Метод простой итерации (Метод Якоби)

Описание метода, расчетные формулы

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

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

Матрицу A можем написать в виде $A = L + D + U$, где D – *диагональная матрица*, L и U – *нижная* и *верхная треугольные матрицы*, соответственно.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix} + \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

Следовательно, из равенства $A \cdot X = B$, заменив A на $L + D + U$ можем получить следующее равенство: $D \cdot X = B - (L + U) \cdot X$.

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} - \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

В результате деления каждого строка на диагональную элемент матрицы D, что не равны 0, получаем итерационную функцию: $X = B' - A' \cdot X$.

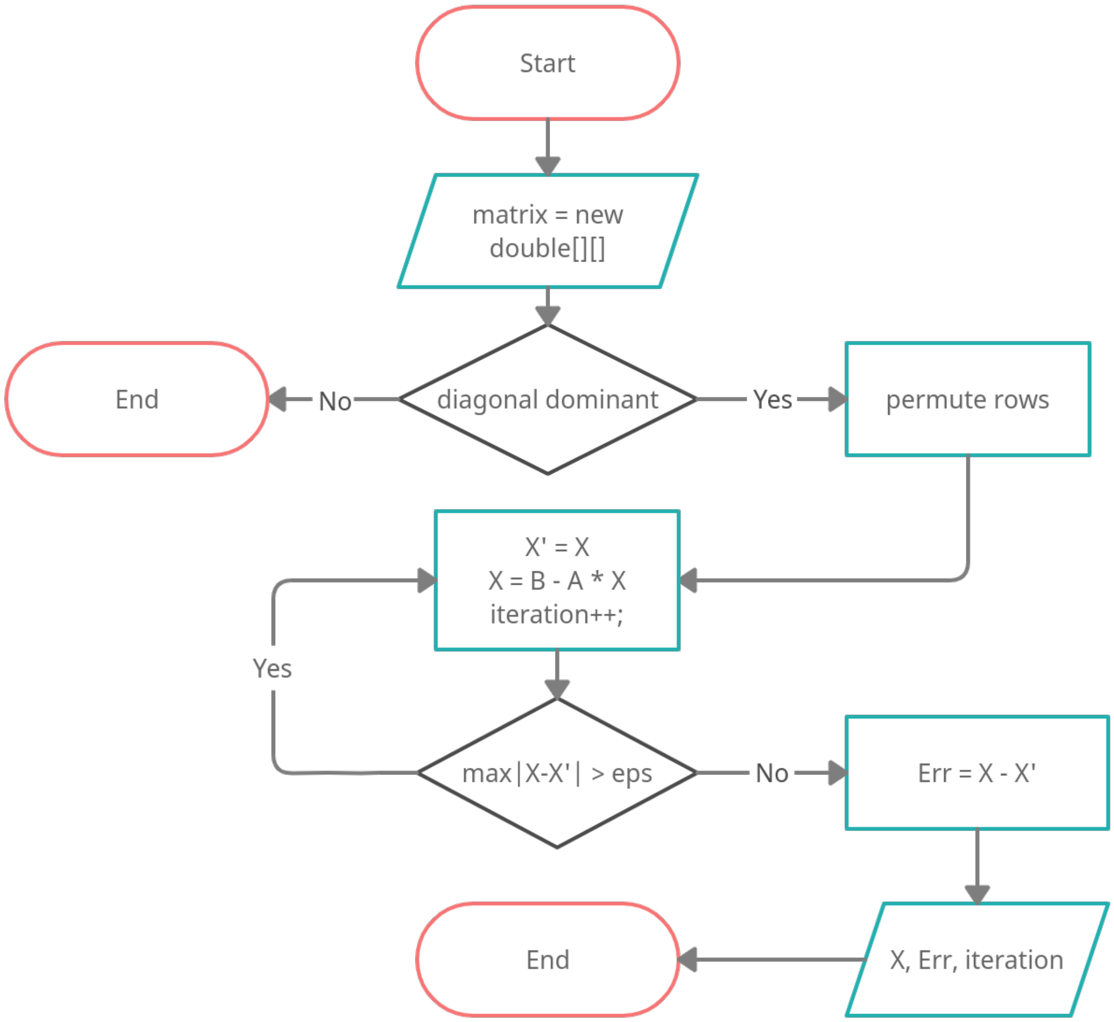
В качестве начального приближения примем столбец B', т.е. $X_i^{(0)} = B'^{(0)}_i$.

В качестве критерия остановки выберем условие – $\max\left\{\left|X_i^{(k+1)} = B' - A' \cdot X_i^{(k)}\right|\right\} < \epsilon$, где ϵ – точность вычисления.

Точность оценивается как $\epsilon = \left|\left|X - X^{(k)}\right|\right|$.

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
public int[] getPermutedRowsIfPossible() {
    int[] rowsIndices = new int[size];
    boolean[] flag = new boolean[size];

    for (int i = 0; i < size; ++i) {
        double absoluteSum = 0D;
        int maxItemIndex = 0;
        for (int j = 0; j < size; ++j) {
            if (Math.abs(extendedMatrix.get(i, maxItemIndex)) <
            ↪   Math.abs(extendedMatrix.get(i, j))) {
                maxItemIndex = j;
            }
            absoluteSum += Math.abs(extendedMatrix.get(i, j));
        }
        if (2 * Math.abs(extendedMatrix.get(i, maxItemIndex)) > absoluteSum &&
        ↪   !flag[maxItemIndex]) {
            flag[maxItemIndex] = true;
            rowsIndices[maxItemIndex] = i;
        } else {
            return null;
        }
    }
    return rowsIndices;
}
```

```java
public JacobiAnswer solveByJacobi(LinearSystem system, double accuracy) {
    int[] rowIndices = system.getPermutedRowsIfPossible();

    if (rowIndices == null) {
        throw new RuntimeException("Impossible to achieve diagonally dominant matrix with
        ↪   row permutations. Iterations diverge.");
    } else {
        system.doRowPermutation(rowIndices);
    }
    Matrix coefficients = system.getMatrixCoefficients();
    Matrix freeMembers = system.getMatrixFreeMembers();

    return iterate(coefficients, freeMembers, accuracy);
}
```

```java
private JacobiAnswer iterate(Matrix coefficients, Matrix freeMembers, double accuracy) {
    double[] diagonal = coefficients.getDiagonalArray();

    Matrix a = coefficients.div(diagonal).minus(Matrix.identity(diagonal.length));
    Matrix b = freeMembers.div(diagonal);

    Matrix prev, next = b.copy();
    int iters = 0;
    do {
       prev = next.copy();
       next = b.minus(a.mult(prev)); // X = B - A * X
       iters++;
    } while (next.minus(prev).getAbsMax() > Math.abs(accuracy));

    Matrix errors = next.minus(prev);

    return new JacobiAnswer(next, errors, iters);
}
```

Примеры и результаты работы программы на разных данных

```
// Example 1.
>>> accuracy = 0.0001
>>> matrix
2
4.54 0.5 3.001
-0.0987 0.12 -0.68
matrix: successfully read
>>> solve
+----+----------------------+----------------------+
| №i |  root (x)            |  infelicity (delta)  |
+----+----------------------+----------------------+
| 01 |  1.1783628207520878  |  0.0000420186456962  |
| 02 | -4.6974978069341600  |  0.0000366054947971  |
+----+----------------------+----------------------+
> Iterations: 9 times.

// Example 2.
>>> matrix
3
0 0 0 1.097
0 0 0 4.5
0 0 0 -0.123
matrix: successfully read
>>> solve
Impossible to achieve diagonally dominant matrix with row permutations. Iterations
↪  diverge.

// Example 3.
>>> accuracy = 1E-9
>>> show -a
accuracy=0.000000001000
>>> matrix -f 'matrix.java'
matrix: successfully read
>>> show -m
[0.23, -0.875, 3.34] * x_1 = -1.43
[-3.4, 3.001, 0.012] * x_2 = 5.43
[2.4, -3.42, -0.124] * x_3 = 0.324
>>> solve
+----+----------------------+----------------------+
| №i |  root (x)            |  infelicity (delta)  |
+----+----------------------+----------------------+
| 01 | -4.3452877538398430  | -0.0000000009622569  |
| 02 | -3.1098483461461592  | -0.0000000002672542  |
| 03 | -0.9436230896653286  | -0.0000000002593603  |
+----+----------------------+----------------------+
> Iterations: 86 times.
```

Вывод

Метод простих итераций более эффективен при решение СЛАУ большой размерности. Они (итерационные методы) не требует хранения всей матрицы в ОЗУ, в отличие от прямых методов, которые более эффетивны при решении небольших СЛАУ, так как позволяют найти решение за конечное число операций. Болбшую роль в скорости выполнения метода Якоби играют диагональные элементы матрицы A и начальные приблежения – чем они ближе к настоящим значениям x, тем быстрее вестор будет найден.