Изображение выглядит как текст, Шрифт, логотип, Графика

Контент, сгенерированный ИИ, может содержать ошибки.

**Отчет по лабораторной работе №2**

**по дисциплине “Вычислительная математика” Вариант №4**

Выполнил студент гр. 5130904/30007 Голиков В.С.

Руководитель Скуднева Е.В.

# Оглавление

[Формулировка задания 3](#_Toc191414321)

[Описание действий при выполнении 4](#_Toc191414322)

[Вывод 6](#_Toc191414323)

[Приложение. Скриншоты выполнения программы 7](#_Toc191414324)

[Приложение. Код программы 9](#_Toc191414325)

# Формулировка задания

Изображение выглядит как текст, Шрифт, документ

Контент, сгенерированный ИИ, может содержать ошибки.

# Описание действий при выполнении

1. Выполнение задания приведено с использованием языка программирования Python. В программе использовались библиотеки **numpy** и **scipy.linalg**
   1. numpy as np:

* np.matmul() -> Matrix Multiplication. Умножение матриц (в том числе векторов). В качестве параметров передаются две матрицы (или вектор(ы))
* np.matrix() -> трансформация данных в матрицу (например из массива или строки)
* np.transpose() -> выполняет транспонирование матрицы. Параметры - матрица
* Изображение выглядит как текст, Шрифт, белый, Графика

  Контент, сгенерированный ИИ, может содержать ошибки.np.linalg.cond() -> вычисление числа обусловленности матрицы. В качестве параметра передается матрица, вычисление происходи по формуле (рис. 1)

рис.

* np.asarray().reshape() -> используется для удаления массива массивов [[]] (для удобства использования)
* np.substruct() - > функция вычитания
  1. scipy.linalg:
* lu\_factor() -> LU разложение матрицы (оно же DECOMP)
* lu\_solve() -> решение системы вида Ax=B, где A – матрица, которая хранит в себе LU разложение
* norm() -> выполняет поиск нормы матрицы

1. После определения нужных для работы библиотек и функций, реализовал собственные функции для удобства решения поставленной задачи:

- initMatrixC – инициализирует матрицу (двумерный массив) заданными элементами Cij (вычисление приведено в задании)

- initVectorD – инициализирует вектор (массив) заданными элементами di (вычисление приведено в задании)

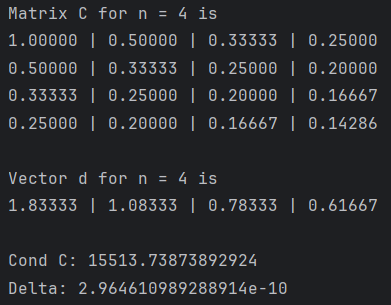
- printMatrix – выводит матрицу на экран

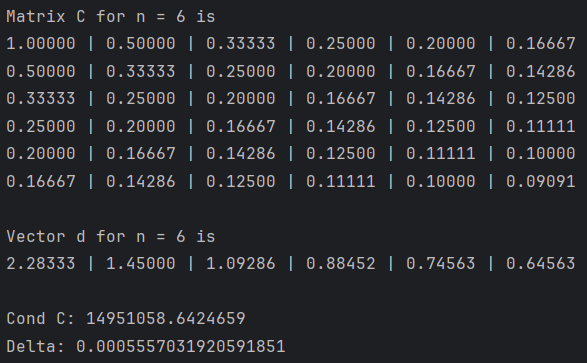
1. Проинициализировав нужные для работы матрицы и вектора, выполняем цикл для n = 4, 6, 8, 10, 12, где n – размерность матрицы. Внутри цикла с помощью функций из библиотеки scipy.lnalg (пункт 1.2) выполняем LU разложение и решение системы уравнений, находим вектора решений X1 и X2 для Сx1=d и CTCx2=CTd соответственно.
2. По заданию требуется проанализировать связь между числом обусловленности cond() и δ = (||X1|| - ||X2||) **/** ||X1||. Для этого выводим после исходной матрицы и вектора результат выполнения функции np.linalg.cond() для получения числа обусловленности и выполняем арифметические действия с полученным в пункте 3 векторами решений (X1 и X2) для вычисления δ. Ознакомиться с результатами вычисления для каждого n из [4, 6, 8, 10, 12] можно в скриншотах выполнения программы

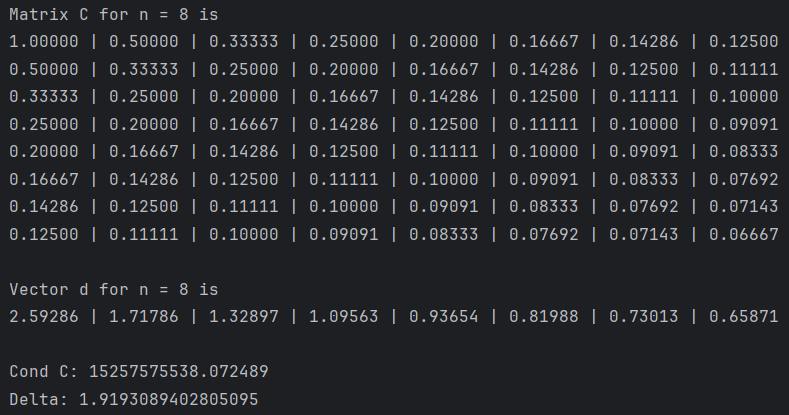
# Вывод

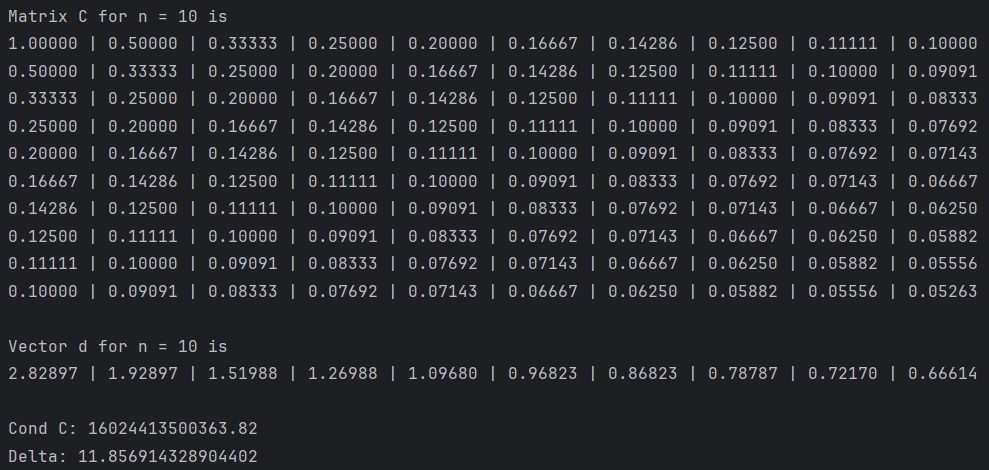
Выполнив работу, я изучил новые функции из стандартных библиотек Python и применил их на практике  
  
В данной лабораторной работе требовалось оценить зависимость значения числа обусловленности и дельты, вычисленной по формуле, при решении линейных систем. Для решения подобного вида систем, использовал функции **DECOMP** и **SOLVE** (в Python: **lu\_factor()** и **lu\_solve** соответственно). Для вычисления cond использовал соответствующую функцию numpy.cond(). Выполнив работу и проведя анализ, я выяснил, что при возрастании **числа обусловленности** растет и **дельта**.

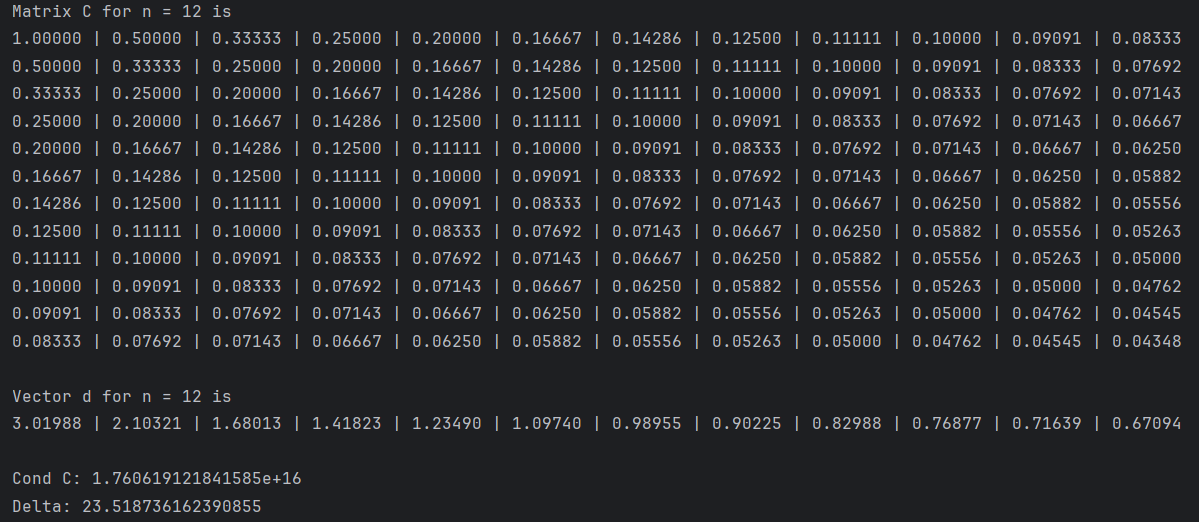
# Приложение. Скриншоты выполнения программы











# Приложение. Код программы

import numpy as np  
from scipy.linalg import lu\_factor, lu\_solve, norm  
  
###-FUNCTIONS-###  
  
#initialization of matrix C  
def initMatrixC(n):  
 array = [[0 for \_ in range(n)] for \_ in range(n)]  
 for i in range(0, n):  
 for j in range (0, n):  
 array[i][j] = 1/((i+1) + (j+1) - 1)  
 return array  
  
#initialization of vector D  
def initVectorD(n):  
 vect = [0 for \_ in range(n)]  
 sum = 0  
 for i in range(0,n):  
 for k in range (1, n):  
 sum += 1.0 / ((i + 1) + k - 1)  
 vect[i] = sum  
 sum = 0  
 return vect  
  
#Print Matrix in correct view  
def printMatrix(array):  
 for row in array:  
 print(" | ".join(f"{elem:.5f}" for elem in row))  
  
###-END-FUNCTIONS-###  
  
###-MAIN-###  
for n in [4, 6, 8, 10, 12]:  
 C = initMatrixC(n)  
 d = initVectorD(n)  
 print(f"Matrix C for n = {n} is")  
 printMatrix(C)  
 print(f"\nVector d for n = {n} is")  
 print(" | ".join(f"{elem:.5f}" for elem in d))  
  
 # C \* X1 = d  
 lu, pivot = lu\_factor(C)  
 X1 = [list(lu\_solve((lu, pivot), d))]  
  
 # C\_transposed \* C \* x2 = C\_transposed \* d ->  
 # C\_transposed \* C = C\_mul \  
 # C\_transposed \* d = d\_mul / -> C\_mul \* X2 = d\_mul  
  
 C\_transposed = np.matrix.transpose(np.matrix(C))  
 C\_mul = np.matmul(C\_transposed, np.matrix(C))  
 d\_mul = np.matmul(C\_transposed, d)  
 d\_mul = np.asarray(d\_mul).reshape(-1) #after previous command d\_mul type is

array of array ([[]]). This command solves issue  
  
 # C\_mul \* X2 = d\_mul  
 lu, pivot = lu\_factor(C\_mul)  
 X2 = [list(lu\_solve((lu, pivot), d\_mul))]  
  
 print(f"\nCond С: {np.linalg.cond(C)}") # cond(A) = ||A|| \* ||A^(-1)||  
 print(f"Delta: {norm(np.subtract(X1, X2)) / norm(X1)}")

# delta = (||X1|| - ||X2||) / ||X1||  
  
 if n != 12:  
 print("\n","-" \* 116,"\n")  
###-END-MAIN-###