华东师范大学软件学院课程作业

课程名称：软件质量分析	年级：2023 级本科	姓名：张梓卫
作业主题：软件可信性的度量值计算	学号：10235101526	作业日期：2024/11/05
指导老师：陈仪香	组号：	

目录

一 作业三

1 题目内容

有 4 个软件系统，分别编号为 1,2,3,4。它们都有可靠性属性 y，含有 5 个子属性，编号为 x_1, x_2, x_3, x_4, x_5，其权重分别为 $\gamma_1(\omega_1) = 0.15$, $\gamma_2(\omega_2) = 0.20$, $\gamma_3(\omega_3) = 0.20$, $\gamma_4(\omega_4) = 0.25$, $\gamma_5(\omega_5) = 0.20$。参数 $\rho_y = 0.01, 0.55$。按照模型 1 (y_1) 和模型 2 (y_2) 分别计算可靠性属性 y 的可信度量值，注意：模型 1 是不需要参数 ρ_y 的。

编号	x_1	x_2	x_3	x_4	x_5	ρ_y	y_1	y_2
1	8.6	9.1	9.2	8.8	8.9	0.01		
	8.6	9.1	9.2	8.8	8.9	0.55		
2	6.8	7.9	5.9	6.6	6.1	0.01		
	6.8	7.9	5.9	6.6	6.1	0.55		
3	9.1	9.9	8.9	8.8	7.8	0.01		
	9.1	9.9	8.9	8.8	7.8	0.55		
4	3.5	4.2	5.6	4.9	5.2	0.01		
	3.5	4.2	5.6	4.9	5.2	0.55		

2 题目分析

2 .1 模型一

根据模型一的公式： $y = x_1^{\gamma_1} x_2^{\gamma_2} \cdots x_n^{\gamma_n}$

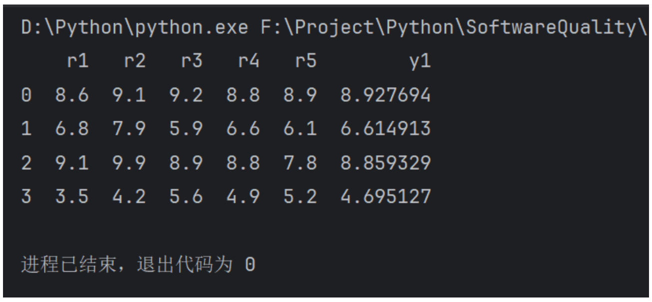
以及上一次作业所写的代码，我们可以稍加改动后对 y_1 进行计算：

```
import pandas as pd
import numpy as np

# weights = {
#     'F': 0.25,'SF': 0.15,'R': 0.20,'SV': 0.23,'M': 0.17
# }
# 将权重修改为本题的内容:
weights = {
    'r1': 0.15,'r2': 0.20,'r3': 0.20,'r4': 0.25,'r5': 0.20
}

# 将数据改为作业三的数据
data = [
    [8.6, 9.1, 9.2, 8.8, 8.9],
    [6.8, 7.9, 5.9, 6.6, 6.1],
    [9.1, 9.9, 8.9, 8.8, 7.8],
    [3.5, 4.2, 5.6, 4.9, 5.2]
]

def main():
    dataframe = pd.DataFrame(data, columns=['r1', 'r2', 'r3', 'r4', 'r5'])
    dataframe['T'] = dataframe.apply(lambda
    row: np.prod([row[attr] ** weights[attr] for attr in weights]), axis=1)
    print(dataframe)

if __name__ == '__main__':
    main()
```

计算软件可信性度量值

2 .2 模型一输出结果

运行结果如下图所示：



```
D:\Python\python.exe F:\Project\Python\SoftwareQuality\
    r1   r2   r3   r4   r5        y1
0  8.6  9.1  9.2  8.8  8.9  8.927694
1  6.8  7.9  5.9  6.6  6.1  6.614913
2  9.1  9.9  8.9  8.8  7.8  8.859329
3  3.5  4.2  5.6  4.9  5.2  4.695127

进程已结束，退出代码为 0
```

2 .3 模型二

以下为模型二的公式：

$$y = \left(\sum_{i=1}^{n} \omega_i x_i^{-\rho_y} \right)^{-\frac{1}{\rho_y}}, \quad 1 \leq i \leq n, \quad 1 \leq x_i \leq 10.$$

这与上一次作业中的形式也有一些相近，我们同样先进行 DataFrame 的构造：

由于 PPT 中有 y_1 和 y_2 的示例结果，我们可以先试着跑一个例子：

```
test_data = {
    'x1': [8, 8, 6, 6, 9, 9, 3, 3],
    'x2': [9, 9, 7, 7, 10, 10, 4, 4],
    'x3': [10, 10, 5, 5, 9, 9, 5, 5],
    'x4': [8, 8, 6, 6, 8, 8, 4, 4],
    'rho_y': [0.10, 0.20, 0.10, 0.20, 0.10, 0.20, 0.10, 0.20]
}
test_weights = np.array([0.25, 0.20, 0.25, 0.30])
```

```python
df = pd.DataFrame(test_data)
```

构造示例参数

```python
def calculate_y2(row):
  x_values = np.array([row['x1'], row['x2'], row['x3'], row['x4']])
  rho_y = row['rho_y']
  summation = np.sum(test_weights * (x_values ** -rho_y))
  y2 = summation ** (-1 / rho_y)
  return y2
```

计算 y_2 的值

```python
def main():
  df['y2'] = df.apply(calculate_y2, axis=1)
  print(df)

if __name__ == "__main__":
  main()
```

运行结果如下所示：

```
D:\Python\python.exe F:\Project\Python\SoftwareQuality\
   x1  x2  x3  x4  rho_y        y2
0   8   9  10   8    0.1  8.656742
1   8   9  10   8    0.2  8.652900
2   6   7   5   6    0.1  5.908350
3   6   7   5   6    0.2  5.904551
4   9  10   9   8    0.1  8.869805
5   9  10   9   8    0.2  8.867066
6   3   4   5   4    0.1  3.929504
7   3   4   5   4    0.2  3.923020

进程已结束，退出代码为 0
```

(a) 示例输出结果

编号	x_1	x_2	x_3	x_4	ρ_y	y_1	y_2
1	8	9	10	8	0.10	8.66	8.66
	8	9	10	8	0.20		8.65
2	6	7	5	6	0.10	5.91	5. 90
	6	7	5	6	0.20		5.90
3	9	10	9	8	0.10	8.87	8.87
	9	10	9	8	0.20		8.87
4	3	4	5	4	0.10	3.94	3.93
	3	4	5	4	0.10		3.92

(b) PPT 上的示例结果对比

图 1: 输出结果和 PPT 对比

可以发现，该代码成功计算了 y_2 的值，与 PPT 中的示例结果基本一致。

【Notice】：PPT 编号 2 上 $\rho_y = 0.10$ 时，y_2 的值应约为 5.91，而非 5.90

接下来我们只需要把数据换成新的题目中的即可。

```python
weights = np.array([0.15, 0.20, 0.20, 0.25, 0.20])
data = {
  'x1': [8.6, 8.6, 6.8, 6.8, 9.1, 9.1, 3.5, 3.5],
  'x2': [9.1, 9.1, 7.9, 7.9, 9.9, 9.9, 4.2, 4.2],
  'x3': [9.2, 9.2, 5.9, 5.9, 8.9, 8.9, 5.6, 5.6],
  'x4': [8.8, 8.8, 6.6, 6.6, 8.8, 8.8, 4.9, 4.9],
  'x5': [8.9, 9.9, 6.1, 6.1, 7.8, 7.8, 5.2, 5.2],
  'rho_y': [0.01, 0.55, 0.01, 0.55, 0.01, 0.55, 0.01, 0.55]
}
df = pd.DataFrame(data)
```

后续只需要把对应的地方替换掉即可（test_data 改为 data，test_weights 改为 weights，注意还有 x_5 的添加）

完整可运行代码如下所示：

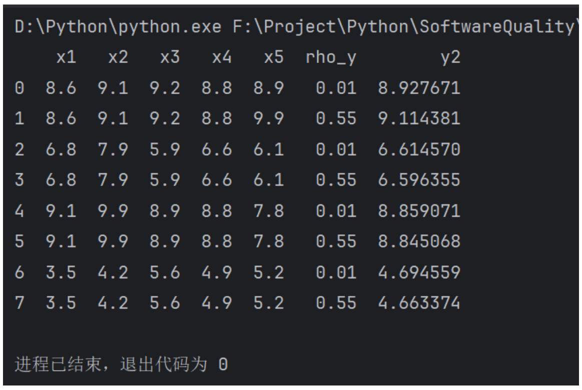
```python
import numpy as np
import pandas as pd

weights = np.array([0.15, 0.20, 0.20, 0.25, 0.20])
data = {
    'x1': [8.6, 8.6, 6.8, 6.8, 9.1, 9.1, 3.5, 3.5],
```

```
    'x2': [9.1, 9.1, 7.9, 7.9, 9.9, 9.9, 4.2, 4.2],
    'x3': [9.2, 9.2, 5.9, 5.9, 8.9, 8.9, 5.6, 5.6],
    'x4': [8.8, 8.8, 6.6, 6.6, 8.8, 8.8, 4.9, 4.9],
    'x5': [8.9, 9.9, 6.1, 6.1, 7.8, 7.8, 5.2, 5.2],
    'rho_y': [0.01, 0.55, 0.01, 0.55, 0.01, 0.55, 0.01, 0.55]
}

df = pd.DataFrame(data)

def calculate_y2(row):
    x_values = np.array([row['x1'], row['x2'], row['x3'], row['x4'], row['x5']])
    rho_y = row['rho_y']
    summation = np.sum(weights * (x_values ** -rho_y))
    y2 = summation ** (-1 / rho_y)
    return y2

def main():
    df['y2'] = df.apply(calculate_y2, axis=1)
    print(df)

if __name__ == "__main__":
    main()
```

计算 y_2 的值

2 .4 模型二输出结果

运行结果如下图所示：



```
D:\Python\python.exe F:\Project\Python\SoftwareQuality\
    x1   x2   x3   x4   x5  rho_y        y2
0  8.6  9.1  9.2  8.8  8.9   0.01  8.927671
1  8.6  9.1  9.2  8.8  9.9   0.55  9.114381
2  6.8  7.9  5.9  6.6  6.1   0.01  6.614570
3  6.8  7.9  5.9  6.6  6.1   0.55  6.596355
4  9.1  9.9  8.9  8.8  7.8   0.01  8.859071
5  9.1  9.9  8.9  8.8  7.8   0.55  8.845068
6  3.5  4.2  5.6  4.9  5.2   0.01  4.694559
7  3.5  4.2  5.6  4.9  5.2   0.55  4.663374

进程已结束，退出代码为 0
```

图 2: 模型二输出结果

3 题目结果

综上所述，对于模型一和模型二的计算，结果如下表所示：

表 1: 作业三的数据

<table>
<tr><th>编号</th><th>x_1</th><th>x_2</th><th>x_3</th><th>x_4</th><th>x_5</th><th>ρ_y</th><th>y_1</th><th>y_2</th></tr>
<tr><td rowspan="2">1</td><td>8.6</td><td>9.1</td><td>9.2</td><td>8.8</td><td>8.9</td><td>0.01</td><td rowspan="2">8.927694</td><td>8.927671</td></tr>
<tr><td>8.6</td><td>9.1</td><td>9.2</td><td>8.8</td><td>8.9</td><td>0.55</td><td>9.114381</td></tr>
<tr><td rowspan="2">2</td><td>6.8</td><td>7.9</td><td>5.9</td><td>6.6</td><td>6.1</td><td>0.01</td><td rowspan="2">6.614913</td><td>6.614570</td></tr>
<tr><td>6.8</td><td>7.9</td><td>5.9</td><td>6.6</td><td>6.1</td><td>0.55</td><td>6.596355</td></tr>
<tr><td rowspan="2">3</td><td>9.1</td><td>9.9</td><td>8.9</td><td>8.8</td><td>7.8</td><td>0.01</td><td rowspan="2">8.859329</td><td>8.859071</td></tr>
<tr><td>9.1</td><td>9.9</td><td>8.9</td><td>8.8</td><td>7.8</td><td>0.55</td><td>8.845068</td></tr>
<tr><td rowspan="2">4</td><td>3.5</td><td>4.2</td><td>5.6</td><td>4.9</td><td>5.2</td><td>0.01</td><td rowspan="2">4.695127</td><td>4.694559</td></tr>
<tr><td>3.5</td><td>4.2</td><td>5.6</td><td>4.9</td><td>5.2</td><td>0.55</td><td>4.663374</td></tr>
</table>

二 作业四

1 题目内容

设 **C919** 飞行控制软件有 **5** 个可信属性：实时性、可靠性、可生存性、可维护性、功能性（其含义见第 4 讲第 **3.2** 节），其正互反判断矩阵 **A** 为

$$\mathbf{A}=\begin{bmatrix}1 & 1/2 & 3 & 2 & 1/2\\ 2 & 1 & 2 & 3 & 2\\ 1/3 & 1/2 & 1 & 2 & 1/3\\ 1/2 & 1/3 & 1/2 & 1 & 2\\ 2 & 1/2 & 3 & 1/2 & 1\end{bmatrix}$$

分别使用右特征向量法 **EV**、对数最小二乘法 **LLSM**、卡方最小二乘法 **CSM** 求出权重向量 $\mathbf{W}^{EV}$、$\mathbf{W}^{LLSM}$、$\mathbf{W}^{CSM}$，在此基础上使用“强度”方法求出最优的权重向量。

2 题目分析

需要完整解答这道题，看来得列出四五段代码：

2 .1 右特征向量法 EV

实在没有使用过 Python 处理矩阵，但显然 Numpy 中的 Array 也可以胜任这个角色。

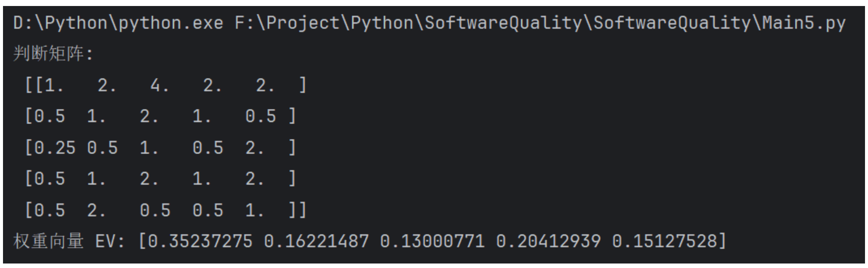
关于特征值和特征向量的代码，我参考了网上的资料：

numpy 求特征值特征向量：https://blog.51cto.com/u_16099238/7589035

```
  # 相似地，我们可以使用 PPT Test 中的示例先作为测试数据
  A = np.array([
    [1, 2, 4, 2, 2],
    [1/2, 1, 2, 1, 1/2],
    [1/4, 1/2, 1, 1/2, 2],
    [1/2, 1, 2, 1, 2],
    [1/2, 2, 1/2, 1/2, 1]
])

  def calculate_ev_weights(matrix):
    eigenvalues, eigenvectors = np.linalg.eig(matrix) # 返回值为元组，第一个元素为特征值，第二个元素为特征向量
    max_index = np.argmax(eigenvalues) # 取最大特征值对应的特征向量
    weights = eigenvectors[:, max_index].real
    # 归一化
    return weights / np.sum(weights)

  weights_ev = calculate_ev_weights(A)
```

右特征向量法

测试效果良好，与 PPT 中的 EA 保持一致（0.3524, 0.1622, 0.1300, 0.2041, 0.1513）：



```
D:\Python\python.exe F:\Project\Python\SoftwareQuality\SoftwareQuality\Main5.py
判断矩阵:
 [[1.   2.   4.   2.   2.  ]
 [0.5  1.   2.   1.   0.5 ]
 [0.25 0.5  1.   0.5  2.  ]
 [0.5  1.   2.   1.   2.  ]
 [0.5  2.   0.5  0.5  1.  ]]
权重向量 EV: [0.35237275 0.16221487 0.13000771 0.20412939 0.15127528]
```

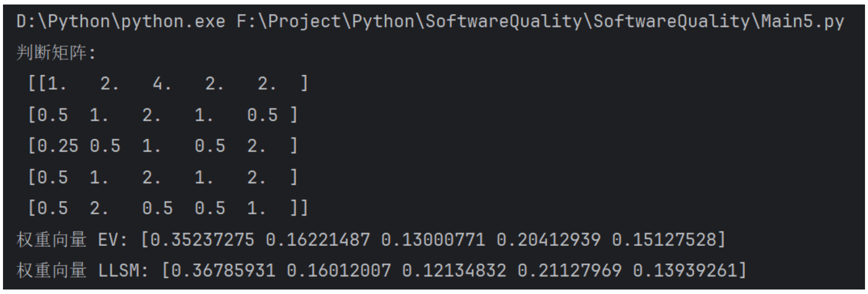
图 3: 右特征向量法权重向量

2 .2 对数最小二乘法 LLSM

```python
def calculate_llsm_weights(matrix):
  numerator = np.prod(matrix, axis=1) ** (1 / matrix.shape[0])   # 分子 = 每行的乘积 × 开维度数次方
  denominator =  np.sum(numerator)   # 分母 = 每行的分子相加
  return numerator / denominator   # 归一化

weights_llsm = calculate_llsm_weights(A)
print("权重向量 W_LLSM:", weights_llsm)
```

对数最小二乘法

测试效果良好，与 PPT 中的 LLSM 保持一致（0.3679, 0.1601, 0.1213, 0.2113, 0.1394）：



```
D:\Python\python.exe F:\Project\Python\SoftwareQuality\SoftwareQuality\Main5.py
判断矩阵:
 [[1.   2.   4.   2.   2.  ]
 [0.5  1.   2.   1.   0.5 ]
 [0.25 0.5  1.   0.5  2.  ]
 [0.5  1.   2.   1.   2.  ]
 [0.5  2.   0.5  0.5  1.  ]]
权重向量 EV: [0.35237275 0.16221487 0.13000771 0.20412939 0.15127528]
权重向量 LLSM: [0.36785931 0.16012007 0.12134832 0.21127969 0.13939261]
```

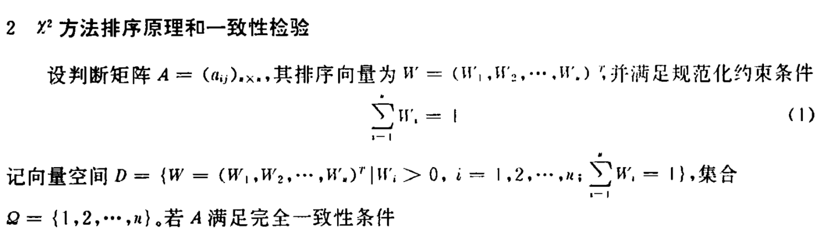
图 4: 对数最小二乘法权重向量

2 .3 卡方最小二乘法 CSM

进入管理工程学报官网：【管理工程学报】

知网检索：知网链接

查阅文献，与 PPT 中的内容相似：



2 χ^2 方法排序原理和一致性检验

设判断矩阵 $A=(a_{ij})_{n\times n}$，其排序向量为 $W=(W_1,W_2,\cdots,W_n)^T$，并满足规范化约束条件

$$\sum_{i=1}^{n} W_i = 1 \qquad (1)$$

记向量空间 $D=\{W=(W_1,W_2,\cdots,W_n)^T \mid W_i>0,\ i=1,2,\cdots,n;\ \sum_{i=1}^{n} W_i = 1\}$，集合 $\Omega=\{1,2,\cdots,n\}$。若 A 满足完全一致性条件

图 5: 文献资料

其他的内容都是一些证明相关的，真正的算法精髓应该在于这里：

由于方程组(13)是一组非线性方程,故为了求解 W^*,可按如下算法步骤进行迭代:

(1) 取定初始排序向量 $W(0)=(W_1(0),W_2(0),\cdots,W_n(0))^T\in D$,并给定迭代精度 ε,同时置 $k=0$,一般情况下,可取 $W(0)=\frac{1}{n}e$,其中:$e=(1,1,\cdots,1)^T$。

(2) 计算

$$\varepsilon_i(W(k))=\sum_{j=1}^{n}\left[(1+a_{ji}^2)\frac{W_i}{W_j}-(1+a_{ij}^2)\frac{W_j}{W_i}\right] \qquad i\in\Omega$$

如果对所有 $i\in\Omega$,恒有 $|\varepsilon_i(W(k))|\leqslant\varepsilon$,则算法停止,$W^*=W(k)$;反之,执行第(3)步。

(3) 确定 m 使 $|\varepsilon_m(W(k))|=\max\limits_{i\in\Omega}\{|\varepsilon_i(W(k))|\}$,

并令

$$\begin{cases} T(k)=\left[\sum_{j\neq m}(1+a_{mj}^2)\frac{W_j(k)}{W_m(k)}\Big/\sum_{j\neq m}(1+a_{jm}^2)\frac{W_m(k)}{W_j(k)}\right]^{1/2} & \\ X_i(k)=\begin{cases} T(k)W_m(k) & i=m \\ W_i(k) & i\neq m \end{cases} & i\in\Omega \\ W_i(k+1)=X_i(k)\Big/\sum_{j=1}^{n}X_j(k) & i\in\Omega \end{cases}$$

(4) 令 $k=k+1$,转(2)。

图 6: 迭代算法

复刻代码:

```python
def calculate_csm_ppt():
n = 5 # 方针阶数
a = []
print("输入方阵：")
for i in range(0, n):
    row = list(map(float, input().split()))
    a.append(row)
# 步骤 （1）
epsilon = 1e-10
w = [1 / n for i in range(0, n)] # 初始解
while True:
    # 步骤 （2）
    m = None
    max_val = None
    for i in range(0, n):
        val = 0
        for j in range(0, n):
            val += (1 + a[j][i] * a[j][i]) * (w[i] / w[j]) - (1 + a[i][j] * a[i][j]) * (w[j] / w[i])
        val = abs(val)
        if max_val == None or val > max_val:
            max_val = val
            m = i
    if max_val != epsilon:
        break
    # 步骤（3）
    up = 0
    bottom = 0
    for j in range(0, n):
        if j != m:
            up += (1 + a[m][j] * a[m][j]) * (w[j] / w[m])
            bottom += (1 + a[j][m] * a[j][m]) * (w[m] / w[j])

    T = pow(up / bottom, 1 / 2)
    X = w
    X[m] *= T
    sum_X = sum(X)
    for i in range(0, n):
        w[i] = X[i] / sum_X
return w
```

卡方最小二乘法

第一次运行时，该矩阵的输出为：[0.2, 0.2, 0.2, 0.2, 0.2] 并不符合预期。为什么看起来全部都是初始解，而没有发生迭代呢？我猜测应该和判断结束的条件有关系，于是查看代码：

（2）计算

$$\varepsilon_i(W(k)) = \sum_{j=1}^{n}\left[(1+a_{ji}^2)\frac{W_i}{W_j} - (1+a_{ij}^2)\frac{W_j}{W_i}\right] \qquad i \in \Omega$$

如果对所有 $i \in \Omega$，恒有 $|\varepsilon_i(W(k))| \leqslant \varepsilon$，则算法停止，$W^* = W(k)$；反之，执行第（3）步。

图 7: 结束条件

而在 PPT 给出的代码中，出现了感叹号和问号，是这里把我误导了：

```
    val = abs(val)
    if max_val == None or val ¿ max-val:
        max_val = val
        m = i
if max-val ¡= epsilon:
    break
```

图 8: PPT 代码

那么，根据文献内容，只要将 `max_val != epsilon` 改为 `max_val <= epsilon` 即可。
代码运行成功，与 PPT 完全一致（0.3767, 0.1546, 0.1186, 0.2121, 0.1379）

```
输入方阵：
1 2 4 2 2
0.5 1 2 1 0.5
0.25 0.5 1 0.5 2
0.5 1 2 1 2
0.5 2 0.5 0.5 1
CSM 权重向量： [0.3766703113826757, 0.15459360515184867, 0.11859209445452762, 0.21219212515053232, 0.13795186386041577]
```

图 9: 卡方最小二乘法权重向量计算结果

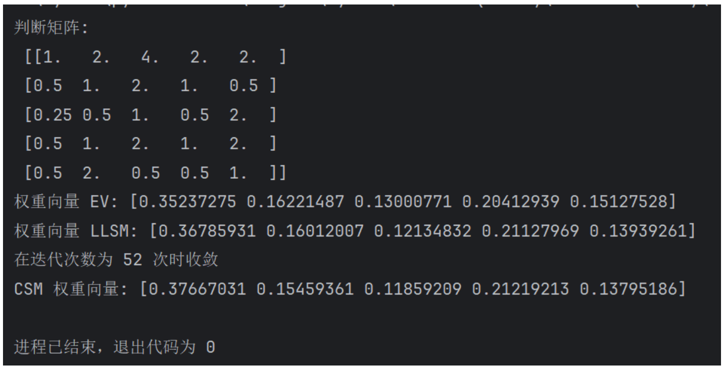
接下来，算法步骤明晰，不妨改写为使用 Numpy 来实现：

```python
def calculate_csm_weights(matrix: np.array, epsilon: float, max_iterations: int) -> np.ndarray:
  '''计算 CSM，传入：matrix, precision, maximum iterations'''
  n = matrix.shape[0]
  # 初始化初始解
  W = np.ones(n) / n

  for k in range(max_iterations):
      e = np.zeros(n)
      for i in range(n):
          e[i] = np.sum([(1 + matrix[j, i] ** 2) * (W[i] / W[j]) - (1 + matrix[i, j] ** 2) * (W[j] / W[i])
                          for j in range(n) if j != i])

      max_e = np.max(np.abs(e))
      if max_e <= epsilon: # 若精度已到达，那么停止迭代
          print(f"在迭代次数为 {k} 次时收敛")
          break

      m = np.argmax(np.abs(e)) # 查找最大无差的索引

      # 计算 T(k)
      up = np.sum([(1 + matrix[m, j] ** 2) * (W[j] / W[m]) for j in range(n) if j != m])
      bottom = np.sum([(1 + matrix[j, m] ** 2) * (W[m] / W[j]) for j in range(n) if j != m])
      T = np.sqrt(up / bottom)

      # 更新矩阵向量，归一化
      X = W.copy()
      X[m] *= T
```

```
        W = X / np.sum(X)

    return W
```

卡方最小二乘法

测试结果与 PPT 一致：(0.3767, 0.1546, 0.1186, 0.2121, 0.1379)

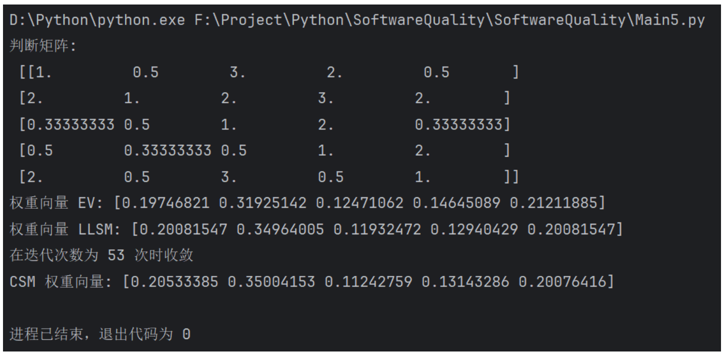


```
判断矩阵:
 [[1.   2.   4.   2.   2.  ]
 [0.5  1.   2.   1.   0.5 ]
 [0.25 0.5  1.   0.5  2.  ]
 [0.5  1.   2.   1.   2.  ]
 [0.5  2.   0.5  0.5  1.  ]]
权重向量 EV: [0.35237275 0.16221487 0.13000771 0.20412939 0.15127528]
权重向量 LLSM: [0.36785931 0.16012007 0.12134832 0.21127969 0.13939261]
在迭代次数为 52 次时收敛
CSM 权重向量: [0.37667031 0.15459361 0.11859209 0.21219213 0.13795186]

进程已结束，退出代码为 0
```

图 10: 卡方最小二乘法权重向量计算结果

2 .4 作业四三种权重向量的结果

将数据修改为作业四中的矩阵，运行结果如下：



```
D:\Python\python.exe F:\Project\Python\SoftwareQuality\SoftwareQuality\Main5.py
判断矩阵:
 [[1.         0.5        3.         2.         0.5       ]
 [2.         1.         2.         3.         2.        ]
 [0.33333333 0.5        1.         2.         0.33333333]
 [0.5        0.33333333 0.5        1.         2.        ]
 [2.         0.5        3.         0.5        1.        ]]
权重向量 EV: [0.19746821 0.31925142 0.12471062 0.14645089 0.21211885]
权重向量 LLSM: [0.20081547 0.34964005 0.11932472 0.12940429 0.20081547]
在迭代次数为 53 次时收敛
CSM 权重向量: [0.20533385 0.35004153 0.11242759 0.13143286 0.20076416]

进程已结束，退出代码为 0
```

图 11: 卡方最小二乘法权重向量计算结果

2 .5 计算 TD 值

对于任意给定的正互反判断矩阵，不同的优先级方法会得到不同的权重向量。为了评价这些权重向量的质量，用两种评价标准：“程度”（strengths）和“方向”（directions）。

作业四要求使用 strengths 标准来衡量：

- 使用“程度”评价准则来评估结果：

$$TD^{(k)} = \sum_{i=1}^{n}\sum_{j=1}^{n}\left|a_{ij} - \frac{w_i^{(k)}}{w_j^{(k)}}\right|, \quad k = 1, 2, 3$$

- 比较不同方法计算出的 $TD^{(1)}$,$TD^{(2)}$,$TD^{(3)}$ 值，选择最小 TD 所对应的权重向量为最优权重向量。

根据公式，可以写出计算 TD 值的代码：

```
def calculate_td(matrix: np.array, weight_vector: np.ndarray) -> float:
    n = len(weight_vector)
    td = 0.0
    for i in range(n):
        for j in range(n):
```

```
        td += abs(matrix[i, j] - (weight_vector[i] / weight_vector[j]))
    return td
```

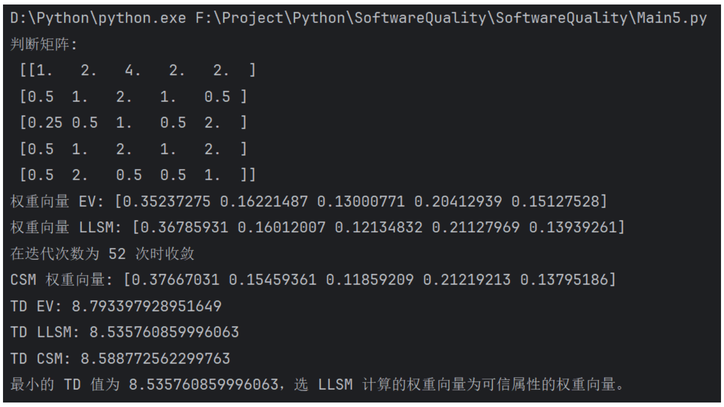
计算 TD 值

相似地，我们先输入 PPT 中的示例，查看是否正确：

```
# PPT 中的权重向量
w1 = np.array([0.3524, 0.1622, 0.1300, 0.2041, 0.1513])
w2 = np.array([0.3679, 0.1601, 0.1213, 0.2113, 0.1394])
w3 = np.array([0.3767, 0.1546, 0.1186, 0.2121, 0.1379])

TD_EV = calculate_td(A, w1)
TD_LLSM = calculate_td(A, w2)
TD_CSM = calculate_td(A, w3)

print("TD EV:", TD_EV)
print("TD LLSM:", TD_LLSM)
print("TD CSM:", TD_CSM)

td_values = {"EV": TD_EV, "LLSM": TD_LLSM, "CSM": TD_CSM}
min_method = min(td_values, key=td_values.get)
print(f"最小的 TD 值为 {td_values[min_method]}, 选 {min_method} 计算的权重向量为可信属性的权重向量。")
```

示例权重向量

输出结果如下：

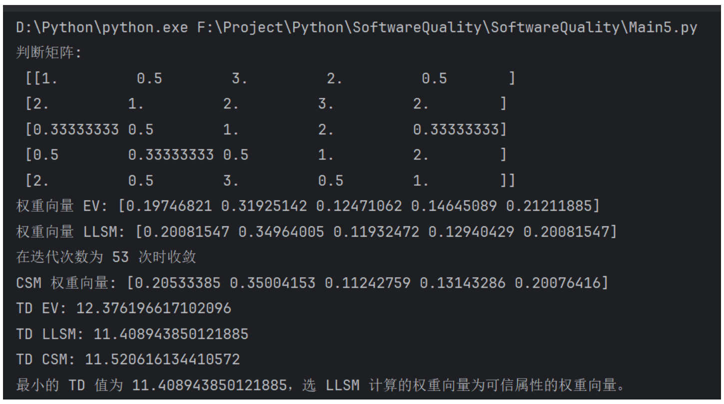


```
D:\Python\python.exe F:\Project\Python\SoftwareQuality\SoftwareQuality\Main5.py
判断矩阵:
 [[1.   2.   4.   2.   2.  ]
 [0.5  1.   2.   1.   0.5 ]
 [0.25 0.5  1.   0.5  2.  ]
 [0.5  1.   2.   1.   2.  ]
 [0.5  2.   0.5  0.5  1.  ]]
权重向量 EV: [0.35237275 0.16221487 0.13000771 0.20412939 0.15127528]
权重向量 LLSM: [0.36785931 0.16012007 0.12134832 0.21127969 0.13939261]
在迭代次数为 52 次时收敛
CSM 权重向量: [0.37667031 0.15459361 0.11859209 0.21219213 0.13795186]
TD EV: 8.793397928951649
TD LLSM: 8.535760859996063
TD CSM: 8.588772562299763
最小的 TD 值为 8.535760859996063, 选 LLSM 计算的权重向量为可信属性的权重向量。
```

图 12: TD 值示例计算结果

3 题目结果

将代码中的权重向量改为我们刚刚计算过后的结果，即整道题的答案如下所示：



```
D:\Python\python.exe F:\Project\Python\SoftwareQuality\SoftwareQuality\Main5.py
判断矩阵:
 [[1.         0.5        3.         2.         0.5       ]
 [2.         1.         2.         3.         2.        ]
 [0.33333333 0.5        1.         2.         0.33333333]
 [0.5        0.33333333 0.5        1.         2.        ]
 [2.         0.5        3.         0.5        1.        ]]
权重向量 EV: [0.19746821 0.31925142 0.12471062 0.14645089 0.21211885]
权重向量 LLSM: [0.20081547 0.34964005 0.11932472 0.12940429 0.20081547]
在迭代次数为 53 次时收敛
CSM 权重向量: [0.20533385 0.35004153 0.11242759 0.13143286 0.20076416]
TD EV: 12.376196617102096
TD LLSM: 11.408943850121885
TD CSM: 11.520616134410572
最小的 TD 值为 11.408943850121885, 选 LLSM 计算的权重向量为可信属性的权重向量。
```

图 13: 作业四结果

最终结果如下所示：

解答 二 .1: 作业四

正互反判断矩阵：

$$A=\begin{bmatrix}1 & \frac{1}{2} & 3 & 2 & \frac{1}{2}\\ 2 & 1 & 2 & 3 & 2\\ \frac{1}{3} & \frac{1}{2} & 1 & 2 & \frac{1}{3}\\ \frac{1}{2} & \frac{1}{3} & \frac{1}{2} & 1 & 2\\ 2 & \frac{1}{2} & 3 & \frac{1}{2} & 1\end{bmatrix}$$

方法	权重向量
EV	(0.1975, 0.3193, 0.1247, 0.1465, 0.2121)
LLSM	(0.2008, 0.3496, 0.1193, 0.1294, 0.2008)
CSM	(0.2053, 0.3500, 0.1124, 0.1314, 0.2008)

三个权重向量的值分别为 $TD^{EV}=12.3762, TD^{LLSM}=11.4089, TD^{CSM}=11.5206$。
最小的 TD 值为 11.408943850121885，选 LLSM 计算的权重向量为可信属性的权重向量。

三 附录

1 参考文献

- 王应明, and 傅国伟. ” 判断矩阵排序的 X^2 方法.” 管理工程学报 8.1 (1994): 26-32.
- **numpy 求特征值特征向量：**https://blog.51cto.com/u_16099238/7589035

2 文件代码

```
# Code By Deralive (10235101526)
# https://github.com/Shichien

import numpy as np

# 作业四矩阵
A = np.array([
    [1, 1/2, 3, 2, 1/2],
    [2, 1, 2, 3, 2],
    [1/3, 1/2, 1, 2, 1/3],
    [1/2, 1/3, 1/2, 1, 2],
    [2, 1/2, 3, 1/2, 1]
])

# # PPT 中的测试矩阵
# A = np.array([
#     [1, 2, 4, 2, 2],
#     [1/2, 1, 2, 1, 1/2],
#     [1/4, 1/2, 1, 1/2, 2],
#     [1/2, 1, 2, 1, 2],
#     [1/2, 2, 1/2, 1/2, 1]
# ])

def calculate_ev_weights(matrix):
    eigenvalues, eigenvectors = np.linalg.eig(matrix)  # 返回值为元组，第一个元素为特征值，第二个元素为特征向量
    max_index = np.argmax(eigenvalues)  # 取最大特征值对应的特征向量
```

```python
        weights = eigenvectors[:, max_index].real
        # 归一化
        return weights / np.sum(weights)

def calculate_llsm_weights(matrix):
    # 分子 = 每行的乘积 × 开维度数次方
    numerator = np.prod(matrix, axis=1) ** (1 / matrix.shape[0])
    # 分母 = 每行的分子相加
    denominator =  np.sum(numerator)
    # 归一化
    return numerator / denominator

def calculate_csm_weights(matrix: np.array, epsilon: float, max_iterations: int) -> np.ndarray:
    '''计算 CSM，传入：matrix, precision, maximum iterations'''
    n = matrix.shape[0]
    # 初始化初始解
    W = np.ones(n) / n

    for k in range(max_iterations):
        e = np.zeros(n)
        for i in range(n):
            e[i] = np.sum([(1 + matrix[j, i] ** 2) * (W[i] / W[j]) - (1 + matrix[i, j] ** 2) * (W[j] / W[i])
                           for j in range(n) if j != i])

        max_e = np.max(np.abs(e))
        if max_e <= epsilon: # 若精度已到达，那么停止迭代
            print(f"在迭代次数为 {k} 次时收敛")
            break

        m = np.argmax(np.abs(e)) # 查找最大无差的索引

        # 计算 T(k)
        up = np.sum([(1 + matrix[m, j] ** 2) * (W[j] / W[m]) for j in range(n) if j != m])
        bottom = np.sum([(1 + matrix[j, m] ** 2) * (W[m] / W[j]) for j in range(n) if j != m])
        T = np.sqrt(up / bottom)

        # 更新矩阵向量，归一化
        X = W.copy()
        X[m] *= T
        W = X / np.sum(X)

    return W

def calculate_csm_weights_ppt():
    n = 5 # 方针阶数
    a = []
    print("输入方阵：")
    for i in range(0, n):
        row = list(map(float, input().split()))
        a.append(row)
    # 步骤 （1）
    epsilon = 1e-10
    w = [1 / n for i in range(0, n)] # 初始解
    while True:
        # 步骤 （2）
        m = None
        max_val = None
        for i in range(0, n):
            val = 0
            for j in range(0, n):
                val += (1 + a[j][i] * a[j][i]) * (w[i] / w[j]) - (1 + a[i][j] * a[i][j]) * (w[j] / w[i])
            val = abs(val)
            if max_val == None or val > max_val:
                max_val = val
```

```python
                    m = i
            if max_val <= epsilon:
                break
            # 步骤（3）
            up = 0
            bottom = 0
            for j in range(0, n):
                if j != m:
                    up += (1 + a[m][j] * a[m][j]) * (w[j] / w[m])
                    bottom += (1 + a[j][m] * a[j][m]) * (w[m] / w[j])

            T = pow(up / bottom, 1 / 2)
            X = w
            X[m] *= T
            sum_X = sum(X)
            for i in range(0, n):
                w[i] = X[i] / sum_X
    return w

def calculate_td(matrix: np.array, weight_vector: np.ndarray) -> float:
    n = len(weight_vector)
    td = 0.0
    for i in range(n):
        for j in range(n):
            td += abs(matrix[i, j] - (weight_vector[i] / weight_vector[j]))
    return td

weights_ev = calculate_ev_weights(A)
print("判断矩阵:\n", A)
print("权重向量 EV:", weights_ev)

weights_llsm = calculate_llsm_weights(A)
print("权重向量 LLSM:", weights_llsm)

# weight_vec_tor_csm_ppt = calculate_csm_ppt()
# print("CSM 权重向量: ", weight_vec_tor_csm_ppt)

weight_csm = calculate_csm_weights(A, 1e-10, 1000)
print("CSM 权重向量:", weight_csm)

# # PPT 中的权重向量
# w1 = np.array([0.3524, 0.1622, 0.1300, 0.2041, 0.1513])
# w2 = np.array([0.3679, 0.1601, 0.1213, 0.2113, 0.1394])
# w3 = np.array([0.3767, 0.1546, 0.1186, 0.2121, 0.1379])

# 新的权重向量
w1 = np.array([0.1975, 0.3193, 0.1247, 0.1465, 0.2121])  # EV
w2 = np.array([0.2008, 0.3496, 0.1193, 0.1294, 0.2008])  # LLSM
w3 = np.array([0.2053, 0.3500, 0.1124, 0.1314, 0.2008])  # CSM

TD_EV = calculate_td(A, w1)
TD_LLSM = calculate_td(A, w2)
TD_CSM = calculate_td(A, w3)

print("TD EV:", TD_EV)
print("TD LLSM:", TD_LLSM)
print("TD CSM:", TD_CSM)

td_values = {"EV": TD_EV, "LLSM": TD_LLSM, "CSM": TD_CSM}
min_method = min(td_values, key=td_values.get)
print(f"最小的 TD 值为 {td_values[min_method]}，选 {min_method} 计算的权重向量为可信属性的权重向量。")
```

Main.py