

Министерство образования Республики Беларусь  
Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей  
Кафедра информатики  
Дисциплина: Методы численного анализа

**ОТЧЁТ**  
к лабораторной работе  
на тему

Численное решение систем линейных уравнений методом простых итераций и  
методом Зейделя

Выполнил: студент группы 253505  
Снежко Максим Андреевич

Проверил: Анисимов Владимир Яковлевич

Минск 2023

## **Вариант 27**

### **Цель выполнения задания:**

- Изучить итерационные методы решения СЛАУ (метод простых итераций, метод Зейделя);
- Составить алгоритм решения СЛАУ указанными методами, применимый для организации вычислений на ЭВМ;
- Составить программу решения СЛАУ по разработанному алгоритму;
- Численно решить тестовые примеры и проверить правильность работы программы. Сравнить трудоёмкость решения методом простых итераций и методом Зейделя.

## Краткие теоретические сведения

Итерационные методы основаны на построении сходящейся к точному решению  $x$  рекуррентной последовательности.

Для решения СЛАУ методом простых итераций преобразуем систему от первоначальной формы  $Ax = b$  или

[illegible]

## К ВИДУ

$$\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{c} \quad (2.2)$$

Здесь  $B$  - квадратная матрица с элементами  $b_{ij}$  ( $i, j = 1, 2, \dots, n$ ),  $c$  – вектор столбец с элементами  $c_i$  ( $i = 1, 2, \dots, n$ ).

В развернутой форме записи система (2.2) имеет следующий вид:

$$\begin{aligned} x_1 &= b_{11}x_1 + b_{12}x_2 + b_{13}x_3 + \dots + b_{1n}x_n + c_1 \\ x_2 &= b_{21}x_1 + b_{22}x_2 + b_{23}x_3 + \dots + b_{2n}x_n + c_2 \\ &\vdots \\ x_n &= b_{n1}x_1 + b_{n2}x_2 + b_{n3}x_3 + \dots + b_{nn}x_n + c_n \end{aligned}$$

Вообще говоря, операция приведения системы к виду, удобному для итераций, не является простой и требует специальных знаний, а также существенного использования специфики системы.

Можно, например, преобразовать систему (2.1) следующим образом

$$\begin{aligned}x_1 &= (b_1 - a_{11} x_1 - a_{12} x_2 - \dots - a_{1n} x_n) / a_{11} + x_1, \\x_2 &= (b_2 - a_{21} x_1 - a_{22} x_2 - \dots - a_{2n} x_n) / a_{22} + x_2, \\&\vdots \\x_n &= (b_n - a_{n1} x_1 - a_{n2} x_2 - \dots - a_{nn} x_n) / a_{nn} + x_n\end{aligned}$$

если диагональные элементы матрицы  $A$  отличны от нуля.

Можно преобразовать систему (2.1) в эквивалентную ей систему

$$\mathbf{x} = (\mathbf{E}-\mathbf{A})\mathbf{x}+\mathbf{b}.$$

Задав произвольным образом столбец начальных приближений  $x_0 = (x_{10}, x_{20}, \dots, x_{n0})^T$ , подставим их в правые части системы (2.2) и вычислим новые приближения  $x_1 = (x_{11}, x_{21}, \dots, x_{n1})$ , которые опять подставим в систему (2.2) и т.д. Таким образом, организуется итерационный процесс  $x_k = Vx_{k-1} + c$ ,  $k = 1, 2, \dots$ . Известно, что система (2.1) имеет единственное решение  $x^*$  и

последовательность  $\{x_k\}$  сходится к этому решению со скоростью геометрической прогрессии, если  $\|B\| < 1$  в любой матричной норме. Т.е. для того, чтобы последовательность простых итераций сходилась к единственному решению достаточно, чтобы выполнялось одно из следующих условий:

$$1) \max_i \left( \sum_{j=1}^n |b_{ij}| \right) < 1 ;$$

$$2) \sum_{i=1}^n \sum_{j=1}^n |b_{ij}|^2 < 1;$$

$$3) \max_j \left( \sum_{i=1}^n |b_{ij}| \right) < 1.$$

### Метод Зейделя

Метод Зейделя является модификацией метода простых итераций. Суть его состоит в том, что при вычислении следующего  $x_{ik} : 2$  в формуле  $x_k = Bx_{k-1} + c$ ,  $k=1,2, \dots$  используются вместо  $x_{1k-1}, \dots$ , уже вычисленные ранее  $x_{ik}, \dots$ , т.е.

$$x_i^k = \sum_{j=1}^{i-1} g_{ij} x_j^k + \sum_{j=i+1}^n g_{ij} x_j^{k-1} + c_i . \quad (2.3)$$

Такое усовершенствование позволяет ускорить сходимость итераций почти в два раза. Кроме того, данный метод может быть реализован на ЭВМ без привлечения дополнительного массива, т.к. полученное новое сразу засыпается на место старого.

Схема алгоритма аналогична схеме метода простых итераций.

Самый простой способ приведения системы к виду, удобному для итераций, состоит в следующем. Из первого уравнения системы выразим неизвестное  $x_1$  :

$$x_1 = a_{11}^{-1} (b_1 - a_{12}x_2 - a_{13}x_3 - \dots - a_{1n}x_n),$$

из второго уравнения – неизвестное  $x_2$ :

$$x_2 = a_{22}^{-1} (b_2 - a_{21}x_1 - a_{23}x_3 - \dots - a_{2n}x_n),$$

и т. д.

В результате получим систему:

[illegible]

в которой на главной диагонали матрицы  $B$  находятся нулевые элементы. Остальные элементы выражаются по формулам

$$b_{ij} = -a_{ij} / a_{ii} \quad c_i = b_i / a_{ii} \quad (i, j = 1, 2, \dots, n, j \neq i) \quad (2.4)$$

Конечно, для возможности выполнения указанного преобразования необходимо, чтобы диагональные элементы матрицы  $A$  были ненулевыми.

Введем нижнюю  $B_1$  (получается из  $B$  заменой нулями элементов стоявших на главной диагонали и выше ее) и верхнюю  $B_2$  (получается из  $B$  заменой нулями элементов стоявших на главной диагонали и ниже ее) треугольные матрицы.

Заметим, что  $B = B_1 + B_2$  и поэтому решение  $x$  исходной системы удовлетворяет равенству

$$x = B_1x + B_2x + c \quad (2.5)$$

Выберем начальное приближение  $x(0) = [x_1(0), x_2(0), \dots, x_n(0)]^T$ . Подставляя его в правую часть равенства при верхней треугольной матрице  $B_2$  и вычисляя полученное выражение, находим первое приближение

$$\mathbf{x}(1) = \mathbf{B}_1 \mathbf{x}(0) + \mathbf{B}_2 \mathbf{x}(1) \quad (2.6)$$

Подставляя приближение  $x(1)$ , получим

$$\mathbf{X}(2) = \mathbf{B}_1 \mathbf{x}(1) + \mathbf{B}_2 \mathbf{x}(2) \quad (2.7)$$

Продолжая этот процесс далее, получим последовательность  $x(0), x(1), \dots, x(n)$ ,  $\dots$  приближений к вычисляемым по формуле

$$\mathbf{X}(\mathbf{k}+1) = \mathbf{B1}(\mathbf{k}+1) + \mathbf{B2}(\mathbf{k}) + \mathbf{c} \quad (2.8)$$

или в развернутой форме записи:

$$x_1^{(k+1)} = b_{12}x_2^{(k)} + b_{13}x_3^{(k)} + \dots + b_{1n}x_n^{(k)} + c_1,$$

$$x_2^{(k+1)} = b_{21}x_1^{(k+1)} + b_{23}x_3^{(k)} + \dots + b_{2n}x_n^{(k)} + c_2,$$

$$x_3^{(k+1)} = b_{31}x_1^{(k+1)} + b_{32}x_2^{(k+1)} + \dots + b_{3n}x_n^{(k)} + c_3,$$

.....

$$x_n^{(k+1)} = b_{n1}x_1^{(k+1)} + b_{n2}x_2^{(k+1)} + b_{n3}x_3^{(k+1)} + \dots + c_n.$$

Объединив приведение системы к виду, удобному для итераций и метод Зейделя в одну формулу, получим

$$x_i^{(k+1)} = x_i^{(k)} - a_{ii}^{-1}(\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} + \sum_{j=i+1}^n a_{ij}x_j^{(k)} - b_i). \quad (2.9)$$

Тогда достаточным условием сходимости метода Зейделя будет условие доминирования диагональных элементов в строках или столбцах матрицы A, т.е.

$a_{ii} > a_{i1} + \dots + a_{in}$  для всех  $i=1, \dots, n$ ,

или  $a_{jj} > a_{1j} + \dots + a_{nj}$  для всех  $j=1, \dots, n$ .

Методы простой итерации и Зейделя сходятся примерно так же, как геометрическая прогрессия со знаменателем  $\|B\|$ .

## ЗАДАНИЕ

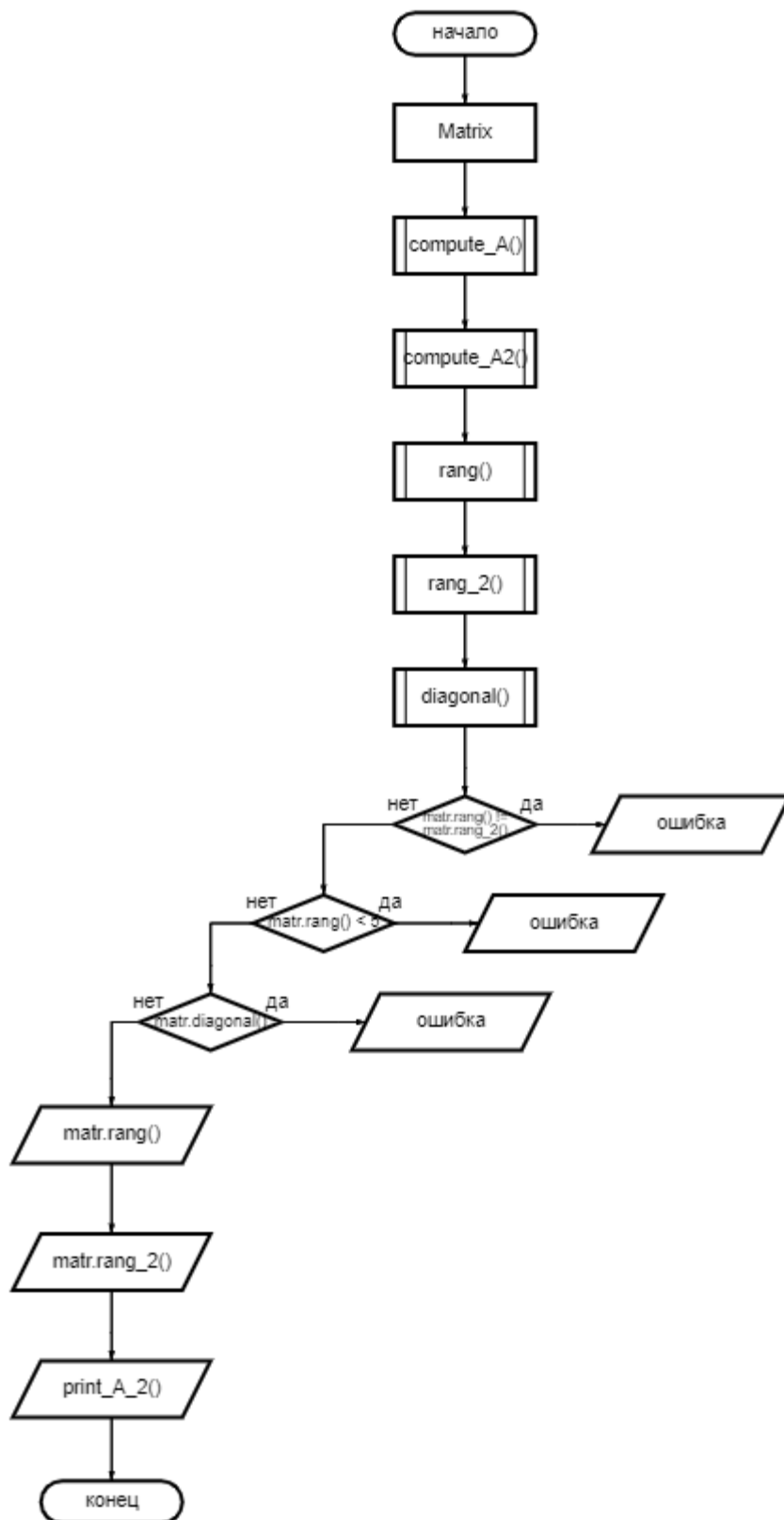
Методом простых итераций и методом Зейделя найти с точностью 0,0001 численное решение системы  $Ax=b$ , где  $A = kC + D$ ,  $A$  - исходная матрица для расчёта,  $k$  - номер варианта (0-15), матрицы  $C$ ,  $D$  и вектор свободных членов  $b$  задаются ниже.

$$C = \begin{bmatrix} 0,01 & 0 & -0,02 & 0 & 0 \\ 0,01 & 0,01 & -0,02 & 0 & 0 \\ 0 & 0,01 & 0,01 & 0 & -0,02 \\ 0 & 0 & 0,01 & 0,01 & 0 \\ 0 & 0 & 0 & 0,01 & 0,01 \end{bmatrix}, \quad D = \begin{bmatrix} 1,33 & 0,21 & 0,17 & 0,12 & -0,13 \\ -0,13 & -1,33 & 0,11 & 0,17 & 0,12 \\ 0,12 & -0,13 & -1,33 & 0,11 & 0,17 \\ 0,17 & 0,12 & -0,13 & -1,33 & 0,11 \\ 0,11 & 0,67 & 0,12 & -0,13 & -1,33 \end{bmatrix}.$$

Вектор  $b = (1,2; 2,2; 4,0; 0,0; -1,2)^T$ .

## **Алгоритм выполнения задания**





## Программная реализация

```
#include <iostream>
#include <iomanip>
#include <vector>

class Matrix
{
private:
    int V;
    double A[5][5];
    double A_2[5][6];
    double C[5][5]
    {
        {1.0 / 100, 0, -2.0 / 100, 0, 0},
        {1.0 / 100, 1.0 / 100, -2.0 / 100, 0, 0},
        {0, 1.0 / 100, 1.0 / 100, 0, -2.0 / 100},
        {0, 0, 1.0 / 100, 1.0 / 100, 0},
        {0, 0, 0, 1.0 / 100, 1.0 / 100}
    };
    double D[5][5]
    {
        {133.0 / 100, 21.0 / 100, 17.0 / 100, 12.0 / 100, -13.0 / 100},
        {-13.0 / 100, -133.0 / 100, 11.0 / 100, 17.0 / 100, 12.0 / 100},
        {12.0 / 100, -13.0 / 100, -133.0 / 100, 11.0 / 100, 17.0 / 100},
        {17.0 / 100, 12.0 / 100, -13.0 / 100, -133.0 / 100, 11.0 / 100},
        {11.0 / 100, 67.0 / 100, 12.0 / 100, -13.0 / 100, -133.0 / 100}
    };
    double b[5]{ 12.0 / 10, 22.0 / 10, 40.0 / 10, 0, -12.0 / 10 };
    double roots[5]{ 0, 0, 0, 0, 0 };
public:
    Matrix(const int V)
    {
        this->V = V;
        std::cout << std::fixed << std::setprecision(4);
        compute_A();
        compute_A_2();
    }
    void compute_A()
    {
        for (int i = 0; i < 5; i++)
        {
            for (int j = 0; j < 5; j++)
            {
                A[i][j] = V * C[i][j] + D[i][j];
            }
        }
    }
    void compute_A_2()
    {
        for (int i = 0; i < 5; i++)
        {
            for (int j = 0; j < 5; j++)
            {
                A_2[i][j] = A[i][j];
            }
        }
    }
};
```

```

    }
    }
    A_2[0][5] = 12.0 / 10;
    A_2[1][5] = 22.0 / 10;
    A_2[2][5] = 40.0 / 10;
    A_2[3][5] = 0;
    A_2[4][5] = -12.0 / 10;
}
void print_A()
{
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            std::cout << std::setw(10) << A[i][j];
        }
        std::cout << std::setw(5) << '|' << std::setw(5) << b[i] <<
'\n';
    }
}
void print_A_2()
{
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            std::cout << std::setw(10) << A_2[i][j];
        }
        std::cout << std::setw(5) << '|' << std::setw(5) << A_2[i][5] <<
std::endl;
    }
}

// Метод простых итераций
std::vector<double> simpleIterations(int& iterations)
{
    // Размер матрицы
    int size = 5;
    // Известные на предыдущей итерации
    std::vector<double> prev_X(size, 0.0);
    // Пока не будет достигнута точность
    bool stop = false;
    while (!stop)
    {
        ++iterations;
        // Известные на текущей итерации
        std::vector<double> current_X(size);
        for (int i = 0; i < size; i++)
        {
            // x_i = b_i
            current_X[i] = A_2[i][size];
            // Вычитаем сумму по всем отличным от i-ой неизвестным
            for (int j = 0; j < size; j++)
            {
                // С прошлой итерации
                if (j != i)
                {

```

```

        current_X[i] -= A_2[i][j] * prev_X[j];
    }
    }
    // x_i /= b_i
    current_X[i] /= A_2[i][i];
}
// Максимальная погрешность
long double max_error = 0.0;
for (int i = 0; i < size; i++)
{
    double new_max_error = abs(current_X[i] - prev_X[i]);
    max_error = new_max_error > max_error ? new_max_error :
max_error;
}
// Достигнута ли точность
if (max_error < 0.0001)
{
    stop = true;
}
// Переход к следующей итерации
prev_X = current_X;
}
return prev_X;
}

// Метод Зейделя

std::vector<double> seidelMethod(int& iterations)
{
    // Размер матрицы
    int size = 5;
    // Известные на предыдущей итерации
    std::vector<double> prev_X(size, 0.0);
    // Пока не будет достигнута точность
    bool stop = false;
    while (!stop)
    {
        ++iterations;
        // Известные на текущей итерации
        std::vector<double> current_X(size);
        for (int i = 0; i < size; i++)
        {
            // x_i = b_i
            current_X[i] = A_2[i][size];
            // Вычитаем сумму по всем отличным от i-ой неизвестным
            for (int j = 0; j < size; j++)
            {
                // С этой итерации
                if (j < i)
                {
                    current_X[i] -= A_2[i][j] * current_X[j];
                }
                // С прошлой итерации
                if (j > i)
                {
                    current_X[i] -= A_2[i][j] * prev_X[j];
                }
            }
        }
    }
}

```

```

        }
        // x_i /= b_i
        current_X[i] /= A_2[i][i];
    }
    // Максимальная погрешность
    long double max_error = 0.0;
    for (int i = 0; i < size; i++)
    {
        double new_max_error = abs(current_X[i] - prev_X[i]);
        max_error = new_max_error > max_error ? new_max_error :
max_error;
    }
    // Достигнута ли точность
    if (max_error < 0.0001)
    {
        stop = true;
    }
    // Переход к следующей итерации
    prev_X = current_X;
}
return prev_X;
}

// Функции для проверки исходной матрицы :
// Нахождение ранга матрицы

int rang()
{
    int rang = 5;
    double sum = 0;
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            sum += A[i][j];
        }
        if (sum == 0)
        {
            rang = rang - 1;
        }
        sum = 0;
    }
    return rang;
}

int rang_2()
{
    int rang = 6;
    double sum = 0;
    for (int i = 0; i < 6; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            sum += A_2[i][j];
        }
        if (sum == 0)
        {

```

```

        rang = rang - 1;
    }
    sum = 0;
}
return rang;
}

// Проверка на преобладание главной диагонали
bool diagonal()
{
    int i, j, k = 1;
    double sum;
    for (i = 0; i < 5; i++)
    {
        sum = 0;
        for (j = 0; j < 5; j++)
        {
            sum += A[i][j];
        }
        sum -= A[i][i];
        if (sum < A[i][i])
        {
            return 0;
        }
    }
    return k == 1;
}

};

int main()
{
    setlocale(0, "");
    Matrix matr(27);
    if (matr.rang() != matr.rang_2())
    {
        std::cout << "Ранг исходной матрицы не равен рангу расширенной матрицы.\nСЛАУ не имеет решений.";
        return 0;
    }
    if (matr.rang() < 5)
    {
        std::cout << "Ранги матрицы равны или меньше числа неизвестных системы.\nСЛАУ имеет бесконечное множество решений.";
        return 0;
    }
    if (matr.diagonal())
    {
        std::cout << "Исходная матрица не обладает свойством диагонального преобладания.\nРешение методом итераций/Зейделя невозможен.";
        return 0;
    }
    std::cout << matr.rang() << ", " << matr.rang_2();
    std::cout << "\t\t\tМетод итераций:" << "\n\n";
    std::cout << "\tИсходная матрица:" << "\n";
    matr.print_A_2();
    int iterations = 0;
    std::vector<double> vect = matr.seidelMethod(iterations);
}

```

```

std::cout << iterations << "\n";
int t = 0;
for (auto i : vect)
{
    std::cout << ++t << ". " << i << "\n";
}
std::cout << "\n\n\t\t\tМетод Зейделя" << std::endl << std::endl;
std::cout << "\tИсходная матрица:" << std::endl;
matr.compute_A();
matr.compute_A_2();
matr.print_A_2();
iterations = 0;
vect = matr.simpleIterations(iterations);
std::cout << iterations << "\n";
for (auto i : vect)
{
    std::cout << ++t << ". " << i << "\n";
}
}

```

## Полученные результаты программы

Ранг матриц 5, 5

Метод итераций:

Исходная матрица:

1.6000	0.2100	-0.3700	0.1200	-0.1300	1.2000
0.1400	-1.0600	-0.4300	0.1700	0.1200	2.2000
0.1200	0.1400	-1.0600	0.1100	-0.3700	4.0000
0.1700	0.1200	0.1400	-1.0600	0.1100	0.0000
0.1100	0.6700	0.1200	0.1400	-1.0600	-1.2000

10

1. -0.0426

2. -0.5180

3. -4.0007

4. -0.5656

5. 0.2726

Метод Зейделя

Исходная матрица:

1.6000	0.2100	-0.3700	0.1200	-0.1300	1.2000
0.1400	-1.0600	-0.4300	0.1700	0.1200	2.2000
0.1200	0.1400	-1.0600	0.1100	-0.3700	4.0000
0.1700	0.1200	0.1400	-1.0600	0.1100	0.0000
0.1100	0.6700	0.1200	0.1400	-1.0600	-1.2000

17

6. -0.0426

7. -0.5180

8. -4.0007

9. -0.5656

10. 0.2726

**Результаты, полученные в онлайн-калькуляторе:**

С сайта [math.semestr.ru](http://math.semestr.ru) :

$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
-0.0426	-0.5180	-4.0007	-0.5656	0.2726



# Тестовые примеры

## Пример 1

Возьмем в качестве матрицы  $A$  нулевую матрицу. В этом случае СЛАУ не имеет решений, так как ранг исходной матрицы не равен рангу расширенной матрицы.

Результат выполнения программы:

```
Ранг исходной матрицы не равен рангу расширенной матрицы.  
СЛАУ не имеет решений.
```

## Пример 2

$A = \{ \{2.33, 0.81, 0.67, 0.92, -0.53\},$   
 $\{-0.53, 2.33, 0.81, 0.67, 0.92\},$   
 $\{0.92, -0.53, 2.33, 0.81, 0.67\},$   
 $\{0.67, 0.92, -0.53, 2.33, 0.81\},$   
 $\{0.81, 0.67, 0.92, -0.53, 2.33\} \};$

Результат выполнения программы:

```
Ранг матриц 5, 5                                Метод итераций:

Исходная матрица:
 2.3300  0.8100  0.6700  0.9200 -0.5300 | 1.2000
-0.5300  2.3300  0.8100  0.6700  0.9200 | 2.2000
 0.9200 -0.5300  2.3300  0.8100  0.6700 | 4.0000
 0.6700  0.9200 -0.5300  2.3300  0.8100 | 0.0000
 0.8100  0.6700  0.9200 -0.5300  2.3300 | -1.2000

28
1. -0.6873
2. 0.2031
3. 1.9892
4. 0.8890
5. -0.9177

Метод Зейделя

Исходная матрица:
 2.3300  0.8100  0.6700  0.9200 -0.5300 | 1.2000
-0.5300  2.3300  0.8100  0.6700  0.9200 | 2.2000
 0.9200 -0.5300  2.3300  0.8100  0.6700 | 4.0000
 0.6700  0.9200 -0.5300  2.3300  0.8100 | 0.0000
 0.8100  0.6700  0.9200 -0.5300  2.3300 | -1.2000

41
6. -0.6873
7. 0.2031
8. 1.9892
9. 0.8890
10. -0.9177
```

### Пример 3

Пусть матрица A содержит в себе следующие данные:

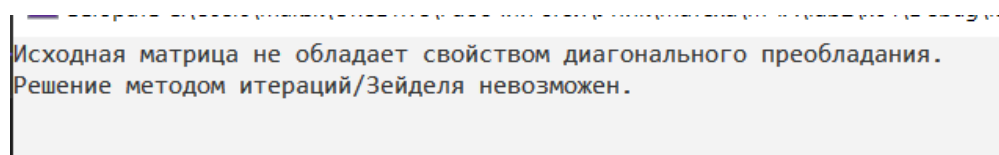
A      { {1, 2, 3, 4, 5},  
          {0, 0, 0, 0, 0},  
          {0, 0, 0, 0, 0},  
          {0, 0, 0, 0, 0},  
          {0, 0, 0, 0, 0} };

Можно заметить, что в первой строке данной матрицы не выполняется следующее правило:

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|,$$

А это значит, что матрица не обладает свойством диагонального преобладания, следовательно решать ее методом итераций или Зейделя нельзя.

Результат выполнения программы:



Исходная матрица не обладает свойством диагонального преобладания.  
Решение методом итераций/Зейделя невозможен.

Данный результат соответствует ожидаемому исходу.

### Пример 4

Пусть матрица A содержит в себе следующие данные:

A      { {1, 2, 3, 4, 5},  
          {0, 0, 0, 0, 0},  
          {0, 0, 5, 0, 0},  
          {0, 0, 0, 4, 0},  
          {0, 3, 0, 0, 0} };

Ранг такой матрицы равен 4, что меньше чем количество неизвестных (5), поэтому по следствию из теоремы Кронекера-Капелли система имеет бесконечное множество решений.

Результат выполнения программы:

Ранги матрицы равны или меньше числа неизвестных системы. СЛАУ имеет бесконечное множество решений. ■

Данный результат соответствует ожидаемому исходу.

### Пример 5

Исходная матрица не обладает диагональным преобладанием.

Консоль отладки Microsoft Visual Studio

Исходная матрица

1.4400	0.2100	-0.0500	0.1200	<u>109.8700</u>
-0.0200	-1.2200	-0.1100	0.1700	0.1200
0.1200	-0.0200	-1.2200	0.1100	-0.0500
0.1700	0.1200	-0.0200	-1.2200	0.1100
0.1100	0.6700	0.1200	-0.0200	-1.2200

Расширенная матрица

1.4400	0.2100	-0.0500	0.1200	109.8700	1.2000
-0.0200	-1.2200	-0.1100	0.1700	0.1200	2.2000
0.1200	-0.0200	-1.2200	0.1100	-0.0500	4.0000
0.1700	0.1200	-0.0200	-1.2200	0.1100	0.0000
0.1100	0.6700	0.1200	-0.0200	-1.2200	-1.2000

Исходная матрица не обладает свойством диагонального преобладания. Решение методом [простых итераций / Зейделя] невозможно.

### Пример 6

СЛАУ имеет бесконечно много решений.

```
Консоль отладки Microsoft Visual Studio

Исходная матрица
1.4400    0.2100   -0.0500    0.1200   -0.1300
1.4400    0.2100   -0.0500    0.1200   -0.1300
1.4400    0.2100   -0.0500    0.1200   -0.1300
1.4400    0.2100   -0.0500    0.1200   -0.1300
1.4400    0.2100   -0.0500    0.1200   -0.1300

Расширенная матрица
1.4400    0.2100   -0.0500    0.1200   -0.1300    1.2000
1.4400    0.2100   -0.0500    0.1200   -0.1300    1.2000
1.4400    0.2100   -0.0500    0.1200   -0.1300    1.2000
1.4400    0.2100   -0.0500    0.1200   -0.1300    1.2000
1.4400    0.2100   -0.0500    0.1200   -0.1300    1.2000

Ранг исходной матрицы: 1
Ранг расширенной матрицы: 1

Ранги матриц равны и меньше числа неизвестных системы.
СЛАУ имеет бесконечно много решений.
```

**Вывод:** в ходе выполнения данной лабораторной работы были изучены и применены итерационные методы решения СЛАУ, в частности метод простых итераций и метод Зейделя для решения системы линейных уравнений; были созданы соответствующие алгоритмы, применимые для организации вычислений на ЭВМ, и реализации программ на языке C++ для решения поставленной задачи; для проверки правильности работы программы были выполнены тестовые примеры и проведена оценка.