МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»

КАФЕДРА ИНЖЕНЕРНОЙ КИБЕРНЕТИКИ

ГРУППА БПМ-16-2

ЛАБОРАТОРНАЯ РАБОТА 1

ПО КУРСУ: ЧИСЛЕННЫЕ МЕТОДЫ

СТУДЕНТ МАЛЫНКОВСКИЙ О.В.

ПРЕПОДАВАТЕЛЬ ГОПЕНГАУЗ В.И.

2018г.

ВАРИАНТ 10

Задание 3

Решить СЛАУ модифицированным методом Гаусса с поиском максимального по модулю ведущего элемента шага в оставшейся (не треугольной) части матрицы и методом Зейделя
В **методе Гаусса** за (n-1) шаг прямого проводим исходную матрицу к треугольному виду:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)}$$

$$b_{i}^{(k)} = b_{i}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} b_{k}^{(k-1)}$$

k – номер очередного шага, k меняется от 1 до (n-1), индексы i,j ={k+1,…,n}

Далее осуществляется обратный ход (k = {n, n-1, …, 2, 1})

$$x_k = \frac{1}{a_{kj}^{(k-1)}} (b_k^{(k-1)} - \sum_{j=k+1}^{n} a_{kj}^{(k-1)} x_j)$$

Этот метод можно модифицировать: так как на диагонали могут быть элементы равные или очень близкие к нулю, то при прямом ходе может произойти деление на ноль. Чтобы избежать этой ситуации, мы на каждом шаге будем находить максимальный по модулю элемент в оставшейся (не треугольной) матрице и перемещать его на место текущего диагонального элемента, обменивая между собой соответствующие строки и столбцы. При этом обмен строк нужно запомнить, так как это повлияет на обратный ход, иначе корни перепутаются местами.

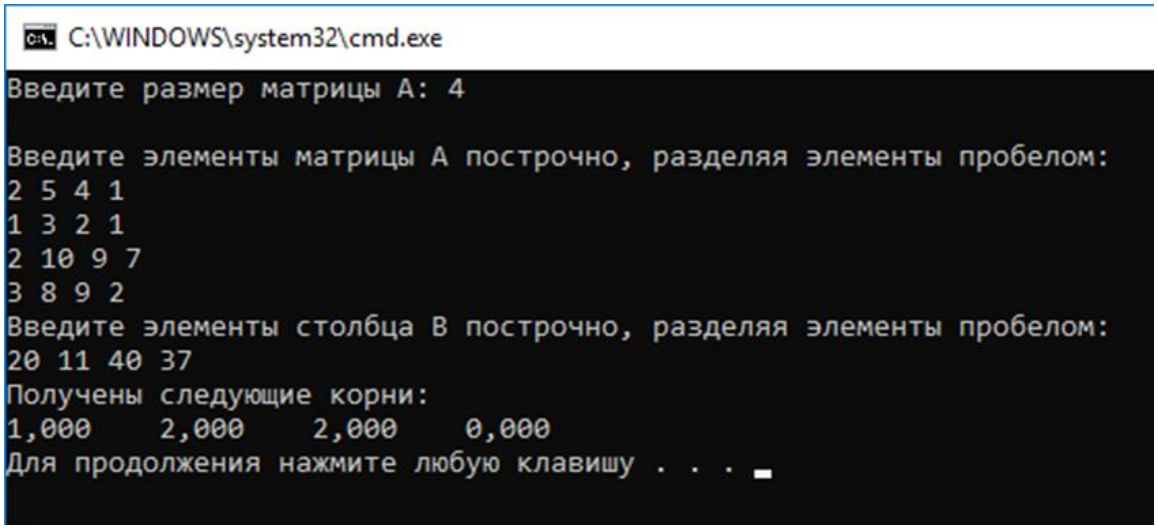
Код:

```
using System;

namespace Gausse_method
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Введите размер матрицы A: ");
            int size = int.Parse(Console.ReadLine());
            Console.WriteLine();
            double[,] A = new double[size, size];
            double[] B = new double[size];
            Console.WriteLine("Введите элементы матрицы A построчно, разделяя элементы
пробелом: ");
            for (int i = 0; i < A.GetLength(0); i++)
            {
                string Matrix = Console.ReadLine();
                string[] massiveMatrix = Matrix.Split(new Char[] { ' ' });
                for (int j = 0; j < massiveMatrix.Length; j++)
                {
                    A[i, j] = double.Parse(massiveMatrix[j]);
                }
            }
            Console.WriteLine("Введите элементы столбца B построчно, разделяя элементы
пробелом: ");
            string enterString = Console.ReadLine();
            string[] massiveString = enterString.Split(new Char[] { ' ' });
```

```
            for (int i = 0; i < B.GetLength(0); i++)
            {
                B[i] = double.Parse(massiveString[i]);
            }
            int[] Memory = new int[size];
            for (int i = 0; i < size; i++)
                Memory[i] = i;//В массиве Memory будем хранить какой столбец какому корню соотвествуеет, что понадибиться
            //при перестовках столбцов в момента поиска максимального элемента в текущей нетреугольной части матрицы
            double[] X = new double[size];
            for (int i = 0; i < size; i++)
                X[i] = 0;
            double t = 0;
            bool q = false;
            for (int k  =0; k < size-1; k++)
            {
                //поиск наибольшего элмента в оставшейся нетреугольной части матрицы
                double max = A[k, k];
                int index_i = k, index_j = k;
                for (int r = k; r < size; r++)
                {
                    for (int w = k; w < size; w++)
                    {
                        if (A[r,w] > max)
                        {
                            max = A[r, w];
                            index_i = r;
                            index_j = w;
                        }
                    }
                }
                if (max < 1e-15)
                {
                    Console.WriteLine("Однозначного решения нет");
                    q = true;
                    break;
                }
                //обмен строк и столбцов если найден такой элемент
                double temp = 0;
                if ((index_i != k) || (index_j != k))
                {
                    //обмен строк
                    if (index_i != k)
                    {
                        for (int j = k; j < size; j++)
                        {
                            temp = A[k, j];
                            A[k, j] = A[index_i, j];
                            A[index_i, j] = temp;
                        }
                        temp = B[k];
                        B[k] = B[index_i];
                        B[index_i] = temp;
                    }

                    //обмен столбцов
                    if (index_j != k)
                    {
                        for (int i = 0; i < size; i++)
                        {
                            temp = A[i, k];
                            A[i, k] = A[i, index_j];
                            A[i, index_j] = temp;
                        }
```

```
                    int u = Memory[k];
                    Memory[k] = Memory[index_j];
                    Memory[index_j] = u;
                }

            }
            //прямой ход метода
            for (int i = k + 1 ; i < size; i++)
            {
                t = A[i, k] / A[k, k];
                B[i] = B[i] - t * B[k];
                for (int j = k  ; j < size; j++)
                {
                    A[i, j] = A[i, j] - t * A[k, j];
                }
            }
        }
        X[size - 1] = B[size - 1] / A[size - 1, size - 1];
        double summa = 0;
        for (int k = size - 2; k >= 0; k--)
        {

            for (int r = k + 1; r < size; r++ )
            {
                summa += A[k, r] * X[r];

            }
            //обратный ход
            X[k] = (B[k] - summa)/A[k,k];
            summa = 0;
        }
        //Расставляем корни по местам
        if (!q)
        {
            double[] Ans = new double[size];
            for (int i = 0; i < size; i++)
                Ans[Memory[i]] = X[i];
            for (int i = 0; i < size; i++)
                Console.Write("{0}    ", Ans[i]);
            Console.WriteLine();
        }
    }

}
}
```

```
C:\WINDOWS\system32\cmd.exe
Введите размер матрицы A: 3

Введите элементы матрицы A построчно, разделяя элементы пробелом:
3 2 -5
2 -1 3
1 2 -1
Введите элементы столбца B построчно, разделяя элементы пробелом:
-1 13 9
Получены следующие корни:
3    5    4
Для продолжения нажмите любую клавишу . . .
```



Метод Зейделя

Метод Зейделя можно рассматривать как модификацию метода Якоби. Основная идея состоит в том, что при вычислении очередного (k + 1)-ого приближения к неизвестному xi при i > 1 используют уже найденные (k + 1)-е приближения к неизвестным $x^{(1)},\ldots x^{(i-1)}$, а не k-е приближение, как в методе Якоби.

На (k + 1)-ой итерации компоненты приближения $x^{(k+1)}$ вычисляются по формулам:

$$x_1^{(k+1)} = \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \cdots - a_{1n}x_n^{(k)})$$

$$x_2^{(k+1)} = \frac{1}{a_{22}}\left(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - \cdots - a_{2n}x_n^{(k)}\right)$$

… … … … … … … … … … … … … … …

$$x_m^{(k+1)} = \frac{1}{a_{mm}}(b_m - a_{m1}x_1^{(k+1)} - a_{m2}x_2^{(k+1)} - \cdots - a_{mm-1}x_{m-1}^{(k+1)} - a_{mm+1}x_{m+1}^{(k)} - \cdots - a_{mn}x_n^{(k)})$$

… … … … … … … … … … … … … … …

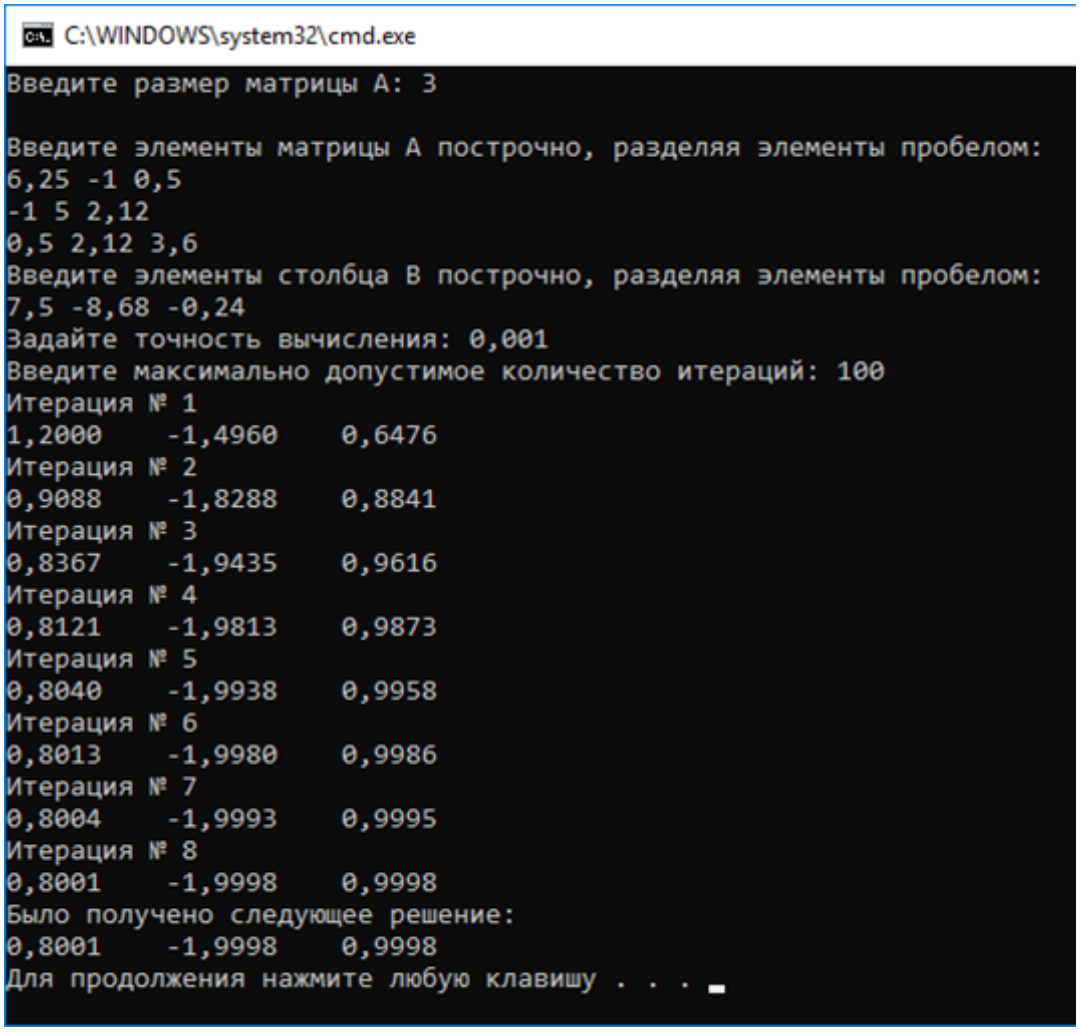
$$x_n^{(k+1)} = \frac{1}{a_{nn}}(b_n - a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} - \cdots - a_{nn-1}x_{n-1}^{(k+1)})$$

И будем так вести итерации до выполнения критерия окончания: $||x^{(k)}-x^{(k-1)}||<\varepsilon$

```
using System;

namespace Seidel_method
{
    class Program
    {
```

```
        static double FindNorma(double[] a, double[] b)
        {
            double temp = 0.0;
            for (int i = 0; i < a.Length; i++)
            {
                temp += a[i] * b[i];
            }
            return Math.Sqrt(temp);
        }

        static void Main(string[] args)
        {
            ……
            // Результирующая матрица
            double[,] matrix;
            double[] X;

            Console.Write("Задайте точность вычисления: ");
            double accuracy = double.Parse(Console.ReadLine());
            Console.Write("Введите максимально допустимое количество итераций: ");
            int iterations = int.Parse(Console.ReadLine());
            int k = 0;
            matrix = new double[A.GetLength(0), A.GetLength(1) + 1];
            X = new double[A.GetLength(0)];
            for(int i = 0; i < matrix.GetLength(0); i++)
            {
                for (int j = 0; j < matrix.GetLength(1) - 1; j++)
                {
                    matrix[i, j] = A[i, j];
                }
                matrix[i, matrix.GetLength(1) - 1] = B[i];
            }// эта матрица объединяет матрицы P и g, если предстваить исходное уравнение
Ax=b в виде x = Px + g

            double[] previousValues = new double[matrix.GetLength(0)];//хранит значения на
предыдущей итерации
            for (int i = 0; i < previousValues.GetLength(0); i++)
            {
                previousValues[i] = 0.0;
            }

            while (true)
            {
                // Введем вектор значений неизвестных на текущем шаге
                double[] currentValues = new double[matrix.GetLength(0)];
                //а где инициализация????????
                for (int i = 0; i < matrix.GetLength(0); i++)
                {
                    currentValues[i] = matrix[i, matrix.GetLength(0)];
                    for (int j = 0; j < matrix.GetLength(0); j++)
                    {
                        if (j < i)
                        {
                            currentValues[i] -= matrix[i, j] * currentValues[j];
                        }
                        if (j > i)
                        {
                            currentValues[i] -= matrix[i, j] * previousValues[j];
                        }
                    }
                    currentValues[i] /= matrix[i, i];
                }
                k++;
                Console.WriteLine("Итерация № {0}", k);
                for (int i = 0; i < currentValues.GetLength(0); i++)
```

```
                    Console.Write("{0:N4}    ", currentValues[i]);// После N меняем цифру, это показывает сколько знаков после запятой выводить
                Console.WriteLine();
                for (int i = 0; i < X.Length; i++)
                    X[i] = currentValues[i] - previousValues[i];
                if ((FindNorma(X,X) <= accuracy)||(k > iterations))
                {
                    previousValues = currentValues;
                    break;
                }
                previousValues = currentValues;
            }
            X = previousValues;
```



Задание 5

Решить СЛАУ методом QR разложения и методом простых итераций

QR-разложение
Требуется решить систему Ax = b, n – размерность матрицы A. Пусть col i-ый столбец матрицы A. Алгоритм состоит из n итераций.
На i-ой итерации зануляются поддиагональные элементы col, для этого матрица A (и вектор b) умножается слева на матрицу отражений $U = E – 2*w*w^T$.

w – вектор, который подбирается так, чтобы col с занулёнными поддиагональными элементами имел напраление заданного единичного вектора (зададим его так чтобы 1 стояла на позиции совпадающей с диагональным элементом)

$$\omega = \frac{y + \sqrt{< y, y >}\, e}{\sqrt{< y + \sqrt{< y, y >}\, e, y + \sqrt{< y, y >}\, e >}}$$

В итоге получается верхнетреугольная матрица, корни ищутся обратным ходом, схожим с обратным ходом метода Гаусса.

```
using System;

namespace QR_decompostion
{
    class Program
    {
        static double ScalatMulti(double[] a, double[] b)
        {
            double temp = 0.0;
            for (int i = 0; i < a.GetLength(0); i++)
                temp += a[i] * b[i];
            return temp;
        }
        static double FindNorma(double[] a)
        {
            return Math.Sqrt(ScalatMulti(a,a)); }

        static double[,] CreateIdentity(int n)//создание единичной матрицы n x n
        static double[,] MultipleMatrix(double[,] A, double[,] B)// перемножение матрицы
        static double[] MultipleMatrix(double[,] A, double[] B)// перемножение матрицы
        static double[,] MultipleNum(double[,] A, double k) // умножение матрицы на число
        static double[] MultipleNum(double[] A, double k)//
        static double[] DivideNum(double[] A, double k) // сложение матриц
        static double[,] addMatrix(double[,] A, double[,] B)
        static double[] addMatrix(double[] A, double[] B)
        // вычитание матриц
        static double[,] substractMatrix(double[,] A, double[,] B)
        {}
        static double[] substractMatrix(double[] A, double[] B)
        {}
        // транспонирование матрицы
        static double[,] transposeMatrix(double[,] matrix)
        {}
        //проверка не являются ли нулями все элементы в столбике под текущим
        static bool CheckZeros(double[] A,int num)
        {
            bool r = true;
            for (int i = num+1; i < A.GetLength(0); i++)
                if (Math.Abs(A[i]) > 1e-9)
                {
                    r = false;
                    break;
                }
            return r;
        }

        static void Main(string[] args)
        {
            // Результирующая матрица
            double[,] U;
            double[,] E = CreateIdentity(A.GetLength(0));
```

```
double[,] temp;
double[] y = new double[A.GetLength(0)];
double[] X = new double[A.GetLength(0)];
for (int i = 0; i < size; i++)
    X[i] = 0;
for (int t = 0; t < A.GetLength(1); t++)
{
    //Высчитываем ортгональную матрицу U и умножаем её на A и B
    for (int i = 0; i < t; i++) y[i] = 0;
    for (int i = t; i < A.GetLength(0); i++) y[i] = A[i, t];
    if (CheckZeros(y,t)) continue;
    double alpha = FindNorma(y);
    Console.WriteLine("альфа {0} ", alpha);
    double[] e = new double[A.GetLength(0)];
    e[t] = 1;
    y = addMatrix(y, MultipleNum(e, alpha));
    y = DivideNum(y,FindNorma(y));
    double[,] w = new double[A.GetLength(0), A.GetLength(0)];
    double[,] wt;
    for (int i = 0; i < A.GetLength(0); i++)
        w[i, t] = y[i];
    wt = transposeMatrix(w);
    temp = MultipleMatrix(w, wt);
    temp = MultipleNum(temp, 2.0);
    U = substractMatrix(E, temp);
    A = MultipleMatrix(U, A);
    B = MultipleMatrix(U, B);
}
….// обратный ход метода Гаусса
}
//вывод результата
```

```
C:\WINDOWS\system32\cmd.exe
Введите размер матрицы A: 4

Введите элементы матрицы A построчно, разделяя элементы пробелом:
1 5 3 2
4 4 3 3
2 1 2 1
0 2 7 1
Введите элементы столбца B построчно, разделяя элементы пробелом:
28 33 14 29
альфа 4,58257569495584
-4,583      -5,019      -4,146      -3,491      -41,0249633643666
0,000       -3,179      -2,120      -0,935      -16,4574312188794
0,000       -2,589      -0,560      -0,967      -10,7287156094397
0,000       2,000       7,000       1,000       29
альфа 4,5617456975947
-4,583      -5,019      -4,146      -3,491      -41,0249633643666
0,000       -4,562      -4,864      -1,639      -30,2724542860976
0,000       0,000       4,578       0,351       15,137628241708
0,000       0,000       3,032       -0,018      9,02125843019457
альфа 5,49057857509068
-4,583      -5,019      -4,146      -3,491      -41,0249633643666
0,000       -4,562      -4,864      -1,639      -30,2724542860976
0,000       0,000       -5,491      -0,283      -17,6020263807788
0,000       0,000       0,000       -0,209      -0,836396878807912
Было получено следующее решение:
1,0000    2,0000    3,0000    4,0000
Для продолжения нажмите любую клавишу . . .
```

Метод простой итерации

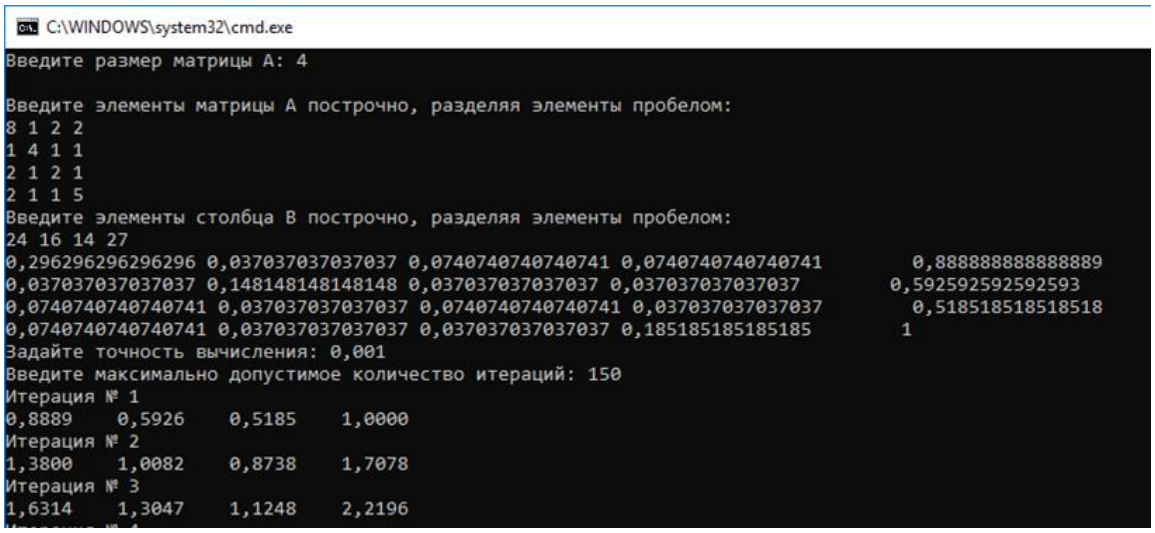
Метод заключается в переходе от системы Ax = b к системе x = Px + g. При этом b = g,

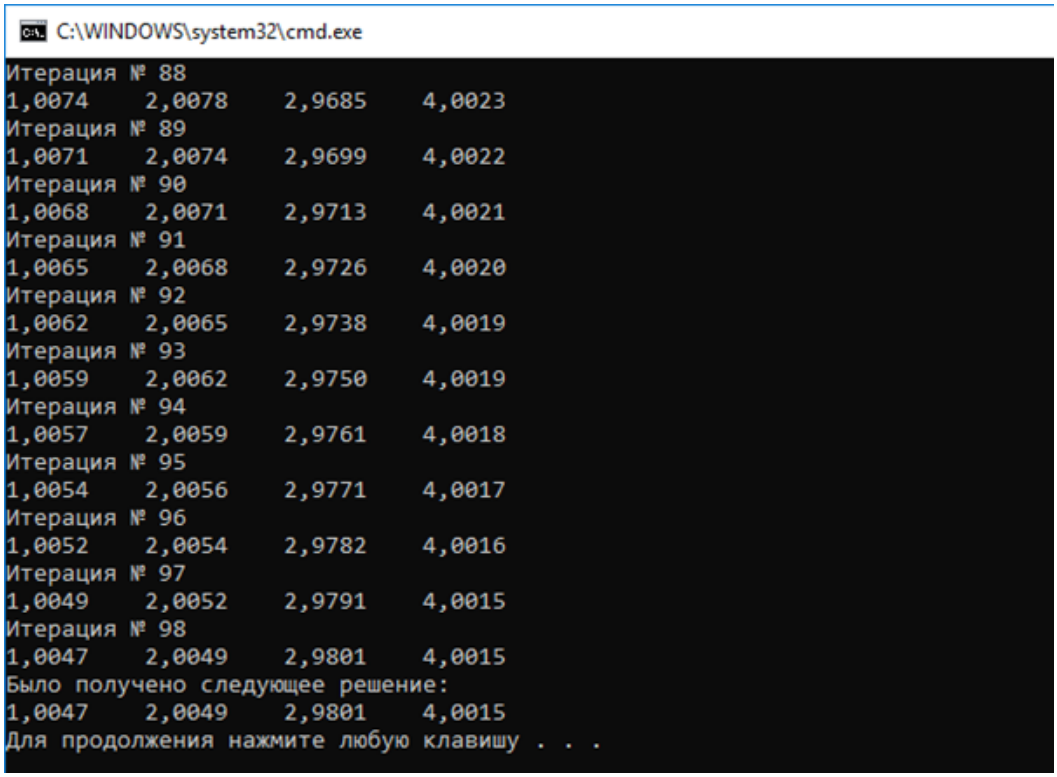
P = $\begin{pmatrix} 1-a_{11} & \cdots & -a_{1n} \\ \vdots & \ddots & \vdots \\ -a_{n1} & \cdots & 1-a_{nn} \end{pmatrix}$

Итерационный процесс происходит в виде $x^{(k+1)} = Px^{(k)} + g$. Критерием остановки можно выбрать, к примеру $\|x^{(k+1)} - x^{(k)}\| < eps$

```
using System;

namespace Simple_iterations
{
    class Program
    {
        static double FindNorma(double[] a, double[] b)
        {
            double temp = 0.0;
            for (int i = 0; i < a.Length; i++)
            {
                temp += a[i] * b[i];
            }
            return Math.Sqrt(temp);
        }

        static void Main(string[] args)
        {
            ……….
            int iterations = int.Parse(Console.ReadLine());
            int k = 0;
            matrix = new double[A.GetLength(0), A.GetLength(1) + 1];
            X = new double[A.GetLength(0)];

            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                for (int j = 0; j < matrix.GetLength(1) - 1; j++)
                {

                    matrix[i, j] = -A[i, j];
                }
                matrix[i, i] = 1.0 - A[i, i];
                matrix[i, matrix.GetLength(1) - 1] = B[i];
            }

            double[] previousValues = new double[matrix.GetLength(0)];//хранит значения на
предыдущей итерации
            for (int i = 0; i < previousValues.GetLength(0); i++)
                previousValues[i] = 0.0;

            while (true)
            {
                // Введем вектор значений неизвестных на текущем шаге
                double[] currentValues = new double[matrix.GetLength(0)];
                for (int i = 0; i < matrix.GetLength(0); i++)
                {
                    currentValues[i] = matrix[i, matrix.GetLength(1)-1];

                    for (int j = 0; j < matrix.GetLength(1)-1; j++)
                    {
                        currentValues[i] += matrix[i, j] * previousValues[j];
                    }
                }
                k++;
```

```
            Console.WriteLine();
            for (int i = 0; i < X.Length; i++)
                X[i] = currentValues[i] - previousValues[i];
            if ((FindNorma(X, X) <= accuracy) || (k > iterations))
            {
                previousValues = currentValues;
                break;
            }
            previousValues = currentValues;
        }
        X = previousValues;
```





Задание 6

Решить СЛАУ методом вращений и вариационным методом (для СЛАУ с симметричной положительно определённой матрицей)

Метод вращений

Решение Ax = b.
Метод по своей сути очень схож с методом Гаусса.
Алгоритм последовательно проходит по столбцам, зануляя поддиагональные элементы, используется преобразование:
a_i = c*a + s*b
a_j = -s*a + c*b

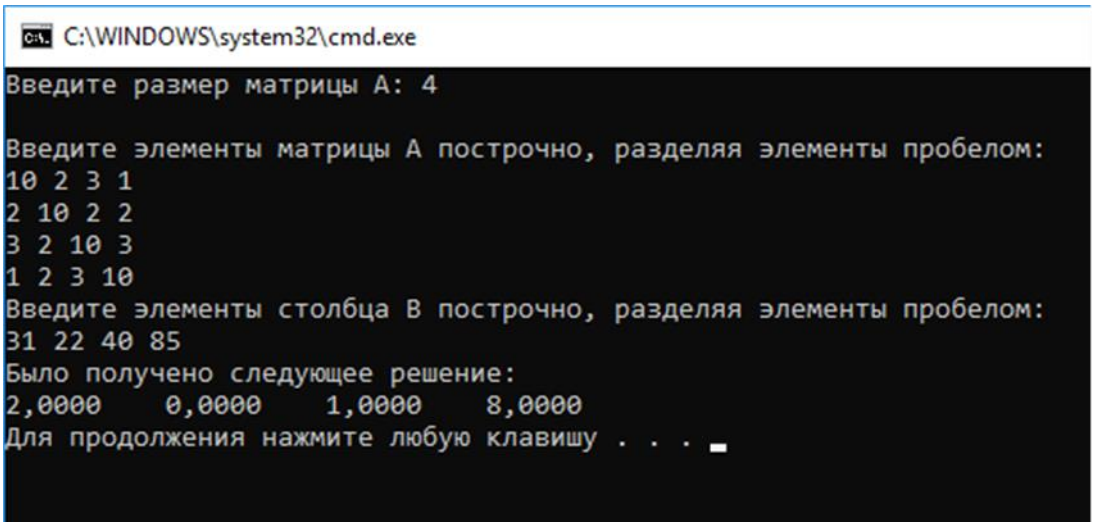
где a_i – i-ая строка матрицы A, a_j – j-ая строка матрицы A (j > i),

$$c = \frac{a_{ii}}{\sqrt{a_{ii}^2+a_{ji}^2}} \qquad s = \frac{a_{ji}}{\sqrt{a_{ii}^2+a_{ji}^2}}$$

Обратный ход подобен обратному ходу метода Гаусса

```
using System;
namespace Rotation_method
{
    class Program
    {
        // проверка на нулевые элементы
        static bool CheckZeros(double num)
        {
            double eps = 1e-9;
            if (Math.Abs(num) < eps)
                return true;
            else
                return false;
        }

        static void Main(string[] args)
        {
            ……….            for (int t = 0; t < A.GetLength(0); t++)
            {
                //защита от деления на ноль
                for (int k = t; k < A.GetLength(0); k++)
                    if (!CheckZeros(A[k,t]))
                    {
                        double temp;
                        for (int i = 0; i < A.GetLength(0); i++)
                        {
                            temp = A[t, i];
                            A[t, i] = A[k, i];
                            A[k, i] = temp;
                            temp = B[k];
                            B[k] = B[t];
                            B[t] = B[k];
                        }
                        break;
                    }

                for (int k = t + 1; k < A.GetLength(0); k++ )
                {
```

```
                double c = A[t, t] / Math.Sqrt(A[t, t] * A[t, t] + A[k, t] * A[k, t]);
                double s = A[k, t] / Math.Sqrt(A[t, t] * A[t, t] + A[k, t] * A[k, t]);
                double[] A1 = new double[A.GetLength(0)];
                double[] A2 = new double[A.GetLength(0)];
                double B1 = B[t]; double B2 = B[k];
                for (int i = 0; i < A.GetLength(0); i++)
                {
                    A1[i] = A[t, i] * c + A[k, i] * s;
                    A2[i] = A[t, i] * (-s) + A[k, i] * c;
                    B1 = B[t] * c + B[k] * s;
                    B2 = B[t] * (-s) + B[k] * c;
                }
                for (int i = 0; i < A.GetLength(0); i++)
                {
                    A[t, i] = A1[i];
                    A[k, i] = A2[i];
                }
                B[t] = B1; B[k] = B2;
            }
        }
        //обратный ход
```



Вариационный метод

Работает для положительно определенных матриц, т.е. для таких, у которых выполняются свойства: $A^T = A$, все собственные значения положительны.

Для матрицы задается функционал:

$\Phi(u) = < Ax, x > -2 < b, x > + C$

Ищется его минимум (можно показать, что минимум $\Phi(u)$ совпадает с x*).

Минимум ищется с помощью градиентного спуска: на каждой итерации к текущему

Приближею прибавляют градиент Ф, умноженный на константу τ так, чтобы неувязка Г (Г = Ax - b) была меньше (по норме), чем текущая.

Задается начальное приближение x_0, осуществляется итерационный переход:

$x^{n+1} = x^n - \tau\Gamma^n$

$$\tau_n = \frac{< A\Gamma_n, \Gamma_n >}{< A\Gamma_n, A\Gamma_n >}$$

Итерации можно прекратить, когда $||x^{(k+1)} - x^{(k)}|| < eps$

```
using System;

namespace Varitional_method
{
    class Program
    {
        static double ScalatMulti(double[] a, double[] b)
        {
            double temp = 0.0;
            for (int i = 0; i < a.GetLength(0); i++)
                temp += a[i] * b[i];
            return temp;
        }
        static double FindNorma(double[] a)
        {
            return Math.Sqrt(ScalatMulti(a, a));
        }

        static double[,] MultipleMatrix(double[,] A, double[,] B)// перемножение матрицы
        {}
        static double[] MultipleMatrix(double[,] A, double[] B)// перемножение матрицы
        {}
        // умножение матрицы на число
        static double[,] MultipleNum(double[,] A, double k)
        {}
        static double[] MultipleNum(double[] A, double k)//
        {}
        // сложение матриц
        static double[,] addMatrix(double[,] A, double[,] B)
        {}
        static double[] addMatrix(double[] A, double[] B)
        {}
        // вычитание матриц
        ……
        static void Main(string[] args)
        {
            // Опущен ввод слау, точности и числа итераций
            int k = 0;
            // Результирующая матрица
            double[] X = new double[A.GetLength(0)];
            for (int i = 0; i < size; i++)
                X[i] = 0;

            double[] previousValues = new double[A.GetLength(0)];//хранит значения на
предыдущей итерации
            double[] discrepancy = new double[A.GetLength(0)];//невязка
            for (int i = 0; i < previousValues.GetLength(0); i++)
            {
                previousValues[i] = 0.0;
            }
            discrepancy = substractMatrix(MultipleMatrix(A, previousValues),B);
            for (int i = 0; i < A.GetLength(0); i++)
                Console.Write("{0}  ", discrepancy[i]);
            Console.WriteLine();
            while (true)
            {
                double[] currentValues = new double[A.GetLength(0)];
                double tau = ScalatMulti(MultipleMatrix(A, discrepancy), discrepancy) /
(FindNorma(MultipleMatrix(A, discrepancy))* FindNorma(MultipleMatrix(A, discrepancy)));
                currentValues = substractMatrix(previousValues, MultipleNum(discrepancy,
tau));
                discrepancy = substractMatrix(MultipleMatrix(A, currentValues), B);
                k++;
```

```
                Console.WriteLine("Итерация № {0}", k);
                for (int i = 0; i < currentValues.GetLength(0); i++)
                    Console.Write("{0:N4}    ", currentValues[i]);// После N меняем цифру,
это показывает сколько знаков после запятой выводить
                Console.WriteLine();
                for (int i = 0; i < X.Length; i++)
                    X[i] = currentValues[i] - previousValues[i];
                if ((FindNorma(X) <= accuracy) || (k > iterations))
                {
                    previousValues = currentValues;
                    break;
                }
                previousValues = currentValues;
            }
            X = previousValues;
```