МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «МЭИ»

КАФЕДРА МАТЕМАТИЧЕСКОГО И КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ

ЧИСЛЕННЫЕ МЕТОДЫ

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ ПРЯМЫМИ МЕТОДАМИ. ТЕОРИЯ ВОЗМУЩЕНИЙ»

Вариант 47

Студент: Жарова С.П.

Группа: А-16-22

Преподаватель: Амосова О. А.

Москва

2024

Лабораторная работа №3

Жарова Светлана А-16-22

Вариант №47

Задача 3.1.

Реализовать решение СЛАУ с помощью LU разложения и LU разложения по схеме частичного выбора. Решить систему небольшой размерности с возмущенной матрицей обоими методами, оценить погрешность и сравнить с теоретической оценкой. Проанализировать поведение методов с ростом числа уравнений

$$(47+4) \bmod 2 = 1$$

- решение с помощью **LU** модифицирует исходную матрицу A; решение с помощью **LU** по схеме частичного выбора реализовано в виде двух функций, одна из которых возвращает две матрицы – **L** и **U**, не модифицируя **A**, а вторая функция решает систему.

$$(47+4) \bmod 5 = 1$$

$$A_{ij} = \frac{1}{70-3i-j}$$

In [6]:

```
import numpy as np
np.isinf
np.isfinite
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator as ML
from numpy.linalg import inv
```

LU разложение:

In [7]:

```
n = 5

A = np.array([[1/(70-3*i - j) for j in range(n)] for i in range(n)])
x_N = np.array([47]*n)
b = np.dot(A, x_N)
print(A)
print(b)
```

```
[[0.01428571 0.01449275 0.01470588 0.01492537 0.01515152]
 [0.01492537 0.01515152 0.01538462 0.015625   0.01587302]
 [0.015625   0.01587302 0.01612903 0.01639344 0.01666667]
 [0.01639344 0.01666667 0.01694915 0.01724138 0.01754386]
 [0.01724138 0.01754386 0.01785714 0.01818182 0.01851852]]
[3.45737821 3.61709742 3.7922964  3.98534154 4.19910777]
```

In [8]:

```
def LU_simple(A, b, n):
    m = np.shape(0)
    for j in range(n - 1):
        for i in range(j + 1, n):
```

```
            m = A[i, j] / A[j, j]
            A[i, j::] = A[i, j::] - m*A[j, j::]
            A[i, j] = m
        #L[j, j] = 1
    #L[n-1, n-1] = 1
    y = np.zeros(n)
    for i in range(0, n):
        y[i] = b[i] - np.sum(A[i, 0:i+1]*y[0:i+1])
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        x[i] = y[i] - np.sum(A[i,i+1::]*x[i+1::])
        x[i] /= A[i, i]
    #print('модифицированная матрица A\n', A)
    #print('вектор решение:')
    return x
```

In [9]:

```
x = LU_simple(A, b, n)
print(A)
print(x)
```

```
[[1.42857143e-02 1.44927536e-02 1.47058824e-02 1.49253731e-02
  1.51515152e-02]
 [1.04477612e+00 9.83226162e-06 2.02606875e-05 3.13265761e-05
  4.30746699e-05]
 [1.09375000e+00 2.19345238e+00 3.25812707e-08 1.02404401e-07
  2.14732358e-07]
 [1.14754098e+00 3.62459016e+00 3.47296471e+00 2.96702068e-10
  1.26614249e-09]
 [1.20689655e+00 5.35027223e+00 8.10163339e+00 4.92002720e+00
  5.96692577e-12]]
[47.0004705  46.99823573 47.00247824 46.99845454 47.000361  ]
```

1. Реализовать метод решения СЛАУ с помощью **LU** разложения по схеме частичного выбора в виде, указанном в приложении. Убедиться в его работоспособности.

In [10]:

```
import numpy as np
from numpy.linalg import inv

def meac(A, n, count):
    max_val = abs(A[count, count])
    k = count
    for i in range(count, n):
        if abs(A[i, count]) > max_val:
            max_val = abs(A[i, count])
            k = i
    return k

def LUP1(A, n):
    L = np.eye(n)
    P = np.eye(n)
    U = np.copy(A)
    for j in range(n - 1):
        k = meac(U, n, j)
        U[[j, k]] = U[[k, j]]
        L[[j, k], :j] = L[[k, j], :j]
        P[[j, k]] = P[[k, j]]
        for i in range(j + 1, n):
            L[i, j] = U[i, j] / U[j, j]
            U[i, j:] -= L[i, j] * U[j, j:]
    return L, U, P

def LUP(A, b):
    n = len(A)
    L, U, P = LUP1(A, n)
    Pb = np.dot(P, b)
    y = np.zeros(n)
```

```
    for i in range(n):
        y[i] = Pb[i] - np.dot(L[i, :i], y[:i])
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (y[i] - np.dot(U[i, i + 1:], x[i + 1:])) / U[i, i]
    return x

A = np.array([[1/(70 - 3*i - j) for j in range(5)] for i in range(5)])


x = LUP(A, b)
print("Solution:", x)
print(A)
print(b)
```

```
Solution: [46.99954006 47.00172627 46.99757284 47.00151509 46.99964573]
[[0.01428571 0.01449275 0.01470588 0.01492537 0.01515152]
 [0.01492537 0.01515152 0.01538462 0.015625   0.01587302]
 [0.015625   0.01587302 0.01612903 0.01639344 0.01666667]
 [0.01639344 0.01666667 0.01694915 0.01724138 0.01754386]
 [0.01724138 0.01754386 0.01785714 0.01818182 0.01851852]]
[3.45737821 3.61709742 3.7922964  3.98534154 4.19910777]
```

In [11]:

```
print(LUP1(A, n))
```

```
(array([[ 1.        ,  0.        ,  0.        ,  0.        ,  0.        ],
       [ 0.82857143,  1.        ,  0.        ,  0.        ,  0.        ],
       [ 0.90625   ,  0.59895833,  1.        ,  0.        ,  0.        ],
       [ 0.95081967,  0.32991803,  0.86824118,  1.        ,  0.        ],
       [ 0.86567164,  0.81919946,  0.65229099, -0.68084626,  1.        ]]), array([[ 1.7
2413793e-02,  1.75438596e-02,  1.78571429e-02,
         1.81818182e-02,  1.85185185e-02],
       [ 0.00000000e+00, -4.35872289e-05, -9.00360144e-05,
        -1.39561931e-04, -1.92400192e-04],
       [ 0.00000000e+00, -3.38813179e-21, -7.56350927e-08,
        -2.38322969e-07, -5.01042168e-07],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        -1.72626642e-10, -7.38583059e-10],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00, -8.25719244e-13]]), array([[0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0.]]))
```

1. Решить систему $A^*x = b$, размера $5x5$, двумя методами. Вектор b задается как $b = Ax$, где $x_i = N$, N - номер варианта. Матрицу A^*_{ij} задать как A_{ij} и к одному элементу прибавить 10^{-3}.

In [12]:

```
A0 = np.copy(A)
A0[0, 1] = A[0, 1] + 10**(-3)
print(A0)
A1 = np.copy(A0)
b1 = np.copy(b)
```

```
[[0.01428571 0.01549275 0.01470588 0.01492537 0.01515152]
 [0.01492537 0.01515152 0.01538462 0.015625   0.01587302]
 [0.015625   0.01587302 0.01612903 0.01639344 0.01666667]
 [0.01639344 0.01666667 0.01694915 0.01724138 0.01754386]
 [0.01724138 0.01754386 0.01785714 0.01818182 0.01851852]]
```

In [16]:

```
x0 = LU_simple(A0, b, n)
print(x0)
```

```
[ 3.45281002e+00  4.36591538e-03  8.33009681e-03 -9.91515816e-04
  2.24919883e+02]
```

2.24919883e+02]

In [15]:

```
x = LUP(A0, b)
print(A0)
print()
print("Solution:", x)
```

```
[[ 1.42857143e-02  1.54927536e-02  1.47058824e-02  1.49253731e-02
   1.51515152e-02]
 [ 1.04477612e+00 -1.03494386e-03  2.02606875e-05  3.13265761e-05
   4.30746699e-05]
 [ 1.09375000e+00  1.03598219e+00  2.34837232e-05  3.63619825e-05
   5.00723789e-05]
 [ 1.14754098e+00  1.07436076e+00  2.20504025e+00  6.62643297e-08
   1.85703323e-07]
 [ 1.20689655e+00  1.11531777e+00  3.66496615e+00  3.50551940e+00
   6.23542263e-10]]

Solution: [ 3.45281002e+00  4.36591538e-03  8.33009681e-03 -9.91515816e-04
  2.24919883e+02]
```

Заметим, что найденное значение не совпадает с исходным. Связано это с тем, что матрица плохо обусловленна.

1. Вычислить погрешность и сравнить ее с теоретической оценкой. Для вычисления обратной матрицы можно воспользоваться встроенными функциями.

In [17]:

```
import numpy as np
from scipy import linalg
n = 5

A = np.array([[1/(70-3*i - j) for j in range(n)] for i in range(n)])
x_N = np.array([47]*n)
b = np.dot(A, x_N)
print(A)
print(b)

A0 = np.copy(A)
A0[0, 0] += 10e-3

x0 = linalg.solve(A0, b)
x1 = LU_simple(A, b, n)
x2 = LUP(A, b)

def dx(x_res, x):
    return linalg.norm(x_res - x, ord=2) / linalg.norm(x, ord=2)

def teor(A_res, A):
    return linalg.norm(A_res - A, ord=2) * np.linalg.cond(A) / linalg.norm(A)
```

```
[[0.01428571 0.01449275 0.01470588 0.01492537 0.01515152]
 [0.01492537 0.01515152 0.01538462 0.015625   0.01587302]
 [0.015625   0.01587302 0.01612903 0.01639344 0.01666667]
 [0.01639344 0.01666667 0.01694915 0.01724138 0.01754386]
 [0.01724138 0.01754386 0.01785714 0.01818182 0.01851852]]
[3.45737821 3.61709742 3.7922964  3.98534154 4.19910777]
```

In [18]:

```
print('Погрешность решение                                :', dx(x0, x_N))
print('Погрешность решение LU                             :', dx(x1, x_N))
print('Погрешность решение LU по схеме частичного выбора:', dx(x2, x_N))
print('Теоретическая оценка погрешности                   :', teor(A0, A))
print('Число обусловленности матрицы A                    :', np.linalg.cond(A))
```

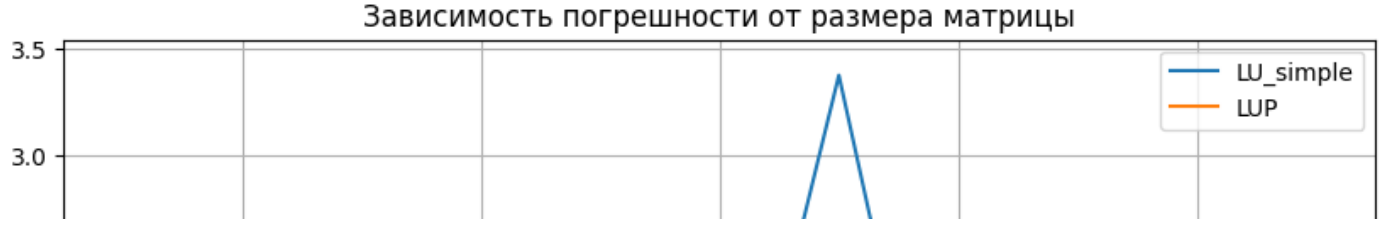
```
Погрешность решение                                : 3.2911686911113236
```

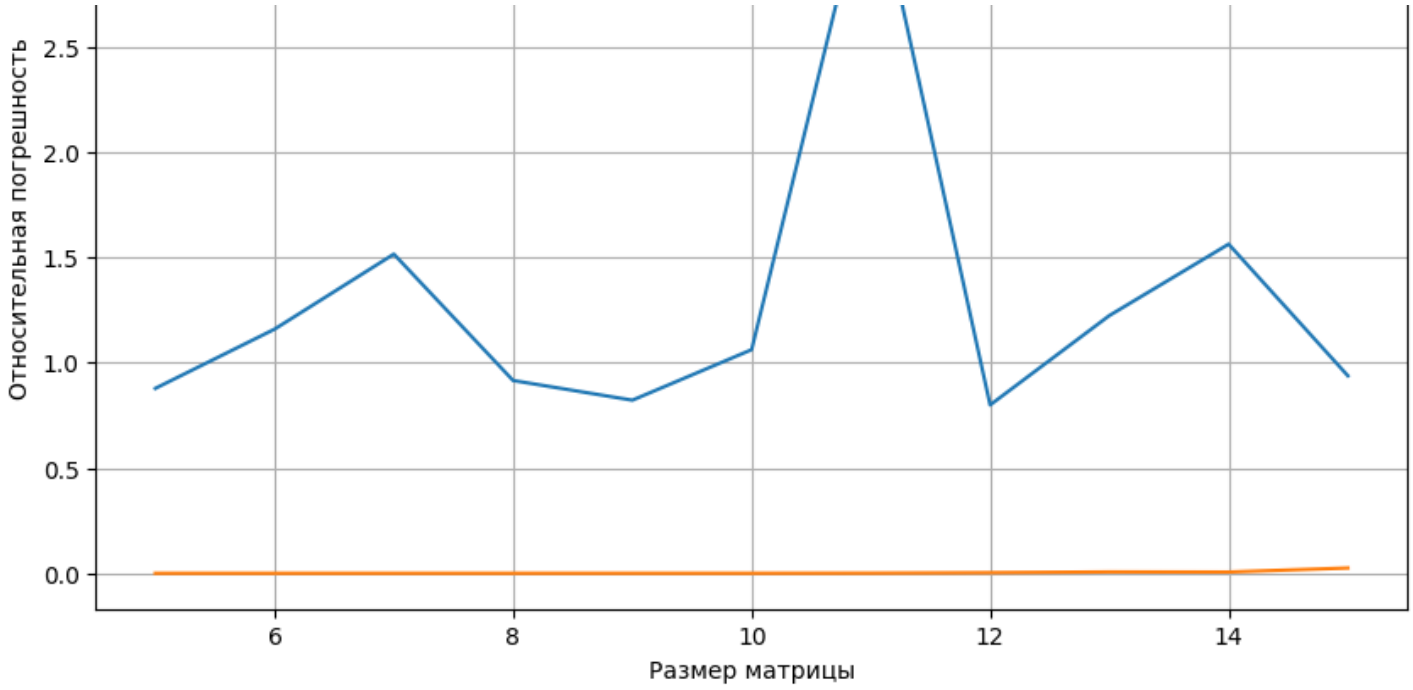
```
Погрешность решение LU                                : 3.295393616171722e-05
Погрешность решение LU по схеме частичного выбора: 1.9073179826863467
Теоретическая оценка погрешности                      : 757.0897445928352
Число обусловленности матрицы A                       : 788.3771126376915
```

1. Задавая вектор b как $b = Ax$, где $x_i = N$, решить систему обоими методам для размера матрицы $n = 5, . 6, . . 15$

In [28]:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import norm, cond, inv

def relative_error(x_res, x):
    return norm(x_res - x, ord=2) / norm(x, ord=2)

# Размеры матрицы от 5 до 15
sizes = range(5, 16)

# Число для компонент вектора x
N = 10

# Списки для хранения погрешностей
errors_lu_simple = []
errors_lup = []

# Генерация вектора x
x = np.array([N] * 5)

# Вычисление и сохранение погрешностей для каждого размера матрицы
for n in sizes:
    # Генерация матрицы A
    A = np.array([[1/(70 - 3*i - j) for j in range(n)] for i in range(n)])
    A0 = np.array([[1/(70-3*i - j) for j in range(n)] for i in range(n)])
    A0[0][1] = A[0][1] + 10**(-3)
    x = [N for i in range(n)]
    b = np.dot(A, x)
    # Решение системы методом LU_simple
    x_lu_simple = LU_simple(A, b, n)
    x_lu_simple1 = LU_simple(A0, b, n)
    # Решение системы методом LUP
    x_lup = LUP(A, b)
    x_lup1 = LUP(A0, b)
    # Вычисление погрешностей
    error_lu_simple = relative_error(x_lu_simple1, x_lu_simple)
    error_lup = relative_error(x_lup1, x_lup)
    # Добавление погрешностей в списки
    errors_lu_simple.append(error_lu_simple)
    errors_lup.append(error_lup)

# Построение графика
plt.figure(figsize=(10, 6))
plt.plot(sizes, errors_lu_simple, label='LU_simple')
plt.plot(sizes, errors_lup, label='LUP')
plt.xlabel('Размер матрицы')
plt.ylabel('Относительная погрешность')
plt.title('Зависимость погрешности от размера матрицы')
plt.legend()
plt.grid(True)
plt.show()
```





Большое значение погрешности в **LU** разложении можно объяснить тем, что матрица **A** плохо обусловлена, в следствие чего производится большее число операций, и полученные значения будут все больше и больше отличаться от точного решения.

3.2.

Дана разреженная матрица **A**. Найти число обусловленности матрицы.

1) Задание матрицы A и вектора **b**

In [2]:

```
import numpy as np

# Значение N
N = 47
m = 20
D = N * m + m / N
# Создание матрицы A с помощью списка списков
A = np.array([
    [D, N, N, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, D, N, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, D, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, N, D, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, N, N, D, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, N, N, 0, D, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, N, N, 0, 0, D, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, N, N, 0, 0, 0, D, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, N, N, N, N, D, N, N, N, N, N, N, N, N, N, N, N],
    [0, 0, 0, 0, N, N, N, N, 0, D, N, N, N, N, N, N, N, N, N, N],
    [0, 0, 0, 0, N, N, N, N, 0, 0, D, N, N, N, N, N, N, N, N, N],
    [0, 0, 0, 0, N, N, N, N, 0, 0, 0, D, N, N, N, N, N, N, N, N],
    [0, 0, 0, 0, N, N, N, N, 0, 0, 0, 0, D, N, N, N, N, N, N, N],
    [0, 0, 0, 0, N, N, N, N, 0, 0, 0, 0, 0, D, N, N, N, N, N, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, D, N, N, N, N, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, 0, D, N, N, N, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, 0, 0, D, N, N, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, D, N, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, D, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, D]
])

x_N = np.array([47]*m)
b = np.dot(A, x_N)
print(A)
```

```
print(b)
```

```
[[940.42553191  47.          47.          47.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.         940.42553191  47.          47.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.         940.42553191  47.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.          47.         940.42553191   0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.          47.          47.         940.42553191
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.          47.          47.           0.
  940.42553191   0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.          47.          47.           0.
    0.         940.42553191   0.          0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.          47.          47.           0.
    0.           0.         940.42553191   0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [  0.           0.          0.           0.          47.
   47.          47.          47.         940.42553191  47.
   47.          47.          47.          47.          47.
   47.          47.          47.          47.          47.        ]
 [  0.           0.          0.           0.          47.
   47.          47.          47.           0.         940.42553191
   47.          47.          47.          47.          47.
   47.          47.          47.          47.          47.        ]
 [  0.           0.          0.           0.          47.
   47.          47.          47.           0.           0.
  940.42553191  47.          47.          47.          47.
   47.          47.          47.          47.          47.        ]
 [  0.           0.          0.           0.          47.
   47.          47.          47.           0.           0.
    0.         940.42553191  47.          47.          47.
   47.          47.          47.          47.          47.        ]
 [  0.           0.          0.           0.          47.
   47.          47.          47.           0.           0.
    0.           0.         940.42553191  47.          47.
   47.          47.          47.          47.          47.        ]
 [  0.           0.          0.           0.          47.
   47.          47.          47.           0.           0.
    0.           0.          0.         940.42553191  47.
   47.          47.          47.          47.          47.        ]
 [ 47.          47.          47.          47.          47.
   47.          47.          47.           0.           0.
    0.           0.          0.           0.         940.42553191
   47.          47.          47.          47.          47.        ]
 [ 47.          47.          47.          47.          47.
   47.          47.          47.           0.           0.
    0.           0.          0.           0.           0.
  940.42553191  47.          47.          47.          47.        ]
 [ 47.          47.          47.          47.          47.
   47.          47.          47.           0.           0.
    0.           0.          0.           0.           0.
    0.         940.42553191  47.          47.          47.        ]
 [ 47.          47.          47.          47.          47.
   47.          47.          47.           0.           0.
```

```
    0.          0.          0.          0.          0.
    0.          0.        940.42553191  47.         47.        ]
 [ 47.         47.         47.         47.         47.
   47.         47.         47.          0.          0.
    0.          0.          0.          0.          0.
    0.          0.          0.        940.42553191  47.        ]
 [ 47.         47.         47.         47.         47.
   47.         47.         47.          0.          0.
    0.          0.          0.          0.          0.
    0.          0.          0.          0.        940.42553191]]
[50827. 50827. 50827. 50827. 53036. 53036. 53036. 53036. 77335. 75126.
 72917. 70708. 68499. 66290. 72917. 70708. 68499. 66290. 64081. 61872.]
```

2) Разработать и реализовать алгоритм решения системы с данной матрицей A прямым методом с учетом нулевых элементов. Произвести тестирование программы. Я выбрала метод Гаусса:

In [3]:

```
import numpy as np

def gauss_elimination(A, b):
    n = len(b)
    # Прямой ход метода Гаусса
    for i in range(n):
        # Пропускаем нулевые элементы
        if A[i][i] == 0:
            continue

        for j in range(i+1, n):
            ratio = A[j][i] / A[i][i]
            # Пропускаем нулевые элементы
            if ratio == 0:
                continue

            for k in range(i, n):
                A[j][k] -= ratio * A[i][k]
            b[j] -= ratio * b[i]

    # Обратный ход метода Гаусса
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        # Пропускаем нулевые элементы
        if A[i][i] == 0:
            continue

        x[i] = b[i] / A[i][i]
        for j in range(i-1, -1, -1):
            # Пропускаем нулевые элементы
            if A[j][i] == 0:
                continue
            b[j] -= A[j][i] * x[i]
            A[j][i] = 0  # Обнуляем элементы, чтобы избежать вычислений над ними в дальн
ейших итерациях

    return x

solution = gauss_elimination(A, b)
print("Решение:", solution)
```

```
Решение: [47. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47.
 47. 47.]
```

3. Найти обратную матрицу

Также используем метод Гаусса:

In [5]:

```
import numpy as np
```

```python
def invers():
    columns = []
    for i in range(20):
        ident_row = np.array([1 if i==j else 0 for j in range(20)], dtype=float)
        columns.append(gauss_elimination(A, ident_row))
    return np.column_stack(columns)

inverse_A = invers()
print(inverse_A)
```

```
[[0.00106335 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.00106601 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.00106843 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.00107064 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.00106335 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.00106335
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.00106335 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.00106335 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.00106335 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.00106335 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.00106335 0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.00106335
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.00106335 0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.00106335 0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.00106335 0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
```

```
  0.          0.          0.          0.00106335  0.          0.
  0.          0.        ]
 [0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.00106335  0.
  0.          0.        ]
 [0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.00106335
  0.          0.        ]
 [0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.00106335  0.        ]
 [0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.00106335]]
```

4. Найти число обусловленности матрицы, используя обратную матрицу

In [6]:

```
def matrix_condition_number(A):
    norm_A = np.linalg.norm(A)
    norm_inverse_A = np.linalg.norm(inverse_A)
    condition_number = norm_A * norm_inverse_A
    return condition_number

condition_number = matrix_condition_number(A)

print("Число обусловленности матрицы A:", condition_number)
```

```
Число обусловленности матрицы A: 20.000131551853702
```

5. Ответить на вопросы: 1. Является ли матрица хорошо обусловленной? 2. Какова трудоемкость полученного метода?

1. Да, так как **condA < 100**
2. Трудоёмкость метода Гаусса для нахождения обратной матрицы A^{-1} в общем случае составляет примерно $\frac{2}{3}n^3$ операций умножения и деления, где **n -** размерность матрицы **A.** Это количество операций включает в себя прямой проход метода Гаусса для приведения матрицы **A** к верхнетреугольному виду и последующий обратный ход для получения обратной матрицы. Каждый из этих этапов включает в себя выполнение примерно $\frac{1}{3}n^3$ операций умножения и деления.

3.3 Дана система уравнений с матрицей системы A из задачи 3.2 и вектором b. Решить систему уравнений методом Якоби с точностью 10^{-10}.

In [13]:

```
import numpy as np

N = 47
m = 20
D = N * m + m / N
# Создание матрицы A с помощью списка списков
A = np.array([
    [D, N, N, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, D, N, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, D, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, N, D, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, N, N, D, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, N, N, 0, D, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, N, N, 0, 0, D, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [N, N, N, N, 0, 0, 0, D, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
    [0, 0, 0, 0, N, N, N, N, D, N, N, N, N, N, N, N, N, N, N, N],
    [0, 0, 0, 0, N, N, N, N, 0, D, N, N, N, N, N, N, N, N, N, N],
    [0, 0, 0, 0, N, N, N, N, 0, 0, D, N, N, N, N, N, N, N, N, N],
    [0, 0, 0, 0, N, N, N, N, 0, 0, 0, D, N, N, N, N, N, N, N, N],
    [0, 0, 0, 0, N, N, N, N, 0, 0, 0, 0, D, N, N, N, N, N, N, N],
    [0, 0, 0, 0, N, N, N, N, 0, 0, 0, 0, 0, D, N, N, N, N, N, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, D, N, N, N, N, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, 0, D, N, N, N, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, 0, 0, D, N, N, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, D, N, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, D, N],
    [N, N, N, N, N, N, N, N, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, D]
])

b = np.array([(abs(N - 25) + 5) * i * N for i in range(1, 21)])

print("Матрица A:")
print(A)
print("\nВектор b:")
print(b)
```

```
Матрица A:
[[940.42553191  47.          47.          47.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.         940.42553191  47.          47.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.         940.42553191  47.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.          47.         940.42553191   0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.          47.          47.         940.42553191
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.          47.          47.           0.
  940.42553191   0.          0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.          47.          47.           0.
    0.         940.42553191   0.           0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [ 47.          47.          47.          47.           0.
    0.           0.         940.42553191   0.           0.
    0.           0.          0.           0.           0.
    0.           0.          0.           0.           0.        ]
 [  0.           0.          0.           0.          47.
   47.          47.          47.         940.42553191  47.
   47.          47.          47.          47.          47.
   47.          47.          47.          47.          47.        ]
 [  0.           0.          0.           0.          47.
   47.          47.          47.           0.         940.42553191
   47.          47.          47.          47.          47.
   47.          47.          47.          47.          47.        ]
 [  0.           0.          0.           0.          47.
   47.          47.          47.           0.           0.
  940.42553191  47.          47.          47.          47.
   47.          47.          47.          47.          47.        ]
 [  0.           0.          0.           0.          47.
   47.          47.          47.           0.           0.
    0.         940.42553191  47.          47.          47.
   47.          47.          47.          47.          47.        ]
 [  0.           0.          0.           0.          47.
   47.          47.          47.           0.           0.
```

```
    0.          0.        940.42553191  47.          47.
   47.         47.         47.          47.          47.        ]
 [  0.          0.          0.           0.          47.
   47.         47.         47.           0.           0.
    0.          0.          0.         940.42553191  47.
   47.         47.         47.          47.          47.        ]
 [ 47.         47.         47.          47.          47.
   47.         47.         47.           0.           0.
    0.          0.          0.           0.         940.42553191
   47.         47.         47.          47.          47.        ]
 [ 47.         47.         47.          47.          47.
   47.         47.         47.           0.           0.
    0.          0.          0.           0.           0.
  940.42553191  47.         47.          47.          47.        ]
 [ 47.         47.         47.          47.          47.
   47.         47.         47.           0.           0.
    0.          0.          0.           0.           0.
    0.        940.42553191  47.          47.          47.        ]
 [ 47.         47.         47.          47.          47.
   47.         47.         47.           0.           0.
    0.          0.          0.           0.           0.
    0.          0.        940.42553191  47.          47.        ]
 [ 47.         47.         47.          47.          47.
   47.         47.         47.           0.           0.
    0.          0.          0.           0.           0.
    0.          0.          0.         940.42553191  47.        ]
 [ 47.         47.         47.          47.          47.
   47.         47.         47.           0.           0.
    0.          0.          0.           0.           0.
    0.          0.          0.           0.         940.42553191]]

Вектор b:
[ 1269  2538  3807  5076  6345  7614  8883 10152 11421 12690 13959 15228
 16497 17766 19035 20304 21573 22842 24111 25380]
```

2. Преобразование системы к виду, удобному для итераций метода Якоби.

In [8]:

```
# Найдем диагональные элементы матрицы A
diagonal_elements = np.diag(A)

# Создадим диагональную матрицу D
D = np.diag(diagonal_elements)

# Вычислим матрицу B и вектор c
B = np.eye(len(A)) - np.linalg.inv(D) @ A
c = np.linalg.inv(D) @ b

print("Матрица B:")
print(B)
print("\nВектор c:")
print(c)
```

```
Матрица B:
[[ 1.11022302e-16 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-4.99773756e-02  1.11022302e-16 -4.99773756e-02 -4.99773756e-02
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-4.99773756e-02 -4.99773756e-02  1.11022302e-16 -4.99773756e-02
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-4.99773756e-02 -4.99773756e-02 -4.99773756e-02  1.11022302e-16
```

```
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[-4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  1.11022302e-16  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[-4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  1.11022302e-16  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[-4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  0.00000000e+00  1.11022302e-16  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[-4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  0.00000000e+00  0.00000000e+00  1.11022302e-16
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  1.11022302e-16 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  1.11022302e-16 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  0.00000000e+00  1.11022302e-16 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  0.00000000e+00  0.00000000e+00  1.11022302e-16
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  1.11022302e-16 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  1.11022302e-16 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02]
[-4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  1.11022302e-16 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02]
[-4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  1.11022302e-16
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02]
[-4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  1.11022302e-16 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02]
[-4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
```

```
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  1.11022302e-16 -4.99773756e-02 -4.99773756e-02]
 [-4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  1.11022302e-16 -4.99773756e-02]
 [-4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
  -4.99773756e-02 -4.99773756e-02 -4.99773756e-02 -4.99773756e-02
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  1.11022302e-16]]

Вектор c:
[ 1.34938914  2.69877828  4.04816742  5.39755656  6.7469457   8.09633484
  9.44572398 10.79511312 12.14450226 13.4938914  14.84328054 16.19266968
 17.54205882 18.89144796 20.2408371  21.59022624 22.93961538 24.28900452
 25.63839367 26.98778281]
```

Реализуем метод Якоби, проверяя условие сходимости. Находим решение с заданной точностью.

In [9]:

```python
import numpy as np

def jacobi_method(A, b, x0, tol=1e-10, max_iter=50):
    n = len(A)
    x = x0.copy()
    x_prev = x0.copy()
    iter_count = 0
    residual_norms = []

    while iter_count < max_iter:
        for i in range(n):
            sum_term = np.dot(A[i, :], x_prev) - A[i, i] * x_prev[i]
            x[i] = (b[i] - sum_term) / A[i, i]

        residual = np.linalg.norm(A @ x - b)
        residual_norms.append(residual)

        if residual < tol:
            break

        x_prev = x.copy()
        iter_count += 1

    return x, iter_count, residual_norms

N = 47

# Выбираем начальное приближение
x0 = np.zeros_like(b)

# Вызываем метод Якоби
solution, num_iterations, residual_norms = jacobi_method(A, b, x0)

# Выводим результаты
print("Решение:", solution)
print("Количество итераций:", num_iterations)
print("Евклидова норма вектора невязки на каждой итерации:", residual_norms)
```

```
Решение: [ 0  2  3  5  6  7  8 10  3  4  6  8  9 11 13 15 17 19 22 24]
Количество итераций: 50
Евклидова норма вектора невязки на каждой итерации: [26354.86908612627, 8403.317335240808
, 2362.7385040603785, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.07508
8381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618
.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152
, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088
381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.
075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152,
2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.07508838
```

1152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.07
5088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2
618.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381152, 2618.075088381
152, 2618.075088381152]

Начиная с **4** итерации норма вектора невязки не изменяет своего значения. В случае методов решения систем линейных уравнений, таких как метод Гаусса-Зейделя или метод простой итерации (метод Якоби), вектор невязки обычно уменьшается на каждой итерации и сходится к нулю по мере приближения к решению. Однако на практике он может оставаться примерно постоянным после определенного количества итераций, что указывает на достижение сходимости. Таким образом, совпадение вектора невязки на последовательных итерациях может означать, что алгоритм сойдется к решению системы линейных уравнений или приблизился к этому решению с заданной точностью.