МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра вычислительной математики

Харченко Роман Кириллович

Отчёт по лабораторной работе №3, вариант 15

(«Методы вычислений»)

Студента 2 курса 12 группы

Преподаватель

Бондарь Иван Васильевич

Минск 2020

Условие:

Вариант 15

15. Задание 3 (15) + Задание 5 (метод 3, задача 1)

Задание 3. Метод релаксации 2

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)
2. Экспериментально подобрать значение параметра $\omega = \omega^*$, при котором сходимость будет наиболее быстрой.
3. Для подтверждения своего вывода построить совмещенную диаграмму сходимости для как минимум пяти различных значений ω (включая ω^*).
4. Теоретически доказать сходимость метода релаксации при $\omega = \omega^*$.

Матрица:

15. $\begin{pmatrix} -2 & -2 & -3 \\ 0 & -4 & 0 \\ 4 & 0 & -1 \end{pmatrix}$

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}.

3. Метод релаксации (параметр ω подобрать экспериментально)

Матрица:

1. Матрицы A_n вида $A_5 = \begin{pmatrix} 5 & 1 & 1 & 1 & 1 \\ 1 & 5 & 0 & 0 & 1 \\ 1 & 0 & 5 & 0 & 1 \\ 1 & 0 & 0 & 5 & 1 \\ 1 & 1 & 1 & 1 & 5 \end{pmatrix}$, $A_7 = \begin{pmatrix} 7 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 7 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 7 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 7 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 7 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 7 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 7 \end{pmatrix}$ и т. д.: по диагонали — размерность матрицы, по краям — единицы.

Задание 3 (15):

Я написал программу на Python, которая позволяет решать СЛАУ вида Ax = b методом релаксации. Для этого в качестве вектора X я взял вектор (10, 10, 10), и соответственно посчитал для него вектор b:

b1 = (-2)*10 + (-2)*10 + (-3)*10 = -20 -20 – 30 = -70

b2 = 0*10 + (-4)*10 + 0*10 = -40

b3 = 4*10 + 0*10 + (-1)*10 = 40 – 10 = 30

Таким образом, мой вектор b = (-70, -40, 30).

В качестве начального приближения я взял вектор $X_0 = (0, 0, 0)$. Именно начиная с этого вектора все процессы стремились к вектору-ответу X = (10, 10, 10).

Наверное, прежде чем я расскажу о своих рассуждения и выводах, которые я получил во время выполнения программы на разных данных, я предоставлю фрагмент кода с реализацией:

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

"""
Лаба 3, Вариант 15, Задание 3, Матрица 15
Задание 3. Метод релаксации 2
Дана матрица А (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-нибуь заданному значению x).
2. Экспериментально подобрать значение параметра w = w*, при котором сходимость
будет наиболее быстрой.
3. Для подтверждения своего вывода построить совмещенную диаграмму сходимости
для как минимум пяти различных
значений w (включая w*).
4. Теоретически доказать сходимость метода релаксации при w = w*.
"""

# Требуемая точность
Epsilon = 0.00000000001

# Размерность
N = 3

# Заданная матрица A
A = np.array([[-2., -2., -3.],
              [0., -4., 0.],
              [4., 0., -1.]])

# Возьмём в качестве вектора-ответа вектор (10, 10, 10), а начальное приближение
(0, 0, 0):
X0 = np.array([0., 0., 0.])

# Тогда вектор b будет таким:
b = np.array([-70., -40., 30.])

# Значения w для экспериментов
w1 = 0.005
w2 = 0.01
w3 = 0.05
w4 = 0.1
w5 = 0.5  # <-- Вероятно, это наш w* (у него по наблюдениям лучшее время
сходимости)


#  || A*X = b || --> min (Норма невязки)
#  Посчитать норму невязки
def ResidualRate(X):
```

```python
    AX = np.dot(A, X)
    AX_b = AX - b
    return np.linalg.norm(AX_b)


# Решение СЛАУ методом релаксации (передаём омегу и массив, в котором будет собирать информацию о невязках на итерациях)
def RelaxationMethod(w, ResRateArr):
    StartTime = time.time()

    print("При X0 =", X0, "и w =", w)

    L = np.zeros((N, N))  # Нижнетреугольная матрица
    for i in range(N):  # да, наверняка есть встроенная в numpy функция, но мне было быстрее ручками прописать
        for j in range(i):
            L[i, j] = A[i, j]

    R = np.zeros((N, N))  # Верхнетреугольная матрица
    for i in range(N):  # да, наверняка есть встроенная в numpy функция, но мне было быстрее ручками прописать
        for j in range(i + 1, N):
            R[i, j] = A[i, j]

    D = np.zeros((N, N))  # Диагональная матрица
    for i in range(N):  # да, наверняка есть встроенная в numpy функция, но мне было быстрее ручками прописать
        for j in range(N):
            if i == j:
                D[i, j] = A[i, j]

    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)
    wObrDL = w*np.dot(ObrD, L)  # wD^(-1)L
    I_wObrDL = UnitMatrix + wObrDL  # I + wD^(-1)L
    I_wOBrDL_Obr = np.linalg.inv(I_wObrDL)  # (I + wD^(-1)L)^(-1)
    wObrDR = w*np.dot(ObrD, R)  # wD^(-1)R
    I_1_w_wObrDR = (1 - w)*UnitMatrix - wObrDR  # (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("Матрциа B =\n", B)

    NormI_wObrDL_Obr = np.linalg.norm(I_wOBrDL_Obr)  # Норма первой части
    print("Норма первой части - Norm((I + wD^(-1)L)^(-1)) =", NormI_wObrDL_Obr)
    NormI_1_w_wObrDR = np.linalg.norm(I_1_w_wObrDR)  # Норма второй части
    print("Норма второй части - Norm((1-w)I - wD^(-1)R) =", NormI_1_w_wObrDR)
    NormB = np.linalg.norm(B)  # Норма матрицы B
    print("Норма матрицы B =", NormB)
    EigenValuesB = np.linalg.eigvals(B)  # Вектор, хранящий в себе собственные значения матрицы B
    print("Собственные значения матрицы B\n", EigenValuesB)
    MaxEigenValueB = 0.
    for i in range(EigenValuesB.size):
        if abs(EigenValuesB[i]) > MaxEigenValueB:
            MaxEigenValueB = abs(EigenValuesB[i])
    print("Максимальное по модулю из собственных значений матрицы B =",
```

```
MaxEigenValueB)

    if NormI_wObrDL_Obr*NormI_1_w_wObrDR < 1.:  # Если произведение норм двух
частей < 1, то
        print("Произведение норм двух частей < 1.0, процесс сходится")
    elif NormB < 1.:  # Если норма самой матрицы B < 1, то
        print("Норма матрицы B < 1.0, процесс сходится")
    elif MaxEigenValueB < 1.:  # Если максимальное по модулю собственное
значение матрицы < 1, то
        print("Максимальное по модулю собственное значение матрицы B < 1.0,
процесс сходится")
    else:
        print("Произведение норм двух частей, Норма матрицы B и наибольшее по
модулю собственное её значение >= 1.0,"
              " процесс не сходится...")

    # 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 - 1):
                FirstSum += (A[i, j] * Xk[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("x1 =", Xk)

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

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

# 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(w1, 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(w2, 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(w3, 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(w4, 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(w5, ResRateArr5)
IterArr5 = np.arange(1, IterAmount5 + 1)  # Массив абсцисс (количества итераций) для графика 5-ого процесса
plt.semilogy(IterArr1, ResRateArr1, label = 'w1')
plt.semilogy(IterArr2, ResRateArr2, label = 'w2')
plt.semilogy(IterArr3, ResRateArr3, label = 'w3')
plt.semilogy(IterArr4, ResRateArr4, label = 'w4')
plt.semilogy(IterArr5, ResRateArr5, label = 'w5')
plt.xlabel("Номер итерации")
plt.ylabel("Норма невязки на этой итерации")
plt.legend()
plt.show()
```

Результат работы:

```
При X0 = [0. 0. 0.] и w = 0.005
Матрциа B =
 [[ 9.9500e-01 -5.0000e-03 -7.5000e-03]
 [ 0.0000e+00  9.9500e-01  0.0000e+00]
 [ 1.9900e-02 -1.0000e-04  9.9485e-01]]
Норма первой части - Norm((I + wD^(-1)L)^(-1)) = 1.73216627377397
Норма второй части - Norm((1-w)I - wD^(-1)R) = 1.7234141260880973
Норма матрицы B = 1.7234424250609592
Собственные значения матрицы B
 [0.994925+0.01221656j 0.994925-0.01221656j 0.995   +0.j        ]
Максимальное по модулю из собственных значений матрицы B = 0.9950000000000002
Максимальное по модулю собственное значение матрицы B < 1.0, процесс сходится
x1 = [10. 10. 10.]
Общее время работы процесса: 0.09945964813232422 seconds

При X0 = [0. 0. 0.] и w = 0.01
Матрциа B =
 [[ 9.900e-01 -1.000e-02 -1.500e-02]
 [ 0.000e+00  9.900e-01  0.000e+00]
 [ 3.960e-02 -4.000e-04  9.894e-01]]
Норма первой части - Norm((I + wD^(-1)L)^(-1)) = 1.732512626216617
Норма второй части - Norm((1-w)I - wD^(-1)R) = 1.7148250639642517
Норма матрицы B = 1.7149360571169994
Собственные значения матрицы B
 [0.9897+0.02437027j 0.9897-0.02437027j 0.99  +0.j        ]
Максимальное по модулю из собственных значений матрицы B = 0.99
Максимальное по модулю собственное значение матрицы B < 1.0, процесс сходится
x1 = [10. 10. 10.]
Общее время работы процесса: 0.046913862228393555 seconds

При X0 = [0. 0. 0.] и w = 0.05
```

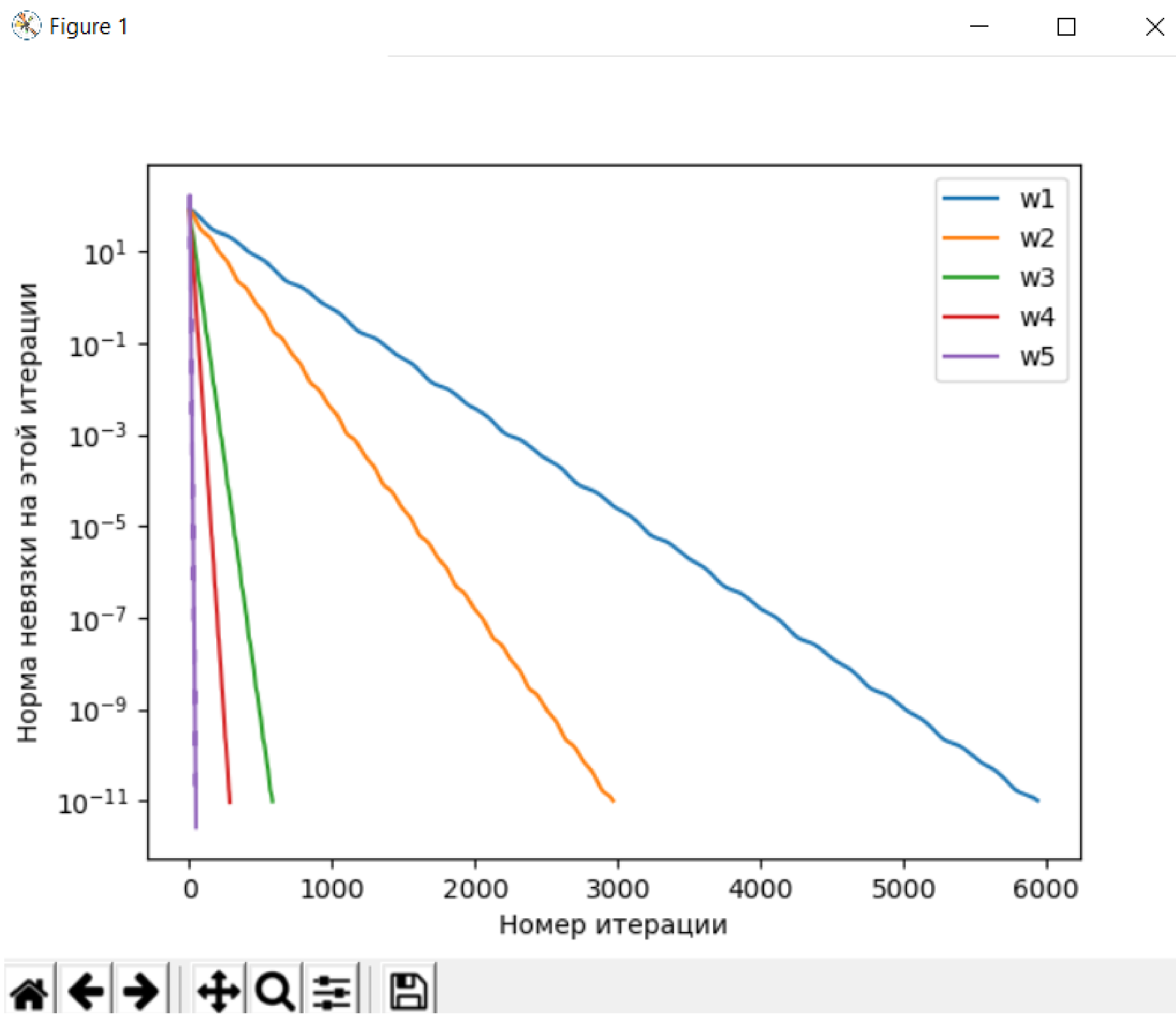
```
При X0 = [0. 0. 0.] и w = 0.05
Матрциа B =
 [[ 0.95  -0.05  -0.075]
 [ 0.     0.95   0.   ]
 [ 0.19  -0.01   0.935]]
Норма первой части - Norm((I + wD^(-1)L)^(-1)) = 1.7435595774162693
Норма второй части - Norm((1-w)I - wD^(-1)R) = 1.6479153497676997
Норма матрицы B = 1.6503181511454086
Собственные значения матрицы B
 [0.9425+0.11913753j 0.9425-0.11913753j 0.95  +0.j        ]
Максимальное по модулю из собственных значений матрицы B = 0.95
Максимальное по модулю собственное значение матрицы B < 1.0, процесс сходится
x1 = [10. 10. 10.]
Общее время работы процесса: 0.017914295196533203 seconds

При X0 = [0. 0. 0.] и w = 0.1
Матрциа B =
 [[ 0.9  -0.1  -0.15]
 [ 0.    0.9   0.  ]
 [ 0.36 -0.04  0.84]]
Норма первой части - Norm((I + wD^(-1)L)^(-1)) = 1.7776388834631178
Норма второй части - Norm((1-w)I - wD^(-1)R) = 1.5692354826475217
Норма матрицы B = 1.5777515647274765
Собственные значения матрицы B
 [0.87+0.23043437j 0.87-0.23043437j 0.9 +0.j        ]
Максимальное по модулю из собственных значений матрицы B = 0.9000000000000001
Максимальное по модулю собственное значение матрицы B < 1.0, процесс сходится
x1 = [10. 10. 10.]
Общее время работы процесса: 0.0060079097747802734 seconds

При X0 = [0. 0. 0.] и w = 0.5
```

```
При X0 = [0. 0. 0.] и w = 0.5
Матрциа B =
 [[ 0.5  -0.5  -0.75]
 [ 0.    0.5   0.  ]
 [ 1.   -1.   -1.  ]]
Норма первой части - Norm((I + wD^(-1)L)^(-1)) = 2.6457513110645907
Норма второй части - Norm((1-w)I - wD^(-1)R) = 1.25
Норма матрицы B = 2.0766559657295187
Собственные значения матрицы B
 [-0.25+0.4330127j -0.25-0.4330127j  0.5 +0.j       ]
Максимальное по модулю из собственных значений матрицы B = 0.5
Максимальное по модулю собственное значение матрицы B < 1.0, процесс сходится
x1 = [10. 10. 10.]
Общее время работы процесса: 0.0009970664978027344 seconds
```

Во время наблюдений я выяснил, что если брать w в диапазоне от нуля и где-то до 0,5, то процесс будет сходится, при чем чем ближе к 0,5, тем быстрее сходится процесс (хотя что w4 = 0.1, что w5 = 0.5 время было приблизительно одинаковым, но будет считать, что w5 чуть быстрее), а чем ближе к нулю, тем процесс сходится медленнее, мне это также показали диаграммы сходимости. Для каждого w из пяти я засёк количество необходимых итераций, а также невязку на каждой итерации и построил 5 диаграмм:



Мы можем наблюдать, что нашему w* = w5 = 0.5 потребовалось не так уж много итераций, чтобы сойтись до нужной точности, что свидетельствуют о высоких показателях быстроты сходимости.

В качестве точности (на которой итерационный процесс останавливает свою работу), к которой стремится значение невязки я взял:

```
# Требуемая точность
Epsilon = 0.00000000001
```

В качестве пяти значений омега (чтобы сравнить время их работы друг с другом), я взял вот такие значения:

```
# Значения w для экспериментов
w1 = 0.005
w2 = 0.01
w3 = 0.05
w4 = 0.1
w5 = 0.5  # <-- Вероятно, это наш w* (у него по наблюдениям лучшее время
сходимости)
```

В качестве w* я взял w5 = 0.5, так как экспериментально именно у него самое быстрое время сходимости. Кстати говоря, на всех этих w процессы сходятся, для этого я сделал три критерия проверки.

Во-первых, я проверял, чему равно произведение норм:

$$||(I + wD^{-1}L)^{-1}||_2 * ||(1-w)I - wD^{-1}R||_2$$

Если оно меньше единицы, то процесс сходится. Если же оно не меньше единицы, то я проверял, чему равна норма матрицы B:

$$||B||_2 = ||((I + wD^{-1}L)^{-1})*((1-w)I - wD^{-1}R)||_2$$

Если она меньше единицы, то процесс сходится. Если же и она не меньше единицы, то я смотрел на наибольшее по модулю собственное значение матрицы B. И если оно меньше единицы, то процесс сходится, иначе – не сходится:

```
if NormI_wObrDL_Obr*NormI_1_w_wObrDR < 1.:  # Если произведение норм двух частей
< 1, то
    print("Произведение норм двух частей < 1.0, процесс сходится")
elif NormB < 1.:  # Если норма самой матрицы B < 1, то
    print("Норма матрицы B < 1.0, процесс сходится")
elif MaxEigenValueB < 1.:  # Если максимальное по модулю собственное значение
матрицы < 1, то
    print("Максимальное по модулю собственное значение матрицы B < 1.0, процесс
сходится")
else:
    print("Произведение норм двух частей, Норма матрицы B и наибольшее по модулю
собственное её значение >= 1.0,"
          " процесс не сходится...")
```

В коде все эти штуки я подробно считаю.

Так вот, я работал со значениями w не большими, чем 0.5 (и понятное дело не меньшими, чем 0), так как при тех значениях процессы у меня не сходились:

При отрицательном w:

```
При X0 = [0. 0. 0.] и w = -0.005
Матрциа B =
 [[ 1.00500e+00  5.00000e-03  7.50000e-03]
 [ 0.00000e+00  1.00500e+00  0.00000e+00]
 [-2.01000e-02 -1.00000e-04  1.00485e+00]]
Норма первой части - Norm((I + wD^(-1)L)^(-1)) = 1.73216627377397
Норма второй части - Norm((1-w)I - wD^(-1)R) = 1.7407343996141396
Норма матрицы B = 1.7407638531690617
Собственные значения матрицы B
 [1.004925+0.0122778j 1.004925-0.0122778j 1.005   +0.j       ]
Максимальное по модулю из собственных значений матрицы B = 1.005
Произведение норм двух частей, Норма матрицы B и наибольшее по модулю собственное её значение >= 1.0, процесс не сходится...
C:/Users/user/PycharmProjects/CalcMethods_Lab_3_V15_Task_3_15/main_1.py:119: RuntimeWarning: overflow encountered in double_scalars
  FirstSum += (A[i, j] * Xk[j])
x1 = [-4.50079699e+307 -3.14585388e+307             inf]
Общее время работы процесса: 2.400629997253418 seconds
```

При w > 0.5:

```
При X0 = [0. 0. 0.] и w = 0.6
Матрциа B =
 [[ 0.4  -0.6  -0.9 ]
 [ 0.    0.4   0.  ]
 [ 0.96 -1.44 -1.76]]
Норма первой части - Norm((I + wD^(-1)L)^(-1)) = 2.9597297173897483
Норма второй части - Norm((1-w)I - wD^(-1)R) = 1.284523257866513
Норма матрицы B = 2.7536884355351456
Собственные значения матрицы B
 [-0.13009092 -1.22990908  0.4       ]
Максимальное по модулю из собственных значений матрицы B = 1.2299090833947006
Произведение норм двух частей, Норма матрицы B и наибольшее по модулю собственное её значение >= 1.0, процесс не сходится...
x1 = [inf 10. inf]
Общее время работы процесса: 0.05684804916381836 seconds
```

Таким образом, процесс быстрее всего сходился при w* = w5 = 0.5. И хотя моя программа проверяет сходимость при этом значении, я также доказал это теоретически на листочке:

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

Омега --> $w = \frac{1}{2}$

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

$$L = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 0 & 0 \end{pmatrix}, \quad D = \begin{pmatrix} -2 & 0 & 0 \\ 0 & -4 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \quad D^{-1} = \begin{pmatrix} -\frac{1}{2} & 0 & 0 \\ 0 & -\frac{1}{4} & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad R = \begin{pmatrix} 0 & -2 & -3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$w \cdot D^{-1} \cdot L = \frac{1}{2} \cdot \begin{pmatrix} -\frac{1}{2} & 0 & 0 \\ 0 & -\frac{1}{4} & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -2 & 0 & 0 \end{pmatrix}$$

$$I + wD^{-1}L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -2 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix}$$

ОБРАЩАЕМ:

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

$$(I + wD^{-1}L)^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix}$$

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

$$= \frac{1}{2} \cdot \left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 & \frac{3}{2} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right) = \frac{1}{2} \cdot \begin{pmatrix} 1 & -1 & -\frac{3}{2} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & -\frac{3}{4} \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{pmatrix}$$

$$\|(I + wD^{-1}L)^{-1}\|_2 = \left\| \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix} \right\|_2 = \sqrt{1^2 + 1^2 + 1^2 + 2^2} = \sqrt{7}.$$

$$\|((1-w)I - wD^{-1}R)\|_2 = \left\| \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & -\frac{3}{4} \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{pmatrix} \right\|_2 = \sqrt{\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{9}{16}} = \frac{5}{4}$$

$$\|(I + wD^{-1}L)^{-1}\|_2 \cdot \|(1-w)I - wD^{-1}R\|_2 = \sqrt{7} \cdot \frac{5}{4} = \frac{5\sqrt{7}}{4}.$$

$\frac{5\sqrt{7}}{4} > 1 \Rightarrow$ Придется искать $\|B\|_2$.

$$\|B\|_2 = \left\| \begin{pmatrix}1&0&0\\0&1&0\\2&0&1\end{pmatrix} \cdot \begin{pmatrix} \frac12 & -\frac12 & -\frac34 \\ 0 & \frac12 & 0 \\ 0&0&\frac12\end{pmatrix}\right\|_2 = \left\|\begin{pmatrix}\frac12&-\frac12&-\frac34\\0&\frac12&0\\1&-1&-1\end{pmatrix}\right\|_2 = \sqrt{\frac14+\frac14+\frac9{16}+\frac14+1+1+1} = \sqrt{\frac{21}{16}+\frac{48}{16}} = \sqrt{\frac{69}{16}} = \frac{\sqrt{69}}{4}.$$

$\frac{\sqrt{69}}{4} > 1 \Rightarrow$ придется искать собственные значения матрицы B.

$$B = \begin{pmatrix}\frac12&-\frac12&-\frac34\\0&\frac12&0\\1&-1&-1\end{pmatrix} \leftarrow \text{видно, что } \boxed{\lambda_1 = \frac12}$$

$$\begin{vmatrix}\frac12-\lambda & -\frac12 & -\frac34\\ 0 & \frac12-\lambda & 0\\ 1 & -1 & -1-\lambda\end{vmatrix} = \left(\frac12-\lambda\right)\left(\frac12-\lambda\right)\cdot(-1-\lambda)+0+0-1\cdot\left(\frac12-\lambda\right)\cdot\left(-\frac34\right)-$$

$$-0-0 = \left(\frac14-\lambda+\lambda^2\right)(-1-\lambda)+\frac38-\frac{3\lambda}{4}=0,$$

$$-\frac14+\lambda-\lambda^2-\frac14\lambda+\lambda^2-\lambda^3+\frac38-\frac34\lambda=0,$$

$$-\lambda^3+\frac18=0,$$

$$\lambda^3=\frac18,$$

$$\boxed{\lambda=\frac12}$$

$\frac12<1 \Rightarrow$ процесс сходится.

Максимальное по модулю собственное значение было равно 0.5, что меньше единицы, значит, можно сделать вывод, что процесс сходится. Кстати, можно заметить, что мои расчёты на листочке соответствуют выводу программы. Код программы будет в файле main_1.py, а рисунок графика в файле Task_3_15_График_Невязка(КоличествоИтераций).png.

Задание 5 (метод 3, задача 1)

Сразу следует сказать, что большая часть кода соответствует заданию 3, но из-за вида матрицы A_n и способа её хранения (полностью хранить запрещено), пришлось изменить алгоритм, а вместе с ним и код в целом.

Пока что в этом задании у меня наблюдаются некоторые проблемы. Первая проблема, это то, что разреженную матрицу своего вида я не храню, а постоянно генерирую с помощью функции GenerateSpecificMatrix по входному параметру N, что очень медленно (наверное). В качестве вектора-ответа я взял вектор X = (10, 10, …, 10, 10), в котором N десяток. Вектор b я генерирую путём умножения моей матрицы An на вектор X. В качестве w я взял w = 0.5 из предыдущего пункта (вроде как программа говорит, что на нём процессы сходятся). В качестве начального приближения беру X_0 = (0, 0, …, 0, 0). В общем, как и в предыдущем задании. Так вот, вторая проблема в том, что когда я начинаю тестировать, скажем для N = 5, процесс для точность E = 10^{-8} никак не может подобрать вектор X, хотя реализация вроде как точно такая же, как и предыдущем пункте. Я также сделал заготовку для диаграммы при N = 100, N = 1000, N = 10000, но пока программа просто не может до неё выполниться, ибо даже для N = 5 по какой-то причине не может подобрать вектор X. Код этой части лабы я оставлю в main_2.py.

Код:

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

"""
Лаба 3, Вариант 15, Задание 5, Метод 3, Задача 1
Задание 5. Итерационные методы для разреженных СЛАУ особого вида
Дана матрица A (указана в варианте, см. список 1 ниже).
1. Написать программу, которая при данном n решает СЛАУ Anx = bn указанным в
варианте методом (метод релаксации, где
параметр w выбирается экспериментально). 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: ||Anx^k - 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 ординат отложено
время работы, которое требуется, чтобы норма невязки не превышала 10^(-8).

ВИД МАТРИЦЫ:
1. Матрица An, где по диагонали - размерность матрицы (число n), а по краям -
единицы (между n и 1 ставятся нули).
"""

# Требуемая точность
Epsilon = 0.00000001  # 10^(-8)

# Значение w* = 0.5
w = 0.5


# Данная функция возвращает нужную форму нашей матрицы лишь по заданной
размерности N
def GenerateSpecificMatrix(N):
    Matrix = np.zeros((N, N))
    for i in range(N):
        Matrix[i][0] = 1
        Matrix[0][i] = 1
        Matrix[N - i - 1][N - 1] = 1
        Matrix[N - 1][N - i - 1] = 1
        Matrix[i][i] = N
    return Matrix


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


# Решение СЛАУ методом релаксации для особой матрицы (передаём омегу и массив, в
котором будет собирать информацию о
# невязках на итерациях, а также размерность N для особой матрицы)
def RelaxationMethod(w, ResRateArr, N):
    PrintInfoCheck = False  # Флажок, который будет разрешать/запрещать печать
информации о завершенном процесса

    if 4 < N < 11:  # Если у нас размерность из диапазона [5, 10], то мы выведем
информацию, чтобы свериться
        PrintInfoCheck = True

    StartTime = time.time()

    X = np.ones(N)  # В качестве вектора-ответа возьмём X = (10, 10, 10, ...,
10, 10, 10)
    X[:] = 10
```

```
    X0 = np.zeros(N)  # А в качестве начального приближения возьмём X = (0, 0,
0, ..., 0, 0, 0)

    if PrintInfoCheck:
        print("При X0 =", X0, ", w =", w, ", N =", N)

    if PrintInfoCheck:
        print("Особая матрица A =\n", GenerateSpecificMatrix(N))

    b = np.dot(GenerateSpecificMatrix(N), X)  # Расчитываем вектор b

    if PrintInfoCheck:
        print("При векторе-ответе X =", X, "вектор b =", b)

    L = np.zeros((N, N))  # Нижнетреугольная матрица
    for i in range(N):  # да, наверняка есть встроенная в numpy функция, но мне
было быстрее ручками прописать
        for j in range(i):
            L[i, j] = GenerateSpecificMatrix(N)[i, j]

    R = np.zeros((N, N))  # Верхнетреугольная матрица
    for i in range(N):  # да, наверняка есть встроенная в numpy функция, но мне
было быстрее ручками прописать
        for j in range(i + 1, N):
            R[i, j] = GenerateSpecificMatrix(N)[i, j]

    D = np.zeros((N, N))  # Диагональная матрица
    for i in range(N):  # да, наверняка есть встроенная в numpy функция, но мне
было быстрее ручками прописать
        for j in range(N):
            if i == j:
                D[i, j] = GenerateSpecificMatrix(N)[i, j]

    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)
    wObrDL = w*np.dot(ObrD, L)  # wD^(-1)L
    I_wObrDL = UnitMatrix + wObrDL  # I + wD^(-1)L
    I_wOBrDL_Obr = np.linalg.inv(I_wObrDL)  # (I + wD^(-1)L)^(-1)
    wObrDR = w*np.dot(ObrD, R)  # wD^(-1)R
    I_1_w_wObrDR = (1 - w)*UnitMatrix - wObrDR  # (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)
    if PrintInfoCheck:
        print("Матрциа B =\n", B)

    NormI_wObrDL_Obr = np.linalg.norm(I_wOBrDL_Obr)  # Норма первой части
    if PrintInfoCheck:
        print("Норма первой части - Norm((I + wD^(-1)L)^(-1)) =",
NormI_wObrDL_Obr)
    NormI_1_w_wObrDR = np.linalg.norm(I_1_w_wObrDR)  # Норма второй части
    if PrintInfoCheck:
        print("Норма второй части - Norm((1-w)I - wD^(-1)R) =",
NormI_1_w_wObrDR)
    NormB = np.linalg.norm(B)  # Норма матрицы B
    if PrintInfoCheck:
```

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

    if PrintInfoCheck:
        if NormI_wObrDL_Obr*NormI_1_w_wObrDR < 1.:  # Если произведение норм
двух частей < 1, то
            print("Произведение норм двух частей < 1.0, процесс сходится")
        elif NormB < 1.:  # Если норма самой матрицы B < 1, то
            print("Норма матрицы B < 1.0, процесс сходится")
        elif MaxEigenValueB < 1.:  # Если максимальное по модулю собственное
значение матрицы < 1, то
            print("Максимальное по модулю собственное значение матрицы B < 1.0,
процесс сходится")
        else:
            print("Произведение норм двух частей, Норма матрицы B и наибольшее
по модулю собственное её значение >= 1.0,"
                  " процесс не сходится...")

    # 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(N, Xk, b)  # 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 - 1):
                FirstSum += (GenerateSpecificMatrix(N)[i, j] * Xk[j])

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

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

        Xk = Xk_1  # Говорим, что вектор Xk+1 в следующей итерации будет просто
Xk
        CurrResRate = ResidualRate(N, Xk, b)  # Текущая невязка
        ResRateArr.append(CurrResRate)  # Добавляем текущую невязку в список
невязок для графика
    if PrintInfoCheck:
        print("x1 =", Xk)

    if PrintInfoCheck:
```

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

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

# Делаем итерации для размерностей с 5 по 10, чтобы свериться
for i in range(5, 11):
    N = i
    RelaxationMethod(w, [], N)


ResRateArr1 = []  # Список ординат (норм невязки на разных итерациях) для
графика 100-ого процесса
IterAmount1 = RelaxationMethod(w, ResRateArr1, 100)
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 1000-ого процесса
IterAmount2 = RelaxationMethod(w, ResRateArr2, 1000)
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 10000-ого процесса
IterAmount3 = RelaxationMethod(w, ResRateArr3, 10000)
IterArr3 = np.arange(1, IterAmount3 + 1)  # Массив абсцисс (количества итераций)
для графика 3-его процесса
plt.semilogy(IterArr1, ResRateArr1, label = 'n = 100')
plt.semilogy(IterArr2, ResRateArr2, label = 'n = 1000')
plt.semilogy(IterArr3, ResRateArr3, label = 'n = 10000')
plt.xlabel("Номер итерации")
plt.ylabel("Норма невязки на этой итерации")
plt.legend()
plt.show()
```