Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра обчислювальної техніки

Лабораторна робота № 2

з дисципліни «Чисельні методи»

на тему

**“** **Розв’язання систем лінійних алгебраїчних рівнянь (СЛАР) прямими методами. Звичайний метод Гауса та метод квадратних коренів”**

Виконав:

студент гр. ІП-93

Завальнюк Максим

Викладач:

доц. Рибачук Л.В.

Київ – 2021

### Зміст

[Зміст 2](#_Toc65353097)

[1 Постановка задачі 3](#_Toc65353098)

[2 Розв’язок 4](#_Toc65353099)

[3 Розв’язок за допомогою NumPy 5](#_Toc65353100)

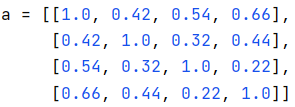
[4 Лістинг програми 6](#_Toc65353101)

### 1 Постановка задачі

Розв’язати систему рівнянь методом квадратних коренів з кількістю значущих цифр m = 6. Вивести матрицю T, вектор у та розв’язок системи. Навести результат перевірки: вектор нев’язки r = b – Ax, де x - отриманий розв’язок.

Порівняти корені рівнянь, отримані у NumPy, із власними результатами за допомогою методу середньоквадратичної похибки.

### 2 Розв’язок

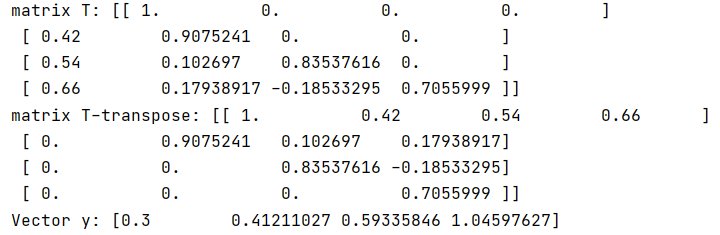
Початкова матриця системи А

Вектор правої частини b



Нижче приведена наведені результати виконання програми.

Проміжні результати (матриця Т та вектор у):

Вектор коренів рівнянь **х**:

Вектор нев’язки r = b – Ax:

З вигляду матриці незв’язки випливає, що метод квадратного кореня є абсолютно точним для розв’язання систем з симетричною матрицею А. Така точність забезпечується в першу чергу тим, що сам метод заснований на строгих математичних перетвореннях.

# 3 Розв’язок за допомогою NumPy

Нижче наведено розв’язок системи у NumPy:

xm = np.linalg.solve(a, b)

print(**'NumPy solution:'**, xm)  
print(**'Residual vector'**, np.matrix(np.subtract(b, np.dot(a, np.linalg.solve(a, b))), int))

Порівняння отриманого результату (п. 2) із результатом з NumPy за допомогою методу середньоквадратичної похибки:

print(**'Fault:'**, round(get\_fault(x, xm)))

# 4 Лістинг програми

import numpy as np  
from math import sqrt  
  
  
def get\_transpose(matrix: list) -> list:  
 *"""  
 Function for getting transpose matrix  
 :param matrix: input matrix  
 :return: transpose input matrix  
 """* matrix\_t = matrix.copy()  
 n = len(matrix\_t)  
 for i in range(n):  
 for j in range(i, n):  
 matrix\_t[i][j], matrix\_t[j][i] = matrix\_t[j][i], matrix\_t[i][j]  
 return matrix\_t  
  
  
def subtract(matrix\_a: list, matrix\_b: list) -> list:  
 *"""  
 Function for substracting two matrices  
 :param matrix\_a: the first input matrix  
 :param matrix\_b: the second input matrix  
 :return: the result matrix  
 """* result = matrix\_a.copy()  
  
 for i in range(len(matrix\_a)):  
 for j in range(len(matrix\_a[0])):  
 result[i][j] = matrix\_a[i][j] - matrix\_b[i][j]  
  
 return result  
  
  
def multiply(matrix\_a: list, matrix\_b: list) -> list:  
 *"""  
 Function for multiplying two matrices  
 :param matrix\_a: the first input matrix A[i][j]  
 :param matrix\_b: the second input matrix B[m][n]  
 :return: the result matrix C[i][n]  
 """* result = []  
 *# Creating result by sizes* for \_ in range(len(matrix\_a)):  
 array = [0] \* len(matrix\_b[0])  
 result.append(array)  
  
 if len(matrix\_a[0]) == len(matrix\_b): *# j == m* for i in range(len(matrix\_a)):  
 for j in range(len(matrix\_b[0])):  
 for k in range(len(matrix\_b)):  
 result[i][j] += matrix\_a[i][k] \* matrix\_b[k][j]  
 else:  
 raise ValueError(**'j != m'**)  
  
 return result  
  
  
def factorization(matrix\_a: list) -> list:  
 *"""  
 Cholesky decomposition  
 :param matrix\_a: start matrix  
 :return: Lower-triangular matrix  
 """* t = np.zeros\_like(matrix\_a)  
 n = len(matrix\_a)  
 for j in range(n):  
 for i in range(j, n):  
 if i == j:  
 sum\_k = 0  
 for k in range(j):  
 sum\_k += t[i][k] \*\* 2  
 t[i][j] = sqrt(matrix\_a[i][j] - sum\_k)  
 else:  
 sum\_k = 0  
 for k in range(j):  
 sum\_k += t[i][k] \* t[j][k]  
 t[i][j] = (matrix\_a[i][j] - sum\_k) / t[j][j]  
 return t  
  
  
def solve(lower\_matrix: list, upper\_matrix: list, vector\_b: list) -> list:  
 *"""  
 The solve main function  
 :param lower\_matrix: Lower-triangular matrix  
 :param upper\_matrix: Upper-triangular matrix  
 :param vector\_b: vector B  
 :return: vector x, the solution  
 """* n = len(lower\_matrix)  
 y = np.zeros(n)  
 vector\_x = np.zeros(n)  
  
 *# forward substitution* for i in range(n):  
 sum\_j = 0  
 for j in range(i):  
 sum\_j += lower\_matrix[i][j] \* y[j]  
 y[i] = (vector\_b[i] - sum\_j) / lower\_matrix[i][i]  
 print(**'Vector y:'**, y)  
 *# backward substitution* for i in range(n - 1, -1, -1):  
 sum\_j = 0  
 for j in range(i + 1, n):  
 sum\_j += upper\_matrix[i][j] \* vector\_x[j]  
 vector\_x[i] = (y[i] - sum\_j) / upper\_matrix[i][i]  
  
 return vector\_x  
  
  
def get\_fault(x: list, xm: list) -> float:  
 *"""  
 Function for finding get\_fault  
 :param x: my solution vector  
 :param xm: NumPy solution vector  
 :return: fault  
 """* sum\_k = 0  
 n = len(x)  
 for k in range(1, n):  
 sum\_k += (x[k] - xm[k]) \*\* 2  
 result = sqrt(sum\_k / n)  
 return result  
  
  
a = [[1.0, 0.42, 0.54, 0.66],  
 [0.42, 1.0, 0.32, 0.44],  
 [0.54, 0.32, 1.0, 0.22],  
 [0.66, 0.44, 0.22, 1.0]]  
b = [0.3, 0.5, 0.7, 0.9]  
T = factorization(a)  
print(**'matrix T:'**, T)  
U = get\_transpose(T)  
print(**'matrix T-transpose:'**, U)  
x = solve(T, U, b)  
xm = np.linalg.solve(a, b)  
print(**'Solution vector:'**, x)  
print(**'Residual vector'**, np.matrix(np.subtract(b, np.dot(a, x)), int))  
print(**'NumPy solution:'**, xm)  
print(**'Residual vector'**, np.matrix(np.subtract(b, np.dot(a, np.linalg.solve(a, b))), int))  
print(**'Fault:'**, round(get\_fault(x, xm)))