Завдання: Обрати на вибір метод Якобі або метод Зейделя та розв’язати систему обраним методом:

$$\begin{cases} 3.738 \cdot x_1 + 0.195 \cdot x_2 + 0.275 \cdot x_3 + 0.136 \cdot x_4 = 0.815 \\ 0.519 \cdot x_1 + 5.002 \cdot x_2 + 0.405 \cdot x_3 + 0.283 \cdot x_4 = 0.191 \\ 0.306 \cdot x_1 + 0.381 \cdot x_2 + 4.812 \cdot x_3 + 0.418 \cdot x_4 = 0.423 \\ 0.272 \cdot x_1 + 0.142 \cdot x_2 + 0.314 \cdot x_3 + 3.935 \cdot x_4 = 0.352 \end{cases}$$

1. Виконаємо перевірку достатніх умов збіжності.

Будемо знаходити розв'язок системи методом Якобі. Достатною умовою збіжності цього методу є умова діагональної переваги, тобто:

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad i = 1, \ldots, n$$

$1. |3.783| > |0.195| + |0.275| + |0.136| = 0.606$

$2. |5.002| > |0.519| + |0.405| + |0.283| = 1.207$

$3. |4.812| > |0.306| + |0.381| + |0.418| = 1.105$

$4. |3.935| > |0.272| + |0.142| + |0.314| = 0.728$

Як бачимо, достатня умова збіжності виконується

2. Перетворимо систему до вигляду: $\vec{x} = B\vec{x} + \vec{c}$

Для методу Якобі треба перетворити систему до вигляду $\vec{x} = B\vec{x} + \vec{c}$, де $B = -D^{-1}(L+R), \quad \vec{c} = D^{-1}\vec{b}$

маємо:

$$L + R = \begin{pmatrix} 0 & 0.195 & 0.275 & 0.136 \\ 0.519 & 0 & 0.405 & 0.283 \\ 0.306 & 0.381 & 0 & 0.418 \\ 0.272 & 0.142 & 0.314 & 0 \end{pmatrix} \qquad D = \begin{pmatrix} 3.738 & 0 & 0 & 0 \\ 0 & 5.002 & 0 & 0 \\ 0 & 0 & 4.812 & 0 \\ 0 & 0 & 0 & 3.935 \end{pmatrix}$$

Знайдемо обернену матрицю D^{-1}, вектор $\vec{c}$, а також добуток $-D^{-1}(L+R)$ за допомогою функцій бібліотеки numpy

In [1]:
```python
import numpy as np

D = np.array([[3.738, 0, 0, 0],
             [0, 5.002, 0, 0],
             [0, 0, 4.812, 0],
             [0, 0, 0, 3.935]])

sum_L_and_R = np.array([
    [0, 0.195, 0.275, 0.136],
    [0.519, 0, 0.405, 0.283],
    [0.306, 0.381, 0, 0.418],
    [0.272, 0.142, 0.314, 0]])

b = np.array([[0.815],
              [0.191],
              [0.423],
              [0.352]])

#Знайдемо обернену до матриці D матрицю
D_inversed = np.linalg.inv(D)

#Тепер можемо знайти матрицю B
B = -np.matmul(D_inversed, sum_L_and_R)

#Знайдемо вектор c
c = np.matmul(D_inversed, b)


print(B,"\n-----------------------")
print(c)
```

```
[[-0.         -0.05216693 -0.07356875 -0.03638309]
 [-0.1037585  -0.         -0.08096761 -0.05657737]
 [-0.06359102 -0.07917706 -0.         -0.08686617]
 [-0.06912325 -0.0360864  -0.0797967  -0.        ]]
-----------------------
[[0.21803103]
 [0.03818473]
 [0.08790524]
 [0.08945362]]
```

Отримали:

$$B = \begin{pmatrix} 0 & -0.05217 & -0.07357 & -0.03638 \\ -0.10376 & 0 & -0.08097 & -0.05658 \\ -0.06359 & -0.07918 & 0 & -0.08687 \\ -0.06912 & -0.03609 & -0.0798 & 0 \end{pmatrix} \qquad \vec{c} = \begin{pmatrix} 0.21803 \\ 0.03818 \\ 0.08791 \\ 0.08945 \end{pmatrix}$$

Отже, отримали таку систему:

$$\begin{cases} x_1 = -0.05217 \cdot x_2 - 0.07357 \cdot x_3 - 0.03638 \cdot x_4 + 0.21803 \\ x_2 = -0.10376 \cdot x_1 - 0.08097 \cdot x_3 - 0.05658 \cdot x_4 + 0.03818 \\ x_3 = -0.06359 \cdot x_1 - 0.07918 \cdot x_2 - 0.08687 \cdot x_4 + 0.08791 \\ x_4 = -0.06912 \cdot x_1 - 0.03609 \cdot x_2 - 0.0798 \cdot x_3 + 0.08945 \end{cases}$$

3. Задамо початкове наближення, а також визначимо критерій зупинки

$q = \|B\|_{\infty} = \max\limits_{1 \leqslant i \leqslant nj} \sum |a_{ij}| = \max \{0.16212;\ 0.2413;\ 0.22963;\ 0.185\} = 0.2413$

Як бачимо, q < 0.5, тому в якості критерію зупинки можна взяти: $\|x^{(k)} - x^{(k+1)}\| < \varepsilon$

За початкове наближення візьмемо вектор $\vec{c}$

4. Реалізуємо даний метод для довільної СЛАР:

In [2]:
```python
import math


#Функція пошуку норми вектора
def norma(arr):

    max = abs(arr[0])
    for i in range(arr.size-1):
        if max < abs(arr[i+1]):
            max = abs(arr[i+1])

    return max


def metod_yakobi(B, c, x_0):
    x_1 = x_0
    x_1 = np.matmul(B, x_0)
    x_1 = x_1 + c
    tab = np.array([x_0.transpose()[0]])
    norm = np.array([[0]])

    tab = np.vstack((tab, x_1.transpose()[0]))
    norm = np.vstack((norm, norma(x_0 - x_1)))


    while norma(x_0 - x_1) > 0.00001:

        x_0 = x_1

        x_1 = np.matmul(B, x_0)
        x_1 = x_1 + c

        tab = np.vstack((tab, x_1.transpose()[0]))
        norm = np.vstack((norm, norma(x_0 - x_1)))


    table = np.hstack((tab, norm))

    return x_1, table
```

In [3]:
```python
x, table = metod_yakobi(B, c, c)
```

5. Продемонструємо отриманий результат у вигляді таблиці:

In [4]:
```python
#Зробимо з двохмірного масиву table об'єкт типу pandas.DataFrame, щоб можна було вивести його у вигляді табли
#Для цього створимо окрему функцію:

import pandas as pd

def to_df(table):

    column_values = ['x_1', 'x_2', 'x_3', 'x_4', '|| x^(k) - x^(k-1)||']

    df = pd.DataFrame(data = table,
                      columns = column_values)

    df['|| x^(k) - x^(k-1)||'][0] = None

    return df
```

In [5]:
```python
to_df(table)
```

Out[5]:

	x_1	x_2	x_3	x_4	|| x^(k) - x^(k-1)||
0	0.218031	0.038185	0.087905	0.089454	NaN
1	0.206317	0.003384	0.063247	0.065990	0.034801
2	0.210801	0.007923	0.068785	0.070023	0.005539
3	0.210010	0.006781	0.067790	0.069108	0.001142
4	0.210176	0.006996	0.068010	0.069283	0.000220
5	0.210142	0.006951	0.067968	0.069246	0.000045
6	0.210149	0.006960	0.067977	0.069254	0.000009

6. Виконаємо перевірку

In [6]:
```python
A = np.array([[3.738, 0.195, 0.275, 0.136],
              [0.519, 5.002,  0.405, 0.283],
              [0.306, 0.381, 4.812, 0.418],
              [0.272, 0.142, 0.314, 3.935]])

b = np.array([[0.815],
              [0.191],
              [0.423],
              [0.352]
             ])


print(np.matmul(A, x))
```

```
[[0.81500522]
 [0.19100924]
 [0.42300862]
 [0.35200594]]
```

7. Обчислимо вектор нев'язки і знайдемо його норму

In [7]:
```python
print("вектор нев'язки:\n\n", b - np.matmul(A, x))
print("\n\nЙого норма: ", norma(b - np.matmul(A, x))[0])
```

```
вектор нев'язки:

 [[-5.21532301e-06]
 [-9.23734056e-06]
 [-8.61796083e-06]
 [-5.93888943e-06]]


Його норма:  9.237340562573415e-06
```

8. Задамо інші початкові наближення і поглянемо чи зміниться при цьому ітераційний процес:

Візьмемо початкове наближення $x^{(0)} = (1, 1, 1, 1)^T$

In [8]:
```python
x_0 = np.array([[1],
                [1],
                [1],
                [1]])

x, table = metod_yakobi(B,c,x_0)

to_df(table)
```

Out[8]:

	x_1	x_2	x_3	x_4	|| x^(k) - x^(k-1)||
0	1.000000	1.000000	1.000000	1.000000	NaN
1	0.055912	-0.203119	-0.141729	-0.095553	1.203119
2	0.242530	0.049265	0.108732	0.104228	0.252384
3	0.203670	-0.001681	0.059528	0.062235	0.050946
4	0.211475	0.008711	0.069681	0.070686	0.010392
5	0.209879	0.006601	0.067627	0.068961	0.002110
6	0.210202	0.007031	0.068046	0.069311	0.000429
7	0.210136	0.006943	0.067961	0.069240	0.000087
8	0.210150	0.006961	0.067978	0.069255	0.000018
9	0.210147	0.006958	0.067975	0.069252	0.000004

Візьмемо випадкове початкове наближення

In [9]:
```python
x_0 = np.random.uniform(low=-5, high=5, size=(4, 1))

x, table = metod_yakobi(B,c,x_0)

to_df(table)
```

Out[9]:

	x_1	x_2	x_3	x_4	|| x^(k) - x^(k-1)||
0	-2.526553	4.533744	-4.113602	-3.576608	NaN
1	0.414280	0.835760	0.200289	0.428743	4.313891
2	0.144098	-0.045274	-0.041855	0.014675	0.881034
3	0.222938	0.025792	0.081052	0.084467	0.122907
4	0.207649	0.003712	0.064349	0.066645	0.022080
5	0.210679	0.007659	0.068618	0.069831	0.004269
6	0.210043	0.006818	0.067836	0.069139	0.000840
7	0.210169	0.006987	0.068003	0.069276	0.000168
8	0.210143	0.006952	0.067969	0.069248	0.000034
9	0.210148	0.006959	0.067976	0.069253	0.000007

Візьмемо ще одне випадкове наближення

In [10]:
```python
x_0 = np.random.uniform(low=-5, high=5, size=(4, 1))

x, table = metod_yakobi(B,c,x_0)

to_df(table)
```

Out[10]:

	x_1	x_2	x_3	x_4	|| x^(k) - x^(k-1)||
0	-2.458849	4.254571	0.810766	3.493940	NaN
1	-0.190684	0.029987	-0.396104	0.041189	4.224583
2	0.244109	0.087711	0.094079	0.133160	0.490182
3	0.201689	-0.002295	0.053870	0.061908	0.090006
4	0.211935	0.009393	0.069884	0.071296	0.016013
5	0.209806	0.006503	0.067491	0.068889	0.002891
6	0.210220	0.007053	0.068065	0.069331	0.000573
7	0.210133	0.006939	0.067956	0.069237	0.000114
8	0.210151	0.006962	0.067979	0.069255	0.000023
9	0.210147	0.006957	0.067974	0.069252	0.000005

9. Розв'яжемо систему за допомогою бібліотеки numpy:

In [11]:
```python
np.linalg.solve(A,b)
```

Out[11]:
```
array([[0.21014759],
       [0.00695823],
       [0.06797513],
       [0.06925225]])
```

10. Висновки

Під час виконання цієї лабораторної роботи був розглянутий метод знаходження коренів СЛАР, а саме метод Якобі. Як видно вище, якщо не брати за початкове наближення вектор $\vec{c}$, то кількість ітерацій збільшиться.