Московский Авиационный Институт
(Национальный исследовательский университет)

Лабораторная работа №1

по курсу: «численные методы»

Выполнил: Коростелев Д.В.

Группа: М8О-308Б-18

Номер по списку: 10

г. Москва 2021

Скриншоты заданий

1 10 Методом Гаусса вычислить решение СЛАУ,
определитель системы, обратную матрицу

-5	-6	5	10	-50
8	7	-4	1	-3
-3	2	9	7	-104
-3	-10	-4	6	51

1 10 Методом ПРОГОНКИ решить слау с трёхдиагональной матрицей

-10	-3	0	0	0	0	0	0	0	2
4	-7	1	0	0	0	0	0	0	35
0	-2	9	4	0	0	0	0	0	43
0	0	3	8	1	0	0	0	0	20
0	0	0	-5	11	-4	0	0	0	-53
0	0	0	0	5	-9	1	0	0	0
0	0	0	0	0	-5	11	-2	0	-5
0	0	0	0	0	0	5	7	3	-29
0	0	0	0	0	0	0	5	-11	39

1_10 Методом Простых итераций и методом Зейделя,
сделав соответственно 10 и 5 итераций,
вычислить решение СЛАУ, доказать сходимость

37	1	10	-357
3	36	-8	-237
-1	6	-39	310

1 10 Степенным методом , сделав 10 итераций, вычислить спектральный радиус и его вектор
Методом Якоби вычислить собственные числа и собственные векторы, сделав 6 итераций,
у матрицы

-17	4	-4
4	-14	-8
-4	-8	-4

Условие:

```
1 10  Методом Гаусса вычислить решение СЛАУ,
      определитель системы, обратную матрицу
        -5   -6    5   10          -50
         8    7   -4    1           -3
        -3    2    9    7         -104
        -3  -10   -4    6           51
```

Теория:

1.1. Метод Гаусса решения СЛАУ

Рассмотрим схему единственного деления метода Гаусса. Пусть дана система n линейных алгебраических уравнений с n неизвестными (СЛАУ):

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases} \qquad (1.1.1)$$

В матричной форме эта система выглядит как $A\bar{x}=\bar{b}$, $A=\{a_{ij}\}$, $(i,j=1,\ldots,n)$, $\bar{x}=(x_1,x_2,\ldots,x_n)^T$, $\bar{b}=(b_1,b_2,\ldots,b_n)^T$. Задача (1.1.1) имеет единственное решение, если определитель (детерминант) матрицы системы не равен нулю ($|A|\neq 0$ или $\det A \neq 0$).

Метод Гаусса заключается в исключении из (1.1.1) тех слагаемых, которые лежат в матрице A ниже главной диагонали (a_{ij}, $i>j$). Исключать слагаемые разрешается только с помощью трёх допустимых преобразований:

1) любую строку (уравнение) можно умножить (разделить) на любое число, кроме нуля;
2) любую строку можно прибавить к другой строке;
3) можно переставить любые две строки.

При каждом применении третьего преобразования определитель будет менять свой знак.

Перепишем систему (1.1.1) в виде расширенной матрицы, которую будем преобразовывать к верхней треугольной форме с единицами на главной диагонали и с нулями под главной диагональю:

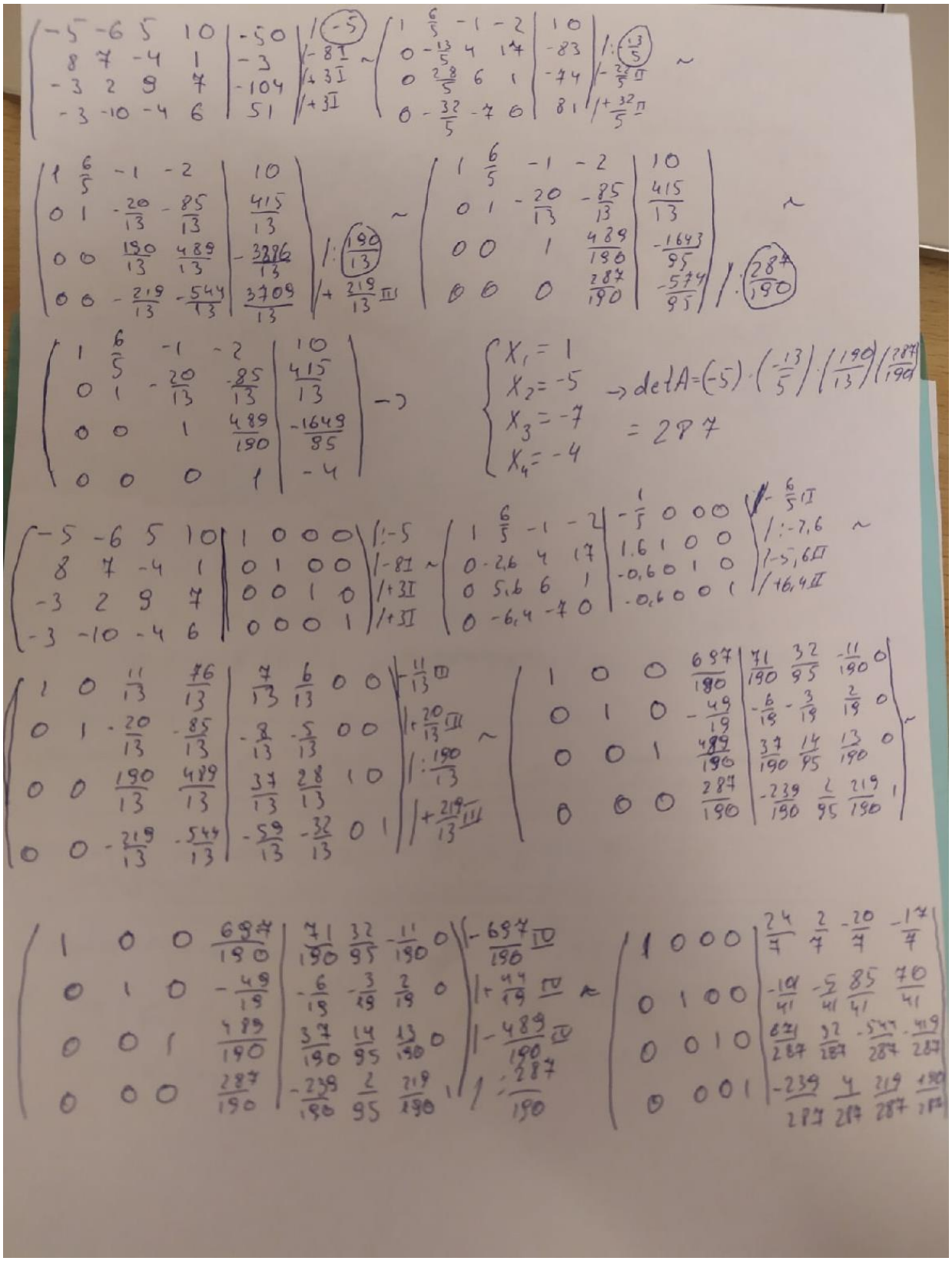
$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \ldots & a_{3n} & b_3 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & a_{n3} & \ldots & a_{nn} & b_n \end{array}\right) \Rightarrow \left(\begin{array}{ccccc|c} 1 & c_{12} & c_{13} & \ldots & c_{1n} & d_1 \\ 0 & 1 & c_{23} & \ldots & c_{2n} & d_2 \\ 0 & 0 & 1 & \ldots & c_{3n} & d_3 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1 & d_n \end{array}\right) . \qquad (1.1.2)$$

С помощью допустимых преобразований обнуляются элементы матрицы, лежащие ниже главной диагонали. Это преобразование расширенной матрицы системы (матрица системы с добавленным столбцом правых частей) называют *прямым ходом* метода Гаусса. Правая из (1.1.2) матрица соответствует системе:

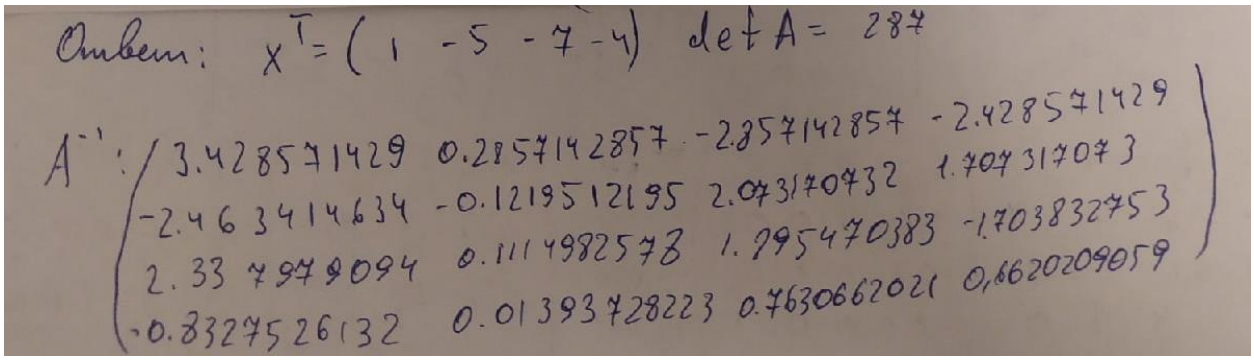
$$\begin{cases} x_1 + c_{12}x_2 + \ldots + c_{1n-1}x_{n-1} + c_{1n}x_n = d_1 \\ x_2 + \ldots + c_{2n-1}x_{n-1} + c_{2n}x_n = d_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_{n-1} + c_{n-1n}x_n = d_{n-1} \\ x_n = d_n \end{cases} \qquad (1.1.3)$$

При *обратном ходе* вычисляются неизвестные x_n, x_{n-1}, x_{n-2}, ... , x_1 . Из последнего уравнения находим x_n , подставляем его в предпоследнее уравнение и получаем x_{n-1}. Так, в обратном порядке находим все неизвестные x.

Решение:



Ответ:


Ответ: $X^T = (1\ -5\ -7\ -4)$ $\det A = 287$
A^{-1}:
3.4285714129 0.2857142857 -2.857142857 -2.4285714129
-2.463414634 -0.12195121195 2.073170732 1.707317073
2.337979094 0.1114982578 1.295470383 -1.703832753
-0.8327526132 0.01393728223 0.7630662021 0,6620209059

Метод прогонки

Условие:

```
1 10  Методом ПРОГОНКИ решить слау с трёхдиагональной матрицей
      -10   -3    0    0    0    0    0    0    0        2
        4   -7    1    0    0    0    0    0    0       35
        0   -2    9    4    0    0    0    0    0       43
        0    0    3    8    1    0    0    0    0       20
        0    0    0   -5   11   -4    0    0    0      -53
        0    0    0    0    5   -9    1    0    0        0
        0    0    0    0    0   -5   11   -2    0       -5
        0    0    0    0    0    0    5    7    3      -29
        0    0    0    0    0    0    0    5  -11       39
```

Теория:

1.6. Решение трёхдиагональной СЛАУ методом прогонки

СЛАУ с трёхдиагональной матрицей встречаются, например, при построении кубических интерполяционных сплайнов, при численном решении дифференциальных уравнений.

Пусть дана СЛАУ специального вида

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i \quad , \; i=1,\ldots,n \; . \tag{1.6.1}$$

Запишем расширенную матрицу для системы (1.3-1) :

$$\left(\begin{array}{ccccccccc|c} b_1 & c_1 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & d_1 \\ a_2 & b_2 & c_2 & 0 & 0 & \ldots & 0 & 0 & 0 & d_2 \\ 0 & a_3 & b_3 & c_3 & 0 & \ldots & 0 & 0 & 0 & d_3 \\ 0 & 0 & a_4 & b_4 & c_4 & \ldots & 0 & 0 & 0 & d_4 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & 0 & \ldots & b_{n-2} & c_{n-2} & 0 & d_{n-2} \\ 0 & 0 & 0 & 0 & 0 & \ldots & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \\ 0 & 0 & 0 & 0 & 0 & \ldots & 0 & a_n & b_n & d_n \end{array}\right) \tag{1.6.2}$$

Заметим, что в первом уравнении нет коэффициента a_1 , а в последнем уравнении отсутствует c_n , т.е. $a_1 = 0$, $c_n = 0$.

При проведении прямого хода метода прогонки вычисляются прогоночные коэффициенты P_i и Q_i .

$P_0 = 0$, $Q_0 = 0$, остальные коэффициенты P_i и Q_i ($i = 1,2,3,\ldots n$) вычисляем по формулам:

$$P_i = \frac{-c_i}{b_i + a_i \cdot P_{i-1}} \; , \; Q_i = \frac{d_i - a_i \cdot Q_{i-1}}{b_i + a_i \cdot P_{i-1}} \qquad \text{для } \; i = 1,2,\ldots n \; . \tag{1.6.3}$$

После вычисления всех прогоночных коэффициентов можно в обратном ходе прогонки вычислить все неизвестные : x_n , x_{n-1} , x_{n-2} , ..., x_1 :

$x_n = Q_n$,

$x_i = Q_i + P_i \cdot x_{i+1}$, $i = n\text{-}1, n\text{-}2, \ldots, 2, 1$. (1.6.4)

Решение:

Код:

```
package first;

public class Progonka {
```

```
    public static int n;
    public static double[][] matrix;
    public static double[] d;
    public static double[] x;
    public static double[] p, q;
    public static double getA(int index){
        return matrix[index-1][0];
    }
    public static double getB(int index){
        return matrix[index-1][1];
    }
    public static double getC(int index){
        return matrix[index-1][2];
    }
    public static double getD(int index){
        return d[index-1];
    }
    //передаем коэффициенты a b c в матрице mAMatrix и матрицу B в mBMatrix
    static void exec(int mN, double[][] mAMatrix, double[] mBMatrix){
        n = mN;
        matrix = mAMatrix;
        d = mBMatrix;
        p = new double[n+1];
        q = new double[n+1];
        x = new double[n];
        p[0] = q[0] = 0;
        System.out.println("P[i]\t\tQ[i]");
        printRes(p[0],q[0]);
        for(int i = 1;i<n+1;++i){
            p[i] = (-getC(i))/(getB(i) + getA(i)*p[i-1]);
            q[i] = (getD(i) - (getA(i)*q[i-1]))/(getB(i) + getA(i)*p[i-1]);
            printRes(p[i],q[i]);
        }
        x[n-1] = q[n];
        for(int i = n-2;i>=0;--i){
            x[i] = q[i+1] + p[i+1]*x[i+1];
        }
        for(int i = 0;i<n;++i){
            System.out.println("x[" + i+ "] : " + x[i]);
        }
    }
    static void printRes(double p, double q){
        System.out.printf("%2.9f %2.9f\n", p,q);
    }
}
```

Вывод консоли:

```
P[i]        Q[i]
0,000000000 0,000000000
-0,300000000 -0,200000000
0,121951220 -4,365853659
-0,456824513 3,913649025
-0,150840336 1,245798319
```

```
0,340303843 -3,979088472
0,137014816 -2,725970369
0,193893782 -1,806106218
-0,376436628 -2,505746510
0,000000000 -4,000000000
x[0] : 1.0
x[1] : -4.0
x[2] : 3.0
x[3] : 2.0
x[4] : -5.0
x[5] : -2.9999999999999996
x[6] : -1.9999999999999998
x[7] : -1.0000000000000002
x[8] : -4.0
```

Пояснение к решению:

Программа на каждом шаге считает значения коэффициентов P_i и Q_i, а затем на основе полученных данных, согласно методу прогонки находит все решения x_i, при этом нумерация x_i в данном решении начинается с 0, так как программа написана на языке Java, в котором нумерация внутри массива начинается с 0.

Приведем пример вычисления коэффициентов P_1 и Q_1:

$$P_1 = \frac{-c_1}{b_1 + a_1 P_0} = \frac{3}{-10 + 0 * 0} = -0.3$$

$$Q_1 = \frac{d_1 - a_1 * Q_0}{b_1 + a_1 * P_0} = \frac{2 - 0 * 0}{-10 + 0 * 0} = -0.2$$

Таблица, полученная в ходе выполнения алгоритма:

Итерация i	P_i	Q_i
0	0,000000000	0,000000000
1	-0,300000000	-0,200000000
2	0,121951220	-4,365853659

3	-0,456824513	3,913649025
4	-0,150840336	1,245798319
5	0,340303843	-3,979088472
6	0,137014816	-2,725970369
7	0,193893782	-1,806106218
8	-0,376436628	-2,505746510
9	0,000000000	-4,000000000

После того как все коэффициента P_k и Q_k были получены, высчитываются все значения x_k, начиная с самого последнего по итерационной формуле.

Приведем пример вычисления x_5:

$$x_4 = Q_5 + P_5 * x_5 = -3{,}979088472 + 0{,}340303843 * -3 = -5$$

Ответ:

$$\begin{cases} x_0 = 1 \\ x_1 = -4 \\ x_2 = 3 \\ x_3 = 2 \\ x_4 = -5 \\ x_5 = -3 \\ x_6 = -2 \\ x_7 = -1 \\ x_8 = -4 \end{cases}$$

Метод простых итераций и метод Зейделя

Условие:

```
1_10     Методом Простых итераций и методом Зейделя,
       сделав соответственно 10 и  5 итераций,
       вычислить решение СЛАУ, доказать сходимость
         37    1   10        -357
          3   36   -8        -237
         -1    6  -39         310
```

Теория:

Метод простых итераций:

2.1 Метод простых итераций решения СЛАУ

Для решения системы $A\bar{x}=\bar{b}$ каким-либо образом преобразуем эту систему к виду (схеме) $\bar{x} = B\bar{x}+\bar{\beta}$. По этой схеме можно построить итерационный процесс :

$$\bar{x}^{(n+1)} = B\,\bar{x}^{(n)} + \bar{\beta} \quad . \qquad (2.1.1)$$

В левой части стоит новый вектор неизвестных $\bar{x}^{(n+1)}$, а в правой части – старый вектор неизвестных $\bar{x}^{(n)}$. После вычисления нового вектора он превращается в старый и вычисляем следующий новый вектор. За начальный вектор $\bar{x}^{(1)}$ можно взять вектор $\bar{\beta}$. Если хотя бы какая-нибудь норма матрицы B окажется меньше 1, то последовательность векторов $\bar{x}^{(n)}$ из (2.1.1) будет сходиться к точному решению $\bar{\mu}$. Сходимость будет тем быстрее, чем меньше норма у матрицы B. В процессе итераций точность (близость вектора $\bar{x}^{(n)}$ к $\bar{\mu}$) можно контролировать с помощью несложного соотношения

$$\| \bar{x}^{(n)} - \bar{\mu} \| < \frac{\| B \|^n}{1-\| B \|} \cdot \| \bar{\beta} \| \quad . \qquad (2.1.2)$$

В этом соотношении нормы $\|B\|$ и $\|\bar{\beta}\|$ должны быть согласованы (см. разд. 2). Поэтому желательно выбрать норму матрицы B с наименьшим значением и уже для неё искать ей согласованную норму вектора $\bar{\beta}$. Итак, на последних шагах итерационного процесса вектора $\bar{x}^{(n)}$ и $\bar{x}^{(n+1)}$ будут мало отличаться друг от друга, т.е. их компоненты попарно будут почти одинаковыми. Вычисления прекращают, если правая часть выражения (2.1.2) стала меньше некоторого заранее заданного числа ε ($\varepsilon>0$). Оценка (2.1.2) значительно завышает число итераций, поэтому иногда для оценки точности используют другое соотношение:

$$\| \bar{x}^{(n)} - \bar{\mu} \| \le \frac{\| B \|}{1-\| B \|} \cdot \| \bar{x}^{(n)} - \bar{x}^{(n-1)} \| \quad .$$

Метод Зейделя:

2.2. Метод Зейделя решения СЛАУ

Вернёмся к решению примера (2.1-3). Исследование сходимости итерационного процесса (2.1-5) уже проведено, найдены номы матрицы B и вектора $\overline{\beta}$. За начальный вектор $\overline{x}^{(0)}$ возьмём вектор $\overline{\beta}$, т.е.
$x^{(0)} = 1/20 = 0{,}05$; $y^{(0)} = -2/15 \approx -0{,}133333333$; $z^{(0)} = -1/6 \approx -0{,}16666666$.

Тогда новый $x^{(1)}$ вычислим, подставив числовые значения $x^{(0)}, y^{(0)}, z^{(0)}$ в первое уравнение (2.1-4):

$$x^{(1)} = 0x^{(0)} - \frac{1}{5}y^{(0)} + \frac{2}{5}z^{(0)} + \frac{1}{20} = -\frac{1}{5}\cdot\left(-\frac{2}{15}\right) + \frac{2}{5}\cdot\left(-\frac{1}{6}\right) + \frac{1}{20} = 0{,}01 .$$

Теперь в правую часть второго уравнения (2.1-4) вместо x, y, z подставим только что вычисленное $x^{(1)}$ и старые $y^{(0)}, z^{(0)}$:

$$y^{(1)} = \frac{1}{5}x^{(1)} + 0y^{(0)} - \frac{1}{3}z^{(0)} - \frac{2}{15} = \frac{1}{5}\cdot 0{,}01 + \frac{1}{3}\cdot\frac{1}{6} - \frac{2}{15} = \frac{341}{4500} \approx -0{,}075777777.$$

В правую часть третьего уравнения (2.1-4) вместо x, y, z подставим только что вычисленные $x^{(1)}$, $y^{(1)}$ и старое $z^{(0)}$:

$$z^{(1)} = \frac{1}{3}x^{(1)} + \frac{1}{6}y^{(1)} + 0z^{(0)} - \frac{1}{6} = \frac{1}{3}\cdot 0{,}01 + \frac{1}{6}\cdot(-0{,}7577777) - \frac{1}{6} \approx -0{,}175963 .$$

При вычислении $x^{(2)}$ используем значения $x^{(1)}$, $y^{(1)}$, $z^{(1)}$. При вычислении $y^{(2)}$ используем значения только что вычисленного $x^{(2)}$ и старых $y^{(1)}$, $z^{(1)}$.При вычислении $z^{(2)}$ используем значения только что вычисленных $x^{(2)}$ и $y^{(2)}$ и старого $z^{(1)}$

Этот итерационный процесс называют *методом Зейделя* и записывают:

$$\begin{cases} x^{(n+1)} = 0x^{(n)} - \frac{1}{5}y^{(n)} + \frac{2}{5}z^{(n)} + \frac{1}{20} \\ y^{(n+1)} = \frac{1}{5}x^{(n+1)} + 0y^{(n)} - \frac{1}{3}z^{(n)} - \frac{2}{15} \\ z^{(n+1)} = \frac{1}{3}x^{(n+1)} + \frac{1}{6}y^{(n+1)} + 0z^{(n)} - \frac{1}{6} \end{cases} . \qquad (2.1.6)$$

Такой способ вычислений позволяет иногда серьёзно увеличить скорость сходимости к точному решению. По сравнением с методом простых итераций количество итераций в методе Зейделя может быть меньше в разы.

Решение:

Код:

Файл SimpleIterations.java

```
package first;
abstract public class SimpleIterations {
    int mN;
    double[] exec(int N, double[][] A, double[] B){
        mN = N;
```

```
        double[][] mA = A.clone();
        for(int i = 0;i<A.length;++i){
            mA[i] = A[i].clone();
        }
        double[] mB = B.clone();
        double[] res = new double[mN];
        double[] prev = new double[mN];

        for(int i = 0;i<mN;++i){
            double t = mA[i][i];
            mA[i][i] = 0;
            for(int j = 0;j<mN;++j){
                mA[i][j] /= (-1*t);
            }
            mB[i] /= t;
        }
        for(int i = 0;i<mN;++i){
            prev[i] = mB[i];
        } double normaFirst = 0;
        for(int i = 0;i<mN;++i){
            double sum = 0;
            for(int j = 0;j<mN;++j){
                sum += Math.abs(mA[i][j]);
            }
            normaFirst = Math.max(normaFirst, sum);
        }
        double normaSecond = 0;
        for(int i = 0;i<mN;++i){
            double sum = 0;
            for(int j = 0;j<mN;++j){
                sum += Math.abs(mA[j][i]);
            }
            normaSecond = Math.max(normaSecond, sum);
        }
        double normaB = Math.min(normaFirst, normaSecond);
        double normaBetta = 0;
        for(int i = 0;i<mN;++i){
            normaBetta = Math.max(normaBetta, Math.abs(mB[i]));
        }
        System.out.println("norma B : "+normaB + " norma Betta : " +
normaBetta);
        printRes(0, prev, -1, -1);
        int iteration = 1;
        while(endCondition()){
            for(int i = 0;i<mN;++i){
                double sum = 0;
                for(int j = 0;j<mN;++j){
                    sum += prev[j] * mA[i][j];
                }
                sum += mB[i];
                res[i] = sum;
            }
            double eps = Math.pow(normaB, iteration)*normaBetta/(1.0 -
normaB);
            double omega = calcOmega(res, prev);
            printRes(iteration, res, eps, omega);
            prev = res.clone();
            ++iteration;
        }
        return res;
    }
    private double calcOmega(double[] current, double[] prev){
        double res = 0;
        for(int i = 0;i<mN;++i){
```

```
            res = Math.max(res, Math.abs(current[i] - prev[i]));
        }
        return res;
    }
    private void printRes(int iteration, double[] res, double eps, double
omega){
        System.out.print(iteration + ":");
        for (double re : res) {
            System.out.print("\t" + String.format("%.10f", re));
        }
        if(eps >= 0 || omega >=0){
            System.out.println("\t" + String.format("%.10f",eps) + "\t" +
String.format("%.10f",omega));
        } else {
            System.out.println();
        }
    }
    abstract boolean endCondition();
}
```

Файл Zeidel.java

```
package first;

abstract public class Zeidel {
    int mN;
    //исполняющий метод, нужно передать размерность
    //матрицы A - N,
    //матрицы A и B
    double[] exec(int N, double[][] A, double[] B){
        mN = N;
        double[][] mA = A.clone();
        for(int i = 0;i<A.length;++i){
            mA[i] = A[i].clone();
        }
        double[] mB = B.clone();
        double[] res = new double[mN];
        double[] prev = new double[mN];
        for(int i = 0;i<mN;++i){
            double t = mA[i][i];
            mA[i][i] = 0;
            for(int j = 0;j<mN;++j){
                mA[i][j] /= (-1*t);
            }
            mB[i] /= t;
        }
        for(int i = 0;i<mN;++i){
            prev[i] = mB[i];
        }
        //вычисляем первую норму
        double normaFirst = 0;
        for(int i = 0;i<mN;++i){
            double sum = 0;
            for(int j = 0;j<mN;++j){
                sum += Math.abs(mA[i][j]);
            }
            normaFirst = Math.max(normaFirst, sum);
        }
        //вычисляем вторую норму
        double normaSecond = 0;
        for(int i = 0;i<mN;++i){
            double sum = 0;
```

```
            for(int j = 0;j<mN;++j){
                sum += Math.abs(mA[j][i]);
            }
            normaSecond = Math.max(normaSecond, sum);
        }
        //находим наилучшую норму
        double normaB = Math.min(normaFirst, normaSecond);
        double normaBetta = 0;
        for(int i = 0;i<mN;++i){
            normaBetta = Math.max(normaBetta, Math.abs(mB[i]));
        }
        printRes(0, prev, -1, -1);
        int iteration = 1;
            //выполняем шаг алгоритма
            while(endCondition()){
            for(int i = 0;i<mN;++i){
                double sum = 0;
                for(int j = 0;j<i;++j){
                    sum += res[j] * mA[i][j];
                }
                for(int j = i;j<mN;++j){
                    sum += prev[j] * mA[i][j];
                }
                sum += mB[i];
                res[i] = sum;
            }
            double eps = Math.pow(normaB, iteration)*normaBetta/(1.0 -
normaB);
            double omega = calcOmega(res, prev);
            printRes(iteration, res, eps, omega);
            prev = res.clone();
            ++iteration;
        }
        return res;
}
    private double calcOmega(double[] current, double[] prev){
        double res = 0;
        for(int i = 0;i<mN;++i){
            res = Math.max(res, Math.abs(current[i] - prev[i]));
        }
        return res;
    }
    private void printRes(int iteration, double[] res, double eps, double
omega){
        System.out.print(iteration + ":");
        for (double re : res) {
            System.out.print("\t" + String.format("%.10f", re));
        }
        if(eps >= 0 || omega >=0){
            System.out.println("\t" + String.format("%.10f",eps) + "\t" +
String.format("%.10f",omega));
        } else {
            System.out.println();
        }
    }
    abstract boolean endCondition();
}
```

Файл Main.java

```
package first;
```

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        //Метод простых итераций
        int n2 = 3;
        double[][] aMatrix2 = {
                {37, 1, 10},
                {3, 36, -8},
                {-1, 6, -39}
        };
        double [] bMatrix2 = {-357, -237, 310};

        SimpleIterations taskSimpleIterations = new SimpleIterations(){
            //задаем условие окончания метода
            int iterations = 10;
            @Override
            boolean endCondition() {
                if(iterations > 0){
                    --iterations;
                    return true;
                } else {
                    return false;
                }
            }
        };
        System.out.println("Метод простых итераций");
        System.out.println(Arrays.toString(taskSimpleIterations.exec(n2,
aMatrix2, bMatrix2)));
        System.out.println();

        //Метод Зейделя
        Zeidel taskZeidel = new Zeidel() {
            int iterations = 5;
            @Override
            boolean endCondition() { //задаем условие окончания метода
                if(iterations > 0){
                    --iterations;
                    return true;
                } else {
                    return false;
                }
            }
        };
        System.out.println("Метод Зейделя");
        System.out.println(Arrays.toString(taskZeidel.exec(n2, aMatrix2,
bMatrix2)));
        System.out.println();
    }
}
```

Вывод консоли:

```
Метод простых итераций

norma B : 0.3055555555555555 norma Betta : 9.64864864864865

0:   -9,6486486486  -6,5833333333  -7,9487179487
```

```
1:   -7,3224185724  -7,5456610457  -8,7141372141  4,2454054054
     2,3262300762

2:   -7,0895396436  -7,9096067221  -8,9218345564  1,2972072072
     0,3639456764

3:   -7,0235688571  -7,9751682645  -8,9837974536  0,3963688689
     0,0659707865

4:   -7,0050501946  -7,9944353627  -8,9955754033  0,1211127099
     0,0192670982

5:   -7,0013462325  -7,9985959067  -8,9990144098  0,0370066614
     0,0041605440

6:   -7,0003043242  -7,9996687939  -8,9997494669  0,0113075910
     0,0010728872

7:   -7,0000766632  -7,9999189656  -8,9999412420  0,0034550972
     0,0002501717

8:   -7,0000180707  -7,9999805541  -8,9999855675  0,0010557242
     0,0000615885

9:   -7,0000044263  -7,9999952869  -8,9999965450  0,0003225824
     0,0000147328

10:  -7,0000010612  -7,9999988634  -8,9999991614  0,0000985668
     0,0000035765

[-7.0000010611737045, -7.999998863360861, -8.999999161410777]


Метод Зейделя

0:   -9,6486486486  -6,5833333333  -7,9487179487

1:   -7,3224185724  -7,7395135520  -8,9516580190  4,2454054054
     2,3262300762

2:   -7,0201055745  -7,9875818730  -8,9975739914  1,2972072072
     0,3023129979

3:   -7,0009913031  -7,9993782784  -8,9998789325  0,3963688689
     0,0191142715

4:   -7,0000495242  -7,9999689691  -8,9999939562  0,1211127099
     0,0009417788

5:   -7,0000024721  -7,9999984509  -8,9999996983  0,0370066614
     0,0000470521

[-7.000002472143059, -7.999998450912896, -8.999999698290624]
```

Здесь в выводе для каждой итерации выводятся первые три числа – приблизительный ответ с точностью до эпсилона, которое представлено четвертым числом. Последнее число – омега. Далее эта табличка будет приведена в читаемом виде ниже.

Пояснение к решению:

Вычисленные нормы:

$$||B|| = 0.3055555555, ||\beta|| = 9.64864864864865$$

Итоговая таблица по методу простых итераций.

i	x_1	x_2	x_3	ε	δ
0	-9,6486486486	-6,5833333333	-7,9487179487	-	-
1	-7,3224185724	-7,5456610457	-8,7141372141	4,2454054054	2,3262300762
2	-7,0895396436	-7,9096067221	-8,9218345564	1,2972072072	0,3639456764
3	-7,0235688571	-7,9751682645	-8,9837974536	0,3963688689	0,0659707865
4	-7,0050501946	-7,9944353627	-8,9955754033	0,1211127099	0,0192670982
5	-7,0013462325	-7,9985959067	-8,9990144098	0,0370066614	0,0041605440
6	-7,0003043242	-7,9996687939	-8,9997494669	0,0113075910	0,0010728872
7	-7,0000766632	-7,9999189656	-8,9999412420	0,0034550972	0,0002501717
8	-7,0000180707	-7,9999805541	-8,9999855675	0,0010557242	0,0000615885
9	-7,0000044263	-7,9999952869	-8,9999965450	0,0003225824	0,0000147328
10	-7,0000010612	-7,9999988634	-8,9999991614	0,0000985668	0,0000035765

Итоговая таблица по методу Зейделя:

i	x_1	x_2	x_3	ε	δ
0	-9,6486486486	-6,5833333333	-7,9487179487	-	-
1	-7,3224185724	-7,7395135520	-8,9516580190	4,2454054054	2,3262300762
2	-7,0201055745	-7,9875818730	-8,9975739914	1,2972072072	0,3023129979
3	7,0009913031	-7,9993782784	-8,9998789325	0,3963688689	0,0191142715
4	-7,0000495242	-7,9999689691	-8,9999939562	0,1211127099	0,0009417788
5	-7,0000024721	-7,9999984509	-8,9999996983	0,0370066614	0,0000470521

Ответ

По методу простых итераций $\begin{cases} x_1 = -7.0000010611 \\ x_2 = -7.9999988633 \\ x_3 = -8.9999991614 \end{cases}$

По методу Зейделя $\begin{cases} x_1 = -7.000002472 \\ x_2 = -7.999998450 \\ x_3 = +8.999999698 \end{cases}$

Степенной метод и метод Якоби:

Условие:

```
1 10  Степенным методом , сделав 10 итераций, вычислить спектральный радиус и его вектор
      Методом Якоби вычислить собственные числа и собственные векторы, сделав  6 итераций,
          у матрицы

        -17    4   -4
          4  -14   -8
         -4   -8   -4
```

Теория:

3.1. Степенной метод вычисления спектрального радиуса

Пусть дана матрица A у которой собственные числа (спектр удовлетворяют соотношению $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n|$. Пусть λ_1 - действительное (не комплексное) число.

Построим последовательность векторов $\bar{v}$, $\bar{\omega}$ и чисел ρ:

$$\bar{v}^{(n+1)} = A \cdot \bar{\omega}^{(n)} \quad , \quad \rho_{n+1} = (\bar{v}^{(n+1)}, \bar{\omega}^{(n)}) \quad , \quad \bar{\omega}^{(n+1)} = \frac{\bar{v}^{(n+1)}}{\sqrt{(\bar{v}^{(n+1)}, \bar{v}^{(n+1)})}} \quad . \qquad (3.1.1)$$

За начальный вектор $\bar{\omega}^{(0)}$ можно взять, например, единичный вектор $\bar{e}$ = $\{1;1;\ldots;1\}^T$. Так можно найти спектральный радиус $\rho=\lambda_1$ как предел последовательности ρ_n и соответствующий ему собственный вектор $\bar{\omega}$ как предел последовательности векторов $\bar{\omega}^{(n)}$.

3.2. Метод вращений Якоби для симметричной матрицы

Рассмотрим задачу нахождения всех собственных чисел и векторов для вещественной симметричной матрицы порядка n. Будем применять к ней преобразование подобия, не изменяющее спектра (собственных чисел). Выберем наибольший по модулю элемент a_{km} матрицы, лежащий выше главной диагонали, и преобразуем матрицу так, чтобы он стал нулём. Это преобразование представляет собой поворот двухмерной плоскости, проходящей через k-ю и m-ю оси координат на специально подобранный угол φ. Преобразование осуществляется перемножением трёх матриц.

Преобразование *подобия* вида $A_1 = H^{-1} * A * H$ не меняет собственные числа, т.е. они для матриц A и A_1 совпадают.

Остальные элементы тоже как-то изменятся. Опять выберем наибольший по модулю элемент матрицы, не лежащий на главной диагонали и опять преобразуем (повернём) матрицу так, чтобы он стал нулём. Там, где до поворота стоял ноль, его может уже и не быть, но максимальные внедиагональные элементы будут по модулю быстро уменьшаться. Через некоторое число преобразований (поворотов-вращений) все внедиагональные элементы будут равняться почти нулю, тогда стоящие на главной диагонали числа и будут собственными числами исходной матрицы. Собственные векторы получают, перемножив все матрицы поворотов. Вычисленная таким образом матрица будет иметь своими столбцами собственные векторы.

Решение:

Код:

Файл Stepennoy.java

```
package first;

abstract public class Stepennoy {
    //функция перемножения матрицы на вектор
    private double[] multMatrixOnVector(double[][] A, double[] v) {
        int n = A.length;
        double[] res = new double[n];
        for(int i = 0;i<n;++i){
            double sum = 0;

            for(int j = 0;j<n;++j){
                sum += A[i][j] * v[j];
            }
            res[i] = sum;
        }
        return res;
    }
    //функция получения длинны вектора
    private double len(double[] v){
        double sum = 0;
        for (double value : v) {
            sum += value * value;
        }
        return Math.sqrt(sum);
    }
    //функция нормализации вектора
    private double[] normalize(double[] v){
        double[] res = v.clone();
        double l = len(v);
        for(int i = 0;i<res.length;++i){
            res[i] /= l;
        }
        return res;
    }
    //скалярное умножение
    private double scalar(double[] a, double[] b){
        int n = a.length;
        double res = 0;
        for(int i = 0;i<n;++i){
            res += (a[i] * b[i]);
        }
        return res;
    }
    //исполняющая функция, передается размер матрицы и сама матрица
    public void exec(int n, double[][] A){
        double[] w = new double[n];
        for(int i = 0;i<n;++i){
            w[i] = 1/Math.sqrt(n);
        }
        double ro;
        int iteration = 1;
        //выполняем шаг алгоритма
        while(endCondition()){
            double[] v = multMatrixOnVector(A, w);
            ro = scalar(v,w);
            w = normalize(v);
            printRes(iteration, w,v, ro);
```

```
            ++iteration;
        }
    }
    void printRes(int iteration, double[] w, double[] v, double ro){
        System.out.print("["+iteration+"]: \tro : "+ ro + '\n');
        System.out.print("v : ");
        for (double item : v) {
            System.out.printf("%.9f  ", item);
        }
        System.out.println();
        System.out.print("w : ");
        for (double value : w) {
            System.out.printf("%.9f  ", value);
        }
        System.out.println();
        System.out.println();
    }
    abstract boolean endCondition();
}
```

Файл Jakoby.java

```
package first;

public class Jakoby {
    private static double[][] multiplication(double[][] a, double[][] b){
        int n = a.length;
        double[][] res = new double[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    res[i][j] += a[i][k] * b[k][j];
                }
            }
        }
        return res;
    }

    private static double[][] unitMatrix(){
        double[][] matrix = new double[3][3];
        for(int i = 0;i<3;++i) {
            for (int j = 0; j < 3; ++j)
                matrix[i][j] = 0;
            matrix[i][i] = 1;
        }
        return matrix;
    }
    private static void normalize(double[][]A){
         A[1][0]/=A[0][0];
         A[2][0]/=A[0][0];
         A[0][0]=1;
         A[0][1]/=A[1][1];
         A[2][1]/=A[1][1];
         A[1][1]=1;
         A[0][2]/=A[2][2];
         A[1][2]/=A[2][2];
         A[2][2]=1;
    }
    private static  double[][] getRotationMatrix(double phi, Element e){
        double[][]matrix = unitMatrix();
        int k = e.k, m = e.m;
        matrix[k][m] = -Math.sin(phi);
```

```
        matrix[m][k] = -matrix[k][m];
        matrix[k][k] = matrix[m][m] = Math.cos(phi);
        return matrix;
    }
    static class Element{
        double value;
        int k,m;
    }
    static double[][] copyMatrix(double[][] A){
        double[][] matrix = new double[3][3];
        for(int i = 0;i<3;++i){
            System.arraycopy(A[i], 0, matrix[i], 0, 3);
        }
        return matrix;
    }
    static void transpose(double[][] m){
        for(int i = 0;i<3;++i){
            for(int j = i;j<3;++j){
                double x = m[i][j];
                m[i][j] = m[j][i];
                m[j][i] = x;
            }
        }
    }
    static Element findMaxElementUpperMainDiagonal(double[][] A){
        int n  = A.length;
        Element max = new Element();
        max.value = A[0][1];
        max.k = 0;
        max.m = 1;
        for(int i = 0 ; i < n;++i){
            for(int j = i+1;j<n;++j){
                if(Math.abs(max.value) < Math.abs(A[i][j])){
                    max.value = A[i][j];
                    max.k = i;
                    max.m = j;
                }
            }
        }
        return max;
    }
    public static  void exec(double[][] A){
        double[][] V = unitMatrix();
        for(int i = 0;i<7;++i){
            Element maxEl = findMaxElementUpperMainDiagonal(A);
            double phi;
            int k = maxEl.k, m = maxEl.m;
            if(A[k][k]-A[m][m] == 0){
                phi = A[k][m] > 0 ? Math.PI/4 : -Math.PI/4;
            } else{
                phi = 0.5 * Math.atan((2.0f*A[k][m])/(A[k][k]-A[m][m]));
            }
            System.out.println("Index: " + i);
            System.out.println("Angle (phi): " + phi);
            System.out.println();
            double[][] H = getRotationMatrix(phi, maxEl);
            System.out.println("H matrix: ");
            printMatrix(H);
            double[][] transposeH = copyMatrix(H);
            System.out.println("H^(-1) matrix: ");
            printMatrix(transposeH);
            transpose(transposeH);
            A = multiplication(multiplication(transposeH,A),H);
            System.out.printf("A%d matrix:\n", i+1);
```

```
            printMatrix(A);
            V = multiplication(V, H);
            System.out.println('\n');
        }
        System.out.println("V matrix:");
        printMatrix(V);
        printEigenValues(A[0][0], A[1][1], A[2][2]);
        normalize(V);
        System.out.println();
        printEigenVectors(V);

    }
    static void printEigenValues(double a, double b, double c){
        System.out.printf("lb1: %f\t lb2: %f\tlb3: %f\n", a,b,c);
    }
    static void printEigenVectors(double[][] A){
        System.out.printf("lb1: %f %f %f\n", A[0][0],A[1][0],A[2][0]);
        System.out.printf("lb2: %f %f %f\n", A[0][1],A[1][1],A[2][1]);
        System.out.printf("lb3: %f %f %f\n", A[0][2],A[1][2],A[2][2]);
    }
    static void printMatrix(double[][] matrix){
        for(int i = 0;i<3;++i){
            for(int j = 0;j<3;++j){
                System.out.printf("%.9f ", matrix[i][j]);
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

Файл Main.java

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        //Степенной метод
        int n3 = 3;
        double[][] aMatrix3 = {
                {-17, 4, -4},
                {4, -14, -8},
                {-4, -8, -4}
        };
        Stepennoy taskStepennoy = new Stepennoy() {
            //задаем условие окончания метода
            int iterations = 10;
            @Override
            boolean endCondition() {
                if(iterations > 0){
                    --iterations;
                    return true;
                }else {
                    return false;
                }
            }
        };
        System.out.println("Степенной метод");
        taskStepennoy.exec(n3, aMatrix3);

        double[][] Matrix4 = {
                {-17, 4, -4},
```

```
                {4, -14, -8},
                {-4, -8, -4}
        };
        //Метод Якоби
        System.out.println("Метод Якоби");
        Jakoby.exec(Matrix4);
    }
}
```

Вывод консоли:

```
Степенной метод
[1]:      ro : -17.000000000000004
v : -9,814954576  -10,392304845  -9,237604307
w : -0,576685502  -0,610608178  -0,542762825
[2]:      ro : -17.04142692750288
v : 9,532272118  10,583875092  9,362658735
w : 0,559231942  0,620926569  0,549281195
[3]:      ro : -17.05760433458359
v : -9,220361516  -10,850293759  -9,401465098
w : -0,540385093  -0,635911834  -0,550999175
[4]:      ro : -17.07915329056655
v : 8,846895944  11,149218704  9,452831743
w : 0,517788669  0,652538376  0,553252711
[5]:      ro : -17.10845576487088
v : -8,405264711  -11,490404277  -9,504472527
w : -0,491030029  -0,671261851  -0,555245026
[6]:      ro : -17.148055979132405
v : 7,883443193  11,875505998  9,555195024
w : 0,459398383  0,692031150  0,556817756
[7]:      ro : -17.201109683352026
v : -7,268918941  -12,305384610  -9,601113755
w : -0,422182909  -0,714703674  -0,557638099
[8]:      ro : -17.271365491023907
v : 6,548847156  12,778224592  9,636913426
w : 0,378703948  0,738933737  0,557279331
```

```
[9]:      ro : -17.362981562698188
v : -5,711349491  -13,288491169  -9,655403009
w : -0,328417336  -0,764122537  -0,555210592
[10]:     ro : -17.48008621002664
v : 4,747446925  13,825730917  9,647492010
w : 0,271055670  0,789380652  0,550823937

Метод Якоби
Index: 0
Angle (phi): 0.5060985057256671
H matrix:
1,000000000 0,000000000 0,000000000
0,000000000 0,874642481 -0,484768532
0,000000000 0,484768532 0,874642481
H^(-1) matrix:
1,000000000 0,000000000 0,000000000
0,000000000 0,874642481 -0,484768532
0,000000000 0,484768532 0,874642481
A1 matrix:
-17,000000000 1,559495795 -5,437644055
1,559495795 -18,433981132 -0,000000000
-5,437644055 -0,000000000 0,433981132

Index: 1
Angle (phi): 0.2788673589243429
H matrix:
0,961367834 0,000000000 -0,275266941
0,000000000 1,000000000 0,000000000
0,275266941 0,000000000 0,961367834
H^(-1) matrix:
0,961367834 0,000000000 -0,275266941
0,000000000 1,000000000 0,000000000
```

```
0,275266941 0,000000000 0,961367834
A2 matrix:
-18,556952075 1,499249094 0,000000000
1,499249094 -18,433981132 -0,429277637
-0,000000000 -0,429277637 1,990933207

Index: 2
Angle (phi): -0.7649042255014488
H matrix:
0,721448682 0,692467905 0,000000000
-0,692467905 0,721448682 0,000000000
0,000000000 0,000000000 1,000000000
H^(-1) matrix:
0,721448682 0,692467905 0,000000000
-0,692467905 0,721448682 0,000000000
0,000000000 0,000000000 1,000000000
A3 matrix:
-19,995975954 0,000000000 0,297260986
0,000000000 -16,994957253 -0,309701785
0,297260986 -0,309701785 1,990933207

Index: 3
Angle (phi): 0.016306423888400062
H matrix:
1,000000000 0,000000000 0,000000000
0,000000000 0,999867053 -0,016305701
0,000000000 0,016305701 0,999867053
H^(-1) matrix:
1,000000000 0,000000000 0,000000000
0,000000000 0,999867053 -0,016305701
0,000000000 0,016305701 0,999867053
A4 matrix:
```

-19,995975954 0,004847049 0,297221466

0,004847049 -17,000007830 0,000000000

0,297221466 0,000000000 1,995983783

Index: 4

Angle (phi): -0.013511715905714634

H matrix:

0,999908718 0,000000000 0,013511305

0,000000000 1,000000000 0,000000000

-0,013511305 0,000000000 0,999908718

H^(-1) matrix:

0,999908718 0,000000000 0,013511305

0,000000000 1,000000000 0,000000000

-0,013511305 0,000000000 0,999908718

A5 matrix:

-19,999992170 0,004846606 0,000000000

0,004846606 -17,000007830 0,000065490

-0,000000000 0,000065490 2,000000000

Index: 5

Angle (phi): -0.0016155382713632508

H matrix:

0,999998695 0,001615538 0,000000000

-0,001615538 0,999998695 0,000000000

0,000000000 0,000000000 1,000000000

H^(-1) matrix:

0,999998695 0,001615538 0,000000000

-0,001615538 0,999998695 0,000000000

0,000000000 0,000000000 1,000000000

A6 matrix:

-20,000000000 0,000000000 -0,000000106

0,000000000 -17,000000000 0,000065490

```
-0,000000106 0,000065490 2,000000000


Index: 6
Angle (phi): -3.4468351864325384E-6
H matrix:
1,000000000 0,000000000 0,000000000
0,000000000 1,000000000 0,000003447
0,000000000 -0,000003447 1,000000000
H^(-1) matrix:
1,000000000 0,000000000 0,000000000
0,000000000 1,000000000 0,000003447
0,000000000 -0,000003447 1,000000000
A7 matrix:
-20,000000000 0,000000000 -0,000000106
0,000000000 -17,000000000 -0,000000000
-0,000000106 0,000000000 2,000000000
V matrix:
0,696310625 0,662266179 -0,276685782
-0,696310621 0,529812943 -0,484200128
-0,174077660 0,529812943 0,830057356


lb1: -20,000000      lb2: -17,000000    lb3: 2,000000


lb1: 1,000000 -1,000000 -0,250000
lb2: 1,250000 1,000000 1,000000
lb3: -0,333333 -0,583333 1,00000
Process finished with exit code 0
```

Ответ:

Спектральный радиус $\rho = -17.48008621002$

Спектральный вектор $\omega^T = (0{,}271055670, 0{,}789380652, 0{,}550823937)$

Собственные значения: $\lambda_1 = -20, \lambda_2 = -17, \lambda_3 = 2$

Собственные вектора:

$\bar{v}_1{}^T = (1, -1, -0.25), \bar{v}_2{}^T = (1.25,1,1), \bar{v}_3{}^T = (-0{,}333333, -0{,}583333,1)$