题干

1. 考虑如下概率图模型(图略):
   - 写出对应的因子图
   - 使用和积算法（sum-product algorithm）计算边缘概率 $p(x_2)$
2. 详细推导高斯混合模型（Gaussian mixture model）的 **E 步** 与 **M 步**
3. 使用平均场方法（mean field method）近似高斯混合模型的边缘概率 $p_\theta(X)$。
   变分分布为乘积形式:

$$q(Z,\pi,\mu_k,\Sigma_k)=q(Z)q(\pi)\prod_{k=1}^{K}q(\mu_k,\Sigma_k)$$

4. 编写一个简单的 **拒绝采样（rejection sampling）** Python 代码
5. 考虑一个转移矩阵为

$$P=\begin{bmatrix}p_{11} & p_{12} & p_{13}\\ p_{21} & p_{22} & p_{23}\\ p_{31} & p_{32} & p_{33}\end{bmatrix}$$

   的马尔可夫链。设 $\pi=(\pi_1,\pi_2,\pi_3)$ 是平稳分布，求 π 应满足的方程。
6. 为以下 MCMC 方法编写 Python 代码:
   (a) 基本 MCMC
   (b) 哈密顿蒙特卡洛（Hamiltonian MC）
   (c) 朗之万动力学（Langevin dynamics）
   用你的代码对一般的二维高斯分布进行采样，比较三种方法的结果（如接受率等）。
7. 证明: **吉布斯采样（Gibbs sampling）** 满足细致平衡条件（detailed balance condition）。

Solution of T1

因子图

$$p(Z_1,Z_2,Z_3,X_1,X_2,X_3)=p(Z_1)p(Z_2\mid Z_1)p(Z_3\mid Z_2)p(X_1\mid Z_1)p(X_2\mid Z_2)p(X_3\mid Z_3).$$

所以我们有 6 个因子（把每个括号内的项作为一个因子）:

$$f_{Z1}=p(Z_1),f_{Z2}=p(Z_2\mid Z_1),f_{Z3}=p(Z_3\mid Z_2),f_{X1}=p(X_1\mid Z_1),f_{X2}=p(X_2\mid Z_2),f_{X3}=p(X_3\mid Z_3).$$

因子图如下（小括号代表圆圈，中括号代表正方形）:

```
[f_Z1]—(Z1)—[f_Z2]—(Z2)—[f_Z3]—(Z3)
         |             |             |
       [f_X1]        [f_X2]        [f_X3]
         |             |             |
        (X1)          (X2)          (X3)
```

边际概率

叶节点的传播:

$$\mu_{X_1\to f_{X1}}(x_1)=1,\qquad \mu_{X_3\to f_{X3}}(x_3)=1.$$

$$\mu_{f_{X1}\to Z_1}(z_1)=\sum_{x_1}p(x_1|z_1)=1,\qquad \mu_{f_{X3}\to Z_3}(z_3)=\sum_{x_3}p(x_3|z_3)=1.$$

从左侧传来的部分:

$$\mu_{f_{Z1}\to Z_1}(z_1)=p(z_1).$$

$$\mu_{Z_1\to f_{Z2}}(z_1)=p(z_1)\cdot 1=p(z_1).$$

$$\mu_{f_{Z2}\to Z_2}(z_2)=\sum_{z_1}p(z_2|z_1)p(z_1)=p(z_2).$$

从右侧传来的部分:

$$\mu_{Z_3\to f_{Z3}}(z_3)=1.$$

$$\mu_{f_{Z3}\to Z_2}(z_2)=\sum_{z_3}p(z_3|z_2)=1.$$

Z_2的总消息:

$$\mu_{Z_2\to f_{X2}}(z_2)=\mu_{f_{Z2}\to Z_2}(z_2)\cdot\mu_{f_{Z3}\to Z_2}(z_2)=p(z_2)\cdot 1=p(z_2).$$

从而X_2的边缘概率:

$$p(x_2)=\mu_{f_{X2}\to X_2}(x_2)=\sum_{z_2}p(x_2|z_2)p(z_2).$$

其中

$$p(z_2)=\sum_{z_1}p(z_1)p(z_2|z_1).$$

即:

$$p(x_2)=\sum_{z_2}p(x_2\mid z_2)\left(\sum_{z_1}p(z_1)p(z_2\mid z_1)\right)$$

Solution of T2

回忆GMM模型的设定：假设我们有N个观测数据 $x_1,x_2,\ldots,x_N$，假设它们独立同分布，服从K个高斯分布的混合：

$$p(x_n\mid\Theta)=\sum_{k=1}^{K}\pi_k\mathcal{N}(x_n\mid\mu_k,\Sigma_k)$$

其中：

- π_k 是混合权重，满足 $\sum_k\pi_k=1$ 且 $\pi_k\geq 0$
- μ_k 和 Σ_k 分别是第k个高斯的均值和协方差矩阵
- $\Theta=\{\pi_k,\mu_k,\Sigma_k\}_{k=1}^{K}$ 是所有参数

隐变量 z_n表示第n个样本属于哪一类：

$$z_n=[z_{n1},z_{n2},\ldots,z_{nK}],\quad z_{nk}\in 0,1,\quad\sum_k z_{nk}=1$$

则complete-data likelihood为：

$$p(\{x_n,z_n\}_{n=1}^{N}\mid\Theta)=\prod_{n=1}^{N}\prod_{k=1}^{K}\left[\pi_k\mathcal{N}(x_n\mid\mu_k,\Sigma_k)\right]^{z_{nk}}$$

对数似然：

$$\ln p(X,Z\mid\Theta)=\sum_{n=1}^{N}\sum_{k=1}^{K}z_{nk}\Big[\ln\pi_k+\ln\mathcal{N}(x_n\mid\mu_k,\Sigma_k)\Big]$$

E-Step

在E-step，我们希望计算隐变量的后验期望

$$\gamma_{nk}\equiv\mathbb{E}[z_{nk}\mid x_n,\Theta^{\text{old}}]=p(z_{nk}=1\mid x_n,\Theta^{\text{old}})$$

根据贝叶斯公式：

$$\gamma_{nk} = p(z_{nk} = 1 \mid x_n, \Theta^{\text{old}}) = \frac{\pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n \mid \mu_j, \Sigma_j)}$$

M-Step

在M-step，我们希望最大化期望对数似然

定义Q 函数：

$$Q(\Theta, \Theta^{\text{old}}) = \mathbb{E}_{Z|X,\Theta^{\text{old}}}[\ln p(X, Z \mid \Theta)] = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} \Big[\ln \pi_k + \ln \mathcal{N}(x_n \mid \mu_k, \Sigma_k) \Big]$$

以下分别对 π_k、μ_k、Σ_k 求最大化。

由于有约束条件 $\sum_k \pi_k = 1$。引入拉格朗日乘子：

$$\mathcal{L} = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} \ln \pi_k + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right)$$

Q 对 π_k 求导导数为 0：

$$\frac{\partial \mathcal{L}}{\partial \pi_k} = \sum_{n=1}^{N} \frac{\gamma_{nk}}{\pi_k} + \lambda = 0 \quad \Rightarrow \quad \pi_k = -\frac{1}{\lambda} \sum_{n=1}^{N} \gamma_{nk}$$

利用约束 $\sum_k \pi_k = 1$：

$$\sum_k \pi_k = -\frac{1}{\lambda} \sum_k \sum_n \gamma_{nk} = -\frac{1}{\lambda} \sum_n \sum_k \gamma_{nk} = -\frac{1}{\lambda} N = 1 \quad \Rightarrow \lambda = -N$$

所以更新公式：

$$\pi_k^{\text{new}} = \frac{1}{N} \sum_{n=1}^{N} \gamma_{nk}$$

因为：

$$\ln \mathcal{N}(x_n \mid \mu_k, \Sigma_k) = -\frac{1}{2} (x_n - \mu_k)^T \Sigma_k^{-1} (x_n - \mu_k) + \text{const}$$

因此 Q 对 μ_k 求导并令导数为 0：

$$\sum_{n=1}^{N} \gamma_{nk} \Sigma_k^{-1} (x_n - \mu_k) = 0 \quad \Rightarrow \quad \mu_k^{\text{new}} = \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{\sum_{n=1}^{N} \gamma_{nk}}$$

Q 对 Σ_k 进行矩阵求导并令导数为 0：考虑到 Σ_k 是对称正定矩阵，使用矩阵求导公式 $\frac{\partial \ln |\Sigma|}{\partial \Sigma} = (\Sigma^{-1})^T = \Sigma^{-1}$，$\frac{\partial}{\partial \Sigma} \left[x^T \Sigma^{-1} x \right] = -\Sigma^{-1} x x^T \Sigma^{-1}$，因此求导结果如下：

$$\frac{\partial L}{\partial \Sigma_k} = -\frac{1}{2} \sum_{n=1}^{N} \gamma_{nk} \left[\Sigma_k^{-1} - \Sigma_k^{-1} (x_n - \mu_k)(x_n - \mu_k)^T \Sigma_k^{-1} \right] = 0$$

化简后解得：

$$\Sigma_k^{\text{new}} = \frac{\sum_{n=1}^{N} \gamma_{nk} (x_n - \mu_k^{\text{new}})(x_n - \mu_k^{\text{new}})^T}{\sum_{n=1}^{N} \gamma_{nk}}$$

综上所述：

E 步：

$$\gamma_{nk} = \frac{\pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n \mid \mu_j, \Sigma_j)}$$

M 步：

$$
\begin{aligned}
\pi_k^{\text{new}} &= \frac{1}{N}\sum_{n=1}^{N}\gamma_{nk} \\
\mu_k^{\text{new}} &= \frac{\sum_{n=1}^{N}\gamma_{nk}x_n}{\sum_{n=1}^{N}\gamma_{nk}} \\
\Sigma_k^{\text{new}} &= \frac{\sum_{n=1}^{N}\gamma_{nk}(x_n-\mu_k^{\text{new}})(x_n-\mu_k^{\text{new}})^T}{\sum_{n=1}^{N}\gamma_{nk}}
\end{aligned}
$$

Solution of T3

整体想法是用ELBO近似 $\log p_\theta(X)$，于是 $p_\theta(X)\approx\exp\{ELBO\}$。

高斯混合模型的联合分布为：

$$p_\theta(X,Z,\pi,\mu,\Sigma)=p(Z|\pi)p(X|Z,\mu,\Sigma)p(\pi)p(\mu,\Sigma)$$

其中 Z 为离散隐变量，π 为混合权重，μ_k,Σ_k 为第k个高斯分量的参数。依题意，变分分布为：

$$q(Z,\pi,\mu_k,\Sigma_k)=q(Z)q(\pi)\prod_{k=1}^{K}q(\mu_k,\Sigma_k)$$

因此ELBO的表达式为：

$$
\begin{aligned}
\mathcal{L}(q) =&\mathbb{E}_q\left[\log\frac{p_\theta(X,Z,\pi,\mu,\Sigma)}{q(Z,\pi,\mu,\Sigma)}\right] \\
=&\mathbb{E}_q[\log p(X|Z,\mu,\Sigma)]+\mathbb{E}_q[\log p(Z|\pi)]+\mathbb{E}_q[\log p(\pi)]+\mathbb{E}_q[\log p(\mu,\Sigma)] \\
&-\mathbb{E}_q[\log q(Z)]-\mathbb{E}_q[\log q(\pi)]-\sum_{k=1}^{K}\mathbb{E}_q[\log q(\mu_k,\Sigma_k)]
\end{aligned}
$$

以下用EM algorithm，固定其他变分因子，分别优化每个因子。先更新 $q(Z)$：

$$\log q^*(Z)=\mathbb{E}_{q(\pi)q(\mu,\Sigma)}[\log p(X,Z,\pi,\mu,\Sigma)]+\text{const}$$

$$q^*(Z)=\prod_{n=1}^{N}\prod_{k=1}^{K}r_{nk}^{z_{nk}}$$

其中：

$$r_{nk}\propto\exp\left(\mathbb{E}[\log\pi_k]-\frac{1}{2}\mathbb{E}[(\mathbf{x}_n-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}_n-\mu_k)]-\frac{1}{2}\mathbb{E}[\log|\Sigma_k|]\right)$$

再更新 $q(\pi)$：

$$\log q^*(\pi)=\mathbb{E}_{q(Z)}[\log p(Z|\pi)]+\log p(\pi)+\text{const}$$

这里的先验通常取Dirichlet分布：$p(\pi)=\text{Dir}(\pi|\alpha)$，此时：

$$q^*(\pi)=\text{Dir}(\pi|\alpha^*),\quad \alpha_k^*=\alpha_k+\sum_{n=1}^{N}\mathbb{E}[z_{nk}]$$

再更新 $q(\mu_k,\Sigma_k)$：

$$\log q^*(\mu_k,\Sigma_k)=\mathbb{E}_{q(Z)}[\log p(X|Z,\mu_k,\Sigma_k)]+\log p(\mu_k,\Sigma_k)+\text{const}$$

这里的先验通常取高斯-逆Wishart先验，此时 $q(\mu_k,\Sigma_k)$ 也是高斯-逆Wishart分布。

重复直到收敛，最后用最终的 $\mathcal{L}(q)$ 近似 $\log p_\theta(X)$，即

$$p_\theta(X)\approx\exp(\mathcal{L}(q))$$

其中 $\mathcal{L}(q)$ 是收敛后的变分下界值。

Solution of T4

以下以生成(-5,5)上的服从 $f(x)=\frac{1}{\sqrt{2\pi}}e^{-x^2/2}$的随机变量为例, 代码如下:

```python
import numpy as np

def simple_rejection_sampling():
    def complex_target(x):
        # 这里写出pdf的表达式
        return (1 / np.sqrt(2 * np.pi)) * np.exp(-x**2 / 2)

    n_samples = 1000
    samples = []

    # 确定M
    x_test = np.linspace(-5, 5, 1000)
    max_value = np.max(complex_target(x_test))
    M = max_value + 0.1

    while len(samples) < n_samples:
        x_proposal = np.random.uniform(-5, 5)
        u = np.random.uniform(0, 1)
        acceptance_prob = complex_target(x_proposal) / M

        if u < acceptance_prob:
            samples.append(x_proposal)

    return samples

# 生成随机样本
simple_samples = simple_rejection_sampling()
print(f"\n简单示例生成了 {len(simple_samples)} 个样本")
```

Solution of T5

由于 $\pi=(\pi_1,\pi_2,\pi_3)$ 是一个平稳分布，因此：

$$\pi P=\pi \quad \pi_1+\pi_2+\pi_3=1, \quad \pi_i\geq 0$$

所以：

$$[\pi_1,\pi_2,\pi_3]\begin{bmatrix}p_{11} & p_{12} & p_{13}\\ p_{21} & p_{22} & p_{23}\\ p_{31} & p_{32} & p_{33}\end{bmatrix}=[\pi_1,\pi_2,\pi_3]$$

即:

$$\begin{bmatrix}p_{11}-1 & p_{21} & p_{31}\\ p_{12} & p_{22}-1 & p_{32}\\ p_{13} & p_{23} & p_{33}-1\end{bmatrix}\begin{bmatrix}\pi_1\\ \pi_2\\ \pi_3\end{bmatrix}=0 \qquad \cdots \quad (*)$$

因为 P 是随机矩阵，有：

$$\begin{aligned}p_{11}+p_{12}+p_{13}=1\\ p_{21}+p_{22}+p_{23}=1\\ p_{31}+p_{32}+p_{33}=1\end{aligned}$$

故在$(*)$两端左乘如下矩阵:

$$\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 1 & 1 & 1\end{bmatrix}$$

得到:

$$\begin{bmatrix}p_{11}-1 & p_{21} & p_{31}\\ p_{12} & p_{22}-1 & p_{32}\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\pi_1\\ \pi_2\\ \pi_3\end{bmatrix}=0$$

即满足两个方程, 再加上方程 $\pi_1+\pi_2+\pi_3=1$, 即可得到 π 满足的方程组:

$$\begin{bmatrix}p_{11}-1 & p_{21} & p_{31}\\ p_{12} & p_{22}-1 & p_{32}\\ 1 & 1 & 1\end{bmatrix}\begin{bmatrix}\pi_1\\ \pi_2\\ \pi_3\end{bmatrix}=0$$

或者等价地写成：

$$\begin{cases}\pi_1(p_{11}-1)+\pi_2p_{21}+\pi_3p_{31}=0\\ \pi_1p_{12}+\pi_2(p_{22}-1)+\pi_3p_{32}=0\\ \pi_1+\pi_2+\pi_3=1\end{cases}$$

Solution of T6

```python
import numpy as np
from scipy.stats import multivariate_normal

class BasicMCMC:
    """Basic MCMC method (Metropolis-Hastings)"""

    def __init__(self, target_dist, proposal_std=1.0):
        self.target_dist = target_dist
        self.proposal_std = proposal_std

    def sample(self, n_samples, initial_state, burn_in=1000):
        current_state = initial_state
        samples = []
        accepted = 0

        for i in range(n_samples + burn_in):
            # Generate candidate sample (normal distribution proposal)
            proposal = current_state + np.random.normal(0, self.proposal_std, size=current_state.shape)

            # Calculate acceptance probability
            current_prob = self.target_dist(current_state)
            proposal_prob = self.target_dist(proposal)
            acceptance_ratio = min(1, proposal_prob / current_prob)

            # Accept or reject
            if np.random.rand() < acceptance_ratio:
                current_state = proposal
                if i >= burn_in:
                    accepted += 1

            if i >= burn_in:
                samples.append(current_state.copy())

        acceptance_rate = accepted / n_samples
        return np.array(samples), acceptance_rate

class HamiltonianMC:
    """Hamiltonian Monte Carlo method"""

    def __init__(self, target_dist, step_size=0.1, n_leapfrog=10, mass=1.0):
        self.target_dist = target_dist
        self.step_size = step_size
        self.n_leapfrog = n_leapfrog
        self.mass = mass

    def kinetic_energy(self, momentum):
        """Kinetic energy"""
        return 0.5 * np.sum(momentum**2) / self.mass

    def potential_energy(self, position):
        """Potential energy (negative log probability)"""
        return -np.log(self.target_dist(position) + 1e-10)

    def hamiltonian(self, position, momentum):
        """Hamiltonian"""
        return self.potential_energy(position) + self.kinetic_energy(momentum)

    def leapfrog_step(self, position, momentum):
        """Leapfrog integration step"""
        # Half step update momentum
        grad = self.numerical_gradient(position)
        momentum = momentum - 0.5 * self.step_size * grad
        # Full step update position
        position = position + self.step_size * momentum / self.mass

        # Half step update momentum
        grad = self.numerical_gradient(position)
        momentum = momentum - 0.5 * self.step_size * grad

        return position, momentum

    def numerical_gradient(self, position, eps=1e-6):
        """Numerical gradient calculation"""
        grad = np.zeros_like(position)
        for i in range(len(position)):
            pos_plus = position.copy()
            pos_minus = position.copy()
            pos_plus[i] += eps
            pos_minus[i] -= eps
            grad[i] = (self.potential_energy(pos_plus) - self.potential_energy(pos_minus)) / (2 * eps)
        return grad

    def sample(self, n_samples, initial_state, burn_in=1000):
        current_state = initial_state
        samples = []
        accepted = 0

        for i in range(n_samples + burn_in):
            # Sample momentum from normal distribution
            current_momentum = np.random.normal(0, np.sqrt(self.mass), size=current_state.shape)

            # Leapfrog integration to simulate trajectory
            proposed_state = current_state.copy()
            proposed_momentum = current_momentum.copy()

            for _ in range(self.n_leapfrog):
                proposed_state, proposed_momentum = self.leapfrog_step(proposed_state, proposed_momentum)

            # Calculate Hamiltonian
            current_hamiltonian = self.hamiltonian(current_state, current_momentum)
            proposed_hamiltonian = self.hamiltonian(proposed_state, -proposed_momentum)  # Momentum reversal

            # Acceptance probability
            acceptance_ratio = min(1, np.exp(current_hamiltonian - proposed_hamiltonian))

            # Accept or reject
            if np.random.rand() < acceptance_ratio:
                current_state = proposed_state
                if i >= burn_in:
                    accepted += 1

            if i >= burn_in:
                samples.append(current_state.copy())

        acceptance_rate = accepted / n_samples
        return np.array(samples), acceptance_rate

class LangevinDynamics:
    """Langevin Dynamics MCMC"""

    def __init__(self, target_dist, step_size=0.1, mass=1.0):
        self.target_dist = target_dist
        self.step_size = step_size
        self.mass = mass
    def potential_energy(self, position):
        """Potential energy (negative log probability)"""
        return -np.log(self.target_dist(position) + 1e-10)

    def numerical_gradient(self, position, eps=1e-6):
        """Numerical gradient calculation"""
        grad = np.zeros_like(position)
        for i in range(len(position)):
            pos_plus = position.copy()
            pos_minus = position.copy()
            pos_plus[i] += eps
            pos_minus[i] -= eps
            grad[i] = (self.potential_energy(pos_plus) - self.potential_energy(pos_minus)) / (2 * eps)
        return grad

    def sample(self, n_samples, initial_state, burn_in=1000):
        current_state = initial_state
        samples = []

        for i in range(n_samples + burn_in):
            # Langevin dynamics update
            grad = self.numerical_gradient(current_state)
            noise = np.random.normal(0, np.sqrt(2 * self.step_size), size=current_state.shape)

            current_state = current_state - self.step_size * grad + noise

            if i >= burn_in:
                samples.append(current_state.copy())

        # Langevin dynamics typically has high acceptance rate (close to 1)
        acceptance_rate = 1.0  # Simplified estimation
        return np.array(samples), acceptance_rate

# Testing and comparison
def main():
    # Set up target distribution: 2D Gaussian
    mean = np.array([1.0, -1.0])
    cov = np.array([[2.0, 0.8], [0.8, 1.5]])
    target_dist = multivariate_normal(mean, cov)

    def target_pdf(x):
        return target_dist.pdf(x)

    # Parameter settings
    n_samples = 5000
    burn_in = 1000
    initial_state = np.array([0.0, 0.0])

    # Basic MCMC
    print("Running Basic MCMC...")
    basic_mcmc = BasicMCMC(target_pdf, proposal_std=1.0)
    samples_basic, acc_basic = basic_mcmc.sample(n_samples, initial_state, burn_in)

    # Hamiltonian Monte Carlo
    print("Running Hamiltonian Monte Carlo...")
    hmc = HamiltonianMC(target_pdf, step_size=0.1, n_leapfrog=10)
    samples_hmc, acc_hmc = hmc.sample(n_samples, initial_state, burn_in)

    # Langevin Dynamics
    print("Running Langevin Dynamics...")
    langevin = LangevinDynamics(target_pdf, step_size=0.01)
    samples_langevin, acc_langevin = langevin.sample(n_samples, initial_state, burn_in)

    # Print acceptance rate comparison
    print("\n=== Acceptance Rate Comparison ===")
    print(f"Basic MCMC: {acc_basic:.3f}")
    print(f"HMC: {acc_hmc:.3f}")
    print(f"Langevin Dynamics: {acc_langevin:.3f}")

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

运行结果如下:

```
=== Acceptance Rate Comparison ===
Basic MCMC: 0.606
HMC: 1.000
Langevin Dynamics: 1.000
```

Solution of T7

设d维随机变量 $\mathbf{x} = (x_1, \dots, x_d)$，目标分布为 $\pi(\mathbf{x})$。Gibbs Sampling每一步更新一个坐标i，从条件分布 $\pi(x_i' \mid \mathbf{x}_{-i})$ 采样新值 x_i'，其中 $\mathbf{x}_{-i} = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_d)$。新状态为 $\mathbf{x}' = (x_1, \dots, x_{i-1}, x_i', x_{i+1}, \dots, x_d)$，满足 $\mathbf{x}_{-i} = \mathbf{x}'_{-i}$。转移概率 $P(\mathbf{x} \to \mathbf{x}') = \pi(x_i' \mid \mathbf{x}_{-i})$ 。下证其满足细致平衡条件：$\pi(\mathbf{x})P(\mathbf{x} \to \mathbf{x}') = \pi(\mathbf{x}')P(\mathbf{x}' \to \mathbf{x})$ 。注意到：

$$LHS = \pi(\mathbf{x})\pi(x_i' \mid \mathbf{x}_{-i}) = \frac{\pi(\mathbf{x})\pi(\mathbf{x}')}{\pi(\mathbf{x}_{-i})}$$

右边：

$$RHS = \pi(\mathbf{x}')\pi(x_i \mid \mathbf{x}'_{-i}) = \pi(\mathbf{x}')\pi(x_i \mid \mathbf{x}_{-i}) = \frac{\pi(\mathbf{x}')\pi(\mathbf{x})}{\pi(\mathbf{x}_{-i})}$$

因此 $LHS = RHS$，吉布斯采样满足细致平衡条件。