Лабораторна робота №2

Методи визначення векторів ваг альтернатив на основі експертних даних

Галета М.С., КМ-91мп

In [1]:
```
import numpy as np
from scipy.stats.mstats import gmean     # Функція для обчислення середнього геометричного
from itertools import combinations       # Функція для генерування комбінацій
```

Введення думок експертів та їх компетентності

In [2]:

```python
def input_experts_data(mpp_dimension, number_of_experts):
    # mpp_dimension (int) - Розмірність матриці мультиплікативних парних порівнянь
    # number_of_experts (int) - Кількість експертів

    # Ініціалізація матриці парних порівнянь для кожного експерта
    mpp = np.array(list(map(np.eye, number_of_experts*[mpp_dimension])))

    for k in range(number_of_experts):
        for i in range(mpp_dimension):
            for j in range(i+1, mpp_dimension, 1):
                try:
                    value = float(input(f"Введіть елемент [{i+1},{j+1}] матриці парних порівнянь для експерта {k+1}:
                    if value <= 0 or value > 9:
                        raise
                    mpp[k][i][j], mpp[k][j][i] = value, 1/value
                except Exception:
                    mpp[k][i][j] = mpp[k][j][i] = np.nan

        print()


    # Ініціалізація вектору коефіцієнтів компетентності експертів
    comp_coeffs = np.zeros(number_of_experts)

    for k in range(number_of_experts):
        comp_coeffs[k] = float(input(f"Введіть коефіцієнт компетентності експерта {k+1}: "))

    # Нормалізація коефіцієнтів компетентності, щоб їх сума  дорівнювала 1
    comp_coeffs /= np.sum(comp_coeffs)

    return mpp, comp_coeffs
```

In [3]:
```
mpp_dimension = 5
number_of_experts = 3
mpp, comp_coeffs = input_experts_data(mpp_dimension, number_of_experts)
```

```
Введіть елемент [1,2] матриці парних порівнянь для експерта 1: 3
Введіть елемент [1,3] матриці парних порівнянь для експерта 1: 5
Введіть елемент [1,4] матриці парних порівнянь для експерта 1: 5
Введіть елемент [1,5] матриці парних порівнянь для експерта 1: 7
Введіть елемент [2,3] матриці парних порівнянь для експерта 1: 3
Введіть елемент [2,4] матриці парних порівнянь для експерта 1: 3
Введіть елемент [2,5] матриці парних порівнянь для експерта 1: 5
Введіть елемент [3,4] матриці парних порівнянь для експерта 1: 3
Введіть елемент [3,5] матриці парних порівнянь для експерта 1: 1
Введіть елемент [4,5] матриці парних порівнянь для експерта 1: 0.33

Введіть елемент [1,2] матриці парних порівнянь для експерта 2: 3
Введіть елемент [1,3] матриці парних порівнянь для експерта 2: 0.2
Введіть елемент [1,4] матриці парних порівнянь для експерта 2:
Введіть елемент [1,5] матриці парних порівнянь для експерта 2: 5
Введіть елемент [2,3] матриці парних порівнянь для експерта 2: 0.2
Введіть елемент [2,4] матриці парних порівнянь для експерта 2: 0.2
Введіть елемент [2,5] матриці парних порівнянь для експерта 2:
Введіть елемент [3,4] матриці парних порівнянь для експерта 2: 1
Введіть елемент [3,5] матриці парних порівнянь для експерта 2: 7
Введіть елемент [4,5] матриці парних порівнянь для експерта 2: 7

Введіть елемент [1,2] матриці парних порівнянь для експерта 3: 0.33
Введіть елемент [1,3] матриці парних порівнянь для експерта 3: 3
Введіть елемент [1,4] матриці парних порівнянь для експерта 3: 0.33
Введіть елемент [1,5] матриці парних порівнянь для експерта 3: 1
Введіть елемент [2,3] матриці парних порівнянь для експерта 3: 3
Введіть елемент [2,4] матриці парних порівнянь для експерта 3: 1
Введіть елемент [2,5] матриці парних порівнянь для експерта 3: 3
Введіть елемент [3,4] матриці парних порівнянь для експерта 3: 0.33
Введіть елемент [3,5] матриці парних порівнянь для експерта 3: 0.33
Введіть елемент [4,5] матриці парних порівнянь для експерта 3: 3

Введіть коефіцієнт компетентності експерта 1: 0.5
Введіть коефіцієнт компетентності експерта 2: 0.2
Введіть коефіцієнт компетентності експерта 3: 0.3
```

In [4]:
```
print("Матриці парних порівнянь для кожного експерта:")
for i in range(number_of_experts):
    print(mpp[i])
print("\nКоефіцієнти компетентності експертів:")
print(comp_coeffs)
```

```
Матриці парних порівнянь для кожного експерта:
[[1.         3.         5.         5.         7.        ]
 [0.33333333 1.         3.         3.         5.        ]
 [0.2        0.33333333 1.         3.         1.        ]
 [0.2        0.33333333 0.33333333 1.         0.33      ]
 [0.14285714 0.2        1.         3.03030303 1.        ]]
[[1.         3.         0.2               nan 5.        ]
 [0.33333333 1.         0.2        0.2               nan]
 [5.         5.         1.         1.         7.        ]
 [       nan 5.         1.         1.         7.        ]
 [0.2               nan 0.14285714 0.14285714 1.        ]]
[[1.         0.33       3.         0.33       1.        ]
 [3.03030303 1.         3.         1.         3.        ]
 [0.33333333 0.33333333 1.         0.33       0.33      ]
 [3.03030303 1.         3.03030303 1.         3.        ]
 [1.         0.33333333 3.03030303 0.33333333 1.        ]]

Коефіцієнти компетентності експертів:
[0.5 0.2 0.3]
```

In [5]:

```
class ExpertsDataAnalysis:
    def __init__(self, mpps, coeffs):
        self.mpps = mpps
        self.coeffs = coeffs
        self.n_experts = mpps.shape[0]
        self.dim = mpps.shape[1]
        self.weights = None
        self.consistency = None

    def __call__(self, method, exclude_nan=True):
        if method == 'eigen_vector':
            return self.eigen_vector_method(exclude_nan)
        elif method == 'combinatorial':
            return self.combinatorial_method()

    def check_consistency(self):
        geom_means = np.array(list(map(lambda x: len(x)*[np.nan] if np.isnan(x).any() else gmean(x, axis=1), self.mp
        self.weights = np.array(list(map(lambda x: x/np.sum(x), geom_means)))
        lambda_max = np.diag(self.weights @ np.array(list(map(lambda x: np.sum(x, axis=0), self.mpps))).T)

        cons_index = (lambda_max-self.dim)/(self.dim-1)
        self.consistency = list(map(lambda x: np.nan if np.isnan(x) else True if x < 0.1 else False, cons_index))

    def eigen_vector_method(self, exclude_nan):
        if self.weights is None:
            self.check_consistency()

        if exclude_nan:
            weighted_gmean = np.prod(
                list(
                    map(lambda x, y: np.nan_to_num(x, nan=1)**y, self.weights, self.coeffs)
                ), axis=0
            )**(1/sum(self.coeffs))
        else:
            weighted_gmean = np.prod(list(map(lambda x, y: x**y, self.weights, self.coeffs)), axis=0)**(1/sum(self.c

        weighted_gmean /= np.sum(weighted_gmean)
        return weighted_gmean

    def get_ideally_consistent_mpps(self, mpp):
        indices = np.triu_indices(self.dim, k=1)
```

```
        indices = np.array(indices).T[np.where(~np.isnan(mpp[indices]))].tolist()

        combs = list(combinations(indices, self.dim - 1))

        ideal_mpps = np.array(list(map(lambda x: np.eye(self.dim), range(len(combs)))))
        for matrix, comb in zip(range(len(ideal_mpps)), combs):
            comb_inv = tuple(np.array(comb).T)
            ideal_mpps[matrix][comb_inv] = mpp[comb_inv]
            ideal_mpps[matrix].T[comb_inv] = mpp.T[comb_inv]

            zero_idcs = np.array(np.where(ideal_mpps[matrix] == 0)).T.tolist()

            excluded = []
            for i,j in zero_idcs:
                if [i,j] in excluded:
                    continue
                for k in range(self.dim):
                    if ideal_mpps[matrix][i,k] !=0 and ideal_mpps[matrix][k,j] !=0:
                        ideal_mpps[matrix][i,j] = ideal_mpps[matrix][i,k]*ideal_mpps[matrix][k,j]
                        ideal_mpps[matrix][j,i] = 1/ideal_mpps[matrix][i,j]
                        excluded.append([[i,j], [j,i]])
                        break

        ideal_mpps = ideal_mpps[np.where(np.all(np.all(np.array(ideal_mpps) != 0, axis=1), axis=1))[0]]

        return ideal_mpps

    def combinatorial_method(self):
        ideal_mpps = np.array(list(map(self.get_ideally_consistent_mpps, self.mpps)))

        weights = np.concatenate(np.array(list(map(lambda x: gmean(x, axis=2), ideal_mpps))), axis=0)
        coeffs = np.concatenate(list(map(lambda i: len(ideal_mpps[i])*[self.coeffs[i]], range(len(self.coeffs)))), a

        weighted_gmean = np.prod(list(map(lambda x, y: x**y, weights, coeffs)), axis=0)**(1/sum(coeffs))
        weighted_gmean /= np.sum(weighted_gmean)

        return weighted_gmean
```

In [6]:

```
eda = ExpertsDataAnalysis(mpp, comp_coeffs)
eda.check_consistency()
```

In [7]:
```
for i, cons in enumerate(eda.consistency):
    if np.isnan(cons):
        print(f"Матриця попарних порівнянь експерта {i+1} неповна")
        continue
    if cons:
        print(f"Матриця попарних порівнянь експерта {i+1} узгоджена")
    else:
        print(f"Матриця попарних порівнянь експерта {i+1} неузгоджена")
```

```
Матриця попарних порівнянь експерта 1 узгоджена
Матриця попарних порівнянь експерта 2 неповна
Матриця попарних порівнянь експерта 3 узгоджена
```

Метод власного вектора

In [8]:
```
weights = eda("eigen_vector", True)
weights
```

Out[8]:
```
array([0.31915841, 0.29167168, 0.11918224, 0.13616866, 0.13381902])
```

Комбінаторний метод

In [9]:
```
weights = eda("combinatorial")
weights
```

Out[9]:
```
array([0.34127018, 0.28750165, 0.11942226, 0.13827555, 0.11353036])
```

Можна побачити, що ваги отримані двома методами, відрізняються між собою
Ваги критеріїв 3 та 5 сприймаються майже рівними в комбінаторному методі, однак вони відрізняються, якщо обчислювати методом власного вектора

Тестування на повних матрицях

In [10]:
```
number_of_experts = 3
mpp_dimension = 5

mpp1 = np.array([
    [1, 3, 5, 5, 7],
    [1/3, 1, 3, 3, 5],
    [1/5, 1/3, 1, 3, 1],
    [1/5, 1/3, 1/3, 1, 1/3],
    [1/7, 1/5, 1, 3, 1]
])

mpp2 = np.array([
    [1, 3, 1/5, 1/5, 5],
    [1/3, 1, 1/5, 1/5, 3],
    [5, 5, 1, 1, 7],
    [5, 5, 1, 1, 7],
    [1/5, 1/3, 1/7, 1/7, 1]
])

mpp3 = np.array([
    [1, 1/3, 3, 1/3, 1],
    [3, 1, 3, 1, 3],
    [1/3, 1/3, 1, 1/3, 1/3],
    [3, 1, 3, 1, 3],
    [1, 1/3, 3, 1/3, 1]
])

coeffs = np.array([0.5, 0.2, 0.3])

mpps = np.array([mpp1, mpp2, mpp3])
```

In [11]:
```
eda = ExpertsDataAnalysis(mpps, coeffs)
eda.check_consistency()
```

In [12]:
```
for i, cons in enumerate(eda.consistency):
    if np.isnan(cons):
        print(f"Матриця попарних порівнянь експерта {i+1} неповна")
        continue
    if cons:
        print(f"Матриця попарних порівнянь експерта {i+1} узгоджена")
    else:
        print(f"Матриця попарних порівнянь експерта {i+1} неузгоджена")
```

```
Матриця попарних порівнянь експерта 1 узгоджена
Матриця попарних порівнянь експерта 2 узгоджена
Матриця попарних порівнянь експерта 3 узгоджена
```

In [13]:
```
weights = eda("eigen_vector", True)
weights
```

Out[13]:
```
array([0.31750742, 0.25990373, 0.14878344, 0.16974975, 0.10405566])
```

In [14]:
```
weights = eda("combinatorial")
weights
```

Out[14]:
```
array([0.31681219, 0.25969621, 0.14889193, 0.17092557, 0.10367411])
```

При повних матрицях ваги критеріїв обчислені двома методами різняться лише чисельно, при точності < 0.01.