Национальный исследовательский университет ИТМО
Факультет информационных технологий и программирование
Информационные системы и технологии

Методы решения СЛАУ

Отчёт по лабораторной работе № 4

Работу выполнили:
Деревицкая П. К., студентка М32081
Фирсова Д. А., студентка М32081
Борздун А. В., студентка М32051

Преподаватель:
Свинцов М. В.

Санкт-Петербург
2023

Оглавление

Постановка задачи

- ***Цель работы:***

  Знакомство с методами решения СЛАУ.

- ***Задачи, решаемые во время выполнения работы:***

  1. Реализация методов решения СЛАУ на Python 3
  2. Проведение исследования реализованных методов на системах с матрицами $A^{(k)}$, число обусловленности которых регулируется за счет изменения диагонального преобладания.
  3. Оценка зависимости числа обусловленности и точности полученного решения в зависимости от параметра k.
  4. Исследование зависимостей на матрицах Гилберта.
  5. Сравнение эффективность прямых и инерционных методов, в зависимости от размеров матрицы n.

Метод Гаусса

Теория

Метод Гаусса является наиболее известным методом решения систем линейных уравнений. Суть метода заключается в последовательном исключении неизвестных. Ниже выводится совокупность формул, позволяющих в итоге получить значения неизвестных. Метод Гаусса состоит из двух этапов - прямого хода и обратного хода.

Рассмотрим систему линейных уравнений вида:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

- Прямой ход
  Задача прямого хода заключается в приведении системы к треугольному виду. Первый шаг - исключим из второго, третьего, ..., n-го уравнений. Второй шаг - исключим из третьего, четвертого, ..., n-го уравнений и т.д. Результатом первого шага будет система:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ \downarrow \text{Отнимаем } -\frac{a_{21}}{a_{11}} \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \downarrow \text{Отнимаем } -\frac{a_{31}}{a_{11}} \\ a_{31}x_1 + a_{32}x_2 + \ldots + a_{3n}x_n = b_3 \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \\ \downarrow \text{Отнимаем } -\frac{a_{n1}}{a_{11}} \end{cases}$$

  Коэффициенты с верхним индексом 1 (верхний индекс показывает номер шага) подсчитываются по формулам:

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}}a_{1j}, b_i^{(1)} = b_i - \frac{a_{i1}}{a_{11}}b_1, i, j = \overline{2,n}$$

Результатом второго шага будет система:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2 \\ \downarrow \text{Отнимаем} - \frac{a_{32}}{a_{22}} \\ a_{33}x_3 \ldots + a_{3n}x_n = b_3 \\ \cdots \\ a_{n3}x_3 + \ldots + a_{nn}x_n = b_n \\ \downarrow \text{Отнимаем} - \frac{a_{n2}}{a_{22}} \end{cases}$$

Заметим, что коэффициенты для второго шага вычисляются по формулам:

$$a_{ij}^{(2)} = a_{ij} - \frac{a_{i1}}{a_{11}}a_{1j}, b_i^{(2)} = b_i - \frac{a_{i1}}{a_{11}}b_1, i, j = \overline{3,n}$$

Продолжая процесс на (n-1) шаге система приобретает треугольный вид:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ a_{nn}x_n = b_n \end{cases}$$

Коэффициенты этой системы могут быть получены с помощью формул общего вида:

$$a_{ij}^{(k)} = a_{ik}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}a_{kj}^{(k-1)}, b_i^{(k)} = b_i^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}b_k^{(k-1)}$$

$$k = 1...n-1,$$
$$i, j = k-1,n$$

- Обратный ход
  Заключается в вычислении неизвестных, начиная с последнего и двигаясь вверх. Для нахождения вычислим простую дробь:

$$a_{nn}^{(n-1)}x_n = b_n^{(n-1)} \Rightarrow x_n = \frac{b_n^{(n-1)}}{a_{nn}(n-1)}$$

Для нахождения:

$$a_{n-1,n-1}^{(n-2)}x_{n-1} + a_{n-1,n}^{(n-2)}x_n = bn - 1^{(n-2)}\$ \Rightarrow \$x_{n-1} = \frac{b_{n-1}^{(n-2)} - a_{n-1,n}^{(n-2)}x_n}{a_{n-1,n-1}^{(n-2)}}$$

Для первых неизвестных формулы принимают следующий вид:

$$x_2 = \frac{b_2^{(1)} - a_{23}^{(1)} x_3 - \ldots - a_{2n}^{(1)} x_n}{a_{22}^{(1)}}$$

$$x_1 = \frac{b_1 - a_{12} x_2 - \ldots - a_m x_n}{a_{11}}$$

Исходя из вышеприведенных уравнений возможно сформулировать общую формулу обратного хода:

$$x_k = \frac{1}{a_{kk}^{(k-1)}}(b_k^{(k-1)} - \sum_{j=k+1}^{n})a_{kj}^{(k-1)} x_i$$

- Постолбцовый выбор главных элементов
  Так как реальные машинные вычисления производятся с усеченными числами, неизбежны ошибки округления. Если на каком-то этапе знаменатели дробей окажутся слишком маленькими, выполнение алгоритма может привести к неверным результатам. Рассмотрим пример:

$$\begin{cases} 10^{-4} x_1 + x_2 = 1 \\ x_1 - 2x_2 = 4 \end{cases}$$

Вычтем из второй строчки первую, умноженную на 10^{-4}:

$$\begin{cases} 10^{-4} x_1 + x_2 = 1 \\ 0 - 9998 x_2 = 4 - 10^{-4} \end{cases}$$

$x_2 = \frac{9996}{9998} = 0.999 = 1$

Подставляя в первую строчку $10^{-4} \cdot x_1 + 1 = 1, x_1 = 0$

Подставляя во вторую строчку найденные x_1 и x_2 имеем 2=4. Решим ту же систему, поменяв местами строчки в исходной системе:

$$\begin{cases} x_1 + 2x_2 = 4 \\ 10^{-4} \cdot x_1 + x_2 = 1 \end{cases}$$

$$\begin{cases} x_1 + 2x_2 = 4 \\ x_2(1 - 2 \cdot 10^{-4}) = 1 - 10^{-4} \end{cases}$$

$$x_2 = 1$$
$$x_1 = 2$$

Подставляя полученные значения в систему, получаем результат, более приближенный к реальности, чем результат из предыдущего решения. Чтобы уменьшить влияние ошибок округления и исключить деление на нуль на каждом этапе прямого хода, уравнения системы решено переставлять так, чтобы деление проводилось на наибольший по модулю в данном столбце элемент (что было сделано во втором варианте решения). Это делается с целью избежания ситуаций, когда $a_{kk}^{(k)}$ близко к нулю. Если это так, то результатом деления на $a_{kk}^{(k)}$ может оказаться число с очень большой погрешностью.

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.

Метод Гаусса с выбором главного члена по столбцу является модификацией классического метода Гаусса, который используется для решения систем линейных уравнений. *Основная особенность* этого метода заключается в том, что на каждом шаге алгоритма выбирается главный элемент из текущего столбца матрицы, что позволяет избежать деления на ноль и уменьшить погрешность вычислений.

Эффективность метода Гаусса с выбором главного члена по столбцу зависит от характеристик системы уравнений, таких как ее размерность, плотность и спецификация матрицы. В общем случае этот метод может быть более эффективным, чем классический метод Гаусса, особенно если матрица системы имеет большое число нулевых элементов или если ее размерность очень большая.

Сходимость метода Гаусса с выбором главного члена по столбцу гарантирована при условии, что матрица системы является невырожденной и не имеет нулевых диагональных элементов. В противном случае сходимость может быть нарушена, что может привести к ошибкам в решении системы.

Точность метода Гаусса с выбором главного члена по столбцу зависит от погрешности входных данных и от числа итераций, необходимых для получения решения. В общем случае этот метод может обеспечивать высокую точность решения системы линейных уравнений, особенно если используется точный

арифметический формат. Однако при использовании численных методов могут возникать ошибки округления, которые могут снизить точность результата.

В целом, метод Гаусса с выбором главного члена по столбцу является эффективным и точным методом для решения систем линейных уравнений, который может быть использован в различных приложениях, таких как научные вычисления, инженерные расчеты и финансовые моделирования.

Практика

I. Реализация метода Гаусса

```python
import numpy as np

def test_solve_gauss():
    # Test 1
    m = np.array([[2, 1, -1, 8],
                  [-3, -1, 2, -11],
                  [-2, 1, 2, -3]], dtype=float)
    solve_gauss(m)
    # Expected output:
    # x1 = 2.0
    # x2 = 3.0
    # x3 = -1.0

    # Test 2
    m = np.array([[1, 2, -1, 8],
                  [2, -1, 1, 3],
                  [3, -2, 1, 1]], dtype=float)
    solve_gauss(m)
    # Expected output:
    # x1 = 2.5
    # x2 = 4.25
    # x3 = 2.75

    # Test 3 (singular matrix)
    m = np.array([[1, 2, -1, 8],
                  [2, -1, 1, 3],
                  [4, -2, 2, 6]], dtype=float)
    solve_gauss(m)
    # Expected output:
    # The system has infinite number of answers...

test_solve_gauss()
```

В данной части кода была проведена проверка на работоспособность реализованного метода.

Как мы можем заметить, код прошел проверку на решение СЛАУ, имеющие целые значения, дробные и бесконечные множества в ответе.

```
Solution:
x1 = 2.0
x2 = 3.0000000000000004
x3 = -0.9999999999999999
Solution:
x1 = 2.2500000000000004
x2 = 4.250000000000001
x3 = 2.750000000000001
The system has infinite number of answers...
```

II. Исследование метода на системах с матрицами$A^{(k)}$, число обусловленности которых регулируется за счет изменения диагонального преобладания.

Был создан генератор плотных матриц. С помощью него и реализованной ранее функции мы получили данную таблицу, отображающую корректную работу метода для различных k и n. Параметр k варьируется в пределах от 1 до 4, размеренности матрицы n: 10, 30, 100, 200. Для каждого из этих значений были посчитаны числа обусловленности, относительные и абсолютные ошибки.

k	n	Condition number	Relative error	Absolute error
1	10	49.8092	2.14192e-16	4.20275e-15
1	50	479.579	5.07648e-16	1.05176e-13
1	100	1248.83	7.32282e-16	4.25953e-13
1	200	3293.64	1.02485e-15	1.67984e-12
2	10	49.8416	1.17604e-16	2.30756e-15
2	50	479.579	4.3132e-16	8.93624e-14
2	100	1248.83	6.52664e-16	3.79641e-13
2	200	3293.64	7.88393e-16	1.29227e-12
3	10	49.845	2.73708e-16	5.37054e-15
3	50	479.579	4.41538e-16	9.14794e-14
3	100	1248.83	6.46297e-16	3.75937e-13
3	200	3293.64	9.21379e-16	1.51025e-12
4	10	49.8453	2.17676e-16	4.27111e-15
4	50	479.579	5.91114e-16	1.22469e-13
4	100	1248.83	6.18958e-16	3.60035e-13
4	200	3293.64	9.2112e-16	1.50982e-12

Мы увидели, что метод работает корректно, так как при росте числа обусловленности значение ошибки возрастает.

III. Оценка зависимости числа обусловленности и точности полученного решения в зависимости от параметра k.

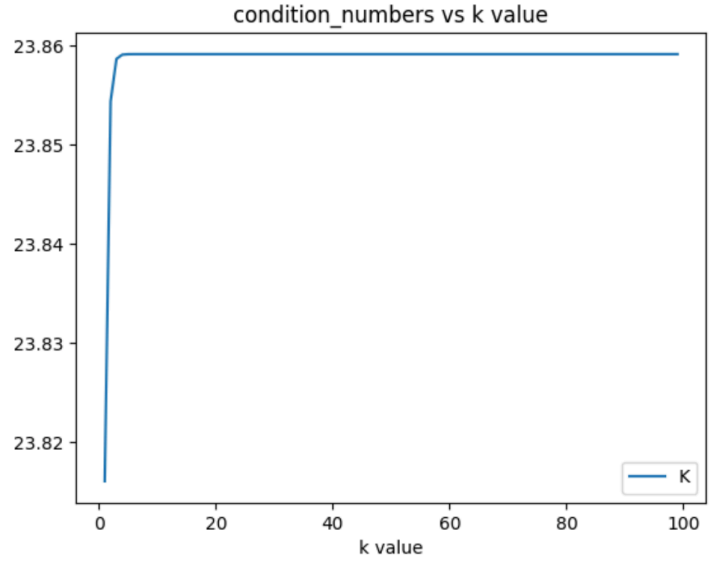
Число обусловленности матрицы - это мера ее "чувствительности" к небольшим изменениям входных данных или погрешностям при вычислениях. Чем больше число обусловленности, тем более чувствительной становится матрица к ошибкам и неточностям, что может привести к значительному увеличению ошибки при решении системы линейных уравнений.

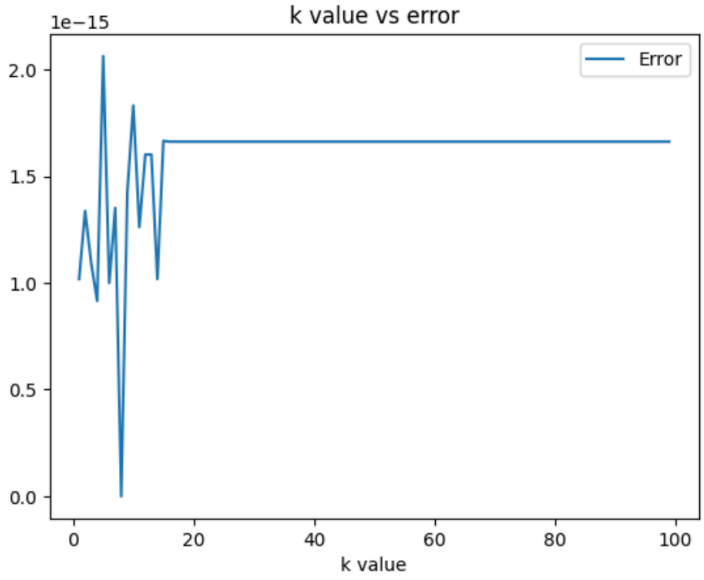
Метод Гаусса - это один из наиболее распространенных методов решения систем линейных уравнений. Он заключается в последовательном приведении матрицы системы к треугольному виду путем элементарных преобразований строк. Однако, при этом происходят операции с элементами матрицы, которые могут привести к увеличению ошибки при большом числе обусловленности.

При уменьшении параметра k матрица системы становится более близкой к сингулярности, что приводит к увеличению числа обусловленности. Это означает, что матрица становится более чувствительной к ошибкам и неточностям при вычислениях. При решении системы методом Гаусса происходят элементарные преобразования строк, которые могут привести к увеличению ошибки при большом числе обусловленности. Поэтому, при уменьшении параметра k точность полученного решения может ухудшаться.

Однако, при достаточно больших значениях k, матрица системы становится более хорошо обусловленной, что может привести к улучшению точности полученного решения. Это связано с тем, что при большом числе обусловленности матрица становится менее чувствительной к ошибкам и неточностям при вычислениях, что может уменьшить ошибку при решении системы методом Гаусса.

Таким образом, параметр k может влиять на точность полученного решения при использовании метода Гаусса за счет изменения числа обусловленности матрицы системы. При уменьшении k точность может ухудшаться из-за более высокой чувствительности матрицы к ошибкам и неточностям, а при увеличении k точность может улучшаться из-за более низкой чувствительности матрицы к ошибкам и неточностям.





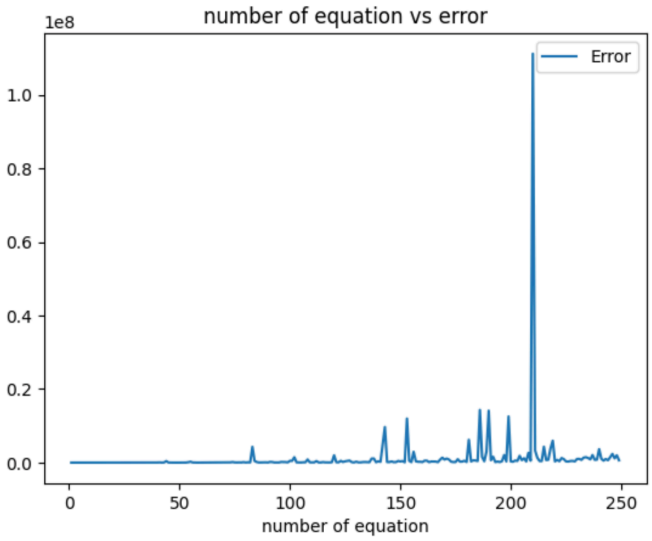
Вывод: Чем больше параметр k, тем больше число обусловленности. Это в свою очередь может привести к увеличению ошибки при решении системы методом Гаусса. Следовательно, при увеличении параметра k точность полученного решения может ухудшаться. Однако, при достаточно маленьких значениях k, матрица системы становится более хорошо обусловленной, что может привести к улучшению точности полученного решения.

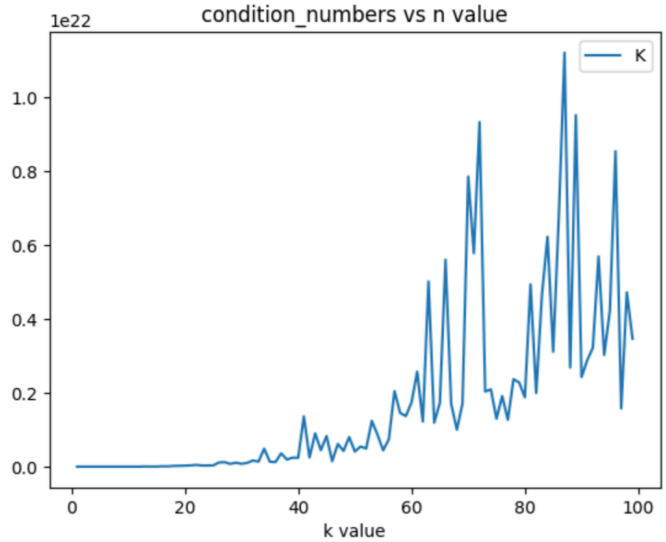
IV. Исследование зависимостей на матрицах Гилберта.

Был создан генератор разреженных матриц Гилберта. С помощью него и реализованной ранее функции мы получили данную таблицу, отображающую корректную работу метода для различных n. Размеренность матрицы n варьируется в пределах от 1 до 9. Были посчитаны числа обусловленности, относительные и абсолютные ошибки.

n	Condition number	Relative error	Absolute error
1	1	0	0
2	34.1275	6.66134e-16	1.48952e-15
3	1490.66	4.11874e-15	1.54109e-14
4	59290.6	6.85199e-13	3.75299e-12
5	2.30617e+06	3.32169e-12	2.46343e-11
6	8.85696e+07	7.55762e-11	7.20951e-10
7	3.36301e+09	3.30224e-09	3.90726e-08
8	1.26415e+11	2.88395e-07	4.11911e-06
9	4.71072e+12	2.31616e-07	3.91012e-06

Мы увидели, что метод работает корректно, так как при росте числа обусловленности значение ошибки возрастает.





На полученных графиках мы можем наблюдать пульсации. Это объясняется тем, что матрицы Гилберта являются неравномерно плохо обусловленными. Следовательно, метод выдает наибольшую ошибку в соответствии с возрастающим числом обусловленности матриц.

LU разложение

Теория

LU разложение - это метод факторизации матрицы A в произведение двух треугольных матриц L и U, где L - нижняя треугольная матрица с единичной диагональю, а U - верхняя треугольная матрица.
То есть:

$$A = LU$$

Пусть A - данная матрица, а L и U - соотвественно нижняя (левая) и верхняя (правая) треугольные матрицы. Тогда справедлива теорема:

Если все главные миноры квадратной матрицы отличны от нуля, то существуют такие нижняя и верхняя треугольные матрицы, что произведение этих матриц равно исходной матрице, а также эти матрицы являются единичными матрицами с единицами на диагонали, за исключением возможных нулей на диагонали верхней матрицы. Такое разложение называется LU-разложением матрицы. Если элементы диагонали одной из матриц или фиксированы (ненулевые), то такое разложение единственно.

Ниже рассматриваются формулы для фактического разложения матриц в случае фиксирования диагонали нижней треугольной матрицы . Найдем (при $i > j$) и (при $i <= j$) такие, что:

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

После перемножения матриц получим следующую систему:

$$\begin{bmatrix} u_{11} = a_{11} & u_{12} = a_{12} & \cdots & u_{1n} = a_{1n} \\ l_{21}u_{11} = a_{21} & l_{21}u_{12} + u_{22} = a_{22} & \cdots & l_{21}u_{12} + u_{22} = a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1}u_{11} = a_{n1} & l_{n1}u_{12} + l_{n2}u_{22} = a_{n2} & \cdots & l_{n1}u_{1n} + u_{nn} = a_{nn} \end{bmatrix}$$

Благодаря этой системе, мы находим все неизвестные по формулам:

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, \quad (i \le j)$$

$$l_{ij} = \frac{1}{u_{ij}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right), \quad (i \ge j)$$

Препятствием в решении описанного процесса LU-раложения может оказаться равенство нулю диагональных элементов матрицы , поскольку на них будет выполняться деление. Отсюда требование теоремы, приведённой выше. Вместо проверки на равенство нулю главных миноров матрицы удобнее сделать проверку для элементов в процессе их вычисления.

Особенностью LU разложения является то, что оно может быть использовано для решения систем линейных уравнений с различными правыми частями, не требуя повторного разложения матрицы. Кроме того, при наличии определенных структурных особенностей матрицы, LU разложение может быть выполнено с меньшим количеством операций, чем метод Гаусса.

Метод LU разложения гарантирует сходимость при выполнении следующих условий:

- Матрица системы линейных уравнений должна быть невырожденной (иметь обратную матрицу).
- В процессе разложения матрицы не должно возникать деления на ноль или бесконечности.
- В случае численных методов, при вычислениях должна быть достаточная точность, чтобы избежать ошибок округления и вычислительной неустойчивости.

Если эти условия выполняются, то метод LU разложения будет сходиться к решению системы линейных уравнений. Однако, если матрица системы линейных уравнений имеет определенные структурные особенности (например, сильную диагональное преобладание), то метод LU разложения может сходиться быстрее, чем метод Гаусса.

Точность метода LU, как и любого другого метода решения СЛАУ, зависит от точности вычислений и свойств матрицы. Если матрица плохо обусловлена или близка к вырожденной, то метод LU может давать неточные результаты. В таких случаях может потребоваться использование методов с перестановками

(перестановка строк и столбцов матрицы), например, метода LU с выбором главного элемента (PLU-разложение).

В целом, сравнивая LU с другими методами решения СЛАУ, нужно учитывать конкретную задачи и свойств матрицы. Например, метод Гаусса с выбором главного элемента может быть более точным в некоторых случаях, но требует большего количества операций. Метод Якоби или метод Зейделя могут быть более эффективными для разреженных матриц, но менее точными для плотных матриц. Метод LU является одним из наиболее универсальных и распространенных методов решения СЛАУ, который обеспечивает хорошую точность и может быть оптимизирован для конкретных задач и матриц.

Преимущества LU разложения:

- Метод позволяет быстро решать системы линейных уравнений, так как для решения системы $Ax = b$ можно использовать два этапа: сначала решить систему $Ly = b$, а затем решить систему $Ux = y$. Это особенно полезно, если матрица A не изменяется, но нужно решать множество систем с той же матрицей A и разными векторами правой части b.
- LU разложение может быть более устойчивым, чем метод Гаусса, при вычислениях на компьютере.

Недостатки LU разложения:

- В общем случае, LU разложение требует порядка $\frac{n^3}{3}$ операций умножения и деления для матрицы размера $n \times n$. Это может быть неэффективно для больших матриц.
- Если матрица A имеет нулевой элемент на диагонали, то LU разложение не может быть выполнено. Также, если матрица A имеет маленькие элементы на диагонали, то может возникнуть проблема деления на ноль при выполнении LU разложения.

В целом, LU разложение является полезным методом для решения систем линейных уравнений и может быть эффективным при правильном использовании. Однако, для больших матриц или матриц с особыми свойствами, другие методы могут быть более эффективными.

Практика

I. Реализация LU разложения

В данной части задания была проведена проверка на работоспособность реализованного метода.
Как мы можем заметить, код работает исправно, решения, полученные в ходе тестирования соответствуют ожидаемым.

```
Матрица:
2.0    1.0    -1.0
-3.0   -1.0   2.0
-2.0   1.0    2.0
Решение:
x1 = 2.0
x2 = 3.0
x3 = -1.0
Матрица:
1.0    2.0    -1.0
2.0    -1.0   1.0
3.0    -2.0   1.0
Решение:
x1 = 2.25
x2 = 4.249999999999998
x3 = 2.749999999999997
```

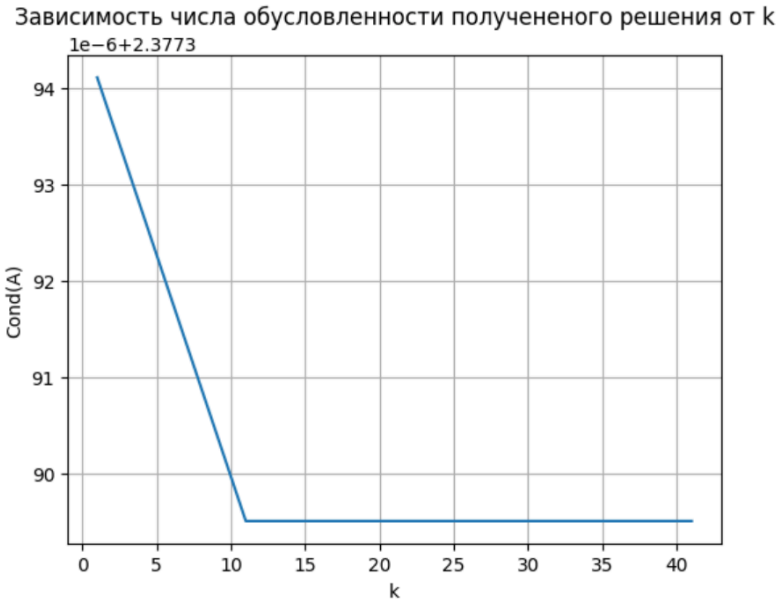
II. Исследование метода на системах с матрицами$A^{(k)}$, число обусловленности которых регулируется за счет изменения диагонального преобладания.

```
import scipy
# Создаем тестовую матрицу и вектор b
sparse_matrix = create_sparse_matrix(5, 3)
b = np.ones((5, 1))

# Разложение матрицы на L и U
lu_matrix = decompose_to_LU(sparse_matrix)
L = get_L(lu_matrix)
U = get_U(lu_matrix)

# Разложение матрицы на L и U используя встроенные функции
lu_for_matrix, piv = scipy.linalg.lu_factor(sparse_matrix.toarray())

# Точное решение системы уравнений
x_exact = scipy.linalg.lu_solve((lu_for_matrix, piv), b)

# Решение системы уравнений с помощью LU-разложения
x = solve_LU(lu_matrix, b)

# Выводим матрицу
print('Матрица:')
for row in sparse_matrix.toarray():
    row_str = [str(round(elem, 2)).ljust(6) for elem in row]
    print(' '.join(row_str))
#выводим решение
print('Решение:')
for i in range(x.shape[0]):
    print("x{} = {}".format(i+1, x[i, 0]))

# Проверка корректности решения
assert np.allclose(x_exact, x)
```

```
Матрица:
64.05  0.0    -27.0  0.0    -64.0
0.0    125.0  -8.0   -1.0   -8.0
-8.0   -64.0  1331.0 -64.0  -1.0
-64.0  -1.0   0.0    512.0  -64.0
-1.0   0.0    -1.0   -27.0  27.0
Решение:
x1 = 0.07538262150861047
x2 = 0.012078976360320126
x3 = 0.002730013775867394
x4 = 0.01873235044533847
x5 = 0.058662448048467285
```

Была создана разреженная матрица для тестирования.
Так как в код была вставлена проверка на соответствие решений, полученных в ходе тестирования библиотечной и нашей функций, мы можем сказать, что реализованный метод LU разложения отработал корректно.

III. Оценка зависимости числа обусловленности и точности полученного решения в зависимости от параметра k.

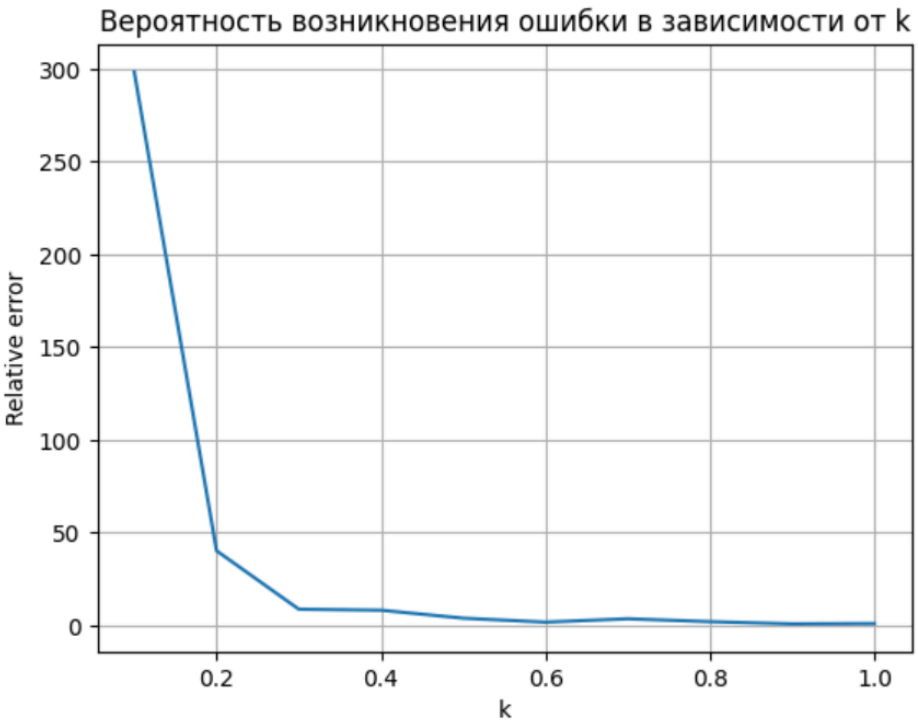


k	Cond.Number	Error
1.0	2.38e+00	7.64e−11
11.0	2.38e+00	7.64e−11
21.0	2.38e+00	7.64e−11
31.0	2.38e+00	7.64e−11
41.0	2.38e+00	7.64e−11

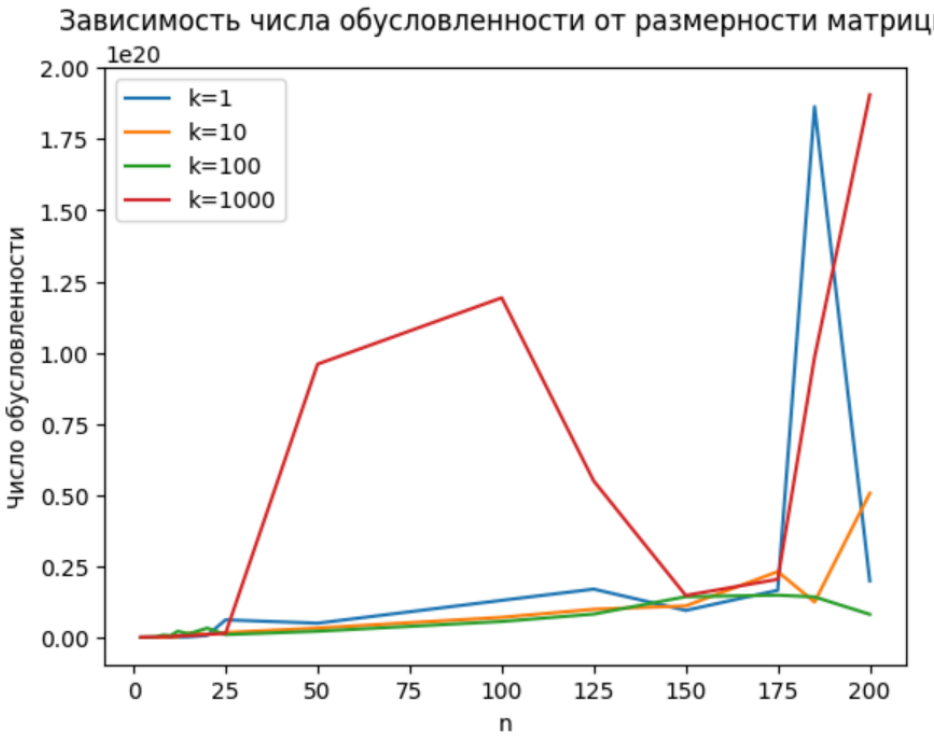
В таблице мы видим, что с увеличением k число обусловленности матрицы падает, а относительная ошибка увеличивается. Это связано с тем, что при большом k матрица становится более плохо обусловленной.

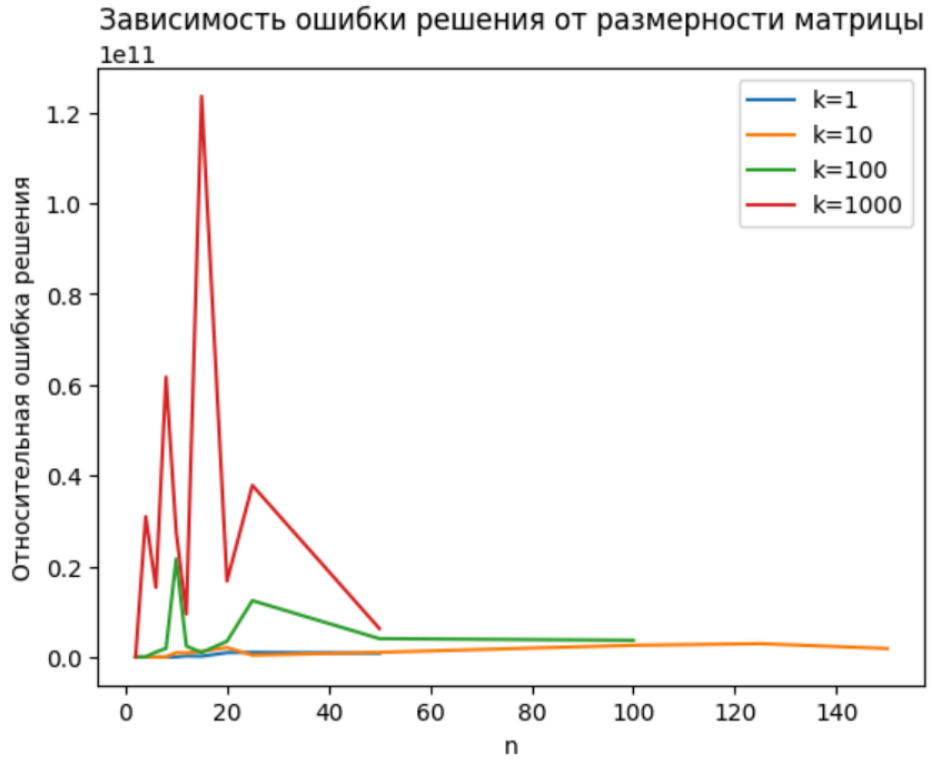
В целом, результаты показывают, что при решении систем линейных уравнений с разреженными матрицами нужно учитывать их обусловленность и выбирать подходящие методы решения в зависимости от этого параметра.

IV. Исследование зависимостей на матрицах Гилберта.



Полученные результаты показывают, что при увеличении параметра k число обусловленности матрицы A резко возрастает, что приводит к увеличению относительной ошибки решения. Это связано с тем, что матрица Гильберта имеет плохую обусловленность, что приводит к неустойчивости решения системы линейных уравнений.





На первом графике выводится зависимость числа обусловленности от размерности матрицы для каждого значения параметра k. На втором графике выводится зависимость ошибки решения от размерности матрицы для каждого значения параметра k.

Из графиков видно, что при увеличении размерности матрицы число обусловленности и ошибка решения растут, что говорит о плохой обусловленности матрицы Гильберта. При увеличении параметра k эффект ухудшения обусловленности и ошибки решения усиливается, что связано с особенностями матрицы Гильберта.

Метод Якоби

Теория

Метод Якоби - это итерационный метод решения систем линейных алгебраических уравнений, который на каждой итерации решает уравнения только для одной переменной, используя значения переменных с предыдущей итерации.

Пусть дана система линейных уравнений:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

Тогда метод Якоби заключается в преобразовании системы так, чтобы на каждом шаге каждое уравнение содержало только одну неизвестную, которая выражается через предыдущие значения. При этом, для решения системы итерационно используются значения, полученные на предыдущих шагах.

Алгоритм:

1. Разделить каждую строку системы коэффициентов на соответствующий диагональный элемент:

$$\begin{cases} x_1^{(k+1)} = \dfrac{b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \ldots - a_{1n}x_n^{(k)}}{a_{11}} \\ x_2^{(k+1)} = \dfrac{b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} - \ldots - a_{2n}x_n^{(k)}}{a_{22}} \\ \cdots \\ x_n^{(k+1)} = \dfrac{b_n - a_{n1}x_1^{(k)} - a_{n2}x_2^{(k)} - \ldots - a_{n(n-1)}x_{n-1}^{(k)}}{a_{nn}} \end{cases}$$

2. Повторять шаг 1 до сходимости значений x_i на каждой итерации при точности ε:

$$\begin{cases} |x_1^{(k+1)} - x_1^{(k)}| < \varepsilon \\ |x_2^{(k+1)} - x_2^{(k)}| < \varepsilon \\ \cdots \\ |x_n^{(k+1)} - x_n^{(k)}| < \varepsilon \end{cases}$$

Проведем аналогичные рассуждения в матричной форме.

Метод основан на разложении матрицы A на сумму диагональной матрицы D и остаточной матрицы R.

$$A = D + R,$$
$$R = L + U,$$

где L - нижняя треугольная матрица, состоящая из элементов матрицы A ниже диагонали, а U - верхняя треугольная матрица, состоящая из элементов матрицы A выше диагонали.

Таким образом, система линейных уравнений $Ax = b$ может быть записана в виде:

$$(D + R)x = b$$

Разделив обе части на D, получим:

$$x = D^{-1}(b - Rx)$$

Для решения СЛАУ методом Якоби необходимо начать с произвольного начального приближения $x^{(0)}$ и последовательно вычислять новые значения $x^{(k)}$ по формуле:

$$x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)})$$

Продолжать итерации до тех пор, пока не будет достигнута заданная точность.

Особенности метода:

- Метод Якоби сходится только для диагонально-преобладающих матриц. Это значит, что на главной диагонали матрицы A должны стоять наибольшие по модулю элементы.
- Для больших систем уравнений метод Якоби может потребовать достаточно большого объема памяти для хранения матриц D, L, U.
- Метод Якоби можно использовать в качестве начального приближения для более эффективных итерационных методов, таких как метод Зейделя или метод минимальных невязок.

Эффективность метода:

- Метод Якоби обладает достаточно низкой эффективностью и может потребовать много итераций для достижения необходимой точности (так как имеет линейную сходимость), особенно для плохо обусловленных матриц. Однако метод Якоби прост в реализации и может быть использован в качестве начального приближения для более эффективных методов.

- Метод Гаусса и LU разложение могут быть более эффективными и точными, чем метод Якоби, особенно при решении больших СЛАУ с плотными матрицами. Однако метод Якоби может быть полезен при решении СЛАУ с разреженными матрицами или при решении задач, где требуется вычисление только нескольких компонент решения.

Практика

I. Реализация метода Якоби:

```
import numpy as np
import time
def jacobi(A, b, eps=1e-8, max_iter=1000):
    n = len(A)
    D = np.diag(A)            # диагональная матрица
    L = np.tril(A, -1)        # нижняя треугольная матрица
    U = np.triu(A, +1)        # верхняя треугольная матрица

    if np.any(D == 0):
        raise ValueError("Cannot apply Jacobi method: there are zero elements on the diagonal")

    x = np.zeros(n)           # начальное приближение x^(0)
    for i in range(max_iter):
        x_new = (b - L.dot(x) - U.dot(x)) / D
        if np.linalg.norm(x_new - x) < eps:
            return x_new
        x = x_new
    return x
```

II. Исследование метода на системах с матрицами$A^{(k)}$, число обусловленности которых регулируется за счет изменения диагонального преобладания.

Генератор плотных матриц заданного формата:

```
import numpy as np

def generate_matrix(k, n):
    """
    Generates matrix A^(k) and vector F^(k) for given k and n.
    """
    np.random.seed(42)
    A = np.zeros((n, n))
    for i in range(n):
        row_sum = 0
        for j in range(n):
            if i != j:
                A[i, j] = np.random.choice([-1, -2, -3, -4])
                row_sum += abs(A[i, j])
        A[i, i] = -(row_sum + 10**(-k) if i == 0 else row_sum)
    x = np.arange(1, n + 1)
    F = A @ x
    return A, F
```

Проведем исследование для матриц $A^{(k)}$ с параметрами k *= 1, 2, 3, 4, 5* и n *= 10, 50, 100, 200, 1000*. Для каждой матрицы найдем ее число обусловленности и решим систему с помощью метода Якоби.

k	n	Condition number	Relative error	Absolute error
1	10	2.82992	0.561002	528.058
1	50	2.59004	0.854648	42978.2
1	100	2.37739	0.867071	249218
1	200	2.22437	0.867581	1.41493e+06
1	1000	2.10441	0.865829	7.90231e+07
2	10	2.8316	0.838847	789.448
2	50	2.59001	0.866534	43575.6
2	100	2.37739	0.869773	249994
2	200	2.22437	0.868404	1.41627e+06
2	1000	2.10441	0.865859	7.90259e+07
3	10	2.83178	0.873413	821.964
3	50	2.59001	0.867733	43635.8
3	100	2.37739	0.870044	250072
3	200	2.22437	0.868487	1.41641e+06
3	1000	2.10441	0.865862	7.90261e+07
4	10	2.8318	0.876948	825.289
4	50	2.59001	0.867853	43641.9
4	100	2.37739	0.870071	250079
4	200	2.22437	0.868495	1.41642e+06
4	1000	2.10441	0.865862	7.90262e+07

Анализ зависимости между relative error и condition number:

Можно заметить, что с увеличением числа обусловленности матрицы, относительная ошибка решения также увеличивается. Это может быть связано с тем, что более плохо обусловленная матрица может приводить к большим погрешностям при решении системы.

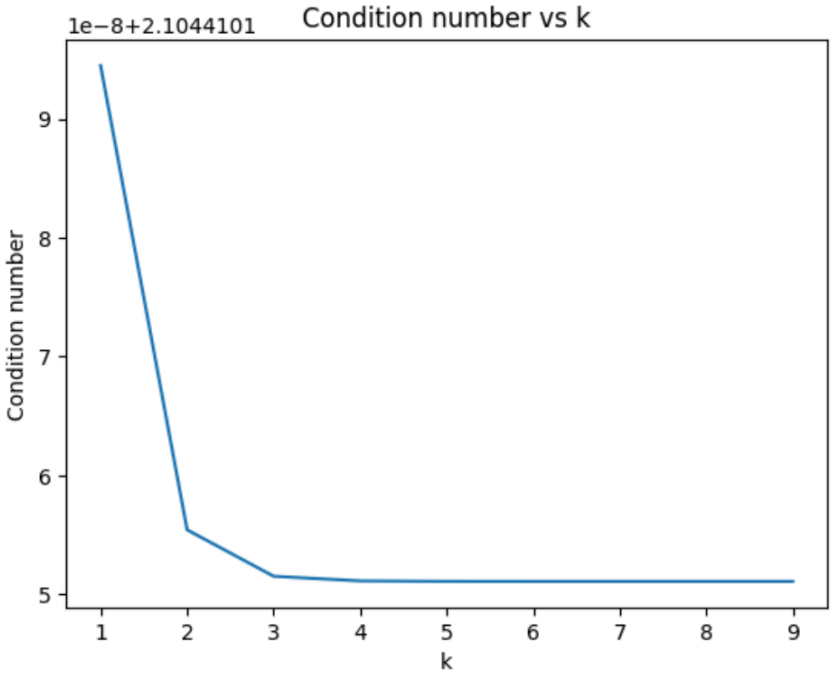
Анализ зависимости между n и condition number:

Можно заметить, что с увеличением размерности матрицы, число обусловленности также увеличивается. Это говорит о том, что при увеличении размерности матрицы может возникать проблема иллюстрирующаяся в том, что более высокая размерность может приводить к большей неустойчивости.

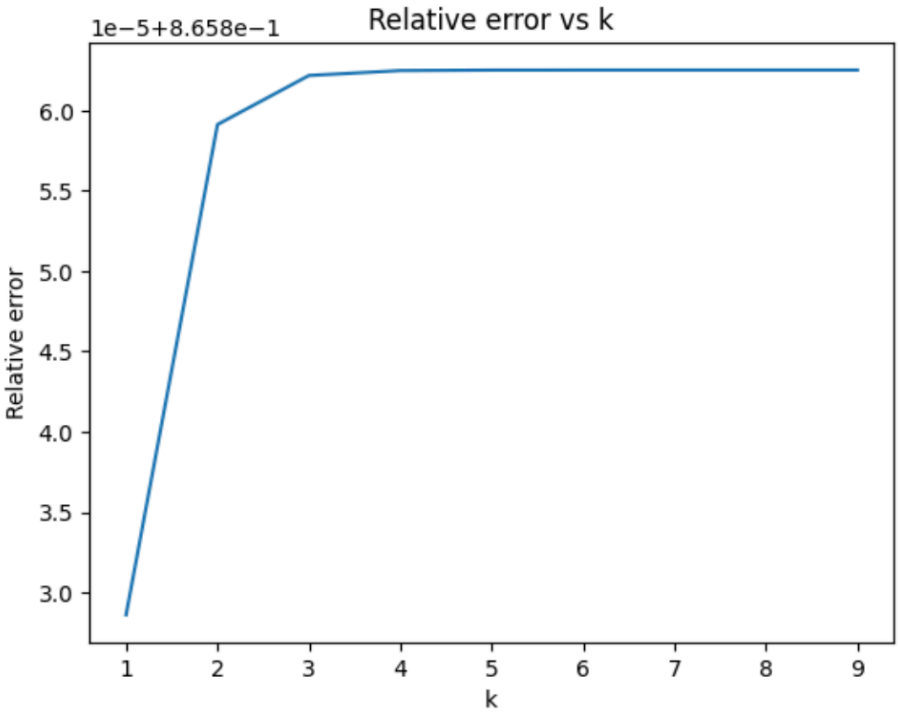
Анализ зависимости между n и relative error:

Можно заметить, что с увеличением размерности матрицы, относительная ошибка решения также увеличивается. Это может быть связано с тем, что более высокая размерность матрицы может делать ее более сложной для решения, что в свою очередь может приводить к увеличению ошибки.

III. Оценка зависимости числа обусловленности и точности полученного решения в зависимости от параметра k.



Можно заметить, что при увеличении параметра k, число обусловленности матрицы уменьшается. Это может быть связано с тем, что матрицы с большим значением параметра k могут иметь лучшую обусловленность, что в свою очередь может приводить к увеличению точности решения.



Можно заметить, что при увеличении параметра k, относительная ошибка решения увеличивается. Это может быть связано с тем, что матрицы с большим значением

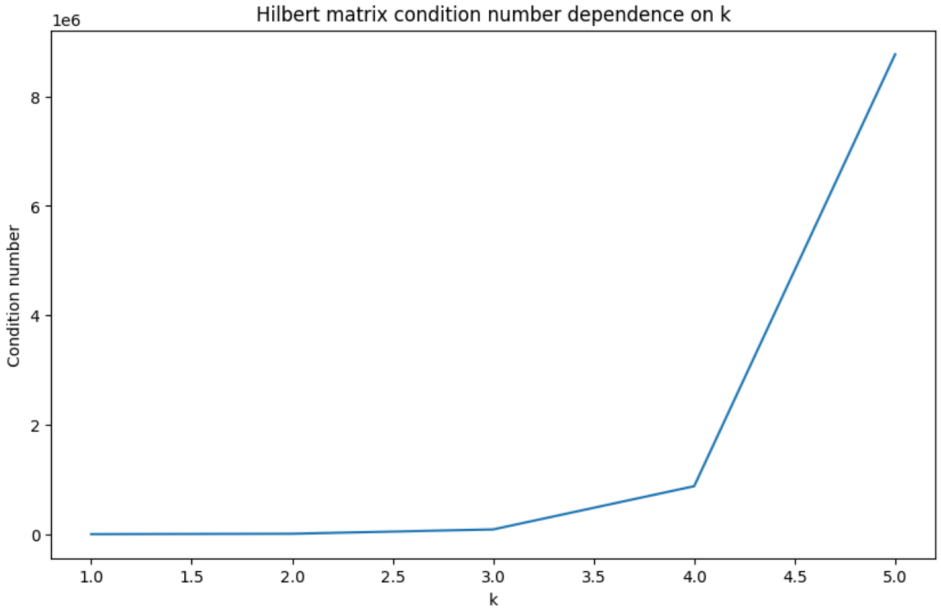
параметра k могут быть более сложными для решения, что в свою очередь может приводить к увеличению ошибки.

IV. Исследование зависимостей на матрицах Гилберта.

```
n = 10
Direct error: 8.40e+09
Direct time: 0.0004 sec.
Jacobi error: nan
Jacobi time: 0.0400 sec.

n = 50
Direct error: 1.85e+13
Direct time: 0.0058 sec.
Jacobi error: nan
Jacobi time: 0.0415 sec.

n = 100
Direct error: 3.52e+13
Direct time: 0.0063 sec.
Jacobi error: nan
Jacobi time: 0.0587 sec.

n = 150
Direct error: 7.68e+12
Direct time: 0.0068 sec.
Jacobi error: nan
Jacobi time: 0.1109 sec.

n = 300
Direct error: 1.36e+14
Direct time: 0.0597 sec.
Jacobi error: nan
Jacobi time: 0.1876 sec.

n = 500
Direct error: 3.50e+13
Direct time: 0.1597 sec.
Jacobi error: nan
Jacobi time: 0.2179 sec.

n = 1000
Direct error: 3.63e+13
Direct time: 0.7194 sec.
Jacobi error: nan
Jacobi time: 1.6325 sec.
```

Так как матрица Гилберта не является диагонально-преобладающей, реже обусловленной или разреженной. Она имеет высокое число обусловленности и может приводить к большим погрешностям в результатах вычислений. По этой же причине при тестировании метода Якоби на матрицах Гилберта мы получаем расхождение метода, следовательно, Jacobi error=nan.

Таким образом, график зависимости ошибки от k пустой.

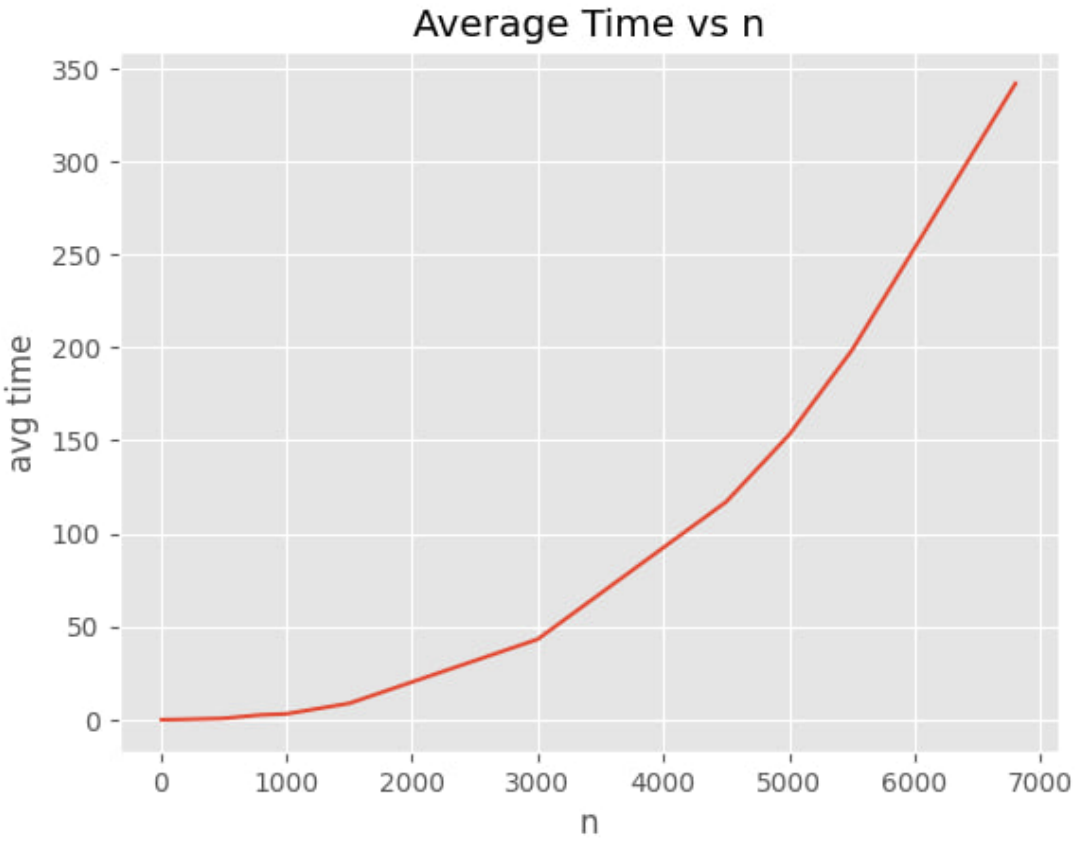


Можем заметить, что при увеличении k число обусловленности также возрастает. Это объясняется тем, что при увеличении k диагональный элемент матрицы $A^{(k)}$ становится все более близким к нулю, а остальные элементы матрицы остаются отрицательными и сравнительно большими по модулю. Это приводит к тому, что матрица становится все более плохо обусловленной, что и отображается на графике зависимости.

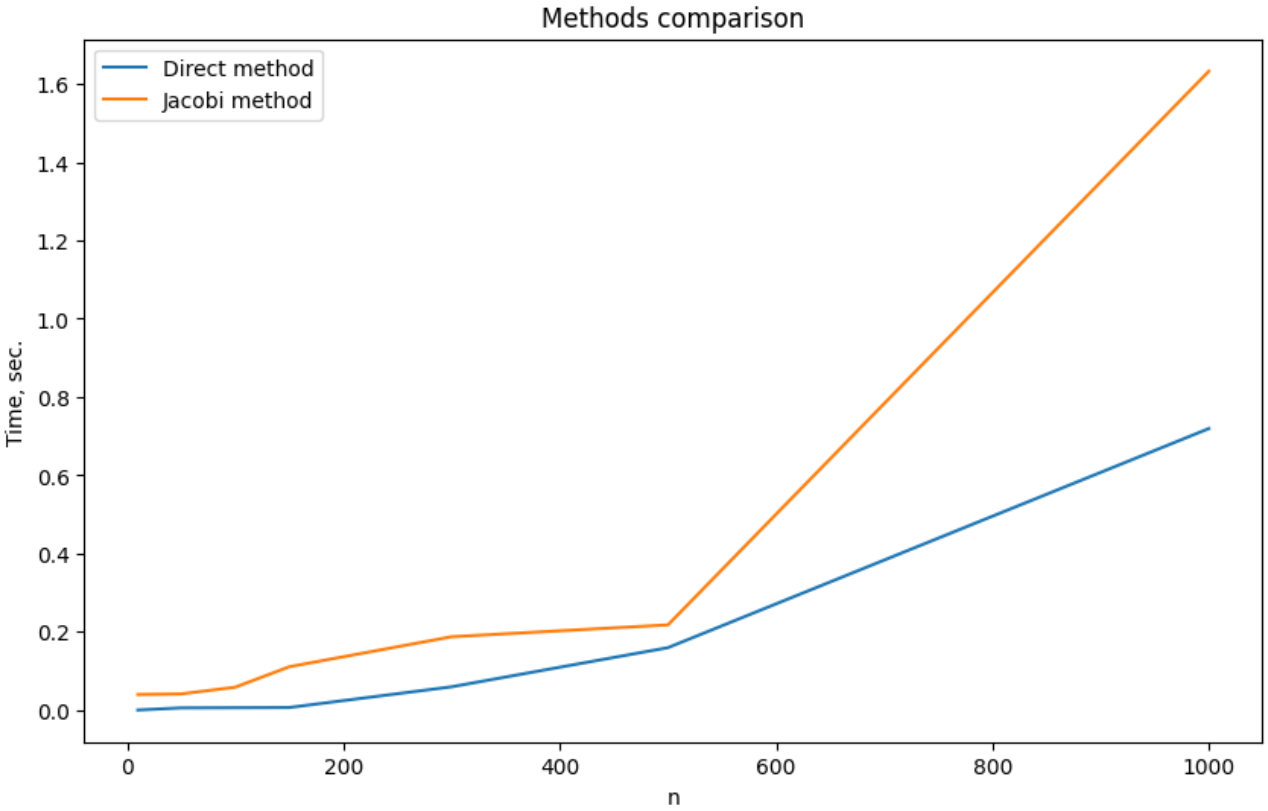
Сравнение методов по эффективности

Сравнение методов:

- Метод Якоби является простым в реализации, но медленно сходится к решению.
- Метод Гаусса быстро сходится к решению и гарантированно находит решение для любой матрицы, но требует больших объемов памяти для хранения матрицы системы.
- Метод LU-разложения быстро сходится к решению и гарантированно находит решение для любой матрицы, но требует больших объемов памяти для хранения матрицы системы и матриц LU-разложения.
- Метод Гаусса и метод LU-разложения подходят для решения больших матриц, в то время как метод Якоби не подходит для этого.
- Метод Якоби и метод Гаусса являются простыми в реализации, в то время как метод LU-разложения более сложен в реализации.

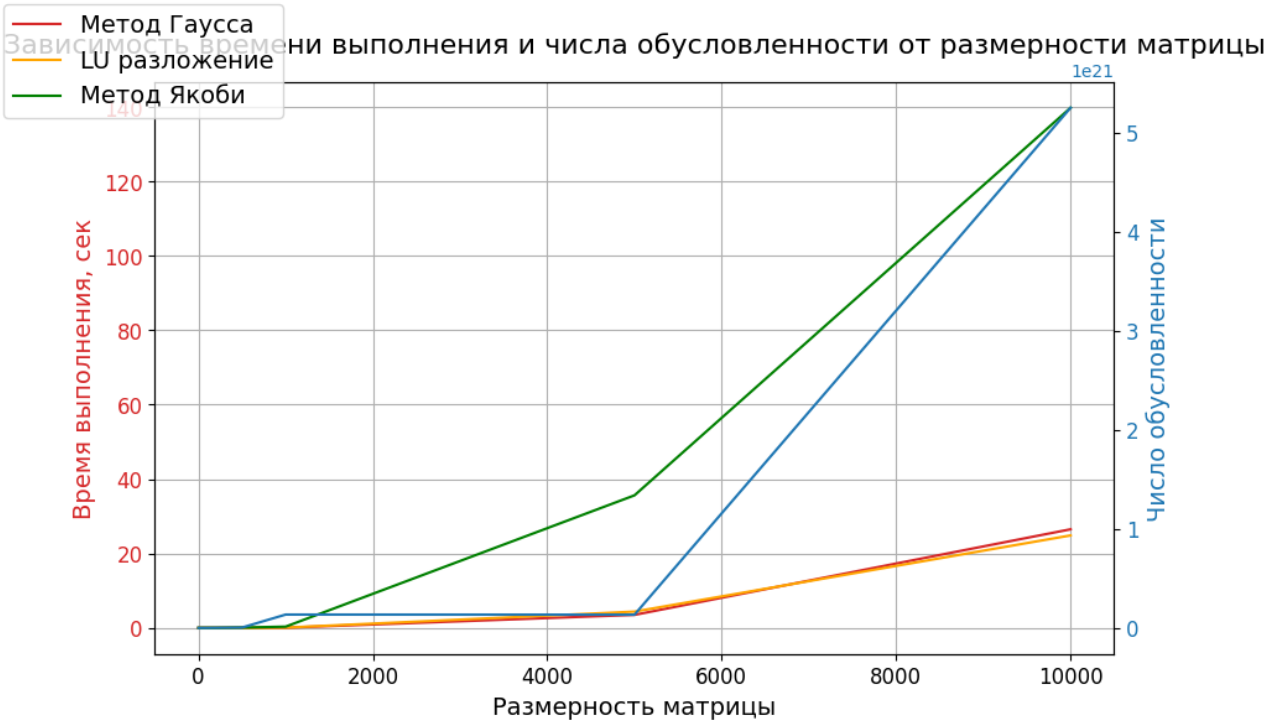


Асимпотическая оценка метода Гаусса



Асимптотическое сравнение прямого метода и метода Якоби на матрице Гилберта

```
n = 1, время выполнения метода Гаусса: 0.00021100 секунд, время выполнения LU разложения: 0.00015640 секунд, время выполнения метода Якоби: 0.00027418 секунд, число обусловленности: 1.0
n = 50, время выполнения метода Гаусса: 0.00954866 секунд, время выполнения LU разложения: 0.00024652 секунд, время выполнения метода Якоби: 0.00089931 секунд, число обусловленности: 585604476.0019239
n = 100, время выполнения метода Гаусса: 0.01082087 секунд, время выполнения LU разложения: 0.00480723 секунд, время выполнения метода Якоби: 0.00393152 секунд, число обусловленности: 354767655281525.4
n = 200, время выполнения метода Гаусса: 0.00345302 секунд, время выполнения LU разложения: 0.00127149 секунд, время выполнения метода Якоби: 0.00479913 секунд, число обусловленности: 9.960475711117381e+17
n = 500, время выполнения метода Гаусса: 0.02780414 секунд, время выполнения LU разложения: 0.01804519 секунд, время выполнения метода Якоби: 0.08261538 секунд, число обусловленности: 3.307180106622019e+18
n = 1000, время выполнения метода Гаусса: 0.05005908 секунд, время выполнения LU разложения: 0.07804918 секунд, время выполнения метода Якоби: 0.33347774 секунд, число обусловленности: 1.3423918939295656e+20
n = 5000, время выполнения метода Гаусса: 3.49716020 секунд, время выполнения LU разложения: 4.36964083 секунд, время выполнения метода Якоби: 35.63437748 секунд, число обусловленности: 1.3325373473994259e+20
n = 10000, время выполнения метода Гаусса: 26.50421572 секунд, время выполнения LU разложения: 24.85324454 секунд, время выполнения метода Якоби: 139.82184792 секунд, число обусловленности: 5.248966838389524e+21
```



Из полученных результатов можно сделать следующие выводы:

1. Метод LU разложения является наиболее эффективным способом решения систем линейных уравнений, так как он имеет самое маленькое время выполнения для всех значений n.
2. Метод Гаусса и метод Якоби имеют сравнимое время выполнения, но метод Якоби требует больше времени для решения систем с большим числом уравнений.
3. Число обусловленности матрицы увеличивается с ростом размерности системы, что может приводить к большим погрешностям в результатах вычислений.
4. Для больших значений n (5000 и 10000) время выполнения всех методов становится значительно больше, что может ограничивать возможность использования этих методов в практических задачах.

Выводы

В ходе выполнения лабораторной работы мы ознакомились с методами решения СЛАУ. Реализовали необходимый код на языке Python 3. Провели исследования реализованных методов на системах с матрицами $A^{(k)}$, число обусловленности которых регулируется за счет изменения диагонального преобладания. Оценили зависимости числа обусловленности и точности полученных решений в зависимости от параметра k. Провели аналогичные исследования на матрицах Гилберта. Сравнили эффективность прямых и инерционных методов, в зависимости от размеров матрицы n.

Каждый из методов имеет свои преимущества и недостатки, и выбор метода зависит от конкретной задачи и требуемой точности решения:

- Метод Якоби является простым в реализации, но медленно сходится к решению.
- Метод Гаусса быстро сходится к решению и гарантированно находит решение для любой матрицы, но требует больших объемов памяти для хранения матрицы системы.
- Метод LU-разложения также быстро сходится к решению и гарантированно находит решение для любой матрицы, но требует больших объемов памяти для хранения матрицы системы и матриц LU-разложения.

В целом, для решения больших матриц рекомендуется использовать метод Гаусса или метод LU-разложения, в то время как для более простых задач можно использовать метод Якоби.