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 153502
Богданов Александр Сергеевич

Проверил: Анисимов Владимир Яковлевич

Минск 2022

Оглавление

Вариант 3

Цели выполнения задания

1. Изучить итерационные методы решения СЛАУ (метод простых итераций, метод Зейделя).
2. Мысленно составить алгоритм решения СЛАУ указанными методами, применимый для организации вычислений на ЭВМ.
3. Составить программу решения СЛАУ по разработанному алгоритму.
4. Численно решить тестовые примеры и проверить правильность работы программы. Сравнить трудоемкость решения методом простых итераций и методом Зейделя.

Краткие теоритические сведения

Итерационные методы основаны на построении сходящейся к точном решению $\boldsymbol{x}$ рекуррентной последовательности.

Для решения СЛАУ **методом простых итераций** преобразуем систему от первоначальной формы $\boldsymbol{Ax} = \boldsymbol{b}$ или

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2, \\ &\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n \end{aligned} \qquad (2.1)$$

к виду

$$\boldsymbol{x} = \boldsymbol{B}\boldsymbol{x} + \boldsymbol{c}. \qquad (2.2)$$

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

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

$$x_1 = b_{11}x_1 + b_{12}x_2 + b_{13}x_3 + \ldots + b_{1n}x_n + c_1$$

$$x_2 = b_{21}x_1 + b_{22}x_2 + b_{23}x_3 + \ldots + b_{2n}x_n + c_2$$

. . . . . .

$$x_n = b_{n1}x_1 + b_{n2}x_2 + b_{n3}x_3 + \ldots + b_{nn}x_n + c_n$$

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

$$x_1 = (b_1 - a_{11}x_1 - a_{12}x_2 - \ldots - a_{1n}x_n) / a_{11} + x_1,$$

$$x_2 = (b_2 - a_{21}x_1 - a_{22}x_2 - \ldots - a_{2n}x_n) / a_{22} + x_2,$$

$$\ldots\ldots$$

$$x_n = (b_n - a_{n1}x_1 - a_{n2}x_2 - \ldots - a_{nn}x_n) / a_{nn} + x_n$$

если диагональные элементы матрицы A отличны от нуля.

Можно преобразовать систему (2.1) в эквивалентную ей систему

$$x = (E-A)x+b.$$

Задав произвольным образом столбец начальных приближений $x^0 = (x_1^0, x_2^0, \ldots, x_n^0)^T$, подставим их в правые части системы (2.2) и вычислим новые приближения $x^1 = (x_1^1, x_2^1, \ldots, x_n^1)^T$, которые опять подставим в систему (2.2) и т.д. Таким образом, организуется итерационный процесс

$x^k = Bx^{k-1} + c$, k = 1,2, …. ,. 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.

Т.е. для того, чтобы последовательность простых итераций сходилась к единственному решению достаточно, чтобы выполнялось одно из следующих условий:

Метод Зейделя. Метод Зейделя является модификацией метода простых итераций. Суть его состоит в том, что при вычислении следующего x_i^k : $2 \le 1 \le n$ в формуле $x^k = Bx^{k-1} + c$, k = 1,2, …. используются вместо $x_1^{k-1},\ldots,x_{i-1}^{k-1}$ уже вычисленные ранее $x_1^k,\ldots,x_{i-1}^k$, т.е.

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

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:

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

из второго уравнения – неизвестное x_2:

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

и т. д. В результате получим систему

$$\begin{aligned}
x_1 &= b_{12}x_2 + b_{13}x_3 + \ldots + b_{1,n-1}x_{n-1} + b_{1n}x_n + c_1 ,\\
x_2 &= b_{21}x_1 + b_{23}x_3 + \ldots + b_{2,n-1}x_{n-1} + b_{2n}x_n + c_2 ,\\
x_3 &= b_{31}x_1 + b_{32}x_2 + \ldots + b_{3,n-1}x_{n-1} + b_{3n}x_n + c_3 ,\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
x_n &= b_{n1}x_1 + b_{n2}x_2 + b_{n3}x_3 + \ldots + b_{n,n-1}x_{n-1} + c_n ,
\end{aligned}$$

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

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

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

Введем нижнюю $\boldsymbol{B}_1$ (получается из $\boldsymbol{B}$ заменой нулями элементов стоявших на главной диагонали и выше ее) и верхнюю $\boldsymbol{B}2$ (получается из $\boldsymbol{B}$ заменой нулями элементов стоявших на главной диагонали и ниже ее) треугольные матрицы.

Заметим, что $\boldsymbol{B} = \boldsymbol{B}_1 + \boldsymbol{B}_2$ и поэтому решение $\boldsymbol{x}$ исходной системы удовлетворяет равенству

$$\boldsymbol{x} = \boldsymbol{B}_1\boldsymbol{x} + \boldsymbol{B}_2\boldsymbol{x} + \boldsymbol{c} \qquad (2.5)$$

Выберем начальное приближение $\boldsymbol{x}^{(0)} = [x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}]^{\mathrm{T}}$. Подставляя его в правую часть равенства при верхней треугольной матрице $\boldsymbol{B}_2$ и вычисляя полученное выражение, находим первое приближение

$$\boldsymbol{x}^{(1)} = \boldsymbol{B}_1\boldsymbol{x}^{(0)} + \boldsymbol{B}_2\boldsymbol{x}^{(1)} \qquad (2.6)$$

Подставляя приближение $\boldsymbol{x}^{(1)}$, получим

$$\boldsymbol{x}^{(2)} = \boldsymbol{B}_1\boldsymbol{x}^{(1)} + \boldsymbol{B}_2\boldsymbol{x}^{(2)} \qquad (2.7)$$

Продолжая этот процесс далее, получим последовательность $\boldsymbol{x}^{(0)}$, $\boldsymbol{x}^{(1)}$, …, $\boldsymbol{x}^{(n)}$, … приближений к вычисляемых по формуле

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{B}_1{}^{(k+1)} + \boldsymbol{B}_2{}^{(k)} + \boldsymbol{c} \qquad (2.8)$$

или в развернутой форме записи

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

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

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

. . . . . .

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

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

$$x_i^{(k+1)} = x_1^{(k)} - a_{ii}^{-1}\left(\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} + \sum_{j=1}^{n} a_{ij}x_i^{(k)} - b_i\right). \qquad (2.9)$$

Необходимое и достаточное условие сходимости: $\rho(\alpha) < 1$, где — $\rho(\alpha)$ спектральный радиус α[2].

В математике , спектральный радиус квадратной матрицы или ограниченного линейного оператора является наибольшим абсолютным значением его собственных значений (т. е. супремум среди абсолютные значения элементов в его спектре). Иногда его обозначают через ρ (·)

Определение: ненулевой вектор $\vec{u}$, который при умножении на некоторую квадратную матрицу A превращается в самого же себя с числовым коэффициентом λ, называется **собственным вектором** матрицы A. Число λ называют **собственным значением** или **собственным числом** данной матрицы.

Задание

Методом простых итераций и методом Зейделя найти с точностью 0,0001 численное решение системы $\boldsymbol{Ax=b}$,

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 $\boldsymbol{b}= (1{,}2;\ 2{,}2;\ 4{,}0;\ 0{,}0;\ -1{,}2)^T$.

Вариант 3

Полученные результаты будем сверять с решением, полученным используя подмодуль numpy linalg:

```
x_real = np.linalg.solve(A, b)
print(f'Real Accurate solution *X*\n{x_real}')
```

Программная реализация: Метод простых итераций

```
def iteration_method(matrix_a, answers_b, error=0.0001, verbose=0):
    A, b = np.array(matrix_a, dtype=float), np.array(answers_b, dtype=float)
    ar_size = len(A)
    A, B = check_zeros_diag(A), np.empty(shape=A.shape)
    for i in range(ar_size):
        for j in range(ar_size):
            if i == j:
                B[i, j] = 0
            else:
                B[i, j] = (-1) * A[i, j] / A[i, i]
    c = np.empty(shape=b.shape[0])
    for i in range(len(c)):
        c[i] = b[i] / A[i, i]
    c = c.reshape((ar_size, 1))
    if check_convergence(B):
        if verbose == 1:
            print('X solution converges by spectrum')
    elif norm_convergence(B):
        if verbose == 1:
            print('X solution converges by its norm')
    else:
        raise Exception("Can't find roots, bcs not converges")
    errors_x = np.empty(shape=ar_size)
    e = 1
    x_current = c.copy()
    iteration = 0
    while e > error:
        iteration += 1
        x_prev = x_current.copy()
        for i in range(ar_size):
            x_current[i] = c[i]
            for j in range(ar_size):
                x_current[i][0] += B[i, j] * x_prev[j][0]
        for i in range(ar_size):         # calculate error
            errors_x[i] = abs(x_prev[i][0] - x_current[i][0])
        errors_y = abs(A.dot(x_current).reshape(ar_size, ) - b)
        e = np.amax(errors_x) + np.amax(errors_y)
    return x_current, iteration
```

Метод Зейделя

```
def zeidels_method(matrix_a, answers_b, error=0.0001, verbose=0):
    A, b = np.array(matrix_a, dtype=float), np.array(answers_b, dtype=float)
    ar_size = len(A)
    A, B = check_zeros_diag(A), np.empty(shape=A.shape)
    for i in range(ar_size):
        for j in range(ar_size):
            if i == j:
                B[i, j] = 0
            else:
                B[i, j] = (-1) * A[i, j] / A[i, i]
    c = np.empty(shape=b.shape[0])
    for i in range(len(c)):
        c[i] = b[i] / A[i, i]
    c = c.reshape((ar_size, 1))
    if check_convergence(B):
        if verbose == 1:
            print('X solution converges by spectrum')
    elif norm_convergence(B):
        if verbose == 1:
            print('X solution converges by its norm')
    else:
        raise Exception("Can't find roots, bcs not converges")
    x_current, e = np.zeros(shape=c.shape, dtype=float), 1
    x_current[0, 0] = c[0, 0]
    iteration = 0
    while e > error:
        iteration += 1
        error_x = 0  # set big value for error
        for i in range(ar_size):
            x_prev = x_current[i, 0]  # save prev to calc error
            x_current[i, 0] = c[i, 0]
            for j in range(ar_size):
                x_current[i][0] += B[i, j] * x_current[j][0]
            if abs(x_prev - x_current[i, 0]) > error_x:
                error_x = abs(x_prev - x_current[i, 0])
        errors_y = abs(A.dot(x_current).reshape((ar_size,)) - b)
        e = (np.amax(errors_y) + error_x) / 2
    return x_current, iteration
```

2. **Вспомогательные функции для подсчета норм и вычисления методов**

```
def norm_column(matrix_a):
    A = np.array(matrix_a, dtype=float)
    n = len(A)
    norms = np.zeros(shape=A.shape)
    for j in range(n):
        for i in range(n):
            norms[j] += abs(A[i, j])
    norm = np.amax(norms)
    return norm


def norm_row(matrix_a):
    A = np.array(matrix_a, dtype=float)
    n = len(A)
    norms = np.zeros(shape=A.shape)
    for i in range(n):
        for j in range(n):
            norms[i] += abs(A[i, j])
    norm = np.amax(norms)
    return norm


def norm_quad(matrix_a):
    A = np.array(matrix_a, dtype=float)
    n = len(A)
    quad = 0
    for i in range(n):
        for j in range(n):
            quad += abs(A[i, j]) ** 2
    quad = quad ** 1 / 2
    return quad
```

```python
def norm_trace(matrix_a):
    A = np.array(matrix_a, dtype=float)
    n = A.shape[0]
    for i in range(n):
        row_sum = 0
        col_sum = 0
        for j in range(n):
            if i == j:
                continue
            row_sum += abs(A[i, j])
            col_sum += abs(A[j, i])
        if not ((row_sum < abs(A[i, i])) or (col_sum < abs(A[i, i]))):
            return False
    return True


def norm_convergence(matrix_a):
    A = np.array(matrix_a, dtype=float)
    if (norm_column(A) < 1) or (norm_row(A) < 1) or (norm_quad(A) < 1):
        return True
    return False


def find_spectrum(matrix_a):
    A = np.array(matrix_a, dtype=float)
    eigen_values = np.linalg.eig(A)[0]
    max_eig = np.amax(abs(eigen_values))
    return max_eig


def check_convergence(matrix_a):
    A = np.array(matrix_a, dtype=float)
    if find_spectrum(A) < 1:
        return True
    return False
```

Полученные результаты

Задание.

```
Default matrix
  1.360000  0.210000  0.110000  0.120000 -0.130000 -->   1.200000
 -0.100000 -1.300000  0.050000  0.170000  0.120000 -->   2.200000
  0.120000 -0.100000 -1.300000  0.110000  0.110000 -->   4.000000
  0.170000  0.120000 -0.100000 -1.300000  0.110000 -->   0.000000
  0.110000  0.670000  0.120000 -0.100000 -1.300000 -->  -1.200000

Real Accurate solution *X*
[ 1.3642732  -1.89871732 -2.80609318  0.20083974 -0.21452747]
#####################################################
X solution converges by spectrum
#####################################################
----Iteration method result----
X = [[ 1.36427323 -1.89871728 -2.80609316  0.20083975 -0.21452751]]
 Done in 13 iterations
#####################################################
X solution converges by spectrum
#####################################################
----Zeidels method result----
X = [[ 1.36427324 -1.89871735 -2.80609319  0.20083973 -0.21452748]]
Done in 7 iterations
```

Тестовый пример 1.

Проверка наличия 0 на главной диагонали.

Исходная матрица:

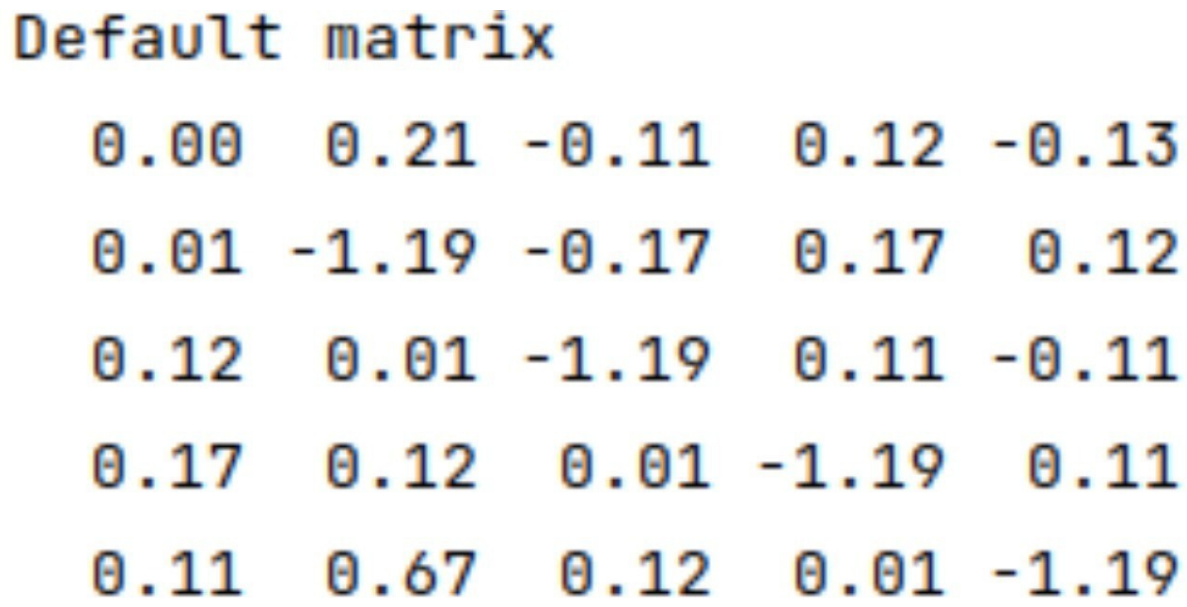
```
[[ 0.    2.33  0.24  0.67  0.92]
 [-0.53  0.    2.33  0.24  0.76]
 [-0.2   0.89  0.    0.77  0.23]
 [-0.87  0.26  3.45  0.   -1.05]
 [ 1.24  0.34 -0.2   0.89  0.  ]]
```

Матрица после функции check_zeros_diag:

```
[[ 0.92  0.    2.33  0.24  0.67]
 [ 0.76 -0.53  0.    2.33  0.24]
 [ 0.23 -0.2   0.89  0.    0.77]
 [-1.05 -0.87  0.26  3.45  0.  ]
 [ 0.    1.24  0.34 -0.2   0.89]]
```

Тестовый пример 2.

Исходная матрица:



Получен ответ

```
Exception: Can't find roots, bcs not converges
```

Данный результат мы можем наблюдать, так как не выполнилось необходимое условие сходимости данных методов.

Тестовый пример 3.

Исходная матрица:

```
Default matrix
  1.47  0.21 -0.11  0.12 -0.13 -->   1.20
  0.01 -1.19 -0.17  0.17  0.12 -->   2.20
  0.12  0.01 -1.19  0.11 -0.11 -->   4.00
  0.01  0.12  0.01 -1.19  0.11 -->   0.00
  0.11  0.67  0.12  0.01 -1.19 -->  -1.20

Real Accurate solution *X*
[ 0.77850599 -1.39778979 -3.30610471 -0.16602161 -0.04140864]
```

Метод простых итераций	Метод Зейделя	Вычисление при помощи встроенной функции
0.77851941 -1.39778476 -3.30612379 -0.1660089	0.7785198 -1.39779865 -3.30610091 -0.16602764	0.77850599 -1.39778979 -3.30610471 -0.16602161

-0.04140403	-0.04141202	-0.04140864
Количество итераций		
8	6	

Тестовый пример 4.

```
Default matrix
  2.39 -0.21  0.05  1.12 -1.13 -->   1.20
 -1.07 -2.27 -0.23  0.01  1.12 -->   2.20
  1.12 -1.07 -2.27 -0.11  0.05 -->   4.00
  0.17  1.12 -1.07 -2.27 -0.11 -->   0.00
 -0.11  0.67  1.12 -1.07 -2.27 -->  -1.20

Real Accurate solution *X*
[ 0.26404764 -1.15863571 -1.09258357 -0.01963189 -0.35595535]
```

Метод простых итераций	Метод Зейделя	Вычисление при помощи встроенной функции
0.26404612 -1.15862116 -1.09258157 -0.01963492 -0.355975	0.26406488 -1.15863686 -1.0925732 -0.01963709 -0.35594895	0.26404764 -1.15863571 -1.09258357 -0.01963189 -0.35595535
Количество итераций		
29	9	

Тестовый пример 5.

```
Default matrix
  9.20  2.50 -3.70 --> -17.50
  0.90  9.00  0.20 -->   4.40
  4.50 -1.60-10.30 --> -22.10
```

Метод простых итераций	Метод Зейделя	Вычисление при помощи встроенной функции
-1.50758347 0.60870472 1.39242074	-1.50759967 0.60870673 1.39241463	-1.50758786 0.60870501 1.39242006
Количество итераций		
15	8	

Тестовый пример 6.

```
Default matrix
 10.00  1.00  1.00 -->  12.00
  2.00 10.00  1.00 -->  13.00
  2.00  2.00 10.00 -->  14.00

Real Accurate solution *X*
[1. 1. 1.]
```

Метод простых итераций	Метод Зейделя
0.99999717 0.99999644 0.99999551	0.99999958 0.9999996 1.00000016
Количество итераций	
9	5

Тестовый пример 7.

```
Default matrix
 10.00  1.00 -1.00 -->  11.00
  2.70 10.00 -1.00 -->  10.00
 -1.00  1.00 10.00 -->  10.00

Real Accurate solution *X*
[1.1232369  0.79995882 1.03232781]
```

Метод простых итераций	Метод Зейделя
1.12323285 0.79996262 1.03232375	1.12323655 0.79995888 1.03232777
Количество итераций	
6	5

Оценка

```
def check_accuracy(w, x):
    for i in range(len(x)):
        if (abs(x[i] - w[i]) / abs(x[i])) >= 0.0001:
            return False
        else:
            print(f"|{x[i]} - {w[i]}| < 0.0001")
    return True
```

Следующая функция производит проверку на допустимость погрешности.

```
|1.3642731969437705 - [1.36427323]| < 0.0001
|-1.8987173225584448 - [-1.89871728]| < 0.0001
|-2.8060931803055285 - [-2.80609316]| < 0.0001
|0.20083973937425273 - [0.20083975]| < 0.0001
|-0.21452746924956326 - [-0.21452751]| < 0.0001
```

Выводы

Таким образом, в ходе выполнения лабораторной работы я изучил итерационные методы решения СЛАУ (метод простых итераций, метод Зейделя), составил мысленный алгоритм решения СЛАУ указанными методами, применимый для организации вычислений на ЭВМ, составил программу решения СЛАУ по разработанному алгоритму, численно решить тестовые примеры и проверил правильность работы программы, сравнили трудоемкость решения методом простых итераций и методом Зейделя.

На основании тестов были получены следующие результаты:

Метод простых итераций как и ожидалось длиннее, чем метод Зейделя

Оба метода могут получить решение с заданной точностью

СЛАУ как и данные методы чувствительны к изменению(даже малому) начальных значений.

Данные методы не всегда могут решить СЛАУ