

浙江工业大学

多元统计分析课程设计实验报告

(2021 级)



学生姓名 温家伟

学生学号 202103151422

学科(专业) 大数据分析 2101 班

所在学院 理学院

提交日期 2024 年 6 月 30 日

一、观察值缺失时均值向量的推断

人们通常得不到观察向量的某些分量值，到目前为止，还没有能够处理这种情形的统计方法。Dempster 等人在1977年提出从不完全数据出发计算极大似然估计的一般方法。这种方法通常被称为EM算法，该方法包含两个迭代计算组成，她们分别称为预测步骤和估计步骤：

预测步骤：给定未知参数的某一估计，预测任何缺失观测值对（完全数据）充分统计量的贡献。

估计步骤：利用预测得到的充分统计量计算参数的修正估计。

1.1 理论推导

观测值 $x_1, x_2, \dots, x_n \sim N_p(\mu, \Sigma)$, 另外 $(n-1)S = \sum_{i=1}^n (x_i - \hat{x})(x_i - \hat{x})^\top$

$$\begin{aligned} T_1 &= \sum_{i=1}^n x_i = n\bar{x} \\ T_2 &= \sum_{i=1}^n x_i x_i^\top \\ &= \sum_{i=1}^n (x_i - \bar{x} + \bar{x})(x_i - \bar{x} + \bar{x})^\top \\ &= \sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x})^\top + \sum_{i=1}^n (x_i - \bar{x})\bar{x}^\top + \sum_{i=1}^n \bar{x}(x_i - \bar{x})^\top + \sum_{i=1}^n \bar{x}\bar{x}^\top \\ &= \sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x})^\top + 0 + 0 + n\bar{x}\bar{x}^\top \\ &= (n-1)S + n\bar{x}\bar{x}^\top \end{aligned}$$

对每一具有缺损值得向量 x_j , 记 $x_j^{(1)}$ 为其缺损向量, $x_j^{(2)}$ 为其可获得分量, 于是

$$x_j = \begin{bmatrix} x_j^{(1)} \\ \dots \\ x_j^{(2)} \end{bmatrix}, \mu = \begin{bmatrix} \mu^{(1)} \\ \dots \\ \mu^{(2)} \end{bmatrix}, \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$$

- 证明分块矩阵求行列式

$$\begin{bmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix}$$

两边取行列式：

$$\begin{aligned} |A| &= |A_{11}| |A_{22} - A_{21}A_{11}^{-1}A_{12}| \\ \begin{bmatrix} I & -\Sigma_{12}\Sigma_{22}^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} &= \begin{bmatrix} \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} & 0 \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \end{aligned}$$

两边取行列式

$$|A| = |A_{22}| |A_{11} - A_{12}A_{22}^{-1}A_{21}|$$

- 证明分块矩阵求逆

$$\begin{aligned} \Sigma &= \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \\ \begin{bmatrix} E & -\Sigma_{12}\Sigma_{22}^{-1} \\ 0 & E \end{bmatrix} \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} &= \begin{bmatrix} \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} & 0 \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \\ \begin{bmatrix} \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} & 0 \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \begin{bmatrix} E & 0 \\ -\Sigma_{22}^{-1}\Sigma_{21} & E \end{bmatrix} &= \begin{bmatrix} \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} & 0 \\ 0 & \Sigma_{22} \end{bmatrix} \end{aligned}$$

即,

$$\begin{bmatrix} E & -\Sigma_{12}\Sigma_{22}^{-1} \\ 0 & E \end{bmatrix} \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \begin{bmatrix} E & 0 \\ -\Sigma_{22}^{-1}\Sigma_{21} & E \end{bmatrix} = \begin{bmatrix} \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} & 0 \\ 0 & \Sigma_{22} \end{bmatrix}$$

两边求逆：

$$\begin{aligned}
& \begin{bmatrix} E & 0 \\ -\Sigma_{22}^{-1}\Sigma_{21} & E \end{bmatrix}^{-1} \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}^{-1} \begin{bmatrix} E & -\Sigma_{12}\Sigma_{22}^{-1} \\ 0 & E \end{bmatrix} = \begin{bmatrix} (\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})^{-1} & 0 \\ 0 & \Sigma_{22}^{-1} \end{bmatrix} \\
\Rightarrow & \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}^{-1} = \begin{bmatrix} E & 0 \\ -\Sigma_{22}^{-1}\Sigma_{21} & E \end{bmatrix} \begin{bmatrix} (\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})^{-1} & 0 \\ 0 & \Sigma_{22}^{-1} \end{bmatrix} \begin{bmatrix} E & -\Sigma_{12}\Sigma_{22}^{-1} \\ 0 & E \end{bmatrix} \\
& = \begin{bmatrix} \Sigma_{11.2} & -\Sigma_{11.2}^{-1}\Sigma_{12}\Sigma_{22}^{-1} \\ -\Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11.2}^{-1} & \Sigma_{22}^{-1} + \Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11.2}^{-1}\Sigma_{12}\Sigma_{22}^{-1} \end{bmatrix}
\end{aligned}$$

其中,

$$\Sigma_{11.2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

下面正式开始推导:

$$\begin{aligned}
f(x_j^{(1)}|x_j^{(2)}) &= \frac{f(x_j)}{f(x_j^{(2)})} \\
&= \frac{(2\pi)^{-\frac{p}{2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(x_j - \mu)^\top \Sigma^{-1}(x_j - \mu)}}{(2\pi)^{-\frac{p_2}{2}} |\Sigma_{22}|^{-\frac{1}{2}} e^{-\frac{1}{2}(x_j^{(2)} - \mu^{(2)})^\top \Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)})}} \\
&= (2\pi)^{-\frac{p_1}{2}} \left(\frac{|\Sigma|}{|\Sigma_{22}|} \right)^{-\frac{1}{2}} e^{-\frac{1}{2} \left\{ [(x_j - \mu)^\top \Sigma^{-1}(x_j - \mu)] - [(x_j^{(2)} - \mu^{(2)})^\top \Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)})] \right\}} \\
&= (2\pi)^{-\frac{p_1}{2}} |\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}|^{-\frac{1}{2}} e^{-\frac{1}{2} \left[(x_j^{(1)} - \mu^{(1)}) - \Sigma_{11}\Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)}) \right]^\top (\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})^{-1} \left[(x_j^{(1)} - \mu^{(1)}) - \Sigma_{11}\Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)}) \right]}
\end{aligned}$$

其中, 指数部分化简如下: $e^{-\frac{1}{2} \times E}$

$$\begin{aligned}
E &= \begin{bmatrix} x_j^{(1)} - \mu^{(1)} \\ x_j^{(2)} - \mu^{(2)} \end{bmatrix}^\top \begin{bmatrix} \Sigma_{11.2} & -\Sigma_{11.2}^{-1}\Sigma_{12}\Sigma_{22}^{-1} \\ -\Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11.2}^{-1} & \Sigma_{22}^{-1} + \Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11.2}^{-1}\Sigma_{12}\Sigma_{22}^{-1} \end{bmatrix} \begin{bmatrix} x_j^{(1)} - \mu^{(1)} \\ x_j^{(2)} - \mu^{(2)} \end{bmatrix} - [(x_j^{(2)} - \mu^{(2)})\Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)})^\top] \\
&= \begin{bmatrix} (x_j^{(1)} - \mu^{(1)})\Sigma_{11.2}^{-1} - (x_j^{(2)} - \mu^{(2)})\Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11.2} \\ -(x_j^{(1)} - \mu^{(1)})\Sigma_{11.2}^{-1}\Sigma_{12}\Sigma_{22}^{-1} + (x_j^{(2)} - \mu^{(2)}) (\Sigma_{22}^{-1} + \Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11.2}^{-1}\Sigma_{12}\Sigma_{22}^{-1}) \end{bmatrix}^\top \begin{bmatrix} x_j^{(1)} - \mu^{(1)} \\ x_j^{(2)} - \mu^{(2)} \end{bmatrix} - [(x_j^{(2)} - \mu^{(2)})\Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)})^\top] \\
&= [(x_j^{(1)} - \mu^{(1)})\Sigma_{11.2}^{-1} - (x_j^{(2)} - \mu^{(2)})\Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11.2}^{-1}] (x_j^{(1)} - \mu^{(1)})^\top \\
&+ [-(x_j^{(1)} - \mu^{(1)})\Sigma_{11.2}^{-1}\Sigma_{12}\Sigma_{22}^{-1} + (x_j^{(2)} - \mu^{(2)}) (\Sigma_{22}^{-1} + \Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11.2}^{-1}\Sigma_{12}\Sigma_{22}^{-1})] (x_j^{(2)} - \mu^{(2)})^\top - [(x_j^{(2)} - \mu^{(2)})\Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)})^\top] \\
&= (x_j^{(1)} - \mu^{(1)})\Sigma_{11.2}^{-1}(x_j^{(1)} - \mu^{(1)})^\top - (x_j^{(2)} - \mu^{(2)})\Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11.2}^{-1}(x_j^{(1)} - \mu^{(1)})^\top - (x_j^{(1)} - \mu^{(1)})\Sigma_{11.2}^{-1}\Sigma_{12}\Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)})^\top \\
&+ (x_j^{(2)} - \mu^{(2)}) (\Sigma_{22}^{-1} + \Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11.2}^{-1}\Sigma_{12}\Sigma_{22}^{-1})(x_j^{(2)} - \mu^{(2)})^\top - [(x_j^{(2)} - \mu^{(2)})\Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)})^\top]
\end{aligned}$$

令:

$$\begin{cases} y &= \Sigma_{12}\Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)}) \\ y^\top &= (x_j^{(2)} - \mu^{(2)})^\top \Sigma_{22}^{-1}\Sigma_{12}^\top \end{cases}$$

那么,

$$\begin{aligned}
E &= (x_j^{(1)} - \mu^{(1)})\Sigma_{11.2}^{-1}(x_j^{(1)} - \mu^{(1)})^\top - y^\top \Sigma_{11.2}^{-1} y \\
&= (x_j^{(1)} - \mu^{(1)} - y)^\top \Sigma_{11.2}^{-1} (x_j^{(1)} - \mu^{(1)} - y)
\end{aligned}$$

所以,

$$\begin{aligned}
f(x_j^{(1)}|x_j^{(2)}) &= (2\pi)^{-\frac{p}{2}} |\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}|^{-\frac{1}{2}} e^{-\frac{1}{2} \left[(x_j^{(1)} - \mu^{(1)}) - \Sigma_{11}\Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)}) \right]^\top (\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})^{-1} \left[(x_j^{(1)} - \mu^{(1)}) - \Sigma_{11}\Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)}) \right]} \\
\Rightarrow & \begin{cases} \widetilde{x_j^{(1)}} &= \widetilde{\mu^{(1)}} + \Sigma_{12}\Sigma_{22}^{-1}(x_j^{(2)} - \mu^{(2)}) \quad \rightarrow T_1 \\ \widetilde{x_j^{(1)}(x_j^{(1)})^\top} &= \widetilde{x_j^{(1)}(x_j^{(1)})^\top} + (\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}) \quad \rightarrow T_2 \\ \widetilde{x_j^{(2)}(x_j^{(2)})^\top} &= \widetilde{x_j^{(2)}(x_j^{(1)})^\top} \quad \rightarrow T_2 \end{cases}
\end{aligned}$$

修正后的极大似然估计:

$$\begin{cases} \widetilde{\mu} &= \frac{1}{n} \widetilde{T_1} \\ \widetilde{\Sigma} &= \frac{1}{n} \widetilde{T_2} - \widetilde{\mu} \widetilde{\mu}^\top \end{cases}$$

1.2 代码

EM算法代码如下。

```
def em(xdata, mu0, Sigma0, times = 0):
    time = 0
    n, p = xdata.shape
    mu1 = mu0 + 1
    Sigma1 = Sigma0 + 1

    # 计算均方误差，前后两次迭代的均值和协方差元素逐个比较
    def err(mu0, Sigma0, mu1, Sigma1):
        th0 = np.concatenate((mu0, Sigma0.flatten()))
        th1 = np.concatenate((mu1, Sigma1.flatten()))
        return np.sqrt(np.sum((th0 - th1) ** 2))

    if times:
        # 循环迭代
        while time < times:
            time += 1
            mu1 = mu0.copy()
            Sigma1 = Sigma0.copy()

            # T_1
            T_1 = np.copy(xdata)
            # T_2的增量
            delta = np.zeros((p, p))

            # 迭代每一组数据
            for i in range(n):
                # 如果这一行数据有缺失值，才继续循环
                if np.any(np.isnan(xdata[i])):
                    # 拿到这一行数据
                    zi = xdata[i]
                    # 找到缺失值的索引
                    na_idx = np.where(np.isnan(zi))[0]
                    # 找到非缺失值的索引
                    cs_idx = np.where(~np.isnan(zi))[0]

                    # 分块
                    Sigma011 = Sigma0[np.ix_(na_idx, na_idx)]
                    Sigma012 = Sigma0[np.ix_(na_idx, cs_idx)]
                    Sigma022_inv = np.linalg.inv(Sigma0[np.ix_(cs_idx, cs_idx)])

                    T_1[i, na_idx] = mu0[na_idx] + np.dot(Sigma012, Sigma022_inv).dot(zi[cs_idx] - mu0[cs_idx])
                    delta[np.ix_(na_idx, na_idx)] += Sigma011 - np.dot(Sigma012, Sigma022_inv).dot(Sigma012.T)

            mu0 = np.mean(T_1, axis=0)
            # T_2 = (n - 1) * np.cov(T_1, rowvar=False) + delta + n * np.dot(mu0, mu0.T)
            # Sigma0 = T_2 / n - np.dot(mu0, mu0.T)
            # 上面两步合并为下面一步
            Sigma0 = (n - 1) * np.cov(T_1, rowvar=False) / n + delta / n
    else:
        # 循环迭代
        while err(mu0, Sigma0, mu1, Sigma1) > 1e-12:
            mu1 = mu0.copy()
            Sigma1 = Sigma0.copy()

            # T_1
            T_1 = np.copy(xdata)
            # T_2的增量
            delta = np.zeros((p, p))

            # 迭代每一组数据
            for i in range(n):
                # 如果这一行数据有缺失值，才继续循环
                if np.any(np.isnan(xdata[i])):
                    # 拿到这一行数据
                    zi = xdata[i]
                    # 找到缺失值的索引
                    na_idx = np.where(np.isnan(zi))[0]
```

```

# 找到非缺失值的索引
cs_idx = np.where(~np.isnan(zi))[0]

# 分块
Sigma011 = Sigma0[np.ix_(na_idx, na_idx)]
Sigma012 = Sigma0[np.ix_(na_idx, cs_idx)]
Sigma022_iv = np.linalg.inv(Sigma0[np.ix_(cs_idx, cs_idx)])

T_1[i, na_idx] = mu0[na_idx] + np.dot(Sigma012, Sigma022_iv).dot(zi[cs_idx] - mu0[cs_idx])
delta[np.ix_(na_idx, na_idx)] += Sigma011 - np.dot(Sigma012, Sigma022_iv).dot(Sigma012.T)

mu0 = np.mean(T_1, axis=0)
# T_2 = (n - 1) * np.cov(T_1, rowvar=False) + delta + n * np.dot(mu0, mu0.T)
# Sigma0 = T_2 / n - np.dot(mu0, mu0.T)
# 上面两步合并为下面一步
Sigma0 = (n - 1) * np.cov(T_1, rowvar=False) / n + delta / n

return {'mu': mu0, 'Sigma': Sigma0}

```

1.3 结果

1.3.1 验证算法的正确性

为了验证代码的正确性，我用例5.13的数据，设置EM算法只迭代一次观察结果。

```

jun = np.array([6, 1, 4])
matrix = np.array([
    [np.nan, 0, 3],
    [7, 2, 6],
    [5, 1, 2],
    [np.nan, np.nan, 5]
])

s = np.array([
    [0.5, 0.25, 1],
    [0.25, 0.5, 0.75],
    [1, 0.75, 2.5]
])

result2 = em(matrix, jun, s, 10)
print(result2['mu'])
print(result2['Sigma'])

```

```

估计 mu:
[6.03181818 1.075      4.        ]
估计 Sigma:
[[0.60530992 0.33329545 1.16818182]
 [0.33329545 0.585625   0.825      ]
 [1.16818182 0.825     2.5        ]]

```

如图所示，代码运行出来的结果与例教材5.13迭代一次的结果相符。

1.3.2 生成正态分布的数据运行EM算法

```

# 设置超参数
np.random.seed(1422)
mu0 = np.array([22, 23, 24, 25, 26])
p = 5
n = 1000
Sig0 = np.array([
    [0.64422976, 0.02235931, 0.00341657, 0.03159973, -0.11107787],
    [0.02235931, 0.55234995, -0.07244604, 0.12820811, 0.23877859],
    [0.00341657, -0.07244604, 0.64107747, 0.02788362, -0.06677286],
    [0.03159973, 0.12820811, 0.02788362, 0.67289534, 0.06010743],
    [-0.11107787, 0.23877859, -0.06677286, 0.06010743, 0.44281487],
])

# 生成指定均值和协方差的数据
triv = np.random.multivariate_normal(mu0, Sig0, n)

```

```

# 随机缺失
misp = 0.2
misidx = np.random.binomial(1, misp, size=(n, p)).astype(bool)
triv[misidx] = np.nan

com_cases = triv[~np.isnan(triv).any(axis=1)]

# 初始化EM算法参数
mu_ini = np.zeros(p)
sigma_ini = np.eye(p)

# EM 估计
result = em(com_cases, mu_ini, sigma_ini)
print("估计 mu:")
print(result['mu'])
print("估计 Sigma:")
print(result['sigma'])

# 画出正态分布图
plt.figure(figsize=(12, 8))
for i in range(p):
    mean = result['mu'][i]
    cov = result['sigma'][i, i] # 取对角元素作为方差

    x = np.linspace(mean - 3*np.sqrt(cov), mean + 3*np.sqrt(cov), 100)
    plt.plot(x, multivariate_normal.pdf(x, mean=mean, cov=cov), label=f'Component {i+1}')

plt.title('Estimated Normal Distributions')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()

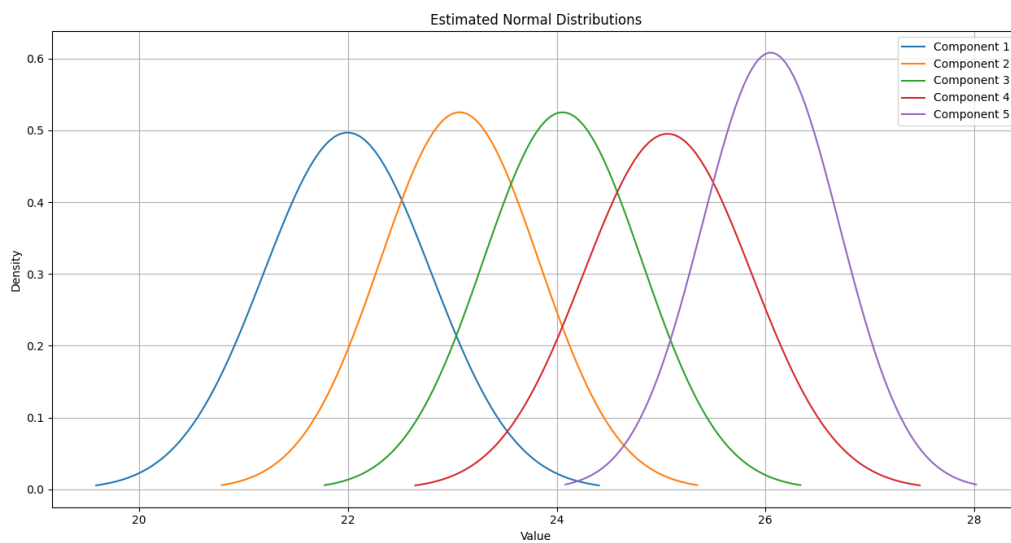
```

```

估计 mu:
[21.99568816 23.06858087 24.05570571 25.0617277 26.04997606]
估计 Sigma:
[[ 0.64430053  0.01062539  0.06854223  0.02172672 -0.0666111 ]
 [ 0.01062539  0.57667491 -0.02605814  0.1324928  0.24202908]
 [ 0.06854223 -0.02605814  0.57688668  0.0653542 -0.04990927]
 [ 0.02172672  0.1324928  0.0653542  0.6486707  0.1256597 ]
 [-0.0666111  0.24202908 -0.04990927  0.1256597  0.4300725 ]]

```

画出的正态分布图如下：



二、验证巴特利特关于多元方差分析的抽样分布定理

2.1 理论推导

设从 n 个总体分别抽取的随机样本为

$$\begin{aligned}\text{总体1: } & X_{11}, X_{12}, \dots, X_{1n} \\ \text{总体2: } & X_{21}, X_{22}, \dots, X_{2n} \\ & \vdots \\ \text{总体g: } & X_{g1}, X_{g2}, \dots, X_{gn}\end{aligned}$$

处理效应的平方和:

$$B = \sum_{l=1}^g n_l (\bar{x}_l - \bar{x})(\bar{x}_l - \bar{x})^T$$

残差平方和:

$$W = \sum_{l=1}^g \sum_{j=1}^{n_l} (x_{lj} - \bar{x}_l)(x_{lj} - \bar{x}_l)^T$$

则

$$\Lambda^* = \frac{|W|}{|B+W|} = \frac{|\sum_{l=1}^g \sum_{j=1}^{n_l} (x_{lj} - \bar{x}_l)(x_{lj} - \bar{x}_l)^T|}{|\sum_{l=1}^g \sum_{j=1}^{n_l} (x_{lj} - \bar{x})(x_{lj} - \bar{x})^T|}$$

假设 $H_0: \mu_1 = \mu_2 = \dots = \mu_g$, 则当 H_0 为真且 $\sum n_l = n$ 充分大, 则

$$-\left(n-1-\frac{(p+g)}{2}\right) \ln \Lambda^* = -\left(n-1-\frac{(p+g)}{2}\right) \ln \left(\frac{|W|}{|B+W|}\right)$$

近似服从自由度为 $p(g-1)$ 的 χ^2 分布, 因此, 当 $\sum n_l = n$ 充分大时, 若

$$-\left(n-1-\frac{(p+g)}{2}\right) \ln \left(\frac{|W|}{|B+W|}\right) > \chi_{p(g-1)}^2(\alpha)$$

我们就以显著性水平 α 拒绝 H_0 , 其中的 $\chi_{p(g-1)}^2(\alpha)$ 为自由度为 $p(g-1)$ 的 χ^2 分布的上100 α 百分位数。

2.2 代码

```
from scipy.stats import norm, chi2
import numpy as np
import random
import matplotlib.pyplot as plt
import os

# 配置matplotlib以支持中文显示
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 数据生成器
def generate_dataset(count, distribution_type=None):
    if distribution_type == '正态分布':
        mean, std_dev = 0, 0.2
        return np.random.normal(loc=mean, scale=std_dev, size=count)
    elif distribution_type == '均匀分布':
        lower_bound, upper_bound = -1, 1
        return np.random.uniform(low=lower_bound, high=upper_bound, size=count)
    elif distribution_type == '泊松分布':
        lambda_param = 4
        return np.random.poisson(lam=lambda_param, size=count)
    elif distribution_type == '指数分布':
        scale_factor = 6
        return np.random.exponential(scale=scale_factor, size=count)
    elif distribution_type == 't-分布':
        degrees_of_freedom = 3
        return np.random.standard_t(df=degrees_of_freedom, size=count)
    elif distribution_type == '卡方分布':
        df_random = 5
        return np.random.chisquare(df=df_random, size=count)
    else:
```

```

        raise ValueError('未知分布类型')

# 计算统计量
def calculate_ld_statistic(dataset, parameters_count, group_count, sample_size):
    mean_per_group = np.mean(dataset, axis=2)
    overall_mean = np.mean(mean_per_group, axis=0)

    # 计算B
    delta_matrix_1 = (mean_per_group - overall_mean[None, :])[:, :, np.newaxis]
    delta_matrix_2 = (mean_per_group - overall_mean[None, :])[:, np.newaxis, :]
    B_matrix = (delta_matrix_1 @ delta_matrix_2).sum(axis=0) * sample_size

    # 计算W
    delta_data_1 = (dataset - mean_per_group[:, :, np.newaxis]).transpose(0, 2, 1)[:, :, :, np.newaxis]
    delta_data_2 = (dataset - mean_per_group[:, :, np.newaxis]).transpose(0, 2, 1)[:, :, np.newaxis, :]
    W_matrix = (delta_data_1 @ delta_data_2).sum(axis=(0, 1))

    # 计算Lambda_star
    det_W = np.linalg.det(W_matrix)
    det_B_plus_W = np.linalg.det(B_matrix + W_matrix)
    lambda_star = det_W / det_B_plus_W

    # 计算统计量
    n = group * sample_size
    test_statistic = -(n - 1 - (parameters_count + group_count) / 2) * np.log(lambda_star)
    return test_statistic

# 计算数据的累积分布概率
def cumulative_distribution_probabilities(data_points):
    ordered_data = np.sort(data_points)
    position_indices = np.arange(1, len(ordered_data) + 1)
    probabilities = (position_indices - 0.5) / len(ordered_data)
    return ordered_data, probabilities

# 绘制Q-Q图
def draw_qq_diagram(data_points, parameters, groups):
    degrees_of_freedom = parameters * (groups - 1)
    sorted_data, prob = cumulative_distribution_probabilities(data_points)
    normal_quantiles = chi2.ppf(prob, df=degrees_of_freedom)
    plt.scatter(sorted_data, normal_quantiles)
    plt.title(f'Q-Q图 参数p={parameters}, 组数g={groups}')
    plt.savefig(f"images/qq_plot_p_{parameters}_g_{groups}.png")
    plt.clf()

# 绘制直方图与理论分布叠加图
def overlay_histogram_and_pdf(data_points, parameters, groups):
    plt.hist(data_points, bins=60, density=True, alpha=0.6, color='blue', label='样本直方图')
    x_range = np.linspace(min(data_points), max(data_points), 10000)
    chi_squared_pdf = chi2.pdf(x_range, df=parameters * (groups - 1))
    plt.plot(x_range, chi_squared_pdf, 'red', label='卡方分布PDF')
    plt.xlabel('值')
    plt.ylabel('密度')
    plt.title(f'直方图与卡方分布 p={parameters}, g={groups}')
    plt.legend()
    plt.savefig(f"images/histogram_chi2_overlay_p_{parameters}_g_{groups}.png")
    plt.clf()

if __name__ == "__main__":
    parameter_group_combinations = [(3, 4), (4, 5), (6, 10), (8, 30)]
    sample_count = 1422
    iterations = 1422
    distribution_types = ['正态分布', '均匀分布', '泊松分布', '指数分布', 't-分布', '卡方分布']

    # 确保输出目录存在
    os.makedirs("images", exist_ok=True)

    for param, group in parameter_group_combinations:
        simulation_results = []
        for _ in range(iterations):
            all_data = []
            name_list = np.random.choice(distribution_types, size=param, replace=True)

```



```

for index in range(group):
    data_list = [generate_dataset(sample_count, name_list[idx])[None, :] for idx in range(param)]
    data = np.concatenate(data_list)
    all_data.append(data[None, :])

data = np.concatenate(all_data)

result = calculate_ld_statistic(data, param, group, sample_count)
simulation_results.append(result)

overlay_histogram_and_pdf(simulation_results, param, group)
draw_qq_diagram(simulation_results, param, group)

```

2.3 结果

结果如下所示:

