БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет прикладной математики и информатики

Никончик Даниил Викторович

ОТЧЕТ ПО МЕТОДАМ ВЫЧИСЛЕНИЙ

студента 2 курса 13 группы

Лабораторная работа №2

Преподаватель
Бондарь И.В.

Минск 2021

Вариант:

5. Задание 2(5) + Задание 5(метод 2, задача 4 В)

Постановка задания 2(5):

Метод релаксации 1

Дана Матрица: $\begin{matrix} -1 & 1 & -1 \\ -1 & 4 & -2 \\ 2 & -3 & -5 \end{matrix}$

1. Написать программу, которая решает СЛАУ Ax = b методом релаксации (в качестве вектора b взять вектор, соответствующий какому-нибудь заданному значению. Экспериментально подобрать значение параметра w, при котором итерационный процесс сходится w_1), а также значение, при котором он расходится (w_0).
2. Путем теоретического анализа подтвердить сходимость и расходимость.
3. Построить логарифмическую диаграмму сходимости (совмещенную для w = w_0, w_1, w = 1 и еще двух любых значений от 0 до 2.

Объяснение происходящего в коде:

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. Я также выбрал вот такое начальное приближение: x_0 = (0, 0, 0).

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 =  [[-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
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.

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.
```

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, то я смотрел на норму самой матрицы B, которую получал путем произведения двух её частей. 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.

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).

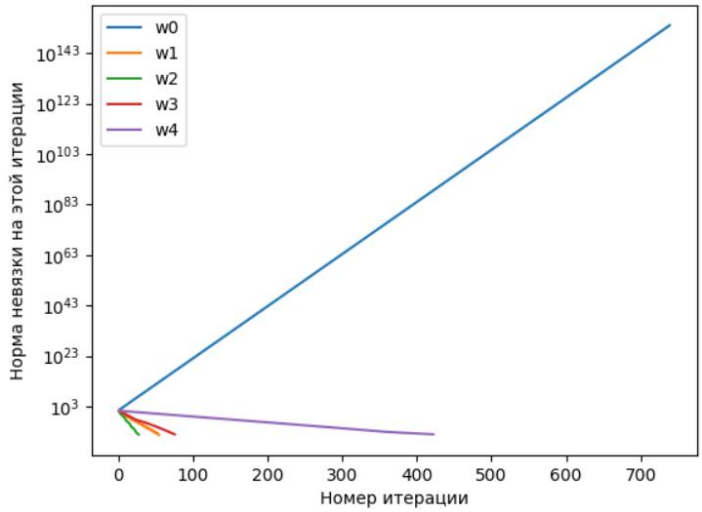
В конце концов программа закончила свои расчёты для каждого параметра релаксации (в случае с w_0 она остановила расчёты перед переполнением стека) вывела следующее:

```
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
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...
После 1488 итерации был подобран X = [-1.15598642e+308 -8.09298537e+307             nan]
Общее время работы процесса: 0.10024189949035645 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: 0.015665292739868164 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: 0.0 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: 0.015620946884155273 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: 0.0 seconds
```

Наши теоретические расчёты для параметров релаксации w_0 и w_1 полностью подтвердились (в том числе и промежуточные расчёты). Также 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 задали (а я задал Epsilon = 10^{-8}) выполняется, значит программа работает корректно.

Программа также вывела следующую логарифмическую диаграмму сходимости:



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.

Постановка задания 5(метод 2, задача 4 В):

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 разреженные матрицы размерности 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: $||A_n x^k - b_n|| < \xi$
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.
   Мой вариант метода:
    - Метод Гаусса-Зейделя

Вид Матрицы:

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

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 = 10. Таким образом, у меня были параметры a = 1, b = -2.

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:

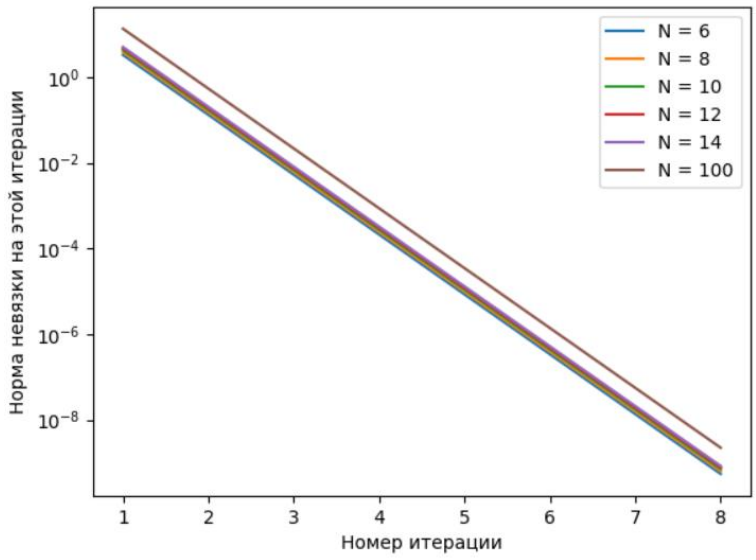
```
Размерность текущей матрицы N = 6
A =
 [[10.  0.  0.  0.  0. -2.]
 [ 0. 10.  0.  0. -2.  0.]
 [ 0.  0. 10. -2.  0.  0.]
 [ 0.  0. -2. 10.  0.  0.]
 [ 0. -2.  0.  0. 10.  0.]
 [-2.  0.  0.  0.  0. 10.]]
При X = [1. 1. 1. 1. 1. 1.]
 b = [8. 8. 8. 8. 8. 8.]
После 8 итерации был подобран Xk = [1. 1. 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: 0.0 seconds
Размерность текущей матрицы N = 12
A =
 [[10.  0.  0.  0.  0.  0.  0.  0.  0.  0. -2.]
 [ 0. 10.  0.  0.  0.  0.  0.  0.  0. -2.  0.]
 [ 0.  0. 10.  0.  0.  0.  0.  0. -2.  0.  0.]
 [ 0.  0.  0. 10.  0.  0.  0. -2.  0.  0.  0.]
 [ 0.  0.  0.  0. 10.  0.  0. -2.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 10. -2.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. -2. 10.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. -2.  0.  0. 10.  0.  0.  0.]
 [ 0.  0.  0. -2.  0.  0.  0.  0. 10.  0.  0.]
 [ 0.  0. -2.  0.  0.  0.  0.  0.  0. 10.  0.]
 [ 0. -2.  0.  0.  0.  0.  0.  0.  0.  0. 10.]
 [-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: 0.0065081119537353516 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: 0.0 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: 0.03124094009399414 seconds
```

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}).

Код задания 2(5):

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

```
import time

# Размерность матрицы(NxN)
N = 3

# Номер итерационного процесса
ProcessNum = 1

# Требуемая точность (для итераций)
Epsilon = 0.00000001

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

# 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


#  Вычисление нормы 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----------------------------------------------------------------------
----------------------\n")

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

    StartTime = time.time()

    print("Заданная точность Epsilon =", Epsilon)
    print("Начальное приближение X0 =", X0)
    print("Параметр релаксации w =", w)

    L = np.tril(A, k=-1)  # Нижнетреугольная матрица

    R = np.triu(A, k=1)  # Верхнетреугольная матрица

    D = np.diag(np.diag(A))  # Диагональная матрица

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

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

```
    I_wOBrDL_Obr = np.linalg.inv(UnitMatrix + w * np.dot(ObrD, L))
    I_1_w_wObrDR = (1 - w) * UnitMatrix - w * np.dot(ObrD, R)

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

    NormI_wObrDL_Obr = np.linalg.norm(I_wOBrDL_Obr)  # Норма первой части
    print("1) Norm((I + wD^(-1)L)^(-1)) =", NormI_wObrDL_Obr)

    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)

    BothPartsMult = NormI_wObrDL_Obr * NormI_1_w_wObrDR  # Произведение обеих
частей

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

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

        if NormB < 1.:
            print("Норма матрицы B =", NormB, "< 1.0 => процесс сходится.")
        else:
            print("Норма матрицы B =", NormB, ">= 1.0, требуются дальнейшие
исследования...")

            EigenValuesB = np.linalg.eigvals(B)  # Вектор, хранящий в себе
собственные значения матрицы B
            print("Собственные значения матрицы B: ", EigenValuesB)
            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)

            if MaxEigenValueB < 1.:
                print("Наибольшее по модулю собственное значение матрицы B
=", MaxEigenValueB,
                      "< 1.0 => процесс сходится.")
            else:
                print("Наибольшее по модулю собственное значение матриц =",
MaxEigenValueB,
                      ">= 1.0 => процесс расходится.")

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

    Xk = X0  # Для нахождения вектора Xk+1 в последующих итерациях
    Xk_1 = np.zeros(N)  # Xk+1 - следующий вектор-ответ
    IterationsAmount = 0  # Количество итераций
    CurrResRate = ResidualRate(Xk)  # Текущая невязка
    while CurrResRate > Epsilon:
        IterationsAmount += 1
        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  # 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)

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

    return IterationsAmount


print("\nМатрица A = ", A)
print("Вектор b = ", b)
ResRateArr1 = []  # Список ординат для графика 1-ого процесса
IterAmount1 = RelaxationMethod(w0, ResRateArr1)
IterArr1 = np.arange(1, IterAmount1 + 1)  # Массив абсцисс для графика 1-ого
процесса
ResRateArr2 = []  # Список ординат для графика 2-ого процесса
IterAmount2 = RelaxationMethod(w1, ResRateArr2)
IterArr2 = np.arange(1, IterAmount2 + 1)  # Массив абсцисс для графика 2-ого
процесса
ResRateArr3 = []  # Список ординат для графика 3-его процесса
IterAmount3 = RelaxationMethod(w2, ResRateArr3)
IterArr3 = np.arange(1, IterAmount3 + 1)  # Массив абсцисс для графика 3-его
процесса
ResRateArr4 = []  # Список ординат для графика 3-его процесса
IterAmount4 = RelaxationMethod(w3, ResRateArr4)
IterArr4 = np.arange(1, IterAmount4 + 1)  # Массив абсцисс для графика 4-ого
процесса
ResRateArr5 = []  # Список ординат для графика 5-ого процесса
IterAmount5 = RelaxationMethod(w4, ResRateArr5)
IterArr5 = np.arange(1, IterAmount5 + 1)  # Массив абсцисс 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()
```

Код задания 5(метод 2, задача 4 В):

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

# Требуемая точность (для итераций)
Epsilon = 0.00000001

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

# Функция генерирующая нашу матрицу по заданной размерности
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

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

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

    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)
    CurrRateArr = []  # Список ординат (норм невязки на разных итерациях) для
графика матрицы размерности i
    A = GenerateSpecificMatrix(i, MatrixParameter_a, MatrixParameter_b)
    print("A =\n", A)
    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, "\n b =", b)
    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()
```