Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

Комп'ютерний практикум №4
Обчислення власних значень

предмет «Методи обчислень»

Роботу виконав:
Студент 3 курсу ФТІ, групи ФІ-91
Цибульник Антон Владиславович

Приймала:
Стьопочкіна Ірина Валеріївна

2022

Завдання

Завдання лабораторної роботи полягає у пошуку власних чисел деякої заданої матриці A. До цього ми знайомилися із способом знаходження цих чисел з рівняння $\det(A - \lambda I) = 0$, що є обчислювано нераціональним. Тож наразі виконаємо необхідні розрахунки методом Якобі: приведемо вихідну матрицю за допомогою подібних обертань до майже діагонального вигляду (сума модулів недіагональних елементів має дорівнювати нулю із точністю $\varepsilon = 10^{-5}$), і тоді на головній діагоналі будуть міститись наближені значення власних чисел.

Варіант завдання

Задана така симетрична матриця A :

$$A = \begin{pmatrix} 7 & 0.88 & 0.93 & 1.21 \\ 0.88 & 4.16 & 1.3 & 0.15 \\ 0.93 & 1.3 & 6.44 & 2 \\ 1.21 & 0.15 & 2 & 6.1 \end{pmatrix}$$

Теоретичні відомості

Метод Якобі діє для симетричних матриць: ітераційно виконуватимемо процес обертання, починаючи з матриці A, й пильнуватимемо момент, коли виконуватиметься умова завершення. Обертанням будемо називати перетворення координат за допомогою матриці T_{ij} виду:

$$T_{ij} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & t_{ii} & \dots & t_{ji} & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & t_{ij} & \dots & t_{jj} & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}, \text{ де } \begin{cases} t_{ii} = t_{jj} = c \\ t_{ij} = \text{-}s, \ t_{ji} = s \end{cases}$$

При цьому для цієї матриці її обернена рівна транспонованій: $T^T = T^{-1}$. Числа c та s знаходимо із таких умов:

$$\mu = \frac{2a_{ij}}{a_{ii} - a_{jj}}, \ c = \sqrt{\frac{1}{2}\left(1 + \frac{1}{\sqrt{1+\mu^2}}\right)}, \ s = sign(\mu)\sqrt{\frac{1}{2}\left(1 - \frac{1}{\sqrt{1+\mu^2}}\right)}$$

Відповідно основна ітерація методу виглядатиме як перетворення виду

$$A_{k+1} = T_{ij}^T A_k T_{ij}$$

доти, доки A_{k+1}-ша матриця не буде задовільняти умові майже діагонального вигляду (сума модулів недіагональних елементів майже рівна нулю).

Залишається питання ідексів i, j: для кожної ітерації ці індекси визначатимуться як індекси максимального по модулю недіагонального елемента поточної матриці A_k, при цьому сам елемент a_{ij} кожного разу слід обнуляти. Для моніторингу поточної ситуації на кожній ітерації введемо величини:

$$S_A = \sum_{i,j=1}^{n} a_{ij}^2 \qquad \text{сферична норма матриці}$$

$$S_d = \sum_{i=1}^{n} a_{ii}^2 \qquad \text{діагональна частина сферичної норми}$$

$$S_{nd} = \sum_{i,j=1, i \neq j}^{n} a_{ij}^2 \qquad \text{недіагональна частина сферичної норми}$$

Тож від ітерації до ітерації S_A має лишатися сталою, S_{nd} – зменшуватися, а S_d, відповідно, збільшуватися.

Програмний код

```
import numpy as np

A = np.array([[7.0, 0.88, 0.93, 1.21],
              [0.88, 4.16, 1.3, 0.15],
              [0.93, 1.3, 6.44, 2.0],
              [1.21, 0.15, 2.0, 9.0]])

length = A[0].size

def find_ij(A):
    max = -1000

    for i in range(length):
        for j in range(length):
            if (abs(A[i][j]) >= max and i != j):
                max = abs(A[i][j])
                index_i = i
                index_j = j

    return index_i, index_j

def count_S(A):
    S = 0
    for i in range(length):
        for j in range(length):
            if (i != j): S = S + abs(A[i][j])

    return S



```

```
def count_Sa(A):
    Sa = 0
    for i in range(length):
        for j in range(length):
            Sa = Sa + A[i][j]*A[i][j]

    return Sa

def count_Snd(A):
    Snd = 0
    for i in range(length):
        for j in range(length):
            if (i != j): Snd = Snd + A[i][j]*A[i][j]

    return Snd

def count_Sd(A):
    Sd = 0
    for i in range(length):
        Sd = Sd + A[i][i]*A[i][i]

    return Sd

e = 0.00001
stop = False
while(stop != True):
    i,j = find_ij(A)
    m = (2*A[i][j])/(A[i][i] - A[j][j])
    c = np.sqrt(0.5 * (1 + 1/np.sqrt(1 + m*m)))
    s = np.sign(m)*np.sqrt(0.5 * (1 - 1/np.sqrt(1 + m*m)))

    T = np.array([[1.0, 0.0, 0.0, 0.0],
                  [0.0, 1.0, 0.0, 0.0],
                  [0.0, 0.0, 1.0, 0.0],
                  [0.0, 0.0, 0.0, 1.0]])

    T[i][j] = -s
    T[j][i] = s
    T[i][i] = T[j][j] = c

    T_t = T.transpose()

    A = np.dot(T_t,A)
    A = np.dot(A,T)
    A[i][j] = 0

    S = count_S(A)
    if (S <= e): stop = True
```

Результати й проміжні кроки

```
Iteration 1:

    Matrix T
    [[ 1.         0.         0.         0.        ]
     [ 0.         1.         0.         0.        ]
     [ 0.         0.         0.87722679 0.48007619]
     [ 0.         0.        -0.48007619 0.87722679]]

    Matrix T'
    [[ 1.         0.         0.         0.        ]
     [ 0.         1.         0.         0.        ]
     [ 0.         0.         0.87722679 -0.48007619]
     [ 0.         0.         0.48007619 0.87722679]]

    Sa = 206.41099999999997, Snd = 9.631799999999998, Sd = 196.7792
    Value of S = 8.8938209584238

Iteration 2:

    Matrix T
    [[ 0.92632469 0.         0.         0.37672612]
     [ 0.         1.         0.         0.        ]
     [ 0.         0.         1.         0.        ]
     [-0.37672612 0.         0.         0.92632469]]

    Matrix T'
    [[ 0.92632469 0.         0.        -0.37672612]
     [ 0.         1.         0.         0.        ]
     [ 0.         0.         1.         0.        ]
     [ 0.37672612 0.         0.         0.92632469]]

    Sa = 206.411, Snd = 5.084183010774322, Sd = 201.32681698922568
    Value of S = 5.8730290283892534

Iteration 3:

    Matrix T
    [[ 1.         0.         0.         0.        ]
     [ 0.         0.86172354 0.5073781  0.        ]
     [ 0.        -0.5073781  0.86172354 0.        ]
     [ 0.         0.         0.         1.        ]]

    Matrix T'
    [[ 1.         0.         0.         0.        ]
     [ 0.         0.86172354 -0.5073781 0.        ]
     [ 0.         0.5073781  0.86172354 0.        ]
     [ 0.         0.         0.         1.        ]]

    Sa = 206.411, Snd = 2.801296818246346, Sd = 203.60970318175364
    Value of S = 4.494040159643841
```

```
Iteration 4:

    Matrix T
    [[ 1.          0.          0.          0.        ]
     [ 0.          0.99333736  0.          0.11524272]
     [ 0.          0.          1.          0.        ]
     [ 0.         -0.11524272  0.          0.99333736]]

    Matrix T'
    [[ 1.          0.          0.          0.        ]
     [ 0.          0.99333736  0.         -0.11524272]
     [ 0.          0.          1.          0.        ]
     [ 0.          0.11524272  0.          0.99333736]]

    Sa = 206.411, Snd = 1.3766717130506674, Sd = 205.03432828694932
    Value of S = 3.0115777795057697

Iteration 5:

    Matrix T
    [[ 1.          0.          0.          0.        ]
     [ 0.          1.          0.          0.        ]
     [ 0.          0.          0.99270285  0.1205863 ]
     [ 0.          0.         -0.1205863   0.99270285]]

    Matrix T'
    [[ 1.          0.          0.          0.        ]
     [ 0.          1.          0.          0.        ]
     [ 0.          0.          0.99270285 -0.1205863 ]
     [ 0.          0.          0.1205863   0.99270285]]

    Sa = 206.411, Snd = 0.6670863655113755, Sd = 205.74391363448862
    Value of S = 1.92920070120573

Iteration 6:

    Matrix T
    [[ 0.85885707  0.         -0.51221532  0.        ]
     [ 0.          1.          0.          0.        ]
     [ 0.51221532  0.          0.85885707  0.        ]
     [ 0.          0.          0.          1.        ]]

    Matrix T'
    [[ 0.85885707  0.          0.51221532  0.        ]
     [ 0.          1.          0.          0.        ]
     [-0.51221532  0.          0.85885707  0.        ]
     [ 0.          0.          0.          1.        ]]

    Sa = 206.411, Snd = 0.26472461148438275, Sd = 206.14627538851562
    Value of S = 1.268398439619539
```

```
Iteration 7:

    Matrix T
    [[ 0.99678393 -0.08013611 0.         0.        ]
     [ 0.08013611 0.99678393 0.         0.        ]
     [ 0.         0.         1.         0.        ]
     [ 0.         0.         0.         1.        ]]

    Matrix T'
    [[ 0.99678393 0.08013611 0.         0.        ]
     [-0.08013611 0.99678393 0.         0.        ]
     [ 0.         0.         1.         0.        ]
     [ 0.         0.         0.         1.        ]]

    Sa = 206.411, Snd = 0.12884386678223536, Sd = 206.28215613321774
    Value of S = 0.7944211503429432

Iteration 8:

    Matrix T
    [[ 1.         0.         0.         0.        ]
     [ 0.         0.99459572 -0.10382364 0.        ]
     [ 0.         0.10382364 0.99459572 0.        ]
     [ 0.         0.         0.         1.        ]]

    Matrix T'
    [[ 1.         0.         0.         0.        ]
     [ 0.         0.99459572 0.10382364 0.        ]
     [ 0.        -0.10382364 0.99459572 0.        ]
     [ 0.         0.         0.         1.        ]]

    Sa = 206.411, Snd = 0.018799217402917574, Sd = 206.39220078259706
    Value of S = 0.335308936475719
```

Наведено результати й проміжні кроки для перших $k = 8$ ітерацій. Назагал, бажана точність на суму модулів недіагональних елементів досягнута при $k = 15$ ітераціях, і отриманий результат рівний:

$$A^* = \begin{pmatrix} 6.67397938 & 0 & 0 & 0 \\ 0 & 3.38753165 & 0 & 0 \\ 0 & 0 & 5.65841896 & 0 \\ 0 & 0 & 0 & 10.88007000 \end{pmatrix}$$

Таким чином знайдені власні числа рівні:

$$\lambda_1 = 6.67397938,\ \lambda_2 = 3.38753165,\ \lambda_3 = 5.65841896,\ \lambda_4 = 10.88007$$

Скориставшись онлайн сайтами знаходження власних чисел, матимемо аналогічні результати.

Контрольні запитання

1. *Які рядки та стовпці матриці A^{k+1} змінює процес обертання T_{ij} порівняно із вихідною матрицею A в методі Якобі?*

   Якщо над матрицею A^k провести процес обертання $A^{k+1} = T_{ij}^T A^k T_{ij} \equiv T_{ij}^T B^k$, то лише рядки та стовпці i та j відповідних матриць будуть змінюватися таким чином:

   $$B_i^k = cA_i^k + sA_j^k \qquad A_i^{k+1} = cB_i^k + sB_j^k$$
   $$B_j^k = -sA_i^k + cA_j^k \qquad A_j^{k+1} = -sB_i^k + cB_j^k$$

2. *Як за методом Якобі визначити власні вектори матриці A за результуючою матрицею A^*, знаючи сукупність перетворень $\{T_{ij}\}$?*

   Довільну симетричну матрицю A можна представити у вигляді $A = TA^*T^{-1}$, де A^* – матриця, складена з власних значень, а T – матриця, що складається із власних векторів.

   У методі Якобі, провівши, скажімо, n штук ітерації виду $A_{k+1} = T_{ij}^T A_k T_{ij}$ над початковою матрицею A віднайшли шукану матрицю власних чисел A^*. Тобто з одного боку

   $$A^* = \underbrace{{}^nT_{ij}^T \cdot {}^{n-1}T_{ij}^T \ldots \cdot {}^1T_{ij}^T}\cdot A \cdot \underbrace{{}^1T_{ij} \cdot \ldots {}^{n-1}T_{ij} \cdot {}^nT_{ij}}$$

   а з іншого

   $$A = TA^*T^{-1} \Rightarrow A^* = T^{-1}AT$$

   Отже, порівнявши ці два представлення матимемо, що шукана матриця власних векторів T із відомою сукупністю перетворень $\{T_{ij}\}$ знаходитиметься так:

   $$T = {}^1T_{ij} \cdot \ldots {}^{n-1}T_{ij} \cdot {}^nT_{ij}$$

3. *Які елементи матриці будуть зменшуватися при обертаннях за методом Якобі, а які будуть збільшуватись?*

   Як вже зазначалося, ввівши величини

   $$S_A = \sum_{i,j=1}^{n} a_{ij}^2, \ S_d = \sum_{i=1}^{n} a_{ii}^2, \ S_{nd} = \sum_{i,j=1, i\neq j}^{n} a_{ij}^2$$

   від ітерації до ітерації S_A має лишатися сталою, сума квадратів недіагональних елементів S_{nd} – зменшуватися, а сума квадратів діагональних елементів S_d, відповідно, збільшуватися.

4. *Коли метод Данилевського неможливо застосувати?*

   Вказаний метод неможливо застосувати тоді, коли при формуванні матриць M_k виникає ділення на нульові коефіцієнти поточної матриці A_k.

5. *У яких випадках система характеристичного поліному із коефіцієнтами p_i буде виродженою?*

   Система характеристичного поліному із коефіцієнтами p_i буде виродженою тоді, коли матриця A має кратні класні власні числа. Тобто у такому разі зазначена система є лінійно залежною, що призводить до обнулення визначника матриці коефіцієнтів системи.