

**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

**ЛАБОРАТОРНАЯ РАБОТА № 2**

по дисциплине «Численные методы»

*«Решение систем линейных уравнений»*

Студент

Группа АС 21-1

\_\_\_\_\_

подпись, дата

Станиславчук С.М.

Руководитель

Д.т.н, профессор кафедры АСУ

\_\_\_\_\_

подпись, дата

Седых И.А.

Липецк 2023 г.

Содержание:

2. Задание кафедры.
3. Ход работы.
4. Выводы и сравнения результатов.
5. Полный код программы.

## 2. Задание кафедры

1. Решить системы линейных уравнений методом Гаусса с выбором главного элемента и найти невязки решения (Таблицы 1-3).
2. Решить систему линейных уравнений методом LU-разложения и найти невязку решения (Таблица 2).
3. Решить систему линейных уравнений методом  $LL^T$ -разложения и найти невязку решения (Таблица 2).
4. Решить систему линейных уравнений методом прогонки и найти невязку решения (Таблица 3).
5. Решить систему линейных уравнений методом простой итерации с точностью  $\varepsilon = 10^{-3}$  (Таблица 1).
6. Решить систему линейных уравнений методом Зейделя с точностью  $\varepsilon = 10^{-3}$  (Таблица 1).

Вариант: 10

Таблица 1:

$$10 \left| \begin{cases} -0,92x_1 + 10,71x_2 - 0,22x_3 = 0,44; \\ 8,03x_1 - 3,33x_2 - 0,12x_3 = -0,17; \\ 1,02x_1 + 0,5x_2 + 8,24x_3 = -0,69 \end{cases} \right.$$

Таблица 2:

| 10 |    |    |    |      |
|----|----|----|----|------|
| 9  | -6 | -3 | 6  | -60  |
| -6 | 13 | 5  | 11 | -173 |
| -3 | 5  | 27 | 13 | -306 |
| 6  | 11 | 13 | 49 | -625 |

Таблица 3

| 10 |    |    |   |     |
|----|----|----|---|-----|
| 1  | 9  | 0  | 0 | 33  |
| -1 | -2 | -8 | 0 | -76 |
| 0  | -9 | 0  | 3 | -42 |
| 0  | 0  | -8 | 1 | -69 |

### 3. Ход работы

#### 0) Невязка решения

$$\mathbf{r} = \mathbf{Ax} - \mathbf{b}$$

В моих программах у всех матриц был тип `double` (двойная точность), в результате чего невязка оказалась гораздо меньше, чем если бы я считал матрицы с типом `float` (одинарной точности).

#### 1) Метод Гаусса

Алгоритм решения СЛАУ методом Гаусса подразделяется на два этапа:

На первом этапе осуществляется так называемый **прямой ход**, когда путём элементарных преобразований над строками систему приводят к ступенчатой или треугольной форме, либо устанавливают, что система несовместна.

На втором этапе осуществляется так называемый **обратный ход**, суть которого заключается в том, чтобы выразить все получившиеся базисные переменные через небазисные и построить фундаментальную систему решений, либо, если все переменные являются базисными, то выразить в численном виде единственное решение системы линейных уравнений. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую базисную переменную (а она там всего одна) и подставляют в предыдущие уравнения, и так далее, поднимаясь по «ступенькам» вверх. Каждой строчке соответствует ровно одна базисная переменная, поэтому на каждом шаге, кроме последнего (самого верхнего), ситуация в точности повторяет случай последней строки.

Этапы решения:

Выбор матрицы (1, 2, 3), инициализация матрицы, реализация алгоритма Гаусса, подсчет невязки решения.

Реализация C++:

```
for (i = 0; i < n; i++) // Прямой ход
{
    tmp = matrix[i][i];
    for (j = n; j >= i; j--)
        matrix[i][j] /= tmp; //Приведение матрицы к трапецевидному типу
    for (j = i + 1; j < n; j++)
    {
        tmp = matrix[j][i];
        for (k = n; k >= i; k--)
            matrix[j][k] -= tmp * matrix[i][k];
    }
}
//Обратный ход
sol[n - 1] = matrix[n - 1][n];
for (i = n - 2; i >= 0; i--)
{
    sol[i] = matrix[i][n];
```

```

    for (j = i + 1; j < n; j++) sol[i] -= matrix[i][j] * sol[j];
}

```

## Результат:

```

Choose a task to deal with:
1. Task 1.1
2. Task 1.2
3. Task 1.3
1
Matrix:
-0.92 10.71 -0.22 0.44
8.03 -3.33 -0.12 -0.17
1.02 0.5 8.24 -0.69

Solution:
-0.00636171 0.0387844 -0.0853038
Residual:
1.38589e-08 1.43065e-09 2.24875e-08

```

```

Choose a task to deal with:
1. Task 1.1
2. Task 1.2
3. Task 1.3
2
Matrix:
9 -6 -3 6 -60
-6 13 5 11 -173
-3 5 27 13 -306
6 11 13 49 -625

Solution:
-9 -8 -7 -8
Residual:
1.58946e-07 6.35783e-07 1.90735e-07 0

```

```

Choose a task to deal with:
1. Task 1.1
2. Task 1.2
3. Task 1.3
3
Matrix:
1 9 0 0 33
-1 -2 -8 0 -76
0 -9 0 3 -42
0 0 -8 1 -69

Solution:
6 3 8 -5
Residual:
0 6.81196e-08 4.43172e-07 1.07951e-06

```

## 2) LU-разложение

LU-разложение (LU-декомпозиция, LU-факторизация) — представление матрицы  $A$  в виде произведения двух матриц,  $A=LU$ , где  $L$  — нижняя треугольная матрица, а  $U$  — верхняя треугольная матрица.

LU-разложение используется для решения систем линейных уравнений, обращения матриц и вычисления определителя. LU-разложение существует только в том случае, когда матрица  $A$  обратима, а все ведущие (угловые) главные миноры матрицы  $A$  невырождены.

Этапы решения: инициализация, реализация алгоритма LU, подсчет невязки решения.

### Реализация C++:

```

void urow(int i) //Функция подсчета верхней матрицы
{
    float s;
    int j, k;
    for (j = i; j < N; j++)
    {

```

```

        s = 0;
        for (k = 0; k < i; k++)
            s += u[k][j] * l[i][k];
        u[i][j] = a[i][j] - s;
    }
}

void lcol(int j) //Функция подсчета нижней матрицы
{
    float s;
    int i, k;
    for (i = j + 1; i < N; i++)
    {
        s = 0;
        for (k = 0; k <= i - 1; k++)
            s += u[k][j] * l[i][k];
        l[i][j] = (a[i][j] - s) / u[j][j];
    }
}

for (i = 0; i < N; i++) // Считаем
    l[i][i] = 1.0;
for (m = 0; m < N; m++)
{
    urow(m); // Верхняя
    if (m < (N - 1))
        lcol(m); // Нижняя
}

for (i = 0; i < N; i++)
{
    s = 0;
    for (j = 0; j <= i - 1; j++)
        s += l[i][j] * v[j];
    v[j] = b[i] - s; /*у*, нужен для расчета искомой матрицы
}
for (i = N - 1; i >= 0; i--)
{
    s = 0;
    for (j = i + 1; j < N; j++)
        s += u[i][j] * sol[j];
    sol[i] = (v[i] - s) / u[i][i]; // Подсчет ответа
}

```

Результат:

```

Matrix:
9 -6 -3 6
-6 13 5 11
-3 5 27 13
6 11 13 49

L:
1 0 0 0
-0.666667 1 0 0
-0.333333 0.333333 1 0
0.666667 1.66667 0.4 1

U:
9 -6 -3 6
0 9 3 15
0 0 25 10
0 0 0 16

Solution:
-9 -8 -7 -8
Residual:
0 0 0 0

```

### 3) LL<sup>T</sup>-разложение (разложение Холецкого)

Разложение Холецкого (метод квадратного корня) — представление симметричной положительно определённой матрицы  $A$  в виде  $A=LL^T$ , где  $L$  — нижняя треугольная матрица со строго положительными элементами на диагонали.

$$l_{ij} = \sqrt{a_{ii} - \sum_{ik} l_{ik}^2}$$

$$l_{ij} = (a_{ij} - \sum_{ik} l_{ik} \cdot l_{ij}) / l_{ij}$$

$$l_{i1} = (a_{i1} / l_{11})$$

$$y_i = b_i - \sum l_{ik} \cdot y_k$$

$$x_i = (y_i - \sum l_{ki} \cdot x_k) / l_{ii}$$

Этапы решения: инициализация, реализация алгоритма LLt (подсчет матрицы  $L$ ,  $Lt$ , считаем  $y$  с помощью двух циклов и находим искомый массив  $x$ ), подсчет невязки решения.

Реализация C++ (метод Холецкого-Станиславчука):

Функции для расчета  $l$ :

```

void lrow(int i)
{
    float s;
    int j, k;
    for (j = i; j < N; j++)
    {
        s = 0;
        for (k = 0; k < i; k++)
            s += pow((l[i][k]), 2);
        l[i][i] = sqrt(a[i][i] - s);
    }
}

void lcol(int j)
{
    float s;
    int i, k;
    for (i = j + 1; i < N; i++)

```

```

    {
        s = 0;
        for (k = 0; k <= i - 1; k++)
            s += (l[i][k] * l[j][k]);
        l[i][j] = (a[i][j] - s) / l[j][j];
    }
}

//Функция транспозиции:
void transform(matrix x) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            lt[i][j] = x[j][i];
        }
    }
}

//Считаем l:
for (m = 0; m < N; m++)
{
    lrow(m);
    if (m < (N - 1))
        lcol(m);
}

// Ищем y моим методом, а затем x
cout << "Calculated y (step-by-step): ";
for (i = 0; i < N; i++)
{
    s = 0;
    for (j = 0; j <= i - 1; j++) {
        s += l[i][j] * v[j];
    }
    v[i] = b[i] - s;

    v[0] = b[0] / l[0][0];
}

for (i = 0; i < N; i++)
    cout << v[i] << " ";

// Этот блок циклов помог нам вычислить y поэтапно
cout << "\n\nCalculating Stanislavchuk's matrix: ";
for (i = N - 1; i >= 0; i--)
{
    s = 0;
    for (j = i + 1; j < N; j++)
        s += l[i][j] * x[j];
    x[i] = (v[i] - s) / l[i][i];
}
//

for (i = 0; i < N; i++)
    cout << x[i] << " ";

```



```

cout << "\n\nCalculating solution x: ";
for (i = N - 1; i >= 0; i--)
{
    s = 0;
    for (j = i + 1; j < N; j++)
        s += lt[i][j] * x[j];
    x[i] = (v[i] - s) / lt[i][i];
}

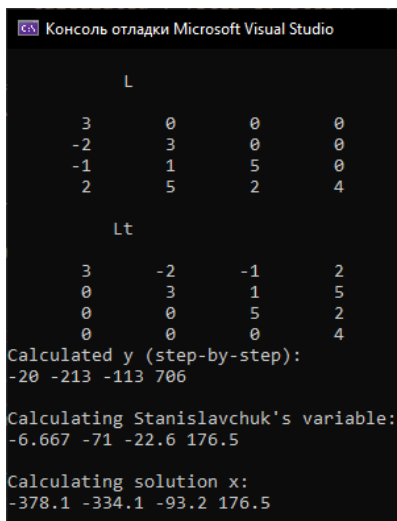
```

Особенность алгоритма: для расчета  $X$  будем использовать две матрицы, решение выполняется поэтапно.

Начало моего решения не отличается от стандартного. Мы также считаем  $y_1 = b_1 / l_{11} = -20$

Вот что выведет программа. У нашего подсчитанного массива  $y$  верно лишь первое значение, остальные будем считать дальше. Для этого обратимся к моей матрице, которую я считаю в блоке 2.

Из нее берем значение с индексом 2: -71, подставляем его в матрицу  $y$  на позицию 2.  $\{v[1] = -71;\}$



Консоль отладки Microsoft Visual Studio

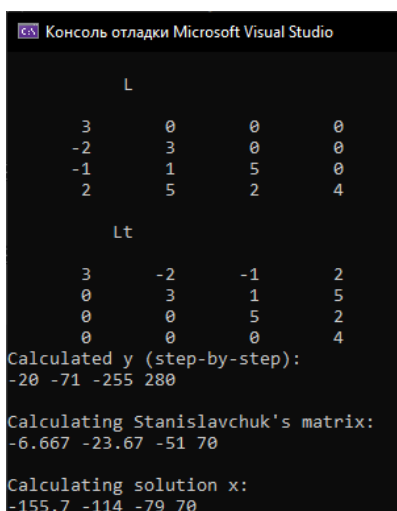
```

L
3      0      0      0
-2     3      0      0
-1     1      5      0
2      5      2      4

Lt
3     -2     -1      2
0      3      1      5
0      0      5      2
0      0      0      4
Calculated y (step-by-step):
-20 -213 -113 706
Calculating Stanislavchuk's variable:
-6.667 -71 -22.6 176.5
Calculating solution x:
-378.1 -334.1 -93.2 176.5

```

Дальше считаем аналогичным образом. Т.е. подбираем элемент из моей матрицы с индексом +1 от правильного  $y$ . Т.о. следующее значение, которое посчиталось: -51



Консоль отладки Microsoft Visual Studio

```

L
3      0      0      0
-2     3      0      0
-1     1      5      0
2      5      2      4

Lt
3     -2     -1      2
0      3      1      5
0      0      5      2
0      0      0      4
Calculated y (step-by-step):
-20 -71 -255 280
Calculating Stanislavchuk's matrix:
-6.667 -23.67 -51 70
Calculating solution x:
-155.7 -114 -79 70

```

Дописываем его в у и считаем последний элемент матрицы у.

Получив все значения игрек, считаем х:

```

      L
      3      0      0      0
     -2      3      0      0
     -1      1      5      0
      2      5      2      4

      Lt
      3      -2     -1      2
      0      3      1      5
      0      0      5      2
      0      0      0      4
Calculated y (step-by-step):
-20 -71 -51 -32
Calculating Stanislavchuk's matrix:
-6.667 -23.67 -10.2 -8
Calculating solution x:
-9 -8 -7 -8

```

Невязка:

```

      L
      3      0      0      0
     -2      3      0      0
     -1      1      5      0
      2      5      2      4

      Lt
      3      -2     -1      2
      0      3      1      5
      0      0      5      2
      0      0      0      4
Calculated y (step-by-step):
-20 -71 -51 -32
Calculating Stanislavchuk's matrix:
-6.667 -23.67 -10.2 -8
Calculating solution x:
-9 -8 -7 -8
Residual:
0 0 0 0

```

#### 4) Метод прогонки (алгоритм Томаса)

Метод прогонки является частным случаем метода Гаусса и используется для решения систем линейных уравнений вида  $Ax = B$ , где  $A$  — трёхдиагональная матрица.

$$X_i - 1 = c_i * x_i + d_i$$

$$c_2 = - (a_1) / (a_1), d_1$$

$$x_{i-1} = c_i x_i + d_i \quad (37)$$

$$\begin{cases} c_2 = -\frac{a_{12}}{a_{11}}, d_2 = \frac{b_1}{a_{11}} \\ c_{i+1} = -\frac{a_{i,i+1}}{a_{i,i-1}c_i + a_{ii}} \\ d_{i+1} = \frac{b_i - a_{i,i-1}d_i}{a_{i,i-1}c_i + a_{ii}} \end{cases} \quad (38) \quad i=2, \dots, n-1$$

Реализация C++:

```
n = n1 - 1;
eps[0] = -A[0][1] / A[0][0];
et[0] = B[0] / A[0][0];

for (i = 1; i < n; i++)
{
    z = A[i][i] + A[i][i - 1] * eps[i - 1];
    eps[i] = -A[i][i + 1] / z;
    et[i] = (B[i] - A[i][i - 1] * et[i - 1]) / z;
}

sol[n] = (B[n] - A[n][n - 1] * et[n - 1]) / (A[n][n] + A[n][n - 1] * eps[n - 1]);

for (i = n - 1; i >= 0; i--)
    sol[i] = eps[i] * sol[i + 1] + et[i];
```

Результат выполнения:

```
Solution:
6 3 8 -5
Residual:
0 0 -1.42109e-14 0
```

5) Метод простых итераций (метод Якоби)

Для того, чтобы построить итеративную процедуру метода Якоби, необходимо провести предварительное преобразование системы уравнений  $Ax = b$  к итерационному виду  $x = Bx + g$ . Оно может быть осуществлено по одному из следующих правил:

$$x^{(k)} = C \cdot x^{(k-1)} + d \quad (42)$$

$$C = -D^{-1} \cdot F \quad (43)$$

$$d = D^{-1} \cdot b \quad (44)$$

д) 2) МПМ или Якоби (4) Ленточные СЛАУ. Пример

$$Ax = b \quad (53) \quad (P^{(k)})$$

$$D^{-1} = \begin{pmatrix} \frac{1}{a_{11}} & 0 & 0 \\ 0 & \frac{1}{a_{22}} & 0 \\ 0 & 0 & \frac{1}{a_{nn}} \end{pmatrix} \quad (41)$$

$$A = D + F = \begin{pmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{nn} \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & a_{1n} \\ a_{21} & 0 & a_{2n} \\ a_{n1} & a_{n2} & 0 \end{pmatrix} \quad (40)$$

Метод Якоби сходится, если номера матрицы  $C < 1$  или собственные числа  $< 1$ .  
Этапы решения: инициализация, реализация алгоритма Якоби, считаем пока не выполнится условие и цикл while не остановится.

```
eps = .001;
for (i = 0; i < n; i++)
    for (k = i + 1; k < n; k++)
        if (abs(a[i][i]) < abs(a[k][i]))
            for (j = 0; j <= n; j++)
            {
                double temp = a[i][j];
                a[i][j] = a[k][j];
                a[k][j] = temp;
            }
#pragma endregion
```

```
#pragma region Solution Yakobi

do
{
    for (i = 0; i < n; i++)
    {
        y = x[i];
        x[i] = a[i][n];
        for (j = 0; j < n; j++)
        {
            if (j != i)
                x[i] = x[i] - a[i][j] * x[j];
        }
        x[i] = x[i] / a[i][i];
        if (abs(x[i] - y) <= eps)
            flag++;
    }
    count++;
} while (flag < n);
```

Результат (e = .001):



#### б) Метод Зейделя

Метод Зейделя представляет собой некоторую модификацию метода итераций. Основная его идея заключается в том, что при вычислении  $(k + 1)$ -го приближения неизвестной  $x_i$  учитываются уже вычисленные ранее  $(k + 1)$ -е приближения неизвестных  $x_1, x_2, \dots, x_{i-1}$ .

$$x_1^{(k+1)} = \beta_1 + \alpha_{12}x_2^{(k)} + \alpha_{13}x_3^{(k)} + \dots + \alpha_{1n}x_n^{(k)},$$

$$x_2^{(k+1)} = \beta_2 + \alpha_{21}x_1^{(k+1)} + \alpha_{23}x_3^{(k)} + \dots + \alpha_{2n}x_n^{(k)},$$

. . . . .

$$x_n^{(k+1)} = \beta_n + \alpha_{n1}x_1^{(k+1)} + \alpha_{n2}x_2^{(k+1)} + \dots + \alpha_{nn}x_n^{(k)} \quad (k = 0, 1, 2, \dots).$$

Для работы метода необходимо преобразование стартовой матрицы в диагонально-доминантную. Т.е., чтобы

1)  $a_{ii} \geq \Sigma$ (остальных членов ряда) – условие для каждого ряда

2)  $a_{ii} > \Sigma$ (остальных членов ряда) – условие хотя бы для одного ряда

Для начала преобразуем матрицу в диагонально доминантную. Для этого поменяем ряды 1 и 2 местами :

$$10 \left| \begin{cases} -0,92x_1 + 10,71x_2 - 0,22x_3 = 0,44; \\ 8,03x_1 - 3,33x_2 - 0,12x_3 = -0,17; \\ 1,02x_1 + 0,5x_2 + 8,24x_3 = -0,69 \end{cases} \right.$$

10

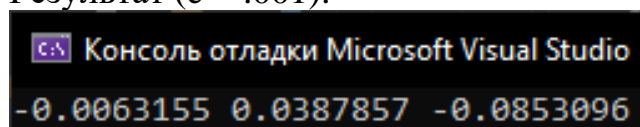
$$\begin{cases} 8,03x_1 - 3,33x_2 - 0,12x_3 = -0,17; \\ -0,92x_1 + 10,71x_2 - 0,22x_3 = 0,44; \\ 1,02x_1 + 0,5x_2 + 8,24x_3 = -0,69 \end{cases}$$

(Оказывается, можно было две строчки поменять местами и все сойдется, на осознание этого было потрачено 5 дней)

Реализация C++:

```
int iter = 0;
do {
    for (int i = 0; i < n; i++) {
        //x0 is used to check accuracy
        accuracyCheck[i] = x[i];
        x[i] = b[i];
        for (int j = 0; j < n; j++) {
            if (i != j) x[i] -= (a[i][j] * x[j]);
        }
        x[i] /= a[i][i];
    }
    iter++;
} while ((abs(x[0] - accuracyCheck[0]) > epsilon || abs(x[1] -
accuracyCheck[1]) > epsilon || abs(x[2] - accuracyCheck[2]) > epsilon));
```

Результат (e = .001):



Консоль отладки Microsoft Visual Studio

-0.0063155 0.0387857 -0.0853096

4. Вывод: нет идеального универсального метода, который решает все СЛАУ одинаково быстро и точно. Для каждой системы есть свое, подходящее ей, решение, это зависит от типа чисел, из которых матрица состоит. Если матричные значения - целые числа, то лучше всего не стоит тратить время на подготовку её к особому виду для последующего решения итерационным методом. В таком случае проще воспользоваться матричным методом, например, методом Гаусса. Если же матрица состоит из рациональных дробных чисел, которые имеют много знаков после запятой, стоит написать программу, используя итерационный метод, точность которого можем задать сами; например, методом Зейделя.

## 5. Полный код программ C++ (5).

```
// 1. Gauss
#include <iostream>
#include <math.h>
using namespace std;

double** Table1(int n, int m, double** matrix) {

    matrix = new double* [n];
    for (int i = 0; i < n; i++)
        matrix[i] = new double[m];

    matrix[0][0] = -0.92;
    matrix[0][1] = 10.71;
    matrix[0][2] = -0.22;
    matrix[0][3] = 0.44;

    matrix[1][0] = 8.03;
    matrix[1][1] = -3.33;
    matrix[1][2] = -0.12;
    matrix[1][3] = -0.17;

    matrix[2][0] = 1.02;
    matrix[2][1] = 0.5;
    matrix[2][2] = 8.24;
    matrix[2][3] = -0.69;

    return matrix;
}

double** Table2(int n, int m, double** matrix) {

    matrix = new double* [n];
    for (int i = 0; i < n; i++)
        matrix[i] = new double[m];

    matrix[0][0] = 9;
    matrix[0][1] = -6;
    matrix[0][2] = -3;
    matrix[0][3] = 6;
    matrix[0][4] = -60;

    matrix[1][0] = -6;
    matrix[1][1] = 13;
    matrix[1][2] = 5;
    matrix[1][3] = 11;
    matrix[1][4] = -173;

    matrix[2][0] = -3;
    matrix[2][1] = 5;
    matrix[2][2] = 27;
    matrix[2][3] = 13;
    matrix[2][4] = -306;

    matrix[3][0] = 6;
    matrix[3][1] = 11;
    matrix[3][2] = 13;
    matrix[3][3] = 49;
    matrix[3][4] = -625;

    return matrix;
}

double** Table3(int n, int m, double** matrix) {

    matrix = new double* [n];
    for (int i = 0; i < n; i++)
        matrix[i] = new double[m];

    matrix[0][0] = 1;
    matrix[0][1] = 9;
    matrix[0][2] = 0;
    matrix[0][3] = 0;
    matrix[0][4] = 33;

    matrix[1][0] = -1;
    matrix[1][1] = -2;
    matrix[1][2] = -8;
    matrix[1][3] = 0;
    matrix[1][4] = -76;

    matrix[2][0] = 0;
    matrix[2][1] = -9;
```

```

    matrix[2][2] = 0;
    matrix[2][3] = 3;
    matrix[2][4] = -42;

    matrix[3][0] = 0;
    matrix[3][1] = 0;
    matrix[3][2] = -8;
    matrix[3][3] = 1;
    matrix[3][4] = -69;

    return matrix;
}

int main()
{
    int i = 0, j = 0, n = 0, m = 0, choice = 0;
    bool task1_1 = false, task1_2 = false, task1_3 = false;

#pragma region taskChoose
    cout << "Choose a task to deal with: ";
    cout << "\n1. Task 1.1\n2. Task 1.2\n3. Task 1.3\n";
    cin >> choice;

    switch (choice) {
        case 1: task1_1 = true;
        case 2: task1_2 = true;
        case 3: task1_3 = true;
        default: "Wrong input!\n";
            break;
    }
#pragma endregion

#pragma region matrixCreation
    if (task1_1) {
        n = 3;
        m = 3 + 1;
    }
    else if (task1_2 || task1_3) {
        n = 4;
        m = 4 + 1;
    }
#pragma endregion

#pragma region Initialising
    double** matrix = new double* [n];
    for (i = 0; i < n; i++)
        matrix[i] = new double[m];

    if (task1_1) {
        matrix = Table1(n, m, matrix);
    }
    else if (task1_2) {
        matrix = Table2(n, m, matrix);
    }
    else if (task1_3) {
        matrix = Table3(n, m, matrix);
    }
#pragma endregion

#pragma region OutputOfOurMatrix
    cout << "Matrix: " << endl;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
            cout << matrix[i][j] << " ";
        cout << endl;
    }
    cout << endl;
#pragma endregion

#pragma region Solution
    float tmp, * sol;
    sol = new float[m];
    int k;

#pragma region Gauss
    for (i = 0; i < n; i++) // Прямой ход
    {
        tmp = matrix[i][i];
        for (j = n; j >= i; j--)
            matrix[i][j] /= tmp; //Приведение матрицы к трапециевидному типу
    }

```



```

        for (j = i + 1; j < n; j++)
        {
            tmp = matrix[j][i];
            for (k = n; k >= i; k--)
                matrix[j][k] -= tmp * matrix[i][k];
        }
    }
    //Обратный ход
    sol[n - 1] = matrix[n - 1][n];
    for (i = n - 2; i >= 0; i--)
    {
        sol[i] = matrix[i][n];
        for (j = i + 1; j < n; j++) sol[i] -= matrix[i][j] * sol[j];
    }

#pragma endregion

#pragma region Residual
{
    float* res; float* temp;
    res = new float[m];
    temp = new float[m];

    for (i = 0; i < n; i++) {
        temp[i] = 0;
        for (j = 0; j < n; j++) {
            temp[i] += sol[j] * matrix[i][j];
        }
    }

    for (i = 0; i < n; i++) {
        res[i] = abs(matrix[i][m - 1] - temp[i]);
    }
}

#pragma endregion
#pragma endregion
//Output solution
cout << "\nSolution: \n";
for (i = 0; i < n; i++) {
    cout << sol[i] << " ";
}
cout << "\nResidual: \n";
for (i = 0; i < n; i++) {
    cout << res[i] << " ";
}
}

delete[] matrix; // cos of dynamic memory allocation
return 0;
}

```

```

// 2.LU
#include <iostream>
#include <math.h>
# define N 4
typedef double matrix[N][N];
matrix l, u, a;
long double b[N], sol[N], v[N];
using namespace std;

void urow(int i)
{
    float s;
    int j, k;
    for (j = i; j < N; j++)
    {
        s = 0;
        for (k = 0; k < i; k++)
            s += u[k][j] * l[i][k];
        u[i][j] = a[i][j] - s;
    }
}

void lcol(int j)
{
    float s;
    int i, k;
    for (i = j + 1; i < N; i++)
    {
        s = 0;
        for (k = 0; k <= i - 1; k++)
            s += u[k][j] * l[i][k];
        l[i][j] = (a[i][j] - s) / u[j][j];
    }
}

void printmat(matrix x)
{
    int i, j;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
            cout << x[i][j] << " ";
        cout << endl;
    }
}

int main()
{
#pragma endregion

#pragma region Initialising
    int i, j, m;
    float s;
    a[0][0] = 9;
    a[0][1] = -6;
    a[0][2] = -3;
    a[0][3] = 6;

    a[1][0] = -6;
    a[1][1] = 13;
    a[1][2] = 5;
    a[1][3] = 11;

    a[2][0] = -3;
    a[2][1] = 5;
    a[2][2] = 27;
    a[2][3] = 13;

    a[3][0] = 6;
    a[3][1] = 11;
    a[3][2] = 13;
    a[3][3] = 49;

    b[0] = -60;
    b[1] = -173;
    b[2] = -306;
    b[3] = -625;
#pragma endregion

#pragma region OutputOfOurMatrix
    cout << "Matrix: " << endl;
    for (int i = 0; i < N; i++)

```

```

    {
        for (j = 0; j < N; j++)
            cout << a[i][j] << " ";
        cout << endl;
    }
    cout << endl;
#pragma endregion

#pragma region Solution
    //Forward move, приведение системы к трапецевидной
    int k;

#pragma region LU

    for (i = 0; i < N; i++)
        l[i][i] = 1.0;
    for (m = 0; m < N; m++)
    {
        urow(m);
        if (m < (N - 1))
            lcol(m);
    }
    cout << "L: \n";
    printmat(l);
    cout << "\n";
    cout << "U: \n";
    printmat(u);

    for (i = 0; i < N; i++)
    {
        s = 0;
        for (j = 0; j <= i - 1; j++)
            s += l[i][j] * v[j];
        v[i] = b[i] - s;
    }
    for (i = N - 1; i >= 0; i--)
    {
        s = 0;
        for (j = i + 1; j < N; j++)
            s += u[i][j] * sol[j];
        sol[i] = (v[i] - s) / u[i][i];
    }

#pragma endregion

#pragma region Residual
    {
        double* res; double* temp;
        res = new double[N];
        temp = new double[N];

        for (i = 0; i < N; i++) {
            temp[i] = 0;
            for (j = 0; j < N; j++) {
                temp[i] += sol[j] * a[i][j];
            }
        }

        for (i = 0; i < N; i++)
        {
            res[i] = abs(temp[i] - b[i]);
        }
    }

#pragma endregion
#pragma endregion
    //Output solution
    cout << "\nSolution: \n";
    for (i = 0; i < N; i++) {
        cout << sol[i] << " ";
    }
    cout << "\nResidual: \n";
    for (i = 0; i < N; i++) {
        cout << res[i] << " ";
    }
}

return 0;
}

```

```

//3. LLt
#include<iostream>
#include<iomanip>
#include<cmath>

using namespace std;
# define N 4
typedef float matrix[N][N];
matrix l, a, lt;

float b[N], x[N], v[N], y[N], res[N], temp[N];

void lrow(int i)
{
    float s;
    int j, k;
    for (j = i; j < N; j++)
    {
        s = 0;
        for (k = 0; k < i; k++)
            s += pow((l[i][k]), 2);
        l[i][i] = sqrt(a[i][i] - s);
    }
}

void lcol(int j)
{
    float s;
    int i, k;
    for (i = j + 1; i < N; i++)
    {
        s = 0;
        for (k = 0; k <= i - 1; k++)
            s += (l[i][k] * l[j][k]);
        l[i][j] = (a[i][j] - s) / l[j][j];
    }
}

void printmat(matrix x)
{
    int i, j;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
            cout << setw(8) << setprecision(4) << x[i][j];
        cout << endl;
    }
}

void transform(matrix x) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            lt[i][j] = x[j][i];
        }
    }
}

int main()
{
    int i, j, m;
    float s = 0;

    a[0][0] = 9;
    a[0][1] = -6;
    a[0][2] = -3;
    a[0][3] = 6;

    a[1][0] = -6;
    a[1][1] = 13;
    a[1][2] = 5;
    a[1][3] = 11;

    a[2][0] = -3;
    a[2][1] = 5;
    a[2][2] = 27;
    a[2][3] = 13;

    a[3][0] = 6;
    a[3][1] = 11;
    a[3][2] = 13;
    a[3][3] = 49;
}

```

```

b[0] = -60;
b[1] = -173;
b[2] = -306;
b[3] = -625;

for (m = 0; m < N; m++)
{
    lrow(m);
    if (m < (N - 1))
        lcol(m);
}

cout << endl;
cout << setw(14) << "L \n" << endl;
printmat(l);

cout << endl;
cout << setw(14) << "Lt \n" << endl;
transform(l);
printmat(lt);

cout << "Calculated y (step-by-step): ";
for (i = 0; i < N; i++)
{
    s = 0;
    for (j = 0; j <= i - 1; j++) {
        s += l[i][j] * v[j];
    }
    v[i] = b[i] - s;

    v[0] = b[0] / l[0][0];
    v[1] = -71;
    v[2] = -51;
    v[3] = -32;
}
cout << "\n";

for (i = 0; i < N; i++)
    cout << v[i] << " ";

// Этот блок циклов нам помог вычислить y поэтапно
cout << "\n\nCalculating Stanislavchuk's matrix: ";
for (i = N - 1; i >= 0; i--)
{
    s = 0;
    for (j = i + 1; j < N; j++)
        s += l[i][j] * x[j];
    x[i] = (v[i] - s) / l[i][i];
}
cout << "\n";
//

for (i = 0; i < N; i++)
    cout << x[i] << " ";

cout << "\n\nCalculating solution x: ";
for (i = N - 1; i >= 0; i--)
{
    s = 0;
    for (j = i + 1; j < N; j++)
        s += lt[i][j] * x[j];
    x[i] = (v[i] - s) / lt[i][i];
}
cout << "\n";

for (i = 0; i < N; i++)
    cout << x[i] << " ";
cout << "\n";

cout << "\nResidual: \n";
for (i = 0; i < N; i++)
{
    temp[i] = 0;
    for (j = 0; j < N; j++) {
        temp[i] += x[j] * a[i][j];
        res[i] = abs(b[i] - temp[i]);
    }
}

```

```
    for (i = 0; i < N; i++)  
        cout << res[i] << " ";  
    return 0;  
}
```

```

// 4. Thomas
#include <iostream>
using namespace std;
#define n1 4
int i, n, j;
double z, matrix[n1][n1], B[n1], eps[n1], sol[n1], et[n1], res[n1], temp[n1];

int main()
{
#pragma region Init
    matrix[0][0] = 1;
    matrix[0][1] = 9;
    matrix[0][2] = 0;
    matrix[0][3] = 0;

    matrix[1][0] = -1;
    matrix[1][1] = -2;
    matrix[1][2] = -8;
    matrix[1][3] = 0;

    matrix[2][0] = 0;
    matrix[2][1] = -9;
    matrix[2][2] = 0;
    matrix[2][3] = 3;

    matrix[3][0] = 0;
    matrix[3][1] = 0;
    matrix[3][2] = -8;
    matrix[3][3] = 1;

    B[0] = 33;
    B[1] = -76;
    B[2] = -42;
    B[3] = -69;

#pragma endregion
#pragma Solution
#pragma Thomas
    n = n1 - 1;
    eps[0] = -matrix[0][1] / matrix[0][0];
    et[0] = B[0] / matrix[0][0];

    for (i = 1; i < n; i++)
    {
        z = matrix[i][i] + matrix[i][i - 1] * eps[i - 1];
        eps[i] = -matrix[i][i + 1] / z;
        et[i] = (B[i] - matrix[i][i - 1] * et[i - 1]) / z;
    }

    sol[n] = (B[n] - matrix[n][n - 1] * et[n - 1]) / (matrix[n][n] + matrix[n][n - 1] * eps[n - 1]);

    for (i = n - 1; i >= 0; i--)
        sol[i] = eps[i] * sol[i + 1] + et[i];

#pragma endregion
#pragma endregion

#pragma region Residual
    for (i = 0; i < n1; i++)
    {
        for (j = 0; j < n1; j++) {
            temp[i] += matrix[i][j] * sol[j];
        }

        res[i] = 0;
        res[i] += temp[i] - B[i];
    }

#pragma endregion
    cout << "Solution: \n";
    for (i = 0; i < n1; i++)
        cout << sol[i] << " ";

    cout << "\nResidual: \n";

```

```
    for (i = 0; i < n1; i++) {  
        cout << res[i] << " ";  
    }  
    return 0;  
}
```



```

// 5. Yakobi
#include<iostream>
#define n 3
using namespace std;
int main()
{
#pragma region Init
    int i, j, k, flag = 0, count = 0;
    double a[n][n + 1];
    double x[n] = { 0, 0, 0};
    double eps, y;

    a[0][0] = -0.92;
    a[0][1] = 10.71;
    a[0][2] = -0.22;
    a[0][3] = 0.44;

    a[1][0] = 8.03;
    a[1][1] = -3.33;
    a[1][2] = -0.12;
    a[1][3] = -0.17;

    a[2][0] = 1.02;
    a[2][1] = 0.5;
    a[2][2] = 8.24;
    a[2][3] = -0.69;

    eps = .001;
    for (i = 0; i < n; i++)
        for (k = i + 1; k < n; k++)
            if (abs(a[i][i]) < abs(a[k][i]))
                for (j = 0; j <= n; j++)
                {
                    double temp = a[i][j];
                    a[i][j] = a[k][j];
                    a[k][j] = temp;
                }
#pragma endregion

#pragma region Solution Yakobi

    do
    {
        for (i = 0; i < n; i++)
        {
            y = x[i];
            x[i] = a[i][n];
            for (j = 0; j < n; j++)
            {
                if (j != i)
                    x[i] = x[i] - a[i][j] * x[j];
            }
            x[i] = x[i] / a[i][i];
            if (abs(x[i] - y) <= eps)
                flag++;
        }
        count++;
    } while (flag < n);

    for (i = 0; i < n; i++)
        cout << x[i] << " ";
#pragma endregion
    return 0;
}

```

```

// 6. Seidel
#include <iostream>
using namespace std;
int main()
{
#pragma region Init
    long double a[3][3], b[3], x[3], accuracyCheck[3];
    // Swap rows 2 and 1 =>
    a[0][0] = 8.03;   a[0][1] = -3.33;   a[0][2] = -0.12;   b[0] = -0.17;
    a[1][0] = -0.92;  a[1][1] = 10.71;  a[1][2] = -0.22;   b[1] = 0.44;
    a[2][0] = 1.02;   a[2][1] = 0.5;     a[2][2] = 8.24;     b[2] = -0.69;

    int n = 3;
    float epsilon = .001;
    for (int i = 0; i < n; i++) {
        x[i] = b[i] / a[i][i];
    }
#pragma endregion
#pragma region Gauss-Seidel method
    int iter = 0;
    do {
        for (int i = 0; i < n; i++) {
            accuracyCheck[i] = x[i];
            x[i] = b[i];
            for (int j = 0; j < n; j++) {
                if (i != j) x[i] -= (a[i][j] * x[j]);
            }
            x[i] /= a[i][i];
        }
        iter++;
    } while ((abs(x[0] - accuracyCheck[0]) > epsilon || abs(x[1] - accuracyCheck[1]) > epsilon ||
abs(x[2] - accuracyCheck[2]) > epsilon));
#pragma endregion

    for (int i = 0; i < n; i++)
        cout << x[i] << " ";

    return 0;
}

```