Санкт-Петербургский Национальный Исследовательский Университет Информационных Технологий, Механики и Оптики

Факультет Программной Инженерии и Компьютерной Техники

Вариант № 17

# «Метод простых итераций»

Лабораторная работа № 1

по дисциплине

’вычислительная математика’

Выполнил:

Ортис Хосе - 288867;

P3232;

Преподаватель:

Малышева Татьяна Алексеевна

Санкт-Петербург 2020г.

**Текст задания:**

Размерность 1...20 (задается из файла или с клавиатуры — по выбору конечного пользователя) Должно быть предусмотрено чтение исходных данных как из файла, так и ввод с клавиатуры. Должна быть реализована возможность ввода коэффициентов матрицы как с клавиатуры, так и из файла. Также предусмотреть случайные коэффициенты.

Для итерационных методов:

- Точность задается с клавиатуры / файла
- Проверка диагонального преобладания (В случае, если диагональное преобладание в изначальной матрице отсутствует — предлагается сделать перестановку строк / столбцов до тех пор, пока преобладание не будет достигнуто. В случае невозможности достижения диагонального преобладания — выводить сообщение.)
- Вектор неизвестных
- Количество итераций, за которое было найдено решение
- Вектор погрешностей

**Code can be checked here:**
https://github.com/joseortiz9/computational-math-itmo-2021/tree/master/lab1-jacobi-iterations-method

**Описание метода, расчетные формулы:**

Пусть дана системы линейных алгебраических уравнений вида: A · X = B, где A – матрицы коэффициентов, X – столбец неизвестных и B – столбец свободных членов.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}; \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}; \quad B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

который аналогично может быть представлен системой уравнений:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \qquad \vdots \qquad\qquad \vdots \quad &\quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

Для каждой строки xi можно выделить переменную x_i:

$$\begin{aligned} x_1 &= \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n) \\ x_2 &= \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n) \\ &\vdots \\ x_n &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1}) \end{aligned}$$

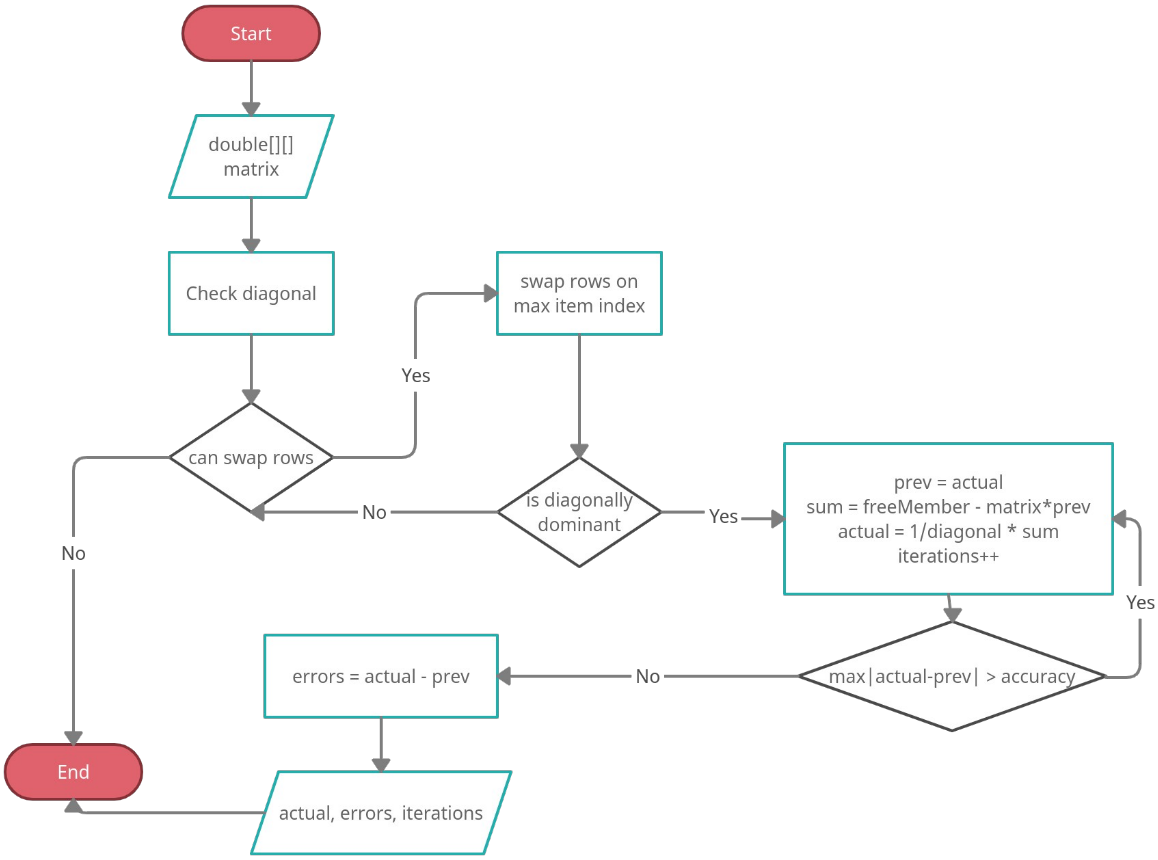
Теперь, чтобы начать итерацию, определяем ноль как начальное значение для переменных (x1, x2, x3, ..., xn). Позже подставим полученные значения xi в переписанное уравнение и повторяем процесс до тех пор, пока следующее условие не выполняется:

*maxElement{abs( actual - prev )} < accurancy*

prev - это последняя вычисленная итерация
actual – очевидно...
accuracy - точность вычисления.

Метод итерации применяют в случае, если сходится последовательность приближений по указанному алгоритму или по другим словам если матрица коэффициентов уравнения обладает свойством *диагонального преобладания*.

### Блок-схема численного метода:




### Листинг реализованного численного метода программы

```java
Jacobi jacobi = new Jacobi(matrix, size);
context.print("Checking if matrix is diagonally dominant...\n");
if (!jacobi.makeDominant()) {
    context.print("Impossible to determinate diagonally dominant: iterations method can diverge.\n");
} else {
    context.print(jacobi.printSystem());
    solver.solveWithJacoby(jacobi, context.getAccuracy());
    context.print(jacobi.toString());
}
```

```java
public void solveWithJacoby(Jacobi jacobi, double accuracy) {
    int iterations = 0;
    int n = jacobi.getSize();
    double[] actual = new double[n]; // Approximations
    double[] prev = new double[n]; // Previous
    Arrays.fill(actual, v: 0);
    Arrays.fill(prev, v: 0);
    do {
        prev = actual.clone();

        for (int i = 0; i < n; i++) {
            double sum = jacobi.getFreeMember(i);

            for (int j = 0; j < n; j++)
                if (j != i)
                    sum -= jacobi.get(i,j) * prev[j];

            actual[i] = 1/jacobi.get(i,i) * sum; // (diagValue)^-1 * (freeMember - [row[i] * prevXValues[i]])
        }
        iterations++;
    } while (getMaxAbsValue(subtractFromVector(actual, prev)) > accuracy && iterations != MAX_ITERATIONS);

    jacobi.setIterations(iterations);
    jacobi.setRootsSolution(actual);
    jacobi.setErrorMargins(subtractFromVector(actual, prev));
}
```

**Примеры и результаты работы программы на разных данных:**

```
+-------------------------------------------+
|        JACOBI ITERATIONS METHOD           |
+-------------------------------------------+
Default accuracy=1.0E-15D
cmath-lab1$ solve_matrix
Enter the size of the matrix: 3
20 1 -2 17
3 20 -1 -18
2 -3 20 25
Checking if matrix is diagonally dominant...
System of Equations:
20.0 * x_1 + 1.0 * x_2 + -2.0 * x_3 = 17.0
3.0 * x_1 + 20.0 * x_2 + -1.0 * x_3 = -18.0
2.0 * x_1 + -3.0 * x_2 + 20.0 * x_3 = 25.0
Jacobi solution:
+----------+----------------------+---------------------+
| variable | value                | Error Margin        |
+----------+----------------------+---------------------+
|   x_01   | 1.0000000000000000   | -0.0000000000000004 |
|   x_02   | -1.0000000000000000  | -0.0000000000000001 |
|   x_03   | 1.0000000000000000   |  0.0000000000000001 |
+----------+----------------------+---------------------+
A total of 19 iterations made.
cmath-lab1$
```

```
cmath-lab1$ set_accuracy
Enter the new accuracy value: 0.0001
New accuracy=1.0E-4D
cmath-lab1$ solve_matrix -s 3
20 1 -2 17
3 20 -1 -18
2 -3 20 25
Checking if matrix is diagonally dominant...
System of Equations:
20.0 * x_1 + 1.0 * x_2 + -2.0 * x_3 = 17.0
3.0 * x_1 + 20.0 * x_2 + -1.0 * x_3 = -18.0
2.0 * x_1 + -3.0 * x_2 + 20.0 * x_3 = 25.0
Jacobi solution:
+----------+----------------------+---------------------+
| variable | value                | Error Margin        |
+----------+----------------------+---------------------+
|   x_01   | 0.9999995000000000   | 0.0000332500000001  |
|   x_02   | -0.9999971250000000  | 0.0000803750000001  |
|   x_03   | 0.9999917500000001   | 0.0000354999999999  |
+----------+----------------------+---------------------+
A total of 6 iterations made.
cmath-lab1$ exit
Thanks for trying me :)

Process finished with exit code 0
```

in.txt

```
1 2 3 1.753
3 4 5 5
6 7 8 -25
```

```
Default accuracy=1.0E-15D
cmath-lab1$ solve_matrix -f in.txt
Enter the size of the matrix: 3
Reading matrix from file in.txt...
Checking if matrix is diagonally dominant...
Impossible to determinate diagonally dominant: iterations method can diverge.
```

in.txt

```
2 6 1.753
6 1 4.1
```

```
cmath-lab1$ solve_matrix -f in.txt
Enter the size of the matrix: 2
Reading matrix from file in.txt...
Checking if matrix is diagonally dominant...
System of Equations:
6.0 * x_1 + 1.0 * x_2 = 4.1
2.0 * x_1 + 6.0 * x_2 = 1.753
Jacobi solution:
+-----------+-----------------------+-----------------------+
| variable  | value                 | Error Margin          |
+-----------+-----------------------+-----------------------+
|   x_01    | 0.6719705882352940    |  0.0000000000000006   |
|   x_02    | 0.0681764705882355    |  0.0000000000000003   |
+-----------+-----------------------+-----------------------+
A total of 25 iterations made.
```

```
cmath-lab1$ solve_matrix -s 4
1 5 3 10 7.4321
4 3 10 0 1.2
0 10 1 8 5.1
10 1 2 3 4
Checking if matrix is diagonally dominant...
System of Equations:
10.0 * x_1 + 1.0 * x_2 + 2.0 * x_3 + 3.0 * x_4 = 4.0
0.0 * x_1 + 10.0 * x_2 + 1.0 * x_3 + 8.0 * x_4 = 5.1
4.0 * x_1 + 3.0 * x_2 + 10.0 * x_3 + 0.0 * x_4 = 1.2
1.0 * x_1 + 5.0 * x_2 + 3.0 * x_3 + 10.0 * x_4 = 7.4321
Jacobi solution:
+----------+----------------------+----------------------+
| variable | value                | Error Margin         |
+----------+----------------------+----------------------+
|   x_01   | 0.1680314092827001   | -0.0000000000000005  |
|   x_02   | -0.0992760843881861  | -0.0000000000000009  |
|   x_03   | 0.0825702616033753   | -0.0000000000000006  |
|   x_04   | 0.7512738227848098   | -0.0000000000000008  |
+----------+----------------------+----------------------+
A total of 170 iterations made.
```

```
Default accuracy=1.0E-15D
cmath-lab1$ solve_matrix
Enter the size of the matrix: 3
0 0 0 1
1 2 3 4
1 4 2 5
Checking if matrix is diagonally dominant...
Impossible to determinate diagonally dominant: iterations method can diverge.
cmath-lab1$ solve_matrix -s 3
0 0 0 1.421
0 0 0 3.1
0 0 0 1
Checking if matrix is diagonally dominant...
Impossible to determinate diagonally dominant: iterations method can diverge.
```

**Выводы**:

Этот метод простой и численно надежен. Каждая итерация достаточно быстра, поэтому он эффективен с большими матрицами. Мне не нравится то, что зависит от предыдущих значений для определения фактических, может быть, лучше использовать Гаусса-Зайделя. Этот метод сильно зависит от диагонального преобладания, мне это тоже не нравится, потому что ограничивает количество разрешимой матрицы и может занять больше времени, чтобы переставить строки для вычисления диагонали.