**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Факультет прикладной математики и информатики

Кафедра вычислительной математики

Жуковский Павел Сергеевич

Отчёт по лабораторной работе №3, вариант 5

(«Методы вычислений»)

Студента 2 курса 13 группы

Преподаватель

Бондарь Иван Васильевич

Минск 2020

**Вариант:**

## 5. Задание 2 (5) + Задание 5 (метод 2, задача 4 B)

**Задание 2:**

**Задание 2. Метод релаксации 1**

Дана матрица $A$ (указана в варианте, см. список 1 ниже).

1. Написать программу, которая решает СЛАУ $Ax = b$ методом релаксации (в качестве вектора $b$ взять вектор, соответствующий какому-нибудь заданному значению $x$). Экспериментально подобрать значение параметра $\omega$, при котором итерационный процесс сходится $\omega_1$), а также значение, при котором он расходится ($\omega_0$).
2. Путем теоретического анализа подтвердить сходимость и расходимость.
3. Построить логарифмическую диаграмму сходимости (совмещенную) для $\omega = \omega_0, \omega_1, \omega = 1$ и еще двух любых значений от 0 до 2.

Матрица:

5. $\begin{pmatrix} -1 & 1 & -1 \\ -1 & 4 & -2 \\ 2 & -3 & 5 \end{pmatrix}$

Итак, я написал программу на языке Python, которая решает СЛАУ Ax = b методом релаксации, где в качестве вектора b я взял вектор (-7, 7, 28), который соответствует решению x = (7, 7, 7). Матрица A мне была дана по условию (картинка с ней прикреплена выше). Я также выбрал вот такое начальное приближение: $x_0$ = (0, 0, 0).

Мне требовалось экспериментально подобрать значение параметра w (он же параметр релаксации), при котором итерационный процесс сходится ($w_1$), а также значение, при котором он расходится ($w_0$).

Для того, чтобы вообще определить, сходится или расходится процесс, я для начала проверял, не меньше ли единицы произведение норм двух частей матрицы B. Если не меньше единицы, ты я смотрел на норму самой матрицы B, которую получал путем произведения двух её частей (что это за части, будет видно в коде, где и реализован алгоритм релаксации). А если даже норма матрицы B не меньше единицы, то я искал собственные значения этой матрицы, выбирал из них наибольшее по модулю и смотрел, не меньше ли единицы оно. Если оно меньше единицы, то процесс сходится, а если – нет, то можно однозначно сказать, что процесс расходится.

Итак, вот исходный код моей программы:

```
import matplotlib.pyplot as plt
import numpy as np
import time

# Лабораторная №3, Вариант №5, Задание №2, Матрица №5
# Задание №2. Метод релаксации 1
# Дана матрица A (указана в варианте, см. список 1 ниже).
```

```python
# 1. Написать программу, которая решает СЛАУ Ax = b методом релаксации (в качестве вектора b взять вектор,
# соответствующий какому-нибудь заданному значению x). Экспериментально подобрать значение параметра w, при котором
# итерационный процесс сходится w1), а также значение, при котором он расходится (w0).
# 2. Путем теоретического анализа подтвердить сходимость и расходимость.
# 3. Построить логарифмическую диаграмму сходимости (совмещенную) для w = w0, w1, w = 1 и еще двух любых значений от 0
# до 2.

# Номер итерационного процесса
ProcessNum = 1

# Требуемая точность (для итераций)
Epsilon = 0.00000001

# Размерность матрицы
N = 3

# Заданная матрица A
A = np.array([[-1., 1., -1.],
              [-1., 4., -2.],
              [2., -3., 5.]])

# Возьмём начальное приближение (0, 0, 0):
X0 = np.array([0., 0., 0.])

# В качестве вектора-ответа возьмём вектор (7, 7, 7), тогда вектор b = A*(7, 7, 7) будет таким:
b = np.array([-7., 7., 28.])

# Значения w для экспериментов (w0 - не сходится, w1 - сходится, w2 - единица, w3 и w4 - любые от 0 до 2)
w0 = 2.5
w1 = 1.5
w2 = 1.0
w3 = 0.5
w4 = 0.1

#  Посчитать норму невязки || A*X = b || --> min
def ResidualRate(X):
    AX = np.dot(A, X)  # A*X
    AX_b = AX - b  # A*X - b
    return np.linalg.norm(AX_b)  # || A*X - b ||

# Решение СЛАУ методом релаксации
def RelaxationMethod(w, ResRateArr):

print("\n[illegible]
[illegible]\n")

    global ProcessNum
    print("ПРОЦЕСС №", ProcessNum, "\n")
    ProcessNum += 1

    StartTime = time.time()

    print("Заданная точность Epsilon =", Epsilon, "\n")
```

```
    print("Начальное приближение X0 =", X0, "\n")
    print("Параметр релаксации w =", w, "\n")

    L = np.tril(A, k=-1)  # Нижнетреугольная матрица (на диагонали нужны нолики)

    R = np.triu(A, k=1)  # Верхнетреугольная матрица (на диагонали нужны
нолики))

    D = np.diag(np.diag(A))  # Диагональная матрица (со всеми нулями, т.к. это
удобнее для последующего умножения)

    ObrD = np.linalg.inv(D)  # Матрица D^(-1), обратная матрице D

    UnitMatrix = np.eye(N)  # Единичная матрица размера NxN

    # B(w) = (I + w*D^(-1)*L)^(-1)*((1 - w)*I - w*D^(-1)*R)
    I_wOBrDL_Obr = np.linalg.inv(UnitMatrix + w*np.dot(ObrD, L))  # 1) (I +
wD^(-1)L)^(-1)
    I_1_w_wObrDR = (1 - w)*UnitMatrix - w*np.dot(ObrD, R)  # 2) (1-w)I - wD^(-
1)R

    B = np.dot(I_wOBrDL_Obr, I_1_w_wObrDR)  # B(w) = (I + w*D^(-1)*L)^(-1)*((1 -
w)*I - w*D^(-1)*R)
    print("B = (I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)\n\nB =\n", B, "\n")

    NormI_wObrDL_Obr = np.linalg.norm(I_wOBrDL_Obr)  # Норма первой части
    print("1) Norm((I + wD^(-1)L)^(-1)) =", NormI_wObrDL_Obr, "\n")
    NormI_1_w_wObrDR = np.linalg.norm(I_1_w_wObrDR)  # Норма второй части
    print("2) Norm((1-w)I - wD^(-1)R) =", NormI_1_w_wObrDR, "\n")

    BothPartsMult = NormI_wObrDL_Obr*NormI_1_w_wObrDR  # Произведение обоих
частей

    if BothPartsMult < 1.:  # Если произведение норм двух частей < 1, то
        print("Произведение норм двух частей =", BothPartsMult, "< 1.0 =>
процесс сходится.\n")
    else:
        print("Произведение норм двух частей =", BothPartsMult, ">= 1.0,
требуются дальнейшие исследования...\n")

        NormB = np.linalg.norm(B)  # Норма матрицы B
        print("Norm(B) = Norm((I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)) =",
NormB, "\n")

        if NormB < 1.:  # Если норма самой матрицы B < 1, то
            print("Норма матрицы B =", NormB, "< 1.0 => процесс сходится.\n")
        else:
            print("Норма матрицы B =", NormB, ">= 1.0, требуются дальнейшие
исследования...\n")

            EigenValuesB = np.linalg.eigvals(B)  # Вектор, хранящий в себе
собственные значения матрицы B
            print("Собственные значения матрицы B:\n", EigenValuesB, "\n")
            MaxEigenValueB = 0.
            for i in range(EigenValuesB.size):
                if abs(EigenValuesB[i]) > MaxEigenValueB:
                    MaxEigenValueB = abs(EigenValuesB[i])
            print("Наибольшее по модулю из собственных значений матрицы B =",
MaxEigenValueB, "\n")
```

```
            if MaxEigenValueB < 1.:  # Если максимальное по модулю собственное
значение матрицы < 1, то
                print("Наибольшее по модулю собственное значение матрицы B =",
MaxEigenValueB, "< 1.0 => процесс сходится.\n")
            else:
                print("Наибольшее по модулю собственное значение матриц =",
MaxEigenValueB, ">= 1.0 => процесс расходится.\n")

    print("Процесс начал вычислительные итерации...\n")

    # x_k+1_i = (1 - w)*x_k_i + (w / a_ii)*(b_i - Sum_i-1_j=1_(a_ij*x_k+1+j) -
Sum_n_j=i+1_(a_ij*x_k_j))
    Xk = X0  # Вектор Xk - нужен для нахождения вектора Xk+1 в последующих
итерациях
    Xk_1 = np.zeros(N)  # Вектор Xk+1 - следующий вектор-ответ
    IterationsAmount = 0  # Количество итераций
    CurrResRate = ResidualRate(Xk)  # Текущая невязка
    while CurrResRate > Epsilon:
        IterationsAmount += 1  # На каждой итерации приплюсовываем единицу к
счетчику итераций
        for i in range(N):
            FirstSum = 0
            for j in range(i):
                FirstSum += (A[i, j] * Xk_1[j])

            SecondSum = 0
            for j in range(i + 1, N):
                SecondSum += (A[i, j] * Xk[j])

            Xk_1[i] = (1 - w) * Xk[i] + (w / A[i, i]) * (b[i] - FirstSum -
SecondSum)

        Xk = Xk_1  # Говорим, что вектор Xk+1 в следующей итерации будет просто
Xk
        CurrResRate = ResidualRate(Xk)  # Текущая невязка
        ResRateArr.append(CurrResRate)  # Добавляем текущую невязку в список
невязок для графика

    print("После", IterationsAmount, "итерации был подобран X =", Xk, "\n")

    print("Общее время работы процесса: %s seconds" % (time.time() - StartTime),
"\n")

    return IterationsAmount  # По завершении процесса возвращаем количество
итераций, которое нам понадобилось


print("\nМатрица A =\n", A, "\n")
print("Вектор b =\n", b)
# Значения t, в которых будем хранить времена работы всех процессов
ResRateArr1 = []  # Список ординат (норм невязки на разных итерациях) для
графика 1-ого процесса
IterAmount1 = RelaxationMethod(w0, ResRateArr1)
IterArr1 = np.arange(1, IterAmount1 + 1)  # Массив абсцисс (количества итераций)
для графика 1-ого процесса
ResRateArr2 = []  # Список ординат (норм невязки на разных итерациях) для
графика 2-ого процесса
IterAmount2 = RelaxationMethod(w1, ResRateArr2)
```

```
IterArr2 = np.arange(1, IterAmount2 + 1)  # Массив абсцисс (количества итераций) для графика 2-ого процесса
ResRateArr3 = []  # Список ординат (норм невязки на разных итерациях) для графика 3-его процесса
IterAmount3 = RelaxationMethod(w2, ResRateArr3)
IterArr3 = np.arange(1, IterAmount3 + 1)  # Массив абсцисс (количества итераций) для графика 3-его процесса
ResRateArr4 = []  # Список ординат (норм невязки на разных итерациях) для графика 3-его процесса
IterAmount4 = RelaxationMethod(w3, ResRateArr4)
IterArr4 = np.arange(1, IterAmount4 + 1)  # Массив абсцисс (количества итераций) для графика 4-ого процесса
ResRateArr5 = []  # Список ординат (норм невязки на разных итерациях) для графика 5-ого процесса
IterAmount5 = RelaxationMethod(w4, ResRateArr5)
IterArr5 = np.arange(1, IterAmount5 + 1)  # Массив абсцисс (количества итераций) для графика 5-ого процесса
plt.semilogy(IterArr1, ResRateArr1, label='w0')
plt.semilogy(IterArr2, ResRateArr2, label='w1')
plt.semilogy(IterArr3, ResRateArr3, label='w2')
plt.semilogy(IterArr4, ResRateArr4, label='w3')
plt.semilogy(IterArr5, ResRateArr5, label='w4')
plt.xlabel("Номер итерации")
plt.ylabel("Норма невязки на этой итерации")
plt.legend()
plt.show()
```

Делая различные эксперименты, я выяснил, что при $w_0 = 2.5$ процесс расходится, а при $w_1 = 1.5$ процесс сходится:

```
Параметр релаксации w = 2.5

B = (I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)

B =
 [[-0.88411079 -1.01311953  0.52478134]
 [ 0.07653061 -0.98979592 -0.40816327]
 [-0.28571429  0.42857143 -1.14285714]]

1) Norm((I + wD^(-1)L)^(-1)) = 0.5605103176258518

2) Norm((1-w)I - wD^(-1)R) = 7.186141175902405

Произведение норм двух частей = 4.027906273009269 >= 1.0, требуются дальнейшие исследования...

Norm(B) = Norm((I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)) = 2.1924956662181647

Норма матрицы B = 2.1924956662181647 >= 1.0, требуются дальнейшие исследования...

Собственные значения матрицы B:
 [-0.90251465+0.64602411j -0.90251465-0.64602411j -1.21173455+0.j        ]

Наибольшее по модулю из собственных значений матрицы B = 1.2117345505537214

Наибольшее по модулю собственное значение матриц = 1.2117345505537214 >= 1.0 => процесс расходится.
```

```
Параметр релаксации w = 1.5

B = (I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)

B =
 [[-0.6092 -0.6312  0.336 ]
 [ 0.078  -0.692  -0.24  ]
 [-0.24    0.36   -0.8   ]]

1) Norm((I + wD^(-1)L)^(-1)) = 0.7617243595947291

2) Norm((1-w)I - wD^(-1)R) = 3.6483729250173975

Произведение норм двух частей = 2.7790545298716256 >= 1.0, требуются дальнейшие исследования...

Norm(B) = Norm((I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)) = 1.5007271837346057

Норма матрицы B = 1.5007271837346057 >= 1.0, требуются дальнейшие исследования...

Собственные значения матрицы B:
 [-0.64698632+0.46441208j -0.64698632-0.46441208j -0.80722735+0.j        ]

Наибольшее по модулю из собственных значений матрицы B = 0.807227350494909

Наибольшее по модулю собственное значение матрицы B = 0.807227350494909 < 1.0 => процесс сходится.
```

Позже я доказал расходимость $w_0$ и сходимость $w_1$ теоретически.

Доказательство расходимости процесса при $w_0 = 2.5$:

$$A = \begin{pmatrix} -1 & 1 & -1 \\ -1 & 4 & -2 \\ 2 & -3 & 5 \end{pmatrix} \quad b = \{-7, 7, 28\}$$

$$x^0 = \{0, 0, 0\}$$

$$\boxed{w = w_0 = \frac{5}{2},}$$

$$B(w) = (I + wD^{-1}L)^{-1} \cdot ((1-w)I - wD^{-1}R)$$

$$L = \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 2 & -3 & 0 \end{pmatrix} \quad D = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

$$D^{-1} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{5} \end{pmatrix}$$

$$wD^{-1}L = \frac{5}{2} \cdot \begin{pmatrix} -1 & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{5} \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 2 & -3 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ -\frac{5}{8} & 0 & 0 \\ 1 & -\frac{3}{2} & 0 \end{pmatrix}$$

$$I + wD^{-1}L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ -\frac{5}{8} & 0 & 0 \\ 1 & -\frac{3}{2} & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{5}{8} & 1 & 0 \\ 1 & -\frac{3}{2} & 1 \end{pmatrix}.$$

ОБРАЩАЕМ:

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ -\frac{5}{8} & 1 & 0 & 0 & 1 & 0 \\ 1 & -\frac{3}{2} & 1 & 0 & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & \frac{5}{8} & 1 & 0 \\ 0 & -\frac{3}{2} & 1 & -1 & 0 & 1 \end{array}\right) \sim$$

$$\sim \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & \frac{5}{8} & 1 & 0 \\ 0 & 0 & 1 & -\frac{1}{16} & \frac{3}{2} & 1 \end{array}\right)$$

$$\left(I + w D^{-1} L\right)^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{5}{8} & 1 & 0 \\ -\frac{1}{16} & \frac{3}{2} & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} 0 & 1 & -1 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\left((1-w)I - w D^{-1} R\right) = \left(1 - \frac{5}{2}\right) \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} -$$

$$- \frac{5}{2} \cdot \begin{pmatrix} -1 & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{5} \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & -1 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -\frac{3}{2} & 0 & 0 \\ 0 & -\frac{3}{2} & 0 \\ 0 & 0 & -\frac{3}{2} \end{pmatrix} +$$

$$+ \begin{pmatrix} 0 & \frac{5}{2} & -\frac{5}{2} \\ 0 & 0 & \frac{5}{4} \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -\frac{3}{2} & +\frac{5}{2} & -\frac{5}{2} \\ 0 & -\frac{3}{2} & +\frac{5}{4} \\ 0 & 0 & -\frac{3}{2} \end{pmatrix}.$$

$$\left\|\left(I+\omega D^{-1}L\right)^{-1}\right\|_2=\left\|\begin{pmatrix}1 & 0 & 0\\ \frac{5}{8} & 1 & 0\\ -\frac{1}{16} & \frac{3}{2} & 1\end{pmatrix}\right\|_2=$$

$$=\sqrt{(1)^2+\left(\frac{5}{8}\right)^2+(1)^2+\left(-\frac{1}{16}\right)^2+\left(\frac{3}{2}\right)^2+(1)^2}=$$

$$=\sqrt{\frac{1024}{1024}+\frac{400}{1024}+\frac{1024}{1024}+\frac{1}{1024}+\frac{2304}{1024}+\frac{1024}{1024}}=$$

$$=\sqrt{\frac{5777}{1024}}=\frac{\sqrt{5777}}{32}\approx 2.3758.$$

$$\left\|(1-\omega)I-\omega D^{-1}R\right\|_2=\left\|\begin{pmatrix}-\frac{3}{2} & \frac{5}{2} & -\frac{5}{2}\\ 0 & -\frac{3}{2} & -\frac{5}{4}\\ 0 & 0 & -\frac{3}{2}\end{pmatrix}\right\|_2=$$

$$=\sqrt{\left(-\frac{3}{2}\right)^2+\left(+\frac{5}{2}\right)^2+\left(-\frac{5}{2}\right)^2+\left(-\frac{3}{2}\right)^2+\left(+\frac{5}{4}\right)^2+\left(-\frac{3}{2}\right)^2}=$$

$$=\sqrt{\frac{9}{4}+\frac{25}{4}+\frac{25}{4}+\frac{9}{4}+\frac{25}{16}+\frac{9}{4}}=$$

$$=\sqrt{\frac{18\cdot 2}{16}+\frac{100}{16}+\frac{100}{16}+\frac{36}{16}+\frac{25}{16}+\frac{36}{16}}=$$

$$=\sqrt{\frac{333}{16}}=\frac{\sqrt{333}}{4}\approx 4.562.$$

$\frac{\sqrt{5777}}{32} \cdot \frac{\sqrt{333}}{32} < 1?$ — НЕТ $\Rightarrow$ требуются дополнительные исследования.

$$B = (I + \omega D^{-1} L)^{-1} \cdot ((1-\omega) I - \omega D^{-1} R) =$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ \frac{5}{8} & 1 & 0 \\ -\frac{1}{16} & \frac{3}{2} & 1 \end{pmatrix} \cdot \begin{pmatrix} -\frac{3}{2} & -\frac{5}{2} & -\frac{5}{2} \\ 0 & -\frac{3}{2} & -\frac{5}{4} \\ 0 & 0 & -\frac{3}{2} \end{pmatrix} = \begin{pmatrix} -\frac{3}{2} & \frac{5}{2} & -\frac{5}{2} \\ -\frac{15}{16} & \frac{1}{16} & -\frac{5}{16} \\ \frac{3}{32} & -\frac{77}{32} & \frac{17}{32} \end{pmatrix}$$

$$\|B\|_2 = \sqrt{\left(-\frac{2\cdot 24}{2\cdot 16}\right)^2 + \left(\frac{40\cdot 2}{16\cdot 2}\right)^2 + \left(-\frac{40\cdot 2}{16\cdot 2}\right)^2 + \left(-\frac{30}{32}\right)^2 + \left(\frac{2}{32}\right)^2 + \left(-\frac{10}{32}\right)^2 + \left(\frac{3}{32}\right)^2 + \left(-\frac{77}{32}\right)^2 + \left(\frac{17}{32}\right)^2} = \frac{\sqrt{22335}}{32} \approx 4{,}67.$$

$\frac{\sqrt{22335}}{32} < 1?$ — НЕТ $\Rightarrow$ придется искать собственные значения матрицы B.

$$\begin{vmatrix} -\frac{3}{2}-\lambda & \frac{5}{2} & -\frac{5}{2} \\ -\frac{15}{16} & \frac{1}{16}-\lambda & -\frac{5}{16} \\ \frac{3}{32} & -\frac{77}{32} & \frac{17}{32}-\lambda \end{vmatrix} = \left(-\frac{3}{2}-\lambda\right)\cdot\left(\frac{1}{16}-\lambda\right)\cdot\left(\frac{17}{32}-\lambda\right) +$$

$$+\frac{5}{2}\cdot\left(-\frac{5}{16}\right)\cdot\frac{3}{32}+\left(-\frac{5}{2}\right)\cdot\left(-\frac{15}{16}\right)\cdot\left(-\frac{77}{32}\right)-\left(-\frac{5}{2}\right)\cdot\left(\frac{1}{16}-\lambda\right)\cdot$$

$$\cdot\left(\frac{3}{32}\right)-\left(-\frac{3}{2}-\lambda\right)\cdot\left(-\frac{5}{16}\right)\cdot\left(-\frac{77}{32}\right)-\left(-\frac{15}{16}\right)\cdot\left(\frac{5}{2}\right)\cdot\left(\frac{17}{32}-\lambda\right)=0,$$

$$-32\lambda^3-29\lambda^2-31\lambda-108=0,$$

$$\lambda_1\approx-1{,}6085$$

$$\lambda_{2,3}\approx 0{,}3511\pm 1{,}4053i$$

Наибольшее по модулю: $\lambda_1=-1{,}6085$,

$|-1{,}6085|>1 \Rightarrow$ процесс расходится

P.S. Для решения того многочлена 3-ей степени я использовал сервис WolframAplha (https://www.wolframalpha.com/input/?i=%28-3%2F2-x%29*%281%2F16-x%29*%2817%2F32-x%29%2B%285%2F2%29*%28-5%2F16%29*%283%2F32%29%2B%28-5%2F2%29*%28-15%2F16%29*%28-77%2F32%29-%28-5%2F2%29*%281%2F16-x%29*%283%2F32%29-%28-3%2F2-x%29*%28%28-5%2F16%29*%28-77%2F32%29%29-%28-15%2F16%29*%285%2F2%29*%2817%2F32-x%29+%3D+0)

Доказательство сходимости процесса при $w_1 = 1.5$:

$$A = \begin{pmatrix} -1 & 1 & -1 \\ -1 & 4 & -2 \\ 2 & -3 & 5 \end{pmatrix} \qquad b = \{-7, 7, 28\}$$

$$x^0 = \{0, 0, 0\}$$

$$\boxed{w = w_2 = \frac{3}{2}}$$

$$B(w) = (I + wD^{-1}L)^{-1} \cdot ((1-w)I - wD^{-1}R)$$

$$L = \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 2 & -3 & 0 \end{pmatrix} \qquad D = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

$$D^{-1} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{5} \end{pmatrix}$$

---

$$wD^{-1}L =$$

$$= \frac{3}{2} \cdot \begin{pmatrix} -1 & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{5} \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 2 & -3 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ -\frac{3}{8} & 0 & 0 \\ \frac{3}{5} & -\frac{9}{10} & 0 \end{pmatrix}$$

$$I + wD^{-1}L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ -\frac{3}{8} & 0 & 0 \\ \frac{3}{5} & -\frac{9}{10} & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{3}{8} & 1 & 0 \\ \frac{3}{5} & -\frac{9}{10} & 1 \end{pmatrix}.$$

ОБРАЩАЕМ:

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ -\frac{3}{8} & 1 & 0 & 0 & 1 & 0 \\ \frac{3}{5} & -\frac{9}{10} & 1 & 0 & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & \frac{3}{8} & 1 & 0 \\ 0 & -\frac{9}{10} & 1 & -\frac{3}{5} & 0 & 1 \end{array}\right) \sim$$

$$\sim\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & \frac{3}{8} & 1 & 0 \\ 0 & 0 & 1 & -\frac{21}{80} & \frac{9}{10} & 1 \end{array}\right)$$

$$(I+\omega D^{-1}L)^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{3}{8} & 1 & 0 \\ -\frac{21}{80} & \frac{9}{10} & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} 0 & 1 & -1 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{pmatrix}$$

$$((1-\omega)I - \omega D^{-1}R) = \left(1-\frac{3}{2}\right)\cdot\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \frac{3}{2}\cdot$$

$$\cdot\begin{pmatrix} -1 & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{3} \end{pmatrix}\cdot\begin{pmatrix} 0 & 1 & -1 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} & 0 & 0 \\ 0 & -\frac{1}{2} & 0 \\ 0 & 0 & -\frac{1}{2} \end{pmatrix} +$$

$$+\begin{pmatrix} 0 & \frac{3}{2} & -\frac{3}{2} \\ 0 & 0 & \frac{3}{4} \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} & \frac{3}{2} & -\frac{3}{2} \\ 0 & -\frac{1}{2} & \frac{3}{4} \\ 0 & 0 & -\frac{1}{2} \end{pmatrix}$$

$$\|(I+\omega D^{-1}L)^{-1}\|_2 = \left\|\begin{pmatrix} 1 & 0 & 0 \\ \frac{3}{8} & 1 & 0 \\ -\frac{21}{80} & \frac{9}{10} & 1 \end{pmatrix}\right\|_2 =$$

$$= \sqrt{1^2+\left(\frac{3}{8}\right)^2+1^2+\left(-\frac{21}{80}\right)^2+\left(\frac{9}{10}\right)^2+1^2} = \frac{\sqrt{1029}}{16} \approx 2{,}005.$$

$$\left\|(1-w)I - wD^{-1}R\right\|_2 = \left\|\begin{pmatrix} -\frac{1}{2} & \frac{3}{2} & -\frac{3}{2} \\ 0 & -\frac{1}{2} & \frac{3}{4} \\ 0 & 0 & -\frac{1}{2} \end{pmatrix}\right\|_2 =$$

$$= \sqrt{\left(-\frac{1}{2}\right)^2 + \left(\frac{3}{2}\right)^2 + \left(-\frac{3}{2}\right)^2 + \left(-\frac{1}{2}\right)^2 + \left(\frac{3}{4}\right)^2 + \left(-\frac{1}{2}\right)^2} =$$

$$= \frac{\sqrt{93}}{4} \approx 2,4109.$$

$\frac{\sqrt{1029}}{16} \cdot \frac{\sqrt{93}}{4} < 1?$ – НЕТ $\Rightarrow$ требуются дополнительные исследования

$$B = (I + wD^{-1}L)^{-1} \cdot ((1-w)I - wD^{-1}R) =$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ \frac{3}{8} & 1 & 0 \\ -\frac{21}{80} & \frac{9}{10} & 1 \end{pmatrix} \cdot \begin{pmatrix} -\frac{1}{2} & \frac{3}{2} & -\frac{3}{2} \\ 0 & -\frac{1}{2} & \frac{3}{4} \\ 0 & 0 & -\frac{1}{2} \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} & \frac{3}{2} & -\frac{3}{2} \\ -\frac{3}{16} & \frac{1}{16} & \frac{3}{16} \\ \frac{21}{160} & -\frac{27}{32} & \frac{91}{160} \end{pmatrix}.$$

$$\|B\|_2 = \sqrt{\left(-\frac{80}{160}\right)^2 + \left(\frac{240}{160}\right)^2 + \left(-\frac{240}{160}\right)^2 + \left(-\frac{30}{160}\right)^2 + \left(\frac{10}{160}\right)^2 + \left(\frac{30}{160}\right)^2 + \left(\frac{21}{160}\right)^2 + \left(\frac{-135}{160}\right)^2 + \left(\frac{91}{160}\right)^2} = \frac{\sqrt{150447}}{160} \approx 2,424.$$

$\dfrac{\sqrt{150447}}{160} < 1?$ — НЕТ $\Rightarrow$ придется искать собственные значения матрицы B.

$$\begin{vmatrix} -\frac{1}{2}-\lambda & \frac{3}{2} & -\frac{3}{2} \\ -\frac{3}{16} & \frac{1}{16}-\lambda & \frac{3}{16} \\ \frac{21}{160} & -\frac{27}{32} & \frac{91}{160}-\lambda \end{vmatrix} = \left(-\frac{1}{2}-\lambda\right)\cdot\left(\frac{1}{16}-\lambda\right)\cdot\left(\frac{91}{160}-\lambda\right)+$$

$$+\left(-\frac{3}{2}\right)\cdot\left(-\frac{3}{16}\right)\cdot\left(-\frac{27}{32}\right)+\frac{3}{2}\cdot\frac{3}{16}\cdot\frac{21}{160}-\left(-\frac{3}{2}\right)\cdot$$

$$\cdot\left(\frac{1}{16}-\lambda\right)\cdot\frac{21}{160}-\left(-\frac{1}{2}-\lambda\right)\cdot\frac{3}{16}\cdot\left(-\frac{27}{32}\right)-\left(-\frac{3}{16}\right)\cdot\frac{3}{2}\cdot$$

$$\cdot\left(\frac{91}{160}-\lambda\right)=0,$$

$$-160\lambda^3+21\lambda^2-57\lambda-20=0,$$

$$\lambda_1 \approx -0,26932$$

$$\lambda_{2,3} \approx 0,20029 \pm 0,65117i$$

Наибольшее по модулю:

$$\lambda_{2,3} = 0,20029 \pm 0,65117i$$

$|0,20029 \pm 0,65117i| = 0,68127 < 1 \Rightarrow$ процесс сходится.

P.S. Для решения того многочлена 3-ей степени я снова использовал сервис WolframAplha (https://www.wolframalpha.com/input/?i=%28-1%2F2-x%29*%281%2F16-x%29*%2891%2F160-x%29%2B%28-3%2F2%29*%28-3%2F16%29*%28-27%2F32%29%2B3%2F2*3%2F16*21%2F160-%28-3%2F2%29*%281%2F16-x%29*21%2F160-%28-1%2F2-x%29*3%2F16*%28-27%2F32%29-%28-3%2F16%29*3%2F2*%2891%2F160-x%29+%3D+0)

В качестве еще трёх значений параметра релаксации ($w_2$, $w_3$ и $w_5$) я взял числа из диапазона от нуля до двух, а именно: $w_2 = 1.0$, $w_3 = 0.5$, $w_5 = 0.1$. Как я выяснил позже, при каждом из этих трёх параметров релаксации процесс сходился, хотя и медленнее (ему требовалось больше итераций для достижения нужной точности).

В конце концов программа закончила свои расчёты для каждого параметра релаксации (в случае с $w_0$ она остановила расчёты перед переполнением стека) вывела следующее:

```
Матрица A =
 [[-1.  1. -1.]
 [-1.  4. -2.]
 [ 2. -3.  5.]]

Вектор b =
 [-7.  7. 28.]


ПРОЦЕСС № 1

Заданная точность Epsilon = 1e-08

Начальное приближение X0 = [0. 0. 0.]

Параметр релаксации w = 2.5

B = (I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)

B =
 [[-1.5      2.5      -2.5    ]
 [-0.9375   0.0625  -0.3125 ]
 [ 0.09375 -2.40625  0.53125]]

1) Norm((I + wD^(-1)L)^(-1)) = 2.375822226093527

2) Norm((1-w)I - wD^(-1)R) = 4.562071897723665

Произведение норм двух частей = 10.838671811648558 >= 1.0, требуются дальнейшие исследования...

Norm(B) = Norm((I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)) = 4.670280873512856
```

```
Norm(B) = Norm((I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)) = 4.670280873512856

Норма матрицы B = 4.670280873512856 >= 1.0, требуются дальнейшие исследования...

Собственные значения матрицы B:
 [-1.60847484+0.j          0.35111242+1.40534019j  0.35111242-1.40534019j]

Наибольшее по модулю из собственных значений матрицы B = 1.6084748390360595

Наибольшее по модулю собственное значение матриц = 1.6084748390360595 >= 1.0 => процесс расходится.

Процесс начал вычислительные итерации...

C:/Users/user/PycharmProjects/CalcMethods_Lab_3_V5_Task_2_5/Task_2_Main.py:124: RuntimeWarning: overflow encountered in double_scalars
  FirstSum += (A[i, j] * Xk_1[j])
C:/Users/user/PycharmProjects/CalcMethods_Lab_3_V5_Task_2_5/Task_2_Main.py:124: RuntimeWarning: invalid value encountered in double_scalars
  FirstSum += (A[i, j] * Xk_1[j])
После 1488 итерации был подобран X = [-1.15598642e+308 -8.09298537e+307              nan]

Общее время работы процесса: 0.04488539695739746 seconds



ПРОЦЕСС № 2

Заданная точность Epsilon = 1e-08

Начальное приближение X0 = [0. 0. 0.]

Параметр релаксации w = 1.5

B = (I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)
```

```
B =
 [[-0.5      1.5     -1.5    ]
 [-0.1875   0.0625   0.1875 ]
 [ 0.13125 -0.84375  0.56875]]

1) Norm((I + wD^(-1)L)^(-1)) = 2.00487686654318

2) Norm((1-w)I - wD^(-1)R) = 2.4109126902482387

Произведение норм двух частей = 4.833583079934078 >= 1.0, требуются дальнейшие исследования...

Norm(B) = Norm((I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)) = 2.4242186241137578

Норма матрицы B = 2.4242186241137578 >= 1.0, требуются дальнейшие исследования...

Собственные значения матрицы B:
 [-0.26932016+0.j          0.20028508+0.65116627j  0.20028508-0.65116627j]

Наибольшее по модулю из собственных значений матрицы B = 0.6812720583254084

Наибольшее по модулю собственное значение матрицы B = 0.6812720583254084 < 1.0 => процесс сходится.

Процесс начал вычислительные итерации...

После 54 итерации был подобран X = [7.00000001 7.         7.        ]

Общее время работы процесса: 0.0019888877868652344 seconds


ПРОЦЕСС № 3
```

```
Заданная точность Epsilon = 1e-08

Начальное приближение X0 = [0. 0. 0.]

Параметр релаксации w = 1.0

B = (I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)

B =
 [[ 0.    1.   -1.  ]
 [ 0.    0.25  0.25]
 [ 0.   -0.25  0.55]]

1) Norm((I + wD^(-1)L)^(-1)) = 1.866815470259447

2) Norm((1-w)I - wD^(-1)R) = 1.5

Произведение норм двух частей = 2.8002232053891705 >= 1.0, требуются дальнейшие исследования...

Norm(B) = Norm((I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)) = 1.57797338380595

Норма матрицы B = 1.57797338380595 >= 1.0, требуются дальнейшие исследования...

Собственные значения матрицы B:
 [0. +0.j  0.4+0.2j 0.4-0.2j]

Наибольшее по модулю из собственных значений матрицы B = 0.4472135954999579

Наибольшее по модулю собственное значение матрицы B = 0.4472135954999579 < 1.0 => процесс сходится.

Процесс начал вычислительные итерации...
```

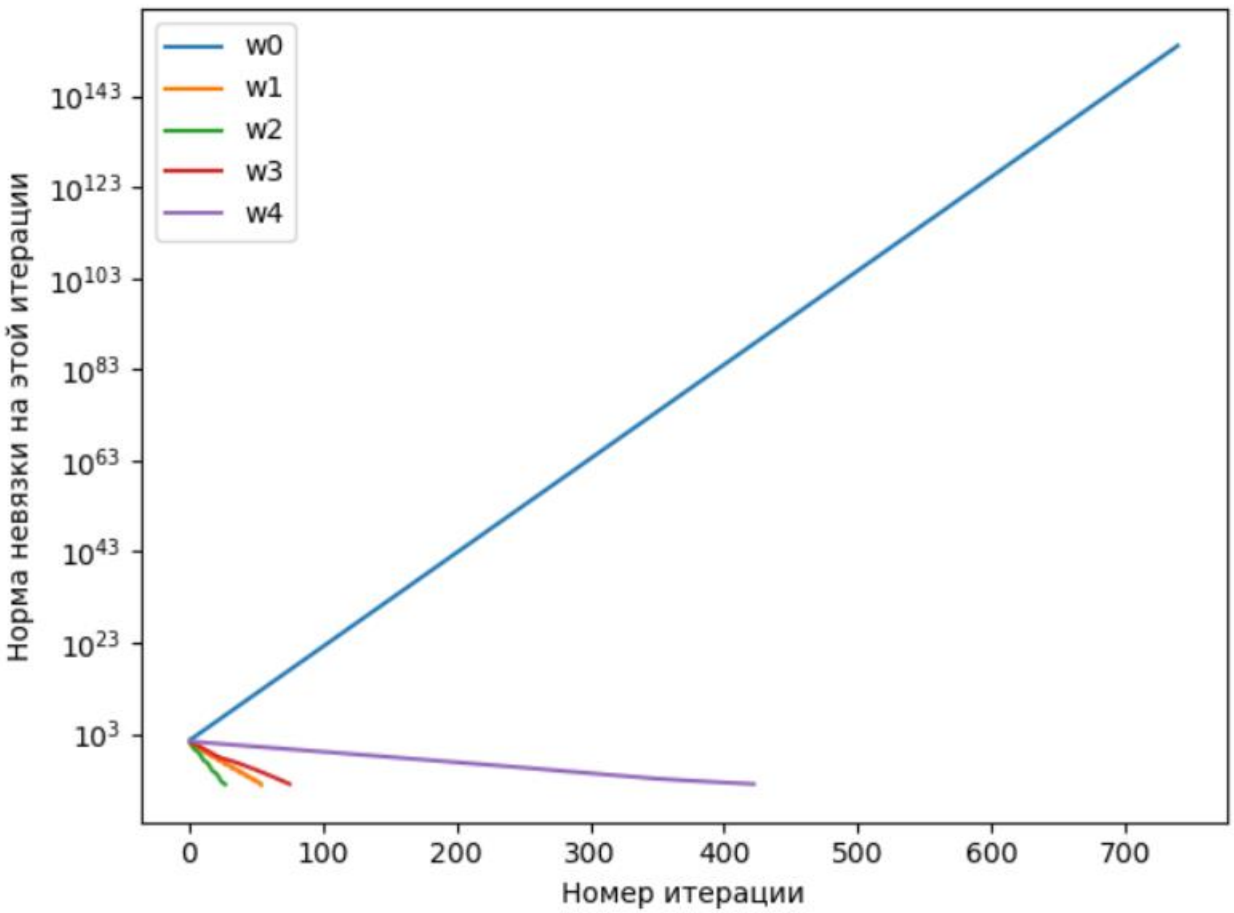
```
После 27 итерации был подобран X = [7. 7. 7.]

Общее время работы процесса: 0.001995086669921875 seconds



ПРОЦЕСС № 4

Заданная точность Epsilon = 1e-08

Начальное приближение X0 = [0. 0. 0.]

Параметр релаксации w = 0.5

B = (I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)

B =
 [[ 0.5      0.5     -0.5    ]
 [ 0.0625   0.5625   0.1875 ]
 [-0.08125  0.06875  0.65625]]

1) Norm((I + wD^(-1)L)^(-1)) = 1.769754573380162

2) Norm((1-w)I - wD^(-1)R) = 1.14564392373896

Произведение норм двух частей = 2.027508573502218 >= 1.0, требуются дальнейшие исследования...

Norm(B) = Norm((I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)) = 1.2439698298190354

Норма матрицы B = 1.2439698298190354 >= 1.0, требуются дальнейшие исследования...

Собственные значения матрицы B:
```

```
 [0.22327492+0.j         0.74773754+0.02713839j 0.74773754-0.02713839j]

Наибольшее по модулю из собственных значений матрицы B = 0.7482298579056177

Наибольшее по модулю собственное значение матрицы B = 0.7482298579056177 < 1.0 => процесс сходится.

Процесс начал вычислительные итерации...

После 75 итерации был подобран X = [7. 7. 7.]

Общее время работы процесса: 0.0019953250885009766 seconds



ПРОЦЕСС № 5

Заданная точность Epsilon = 1e-08

Начальное приближение X0 = [0. 0. 0.]

Параметр релаксации w = 0.1

B = (I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)

B =
 [[ 0.9      0.1     -0.1    ]
 [ 0.0225   0.9025   0.0475 ]
 [-0.03465  0.05015  0.90685]]

1) Norm((I + wD^(-1)L)^(-1)) = 1.7336975658978127
```

```
2) Norm((1-w)I - wD^(-1)R) = 1.5660459763365826

Произведение норм двух частей = 2.7150500972587968 >= 1.0, требуются дальнейшие исследования...

Norm(B) = Norm((I + wD^(-1)L)^(-1)*((1-w)I - wD^(-1)R)) = 1.57269237853434

Норма матрицы B = 1.57269237853434 >= 1.0, требуются дальнейшие исследования...

Собственные значения матрицы B:
 [0.79933324 0.96056638 0.94945038]

Наибольшее по модулю из собственных значений матрицы B = 0.9605663804966931

Наибольшее по модулю собственное значение матрицы B = 0.9605663804966931 < 1.0 => процесс сходится.

Процесс начал вычислительные итерации...

После 422 итерации был подобран X = [6.99999999 7.         7.00000001]

Общее время работы процесса: 0.006980657577514648 seconds


Process finished with exit code 0
```

Мы можем наблюдать, что наши теоретические расчёты для параметров релаксации $w_0$ и $w_1$ полностью подтвердились (в том числе и промежуточные расчёты). Также мы можем наблюдать некоторые погрешности в ответах. Так, например, в 5-ом процессе при $w_4 = 0.1$ вектор x равен {6.99999999, 7.0, 7.0} вместо истинных {7.0, 7.0, 7.0}. Однако самое главное, что требуемая точность, которую мы задали (а я задал Epsilon = $10^{-8}$) выполняется, значит программа работает корректно.

Программа также вывела следующую логарифмическую диаграмму сходимости:




Мы можем наблюдать, что при $w_0 = 2.5$ (при котором итерационный процесс расходится) с каждой итерацией норма невязки становилась все больше и больше, что не удивительно. А что касается остальных параметров релаксации ($w_1$, $w_2$, $w_3$, $w_4$), то их итерационные процессы вполне себе сошлись, при чём можно заметить, что быстрее всего сошёлся процесс w2 = 0.5. Значится, «самый быстрый» параметр сходимости около $w^* = 0.5$ для нашей матрицы.

Код этой программе будет в файле Task_2_Main.py, а картинка с графиком в файле Task_2_Diagram.png.

**Задание 5:**

**Задание 5. Итерационные методы для разреженных СЛАУ особого вида**

1. Написать программу, которая при данном $n$ решает СЛАУ $A_n x = b_n$ указанным в варианте методом. Здесь $A_n$ – разреженные матрицы размерности $n$ из списка 2 (см. ниже), указанные в варианте.
   - Матрицу $A_n$ следует либо хранить в одном из форматов для разреженных матриц, либо сразу реализовать итерационный метод, учитывая известную структуру матрицы. *Хранить в памяти матрицу $A_n$ целиком со всеми нулями запрещено!*
   - Вектор $b_n$ выбирать таким образом, чтобы он соответствовал некоторому заранее заданному решению.
   - Критерий остановки итераций: $\|A_n x^k - b_n\| < \varepsilon$
2. Подвердить правильность работы программы на примере нескольких СЛАУ размерности 5-10.
3. Построить диаграмму сходимости (общую) для $n = 100, 1000, 10000$.
4. Построить диаграмму, в которой по оси абсцисс изменяется $n = [10^{k/2}]$, $k = 1, \ldots, 12$, а на оси ординат отложено время работы, которое требуется, чтобы норма невязки не превышала $10^{-8}$.

**Вариант метода:**

2. Метод Гаусса-Зейделя

**Вид матриц:**

4. Матрицы вида $A_n = \begin{pmatrix} a & & & & & & & b \\ & \ddots & & & & & \dots & \\ & & a & & & b & & \\ & & & a & b & & & \\ & & & b & a & & & \\ & & b & & & a & & \\ & \dots & & & & & \ddots & \\ b & & & & & & & a \end{pmatrix}$. Здесь $a$, и $b$ — параметры, $n$ четное.

В. $a = 1, b = -2$

Итак, я написал программу на языке Python, которая при заданном N решает СЛАУ $A_n x = b_n$ методом Гаусса-Зейделя. Для этого мне был предложен вариант матрицы, указанный на картинке выше с параметрами a = 1, b = -2. Однако после долгих исследований я пришел к выводу, что метод Гаусса-Зейделя будет расходится для матриц с такими параметрами, поэтому я взял параметр a равным 10. Таким образом, у меня были параметры a = 1, b = -2.

Прилагаю исходный код программы:

```
import matplotlib.pyplot as plt
import numpy as np
import time

# Лабораторная №3, Вариант №5, Задание №5-2, Матрица №4В
# Задание №2. Метод релаксации 1
# 1. Написать программу, которая при данном n решает СЛАУ Anx = bn указанным в
варианте методом. Здесь An - разреженные
# матрицы размерности n из списка 2 (см. ниже), указанные в варианте.
# - Матрицу An следует либо хранить в одном из форматов для разреженных матриц,
либо сразу реализовать итерационный
# метод, учитывая известную структуру матрицы. Хранить в памяти матрицу An
целиком со всеми нулями запрещено!
# - Вектор bn выбирать таким образом, чтобы он соответствовал некоторому заранее
заданному решению
# - Критерий остановки итераций: ||AnXk - bn|| < Epsilon
# 2. Подтвердить правильность работы программы на примере нескольких СЛАУ
размерности 5-10.
# 3. Построить диаграмму сходимости (общую) для n = 100, 1000, 10000
# 4. Построить диаграмму, в которой по оси абсцисс изменяется n = [10_k/2], k =
1, ..., 12, а на оси оридант отложено
# время работы, которое требуется, чтобы норма невязки не превышала 10^(-8).

# 2. МЕТОД ГАУССА-ЗЕЙДЕЛЯ

# Требуемая точность (для итераций)
Epsilon = 0.00000001

# Параметры для наших разреженных матриц
MatrixParameter_a = 10  # При параметре a = 1 из условия сходимость не
наблюдалась
MatrixParameter_b = -2

# ФункцияЮ генерирующая нашу матрицу по заданной размерности (a и b - параметры)
```

```
def GenerateSpecificMatrix(N, a, b):
    NeededMatrix = np.zeros((N, N))
    for i in range(N):
        NeededMatrix[i][i] = a
        NeededMatrix[N - i - 1][i] = b
    return NeededMatrix

#  Посчитать норму невязки || A*X = b || --> min
def ResidualRate(A, X):
    AX = np.dot(A, X)  # A*X
    AX_b = AX - b  # A*X - b
    return np.linalg.norm(AX_b)  # || A*X - b ||

# Решение СЛАУ методом Гаусса-Зейделя
def GaussSeidel(A, b, ResRateArr):
    StartTime = time.time()
    N = len(A)  # Запоминаем размер текущей матрицы A
    Xk = np.zeros(N)  # Текущий вектор-ответ
    CurrResidualRate = 1  # Текущая невязка
    IterationsAmount = 0  # Количество итераций
    while CurrResidualRate > Epsilon:  # До тех пор, пока невязка не станет
меньше точности, выполняем итерации
        IterationsAmount += 1  # На каждой итерации приплюсовываем единицу к
счетчику итераций
        Xk_1 = np.zeros(N)  # Вектор-ответ, который будет получен на следующей
итерации
        for i in range(N):
            FirstSum = 0
            for j in range(i):
                FirstSum += (A[i][j] * Xk_1[j])

            SecondSum = 0
            for j in range(i + 1, N):
                SecondSum += (A[i][j] * Xk[j])

            Xk_1[i] = (-FirstSum - SecondSum + b[i]) / A[i][i]
        Xk = np.copy(Xk_1)  # Говорим, что теперь текущий вектор-ответ - это
насчитанный нами в новой итерации вектор
        CurrResidualRate = ResidualRate(A, Xk)
        ResRateArr.append(CurrResidualRate)  # Добавляем текущую невязку в
список невязок для графика

    print("После", IterationsAmount, "итерации был подобран Xk =", Xk, "\n")

    print("Общее время работы процесса: %s seconds" % (time.time() - StartTime),
"\n")

    return IterationsAmount  # По завершении процесса возвращаем количество
итераций, которое нам понадобилось

# Проверим работу программы на матрицах размерностей: 6, 8, 10, 12, 14, 100
ResRateArr = []  # Список списков ординат (норм невязки на разных итерациях для
графика матрицы размерности i)
IterAmount = []  # Список количеств итераций, который понадобились каждому
процессу
for i in range(6, 17, 2):
    if i == 16:
        i = 100  # Для размерности 100
```

```
print("\n
\n")
    print("Размерность текущей матрицы N =", i, "\n")
    CurrRateArr = []  # Список ординат (норм невязки на разных итерациях) для графика матрицы размерности i
    A = GenerateSpecificMatrix(i, MatrixParameter_a, MatrixParameter_b)
    print("A =\n", A, "\n")
    X = np.ones(i)  # Пуская вектором-ответом всегда будет вектор, состоящий из N единичек (так удобнее)
    b = np.dot(A, X)  # Тогда вектор b = A*X
    print("При X =", X, ", b =", b, "\n")
    IterAmount.append(GaussSeidel(A, b, CurrRateArr))  # Запоминаем потребовавшееся количество итераций
    ResRateArr.append(CurrRateArr)  # Также запоминаем список невязок для текущего процесса

for i in range(6):
    LabelNum = 6 + i*2
    if LabelNum == 16:
        LabelNum = 100
    plt.semilogy(np.arange(1, IterAmount[i] + 1), ResRateArr[i], label="N = "+str(LabelNum))
plt.xlabel("Номер итерации")
plt.ylabel("Норма невязки на этой итерации")
plt.legend()
plt.show()
```

Чтобы проверить работу программы, я протестировал её на матрицах размерностей $N_1 = 6$, $N_2 = 8$, $N_3 = 10$, $N_4 = 12$, $N_5 = 14$, $N_6 = 100$. В качестве вектора-ответа я генерировал вектор X, состоящий из столько единичек, какой размерности была текущая матрица. Соответственно и рассчитывался вектор b. На этих матрицах программа выдала следующие результаты:

```
Размерность текущей матрицы N = 6

A =
 [[10.  0.  0.  0.  0. -2.]
 [ 0. 10.  0.  0. -2.  0.]
 [ 0.  0. 10. -2.  0.  0.]
 [ 0.  0. -2. 10.  0.  0.]
 [ 0. -2.  0.  0. 10.  0.]
 [-2.  0.  0.  0.  0. 10.]]

При X = [1. 1. 1. 1. 1. 1.] , b = [8. 8. 8. 8. 8. 8.]

После 8 итерации был подобран Xk = [1. 1. 1. 1. 1. 1.]

Общее время работы процесса: 0.0 seconds



Размерность текущей матрицы N = 8

A =
 [[10.  0.  0.  0.  0.  0.  0. -2.]
 [ 0. 10.  0.  0.  0.  0. -2.  0.]
 [ 0.  0. 10.  0.  0. -2.  0.  0.]
 [ 0.  0.  0. 10. -2.  0.  0.  0.]
 [ 0.  0.  0. -2. 10.  0.  0.  0.]
 [ 0.  0. -2.  0.  0. 10.  0.  0.]
 [ 0. -2.  0.  0.  0.  0. 10.  0.]
 [-2.  0.  0.  0.  0.  0.  0. 10.]]
```

```
При X = [1. 1. 1. 1. 1. 1. 1. 1.] , b = [8. 8. 8. 8. 8. 8. 8. 8.]

После 8 итерации был подобран Xk = [1. 1. 1. 1. 1. 1. 1. 1.]

Общее время работы процесса: 0.0 seconds



Размерность текущей матрицы N = 10

A =
 [[10.  0.  0.  0.  0.  0.  0.  0.  0. -2.]
 [ 0. 10.  0.  0.  0.  0.  0.  0. -2.  0.]
 [ 0.  0. 10.  0.  0.  0.  0. -2.  0.  0.]
 [ 0.  0.  0. 10.  0.  0. -2.  0.  0.  0.]
 [ 0.  0.  0.  0. 10. -2.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. -2. 10.  0.  0.  0.  0.]
 [ 0.  0.  0. -2.  0.  0. 10.  0.  0.  0.]
 [ 0.  0. -2.  0.  0.  0.  0. 10.  0.  0.]
 [ 0. -2.  0.  0.  0.  0.  0.  0. 10.  0.]
 [-2.  0.  0.  0.  0.  0.  0.  0.  0. 10.]]

При X = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] , b = [8. 8. 8. 8. 8. 8. 8. 8. 8. 8.]

После 8 итерации был подобран Xk = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Общее время работы процесса: 0.0009970664978027344 seconds
```

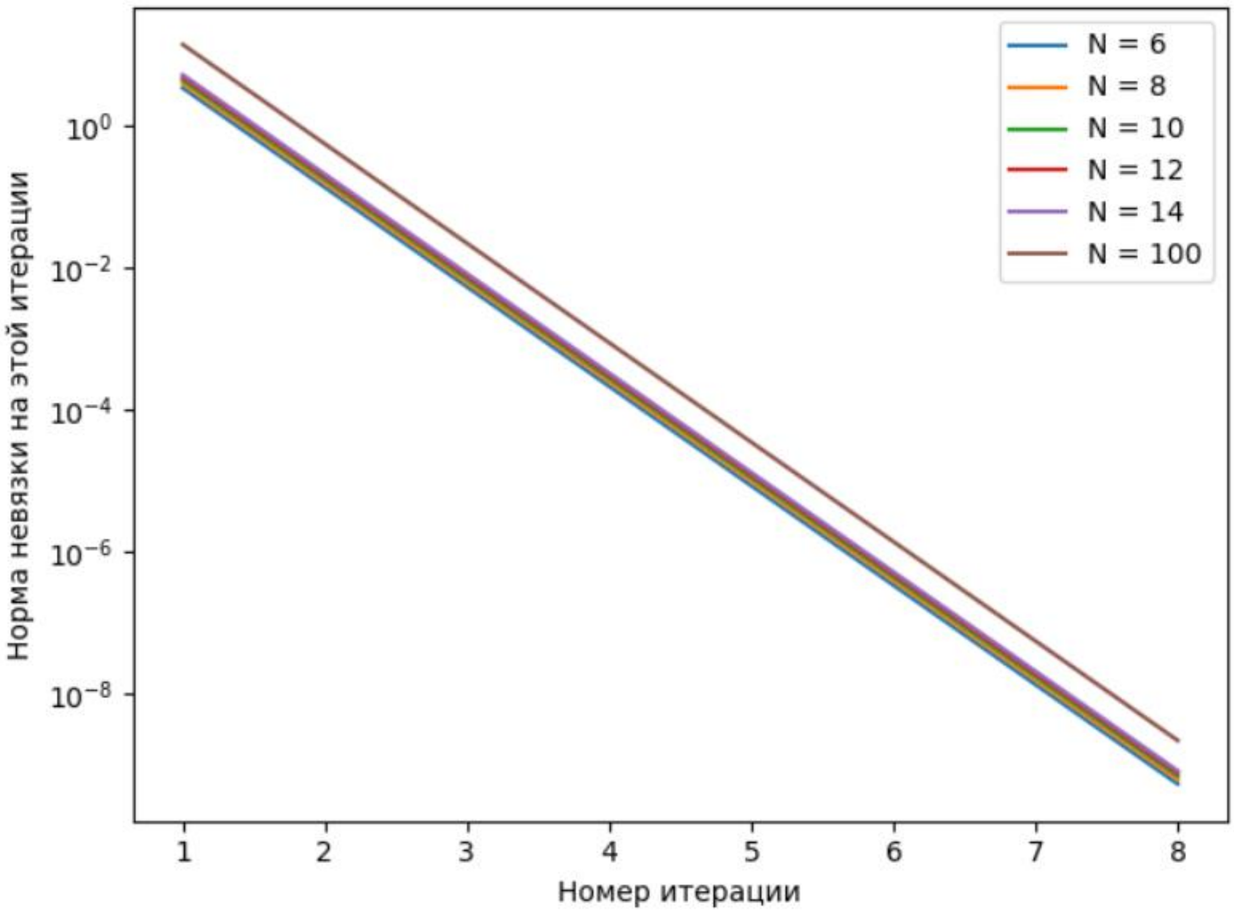
```
Размерность текущей матрицы N = 12

A =
 [[10.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -2.]
 [ 0. 10.  0.  0.  0.  0.  0.  0.  0.  0. -2.  0.]
 [ 0.  0. 10.  0.  0.  0.  0.  0.  0. -2.  0.  0.]
 [ 0.  0.  0. 10.  0.  0.  0.  0. -2.  0.  0.  0.]
 [ 0.  0.  0.  0. 10.  0.  0. -2.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 10. -2.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. -2. 10.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. -2.  0.  0. 10.  0.  0.  0.  0.]
 [ 0.  0.  0. -2.  0.  0.  0.  0. 10.  0.  0.  0.]
 [ 0.  0. -2.  0.  0.  0.  0.  0.  0. 10.  0.  0.]
 [ 0. -2.  0.  0.  0.  0.  0.  0.  0.  0. 10.  0.]
 [-2.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. 10.]]

При X = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] , b = [8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8.]

После 8 итерации был подобран Xk = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Общее время работы процесса: 0.0009961128234863281 seconds



Размерность текущей матрицы N = 14
```

```
A =
 [[10.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -2.]
 [ 0. 10.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -2.  0.]
 [ 0.  0. 10.  0.  0.  0.  0.  0.  0.  0.  0. -2.  0.  0.]
 [ 0.  0.  0. 10.  0.  0.  0.  0.  0.  0. -2.  0.  0.  0.]
 [ 0.  0.  0.  0. 10.  0.  0.  0.  0. -2.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 10.  0.  0. -2.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. 10. -2.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. -2. 10.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. -2.  0.  0. 10.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. -2.  0.  0.  0.  0. 10.  0.  0.  0.  0.]
 [ 0.  0.  0. -2.  0.  0.  0.  0.  0.  0. 10.  0.  0.  0.]
 [ 0.  0. -2.  0.  0.  0.  0.  0.  0.  0.  0. 10.  0.  0.]
 [ 0. -2.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. 10.  0.]
 [-2.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. 10.]]

При X = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] , b = [8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8.]

После 8 итерации был подобран Xk = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Общее время работы процесса: 0.0019948482513427734 seconds


Размерность текущей матрицы N = 100
```

```
A =
 [[10.  0.  0. ...  0.  0. -2.]
 [ 0. 10.  0. ...  0. -2.  0.]
 [ 0.  0. 10. ... -2.  0.  0.]
 ...
 [ 0.  0. -2. ... 10.  0.  0.]
 [ 0. -2.  0. ...  0. 10.  0.]
 [-2.  0.  0. ...  0.  0. 10.]]

При X = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1.] , b = [8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8.
 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8.
 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8.
 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8. 8.
 8. 8. 8. 8.]

После 8 итерации был подобран Xk = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1.]

Общее время работы процесса: 0.05037426948547363 seconds


Process finished with exit code 0
```

Можно наблюдать, что программа достаточно точно нашла истинное решение, при чём достаточно быстро, что говорит о корректности её работы.

Помимо всего этого, программа также выдала на экран диаграмму сходимости для всех вышеуказанных матриц:




На ней мы можем видеть, что все процессы успешно сошлись, при чём каждому потребовалось 8 итераций, чтобы достигнуть нужной точности (я выбрал точность Epsilon = $10^{-8}$).

Исходный код прикреплю в файле Task_5_Main.py, а картинку с диаграммой в файле Task_5_Diagram.png.