НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС
"ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ"
НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО"
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Лабораторна робота № 2
з предмету "Чисельні методи"
з теми «Ітераційні методи розв'язання СЛАР»
Варіант № 13

Виконала:
студентка групи
КА-02
Шапошнікова Софія

Перевірила:
Хоменко О.В.

Київ 2022

Завдання

1. В допрограмовому етапі виконати перевірку достатніх умов збіжності з поясненням, задати початкове наближення, визначити критерій зупинки ітераційного процесу.
   Перетворення системи до вигляду $x = Bx + c$ можна робити у допрограмовому етапі або запрограмувати та написати відповідні коментарі в програмі.

2. Реалізувати обраний метод для довільної СЛАР. Текст програми з коментарями, які описують основні етапи алгоритму, вставити в звіт. Розв'язати СЛАР з точністю $\epsilon = 10^{-5}$

3. Отримані результати записати у звіт у вигляді таблиці

4. Виконати перевірку. Обчислити вектор нев'язки $b - Ax^*$

5. Задати інші початкові наближення та з'ясувати чи змінюється при цьому ітераційний процес, написати про це у висновку.

6. Розв'язати систему за допомогою функції numpy.linalg.solve (мова Python).

Варіант 13

$$\begin{cases} 5,554 \cdot x_1 + 0,252 \cdot x_2 + 0,496 \cdot x_3 + 0,237 \cdot x_4 = 0,442 \\ 0,580 \cdot x_1 + 4,953 \cdot x_2 + 0,467 \cdot x_3 + 0,028 \cdot x_4 = 0,464 \\ 0,319 \cdot x_1 + 0,372 \cdot x_2 + 8,935 \cdot x_3 + 0,520 \cdot x_4 = 0,979 \\ 0,043 \cdot x_1 + 0,459 \cdot x_2 + 0,319 \cdot x_3 + 4,778 \cdot x_4 = 0,126 \end{cases}$$

1 Допрограмовий етап

Приведемо систему до вигляду: $y = Bx + C$

$$\begin{cases} x_1 = 0.0796 - (0,0454 \cdot x_2 + 0,0893 \cdot x_3 + 0,0427 \cdot x_4) \\ x_2 = 0.0937 - (0,1171 \cdot x_1 + 0,0943 \cdot x_3 + 0,0057 \cdot x_4) \\ x_3 = 0.1096 - (0,0357 \cdot x_1 + 0,0416 \cdot x_2 + 0,0582 \cdot x_4) \\ x_4 = 0.0264 - (0,0089 \cdot x_1 + 0,0961 + \cdot x_2 + 0,0668 \cdot x_3) \end{cases}$$

Перевіримо достатні умови збіжності:
Для цього знайдемо норму матриці B

$$\|B\|_\infty = \max\{0,0454 + 0,0893 + 0,0427; 0,1171 + 0,0943 + 0,0057;$$
$$0,0357 + 0,0416 + 0,0582; 0,0089 + 0,0961 + 0,0668\} =$$
$$= \max\{0,1774; 0,2171; 0,1335; 0.1717\} = 0,2171$$

З теореми **про достатню умову збіжності** випливає, що збіжність методу до єдиного розв'язку x_* виконується, оскільки: $\|B\|_\infty < 1$

За теоремою про **оцінку похибок методу простої ітерації**, знайдемо оцінки похибок, поклавши: $q = \|B\|_\infty$

1) **Апотеріорна**: виконання цієї умови буде критерієм зупинки ітераційного процесу

$\|x^{(k)} - x^{(k-1)}\| \leq \frac{1-\|B\|}{\|B\|} \cdot \epsilon, \epsilon = 10^{-5}$

2) **Апріорна**: дозволяє завчасно підрахувати к-ть ітерацій, необхідних для отримання розв'язку x_*

$\frac{q^k}{1-q} \cdot \|x^{(1)} - x^{(0)}\| \leq \epsilon, q = \|B\|, \epsilon = 10^{-5}$

Початкове наближення: $x^{(0)} = \begin{pmatrix} 0.0796 \\ 0.0937 \\ 0.1096 \\ 0.0264 \end{pmatrix}$

2 Реалізація обраного методу для розв'язання СЛАР

2.1 Метод Якобі

Схема розв'язання СЛАР методом Якобі

1) Перевірити виконання умов збіжності. Для збіжності методу достатньо виконання хоча б однієї з умов:

∘ для матриці $\mathbf{A}$ виконується умова діагональної переваги;

∘ $\|\mathbf{B}\| \le q < 1$;

∘ всі власні числа матриці $\mathbf{B}$ були за модулем менше 1 – це необхідна і достатня умова;

∘ всі корені рівняння

$$\begin{vmatrix} a_{11}\lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22}\lambda & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn}\lambda \end{vmatrix} = 0$$

за модулем менші одиниці – це необхідна і достатня умова.

2) Перетворити систему $\mathbf{Ax} = \mathbf{b}$ до виду

$$\mathbf{x} = \mathbf{Bx} + \mathbf{c},$$

де $\mathbf{B} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{R})$, $\mathbf{c} = \mathbf{D}^{-1}\mathbf{b}$.

3) Задати довільним чином початкове наближення $\mathbf{x}^{(0)}$, або покласти $\mathbf{x}^{(0)} = \mathbf{c}$. Покласти $k = 0$.

4) Обчислити наступне наближення $\mathbf{x}^{(k+1)}$ за формулою

$$\mathbf{x}^{(k+1)} = \mathbf{Bx}^{(k)} + \mathbf{c}.$$

5) Перевірити умови зупинки. Якщо виконується умова $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| \leq \varepsilon$ при $q \leq \dfrac{1}{2}$ або умова $\dfrac{q}{1-q}\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| \leq \varepsilon$, де $\|\mathbf{B}\| \leq q < 1$, зупинитись і покласти $\mathbf{x}_* = \mathbf{x}^{(k)}$. Інакше покласти $k = k + 1$ і перейти до пункту 4.

2.2 Програмна реалізація алгоритму

In [2]:
```
import numpy as np
import copy
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import handcalcs.render
```

In [3]:
```
A = np.array([[5.554, 0.252, 0.496, 0.237],
[0.580, 4.953, 0.467, 0.028],
[0.319, 0.372, 8.935, 0.520],
[0.043, 0.459, 0.319, 4.778]])
A
```

Out[3]:
```
array([[5.554, 0.252, 0.496, 0.237],
       [0.58 , 4.953, 0.467, 0.028],
       [0.319, 0.372, 8.935, 0.52 ],
       [0.043, 0.459, 0.319, 4.778]])
```

In [4]:
```
B = [0.442, 0.464, 0.979, 0.126]
eps = 0.00001
```

In [5]:
```
progr_x = np.linalg.solve(np.array(A), np.array(B))
progr_x
```

Out[5]:
```
array([0.06640598, 0.07609387, 0.10335724, 0.01156268])
```

In [6]:
```
np.allclose(np.dot(np.array(A), progr_x), np.array(B))
```

Out[6]:
```
True
```

Перевірка умови діагональної переваги

In [21]:
```
def condOfDiagAdvantage(A):
    diag = np.diag(A)
    A = copy.deepcopy(A[~np.eye(A.shape[0],dtype=bool)].reshape(A.shape[0],-1))
    for i, j in enumerate(A):
        if sum(abs(np.array(j))) > abs(diag[i]):
            return False
    return True
```

In [22]:
```
condOfDiagAdvantage(A)
```

Out[22]:
```
True
```

Зведення системи до вигляду x = Bx + c:

In [23]:
```
def reformMatr(A, B):
    diag = np.diag(A)
    A = copy.deepcopy(A[~np.eye(A.shape[0],dtype=bool)].reshape(A.shape[0],-1))
    B = copy.deepcopy(B)
    for i, j in enumerate(A):
        A[i] = np.array(j)/diag[i]
        B[i] = B[i]/diag[i]
    return A, np.array(B)
```

In [24]:
```
newA, newB = reformMatr(A, B)
newA, newB
```

Out[24]:
```
(array([[0.0453727 , 0.08930501, 0.04267195],
        [0.11710075, 0.09428629, 0.00565314],
        [0.03570229, 0.04163402, 0.0581981 ],
        [0.00899958, 0.0960653 , 0.06676434]]),
 array([0.07958228, 0.0936806 , 0.10956911, 0.02637087]))
```

Норма Чебишова (норма-максимум або максимум серед модулів елементів)

In [27]:
```
def chebNorm(A):
    matrSum = np.zeros(A.shape[0])
    for i, j in enumerate(A):
         matrSum[i] = sum(abs(np.array(j)))
    return max(matrSum)
```

In [28]:
```
chebNorm(reformMatr(A, B)[0])
```

Out[28]:
```
0.21704017767009892
```

Реалізація методу Якобі для розв'язання СЛАР

In [29]:
```
def jacobiMethod(A, B, eps):
    stopCriterion, res = [], []
    x_next, x =  np.zeros(B.shape),np.zeros(B.shape)
    checkStopCriterion = lambda x, x_next, eps: chebNorm((x_next-x).reshape(B.shape[0], 1)) < eps
    while True:
        for i, j in enumerate(A):
            x_next[i] = B[i] - np.dot(j, np.delete(x, i))
        stopCriterion.append(chebNorm((x_next-x).reshape(B.shape[0], 1)))
        res.append(copy.deepcopy(x_next))
        if checkStopCriterion(x, x_next, eps): break
        x = copy.deepcopy(x_next)
    return x_next, np.array(stopCriterion), np.array(res)
#         print("%01.10f" % (chebNorm((x_next-x).reshape(B.shape[0], 1))))
```

In [30]:
```
jacobiMethod(newA, newB, eps)
```

Out[30]:
```
(array([0.0664063 , 0.07609425, 0.10335751, 0.01156301]),
 array([1.09569110e-01, 1.97990880e-02, 2.65197420e-03, 5.13707354e-04,
        8.67201903e-05, 1.48216052e-05, 2.57329826e-06]),
 array([[0.07958228, 0.0936806 , 0.10956911, 0.02637087],
        [0.06442137, 0.07388151, 0.10129281, 0.0093399 ],
        [0.06678557, 0.07653348, 0.10364957, 0.0119309 ],
        [0.06634421, 0.07601978, 0.10330396, 0.01149752],
        [0.06641688, 0.0761065 , 0.10336633, 0.01157391],
        [0.06640411, 0.07609168, 0.10335568, 0.01156076],
        [0.0664063 , 0.07609425, 0.10335751, 0.01156301]]))
```

Перевірка достатніх умов збіжності та розв'язання СЛАР вказаним методом

- Для матриці A виконується умова діагональної переваги
- $\|B\|_\infty < 1$

In [31]:
```
def solveSystemIter(A, B, eps, func):
    if not cond0fDiagAdvantage(A):
        print("Your matrix doesn't follow the condition of diagonal advantage")
        return -1
    else:
        newA, newB = reformMatr(A, B)
        if chebNorm(newA) >= 1:
            print("Inputs don't follow the sufficient condition of convergence")
            return -1
        else:
            return func(newA, newB, eps)
```

In [32]:
```
jacobiResults = solveSystemIter(A, B, eps, jacobiMethod)[2]
jacobiResults
```

Out[32]:
```
array([[0.07958228, 0.0936806 , 0.10956911, 0.02637087],
       [0.06442137, 0.07388151, 0.10129281, 0.0093399 ],
       [0.06678557, 0.07653348, 0.10364957, 0.0119309 ],
       [0.06634421, 0.07601978, 0.10330396, 0.01149752],
       [0.06641688, 0.0761065 , 0.10336633, 0.01157391],
       [0.06640411, 0.07609168, 0.10335568, 0.01156076],
       [0.0664063 , 0.07609425, 0.10335751, 0.01156301]])
```

Двосторонній характер збіжності ітераційної послідовності

Зобразимо на графіку знайдені наближення з допомогою методу Якобі та порівняємо з результатом, знайденим з допомогою функції linalg.solve

In [33]:
```
figure(num=None, figsize=(10, 8), dpi=80, facecolor='w', edgecolor='k')
plt.title('Convergence nature of the iterative sequence')
col = ['b-o', 'c-o', 'r-o', 'y-o']
plt.grid(True)
for i, j in enumerate(np.transpose(jacobiResults)):
    plt.plot([i+1 for i in range(np.transpose(jacobiResults).shape[1])], j, col[i],
             linewidth=2, label="x"+str(i+1))
plt.yticks(np.arange(0, 0.15, 0.005))
plt.xticks(np.arange(0, 8, 1))
plt.xlabel('iteration k')
plt.ylabel('approximation x^(k)')
plt.plot( np.full((4,), 7, dtype=float), progr_x, 'ks', label="linalg.solve")
plt.legend(bbox_to_anchor=(0.2, 0.5))
plt.savefig('convergence_iter_seq.jpg', dpi = 300)
plt.show()
```

3 Результати роботи

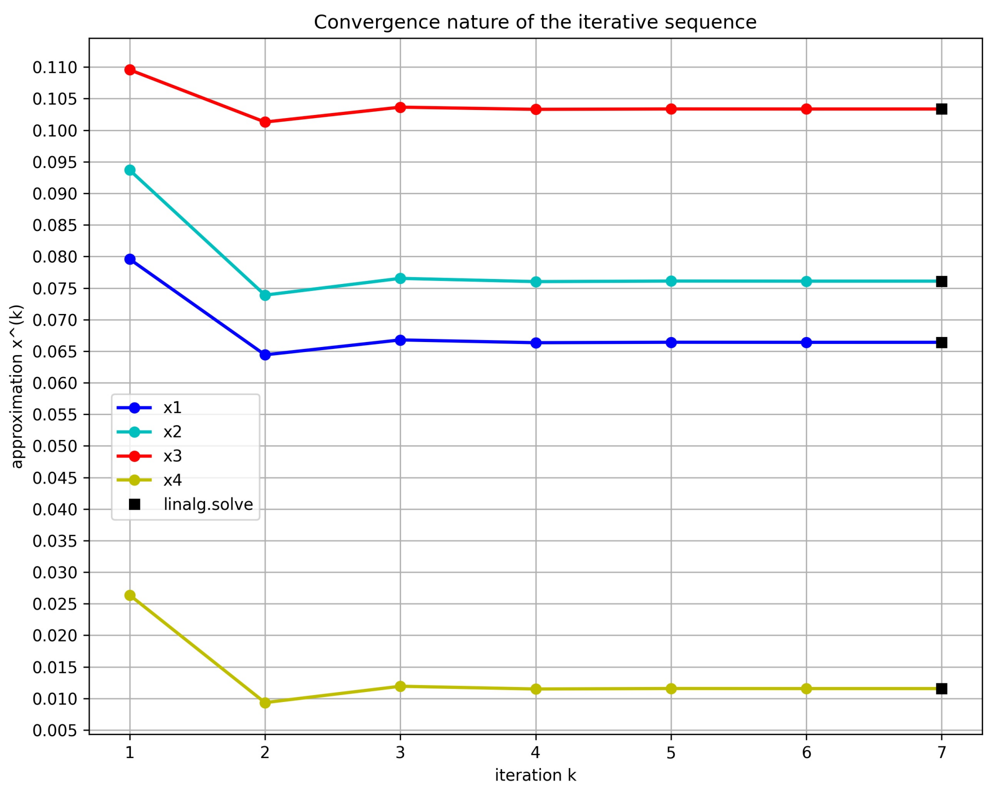
3.1 Табличне представлення:

Результати обчислень занесемо до таблиці:

	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	$\mathbf{x_4}$	$\left\Vert x^{(k)} - x^{(k-1)} \right\Vert$
1	0.07958228	0.0936806	0.10956911	0.02637087	0.1095691102
2	0.06442137	0.07388151	0.10129281	0.0093399	0.0197990880
3	0.06678557	0.07653348	0.10364957	0.0119309	0.0026519742
4	0.06634421	0.07601978	0.10330396	0.01149752	0.0005137074
5	0.06641688	0.0761065	0.10336633	0.01157391	0.0000867202
6	0.06640411	0.07609168	0.10335568	0.01156076	0.0000148216
7	0.0664063	0.07609425	0.10335751	0.01156301	0.0000025733

3.2 Графічне представлення:

Зобразимо ітераційний процес на графіку:



4 Перевірка

4.1 Текст програми:

```
In [64]: A

Out[64]: array([[5.554, 0.252, 0.496, 0.237],
                [0.58 , 4.953, 0.467, 0.028],
                [0.319, 0.372, 8.935, 0.52 ],
                [0.043, 0.459, 0.319, 4.778]])

In [50]: final_x = solveSystemIter(A, B, eps, jacobiMethod)[0]
         final_x

Out[50]: array([0.0664063 , 0.07609425, 0.10335751, 0.01156301])

In [59]: b = np.dot(A,final_x)
         b

Out[59]: array([0.44200209, 0.46400219, 0.97900282, 0.12600186])

In [60]: B

Out[60]: [0.442, 0.464, 0.979, 0.126]

In [63]: for i in (B - b):
             print("%01.10f" % i)

         -0.0000020933
         -0.0000021885
         -0.0000028241
         -0.0000018614
```

4.2 Обчислення:

$$\mathrm{A} = \begin{pmatrix} 5.554 & 0.252 & 0.496 & 0.237 \\ 0.580 & 4.953 & 0.467 & 0.028 \\ 0.319 & 0.372 & 8.935 & 0.520 \\ 0.043 & 0.459 & 0.319 & 4.778 \end{pmatrix}, \quad \mathrm{x}_* = \begin{pmatrix} 0.0664063 \\ 0.07609425 \\ 0.1033575 \\ 0.01156301 \end{pmatrix}$$

$$A \cdot x_* = \begin{pmatrix} 0.44200209 \\ 0.46400219 \\ 0.97900282 \\ 0.12600186 \end{pmatrix}$$

Вектор нев'язки :

$$\mathrm{b} - \mathrm{A} \cdot \mathrm{x}_* = \begin{pmatrix} 0.442 \\ 0.464 \\ 0.979 \\ 0.126 \end{pmatrix} - \begin{pmatrix} 0.44200209 \\ 0.46400219 \\ 0.97900282 \\ 0.12600186 \end{pmatrix} = \begin{pmatrix} -0.0000020933 \\ -0.0000021885 \\ -0.0000028241 \\ -0.0000018614 \end{pmatrix}$$

5 Інші початкові наближення

```
In [106]: def jacobiMethod(A, B, eps):
              stopCriterion, res = [], []
              x_next, x =  np.zeros(B.shape),np.zeros(B.shape)
              checkStopCriterion = lambda x, x_next, eps: chebNorm((x_next-x).reshape(B.shape[0], 1)) < eps
              while True:
                  for i, j in enumerate(A):
                      x_next[i] = B[i] - np.dot(j, np.delete(x, i))
                  stopCriterion.append(chebNorm((x_next-x).reshape(B.shape[0], 1)))
                  res.append(copy.deepcopy(x_next))
                  if checkStopCriterion(x, x_next, eps): break
                  x = copy.deepcopy(x_next)
              return x_next, np.array(stopCriterion), np.array(res)

In [111]: changedB = np.array([0.5,0.5,0.5,0.5])
          changedB

Out[111]: array([0.5, 0.5, 0.5, 0.5])

In [112]: jacobiMethod(newA, changedB, eps)

Out[112]: (array([0.42375527, 0.4061844 , 0.44307496, 0.42758415]),
           array([5.00000000e-01, 1.08520089e-01, 1.72590946e-02, 2.99954112e-03,
                  5.21390881e-04, 8.92063901e-05, 1.54005182e-05, 2.65239743e-06]),
           array([[0.5       , 0.5       , 0.5       , 0.5       ],
                  [0.41132517, 0.39147991, 0.43223279, 0.41408539],
                  [0.42596712, 0.40873901, 0.44491688, 0.42983288],
                  [0.4233793 , 0.40573946, 0.44275909, 0.42719626],
                  [0.4238206 , 0.40626086, 0.44312981, 0.42765176],
                  [0.4237444 , 0.40617165, 0.44306584, 0.42757295],
                  [0.42375753, 0.40618705, 0.44307686, 0.42758648],
                  [0.42375527, 0.4061844 , 0.44307496, 0.42758415]]))
```

Бачимо, що при зміні даних у стовпчику вільних членів(B), кількість ітерацій змінюється

6 Розв'язок системи з допомогою спеціальної функції linalg.solve

```
In [2]: import numpy as np
        import copy
        import matplotlib.pyplot as plt
        from matplotlib.pyplot import figure
        import handcalcs.render

In [3]: A = np.array([[5.554, 0.252, 0.496, 0.237],
        [0.580, 4.953, 0.467, 0.028],
        [0.319, 0.372, 8.935, 0.520],
        [0.043, 0.459, 0.319, 4.778]])
        A

Out[3]: array([[5.554, 0.252, 0.496, 0.237],
               [0.58 , 4.953, 0.467, 0.028],
               [0.319, 0.372, 8.935, 0.52 ],
               [0.043, 0.459, 0.319, 4.778]])

In [4]: B = [0.442, 0.464, 0.979, 0.126]
        eps = 0.00001

In [5]: progr_x = np.linalg.solve(np.array(A), np.array(B))
        progr_x

Out[5]: array([0.06640598, 0.07609387, 0.10335724, 0.01156268])

In [6]: np.allclose(np.dot(np.array(A), progr_x), np.array(B))

Out[6]: True
```

Отримали результат: $x_* = \begin{pmatrix} 0.06640598 \\ 0.07609387 \\ 0.10335724 \\ 0.01156268 \end{pmatrix}$

Даний розв'язок також позначений на графіку вище

7 Висновки

Отримали розв'язок системи за допомогою методу Якобі: $x_* = \begin{pmatrix} 0.0664063 \\ 0.07609425 \\ 0.1033575 \\ 0.01156301 \end{pmatrix}$

Записали результати до таблиці та прослідкували за ітераційним процесом з допомогою графіка. З'ясували, що при зміні даних у стовпчику вільних членів(B), кількість ітерацій також змінюється. Робота дала змогу попрактикуватися в застосовуванні ітераційних чисельних методів (конкретно методу Якобі) для розв'язування СЛАР.