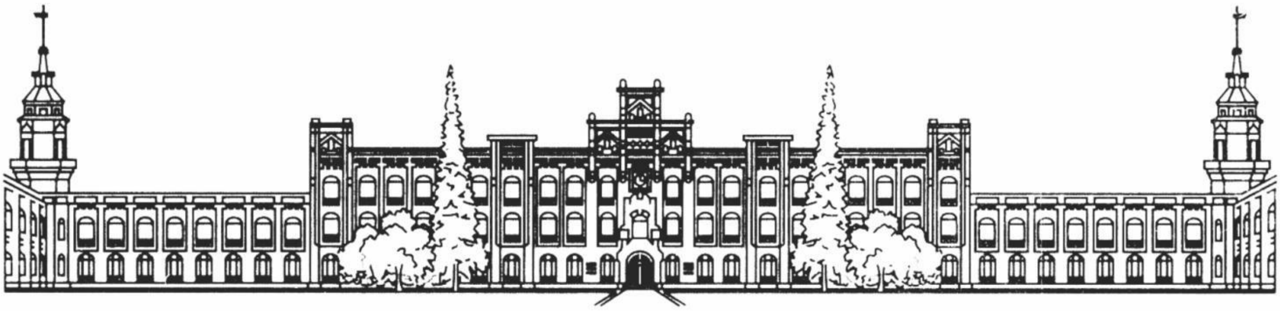

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського» Інститут прикладного системного аналізу

Лабораторна робота №3

курсу «Чисельні методи 1»

з теми «Ітераційні методи розв’язання СЛАР»

Варіант №9

Виконав студент 2 курсу групи КА-91
Косицький Вадим Вікторович
перевірила старший викладач
Хоменко Ольга Володимирівна

Київ-2021

Завдання 1

1. Розв'язати систему методом Якобі. Для цього в допрограмовому етапі виконати перевірку достатніх умов збіжності з поясненням, задати початкове наближення, визначити критерій зупинки ітераційного процесу (можна робити фото написаного і вставляти в звіт).
2. Реалізувати метод Якобі для довільної СЛАР розмірності n×n. Розв'язати СЛАР з точністю $\boldsymbol{\varepsilon} = \mathbf{10^{-5}}$.
3. Задати інші початкові наближення та з'ясувати чи змінюється при цьому ітераційний процес, написати про це у висновку.

9	$2{,}389 \cdot x_1 + 0{,}273 \cdot x_2 + 0{,}126 \cdot x_3 + 0{,}418 \cdot x_4 = 0{,}144$ $0{,}329 \cdot x_1 + 2{,}796 \cdot x_2 + 0{,}179 \cdot x_3 + 0{,}278 \cdot x_4 = 0{,}297$ $0{,}186 \cdot x_1 + 0{,}275 \cdot x_2 + 2{,}987 \cdot x_3 + 0{,}316 \cdot x_4 = 0{,}529$ $0{,}197 \cdot x_1 + 0{,}219 \cdot x_2 + 0{,}274 \cdot x_3 + 3{,}127 \cdot x_4 = 0{,}869.$

Текст програми:

Файл main.py

```
from func import *

eps = 0.00001

#main
n, data = input_data()
solution = Jakoby(data, n, eps)
verification (data, solution, n)
```

Файл func.py (Тут реалізовано усі потрібні алгоритми)

```
import numpy as np
import pandas as pd
import pathlib
from pathlib import Path
eps1 = 0.0001


def input_data():
    print("Enter number of equations: ")
    n = int(input())
    print("Enter koef of  equations: ")
    data = []
    for j in range(n):
        data.append([float(i) for i in input().split()])
        pass
    return n, data

def output(data, n):
    Data = data.copy()
    for i in range(n):
        print("%+.7F  %+.7F  %+.7F  %+.7F  %+.7F  " % tuple(data[i]))
    print("\n")


def Jakoby(data, n, eps):
    verif_diag_cond(data, n)
    Data, Vec = np.delete(data,n,1) , np.delete(data, np.s_[:n:],1)
    D = np.diag(np.diag(Data))
    B = np.dot(-np.linalg.inv(D), Data - D)
    C = np.dot(np.linalg.inv(D), Vec)
    q = np.linalg.norm(B, ord= np.inf )
    Delta = [0]
    Solve = []
    answer = input("Enter X(0). If you enter auto it can be C: ")
    if(answer=="auto"):
        Solve.append(np.copy(C))

    else:
        Solve.append(np.array(list(map(float, answer.split()))).reshape(4,1))

    Solve.append(np.dot(B, Solve[0])+C)
    Delta.append(np.linalg.norm(Solve[1]-Solve[0]))
    while 1:
        if((Delta[len(Delta)-1] < eps) and (q <= 1/2)) or (Delta[len(Delta)-1] < eps*(1-
q)/q and (q > 1/2)): break
```

```
        Solve.append(np.dot(B,Solve[len(Solve)-1])+C)
        Delta.append(np.linalg.norm(Solve[len(Solve)-1]-Solve[len(Solve)-2]))
    Solution = np.c_[np.matrix(np.array(Solve)), np.array(Delta)]
    # df = pd.DataFrame(Solution, columns = ['x{}'.format(i) for i in
range(n)].append('delta'))
    col = ['x_{}'.format(i) for i in range(n)]
    col.append('|x(k)-x(k-1)|')
    df = pd.DataFrame(Solution, columns=col)
    print(df)
    return Solve[len(Solve)-1]


def verification (data, solution, n):
    b = [data[i][n] for i in range(n)]
    Data = np.delete(data, n, 1)
    Res = np.dot(Data, solution.reshape(n,))
    print("\nb - Ax*: ", b-Res)


    pass

def verif_diag_cond(data,n):
    key = 1
    for i in range(n):
        s = 0
        for j in data[i]: s +=abs(j)
        if (abs(data[i][i]) < s - abs(data[i][i]) - abs(data[i][n])): key = 0
        pass
    if (key):  print("\nVerification is done")
    else: print("\nVerification is false")
```

Результати роботи:

Першим розглянемо приклад, де за початкове наближення обрано вектор С

```
Run:    main

C:\python3.8\python.exe C:/Users/kosva/Desktop/КП3/main.py
Enter number of equations:
4
Enter koef of  equations:
2.389 0.273 0.126 0.418 0.144
0.329 2.796 0.179 0.278 0.297
0.186 0.275 2.987 0.316 0.529
0.197 0.219 0.274 3.127 0.869

Verification is done
Enter X(0). If you enter auto it can be C: auto
        x_0       x_1       x_2       x_3  |x(k)-x(k-1)|
0  0.060276  0.106223  0.177101  0.277902       0.000000
1 -0.009827  0.060161  0.134168  0.251147       0.097955
2  0.002382  0.073819  0.145605  0.262551       0.024422
3 -0.001777  0.070516  0.142380  0.259824       0.006786
4 -0.000752  0.071483  0.143232  0.260600       0.001820
5 -0.001043  0.071231  0.142997  0.260393       0.000496
6 -0.000966  0.071301  0.143060  0.260449       0.000134
7 -0.000987  0.071282  0.143043  0.260434       0.000037
8 -0.000982  0.071287  0.143048  0.260438       0.000010

b - Ax*:  [-3.72756016e-06 -3.88042577e-06 -3.79135222e-06 -3.53173930e-06]

Process finished with exit code 0
```

Тепер розглянемо такі випадки, де початкове наближення це нульовий вектор, та вектор (3, 3, 3, 4).

```
Run:   main ×
C:\python3.8\python.exe C:/Users/kosva/Desktop/КП3/main.py
Enter number of equations:
4
Enter koef of  equations:
2.389 0.273 0.126 0.418 0.144
0.329 2.796 0.179 0.278 0.297
0.186 0.275 2.987 0.316 0.529
0.197 0.219 0.274 3.127 0.869

Verification is done
Enter X(0). If you enter auto it can be C: 0 0 0 0
        x_0       x_1       x_2       x_3   |x(k)-x(k-1)|
0  0.000000  0.000000  0.000000  0.000000       0.000000
1  0.060276  0.106223  0.177101  0.277902       0.351441
2 -0.009827  0.060161  0.134168  0.251147       0.097955
3  0.002382  0.073819  0.145605  0.262551       0.024422
4 -0.001777  0.070516  0.142380  0.259824       0.006786
5 -0.000752  0.071483  0.143232  0.260600       0.001820
6 -0.001043  0.071231  0.142997  0.260393       0.000496
7 -0.000966  0.071301  0.143060  0.260449       0.000134
8 -0.000987  0.071282  0.143043  0.260434       0.000037
9 -0.000982  0.071287  0.143048  0.260438       0.000010

b - Ax*:  [-3.72756016e-06 -3.88042577e-06 -3.79135222e-06 -3.53173930e-06]

Process finished with exit code 0
```

```
Run:   main ×
C:\python3.8\python.exe C:/Users/kosva/Desktop/КП3/main.py
Enter number of equations:
4
Enter koef of  equations:
2.389 0.273 0.126 0.418 0.144
0.329 2.796 0.179 0.278 0.297
0.186 0.275 2.987 0.316 0.529
0.197 0.219 0.274 3.127 0.869

Verification is done
Enter X(0). If you enter auto it can be C: 3 3 3 4
         x_0       x_1       x_2       x_3   |x(k)-x(k-1)|
0    3.000000  3.000000  3.000000  4.000000       0.000000
1   -1.140645 -0.836552 -0.709073 -0.384074       8.052416
2    0.260471  0.324023  0.365778  0.470482       2.279393
3   -0.078362  0.005378  0.081277  0.206749       0.605673
4    0.019201  0.089684  0.159613  0.275341       0.165733
5   -0.006566  0.066369  0.138520  0.256426       0.044836
6    0.000520  0.072632  0.144272  0.261530       0.012189
7   -0.001392  0.070922  0.142714  0.260141       0.003307
8   -0.000872  0.071385  0.143137  0.260518       0.000898
9   -0.001013  0.071259  0.143022  0.260415       0.000244
10  -0.000975  0.071294  0.143054  0.260443       0.000066
11  -0.000985  0.071284  0.143045  0.260436       0.000018
12  -0.000982  0.071287  0.143047  0.260438       0.000005

b - Ax*:  [-1.83205603e-06 -1.90991612e-06 -1.86411956e-06 -1.73745217e-06]

Process finished with exit code 0
```

Висновок:

Виконуючи дану лабораторну роботу я навчився застосовувати такий ітераційний метод розв'язання СЛАР як метод Якобі. Спочатку я написав функцію **verif_diag_cond**, яка перевіряє чи виконуються умови діагональної переваги в матриці коефіцієнтів. Саме це я використав для перевірки умов збіжності методу. Після підтвердження збіжності я визначив матрицю B та вектор C, згідно із алгоритмом розв'язання методу, знайшов $q = |B|_\infty$, та запустив ітераційний процес, умовою зупинки якого є

або що $\left|\left| x^{(k)} - x^{(k-1)} \right|\right| \leq \varepsilon$ при $q \leq \frac{1}{2}$, або $\left|\left| x^{(k)} - x^{(k-1)} \right|\right| \leq \frac{1-q}{q}\varepsilon$.

Після чого за допомогою бібліотеки Pandas вивів таблицю з отриманими результатами. Тепер виводжу отриманий вектор нев'язки, та переконуюсь, що усі розрахунки і справді були зроблені правильно.

Як видно із прикладів роботи програми, при підборі початкового наближення як вектора C, ітераційний процес закінчується на 8-му кроці, при підборі більш віддалених значень процес відбувався відповідно 9 та 12 кроків.

Зауваження про введення даних в консоль: спочатку вводимо кількість невідомих, потім вводимо матрицю A|b (відповідно з Ax = b). Далі вводимо «auto», що буде повідомляти системі обрати за початкове наближення саме вектор C, або ж вводимо своє власне наближення. (Вектор C обирається з

x $= Bx + c$)