МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчет по Лабораторной работе №2
«Прямые методы решения СЛАУ»

Вариант 5

Подготовила:
Врублевская Екатерина Александровна,
2 курс 13 группа

2021

1. Постановка задачи

Обращение матрицы методом Гаусса с выбором ГЭ по столбцу

1. Написать программу, которая обращает матрицу методом Гаусса с выбором главного элемента по столбцу. Применить программу к следующим ниже входным данным и вывести результат.

$$A = \begin{pmatrix} -1 & -4 & -5 & -1 & 0 & -1 & -1 \\ -1 & -4 & 0 & 4 & -4 & 0 & -4 \\ -2 & -8 & -5 & -1 & 0 & -4 & 0 \\ -4 & -16 & -10 & 2 & -4 & -2 & 0 \\ -8 & -32 & -20 & 4 & -8 & 0 & -2 \\ -16 & -64 & -40 & 8 & -16 & -7 & -4 \\ 5 & -5 & -5 & -2 & 0 & -2 & -1 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 & 32 & 64 & 128 & 256 & 512 \\ 1 & 3 & 9 & 27 & 81 & 243 & 729 & 2187 & 6561 & 19683 \\ 1 & 4 & 16 & 64 & 256 & 1024 & 4096 & 16384 & 65536 & 262144 \\ 1 & 5 & 25 & 125 & 625 & 3125 & 15625 & 78125 & 390625 & 1953125 \\ 1 & 6 & 36 & 216 & 1296 & 7776 & 46656 & 279936 & 1679616 & 10077696 \\ 1 & 7 & 49 & 343 & 2401 & 16807 & 117649 & 823543 & 5764801 & 40353607 \\ 1 & 8 & 64 & 512 & 4096 & 32768 & 262144 & 2097152 & 16777216 & 134217728 \\ 1 & 9 & 81 & 729 & 6561 & 59049 & 531441 & 4782969 & 43046721 & 387420489 \\ 1 & 10 & 100 & 1000 & 10000 & 100000 & 1000000 & 10000000 & 100000000 & 1000000000 \end{pmatrix}$$

$$A = \begin{pmatrix} 5 & 3 & 3 & -4 & 5 & -5 & -4 & -5 & 0 & 2 \\ 5 & 3 & -1 & 2 & 3 & 0 & -4 & 1 & -4 & -5 \\ 10 & 6 & 2 & -3 & -2 & -2 & -1 & 0 & -3 & -5 \\ 20 & 12 & 4 & -5 & -1 & -4 & 4 & -2 & -2 & -4 \\ 40 & 24 & 8 & -10 & 5 & -1 & 5 & 2 & 0 & -3 \\ 80 & 48 & 16 & -20 & 10 & -12 & -3 & -3 & 3 & 2 \\ 160 & 96 & 32 & -40 & 20 & -24 & -3 & -4 & 1 & 4 \\ 320 & 192 & 64 & -80 & 40 & -48 & -6 & -11 & 0 & -3 \\ 640 & 384 & 128 & -160 & 80 & -96 & -12 & -22 & -5 & 2 \\ -3 & -3 & 0 & 0 & 5 & 3 & -2 & 2 & 5 & -2 \end{pmatrix}$$

2. Найти точные обратные матрицы с использованием библиотеки SymPy и вычислить нормы разности между точной и приближенной матрицами A^{-1}.
3. Проведите экспериментальное исследование скорости обращения матрицы в зависимости от размерности системы, используя для тестов матрицу A со случайными числами. Постройте график зависимости времени работы от размерности. Матрицу размерности ваша программа на вашем компьютере может обратить за одну минуту?

2. Основная часть

Входные данные:

$$A = \begin{pmatrix} -1 & -4 & -5 & -1 & 0 & -1 & -1 \\ -1 & -4 & 0 & 4 & -4 & 0 & -4 \\ -2 & -8 & -5 & -1 & 0 & -4 & 0 \\ -4 & -16 & -10 & 2 & -4 & -2 & 0 \\ -8 & -32 & -20 & 4 & -8 & 0 & -2 \\ -16 & -64 & -40 & 8 & -16 & -7 & -4 \\ 5 & -5 & -5 & -2 & 0 & -2 & -1 \end{pmatrix}$$

Результат выполнения:

```
The determinant of the matrix is zero

Process finished with exit code 0
```

Входные данные:

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 & 32 & 64 & 128 & 256 & 512 \\ 1 & 3 & 9 & 27 & 81 & 243 & 729 & 2187 & 6561 & 19683 \\ 1 & 4 & 16 & 64 & 256 & 1024 & 4096 & 16384 & 65536 & 262144 \\ 1 & 5 & 25 & 125 & 625 & 3125 & 15625 & 78125 & 390625 & 1953125 \\ 1 & 6 & 36 & 216 & 1296 & 7776 & 46656 & 279936 & 1679616 & 10077696 \\ 1 & 7 & 49 & 343 & 2401 & 16807 & 117649 & 823543 & 5764801 & 40353607 \\ 1 & 8 & 64 & 512 & 4096 & 32768 & 262144 & 2097152 & 16777216 & 134217728 \\ 1 & 9 & 81 & 729 & 6561 & 59049 & 531441 & 4782969 & 43046721 & 387420489 \\ 1 & 10 & 100 & 1000 & 10000 & 100000 & 1000000 & 10000000 & 100000000 & 1000000000 \end{pmatrix}$$

Результат выполнения:

Полученная мной матрица:

```
9,9999999999E0 -4,5E1 1,2E2 -2,1E2 2,52E2 -2,1E2 1,2E2 -4,4999999999E1 9,9999999999E0 -9,9999999999E-1
-1,928968254E1 1,0930357143E2 -3,1147619047E2 5,6258333333E2 -6,8769999999E2 5,8008333333E2 -3,3433333333E2 1,2617857143E2 -2,8178571428E1 2,8289682539E0
1,5855753968E1 -1,0350267857E2 3,1791984127E2 -5,9740833333E2 7,4812499999E2 -6,413736111E2 3,7398333333E2 -1,4238214286E2 3,2014484127E1 -3,2316468254E0
-7,3185736331E0 5,253313492E1 -1,7211865079E2 3,3730879629E2 -4,3436805555E2 3,7976527777E2 -2,2466574074E2 8,6486706348E1 -1,9617162698E1 1,9942680776E0
2,0975694444E0 -1,6106076389E1 5,5620833333E1 -1,1341180555E2 1,5041319444E2 -1,3444479167E2 8,089861111E1 -3,1561805555E1 7,2364583333E0 -7,4218749999E-1
-3,8825231481E-1 3,1331597222E0 -1,1289583333E1 2,3849305555E1 -3,2560069444E1 2,9794791666E1 -1,8272916666E1 7,2409722222E0 -1,6817708333E0 1,7436342592E-1
4,6527777777E-2 -3,8993055555E-1 1,4541666666E0 -3,1680555555E0 4,4444444444E0 -4,1645833333E0 2,6069444444E0 -1,0513888889E0 2,4791666666E-1 -2,6041666666E-2
-3,4887566137E-3 3,0109126984E-2 -1,1547619047E-1 2,5833333333E-1 -3,7152777777E-1 3,5625E-1 -2,2777777778E-1 9,365079365E-2 -2,2470238095E-2 2,3974867725E-3
1,4880952381E-4 -1,314484127E-3 5,1587301586E-3 -1,1805555555E-2 1,7361111111E-2 -1,7013888889E-2 1,1111111111E-2 -4,6626984127E-3 1,1408730159E-3 -1,2400793651E-4
-2,7557319223E-6 2,4801587301E-5 -9,9206349205E-5 2,3148148148E-4 -3,4722222222E-4 3,4722222222E-4 -2,3148148148E-4 9,9206349206E-5 -2,4801587301E-5 2,7557319224E-6
```

Полученная методом из библиотеки sympy матрица:

```
Binv
Matrix([
[          10,         -45,         120,        -210,         252,        -210,         120,         -45,          10,         -1],
[    -4861/252,      6121/56,     -6541/21,      6751/12,     -6877/10,     6961/12,     -1003/3,      3533/28,      -789/28,  7129/2520],
[   79913/5040, -115923/1120,  400579/1260,   -71689/120,      5985/8, -461789/720,     22439/60,   -39867/280,  161353/5040, -6515/2016],
[-663941/90720,  264767/5040, -433739/2520,  728587/2160,  -62549/144,  273431/720, -242639/1080, 435893/5040, -197741/10080,  4523/2268],
[    6041/2880,  -92771/5760,    13349/240, -163313/1440,    43319/288, -129067/960,    58247/720,  -45449/1440,      6947/960,    -95/128],
[  -6709/17280,   18047/5760,    -5419/480,   34343/1440, -93773/2880,    28603/960,    -8771/480,   10427/1440,    -3229/1920, 3013/17280],
[      67/1440,   -1123/2880,      349/240,    -2281/720,        40/9,    -1999/480,     1877/720,     -757/720,       119/480,    -5/192],
[   -211/60480,    607/20160,     -97/840,       31/120,    -107/288,       57/160,     -41/180,       59/630,     -151/6720,   29/12096],
[       1/6720,    -53/40320,      13/2520,     -17/1440,       5/288,     -49/2880,       1/90,    -47/10080,      23/20160,    -1/8064],
[    -1/362880,     1/40320,     -1/10080,      1/4320,     -1/2880,      1/2880,     -1/4320,      1/10080,     -1/40320,  1/362880]])
```

Норма разности между полученными матрицами: $4.994296753935146 * 10^{-7}$

```
4.994296753935146E-7
```

Входные данные:

$$A = \begin{pmatrix} 5 & 3 & 3 & -4 & 5 & -5 & -4 & -5 & 0 & 2 \\ 5 & 3 & -1 & 2 & 3 & 0 & -4 & 1 & -4 & -5 \\ 10 & 6 & 2 & -3 & -2 & -2 & -1 & 0 & -3 & -5 \\ 20 & 12 & 4 & -5 & -1 & -4 & 4 & -2 & -2 & -4 \\ 40 & 24 & 8 & -10 & 5 & -1 & 5 & 2 & 0 & -3 \\ 80 & 48 & 16 & -20 & 10 & -12 & -3 & -3 & 3 & 2 \\ 160 & 96 & 32 & -40 & 20 & -24 & -3 & -4 & 1 & 4 \\ 320 & 192 & 64 & -80 & 40 & -48 & -6 & -11 & 0 & -3 \\ 640 & 384 & 128 & -160 & 80 & -96 & -12 & -22 & -5 & 2 \\ -3 & -3 & 0 & 0 & 5 & 3 & -2 & 2 & 5 & -2 \end{pmatrix}$$

Полученная мной матрица:

```
-6,7482142857E-1 -4,2482142857E-1 -2,9982142857E-1 -4,9821428571E-2 -7,7482142857E-1 -7,8732142857E-1 -8,1232142857E-1 -7,2321428571E-2 4,0482142857E-1 5E-1
3,0674886621E-1 5,6748866213E-2 -6,8251133787E-2 -7,9444160998E-1 5,1151077098E-1 -1,2360827664E-1 1,458361678E-1 1,0623441043E0 -5,6500283447E-1 -8,3333333333E-1
1,0169019274E1 9,6690192744E0 6,9190192744E0 1,2847590703E1 1,045473356E1 2,213925737E1 1,5000368481E1 -1,4186933107E1 -7,4203514739E-1 0E0
5,7498639456E0 5,7498639456E0 3,7498639456E0 7,6070068027E0 5,921292517E0 1,2907006803E1 8,6403401361E0 -8,218707483E0 -4,2034013605E-1 0E0
-5,6424036281E-1 -5,6424036281E-1 -5,6424036281E-1 -8,4995464853E-1 -6,2138321995E-1 -1,6166213152E0 -8,6106575964E-1 1,0252834467E0 -1,2267573696E-2 0E0
2,1253968254E-1 2,1253968254E-1 2,1253968254E-1 2,1253968254E-1 4,1253968254E-1 9,7920634921E-1 -1,0968253968E-1 -6,3412698413E-1 1,8301587302E-1 0E0
-4,1587301587E-1 -4,1587301587E-1 -4,1587301587E-1 -4,1587301587E-1 -4,1587301587E-1 -1,0825396825E0 -5,2698412698E-1 6,5079365079E-1 -6,3492063492E-3 0E0
3,380952381E-1 3,380952381E-1 3,380952381E-1 3,380952381E-1 3,380952381E-1 3,380952381E-1 1,0047619048E0 -2,619047619E-1 -2,0476190476E-1 0E0
-1,1428571429E-1 -1,1428571429E-1 -1,1428571429E-1 -1,1428571429E-1 -1,1428571429E-1 -1,1428571429E-1 -1,1428571429E-1 2,8571428571E-1 -8,5714285714E-2 0E0
-7,1428571429E-2 -7,1428571429E-2 -7,1428571429E-2 -7,1428571429E-2 -7,1428571429E-2 -7,1428571429E-2 -7,1428571429E-2 -7,1428571429E-2 7,1428571429E-2 -0E0
```

Полученная методом из библиотеки sympy матрица:

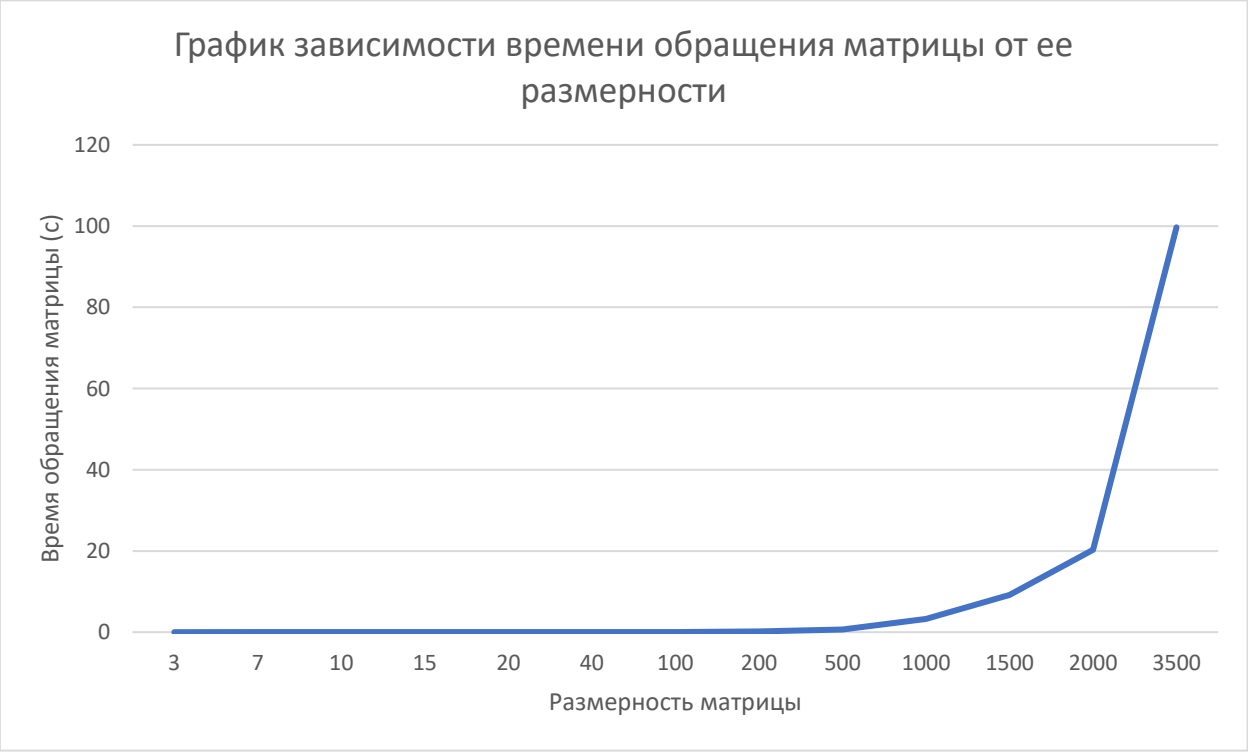
```
Cinv
Matrix([
[   -3779/5600,   -2379/5600,    -1679/5600,      -279/5600,    -4339/5600,    -4409/5600,   -4549/5600,     -81/1120,      2267/5600,  1/2],
[108221/352800, 20021/352800, -24079/352800, -280279/352800, 180461/352800, -43609/352800, 51451/352800,  74959/70560, -199333/352800, -5/6],
[ 358763/35280, 341123/35280,  244103/35280,   453263/35280,  368843/35280,  781073/35280, 529213/35280, -100103/7056,   -26179/35280,    0],
[  84523/14700,  84523/14700,   55123/14700,   111823/14700,   87043/14700,  189733/14700, 127013/14700,  -24163/2940,    -6179/14700,    0],
[ -24883/44100, -24883/44100,  -24883/44100,  -37483/44100,  -27403/44100,  -71293/44100, -37973/44100,    9043/8820,     -541/44100,    0],
[    1339/6300,    1339/6300,     1339/6300,     1339/6300,     2599/6300,     6169/6300,    -691/6300,    -799/1260,      1153/6300,    0],
[     -131/315,     -131/315,      -131/315,      -131/315,      -131/315,      -341/315,     -166/315,       41/63,         -2/315,    0],
[       71/210,       71/210,        71/210,        71/210,        71/210,        71/210,       211/210,      -11/42,        -43/210,    0],
[        -4/35,        -4/35,         -4/35,         -4/35,         -4/35,         -4/35,        -4/35,         2/7,           -3/35,    0],
[        -1/14,        -1/14,         -1/14,         -1/14,         -1/14,         -1/14,        -1/14,        -1/14,           1/14,    0]])
```

Норма разности между полученными матрицами: $6.1755308999963336 * 10^{-12}$

```
6.1755308999963336E-12
```

Провожу экспериментальное исследование скорости обращения матрицы в зависимости от размерности системы, используя для тестов матрицу A со случайными числами.

График зависимости времени работы программы от размерности матрицы



Как мы видим, начиная с размерности матрицы 1000*1000 резко возрастает время обращения матрицы.

Моя программа на моем компьютере может обратить за одну минуту матрицу размерности 3100 (3100*3100).

Листинг кода программы:

```
public class GaussianMethod {
    private final double[][] matrix;
    @Getter
    private final double[][] decomposition;
    @Getter
    private final int[] permutations;
    @Getter
    private boolean isDecomposed = false;

    public GaussianMethod(double[][] matrix) {
        this.matrix = matrix;
        this.decomposition = copyMatrix(matrix);
        permutations = IntStream.range(0, matrix.length).toArray();
    }

    public void calcDecomposition() {
        if (matrix == null)
            return;
        int n = matrix.length;
        for (int k = 0; k < n - 1; k++) {
            var mainInd = findMainValue(decomposition, k);
            if (mainInd == -1)
                continue;
            rearrangeRows(decomposition, k, mainInd);
            for (int i = k + 1; i < n; i++) {
                decomposition[i][k] /= decomposition[k][k];
                for (int j = k + 1; j < n; j++) {
                    decomposition[i][j] -= decomposition[i][k] *
decomposition[k][j];
                }
            }
        }
        isDecomposed = true;
    }

    private void rearrangeRows(double[][] matrix, int k, int i) {
        var tmp = matrix[i];
        matrix[i] = matrix[k];
        matrix[k] = tmp;
        var num = permutations[i];
        permutations[i] = permutations[k];
        permutations[k] = num;
    }

    private int findMainValue(double[][] matrix, int k) {
        double max = decomposition[k][k];
        int iMax = k;
        for (int i = k; i < matrix.length; i++) {
            if ((Math.abs(decomposition[i][k]) > max)){ //&&
(decomposition[i][k] != 0)) {
                max = decomposition[i][k];
                iMax = i;
            }
        }
        iMax = (max == 0) ? -1 : iMax;
        return iMax;
    }
```

```
    public double[] solve(double[] b) {
        if (!isDecomposed()) {
            calcDecomposition();
        }
        var y = new double[b.length];
        for (int i = 0; i < y.length; i++) {
            double sum = 0;
            for (int j = 0; j < i; j++) {
                sum += decomposition[i][j] * y[j];
            }
            y[i] = b[i] - sum;
        }
        var x = new double[b.length];
        for (int i = x.length - 1; i >= 0; i--) {
            double sum = 0;
            for (int j = y.length - 1; j > i; j--) {
                sum += decomposition[i][j] * x[j];
            }
            x[i] = (y[i] - sum) / decomposition[i][i];
        }
        return x;
    }

    public double[][] findInverseMatrix() {
        double[][] result = new double[matrix.length][matrix.length];

        if (!isDecomposed) {
            calcDecomposition();
        }

        for (int i = 0; i < matrix.length; i++) {
            double[] b = new double[matrix.length];
            for (int j = 0; j < matrix.length; j++) {
                b[j] = (permutations[j] == i) ? 1.0 : 0.0;
            }
            result[i] = solve(b);
        }
        transposeMatrix(result);
        return result;
    }

    private void transposeMatrix(double[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = i; j < matrix.length; j++) {
                double tmp = matrix[i][j];
                matrix[i][j] = matrix[j][i];
                matrix[j][i] = tmp;
            }
        }
    }

    private boolean isDecomposed() {
        return isDecomposed;
    }

    private double[][] copyMatrix(double[][] from) {
        double[][] to = new double[from.length][from.length];
        for (int i = 0; i < to.length; ++i) {
            System.arraycopy(from[i], 0, to[i], 0, to.length);
        }
        return to;
    }
}
```

```
public class MainApp {
    private static final NumberFormat numFormat = new
DecimalFormat("0.##########E0");

    public static void main(String[] args) {

        double[][] A = {{-1, -4, -5, -1, 0, -1, -1},
                {-1, -4, 0, 4, -4, 0, -4},
                {-2, -8, -5, -1, 0, -4, 0},
                {-4, -16, -10, 2, -4, -2, 0},
                {-8, -32, -20, 4, -8, 0, -2},
                {-16, -64, -40, 8, -16, -7, -4},
                {5, -5, -5, -2, 0, -2, -1}};


        double[][] B = {{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
                {1, 2, 4, 8, 16, 32, 64, 128, 256, 512},
                {1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683},
                {1, 4, 16, 64, 256, 1024, 4096, 16384, 65536, 262144},
                {1, 5, 25, 125, 625, 3125, 15625, 78125, 390625, 1953125},
                {1, 6, 36, 216, 1296, 7776, 46656, 279936, 1679616,
10077696},
                {1, 7, 49, 343, 2401, 16807, 117649, 823543, 5764801,
40353607},
                {1, 8, 64, 512, 4096, 32768, 262144, 2097152, 16777216,
134217728},
                {1, 9, 81, 729, 6561, 59049, 531441, 4782969, 43046721,
387420489},
                {1, 10, 100, 1000, 10000, 100000, 1000000, 10000000,
100000000, 1000000000}};


        double[][] C = {{5, 3, 3, -4, 5, -5, -4, -5, 0, 2},
                {5, 3, -1, 2, 3, 0, -4, 1, -4, -5},
                {10, 6, 2, -3, -2, -2, -1, 0, -3, -5},
                {20, 12, 4, -5, -1, -4, 4, -2, -2, -4},
                {40, 24, 8, -10, 5, -1, 5, 2, 0, -3},
                {80, 48, 16, -20, 10, -12, -3, -3, 3, 2},
                {160, 96, 32, -40, 20, -24, -3, -4, 1, 4},
                {320, 192, 64, -80, 40, -48, -6, -11, 0, -3},
                {640, 384, 128, -160, 80, -96, -12, -22, -5, 2},
                {-3, -3, 0, 0, 5, 3, -2, 2, 5, -2}};

        GaussianMethod method = new GaussianMethod(C);

        var matrix = MatrixUtils.createRealMatrix(C);
        var res = MatrixUtils.inverse(matrix).getData();

        LUDecomposition decomposition = new LUDecomposition(matrix);
        var determinant = decomposition.getDeterminant();
        if(determinant == 0)
            System.out.println("The determinant of the matrix is zero");
        else{
            double[][] result = method.findInverseMatrix();
            printMatrix(result);
            System.out.println("\nExact value:");
            printMatrix(res);

            System.out.println(getNorm(result, res));
        }


        testInverse();
```

```
    }

    private static void testInverse() {

        var A = generateRandomMatrix(10);
        GaussianMethod method = new GaussianMethod(A);

        var startTime = System.nanoTime();
        var inverse = method.findInverseMatrix();
        var stopTime = System.nanoTime();
        System.out.println("Inversing time " + (stopTime - startTime));
    }


    public static void printMatrix(double[][] A) {
        for (int i = 0; i < A.length; i++) {
            for (int j = 0; j < A.length; j++) {
                System.out.print(numFormat.format(A[i][j]) + " ");
            }
            System.out.println();
        }
    }

    public static double getNorm(double[][] matrix1, double[][] matrix2) {
        double[][] result = new double[matrix1.length][matrix1.length];
        for (int i = 0; i < matrix1.length; i++) {
            for (int j = 0; j < matrix1.length; j++) {
                result[i][j] = Math.abs(matrix1[i][j] - matrix2[i][j]);
            }
        }
        double norm = Arrays.stream(result[0]).sum();
        for (int i = 1; i < result.length; i++) {
            double tmp = Arrays.stream(result[i]).sum();
            norm = Math.max(tmp, norm);
        }
        return norm;
    }

    public static double[][] generateRandomMatrix(int dimension){
        double[][] matrix = new double[dimension][dimension];
        Random random = new Random(System.currentTimeMillis());
        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                matrix[i][j] = random.nextDouble();
            }
        }
        return matrix;
    }
}
```

3. Заключение

Метод Гаусса с выбором ГЭ по столбцу получает LU-разложение исходной матрицы один раз и потом использует ее для дальнейших расчетов. Это позволяет неплохо уменьшить время вычисления обратной матрицы, так как построение разложения здесь самая трудоемкая операция ($O(n^3)$). Для вычисления обратной матрицы я использую класс GaussianMethod. В нем есть поля matrix и decomposition, которые представляют собой исходную матрицу и ее LU-разложение, хранящееся в одной матрице, соответственно. Также т.к. реализовываю метод с выбором ГЭ по столбцу, в классе есть поле permutations, которое хранит вектор перестановок строк. LU-разложение получаем в

методе calcDecomposition(), далее рассчитывается обратная матрица в методе findInverseMatrix(). Как именно она считается? Исходя из того, что $AB = I$, где B=A^{-1}, I — единичная матрица. Каждый столбец матрицы рассчитывается по следующей схеме:

$\begin{cases} Ly_i = l_i \\ Ub_i = y_i \end{cases}$, где b_i— искомый столбец обратной матрицы, l_i— столбец единичной матрицы, в котором i-я компонента равна 1.

Каждая СЛАУ решается в методе solve().

Вычислительная сложность алгоритма — $O(n^3)$.