МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине

«ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА»

Вариант “Метод Гаусса-Зейделя”

Выполнил:
Студент группы P32312
Лысенко А.К.
Преподаватель:
Перл О.В.

Санкт-Петербург, 2023

Описание метода, расчётные формулы

Метод Гаусса-Зейделя предназначен для решения СЛАУ. Он является итерационным методом, что означает, что решение будет найдено с какой-то погрешностью. Этот метод подходит для решения СЛАУ с большим количеством неизвестных. Правда, данный метод способен решать далеко не все системы. Данный метод имеет условие сходимости, при невыполнении которого, метод не работает. Условие сходимости представляет собой наличие диагонально 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

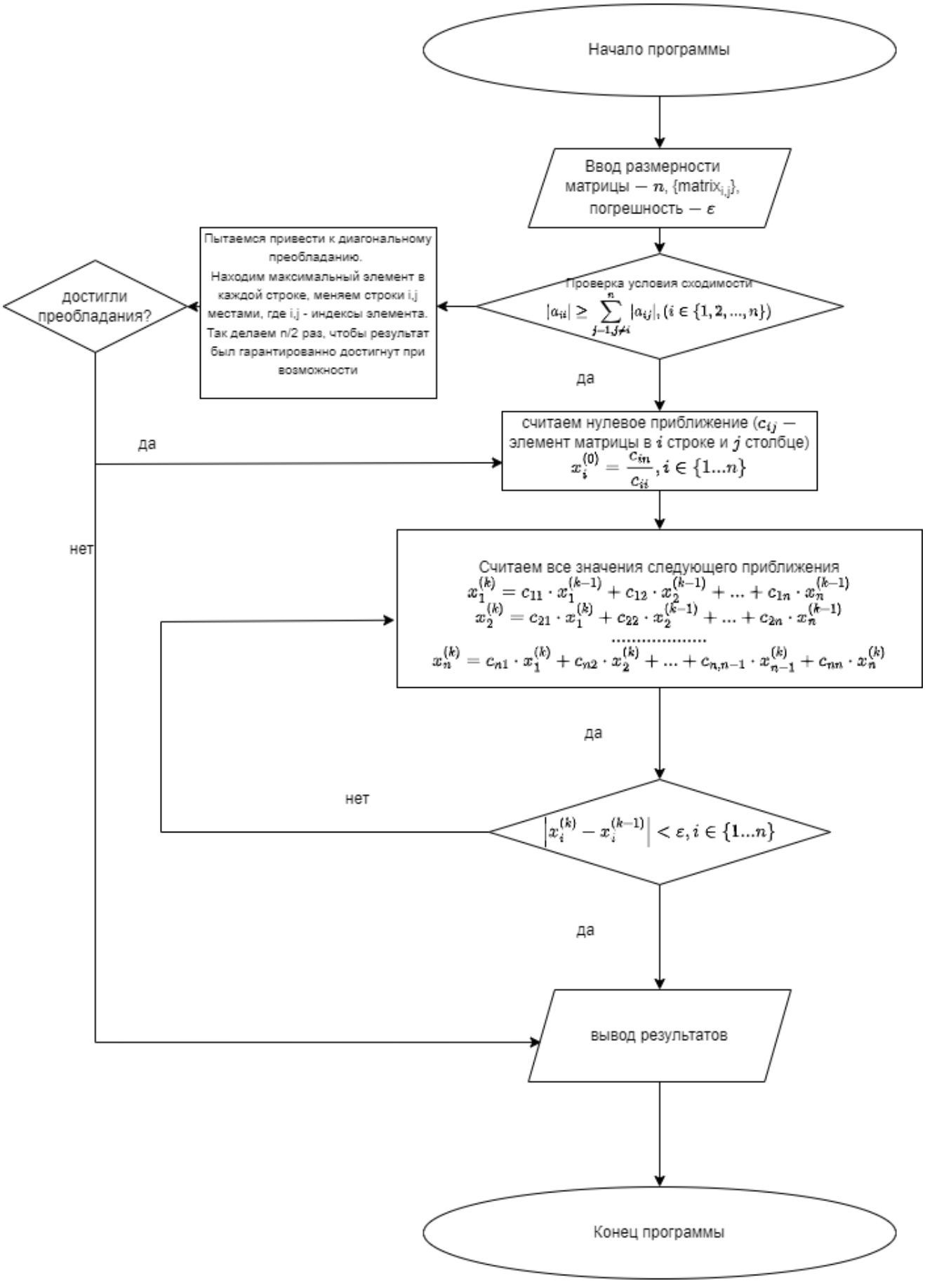
$$x_i^{(0)} = \frac{c_{in}}{c_{ii}}, i \in \{1...n\}$$

c_{in} – обозначает последний элемент каждой строки, то есть то число, которое стоит справа от равно. Делим мы его на диагональной элемент, при условии, что есть диагональное преобладание. Собственно, нулевое приближение мы сделали. Далее мы циклично повторяем расчёт приближений по формуле до тех пор, пока мы не добьемся нужной точности.

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

То есть на каждой итерации мы используем значения с прошлой.

Блок-схема


Начало программы
Ввод размерности матрицы — n, {matrix$_{i,j}$}, погрешность — ε
Проверка условия сходимости
$|a_{ii}| \geq \sum_{j=1, j \neq i}^{n} |a_{ij}|, (i \in \{1, 2, .., n\})$
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
нет
считаем нулевое приближение (c_{ij} — элемент матрицы в i строке и j столбце)
$x_i^{(0)} = \frac{c_{in}}{c_{ii}}, i \in \{1...n\}$
да
Считаем все значения следующего приближения
$x_1^{(k)} = c_{11} \cdot x_1^{(k-1)} + c_{12} \cdot x_2^{(k-1)} + ... + c_{1n} \cdot x_n^{(k-1)}$
$x_2^{(k)} = c_{21} \cdot x_1^{(k)} + c_{22} \cdot x_2^{(k-1)} + ... + c_{2n} \cdot x_n^{(k-1)}$
.....
$x_n^{(k)} = c_{n1} \cdot x_1^{(k)} + c_{n2} \cdot x_2^{(k)} + ... + c_{n,n-1} \cdot x_{n-1}^{(k)} + c_{nn} \cdot x_n^{(k)}$
да
нет
$\left|x_i^{(k)} - x_i^{(k-1)}\right| < \varepsilon, i \in \{1...n\}$
да
вывод результатов
Конец программы

Листинг численного метода

```
private boolean shouldStop() {
    int c = 0;
    for (int i = 0; i < matrix.getN(); i++) {
        stop = stop && (Math.abs(currentApproximation[i] - prevApproximation[i]))
>= epsilon;
        if ((Math.abs(currentApproximation[i] - prevApproximation[i])) >=
epsilon) {
            c += 1;
        }
    }
    if (c == 0) {
        return false;
    }
    return stop;

}

private void zeroApproximation() {
    for (int i = 0; i < matrix.getN(); i++) {
        currentApproximation[i] = matrix.getB()[i] / matrix.getArray()[i][i];
        zeroApproximation[i] = matrix.getB()[i] / matrix.getArray()[i][i];
        prevApproximation[i] = matrix.getB()[i] / matrix.getArray()[i][i];
    }
    counter++;
}

public void approximation() {
    zeroApproximation();
    while (stop) {
        double multiplier;
        for (int i = 0; i < matrix.getN(); i++) {
            for (int j = 0; j < matrix.getN(); j++) {
                if (i > j) {
                    multiplier = currentApproximation[j];
                } else {
                    multiplier = prevApproximation[j];
                }
                if (i != j) {
                    currentApproximation[i] += ((-1) * matrix.getArray()[i][j] *
multiplier) / matrix.getArray()[i][i];
                }
            }

        }
        stop = shouldStop();
        double[] deltas = new double[matrix.getN()];
        for (int i = 0; i < deltas.length; i++) {
            deltas[i] = Math.abs(currentApproximation[i] - prevApproximation[i])
/ Math.abs(currentApproximation[i]);
        }
        if (!stop) {
            drawNewLine(counter, currentApproximation, deltas);
        }
        if (stop) {
            System.arraycopy(currentApproximation, 0, prevApproximation, 0,
matrix.getN());
            System.arraycopy(zeroApproximation, 0, currentApproximation, 0,
matrix.getN());
            counter++;
        }
    }
}
```

Примеры и результаты работы

Input file

```
3

2 -1 5 10
5 -1 3 5
1 -4 2 20
```

Output

Input file

```
20
96,532 -8,822 5,046 -0,466 -4,378 -0,656 8,17 -9,122 7,558 0,491 -7,101 -1,858 4,285 6,065 4,443 2,589 -7,426 5,664 -3,834 -7,558 -736,63113
-2,358 101,002 -3,899 -2,048 -4,241 -6,544 4,306 -7,248 3,239 1,725 3,116 5,062 8,106 -0,025 6,727 -7,689 6,059 3,242 9,509 4,451 -1011,40689
-0,647 9,114 92,375 2,049 -4,573 4,965 -1,183 1,807 9,593 4,805 -3,668 -1,432 -8,824 -2,749 6,382 -1,056 -8,417 9,863 -4,112 -6,136 595,03945
-5,681 8,765 -0,217 93,962 6,178 -3,91 7,717 0,278 -3,485 -7,578 -3,344 -2,009 -7,975 8,289 -4,318 -1,627 5,903 -9,668 -1,509 -4,511 749,0368
3,647 6,92 -3,217 -3,507 89,19 -7,276 -9,655 3,367 4,476 3,379 3,622 0,587 9,92 -4,966 -9,651 -2,055 8,968 -0,752 -1,125 1,1 -11,09666
3,078 2,383 9,676 -9,423 9,152 121,852 0,7 -9,429 -4,32 9,531 7,397 8,374 -2,818 0,672 -8,155 -7 -3,412 9,248 -6,634 7,372 -747,19795
2,038 -0,957 3,034 7,405 -7,896 3,256 100,872 -9,572 6,935 1,084 9,616 4,389 -0,585 -9,432 -5,463 -2,893 -4,084 7,778 3,763 -0,356 727,99241
-1,203 3,849 6,224 2,212 2,738 4,361 8,504 99,374 2,381 0 -4,492 3,728 -5,381 -8,96 -6,66 -3,358 -6,828 7,047 5,706 -4,632 373,30936
-1,255 -1,553 7,781 -6,014 7,767 0,104 7,308 5,893 110,686 -8,53 -0,396 -2,272 -5,885 -6,758 5,194 6,81 -4,808 -0,634 8,894 -3,172 -1176,39383
-5,446 -8,916 4,107 -9,339 -5,204 9,528 -8,803 -9,535 -7,256 148,384 8,057 -5,724 2,652 -9,948 -4,598 8,443 -1,773 4,971 -5,743 -8,803 834,50263
1,71 -1,47 5,905 -3,999 -9,497 7,798 -1,854 -0,168 7,98 -2,319 88,685 8,54 -5,34 -5,809 -2,637 -6,326 -6,675 2,971 0,576 6,111 -158,56869
7,927 -9,261 1,547 -6,548 1,463 -6,81 -0,208 8,367 6,965 4,395 4,637 106,722 -3,568 2,845 2,554 8,203 4,66 -5,707 -8,27 3,051 594,30402
-5,236 3,05 -8,755 -4,264 -2,008 5,861 8,039 4,769 -5,098 -8 3,681 -4,914 97,681 3,783 -2,209 -5,642 -7,451 -0,96 4,608 8,353 -36,75088
-8,985 3,286 -7,825 2,908 8,228 1,346 4,809 -4,529 -1,335 -1,322 -1,154 -6,941 -5,206 80,152 -2,31 0,301 7,715 -6,529 -2,67 1,753 95,36961
2,063 2,395 -9,044 -9,454 2,808 9,062 -5,238 3,845 2,111 5,253 -2,878 4,698 9,169 -6,251 99,66 9,071 2,828 -0,236 -5,486 -6,77 -883,73857
6,237 3,322 -5,165 -7,973 -1,17 -9,38 -9,898 4,057 -5,166 -4,431 -2,721 -3,4 -2,133 -1,048 -9,108 103,046 2,363 -4,58 2,285 0,047 -236,03629
6,554 3,425 -8,275 -1,639 -4,375 -0,586 -2,074 1,231 -5,113 3,707 -7,185 -1,477 4,3 5,826 9,073 6,121 91,764 -6,022 8,442 -5,339 -151,28169
-2,723 -0,316 9,843 -1,902 1,302 4,798 8,872 7,852 -9,534 -5,656 2,229 3,184 7,672 9,683 9,896 -6,89 3,666 113,989 1,206 2,175 1293,62312
9,947 1,862 -5,537 8,077 3,567 3,125 -8,181 -4,902 3,844 -8,558 3,843 -4,614 1,839 -0,724 -6,259 -8,118 -9,737 4,323 99,002 -0,945 -932,39781
-5,579 -0,637 0,586 5,339 6,628 6,143 9,177 3,011 -5,147 -9,226 -2,764 2,485 6,962 -2,67 7,822 5,38 0,021 7,735 -8,328 96,64 554,26341
```

Output

```
i               X1              X2              X3              X4              X5              X6              X7              X8              X9              X10
 X11             X12             X13             X14             X15             X16             X17             X18             X19             X20             delta X1
 delta X2        delta X3        delta X4        delta X5        delta X6        delta X7        delta X8        delta X9        delta X10       delta X11       delta X12
   delta X13       delta X14       delta X15       delta X16       delta X17       delta X18       delta X19       delta X20
7       -8,75299922     -9,92899992     8,04700061      8,12599972      0,94399898      -8,47000031     6,91099968      2,79499999      -9,85699961     5,48299969
 -0,94800005     5,42800002      0,6340002       1,46500023      -6,40199999     -0,70199994     1,62600019      9,788           -7,17400004     3,3790001       0,00000036
     0,00000003      0,00000114      0,0000003       0,00000955      0,00000031      0,00000032      0,00000047      0,00000004      0       0,00000077      0,00000026
         0,00000288      0,00000041      0,00000046      0,00000081      0,00000009      0,00000016      0,00000027      0,00000008
```

Выходные данные совпали с ожидаемыми в тесте.

Input

```
Введите матрицу размера 3x4
1 40 1 5
40 1 1 7
1 1 40 8
```

Output

```
    i                X1                X2                X3          delta X1          delta X2          delta X3
    2       0,16728099        0,11599466        0,19291811        0,00242697        0,00143575        0,00007419
```

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.

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

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.