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

ФКТиУ, кафедра Вычислительной техники

Лабораторная работа №1

по дисциплине
«Вычислительная математика»
Вариант: Метод Гаусса

Выполнил: Студент группы P3233
Сабитов Д.Т.

Преподаватель:
Перл О. В.

Санкт-Петербург
2022 г.

Описание метода

Метод Гаусса логически состоит из двух стадий – прямого и обратного хода. Во время прямого хода матрица системы приводится к треугольному виду путем элементарных операций над уравнениями. Для этого мы обнуляем ненулевые элементы, лежащие под главной диагональю путем вычитания из строки вышележащей строки, умноженной на найденный коэффициент. Если на какой-то из итераций среди элементов столбца не нашелся ненулевой элемент, то мы переходим к следующему столбцу. Во втором этапе (обратный ход) мы выражаем решение системы в численном виде. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую переменную и подставляют в предыдущие уравнения.

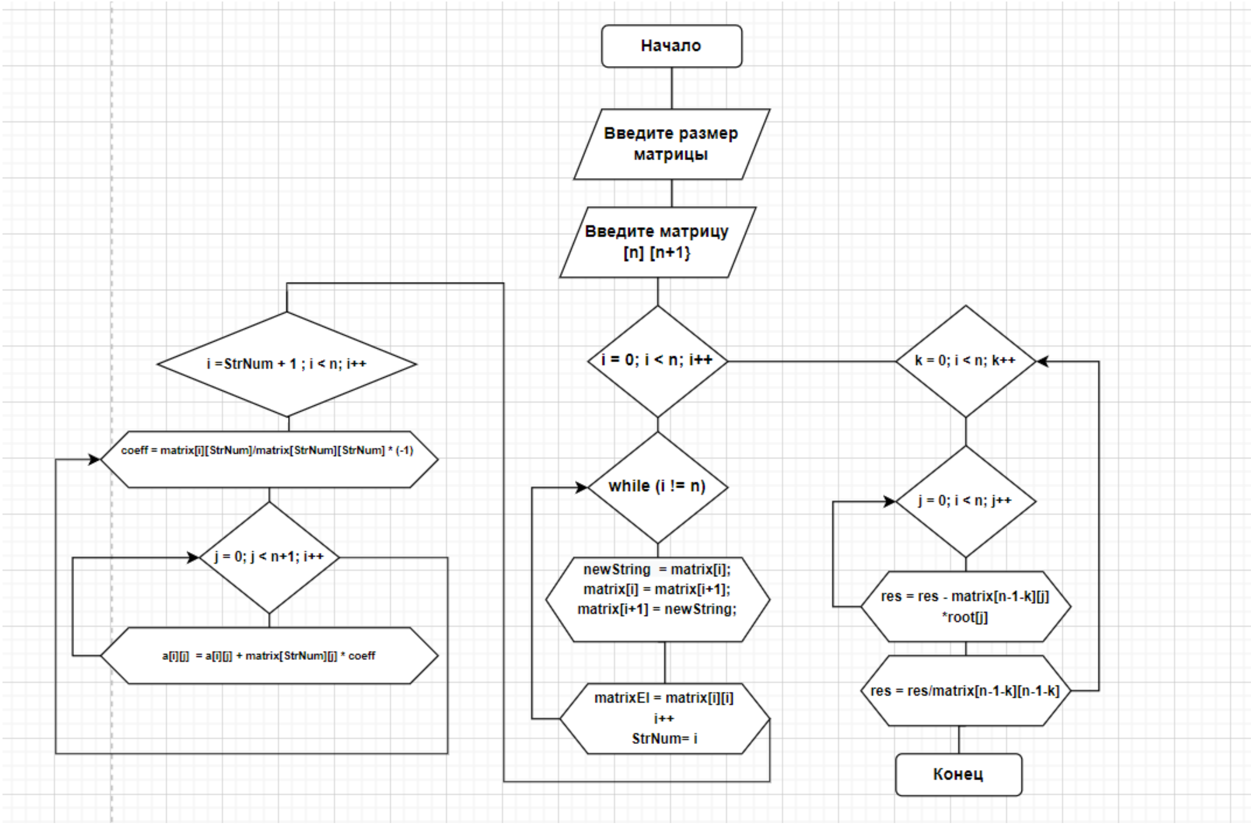
Расчетные формулы метода

Преобразованная система:

$$\begin{cases} \alpha_{1j_1}x_{j_1} + \alpha_{1j_2}x_{j_2} + \ldots + \alpha_{1j_r}x_{j_r} + \ldots + \alpha_{1j_n}x_{j_n} & = & \beta_1 \\ \alpha_{2j_2}x_{j_2} + \ldots + \alpha_{2j_r}x_{j_r} + \ldots + \alpha_{2j_n}x_{j_n} & = & \beta_2 \\ & \ldots & \\ \alpha_{rj_r}x_{j_r} + \ldots + \alpha_{rj_n}x_{j_n} & = & \beta_r \\ 0 & = & \beta_{r+1} \\ & \ldots & \\ 0 & = & \beta_m \end{cases},$$

где $\alpha_{1j_1}, \ldots, \alpha_{rj_r} \neq 0$.

Блок-схема метода



Листинг

Метод для приведения матрицы к треугольной:

```
public double[][] calculateGauss(double[][] matrix, int size) {
    for (int i = 0; i < size; i++) {
        matrix = swapColumns(matrix, size, i);
        matrix = addingZeroes(matrix, size, i);
    }
    determinant = calculateDeterminant(matrix, size) * this.substitution;

    return matrix;
}
```

Метод для замены строк матрицы:

```
private double[][] swapColumns(double[][] matrix, int size, int stringNum){
    double[][] resultMatrix = matrix;
    double[] newString;
    double matrixElement = matrix[stringNum][stringNum];
    if (matrixElement == 0){
        while (stringNum != size - 1 && matrixElement == 0){
            newString = matrix[stringNum];
            matrix[stringNum] = matrix[stringNum + 1];
            matrix[stringNum + 1] =newString;
            matrixElement =resultMatrix[stringNum][stringNum];
            stringNum = stringNum + 1;
            this.substitution =this.substitution * (-1);
        }
    }
    return resultMatrix;
}
```

Метод для обнуления чисел матрицы в столбце под главной диагональю:

```
private double[][] addingZeroes(double[][] matrix, int size, int stringNum){
    double[][] resultMatrix = matrix;
    double coeff;
    for (int i = stringNum + 1; i < size; i++){
        if (matrix[stringNum][stringNum] != 0){
            coeff = matrix[i][stringNum]/matrix[stringNum][stringNum] * (-1);
            for (int j = 0; j < size + 1; j++){
                resultMatrix[i][j] = resultMatrix[i][j] +
matrix[stringNum][j] * coeff;
            }
        }
        else {
            break;
        }
    }
    return resultMatrix;
}
```

Метод для вычисления определителя:

```
private double calculateDeterminant(double[][] matrix, int size){
    double determinant = 1;
    for (int i = 0; i < size; i++){
        determinant = determinant * matrix[i][i];
    }
    return determinant;
}
```

Методы для вычисления столбца неизвестных:

```
public double[] getRoots(double[][] matrix, int size){
    double[] root = new double[size];
    for (int i = 0; i < size; i++) {
        root[size - 1 - i] = utilRoots(matrix, size, root, size-1-i);
    }
    return root;
}

private double utilRoots(double[][] matrix, int size, double[] root, int
num){
    double result = matrix[num][size];
    for (int j = 0; j < size; j++){
        result = result - matrix[num][j] *root[j];
    }
    result = result/matrix[num][num];
    return result;
}
```

Метод для вычисления невязки:

```
public double[] getDiscrepancies(double[][] matrix, int size, double[] root)
{
    double[] discrepancies = new double[size];
    for (int i = 0; i < size; i++) {
        discrepancies[i] = matrix[i][size];
        for (int j = 0; j < size; j++) {
            discrepancies[i] = discrepancies[i] - matrix[i][j] * root[j];
        }
```

```
    }
    return discrepancies;
}
```

Примеры

1) **Матрица:**

   3

   1 2 3 4

   1 2 2 4

   1 2 6 6

   Ответ:

   # Determinant equals: -55.0

   # Triangular matrix:

   1.0 2.0 3.0 5.0

   0.0 -8.0 -7.0 -19.0

   0.0 0.0 -6.875 2.625

   # Equation roots:

   X_1 = 0.7272727272727275

   X_2 = 2.709090909090909

   X_3 = -0.38181818181818183

   # Discrepancies:

   Dis_1 = 0.0

   Dis_2 = -8.881784197001252E-16

   Dis_3 = 0.0

2) **Матрица:**
   4
   4.744121304981734 0.780936533690596 5.5301640429377645 0.6889913860207852 9.62583547042367
   6.558317463842309 8.206266311243626 6.524944556148968 6.575910300306874 9.685585746710684
   2.422719302483316 8.52756805357041 2.73408333999299 0.909542080994804 6.979524449144925
   3.87227607763446 5.902833179148996 2.5534413545030876 5.891378274567435 3.2069617225123648

Ответ:

Determinant equals: 177.18466622798394
Triangular matrix:
4.744121304981734 0.780936533690596 5.5301640429377645 0.6889913860207852 9.62583547042367
0.0 7.126692375881273 -1.1200057073753698 5.623442193171399 -3.6212588323815798
0.0 0.0 1.187436378691472 -5.856452731626324 6.194256180901585
0.0 0.0 0.0 -4.4133867494804075 3.9355196163095774

# Equation roots:
X_1 = 1.151031917188143
X_2 = 0.3241369286430268
X_3 = 0.81850369519474
X_4 = -0.8917232591893085

# Discrepancies:
Dis_1 = -1.1102230246251565E-16
Dis_2 = 0.0
Dis_3 = 0.0
Dis_4 = 0.0

3) **Матрица:**
5
0.44370269577217014 9.09285428288397 4.935512939974451 7.52043609961561 3.542283949675671 0.21544176898191214
7.823721736467269 3.4653905668627183 2.7772288849800564 9.109085877940302 5.946629081697918 8.713335564993766
8.907164428969988 6.466964170104741 3.773030529140785 8.06936037079565 3.580408527545255 3.754795625637757
7.93439128913036 0.5374480532312065 0.2758966417591602 7.654624743399975 1.784360884840599 7.302473233386879
8.199745037280515 1.2811527546402435 8.795426243324155 0.6828960250188598 3.0903924646196055 6.760762914718331

Ответ:
Determinant equals: 8320.506903836907
Triangular matrix:
0.44370269577217014 9.09285428288397 4.935512939974451 7.52043609961561 3.542283949675671 0.21544176898191214
0.0 -156.8671076969237 -84.24969305403138 -123.49727405837045 -56.51376207185734 4.914493531831463
0.0 0.0 -0.7431299765873689 -4.286419022465566 -4.098193840136737 -6.086176129980279
8.881784197001252E-16 0.0 0.0 6.19249837571287 2.0197033662706865 6.085558730539221

6.918856045027392E-15 -2.8421709430404007E-14 0.0 0.0 -25.977368025002406 -13.574478661214826

# Equation roots:
X_1 = 0.07837606489530896
X_2 = -1.1935696041626898
X_3 = 0.6227831184279418
X_4 = 0.8122993406828278
X_5 = 0.5225501924656036
# Discrepancies:
Dis_1 = 8.881784197001252E-16
Dis_2 = -4.973799150320701E-14
Dis_3 = 4.440892098500626E-16
Dis_4 = -2.220446049250313E-16
Dis_5 = -3.552713678800501E-14

Вывод:

Прямые методы при отсутствии ошибок округления за конечное число арифметических операций позволяют получить точное решение. Эти методы сравнительно просты и наиболее универсальны.

Недостатками моего метода являются:

1) Требование хранения в оперативной памяти всей матрицы, при больших n расходуется память.
2) Происходит накапливание погрешностей в процессе решения, поскольку вычисления на любом этапе используют результаты предыдущих вычислений.

В отличие от прямых методов, итерационные не требуют такого большого расхода памяти, каждую итерацию можно распараллеливать по строкам, а точность вычисления в них поддается. Однако на практике итерационные методы куда менее универсальны, особенно для больших систем, ввиду специфического условия сходимости. Не все системы можно решать с их помощью. А при наличии неограниченных ресурсов времени и памяти, а также достаточно большой разрядной сетки прямыми методами можно решить любую систему. Алгоритмическая сложность метода равна $O(n^3)$.