МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 2

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

Доцент		С.Л. Козенко
должность, уч. Степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №2

РЕШЕНИЕ СЛАУ

по курсу: ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4136		Бобрович Н. С.
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

Цель работы:

а) освоение основных методов решения систем линейных алгебраических уравнений (СЛАУ);

б) совершенствование навыков по алгоритмизации и программированию вычислительных задач.

Задание:

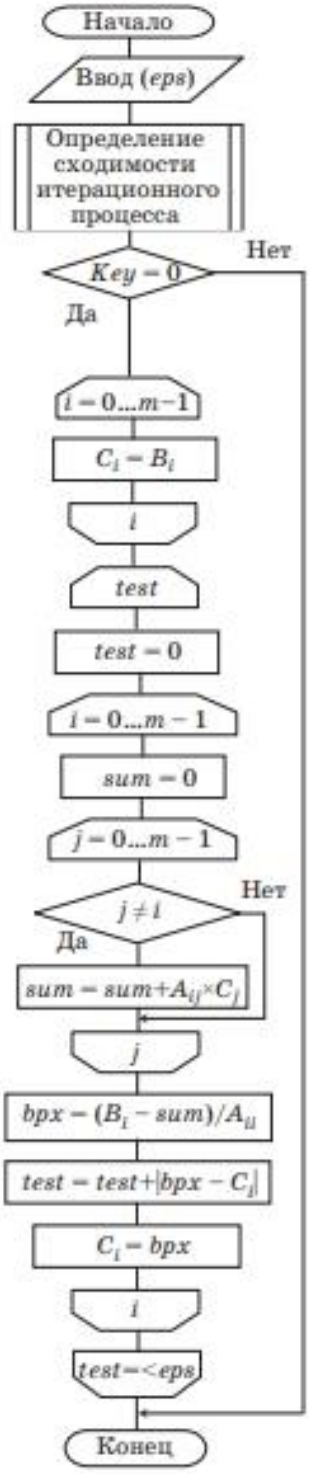
3. Решить систему линейных уравнений $AX = B$ методом Зейделя, где

$$A = \begin{pmatrix} 21 & 4 & 2 & 5 \\ 4 & 24 & 5 & 3 \\ 1 & 2 & 26 & 8 \\ 3 & 7 & 3 & 22 \end{pmatrix}, \; B = \begin{pmatrix} 3 \\ 8 \\ 1 \\ 7 \end{pmatrix}$$

Математическая часть:

Метод Зейделя относится к итерационным методам решения систем линейных уравнений, обеспечивающим хорошую сходимость итерационного процесса поиска корней системы. Пусть задана система (3.2). Прежде всего необходимо задать значения начальных (нулевых) приближений корней X1 (0), X2 (0),..., Xm(0). Если отсутствует какая–нибудь информация об этих значениях, то их можно принять равными свободным членам системы уравнений (3.2) или даже принять равными нулю. Выбранные таким образом нулевые приближения корней подставим в первое уравнение системы (3.2) и получим первое приближение корня X1 X1 (1) = β1 + α12 X2 (0) + α13 X3 (0) + … + α1 m Xm(0). Используя во втором уравнении системы (3.2) найденное первое приближение корня X1 и нулевые приближения остальных корней, получим первое приближение корня X2 X2 (1) = β2 + α21 X1 (1) + α23 X2 (0) + … + α2 m Xm(0). Повторяя эту процедуру последовательно для всех уравнений системы (3.2), получим в итоге первое приближение корня Xm(1) = βm + αm 1 X1 (1) + αm2 X2 (1) + … + αm m–1 Xm–1(1). Используя первые приближение корня системы, можно аналогичным образом найти вторые приближения X1 (2) = β1 + α12 X2 (1) + α13 X3 (1) + … + α1 mCm(1) X2 (2) = β1 + α21 X1 (2) + α23 X3 (1) + … + α2 mXm(1) ..... Cm(2) = βm + αm 1 X1 (2) + αm 2 X2 (2) + … + αm m–1 Xm–1(2). Затем, используя

вторые приближения, можно вычислить третьи и т.д. Итерационный процесс решения системы линейных уравнений методом Зейделя сходится к единственному решению при любом выборе начальных приближений искомых корней, если выполняется одно из условий (3.5). Погрешность решения системы уравнений методом Зейделя принято оценивать по формулам (3.7) — (3.9). 3.4. Схема алгоритма решения системы линейных уравнений методом Зейделя Схема алгоритма решений системы уравнений (1.7) методом Зейделя представлена на рис. 3.3. В схему алгоритма определения сходимости итерационного процесса (рис.3.1), которая используется в методе Зейделя, необходимо внести следующие изменения: заменить блок вычисления значения переменной Flag = Flag + 1 на Flag = 1; условие Flag = m заменить на условие Flag = 1. Эта замена объясняется Замечанием, приведенным на стр. 29, и связано с тем, что для метода Зейделя достаточно, чтобы неравенство (3.6) выполнялось хотя бы для одной строки матрицы A.



Ввод значения точности вычислений *eps*

Выполнение действий в соответствии с алгоритмом (рис. 3.1)

Проверка условия сходимости интерационного процесса

Начало цикла копирования элементов массива **B** в массив **C**

Начало цикла с постусловием

Обнуление накопителя суммы модулей разностей k-го и $(k + 1)$-го приближений (3.9)

Обнуление накопителя суммы слагаемых левых частей i-го уравнения системы

Вычисление суммы слагаемых левой части i-го уравнения системы без i-го слагаемого

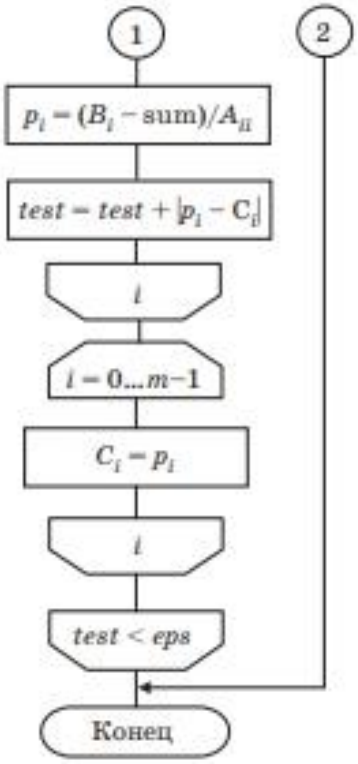
Вычисление очередного приближения неизвестного C_i

Добавление в накопитель *test* очередного слагаемого

Запись нового приближения в массив **C**

Конец цикла с постусловием
(выход из цикла при достижении заданной точности *eps*)

Рис. 3.3. Схема алгоритма решения системы уравнений методом Зейделя



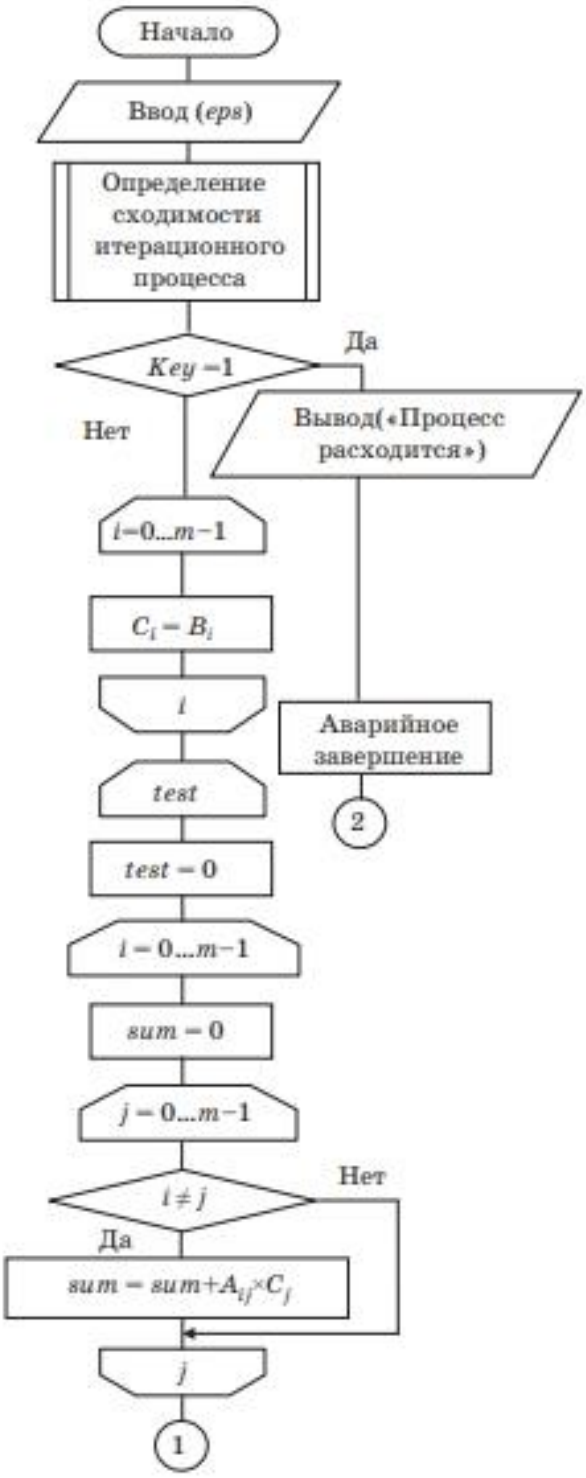
Вычисление очередного приближения

Добавление в накопитель *test* очередного слагаемого

Копирование элементов $(k + 1)$ -го приближения в соответствующие элементы массива **C**

Окончание цикла с постусловием. Выход из цикла с постусловием при достижении заданной точности *eps*

Рис.3.2. Схема алгоритма решения системы линейных уравнений методом последовательных приближений (окончание)



Ввод точности вычислений *eps*

Выполнение действий в соответствии со схемой алгоритма (рис. 3.1)

Проверка условия сходимости итерационного процесса

Начало цикла копирования элементов массива **B** в массив **C**

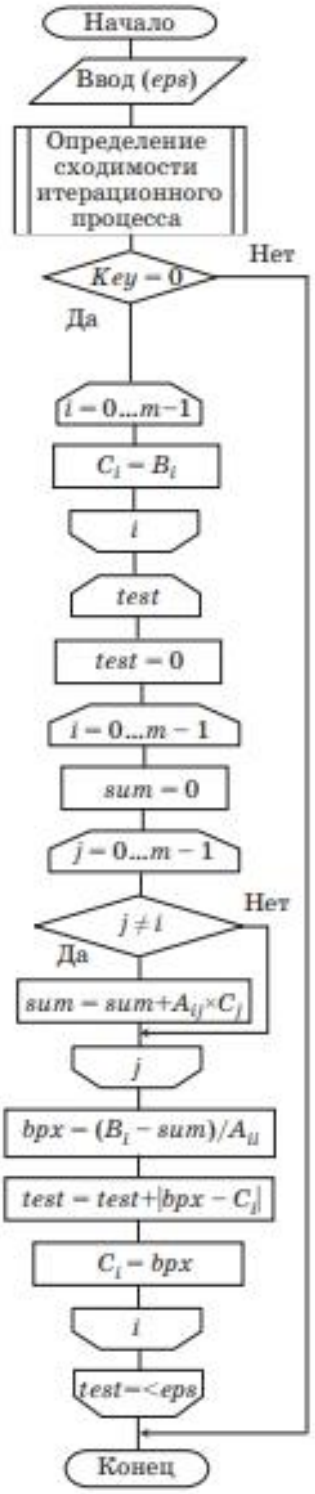
Начало цикла с постусловием

Обнуление накопителя суммы модулей разностей k-го и $(k+1)$-го приближений

Обнуление накопителя суммы слагаемых левых частей i-го уравнения системы

Вычисление суммы слагаемых левой части i-го уравнения системы без i-го слагаемого

Рис.3.2. Схема алгоритма решения системы линейных уравнений методом последовательных приближений (начало)



Ввод значения точности вычислений *eps*

Выполнение действий в соответствии с алгоритмом (рис. 3.1)

Проверка условия сходимости интерационного процесса

Начало цикла копирования элементов массива **B** в массив **C**

Начало цикла с постусловием

Обнуление накопителя суммы модулей разностей k-го и $(k + 1)$-го приближений (3.9)

Обнуление накопителя суммы слагаемых левых частей i-го уравнения системы

Вычисление суммы слагаемых левой части i-го уравнения системы без i-го слагаемого

Вычисление очередного приближения неизвестного C_i

Добавление в накопитель *test* очередного слагаемого

Запись нового приближения в массив **C**

Конец цикла с постусловием (выход из цикла при достижении заданной точности *eps*)

Рис. 3.3. Схема алгоритма решения системы уравнений методом Зейделя

Аналитические расчеты:

Для оценки погрешности вычисляем коэффициент α:

$$max[\sum|\alpha_{ij}|] = 0.136 + 0.318 + 0.136 = 0.591 < 1$$

max[|x⁴,x⁵|] = ρ(x⁴, x⁵) = |0.222 - 0.222| = 0.00027

Вычисляем погрешность:

$$\rho(x,x^5) \le \frac{\alpha}{1-\alpha}\rho(x^4,x^5) = \frac{0.591}{1-0.591}0.00027 \le 0.00039$$

N=3

x_1=0.143 - 0.301*0.19 - (-0.0473)*0.0952 - 0.224*0.238=0.0367

x_2=0.333 - 0.0367*0.167 - (-0.0473)*0.208 - 0.224*0.125=0.309

x_3=0.0385 - 0.0367*0.0385 - 0.309*0.0769 - 0.224*0.308=-0.0557

x_4=0.318 - 0.0367*0.136 - 0.309*0.318 - (-0.0557)*0.136=0.222

Остальные расчеты сведем в таблицу.

N	x_1	x_2	x_3	x_4	e_1	e_2	e_3	e_4
0	0	0	0	0				
1	0.143	0.31	0.00916	0.199	0.143	0.31	0.00916	0.199
2	0.0357	0.301	-0.0473	0.224	-0.107	-0.00891	0.0381	0.0251
3	0.0367	0.309	-0.0557	0.222	0.00108	0.00843	0.00843	-0.00168
4	0.0363	0.311	-0.0553	0.222	-0.000403	0.00203	-0.000376	-0.000643
5	0.0361	0.311	-0.0551	0.222	-0.00027	4.7E-5	-0.000205	-6.0E-6

x_1=0.143 - (0.19x_2+0.0952x_3+0.24x_4)

x_2=0.333 - (0.17x_1+0.21x_3+0.13x_4)

x_3=0.0385 - (0.0385x_1+0.0769x_2+0.31x_4)

x_4=0.318 - (0.14x_1+0.32x_2+0.14x_3)

Покажем вычисления на примере нескольких итераций.

N=1

x_1=0.143 - 0*0.19 - 0*0.0952 - 0*0.238=0.143

x_2=0.333 - 0.143*0.167 - 0*0.208 - 0*0.125=0.31

x_3=0.0385 - 0.143*0.0385 - 0.31*0.0769 - 0*0.308=0.00916

x_4=0.318 - 0.143*0.136 - 0.31*0.318 - 0.00916*0.136=0.199

N=2

x_1=0.143 - 0.31*0.19 - 0.00916*0.0952 - 0.199*0.238=0.0357

x_2=0.333 - 0.0357*0.167 - 0.00916*0.208 - 0.199*0.125=0.301

x_3=0.0385 - 0.0357*0.0385 - 0.301*0.0769 - 0.199*0.308=-0.0473

x_4=0.318 - 0.0357*0.136 - 0.301*0.318 - (-0.0473)*0.136=0.224

$x^{k+1}_1=\beta_1 - \sum\alpha_{1j}x^k_j$

$x^{k+1}_2=\beta_2 - \alpha_{21}x^{k+1}_1 - \sum\alpha_{2j}x^k_j$

$x^{k+1}_i=\beta_i - \sum\alpha_{ij}x^{k+1}_1 - \sum\alpha_{2j}x^k_j$

Прежде чем применять метод, необходимо переставить строки исходной системы таким образом, чтобы на диагонали стояли наибольшие по модулю коэффициенты матрицы.

21	4	2	5
4	24	5	3
1	2	26	8
3	7	3	22

Приведем к виду:

Метод Зейделя.

Метод Зейделя представляет собой модификацию метода простой итераций.

Имеем СЛАУ: A x =b (1)

Предполагая, что $a_{ii} \neq 0$ разрешим новое уравнение системы (1) относительно x_1, второе – относительно x_2,..., *n-ое* уравнение – относительно x_n. В результате получим:

$x_1=\beta_1 - \alpha_{12}x_2 - \alpha_{13}x_3 - ... - \alpha_{1n}x_n$

$x_2=\beta_2 - \alpha_{21}x_1 - \alpha_{23}x_3 - ... - \alpha_{2n}x_n$

$x_n=\beta_n - \alpha_{n1}x_n - \alpha_{n3}x_3 - ... - \alpha_{nn-1}x_{n-1}$

где $\beta_i=b_i/a_{ii}$; $\alpha_{ij}=a_{ij}/a_{ii}$ при $i \neq j$; $\alpha_{ii}=0$

Известно начальное приближение: $x^0=(x^0_1, x^0_2, ..., x^0_n)$.

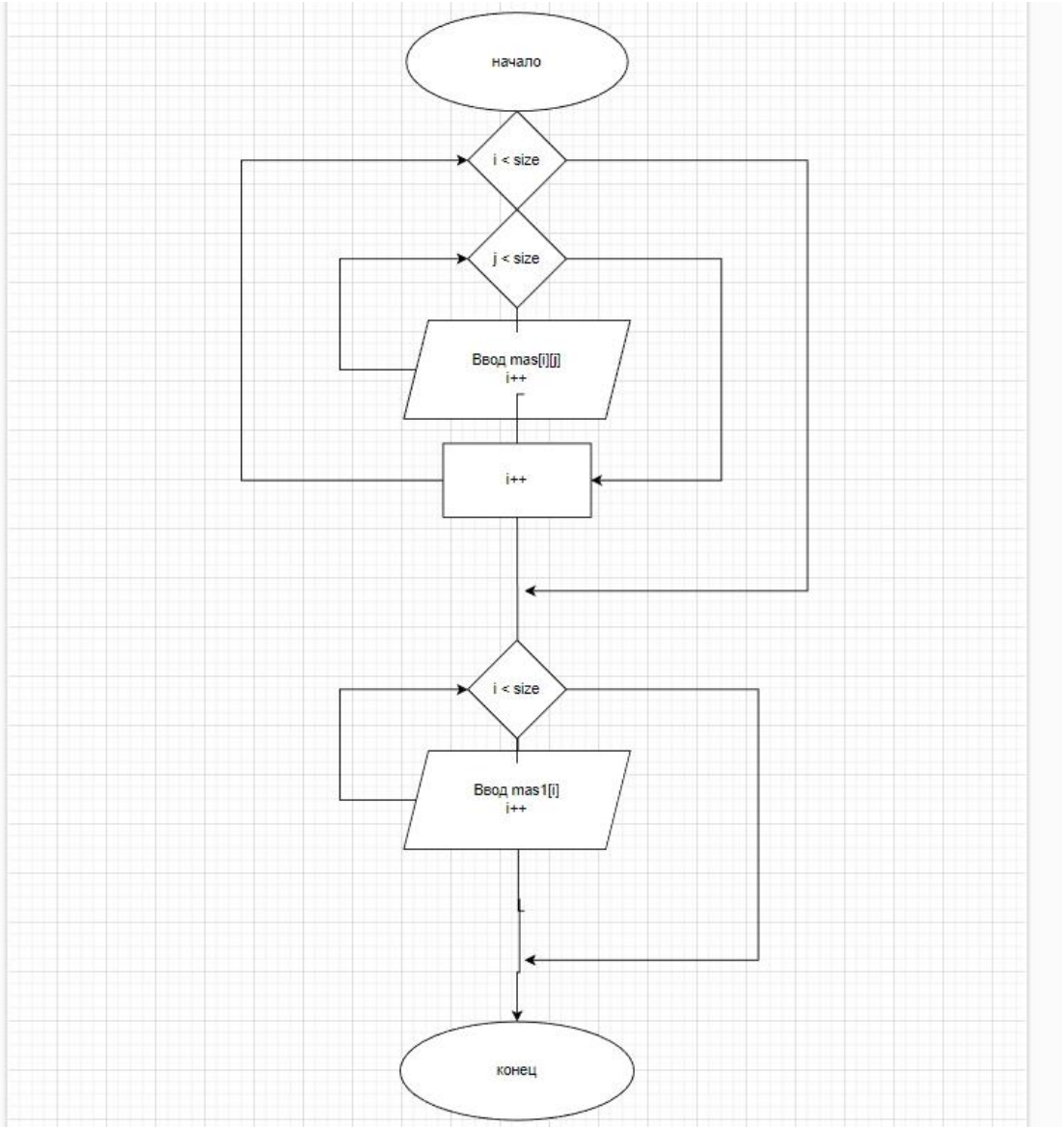
Основная идея заключается в том, что при вычислении *(k+1)-го* приближения неизвестной x_i учитываются уже вычисленные ранее (k+1) - приближение неизвестных $x_1, x_2, ..., x_n$.

Итерационная схема имеет вид:

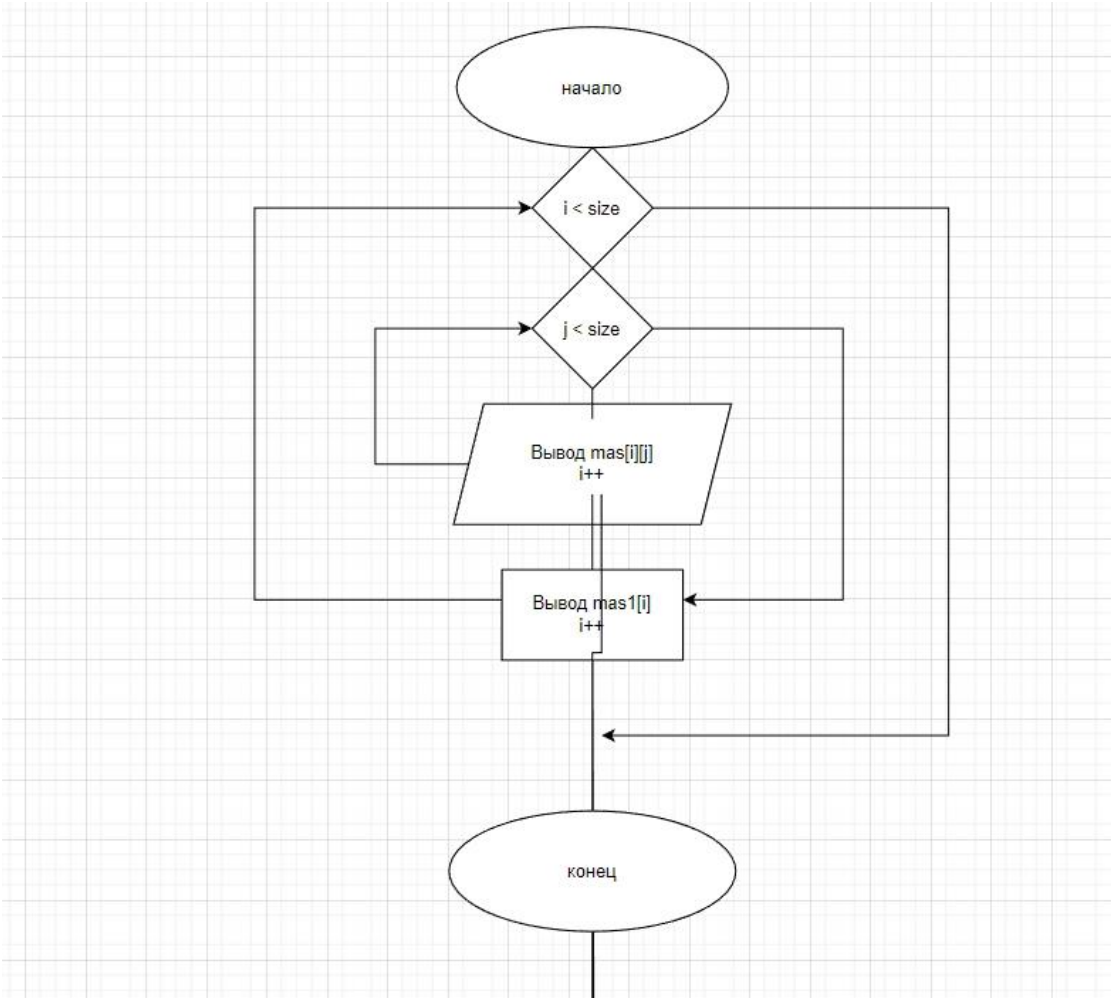
Схема алгоритма:

Class Matr:

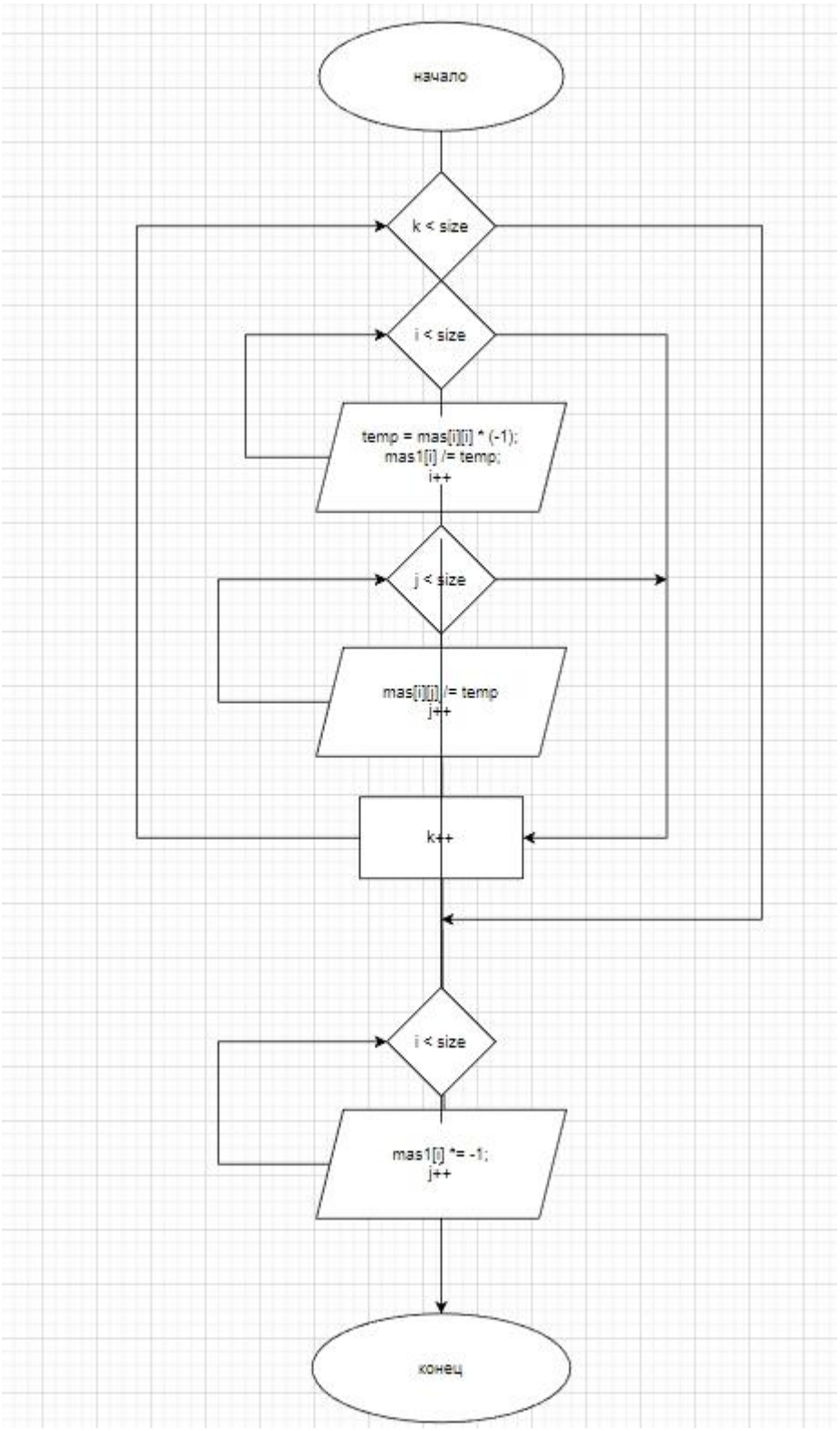
Add:



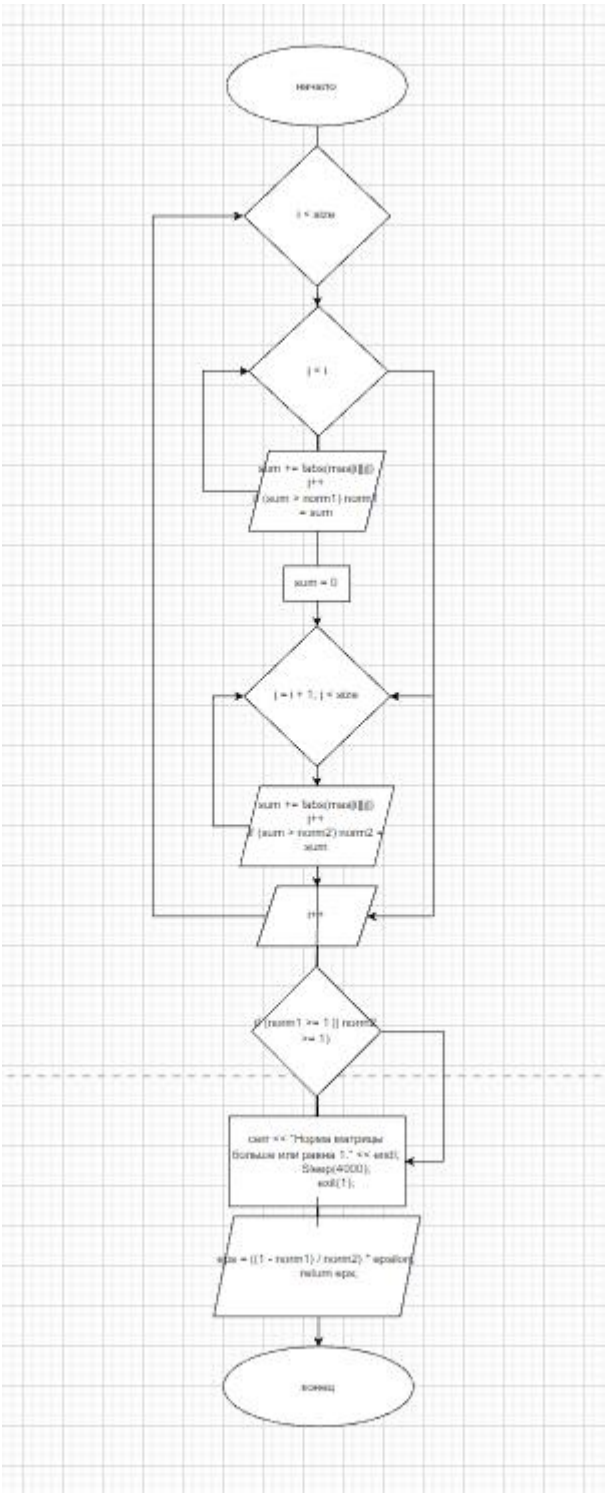
Print:



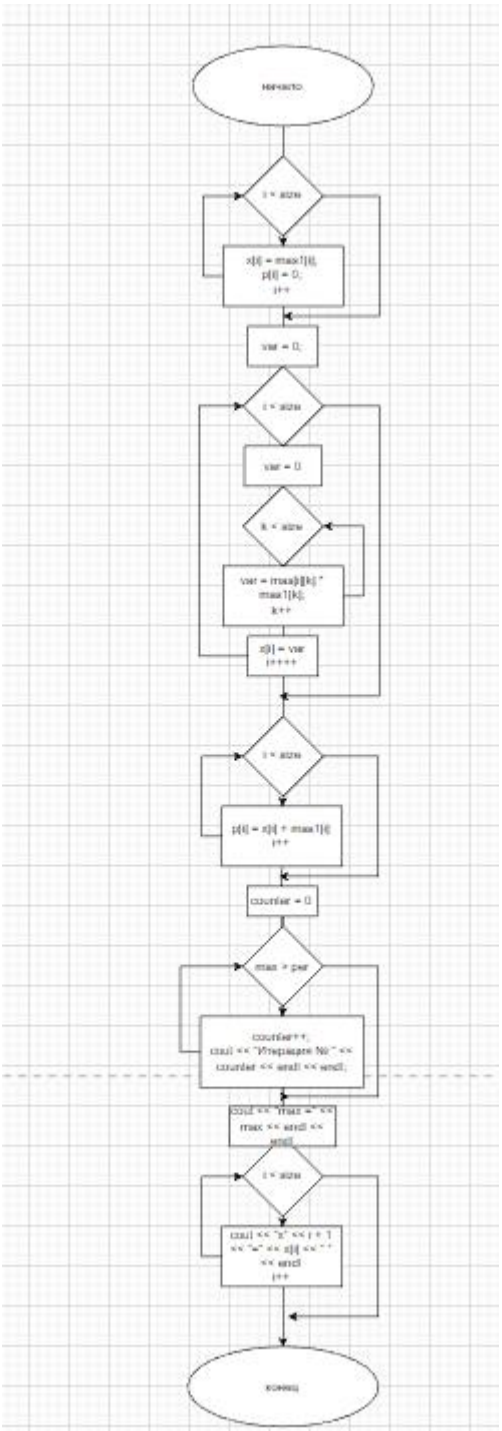
Preob:


начало
k < size
i < size
temp = mas[i][i] * (-1);
mas1[i] /= temp;
i++
j < size
mas[i][j] /= temp
j++
k++
i < size
mas1[j] *= -1;
j++
конец

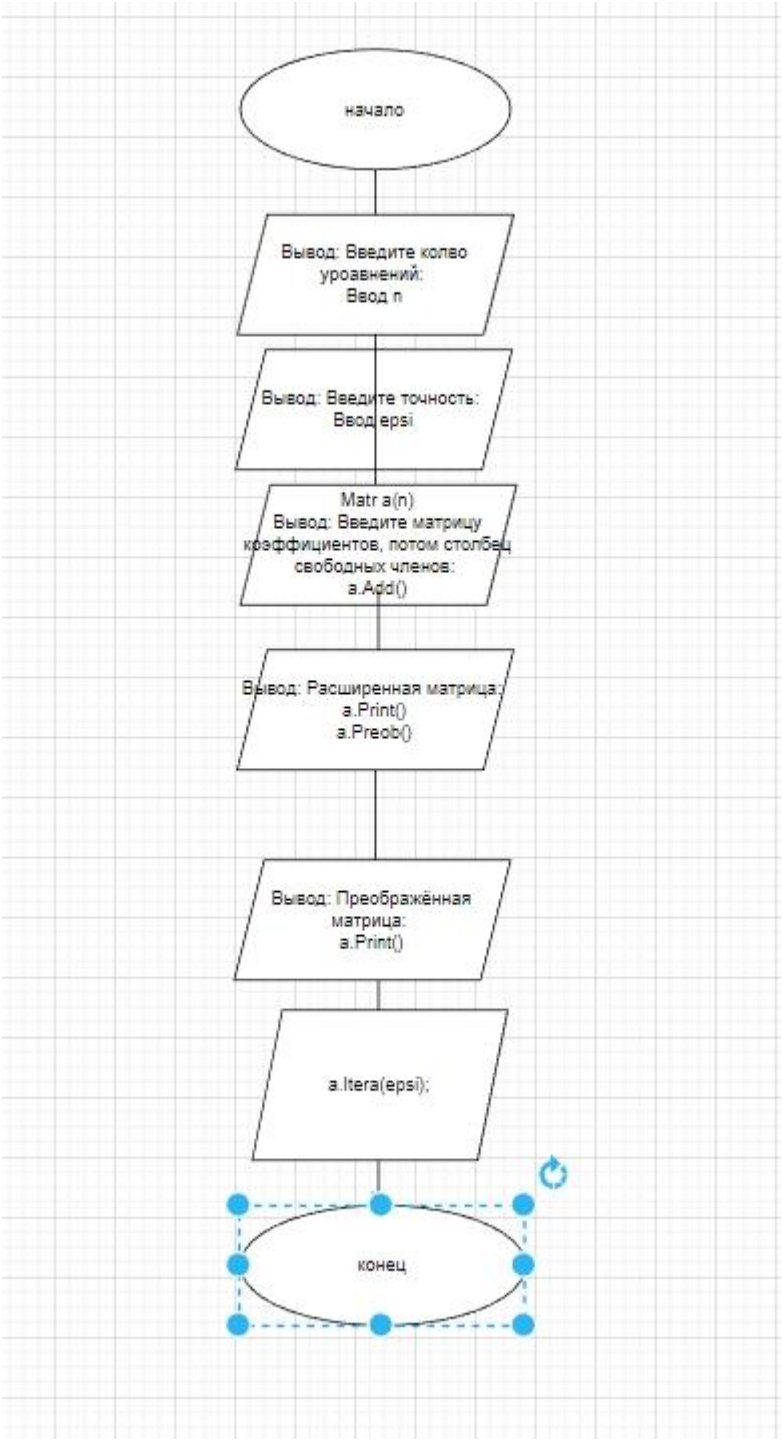
Pogr:


начало
i < size
j < i
sum += fabs(mas[i][j])
j++
if (sum > norm1) norm1 = sum
sum = 0
j = i + 1; j < size
sum += fabs(mas[i][j])
j++
if (sum > norm2) norm2 = sum
i++
if (norm1 >= 1 || norm2 >= 1)
cout << "Норма матрицы больше или равна 1." << endl;
Sleep(4000);
exit(1);
eps = ((1 - norm1) / norm2) * epsilon;
return eps;
конец

Itera:


i < size
x[i] = max1[i];
p[i] = 0;
i++
var = 0;
i < size
var = 0
k < size
var = max[i][k] * max1[k];
k++
x[i] = var
i++++
i < size
p[i] = x[i] + max1[i]
i++
counter = 0;
max > per
counter++;
cout << "Итерация №" << counter << endl << endl;
cout << "max =" << max << endl << endl;
i < size
cout << "x" << j + 1 << "=" << x[i] << " " << endl
i++

main:



Листинг кода программы:

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <windows.h>
using namespace std;
```

```
class Matr
{
private:
   int size;
   double** mas;
   double* mas1;
public:
   Matr()
   {
      size = 0;
      mas = NULL;
      mas1 = NULL;
   }
   Matr(int l)
   {
      size = l;
      mas = new double* [l];
      for (int i = 0; i < l; i++)
         mas[i] = new double[l];
      mas1 = new double[l];
   }
   void Add()
   {
      for (int i = 0; i < size; i++)
         for (int j = 0; j < size; j++)
            cin >> mas[i][j];
      for (int i = 0; i < size; i++)
      {
         cin >> mas1[i];
      }
   }
   void Print()
   {
      for (int i = 0; i < size; i++)
      {
```

```
        for (int j = 0; j < size; j++)
        {
            cout << setw(4) << mas[i][j] << " ";
        }
        cout << " " << mas1[i] << endl;
    }
}
void Preob()
{
    double temp = 0;
    for (int k = 0; k < size; k++)
    {
        for (int i = 0; i < size; i++)
        {
            temp = mas[i][i] * (-1);
            mas1[i] /= temp;
            for (int j = 0; j <= size; j++)
            {
                mas[i][j] /= temp;
            }
        }
    }
    for (int i = 0; i < size; i++)
    {
        mas1[i] *= -1;
        for (int j = 0; j < size; j++)
            mas[i][i] = 0;
    }
}
double Pogr(double** mas, double epsilon)
{
    double eps = 0; double sum = 0, max = 0;
    double norm1 = 0, norm2 = 0;
    for (int i = 0; i < size; i++)
    {
```

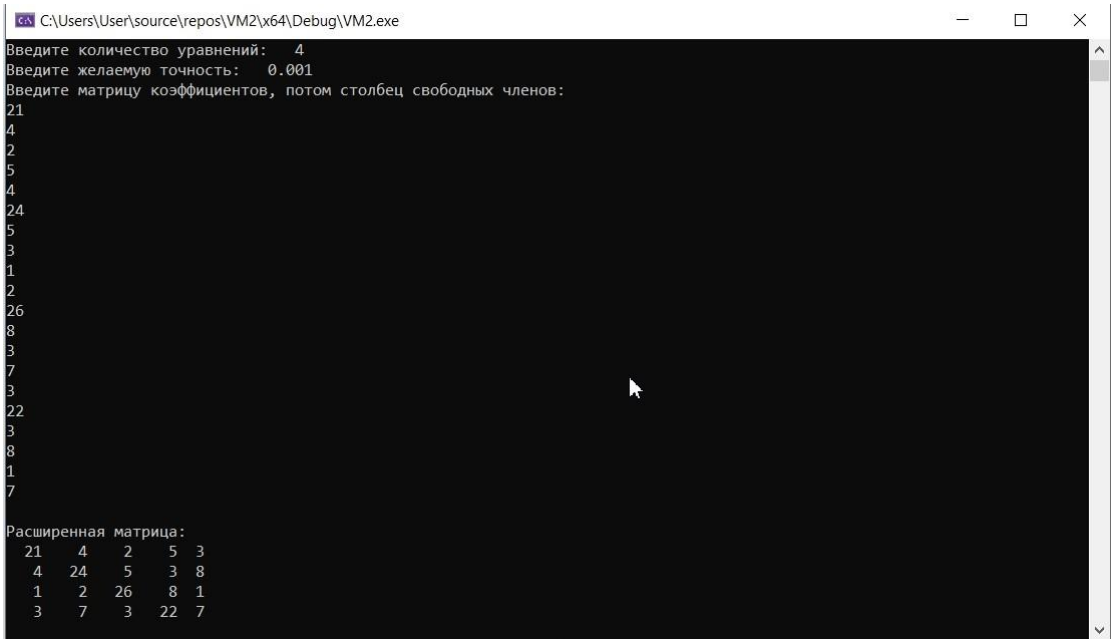
```
        for (int j = 0; j < i; j++)
        {
            sum += fabs(mas[i][j]);
            if (sum > norm1) norm1 = sum;
        }
        sum = 0;
        for (int j = i + 1; j < size; j++)
        {
            sum += fabs(mas[i][j]);
            if (sum > norm2) norm2 = sum;
        }
        sum = 0;
    }
    if (norm1 >= 1 || norm2 >= 1)
    {
        cerr << "Норма матрицы больше или равна 1." << endl;
        Sleep(4000);
        exit(1);
    } //проверка
    eps = ((1 - norm1) / norm2) * epsilon;
    return eps;
}
void Itera(double epsilon)
{
    double* x = new double[size];
    double* p = new double[size];
    double* a = new double[size];
    double* E = new double[size];
    double per = Pogr(mas, epsilon), max = 0;
    for (int i = 0; i < size; i++)
    {
        x[i] = mas1[i];
        p[i] = 0;
    }
    double var = 0;
```

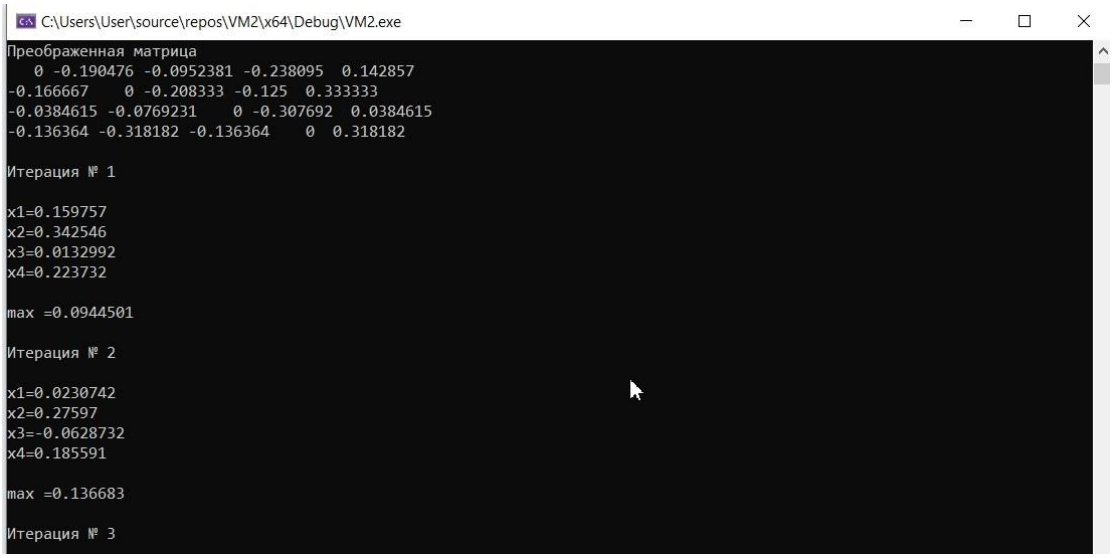
```
for (int i = 0; i < size; i++)
{
    var = 0;
    for (int k = 0; k < size; k++)
        var = mas[i][k] * mas1[k];
    x[i] = var;
}
for (int i = 0; i < size; i++)
    p[i] = x[i] + mas1[i];
int counter = 0;
do
{
    counter++;
    cout << "Итерация № " << counter << endl << endl;
    for (int i = 0; i < size; i++)
    {
        var = 0;
        for (int j = 0; j < i; j++)
            var += (mas[i][j] * p[j]);
        for (int j = i + 1; j < size; j++)
            var += (mas[i][j] * x[j]);
        a[i] = var;
        x[i] = mas1[i] + a[i];
    }
    max = 0;
    for (int i = 0; i < size; i++)
    {
        E[i] = fabs(x[i] - p[i]);
        if (max < E[i]) max = E[i];
        p[i] = x[i];
        cout << "x" << i + 1 << "=" << x[i] << " " << endl;
    }
    cout << endl;
    cout << "max =" << max << endl << endl;
} while (max > per);
```

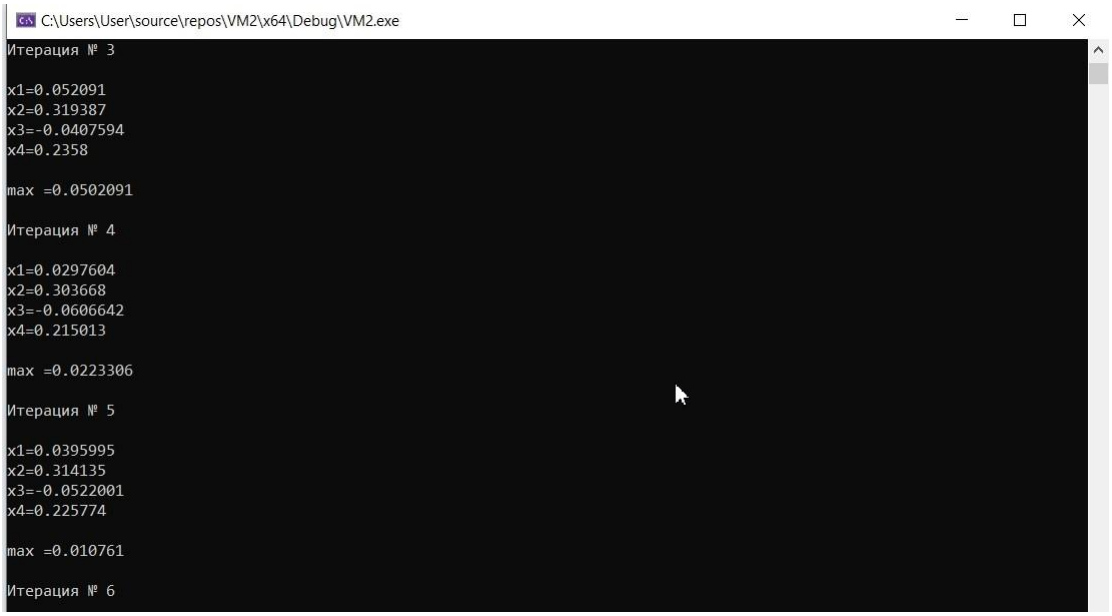
```
        cout << endl << "Результат: \n\n";
        for (int i = 0; i < size; i++)
            cout << "x" << i + 1 << "=" << x[i] << " " << endl;
        delete[] x;
        delete[] p;
        delete[] E;
        delete[] a;
    }
    ~Matr()
    {
        for (int i = 0; i < size; i++)
            delete mas[i];
        delete mas;
    }
};
int main()
{
    setlocale(LC_ALL, "rus");
    int n; double epsi;
    cout << "Введите количество уравнений:   ";
    cin >> n;
    cout << "Введите желаемую точность:   ";
    cin >> epsi;
    Matr a(n);
    cout << "Введите матрицу коэффициентов, потом столбец свободных членов:" << endl;
    a.Add();
    cout << endl << "Расширенная матрица:" << endl;
    a.Print();
    a.Preob();
    cout << endl << "Преображенная матрица" << endl;
    a.Print();
    cout << endl;
    a.Itera(epsi);
    cout << endl;
    system("pause");
```

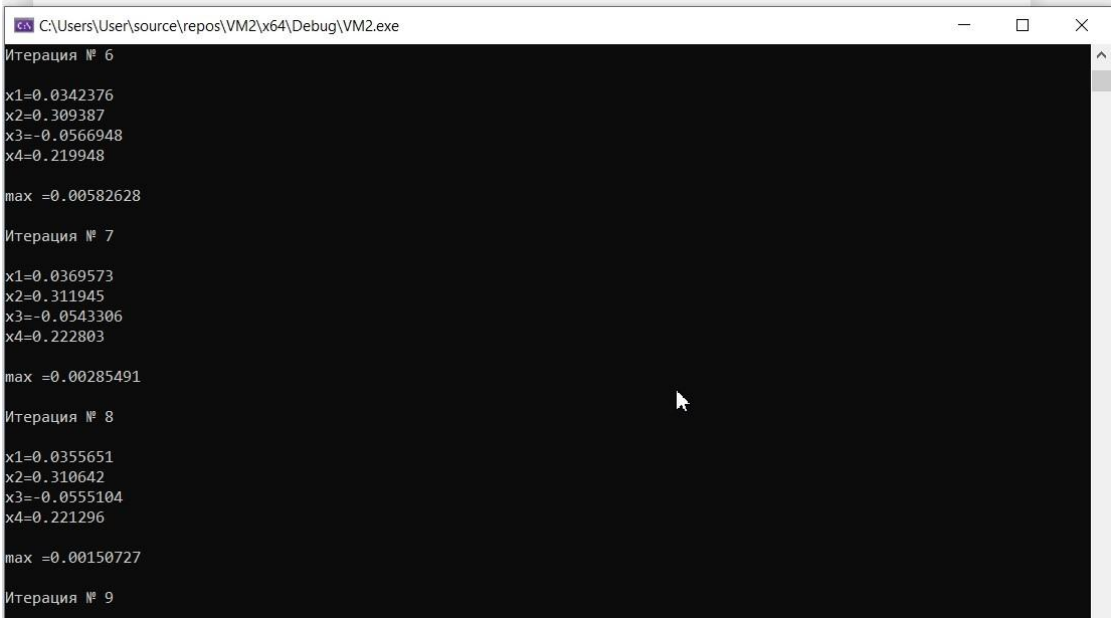
```
    return 0;
}
```

Результаты программных расчетов:









Сравнение результатов программных и аналитических расчетов:

Исходя из результатов мы видим, что результаты сходятся с допустимой разницей.

Вывод

В ходе выполнения практической работы №2 был освоен метод решения СЛАУ – метод Зейделя. Также были улучшены навыки по алгоритмизации и программированию вычислительной задачи на языке C++ в программе Microsoft Visual Studio.