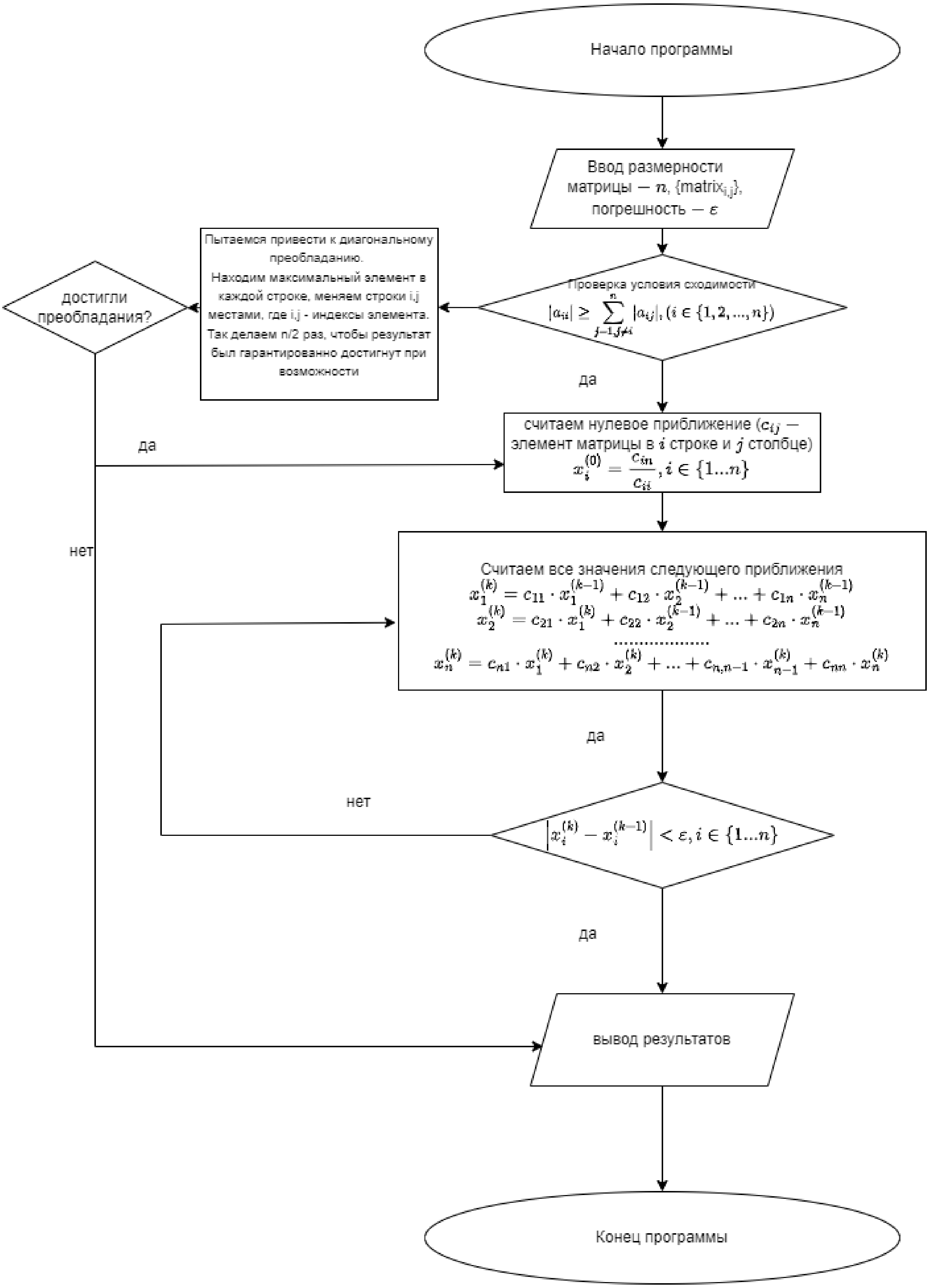
Дальнейший метод – вычисление приближений. Приближение – это один из корней заданного СЛАУ, полученный с какой-то погрешностью. За нулевое приближение берутся следующие значения

$$x_i^{(0)} = \frac{c_{in}}{c_{ii}}, i \in \{1...n\}$$

Все остальные приближения считаются по следующей формуле:

$$x_1^{(k)} = c_{11} \cdot x_1^{(k-1)} + c_{12} \cdot x_2^{(k-1)} + ... + c_{1n} \cdot x_n^{(k-1)}$$
$$x_2^{(k)} = c_{21} \cdot x_1^{(k)} + c_{22} \cdot x_2^{(k-1)} + ... + c_{2n} \cdot x_n^{(k-1)}$$
$$\dots\dots\dots\dots\dots\dots$$
$$x_n^{(k)} = c_{n1} \cdot x_1^{(k)} + c_{n2} \cdot x_2^{(k)} + ... + c_{n,n-1} \cdot x_{n-1}^{(k)} + c_{nn} \cdot x_n^{(k)}$$

Блок – схема:



Начало программы
Ввод размерности матрицы — n, {matrix$_{i,j}$}, погрешность — ε
Проверка условия сходимости
$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}|, (i \in \{1, 2, \dots, n\})$
Пытаемся привести к диагональному преобладанию.
Находим максимальный элемент в каждой строке, меняем строки i,j местами, где i,j - индексы элемента.
Так делаем n/2 раз, чтобы результат был гарантированно достигнут при возможности
достигли преобладания?
да
да
нет
считаем нулевое приближение (c_{ij} — элемент матрицы в i строке и j столбце)
$x_i^{(0)} = \frac{c_{in}}{c_{ii}}, i \in \{1 \dots n\}$
Считаем все значения следующего приближения
$x_1^{(k)} = c_{11} \cdot x_1^{(k-1)} + c_{12} \cdot x_2^{(k-1)} + \dots + c_{1n} \cdot x_n^{(k-1)}$
$x_2^{(k)} = c_{21} \cdot x_1^{(k)} + c_{22} \cdot x_2^{(k-1)} + \dots + c_{2n} \cdot x_n^{(k-1)}$
....
$x_n^{(k)} = c_{n1} \cdot x_1^{(k)} + c_{n2} \cdot x_2^{(k)} + \dots + c_{n,n-1} \cdot x_{n-1}^{(k)} + c_{nn} \cdot x_n^{(k)}$
да
нет
$\left|x_i^{(k)} - x_i^{(k-1)}\right| < \varepsilon, i \in \{1 \dots n\}$
да
вывод результатов
Конец программы

Листинг численного метода:

```java
@Override
public void invoke() {
    makeDiagonal();
    if (isSolvable()){
        approximation();
        printAnswer();
    }
    else
        System.out.println("Не соблюдено условие сходимости.");
}

private void makeDiagonal(){
    int size = matrix.getSize();
    Set<Integer> swappedRows = new HashSet<>();
    for (int i = 0; i < size; i++){
        double maxValue = Double.MIN_VALUE;
        int index = -1;
        for (int j = 0; j < size; j++){
            if (Math.abs(matrix.getA()[i][j]) > maxValue){
                maxValue = Math.abs(matrix.getA()[i][j]);
                index = j;
            }
        }
        if (!swappedRows.contains(index)){
            swapRows(matrix, i, index);
            swappedRows.add(index);
            i--;
        }
    }
}

private void swapRows(Matrix matrix, int firstRow, int secondRow){
    double[] temp = matrix.getA()[firstRow];
    matrix.getA()[firstRow] = matrix.getA()[secondRow];
    matrix.getA()[secondRow] = temp;
    double tempB = matrix.getB()[firstRow];
    matrix.getB()[firstRow] = matrix.getB()[secondRow];
    matrix.getB()[secondRow] = tempB;
}

private boolean isSolvable(){
    double sum;
    for (int i = 0; i < matrix.getSize(); i++){
        sum = 0;
        for (int j = 0; j < matrix.getSize(); j++){
            sum += Math.abs(matrix.getA()[i][j]);
        }
        sum -= Math.abs(matrix.getA()[i][i]);
        if (sum > Math.abs(matrix.getA()[i][i])) return false;
    }
    return true;
}

private boolean isFinish(){
    for (int i = 0; i < matrix.getSize(); i++){
        if (Math.abs(curApproximation[i] - prevApproximation[i]) >
matrix.getEpsilon()) return false;
    }
    return true;
}
```

```java
private void zeroApproximation(){
    stepsCount++;
    curApproximation = new double[matrix.getSize()];
    prevApproximation = new double[matrix.getSize()];
    for (int i = 0; i < matrix.getSize(); i++){
        curApproximation[i] = matrix.getB()[i]/matrix.getA()[i][i];
    }
}

private void approximation(){
    zeroApproximation();
    while (!isFinish()){
        stepsCount++;
        double num;
        prevApproximation = curApproximation;
        curApproximation = new double[matrix.getSize()];
        for (int i = 0; i < matrix.getSize(); i++){
            curApproximation[i] = matrix.getB()[i];
            for (int j = 0; j < matrix.getSize(); j++){
                if (i > j) num = curApproximation[j];
                else num = prevApproximation[j];
                if (i != j){
                    curApproximation[i] -= matrix.getA()[i][j] * num;
                }
            }
            curApproximation[i] /= matrix.getA()[i][i];
        }
    }
    uncertainties = new double[matrix.getSize()];
    for (int i = 0; i  < matrix.getSize(); i++){
        uncertainties[i] = Math.abs(curApproximation[i] -
prevApproximation[i]);
    }
}
```

Примеры и рез-ты работы:

Console input:

```
Введите номер источника входных данных:
1) Файл
2) Консоль
3) Автозаполнение
2
Введите размер матрицы (от 2 до 20):
3
Введите матрицу.
1 1 10 12
12 1 1 14
1 13 1 15
Введите погрешность:
0.001
```

Output:

```
Матрица неизвестных:
     x1 = 1,00003          x2 = 1          x3 = 1
Погрешность значений:
      x1 = 0,0007     x2 = 0,00038     x3 = 0,00003
Решение было найдено за 4 итераций.
```

File input:

```
Введите путь к файлу (Полный или относительный).
Изначальная директория - C:\Users\Honor\IdeaProjects\Math_lab1\.
./src/main/resources/20x20.txt
Введите погрешность:
0.001
```

Output:

```
Матрица неизвестных:
    x1 = -8,75298    x2 = -9,92893     x3 = 8,04695      x4 = 8,1261      x5 = 0,94397    x6 = -8,46996     x7 = 6,91104     x8 = 2,79499    x9 = -9,85699     x10 = 5,483
     x11 = -0,948    x12 = 5,42803    x13 = 0,63399    x14 = 1,46499      x15 = -6,402   x16 = -0,70199      x17 = 1,626    x18 = 9,78801   x19 = -7,17401    x20 = 3,37899
Погрешность значений:
     x1 = 0,00048     x2 = 0,00059     x3 = 0,00014      x4 = 0,0003      x5 = 0,00069     x6 = 0,00058     x7 = 0,00065     x8 = 0,00024      x9 = 0,0003    x10 = 0,00019
      x11 = 0,0001    x12 = 0,00007     x13 = 0,00014    x14 = 0,00003    x15 = 0,00012     x16 = 0,00006    x17 = 0,00008     x18 = 0,00001     x19 = 0,00003     x20 = 0,00002
Решение было найдено за 6 итераций.
```

Вывод:

Метод Гаусса-Зейделя основан на методе простых итераций. В целом, эти методы почти одинаковы, но есть одно отличие – нам нужно использовать все значения предыдущих приближений. А в методе Гаусса-Зейделя происходит комбинирование старых и новых значений, из-за чего метод Гаусса-Зейделя выигрывает по скорости.

$$\left.\begin{array}{l} x_1^{(k+1)} = c_{11}x_1^{(k)} + c_{12}x_2^{(k)} + \ldots + c_{1n}x_n^{(k)} + f_1, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n^{(k+1)} = c_{n1}x_1^{(k)} + c_{n2}x_2^{(k)} + \ldots + c_{nn}x_n^{(k)} + f_n. \end{array}\right\}$$

В целом, главное отличие прямых методов от итерационных – точность решения. Первые дают точное решение, а вторые – приближенные. Но эти методы применимы и не являются взаимоисключающими. В жизни достаточно много моделей, в которой нужны, как и точные значения, так и приближенные.