МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

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 «Решение СЛАУ методом простых итераций»

Студент:
Лунева Арина Алексеевна

Группа P3231

Преподаватель:
Перл Ольга Вячеславовна

Санкт-Петербург, 2022

1. Описание реализованного метода, расчетные формулы

Для вычисления неизвестных используется следующая формула:

$$x_i^{[j+1]} = x_i^{[j]} - \frac{1}{a_{ii}} * \left(\sum_{k=1}^{n} a_{ik} * x_k^{[j]} - b_i\right),$$

где A –матрица коэффициентов исходного СЛАУ, B – массив свободных членов уравнений.

Для выполнения первой итерации значений $x_i^{[j]}$ итерационного процесса используются различные варианты:

- Заполнение массива начальных значений нулями
- Подстановка массива свободных членов уравнений B
- Подстановка некоторых предварительно рассчитанных значений для повышения точности результата
- Любые другие значения

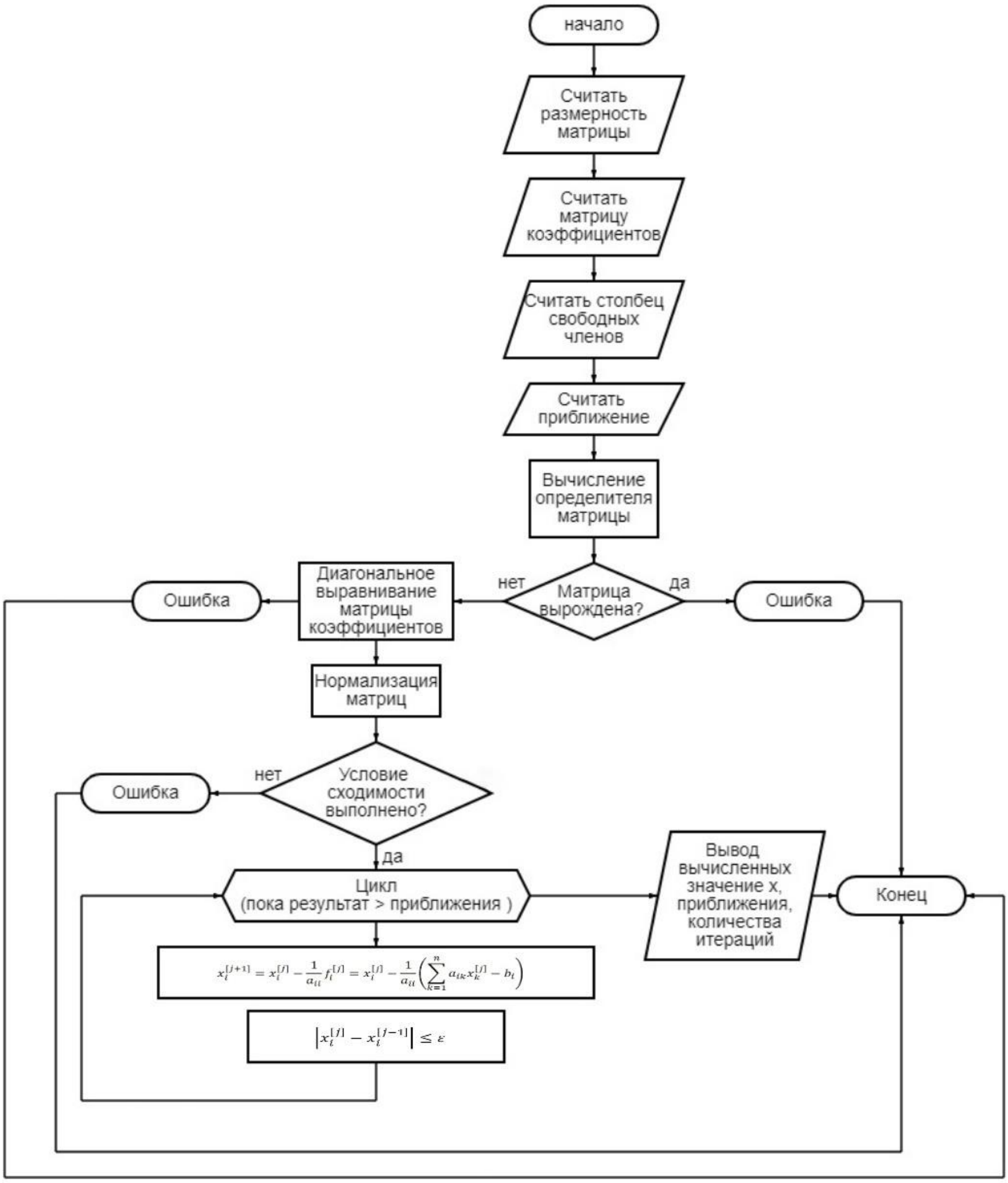
Условия применения метода:

- Матрица коэффициентов СЛАУ является невырожденной
- Для матрицы коэффициентов СЛАУ должно выполняться условие преобладания диагональных элементов
- Должно выполняться условие сходимости итерационного процесса:
  $|a_{ii}| = \sum_{i \neq k} |a_{ik}| \quad (i, k = 1, 2, \dots, n)$

Итерации прекращаются при выполнении следующего условия:

$\forall \varepsilon > 0: \max\limits_{1 \le i \le n}: \left|x_j^{[i]} - x_j^{[i-1]}\right| \le \varepsilon$, где ε – точность ответа.

2. Блок-схема численного метода



3. Листинг реализованного метода

```
public class Determinate {
    public double calcDet(double[][] matrix, int dim) {
        double[][] newMatrix = new double[matrix.length][matrix.length];
        for (int i = 0; i < dim; i++) {
            System.arraycopy(matrix[i], 0, newMatrix[i], 0, matrix.length);
        }
        int multiplier = 1;
        for (int i = 0; i < dim - 1; i++) {
            int j = i;
            while (newMatrix[j][i] == 0 && j < dim - 1) {
                j++;
            }
            if (j == dim) {
                return 0;
            } else if (j != i) {
                for (int k = i; k < dim; k++) {
                    double temp = newMatrix[i][k];
                    newMatrix[i][k] = newMatrix[j][k];
                    newMatrix[j][k] = temp;
                    multiplier = multiplier * (-1);
                }
            }
            for (j = i + 1; j < dim; j++) {
                if (newMatrix[j][i] != 0) {
                    double coefficient = newMatrix[j][i] / newMatrix[i][i];
                    for (int k = i; k < dim; k++) {
                        newMatrix[j][k] -= newMatrix[i][k] * coefficient;
                    }
                }
            }
        }
        double result = 1;
        for (int i = 0; i < dim; i++) {
            result *= newMatrix[i][i];
        }
        return multiplier * result;
    }
}
```

```
public boolean checkDiagonallyDominant(int dim, double[][] matrix, double[]
freeTerms) {
    for (int i = 0; i < dim; i++) {
        double rowSum = 0;
        double maxInRow = 0;
        int indexOfMax = -1;
        for (int j = 0; j < dim; j++) {
            rowSum += Math.abs(matrix[i][j]);
            if (Math.abs(matrix[i][j]) > maxInRow) {
                maxInRow = Math.abs(matrix[i][j]);
                indexOfMax = j;
            }
        }
        if (maxInRow >= rowSum - maxInRow) {
            double[] tempMatrix = matrix[i];
            matrix[i] = matrix[indexOfMax];
            matrix[indexOfMax] = tempMatrix;
```

```java
            double tempFreeTerm = freeTerms[i];
            freeTerms[i] = freeTerms[indexOfMax];
            freeTerms[indexOfMax] = tempFreeTerm;
        } else {
            return false;
        }
    }
    for (int i = 0; i < dim; i++) {
        double maxInRow = 0;
        int indexOfMax = -1;
        for (int j = 0; j < dim; j++) {
            if (matrix[i][j] > maxInRow) {
                maxInRow = matrix[i][j];
                indexOfMax = j;
            }
        }
        if (indexOfMax != i) {
            return false;
        }
    }
    return true;
}
```

```java
public void normalize(int dim, double[][] matrix, double[] freeTerms) {
    for (int i = 0; i < dim; i++) {
        double coefficient = matrix[i][i];
        for (int j = 0; j < dim; j++) {
            matrix[i][j] = (-1) * matrix[i][j] / coefficient;
        }
        freeTerms[i] = freeTerms[i] / coefficient;
        matrix[i][i] = 0;
    }
}
```

```java
public boolean checkConvergenceCondition(int dim, double[][] matrix) {
    double matrixNorm = 0;
    for (int i = 0; i < dim; i++) {
        double rowSum = 0;
        for (int j = 0; j < dim; j++) {
            rowSum += Math.abs(matrix[i][j]);
        }
        if (rowSum > matrixNorm) {
            matrixNorm = rowSum;
        }
    }
    return (matrixNorm < 1);
}
```

```java
public double[] calcRoots(int dim, double[][] matrix, double[] freeTerms,
double epsilon, double[] roots) {
    int countIteration = 0;
    while (true) {
        double[] previousRoots = roots.clone();
        for (int i = 0; i < dim; i++) {
            roots[i] = 0;
            for (int j = 0; j < dim; j++) {
                if (i != j) {
                    roots[i] += matrix[i][j] * previousRoots[j];
```

```
                }
            }
            roots[i] += freeTerms[i];
        }
        double resultEpsilon = -1;
        for (int i = 0; i < dim; i++) {
            if (Math.abs(roots[i] - previousRoots[i]) > resultEpsilon) {
                resultEpsilon = Math.abs(roots[i] - previousRoots[i]);
            }
        }
        if (resultEpsilon <= epsilon) {
            countIteration++;
            this.countIterations = countIteration;
            this.resultEpsilon = resultEpsilon;
            return roots;
        } else {
            countIteration++;
        }
    }
}
```

4. Примеры результата работы программы

Выберите способ ввода данных в программу. Введите номер желаемого метода
1. Пользовательский ввод
2. Ввод данных из файла
3. Генерация случайных матрицы
2
Введите имя файла
test1
Введите приближение
0.01

Введенная матрица коэффициентов:
10.0 1.0 1.0
2.0 2.0 10.0
2.0 10.0 1.0

Введенный массив свободных членов
12.0 14.0 13.0

Определитель матрицы: -946.0000000000001

Приведем матрицу к виду с преобладанием диагональных элементов
10.0 1.0 1.0
2.0 10.0 1.0
2.0 2.0 10.0

Нормализуем матрицу
0.0 -0.1 -0.1
-0.2 0.0 -0.1
-0.2 -0.2 0.0

Проверим условие сходимости
Условие сходимости выполнено

Ответ:
x1 = 0.999568
x2 = 0.99946
x3 = 0.9993159999999999

Погрешности:
x1: 0.0019320000000000448
x2: 0.0024600000000001288
x3: 0.0030840000000000867

Количество итераций: 5

Выберите способ ввода данных в программу. Введите номер желаемого метода
1. Пользоватльеский ввод
2. Ввод данных из файла
3. Генерация случайных матрицы
1
Введите размер матрицы из отрезка [3,20]
3
Введите построчно матрицу коэффициентов
1 2 3

4 5 6
7 8
Количество коэффициентов не совпадает с размером матрицы

Введите построчно матрицу коэффициентов
1 2 3
4 5 6
7 8 9
Введите столбец свободных членов
14
12
13
Введите приближение
0.05

Введенная матрица коэффициентов:
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0

Введенный массив свободных членов
14.0 12.0 13.0

Определитель матрицы: 0.0
Определитель матрицы равен 0. Невозможно применить метод простых итераций
Выберите способ ввода данных в программу. Введите номер желаемого метода
1. Пользовательский ввод
2. Ввод данных из файла
3. Генерация случайных матрицы
1
Введите размер матрицы из отрезка [3,20]
3
Введите построчно матрицу коэффициентов
1 2 1
2 3 -1
1 1 2
Введите столбец свободных членов
1
2
3
Введите приближение
0.2

Введенная матрица коэффициентов:
1.0 2.0 1.0
2.0 3.0 -1.0
1.0 1.0 2.0

Введенный массив свободных членов
1.0 2.0 3.0

Определитель матрицы: -4.0

Невозможно привести матрицу к виду с преобладанием диагональных элементов

5. Вывод

Метод простых итераций позволяет вычислять значения с заданной точностью и позволяет минимизировать погрешности. К недостаткам данного метода можно отнести медленную скорость сходимости и сложность выполнения всех условий применения данного метода (особенно для больших матриц).

Сравнивая метод простых итераций с методом Гаусса-Зейделя, можно сделать следующие выводы:

1. Метод Гаусса-Зейделя обеспечивает более быструю сходимость к решению систему, чем метод простых итераций
2. Для метода Гаусса-Зейделя сложнее найти систему, для которой будет выполняться условие сходимости.

Сравнивая прямые и итерационные методы, можно сделать следующие выводы:

1. Алгоритмы итерационных методов более сложные, чем алгоритмы прямых методов
2. Прямые методы используют конечные соотношения и позволяют найти решение за конечное известное число арифметических операций, при использовании итерационных методов неизвестно, как много итераций потребуется
3. Прямые методы требует хранения в памяти полной матрицы, итерационные лишь несколько векторов
4. При итерационных методах погрешности не накапливаются, при прямых методах происходит накапливание погрешностей, так как вычисления используют результаты предыдущих операций.