Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра обчислювальної техніки

Лабораторна робота № 3

з дисципліни «Чисельні методи»

на тему

**“Розв’язання систем лінійних алгебраїчних рівнянь (СЛАР) ітераційними методами. Метод простої ітерації. Метод Зейделя”**

Виконав:

студент гр. ІП-93

Завальнюк Максим

Викладач:

доц. Рибачук Л.В.

Київ – 2021

### Зміст

[Зміст 2](#_Toc65353798)

[1 Постановка задачі 3](#_Toc65353799)

[2 Розв’язок 4](#_Toc65353800)

[3 Розв’язок за допомогою NumPy 5](#_Toc65353801)

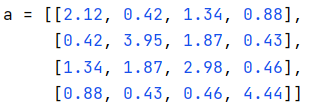
[4 Лістинг програми 6](#_Toc65353802)

### 1 Постановка задачі

Розв’язати систему рівнянь методом простих ітерацій з кількістю значущих цифр m = 6. Якщо матриця не є матрицею із діагональною перевагою, привести систему до еквівалентної, у якій є діагональна перевага (письмово). Вивести проміжні результати та кінцевий результат, на кожній ітерації вектори нев’язки та результати перших трьох та останньої ітерацій методу. Навести результат перевірки: вектор нев’язки r = b – Ax, де x - отриманий розв’язок.

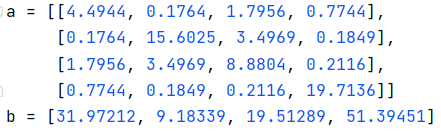
Порівняти корені рівнянь, отримані у NumPy, із власними результатами за допомогою методу середньоквадратичної похибки і вивести вектор нев’язки для цього розв’язку.

### 2 Розв’язок

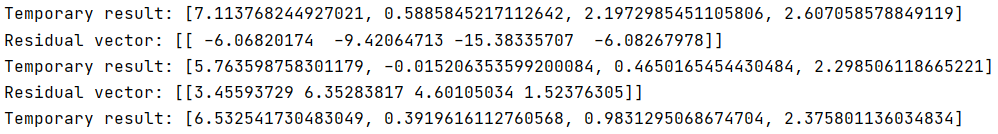
Початкова матриця системи А

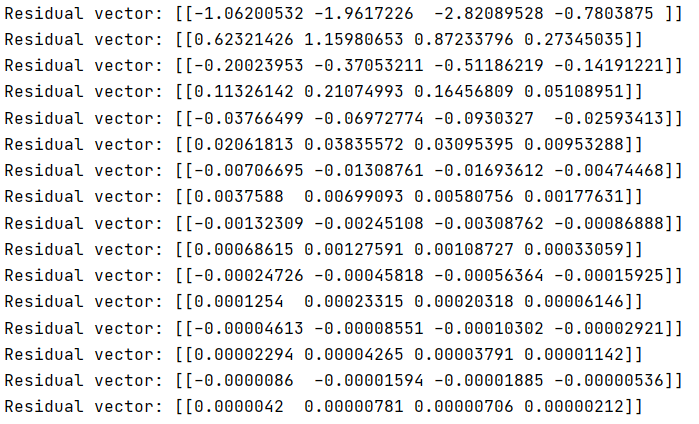
Вектор правої частини b

Як видно, матриця А не є матрицею з діагональною перевагою, тому домножимо ліву і прави частину рівняння на транспоновану матрицю А.

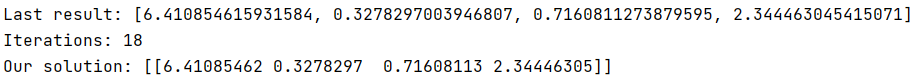
Результати:

Нижче приведена наведені результати виконання програми.

Результати перших трьох ітерацій:

Вектори нев’язки на кожній ітерації:

Останній результат, к-сть ітерацій та вектор **x**:



Вектор нев’язки r = b – Ax:



З к-сті ітерацій випливає, що при великому числі невідомих метод Гаусса стає вельми складним у плані обчислювальних і тимчасових витрат. Тому іноді зручніше використовувати наближені (ітераційні) чисельні методи, зокрема метод Якобі.

### 3 Розв’язок за допомогою NumPy

Нижче наведено розв’язок системи у NumPy:

sol\_np = np.linalg.solve(a, b)  
print(**f'NumPy solution:** {sol\_np}**'**)  
print(**f'Residual vector for NumPy:** {np.matrix(np.subtract(b, np.dot(a, sol\_np)))}**'**)

Порівняння отриманого результату (п. 3) із результатом з NumPy за допомогою методу середньоквадратичної похибки:

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



Це дуже добре, оскільки результат вважають гарним, якщо відносна похибка не перевищує 0.1 %. У даному випадку взагалі 0 %.

### 4 Лістинг програми

import numpy as np  
from checker.fault import get\_fault  
  
np.set\_printoptions(suppress=True)  
  
  
def solve\_jacobi(matrix\_a: list, vector\_b: list, epsilon=10 \*\* (-6)) -> list:  
 *"""  
 Solve by Jacobi method(simple iteration)  
 :param matrix\_a: start matrix  
 :param vector\_b: start vector  
 :param epsilon: number for comparing  
 :return: solution vector  
 """* solution\_vector = [0 for \_ in range(len(matrix\_a[0]))]  
  
 matrix\_c = []  
 vector\_d = []  
 *# Create matrix C and vector D* for i in range(len(matrix\_a)):  
 element\_d = vector\_b[i] / matrix\_a[i][i]  
 vector\_d.append(element\_d)  
  
 element\_c = []  
 for j in range(len(matrix\_a[0])):  
 if i == j:  
 element\_c.append(0)  
 else:  
 element\_c.append((-1) \* matrix\_a[i][j] / matrix\_a[i][i])  
 matrix\_c.append(element\_c)  
  
 iterations = 0  
 tmp = 0  
 *# Start the main algorithm* while True:  
 divs = []  
 left\_part = np.dot(matrix\_c, solution\_vector)  
 for i in range(len(solution\_vector)):  
 x\_next = left\_part[i] + vector\_d[i]  
 divs.append(abs(x\_next - solution\_vector[i]))  
 solution\_vector[i] = x\_next  
 *# Check if we need to stop* if max(divs) < epsilon:  
 print(**f'Last result:** {solution\_vector}**'**)  
 *# It's time to stop!* break  
 else:  
 if tmp < 3:  
 print(**f'Temporary result:** {solution\_vector}**'**)  
 tmp += 1  
 print(**f'Residual vector:** {np.matrix(np.subtract(vector\_b, np.dot(matrix\_a, solution\_vector)), float)}**'**)  
 iterations += 1  
 print(**f'Iterations:** {iterations}**'**)  
 return solution\_vector  
  
  
a = [[4.4944, 0.1764, 1.7956, 0.7744],  
 [0.1764, 15.6025, 3.4969, 0.1849],  
 [1.7956, 3.4969, 8.8804, 0.2116],  
 [0.7744, 0.1849, 0.2116, 19.7136]]  
b = [31.97212, 9.18339, 19.51289, 51.39451]  
print(**f'Matrix A:'**, np.matrix(a))  
print(**f'Vector b:'**, b)  
sol = solve\_jacobi(a.copy(), b.copy())  
sol\_np = np.linalg.solve(a, b)  
print(**f'Our solution:** {np.matrix(sol)}**'**)  
print(**f'Residual vector:** {np.matrix(np.subtract(b, np.dot(a, sol)))}**'**)  
print(**f'NumPy solution:** {sol\_np}**'**)  
print(**f'Residual vector for NumPy:** {np.matrix(np.subtract(b, np.dot(a, sol\_np)))}**'**)  
print(**'Fault:'**, round(get\_fault(sol, sol\_np), 6))