ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ»

Вычислительная математика

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

Вариант : Метод Гаусса-Зейделя

Студент**:** Назирджанов Н.Ф

Преподаватель: Перл О.В.

Санкт-Петербург
2023г.

1. Описание метода, расчетные формулы

Метод Гаусса-Зейделя

Метод Гаусса-Зейделя является итерационным методом и находит конечные значение переменных последовательным приближением после каждой итерации. Так, имея в исходную матрицу A*X = B строится матрица с вынесенными переменными в каждой строчке (так в первой строке $x_1 = f(x_2, x_3 \ldots) + b_1/a_{11}$, для второй строчки $x_2 = f(x_1, x_3, \ldots) + b_2/a_{22}$ и т.д.)

Отсюда конечные формулы для итераций:

$x_1^0 = d_1 = b_1 / a_{11}$,

$x_2^0 = d_2 = b_2 / a_{22}$,

На каждой итерации вычисляются новые значения для всех переменных x_1, x_2 … методом подстановки в уравнение для соответствующей переменной вместо других переменных их последние значения (так для x_1^1 будут использоваться значения x_2^0, x_3^0, … x_n^0, когда как для вычисления x_2^1 вместо x_1^0 будет подставляться x_1^1 найденное на прошлом шаге этой итерации и так далее – это и есть отличие данного метода от метода простых итераций).

Проверяется присутствует ли диагональное преобладание в данной матрице. Для каждого элемента на главной диагонали должно выполняться:

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

В случае выполнения данного условия:

Тогда приближения к решению системы методом Зейделя определяются следующей системой равенств:

$$x_1^{(k+1)} = c_{11}x_1^{(k)} + c_{12}x_2^{(k)} + \cdots + c_{1n}x_n^{(k)} + d_1$$
$$x_2^{(k+1)} = c_{21}x_1^{(k+1)} + c_{22}x_2^{(k)} + \cdots + c_{2n}x_n^{(k)} + d_2$$
$$x_3^{(k+1)} = c_{31}x_1^{(k+1)} + c_{32}x_2^{(k+1)} + c_{33}x_3^{(k)} \ldots + c_{3n}x_n^{(k)} + d_3$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$x_n^{(k+1)} = c_{n1}x_1^{(k+1)} + c_{n2}x_2^{(k+1)} + \cdots + c_{n\,n-1}x_{n-1}^{(k+1)} + c_{nn}x_n^{(k)} + d_n$$

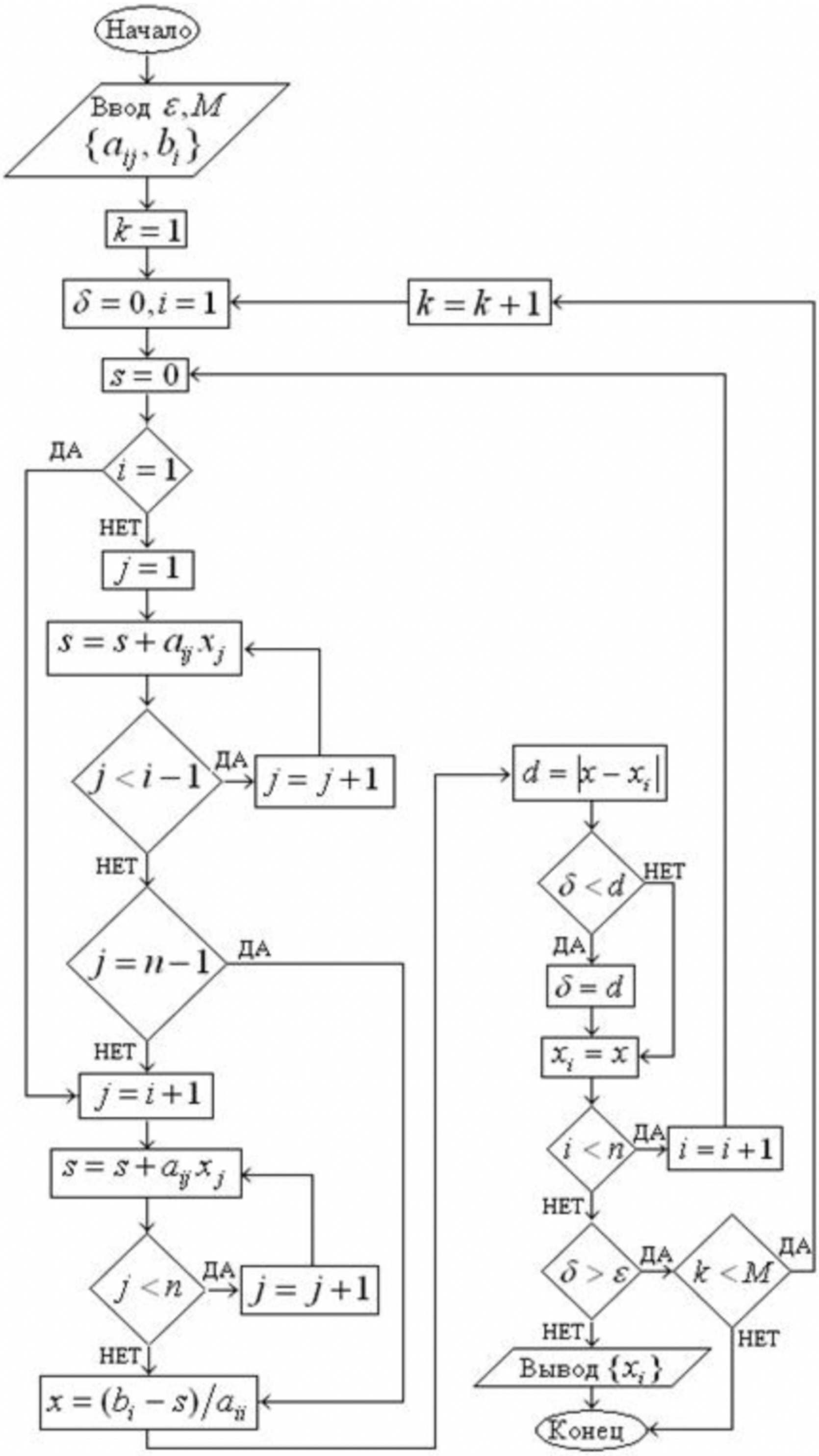
Рабочая формула метода Гаусса-Зейделя:

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

Сам итерационный процесс продолжается до тех пор, пока значения разниц вычисленных переменных от их значений на прошлом шаге не будет меньше, чем заданное заранее значение погрешности ε

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

2. Блок-схема



3. Часть реализации вычисления:

```java
public static void iteration() {
        for (int i = 0; i < size; i++) {
            matrixX1[i][0] = matrixX2[i][0];
        }
         double sumOther;
        for (int i = 0; i < size; i++) {
            sumOther = 0;

            for (int j = 0; j < size; j++) {
                if (j < i) {
                    sumOther += matrixA[i][j] * matrixX2[j][0] / matrixA[i][i];
                } else if (j == i) {
                } else {
                    sumOther += matrixA[i][j] * matrixX1[j][0] / matrixA[i][i];
                }
            }

            matrixX2[i][0] = matrixB[i][0] / matrixA[i][i] - sumOther;
        }
    }
     public static boolean checkAllNewX() {
        for (int i = 0; i < size; i++) {
            if (Math.abs(matrixX2[i][0] - matrixX1[i][0]) > epsilon) {
                return false;
            }
        }
        return true;
    }

    public static void startComputed() {
        int count = 0;

        while (true) {
            iteration();
            count++;
            if (checkAllNewX() || count >= M) {
                break;
            }
        }

        System.out.println("\nПосле всех итераций");
        for (int i = 0; i < size; i++) {
            System.out.println("X" + (i + 1) + " = " + matrixX2[i][0]);
        }

        if (count >= M) {
            System.out.println("\nИтерации не сходятся(на заданном максимальном их количестве)");
        } else {
            System.out.println("\nОбщее количество проделанных итераций = " + count + "\n");
        }

        for (int i = 0; i < size; i++) {
            System.out.println("вектор погрешности вектора X_" + (i + 1) + " = " + Math.abs(matrixX2[i][0] - matrixX1[i][0]));
        }
    }
}
                                                  //If I were asked to choose a modern language to replace Java, I would choose Scala.
```

4. Примеры и результаты работы программы на разных примерах

Матрицы исходных коэффициентов перед переменными взята переменная matrixA

В качестве исходной матрицы свободных членов взята переменная matrixB

после каждой итерации новые значения переменных оказываются в matrixX2

В начале итерации они становятся старыми значениями и переходят в матрицу matrixX1

5. Тесты :

1. Входные данные : (передаются с файла)

```
2 0,00001 50
1 5 4
5 2 3
```

Результат:

Before

1.0 5.0 4.0

5.0 2.0 3.0

After reverse:

5.0   2.0   3.0

1.0   5.0   4.0

После всех итераций

X1 = 0.30434774630399997

X2 = 0.7391304507392

Общее количество проделанных итераций = 6

вектор погрешности вектора X_1 = 9.175039999975709E-7

вектор погрешности вектора X_2 = 1.8350079999951419E-7

2. Входные данные :

```
4 0,001 20
7 0,13 5 1 60
0,51 8 85 0,51 40
86 90 0,075 3 34
9 5 2 100 28
```

Результат:

Before

7.0 0.13 5.0 1.0 60.0

0.51 8.0 85.0 0.51 40.0

86.0 90.0 0.075 3.0 34.0

9.0 5.0 2.0 100.0 28.0

After reverse:

7.0   0.13   5.0   1.0   60.0

86.0   90.0   0.075   3.0   34.0

0.51   8.0   85.0   0.51   40.0

9.0   5.0   2.0   100.0   28.0

После всех итераций

X1 = 7.932272406036654

X2 = -7.199668312070536

X3 = 1.101186180939743

X4 = -0.09594482455856695

Общее количество проделанных итераций = 4

вектор погрешности вектора X_1 = 2.791191390212333E-4

вектор погрешности вектора X_2 = 2.780277611869053E-4

вектор погрешности вектора X_3 = 2.2353637719518815E-5

вектор погрешности вектора X_4 = 1.1666407206956109E-5

6. Вывод

Метод – Зейделя , даёт нам большой выигрыш в точности, так как, во-первых, метод Зейделя существенно уменьшает число умножений и делений, во-вторых, позволяет накапливать сумму произведений без записи промежуточных результатов. Кроме этого метод Зейделя может оказаться более удобным при программировании, так как при вычислении $x_i^{(k+1)}$ нет необходимости хранить значения $x_1^{(k)}$, $x_2^{(k)}$, …, $x_{i-1}^{(k)}$
Метод Зейделя требует несколько меньшей памяти, чем простая итерация, так как необходимо помнить только один вектор переменных.