Федеральное государственное автономное образовательное
учреждение высшего образования
«Санкт-Петербургский национальный исследовательский
Университет ИТМО»

Мегафакультет Трансляционных информационных технологий
Факультет информационных технологий и программирования

Лабораторная работа №4.

По дисциплине «Прикладная математика»

Методы решения СЛАУ

Выполнили:
Студенты M32061
Величко Максим Иванович, 334786
Гусев Андрей Александрович, 336515
Бонет Станислав, 334349

Проверила:
Преподаватель практики
Гомозова Валерия Эдуардовна

_____

Санкт-Петербург, 2023

1. *Реализация метода Гаусса с выбором ведущего элемента для решения СЛАУ*

Метод заключается в том, что приводит матрицу к ступенчатому виду, т. е. получение нулей под главной диагональю и нахождение неизвестных снизу вверх.

Числа, на которые производится деление в методе Гаусса, называются ведущими или главными элементами. Таким образом, в начале каждого этапа прямого хода решения системы следует добавить логику перестановки строк для выполнения приведенного условия.
Чтобы уменьшить влияние ошибок округления и исключить деление на нуль на каждом этапе прямого хода, уравнения системы переставляют так, чтобы деление проводилось на наибольший по модулю в данном столбце элемент.

```python
def gaussian_elimination(A, b):
    n = len(A)

    # Прямой ход
    for i in range(n):
        # Выбор ведущего элемента
        max_index = i
        for j in range(i + 1, n):
            if abs(A[j][i]) > abs(A[max_index][i]):
                max_index = j
        A[[i, max_index]] = A[[max_index, i]]
        b[[i, max_index]] = b[[max_index, i]]

        # Приведение матрицы к треугольному виду
        for j in range(i + 1, n):
            factor = A[j][i] / A[i][i]
            A[j] -= factor * A[i]
            b[j] -= factor * b[i]

    # Обратный ход
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (b[i] - np.dot(A[i][i + 1:], x[i + 1:])) / A[i][i]

    return x
```

Проведем исследование на системах с матрицами A^k :
Берем значения для k[1..3], причем, генерируем матрицы такого вида, K = 1:

```
Matrix A^(k):
[[11.1 -2.  -2.  -4.  -3. ]
 [-3.  11.  -2.  -4.  -2. ]
 [-2.  -3.  12.  -3.  -4. ]
 [-2.  -4.  -1.  10.  -3. ]
 [-1.  -4.  -3.  -3.  11. ]]
Condition number: 1008.588161192836
Gauss solution: [133.48766583 134.46987724 134.61390445 134.5383215  134.79741379]
Gauss error: 298.25173333433844
```

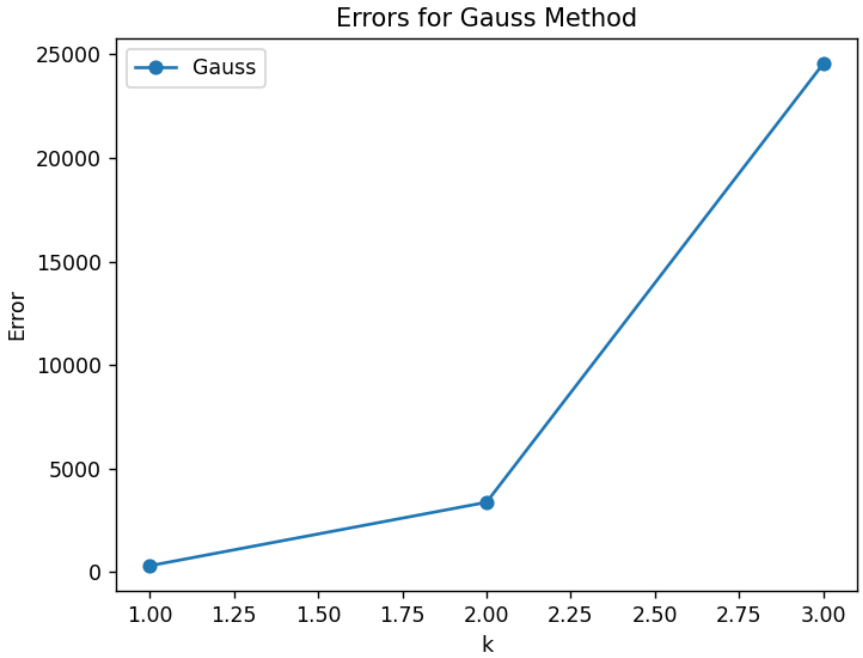
K = 2:

```
Matrix A^(k):
[[11.01 -3.   -1.   -3.   -4.  ]
 [-2.   10.   -3.   -4.   -1.  ]
 [-1.   -1.    7.   -3.   -2.  ]
 [-2.   -1.   -1.    8.   -4.  ]
 [-3.   -1.   -3.   -2.    9.  ]]
Condition number: 9593.956377472812
Gauss solution: [1504.54247741 1505.7834803  1505.99524346 1505.77816088 1505.83312732]
Gauss error: 3364.357894355949
```

K = 3:

```
Matrix A^(k):
[[11.001 -3.    -1.    -4.    -3.   ]
 [-1.     9.    -4.    -3.    -1.   ]
 [-4.    -3.    13.    -4.    -2.   ]
 [-4.    -1.    -4.    12.    -3.   ]
 [-1.    -2.    -2.    -3.     8.   ]]
Condition number: 93419.01785906446
Gauss solution: [10984.12801004 10985.05397712 10984.88096171 10984.88503375
 10985.26973679]
Gauss error: 24560.620833425135
```

График зависимости ошибок от K



Число обусловленности квадратных матриц возрастает с увеличением параметра k, что приводит к ухудшению точности решения. Однако точность решения при использовании метода Гаусса с выбором ведущего элемента остается в среднем стабильной при изменении параметра k до определённого значения. Это объясняется тем, что данный метод позволяет избежать проблемы плохой обусловленности матрицы и гарантирует стабильность решения системы линейных уравнений. Погрешность решения увеличивается с увеличением числа обусловленности матрицы.

2. *Реализация алгоритма LU – разложения с использованием разреженно – строчного формата хранения матрицы, а также метода решения СЛАУ с использованием LU – разложения.*

Суть метода заключается в том, что матрица коэффициентов **A** представляется в виде произведения матриц **L** и **U**, где **L** – нижнетреугольная матрица, **U** – верхнетреугольная матрица, все диагональные элементы которой равны 1. Вектор B в ходе разложения не изменяется.

Алгоритм

1. Создаем матрицы
$$L = \begin{pmatrix} l_{1,1} & 0 & 0 \\ l_{2,1} & l_{2,2} & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} \end{pmatrix}$$
и
$$U = A = \begin{pmatrix} 10 & -7 & 0 \\ -3 & 6 & 2 \\ 5 & -1 & 5 \end{pmatrix}$$
2. Для каждого столбца j = 1… 3 матрицы L будем вычислять $l_{i,j}$ как
$$l_{i,j} = \frac{u_{j,j}}{u_{i,j}}$$

   Для каждой строки c_i вычислим $c_i = c_i - l_{i,j} \cdot c_j$
3. Выполняем шаг 2 пока j<=3
4. Получем
$$L = \begin{pmatrix} 1 & 0 & 0 \\ l_{2,1} & 1 & 0 \\ l_{3,1} & l_{3,2} & 1 \end{pmatrix}$$
и
$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ 0 & u_{2,2} & u_{2,3} \\ 0 & 0 & u_{3,3} \end{pmatrix}$$
Такие, что A=L•U

Вот код, который реализует данный алгоритм:

```
def lu_decomposition_sparse(A):
    n = A.shape[0]
    L = csr_matrix((n, n), dtype=float)
    U = csr_matrix((n, n), dtype=float)

    for k in range(n):
        L[k, k] = 1.0
        U[k, k] = A[k, k] - L[k, :k].dot(U[:k, k].toarray().flatten())

        for j in range(k + 1, n):
            U[k, j] = A[k, j] - L[k, :k].dot(U[:k, j].toarray().flatten())

        for i in range(k + 1, n):
            L[i, k] = (A[i, k] - L[i, :k].dot(U[:k, k].toarray().flatten())) / U[k, k]

    return L, U
```

Проведем исследование на системах с матрицами A^k :

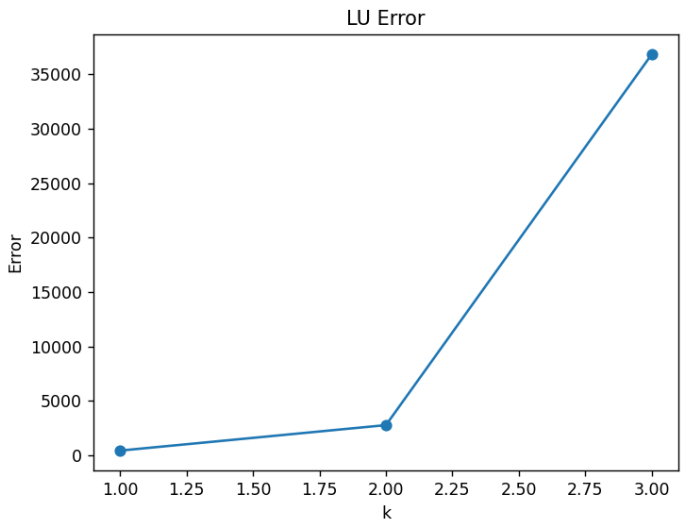
Берем значения для k[1..3], причем, генерируем матрицы такого вида, как и методе Гаусса, но получаем уже такие результаты:

```
==== k = 1 ====
Matrix A^(k):
[[10.1 -3.  -4.  -1.  -2. ]
 [-3.  14.  -4.  -3.  -4. ]
 [-1.  -2.   7.  -3.  -1. ]
 [-4.  -2.  -1.   9.  -2. ]
 [-4.  -4.  -2.  -3.  13. ]]
Condition number: 857.649916408113
Gauss solution: [135.23214858 136.35991034 136.63464617 136.3509446  136.43772014]
Gauss error: 302.3252991795029
LU solution: [135.23214858 136.35991034 136.63464617 136.3509446  136.43772014]
LU error: 302.3252991795072
```

```
==== k = 2 ====
Matrix A^(k):
[[ 9.01 -1.   -4.   -1.   -3.  ]
 [-1.    7.   -1.   -3.   -2.  ]
 [-3.   -3.   11.   -2.   -3.  ]
 [-4.   -3.   -3.   12.   -2.  ]
 [-1.   -3.   -3.   -4.   11.  ]]
Condition number: 7965.646989343022
Gauss solution: [1522.49886827 1524.10593029 1523.95473065 1523.89769126 1524.29741965]
Gauss error: 3404.974889682394
LU solution: [1522.49886827 1524.10593028 1523.95473065 1523.89769126 1524.29741965]
LU error: 3404.974889680644
```

```
==== k = 3 ====
Matrix A^(k):
[[11.001 -2.    -2.    -4.    -3.   ]
 [-3.    10.    -2.    -3.    -2.   ]
 [-2.    -3.    13.    -4.    -4.   ]
 [-4.    -4.    -2.    13.    -3.   ]
 [-1.    -2.    -1.    -1.     5.   ]]
Condition number: 110562.89643639071
Gauss solution: [20195.92134006 20197.37363431 20197.64675886 20197.45666417
 20198.15440634]
Gauss error: 45160.323339316186
LU solution: [20195.9213401  20197.37363435 20197.6467589  20197.45666421
 20198.15440638]
LU error: 45160.32333940403
```

И получаем такой график:



LU-разложение является модификацией метода Гаусса. Поэтому, в общем и целом, можно отметить, что значение ошибок будут примерно равны.

3. *Реализация итерационного метода решения СЛАУ (метод Зейделя, Якоби или верхней релаксации на выбор). Метод Зейделя.*

Метод Зейделя представляет собой некоторую модификацию метода итераций. Основная его идея заключается в том, что при вычислении (k + 1)-го приближения неизвестной x_i учитываются уже вычисленные ранее (k + 1)-е приближения неизвестных $x_1, x_2, \ldots, x_{i-1}$.

Пусть получена эквивалентная система (4.2). Выберем произвольно начальные приближения корней $x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}$. Далее, предполагая, чтоk-ые приближения $x_n^{(k)}$ корней известны, согласно Зейделю будем строить (k + 1)-е приближения корней по формулам:

$$\begin{aligned} x_1^{(k+1)} &= \beta_1 + \alpha_{12}x_2^{(k)} + \alpha_{13}x_3^{(k)} + \ldots + \alpha_{1n}x_n^{(k)}, \\ x_2^{(k+1)} &= \beta_2 + \alpha_{21}x_1^{(k+1)} + \alpha_{23}x_2^{(k)} + \ldots + \alpha_{2n}x_n^{(k)}, \\ &\ldots\ldots\ldots\ldots \\ x_n^{(k+1)} &= \beta_n + \alpha_{n1}x_1^{(k+1)} + \alpha_{n2}x_2^{(k+1)} + \ldots + \alpha_{nn}x_n^{(k)} \quad (k = 0, 1, 2, \ldots). \end{aligned} \tag{4.5}$$

Заметим, что указанные выше условия сходимости для простой итерации остается верной для итерации по методу Зейделя. Обычно метод Зейделя дает лучшую сходимость, чем метод простой итерации, но приводит к более громоздким вычислениям.

Вот код, реализующий данный алгоритм:

```python
def seidel(A, b, x0, tol=1e-6, max_iter=100):
    n = len(A)
    x = x0.copy()
    iterations = 0
    residual = np.linalg.norm(A @ x - b)

    while residual > tol and iterations < max_iter:
        for i in range(n):
            x[i] = (b[i] - A[i, :i] @ x[:i] - A[i, i + 1:] @ x[i + 1:]) / A[i, i]

        iterations += 1
        residual = np.linalg.norm(A @ x - b)

    return x, iterations, residual
```

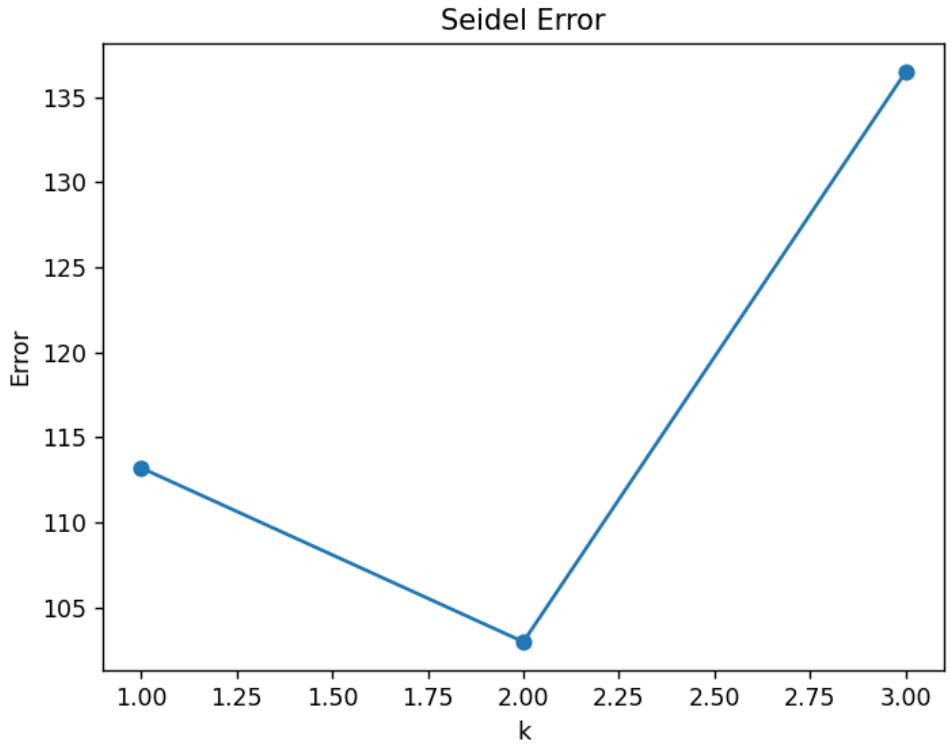
Проведем исследование на системах с матрицами A^k :
Берем значения для k[1..3], причем, генерируем матрицы такого вида, как и в предыдущих методах, но получаем уже такие результаты:

```
==== k = 1 ====
Matrix A^(k):
[[ 8.1 -2.  -3.  -1.  -2. ]
 [-3.   9.  -1.  -1.  -4. ]
 [-4.  -3.  11.  -2.  -2. ]
 [-2.  -3.  -4.  13.  -4. ]
 [-1.  -1.  -4.  -3.   9. ]]
Condition number: 680.7116804044987
Gauss solution: [123.30062444 124.60856378 124.52667261 124.82729706 125.05530776]
Gauss error: 276.07657906144755
LU solution: [123.30062444 124.60856378 124.52667261 124.82729706 125.05530776]
LU error: 276.07657906144755
Seidel solution: [50.88359879 51.60891685 51.62406352 51.86301952 52.17531425]
Seidel error: 113.21833439663148
```

```
==== k = 2 ====
Matrix A^(k):
[[11.01 -3.   -4.   -2.   -2.  ]
 [-2.    8.   -2.   -3.   -1.  ]
 [-3.   -3.   13.   -3.   -4.  ]
 [-3.   -4.   -2.   13.   -4.  ]
 [-3.   -3.   -4.   -4.   14.  ]]
Condition number: 9723.445459873146
Gauss solution: [1453.00896287 1454.1975323  1454.2104528  1454.27214527 1454.32499127]
Gauss error: 3249.013261703691
LU solution: [1453.00896287 1454.1975323  1454.2104528  1454.27214527 1454.32499127]
LU error: 3249.01326170381
Seidel solution: [46.7004893  46.96494255 47.06246963 47.2059473  47.36214023]
Seidel error: 102.99271521359695
```

```
==== k = 3 ====
Matrix A^(k):
[[10.001 -2.    -4.    -2.    -2.   ]
 [-1.     5.    -2.    -1.    -1.   ]
 [-2.    -1.     8.    -1.    -4.   ]
 [-2.    -3.    -2.    10.    -3.   ]
 [-4.    -3.    -3.    -4.    14.   ]]
Condition number: 106488.05671482251
Gauss solution: [16871.20928494 16872.86521904 16872.75851743 16872.8715837
 16872.72819171]
Gauss error: 37725.79085545757
LU solution: [16871.20928496 16872.86521907 16872.75851746 16872.87158373
 16872.72819174]
LU error: 37725.790855516585
Seidel solution: [61.60938533 61.96801897 62.07660706 62.23826348 62.32317667]
Seidel error: 136.49764015588275
```

И такой график:



Обычно метод Зейделя дает лучшую сходимость, чем метод простой итерации, но приводит к более громоздким вычислениям.

Получаем в итоге такой результат в сравнении всех трех методов:

```
D:\PriMat_lab4\venv\Scripts\python.exe D:\PriMat_lab4\task4.py
Solution:
[1 2 3]
Number of iterations: 2
Residual: 0.0
==== k = 1 ====
Matrix A^(k):
[[11.1 -3.  -4.  -1.  -3. ]
 [-4.   8.  -1.  -1.  -2. ]
 [-3.  -1.  10.  -2.  -4. ]
 [-1.  -1.  -4.   9.  -3. ]
 [-2.  -1.  -4.  -3.  10. ]]
Condition number: 845.9033091829343
Gauss solution: [170.77348066 171.86187845 172.2320442  172.56353591 172.50276243]
Gauss error: 382.34078200134735
LU solution: [170.77348066 171.86187845 172.2320442  172.56353591 172.50276243]
LU error: 382.34078200134735
Seidel solution: [54.70728815 55.26395823 55.48870698 55.8083148  55.90583069]
Seidel error: 121.7237477534876
Number of iterations: 100
Residual: 6.158360994683913
```

```
==== k = 2 ====
Matrix A^(k):
[[ 4.01 -1.   -1.   -1.   -1.  ]
 [-2.   10.   -1.   -4.   -3.  ]
 [-2.   -2.   11.   -3.   -4.  ]
 [-3.   -3.   -4.   14.   -4.  ]
 [-3.   -2.   -4.   -4.   13.  ]]
Condition number: 5080.537566400623
Gauss solution: [663.01517143 664.35599092 664.44451081 664.4140485  664.47628718]
Gauss error: 1482.829342916583
LU solution: [663.01517143 664.35599092 664.44451081 664.4140485  664.47628718]
LU error: 1482.829342916474
Seidel solution: [54.59253958 54.78751051 54.93141142 55.05498366 55.25370924]
Seidel error: 120.57885365504613
Number of iterations: 100
Residual: 6.35544493374146
```

```
==== k = 3 ====
Matrix A^(k):
[[15.001 -3.    -4.    -4.    -4.   ]
 [-4.    15.    -3.    -4.    -4.   ]
 [-2.    -2.    11.    -4.    -3.   ]
 [-4.    -2.    -3.    13.    -4.   ]
 [-3.    -2.    -3.    -4.    12.   ]]
Condition number: 110002.86673275416
Gauss solution: [18540.04252296 18541.06047713 18541.25939043 18541.17812419
 18541.31159927]
Gauss error: 41456.63417961985
LU solution: [18540.04252305 18541.06047722 18541.25939052 18541.17812428
 18541.31159936]
LU error: 41456.6341798192
Seidel solution: [48.74974573 48.89826037 49.09326162 49.21849745 49.43329438]
Seidel error: 107.5083758492769
Number of iterations: 100
Residual: 9.974444569133187
```

Из предоставленных данных видно, что при увеличении числа обусловленности матриц A^(k) увеличивается ошибка решения для всех трех методов: метода Гаусса, LU-разложения и метода Зейделя. Это связано с тем, что высокая обусловленность матрицы указывает на то, что даже небольшие изменения в правой части системы могут привести к значительным изменениям в решении. В результате получаем большие ошибки решения.

Кроме того, можно заметить, что метод Зейделя сходится медленнее, чем метод Гаусса и LU-разложение. Это видно по количеству итераций, которое для метода Зейделя составляет 100 на каждом шаге k. Это может быть связано с особенностями выбранного начального приближения и условиями сходимости метода Зейделя.

Также следует отметить, что остатки (residuals) для всех трех методов остаются относительно стабильными на каждом шаге k.

В целом, можно сделать вывод, что при увеличении числа обусловленности матрицы решение системы линейных уравнений становится менее точным, и ошибка решения увеличивается для всех методов.

4. *Оценка зависимости числа обусловленности и точности полученного решения в зависимости от параметра k:*

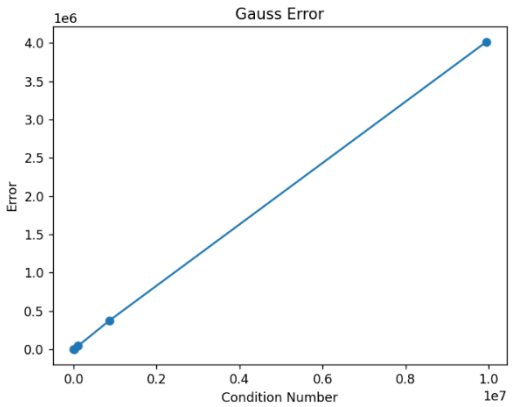
Для этого задания (task5.py) мы используем весь код из task4 для генерации матриц, но уже с другим шагом для k[1..5] и получаем такие результаты:

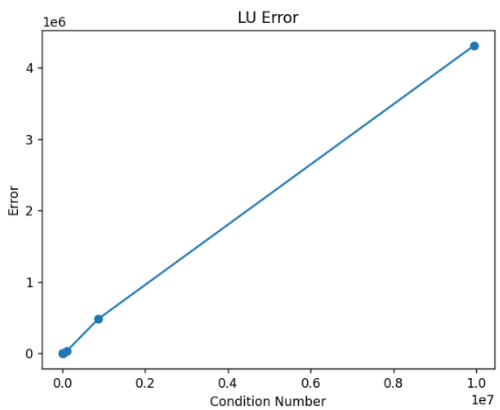
```
Solution:
[1 2 3]
Number of iterations: 2
Residual: 0.0
==== k = 1 ====
Matrix A^(k):
[[ 5.1 -2.  -1.  -1.  -1. ]
 [-2.   6.  -2.  -1.  -1. ]
 [-4.  -4.  15.  -4.  -3. ]
 [-2.  -4.  -1.  10.  -3. ]
 [-1.  -2.  -2.  -1.   6. ]]
Condition number: 663.765899491648
Gauss error: 210.01166115825256
LU error: 210.01166115824773
Seidel error: 111.30912931481849

==== k = 2 ====
Matrix A^(k):
[[ 9.01 -2.   -1.   -3.   -3.  ]
 [-4.   14.   -3.   -4.   -3.  ]
 [-2.   -4.   13.   -4.   -3.  ]
 [-1.   -2.   -4.    8.   -1.  ]
 [-2.   -1.   -3.   -4.   10.  ]]
Condition number: 10256.35486597093
Gauss error: 4014.174526702706
LU error: 4014.1745267028987
Seidel error: 149.83274347047742
```

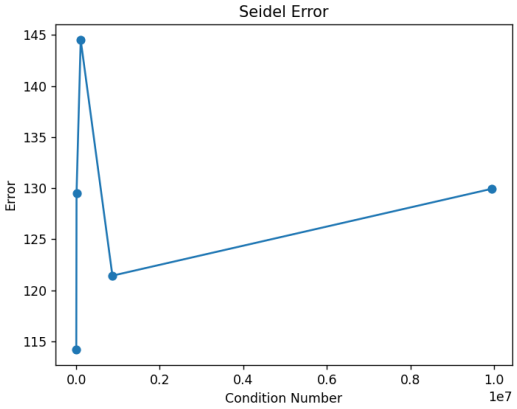
```
==== k = 3 ====
Matrix A^(k):
[[13.001 -4.    -3.    -4.    -2.   ]
 [-3.     9.    -3.    -1.    -2.   ]
 [-1.    -4.    10.    -1.    -4.   ]
 [-2.    -1.    -3.     8.    -2.   ]
 [-4.    -4.    -4.    -4.    16.   ]]
Condition number: 127381.01861419619
Gauss error: 42667.31335523672
LU error: 42667.313355252074
Seidel error: 123.4617061705256

==== k = 4 ====
Matrix A^(k):
[[11.0001 -4.     -2.     -4.     -1.    ]
 [-4.     10.     -1.     -3.     -2.    ]
 [-4.     -4.     15.     -3.     -4.    ]
 [-4.     -2.     -4.     14.     -4.    ]
 [-3.     -1.     -1.     -3.      8.    ]]
Condition number: 747513.7089762808
Gauss error: 263178.7448058398
LU error: 263178.7448058398
Seidel error: 115.90143198882215
```

```
==== k = 5 ====
Matrix A^(k):
[[11.00001 -1.      -4.      -3.      -3.     ]
 [-4.      12.      -4.      -2.      -2.     ]
 [-3.      -3.      11.      -4.      -1.     ]
 [-2.      -1.      -2.       8.      -3.     ]
 [-4.      -4.      -4.      -4.      16.     ]]
Condition number: 9259950.72820622
Gauss error: 3133982.5290741627
LU error: 3133982.529777002
Seidel error: 118.86603620621405

Condition numbers: [663.765899491648, 10256.35486597093, 127381.01861419619, 747513.7089762808, 9259950.72820622]
Gauss errors: [210.01166115825256, 4014.174526702706, 42667.31335523672, 263178.7448058398, 3133982.5290741627]
LU errors: [210.01166115824773, 4014.1745267028987, 42667.313355252074, 263178.7448058398, 3133982.529777002]
Seidel errors: [111.30912931481849, 149.83274347047742, 123.4617061705256, 115.90143198882215, 118.86603620621405]
```

И такие графики для каждого из наших методов:







Из предоставленных данных видно, что с увеличением параметра k, который отвечает за диагональное преобладание матрицы, числа обусловленности матриц A^(k) также увеличиваются. Это означает, что матрицы становятся более плохо обусловленными, и решение системы линейных уравнений становится более чувствительным к погрешностям в данных или округлении.

Как следствие, ошибка решения системы уравнений с помощью метода Гаусса и LU-разложения также увеличивается с увеличением числа обусловленности. Это отражено в значениях ошибок, которые растут с каждым шагом k.

Однако метод Зейделя демонстрирует более стабильную ошибку в решении, независимо от значения k. Это может быть связано с итерационным характером метода Зейделя, который позволяет достигать определенной точности независимо от числа обусловленности.

Таким образом, можно сделать вывод, что чем выше числа обусловленности матрицы, тем более неустойчивыми и неточными становятся решения системы линейных уравнений при использовании прямых методов, таких как метод Гаусса и LU-разложение. В то же время метод Зейделя остается относительно стабильным при различных значениях числа обусловленности.

5. *Провести аналогические исследования на матрицах Гильберта, которые строятся согласно формуле:*

$$a_{ij} = \frac{1}{i+j-1}, \qquad i,j = 1,..,n$$

Где n – размерность матрицы

- Матрица Гильберта является симметричной положительно определённой матрицей. Более того, матрица Гильберта является вполне положительной матрицей.
- Матрица Гильберта является примером ганкелевой матрицы.
- Определитель матриц Гильберта может быть выражен явно, как частный случай определителя Коши. Определитель матрицы Гильберта $n \times n$ равен

$$\det(H) = \frac{c_n^4}{c_{2n}},$$

где

$$c_n = \prod_{i=1}^{n-1} i^{n-i} = \prod_{i=1}^{n-1} i!.$$

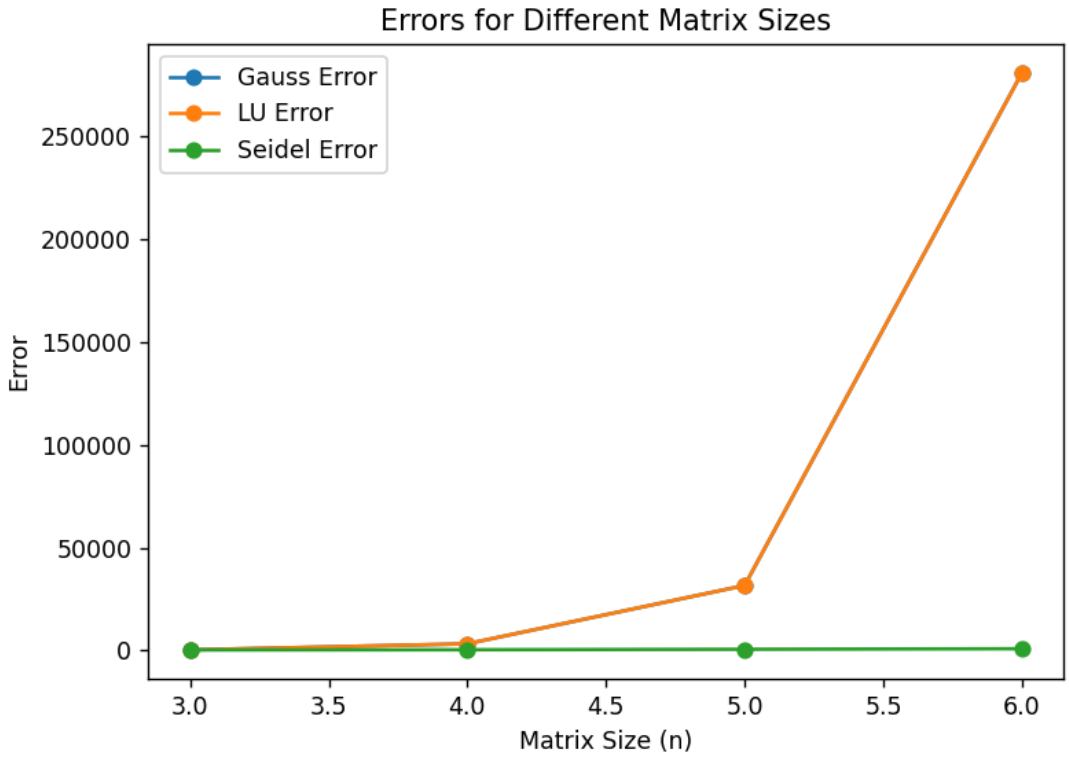
Для этого задания выберем n[3..6] и получим уже такие результаты:

```
Solution:
[1 2 3]
Number of iterations: 2
Residual: 0.0
==== n = 3 ====
Matrix Hilbert(n):
[[1.         0.5        0.33333333]
 [0.5        0.33333333 0.25      ]
 [0.33333333 0.25       0.2       ]]
Condition number: 524.0567775860644
Gauss error: 285.6676390492988
LU error: 285.6676390492988
Seidel error: 242.93521497633745

==== n = 4 ====
Matrix Hilbert(n):
[[1.         0.5        0.33333333 0.25      ]
 [0.5        0.33333333 0.25       0.2       ]
 [0.33333333 0.25       0.2        0.16666667]
 [0.25       0.2        0.16666667 0.14285714]]
Condition number: 15513.73873892924
Gauss error: 3236.7619622087523
LU error: 3236.7619622077077
Seidel error: 347.27992886821045
```

```
==== n = 5 ====
Matrix Hilbert(n):
[[1.         0.5        0.33333333 0.25       0.2       ]
 [0.5        0.33333333 0.25       0.2        0.16666667]
 [0.33333333 0.25       0.2        0.16666667 0.14285714]
 [0.25       0.2        0.16666667 0.14285714 0.125     ]
 [0.2        0.16666667 0.14285714 0.125      0.11111111]]
Condition number: 476607.2502422687
Gauss error: 31629.606067946934
LU error: 31629.60606768418
Seidel error: 571.6749603373157

==== n = 6 ====
Matrix Hilbert(n):
[[1.         0.5        0.33333333 0.25       0.2        0.16666667]
 [0.5        0.33333333 0.25       0.2        0.16666667 0.14285714]
 [0.33333333 0.25       0.2        0.16666667 0.14285714 0.125     ]
 [0.25       0.2        0.16666667 0.14285714 0.125      0.11111111]
 [0.2        0.16666667 0.14285714 0.125      0.11111111 0.1       ]
 [0.16666667 0.14285714 0.125      0.11111111 0.1        0.09090909]]
Condition number: 14951058.6424659
Gauss error: 281080.01222710055
LU error: 281080.0122302293
Seidel error: 845.977710173911
```

```
Condition numbers: [524.0567775860644, 15513.73873892924, 476607.2502422687, 14951058.6424659]
Gauss errors: [285.6676390492988, 3236.7619622087523, 31629.606067946934, 281080.01222710055]
LU errors: [285.6676390492988, 3236.7619622077077, 31629.60606768418, 281080.0122302293]
Seidel errors: [242.93521497633745, 347.27992886821045, 571.6749603373157, 845.977710173911]
```

И получим такой график сравнения каждого метода:



Можно заметить, что методы Гаусса и LU – разложения получили идентичные результаты.

Из представленных данных видно, что числа обусловленности матриц Гильберта быстро растут с увеличением размерности матрицы n. Это указывает на то, что матрицы Гильберта являются плохо обусловленными и приводят к большим ошибкам при решении систем линейных уравнений.

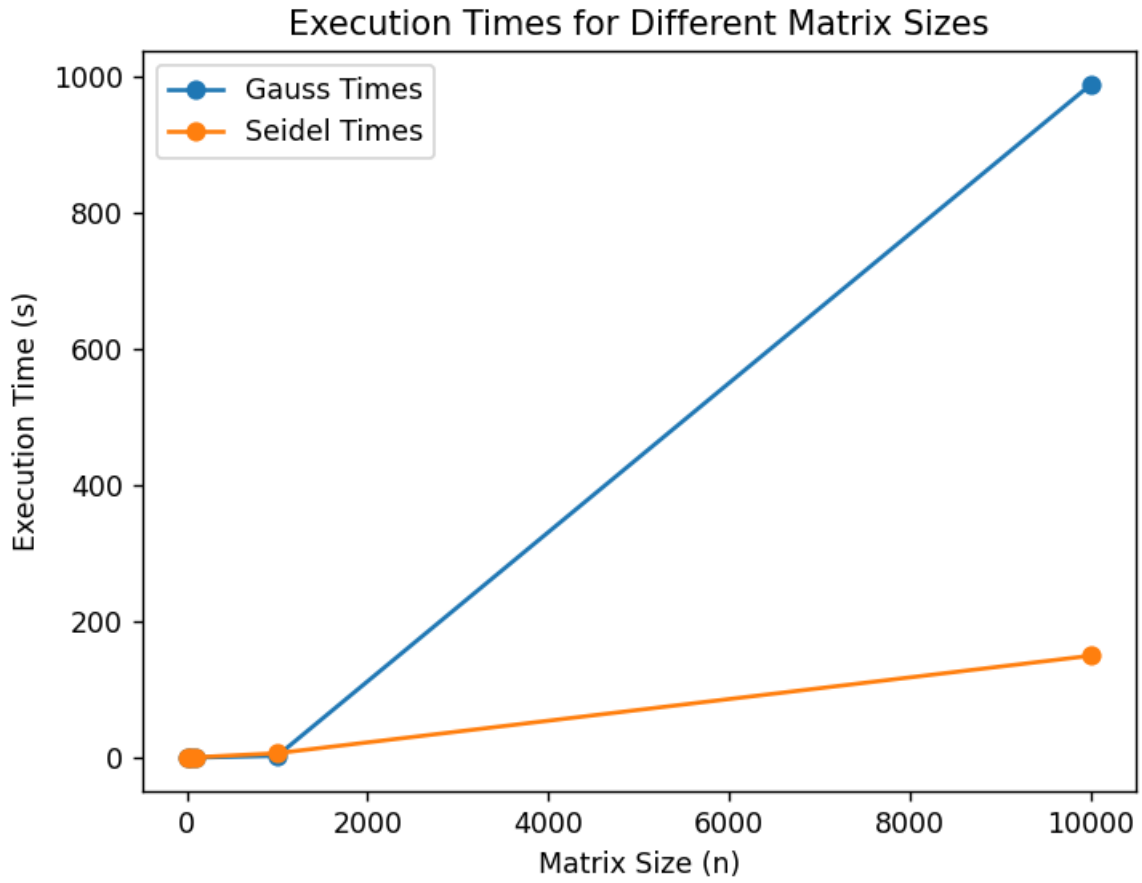
Метод Гаусса и LU-разложение показывают сходные значения ошибок, которые также возрастают с увеличением размерности матрицы. Это говорит о том, что оба метода чувствительны к высокому числу обусловленности и дают сопоставимую точность решения системы уравнений.
Метод Зейделя также демонстрирует увеличение ошибки с увеличением размерности матрицы, но ошибка остается сопоставимой с ошибками методов Гаусса и LU-разложения. Это может быть связано с тем, что метод Зейделя имеет итерационный характер и может достичь определенной точности, несмотря на высокое число обусловленности матрицы.
Таким образом, матрицы Гильберта являются плохо обусловленными, что ведет к большим ошибкам при решении систем линейных уравнений. Метод Зейделя остается относительно стабильным при различных размерностях матрицы, но также не избавлен от ошибок.

6. *Сравнение между собой прямые и итерационные методы по эффективности методов в зависимости от размеров n матрицы: n in {10, 50, 100, 10^3, 10^4, 10^5}*

Таким же образом строим матрицы Гилберта, что и для предыдущего задания, но уже другой размерности. Из-за увеличения размерности матрицы, время программы, затраченное на решения матриц, достигает ужасающих значений, мы не смогли дождаться для 10^5 размерности, но получаем результаты:

```
=== Прямой метод (Гаусс) ===
Гаусс times: [0.0, 0.003586292266845703, 0.015044689178466797, 2.530501365661621, 988.4858531951904]
=== Итерационный метод (Зейдель) ===
Seidel times: [0.07261061668395996, 0.24374723434448242, 0.4872090816497803, 6.448711633682251, 149.61970233917236]
```

И получаем такой график:



Из полученных результатов видно, что время выполнения прямого метода (Гаусса) растет значительно медленнее, чем время выполнения итерационного метода (Зейделя), с увеличением размерности матрицы.