EM 算法的简易教程及应用

舒双林

2025 年 4 月 8 日

目录

1 EM 算法的理论基础

EM 算法是过去二十年，统计对人类科技发展的最大贡献。

——刘传海教授 (美国普渡大学)

1.1 什么是 EM 算法?

EM 算法（期望最大化算法，Expectation Maximization）是一种用于从不完全数据中估计参数的迭代方法。通过在每次迭代中交替进行期望（E 步）和最大化（M 步），该算法能够在给定观测数据和隐变量的条件下求解出模型的最大似然估计。

EM 算法最早由 Dempster 等人于 1977 年提出，并广泛应用于统计学、机器学习等领域，尤其在缺失数据和隐变量模型中具有重要应用。EM 算法的核心思想是通过迭代地估计隐变量的期望和更新模型参数，逐步逼近最大似然估计。具体步骤为:

1. E 步（期望步）: 在当前参数估计下，计算隐变量的期望。
2. M 步（最大化步）: 最大化期望函数，更新参数估计值。

通过交替进行 E 步和 M 步，EM 算法逐步逼近对数似然函数的最大值。

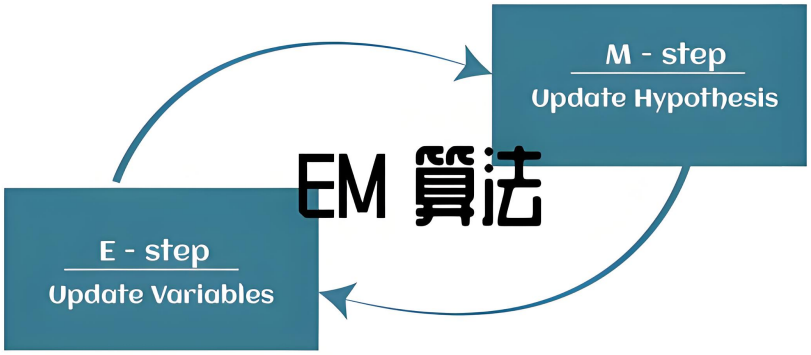

图 1: EM 算法

1.2 EM 算法的数学推导

设 Y 为观测随机变量的数据，Z 为隐变量的数据。Y 和 Z 组合起来称为完全数据，而观测数据 Y 称为不完全数据。假设给定观测数据 Y，其概率分布为 $P(Y|\theta)$，其中 θ 是需要估计的模型参数，那么不完全数据 Y 的似然函数为 $P(Y|\theta)$，对数似然函数为 $L(\theta) = \log P(Y|\theta)$。假设 Y 和 Z 的联合概率分布为 $P(Y, Z|\theta)$，那么完全数据 Y 和 Z 的对数似然函数为:

$$L(\theta) = \log P(Y, Z|\theta) \tag{1.1}$$

对于含有隐变量 Z 的概率模型，直接极大化对数似然函数 $L(\theta) = \log P(Y|\theta)$ 是困难的，因为其含有未观测数据，并且涉及求和（或积分）的对数。

假设在第 i 次迭代后，参数的估计值为 $\theta^{(i)}$。考虑新的估计值 θ 能否使 $L(\theta)$ 增加。对 $L(\theta)$ 和 $L(\theta^{(i)})$ 作差:

$$L(\theta) - L(\theta^{(i)}) = \log\left(\sum_Z P(Y|Z,\theta)P(Z|\theta)\right) - \log P(Y|\theta^{(i)}) \tag{1.2}$$

该公式表达了在不同参数下，对数似然函数的变化，用于说明参数更新的效果。

接下来，利用 Jensen 不等式[1]可得到该差值的下界:

$$\begin{aligned} L(\theta) - L(\theta^{(i)}) &= \log\left(\sum_Z P(Z|Y,\theta^{(i)})\frac{P(Y|Z,\theta)P(Z|\theta)}{P(Z|Y,\theta^{(i)})}\right) - \log P(Y|\theta^{(i)}) \\ &\geq \sum_Z P(Z|Y,\theta^{(i)})\log\frac{P(Y|Z,\theta)P(Z|\theta)}{P(Z|Y,\theta^{(i)})} - \log P(Y|\theta^{(i)}) \\ &= \sum_Z P(Z|Y,\theta^{(i)})\log\frac{P(Y|Z,\theta)P(Z|\theta)}{P(Z|Y,\theta^{(i)})P(Y|\theta^{(i)})} \end{aligned} \tag{1.3}$$

令

$$B(\theta,\theta^{(i)}) = L(\theta^{(i)}) + \sum_Z P(Z|Y,\theta^{(i)})\log\frac{P(Y|Z,\theta)P(Z|\theta)}{P(Z|Y,\theta^{(i)})P(Y|\theta^{(i)})} \tag{1.4}$$

则

$$L(\theta) \geq B(\theta,\theta^{(i)})$$

即函数 $B(\theta,\theta^{(n)})$ 是 $L(\theta)$ 的一个下界，而且由式(1.4)可知:

$$L(\theta^{(i)}) = B(\theta^{(i)},\theta^{(i)}) \tag{1.5}$$

因此，任何可以使 $B(\theta,\theta^{(i)})$ 增大的 θ，也可以使 $L(\theta)$ 增大。为了使 $L(\theta)$ 有尽可能大的增长，选择 $\theta^{(i+1)}$ 使 $B(\theta,\theta^{(i)})$ 达到极大，即

$$\theta^{(n+1)} = \arg\max_\theta B(\theta,\theta^{(n)}) \tag{1.6}$$

定义 1.1 (Q 函数). *设 $P(Y,Z|\theta)$ 为观测数据和隐变量的联合分布，$P(Z|Y,\theta^{(i)})$ 为在给定观测数据 Y 和当前参数估计 $\theta^{(i)}$ 下隐变量 Z 的条件概率分布，则定义函数 $Q(\theta,\theta^{(i)})$ 为:*

$$Q(\theta,\theta^{(i)}) = \sum_Z P(Z|Y,\theta^{(i)})\log P(Y,Z|\theta) \tag{1.7}$$

[1]Jensen 不等式指出，对于任意凹函数 f 和随机变量 X，有 $\mathbb{E}[f(X)] \leq f(\mathbb{E}[X])$。对于凸函数，不等号方向相反。

此处定义的 $Q(\theta, \theta^{(i)})$ 函数是 EM 算法的核心，为 EM 算法中 E 步和 M 步的关键。可以证明，最大化 $Q(\theta, \theta^{(i)})$ 将确保 $L(\theta)$ 不减。因此，EM 算法的迭代公式为：

$$\theta^{(i+1)} = \arg\max_{\theta} Q(\theta, \theta^{(i)}) \tag{1.8}$$

证明. 若去掉 θ 的极大化而言是常数的项，由式(1.6)和式(1.4)，有

$$\begin{aligned}
\theta^{(i+1)} &= \arg\max_{\theta} \left(L(\theta^{(i)}) + \sum_{Z} P(Z|Y, \theta^{(i)}) \log \frac{P(Y|Z, \theta) P(Z|\theta)}{P(Z|Y, \theta^{(i)}) P(Y|\theta^{(i)})} \right) \\
&= \arg\max_{\theta} \left(\sum_{Z} P(Z|Y, \theta^{(i)}) \log (P(Y|Z, \theta) P(Z|\theta)) \right) \\
&= \arg\max_{\theta} \left(\sum_{Z} P(Z|Y, \theta^{(i)}) \log P(Y, Z|\theta) \right) \\
&= \arg\max_{\theta} Q(\theta, \theta^{(i)})
\end{aligned} \tag{1.9}$$

□

式(1.9)等价于 EM 算法的一次迭代，即求 Q 函数及其极大化。EM 算法是通过不断求解下界的极大化逼近求解对数似然函数极大化的算法。

上述推导包含了 EM 算法的两个主要步骤：

1. E 步：计算 $Q(\theta, \theta^{(i)})$
2. M 步：求解 $\theta^{(i+1)} = \arg\max_{\theta} Q(\theta, \theta^{(i)})$

通过不断求解下界的极大化，EM 算法逐步逼近对数似然函数的极大值，但无法保证找到全局最优解。

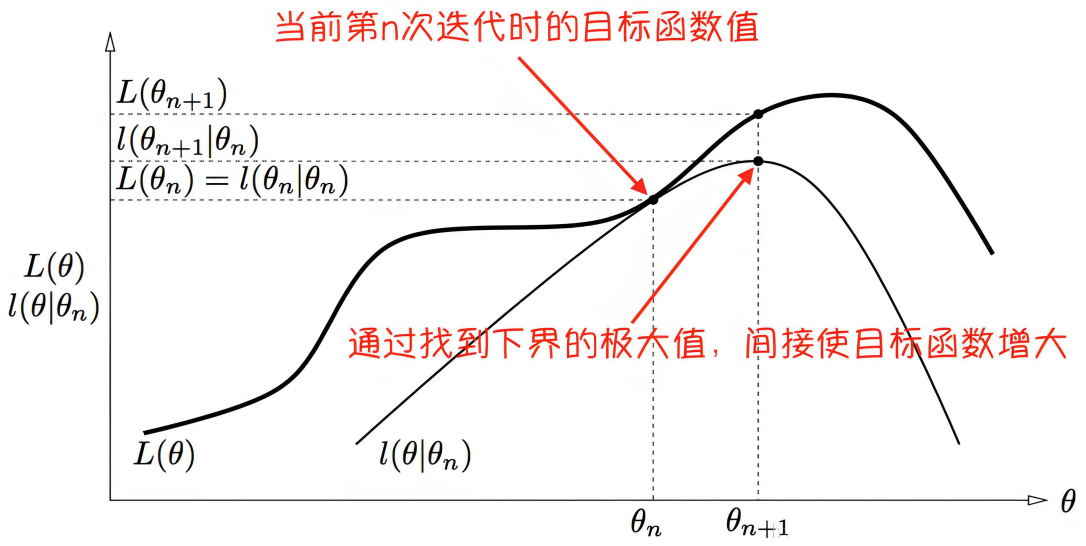

图 2: EM 算法示意图

1.3 EM 算法的收敛性

EM 算法的收敛性是通过以下定理来保证的:

Theorem 1 (单调性). *设 $P(Y|\theta)$ 为观测数据的似然函数，$\theta^{(i)}$ $(i=1,2,\ldots)$ 为 EM 算法得到的参数估计序列，则 $P(Y|\theta^{(i)})$ 是单调递增的，即:*

$$P(Y|\theta^{(i+1)}) \geq P(Y|\theta^{(i)}) \tag{1.10}$$

证明. 令

$$H(\theta,\theta^{(i)}) = \sum_Z \log P(Z|Y,\theta) P(Z|Y,\theta^{(i)}) \tag{1.11}$$

于是对数似然函数可以写成

$$\log P(Y|\theta) = Q(\theta,\theta^{(i)}) - H(\theta,\theta^{(i)}) \tag{1.12}$$

在式 (1.12) 中分别取 θ 为 $\theta^{(i)}$ 和 $\theta^{(i+1)}$ 并相减，有

$$\log P(Y|\theta^{(i+1)}) - \log P(Y|\theta^{(i)}) = \left[Q(\theta^{(i+1)},\theta^{(i)}) - Q(\theta^{(i)},\theta^{(i)})\right] - \left[H(\theta^{(i+1)},\theta^{(i)}) - H(\theta^{(i)},\theta^{(i)})\right] \tag{1.13}$$

为证式 (1.13)，只需证式右端是非负的。式 (1.13) 右端的第 1 项，由于 $\theta^{(i+1)}$ 使得 $Q(\theta,\theta^{(i)})$ 达到极大，所以有

$$Q(\theta^{(i+1)},\theta^{(i)}) - Q(\theta^{(i)},\theta^{(i)}) \geq 0 \tag{1.14}$$

其第 2 项，由式 (1.11) 可得:

$$H(\theta^{(i+1)},\theta^{(i)}) - H(\theta^{(i)},\theta^{(i)}) = \sum_Z \left(\log \frac{P(Z|Y,\theta^{(i+1)})}{P(Z|Y,\theta^{(i)})}\right) P(Z|Y,\theta^{(i)}) \tag{1.15}$$

由 Jensen 不等式，有

$$\sum_Z \left(\log \frac{P(Z|Y,\theta^{(i+1)})}{P(Z|Y,\theta^{(i)})}\right) P(Z|Y,\theta^{(i)}) \leq \log\left(\sum_Z \frac{P(Z|Y,\theta^{(i+1)})}{P(Z|Y,\theta^{(i)})} P(Z|Y,\theta^{(i)})\right) \tag{1.16}$$

因此

$$H(\theta^{(i+1)},\theta^{(i)}) - H(\theta^{(i)},\theta^{(i)}) \leq \log\left(\sum_Z P(Z|Y,\theta^{(i+1)})\right) = 0 \tag{1.17}$$

由式 (1.14) 和式 (1.17) 即知式 (1.13) 右端是非负的。

□

Theorem 2 (收敛性). *设 $L(\theta) = \log P(Y|\theta)$ 为观测数据的对数似然函数，$\theta^{(i)}(i=1,2,\ldots)$*

为 EM 算法得到的参数估计序列，$L(\theta^{(i)})$ $(i=1,2,\ldots)$ 为对应的对数似然函数序列。则：

1. 收敛性：如果 $P(Y|\theta)$ 有上界，则 $L(\theta^{(i)})$ 收敛到某一值 L^*。

2. 稳定点性质：在函数 $Q(\theta,\theta')$ 与 $L(\theta)$ 满足一定条件下，由 EM 算法得到的参数估计序列 $\theta^{(i)}$ 的收敛值 θ^* 是 $L(\theta)$ 的稳定点。

证明. (1) 由 $L(\theta)=\log P(Y|\theta^{(i)})$ 的单调性及 $P(Y|\theta)$ 的有界性立即得到。

(2) 设 $L(\theta)$ 的极大值点为 θ^*，则有

$$\begin{aligned} L(\theta^{(i)})-L(\theta^*) &= Q(\theta^{(i)},\theta^*)-H(\theta^{(i)},\theta^*) \le Q(\theta^{(i)},\theta^{(i)})-H(\theta^{(i)},\theta^*) \\ &\le Q(\theta^{(i)},\theta^{(i)})-H(\theta^{(i)},\theta^{(i)}) = L(\theta^{(i)})-L(\theta^{(i)}) = 0 \end{aligned} \tag{1.18}$$

由此可知，$L(\theta^{(i)})$ 收敛到 $L(\theta^*)$，即 $\theta^{(i)}$ 收敛到 θ^*。由于 $L(\theta)$ 的极大值点是唯一的，所以 θ^* 是 $L(\theta)$ 的稳定点。

□

1.4 EM 算法流程

算法 1 EM 算法

输入: 观测变量数据 Y、隐变量数据 Z、联合分布 $P(Y,Z|\theta)$、条件分布 $P(Z|Y,\theta)$

过程:

1: 选择参数初值 $\theta^{(0)}$、最大迭代次数 N、迭代精度 δ，开始迭代

2: **for** $i=1$ to N **do**

3: 令 $\theta^{(i-1)}$ 为第 $i-1$ 次迭代参数 θ 的估计值

4: E-step: 计算在给定观测数据 Y 和当前参数 $\theta^{(i-1)}$ 下 Z 的条件概率分布期望

$$Q(\theta,\theta^{(i-1)})=\sum_Z \log P(Y,Z|\theta)P(Z|Y,\theta^{(i-1)}) \tag{1.19}$$

5: M-step: 极大化 $Q(\theta,\theta^{(i-1)})$，确定第 i 次迭代的参数估计值 $\theta^{(i)}$

$$\theta^{(i)}=\arg\max_\theta Q(\theta,\theta^{(i-1)}) \tag{1.20}$$

6: 计算 $\theta^{(i-1)}$ 与 $\theta^{(i)}$ 的差值的二范数 $\delta^{(i)}=||\theta^{(i)}-\theta^{(i-1)}||$

7: **if** $\delta^{(i)}<\delta$ **then**

8: 迭代结束，$\hat{\theta}=\theta^{(i)}$ 为参数的极大似然估计值

9: **else**

10: 继续迭代，$i=i+1$，返回第 3 步

11: 迭代结束，$\hat{\theta}=\theta^{(N)}$ 为参数的极大似然估计值

输出: 模型的参数估计值 $\hat{\theta}$

关于 EM 算法的实施过程，需要注意以下几点：

1. 参数的初值选择可以任意，但需注意 EM 算法对初值敏感性较高，不同的初值可能导致不同的收敛结果。

2. 迭代停止的条件可以基于参数变化或 Q 函数增益，即：

$$\|\theta^{(i+1)} - \theta^{(i)}\| < \delta_1 \quad 或 \quad \|Q(\theta^{(i+1)}, \theta^{(i)}) - Q(\theta^{(i)}, \theta^{(i)})\| < \delta_2 \tag{1.21}$$

3. M 步求解 $Q(\theta, \theta^{(i)})$ 的极大化，得到 $\theta^{(i+1)}$，完成一次迭代 $\theta^{(i)} \to \theta^{(i+1)}$。前述的单调性和收敛性定理保证了 EM 算法的收敛性。

1.5 EM 算法的应用场景

EM 算法的优势在于能够有效地从不完全数据中估计参数，且实现相对简单。该算法被广泛应用于统计学、机器学习、数据挖掘等领域，尤其在处理缺失数据和隐变量模型时具有重要作用。常见的应用场景包括：

- 高斯混合模型 (GMM)：对多模态数据进行建模和聚类
- 隐马尔可夫模型 (HMM)：时序数据分析与模式识别
- 潜在语义分析 (LSA)：文本挖掘与主题建模
- 概率主成分分析 (PPCA)：降维与特征提取

这些应用场景共同点在于均存在隐变量或缺失数据，需要通过迭代方法进行参数估计。

2 EM 算法的应用

EM 算法广泛应用于统计学、机器学习以及数据科学等领域，尤其在缺失数据、隐变量模型、聚类分析等问题中展现了其强大的能力。以下将介绍 EM 算法在几个经典应用中的具体实现与分析。

2.1 硬币问题

2.1.1 问题描述

假设有两枚不规则硬币 A 和 B，硬币 A 正面出现的概率为 p，硬币 B 正面出现的概率为 q。随机选择其中一枚硬币，并投掷 10 次，共进行 5 轮，记录如下数据，其中"H"表示正面，"T" 表示反面。任务是估计硬币 A 和硬币 B 正面出现的概率 p 和 q。

表 1: 硬币投掷结果

	coin	1	2	3	4	5	6	7	8	9	10	result
1	?	H	T	T	T	H	H	T	H	T	H	5H5T
2	?	H	H	H	H	T	H	H	H	H	H	9H1T
3	?	H	T	H	H	H	H	H	T	H	H	8H2T
4	?	H	T	H	T	T	T	H	H	T	T	4H6T
5	?	T	H	H	H	T	H	H	H	T	H	7H3T

2.1.2 EM 算法求解硬币问题

为了估计硬币 A 和 B 的正面概率 p 和 q，应用 EM 算法进行求解。EM 算法通过交替进行 E 步和 M 步来估计这两个参数。

在 E 步，根据当前参数估计，计算出每轮投掷中选择硬币 A 或硬币 B 的概率。M 步则基于 E 步计算得到的概率，更新硬币 A 和硬币 B 的正面概率 p 和 q。

通过不断迭代 E 步和 M 步，最终得到收敛的参数估计值。

EM 算法的具体步骤如下：

1. 初始化参数 $p^{(0)} = 0.6, q^{(0)} = 0.5$

2. E 步：

   - 计算隐变量的期望

$$
\begin{aligned}
p_1^{(1)} &= P(Z_1 = A|X, p^{(0)}, q^{(0)}) = \frac{p^5(1-p)^5}{p^5(1-p)^5 + q^5(1-q)^5} \approx 0.4491 \\
q_1^{(1)} &= P(Z_1 = B|X, p^{(0)}, q^{(0)}) = 1 - P(Z_1 = A|X, p^{(0)}, q^{(0)}) \approx 0.5509 \\
&\vdots \\
p_5^{(1)} &= P(Z_5 = A|X, p^{(0)}, q^{(0)}) = \frac{p^7(1-p)^3}{p^7(1-p)^3 + q^7(1-q)^3} \approx 0.6472 \\
q_5^{(1)} &= P(Z_5 = B|X, p^{(0)}, q^{(0)}) = 1 - P(Z_5 = A|X, p^{(0)}, q^{(0)}) \approx 0.3528
\end{aligned}
$$

   - 计算 A、B 硬币的正反面次数的期望并更新参数

$$
\begin{aligned}
E_A^{(1)}(H) &= p_1^{(1)} \times 5 + p_2^{(1)} \times 9 + \cdots + p_5^{(1)} \times 7 = 21.2975 \\
E_A^{(1)}(T) &= p_1^{(1)} \times 5 + p_2^{(1)} \times 1 + \cdots + p_5^{(1)} \times 3 = 8.5722 \\
E_B^{(1)}(H) &= q_1^{(1)} \times 5 + q_2^{(1)} \times 9 + \cdots + q_5^{(1)} \times 7 = 11.7025 \\
E_B^{(1)}(T) &= q_1^{(1)} \times 5 + q_2^{(1)} \times 1 + \cdots + q_5^{(1)} \times 3 = 8.4278
\end{aligned}
$$

3. M 步：计算参数的极大似然估计

$$p^{(1)} = \frac{E_A^{(1)}(H)}{E_A^{(1)}(H) + E_A^{(1)}(T)} = \frac{21.2975}{21.2975 + 8.5722} \approx 0.7130$$
$$q^{(1)} = \frac{E_B^{(1)}(H)}{E_B^{(1)}(H) + E_B^{(1)}(T)} = \frac{11.7025}{11.7025 + 8.4278} \approx 0.5813$$

4. 重复 E 步和 M 步，直到收敛，最终得到结果：

$$\hat{p} = p^{(11)} = 0.7968, \hat{q} = q^{(11)} = 0.5196$$

2.1.3 结果分析

通过 EM 算法的迭代过程，最终获得了硬币 A 和硬币 B 正面出现的概率估计。最终的结果为：

$$\hat{p} = p^{(11)} = 0.7968, \hat{q} = q^{(11)} = 0.5196$$

表 2: EM 算法迭代过程 (收敛阈值 $\delta = 10^{-4}$)

迭代次数	p 值	Δp	q 值	Δq
0	0.6000	–	0.5000	–
1	0.7130	0.1130	0.5813	0.0813
2	0.7453	0.0323	0.5693	-0.0120
⋮	⋮	⋮	⋮	⋮
10	0.7967	0.0001	0.5197	-0.0001
11	0.7968	0.0000	0.5196	-0.0000

这些结果表明，硬币 A 的正面概率较高，而硬币 B 的正面概率较低。随着迭代次数的增加，参数估计逐渐收敛，EM 算法成功地从不完全的投掷数据中估计出硬币的正面概率。这表明 EM 算法在处理缺失数据和隐变量问题时能够高效且准确地给出估计值。

本案例的源代码见附录A.1。

2.2 桦尺蛾问题

2.2.1 背景介绍

桦尺蛾是一个典型的自然选择和工业污染的进化案例。这些蛾子的颜色由一个基因决定，存在三个可能的等位基因：C、I 和 T。C 是显性基因，T 是隐性基因，而 I 是对 C 显性、对 T 隐性的基因。桦尺蛾的表现型包括：

- **Carbonaria 型**：具有纯黑的颜色（由 CC、CI 和 CT 基因型表现）。

- **Typica 型**：具有浅色的有图案的翅膀（由 TT 基因型表现）。
- **Insularia 型**：颜色中等，通常表现为斑驳的中等色（由 II 和 IT 基因型表现）。

图 3: 桦尺蛾的不同表现型

由于环境污染的影响，Carbonaria 型蛾子数量显著增加，因此需要估计三种等位基因 C、I 和 T 的频率，进而理解基因型的变化。

2.2.2 EM 算法估计基因频率

桦尺蛾有六种可能的基因型，但现场调查中只能够测量三种表现型。

- **基因型 CC、CI、CT**：表现为 Carbonaria 表现型（黑色）
- **基因型 II、IT**：表现为 Insularia 表现型（中等颜色，斑驳）
- **基因型 TT**：表现为 Typica 表现型（浅色）

为了从仅有的表现型数据中估计等位基因频率，需要应用 EM 算法。其中 E 步计算在已知表现型数据的情况下，各基因型的期望计数，而 M 步则根据 E 步的期望更新基因频率。

EM 算法步骤：

1. 初始化等位基因频率
   - 假设初始等位基因频率为 $p_C^{(0)}$、$p_I^{(0)}$、$p_T^{(0)}$，满足 $p_C^{(0)} + p_I^{(0)} + p_T^{(0)} = 1$。
2. E 步
   - 在已知表现型数据的情况下，估计每种基因型的期望计数。

- 对于碳化表现型（carbonaria），可能的基因型为 CC、CI、CT。计算每种基因型的概率：

$$P(CC|\text{carbonaria}) = \frac{p_C^2}{p_C^2 + 2p_Cp_I + 2p_Cp_T}$$
$$P(CI|\text{carbonaria}) = \frac{2p_Cp_I}{p_C^2 + 2p_Cp_I + 2p_Cp_T}$$
$$P(CT|\text{carbonaria}) = \frac{2p_Cp_T}{p_C^2 + 2p_Cp_I + 2p_Cp_T}$$

- 使用概率计算期望基因型计数（其他基因型类似）：

$$n_{CC}^{(t)} = n_C \cdot P(CC|\text{carbonaria})$$
$$n_{CI}^{(t)} = n_C \cdot P(CI|\text{carbonaria})$$
$$n_{CT}^{(t)} = n_C \cdot P(CT|\text{carbonaria})$$

3. M 步：使用 E 步得到的期望基因型计数，更新等位基因频率：

$$p_C^{(t+1)} = \frac{2n_{CC}^{(t)} + n_{CI}^{(t)} + n_{CT}^{(t)}}{2n}$$
$$p_I^{(t+1)} = \frac{2n_{II}^{(t)} + n_{CI}^{(t)} + n_{IT}^{(t)}}{2n}$$
$$p_T^{(t+1)} = \frac{2n_{TT}^{(t)} + n_{CT}^{(t)} + n_{IT}^{(t)}}{2n}$$

4. 迭代：重复 E 步和 M 步，直到等位基因频率收敛。

2.2.3 结果分析

经过多次迭代，EM 算法成功估计出三种等位基因的频率，最终结果为：

$$p_C = 0.4, \quad p_I = 0.3, \quad p_T = 0.3$$

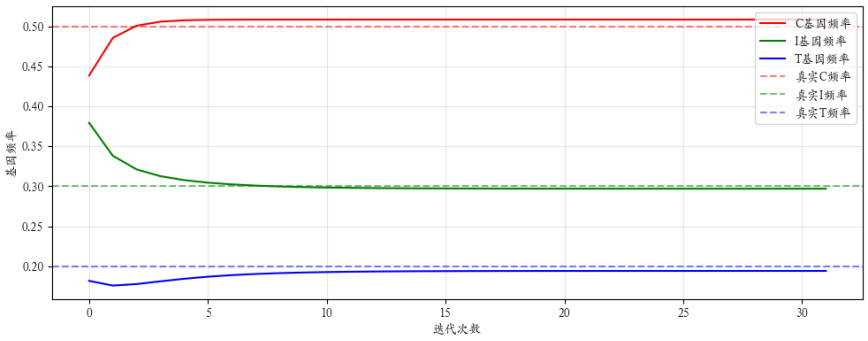

图 4: 等位基因频率迭代收敛过程

这些结果表明，基因 C 的频率最高，I 和 T 的频率相对较低。这些信息有助于解释桦尺蛾的表现型在不同环境条件下的变化，并为后续的生物学研究提供了依据。

详细实现代码请参见A.2。

2.3 高斯混合模型

高斯混合模型（Gaussian Mixture Model, GMM）是一种常用的聚类方法，具有如下形式的概率分布形式：

$$P(y|\theta) = \sum_{k=1}^{K} \alpha_k \phi(y|\theta_k) \tag{2.1}$$

其中，α_k 为混合系数，$\phi(y|\theta_k)$ 为高斯分布密度函数，$\theta_k = (\mu_k, \sigma_k^2)$ 为高斯分布的均值和方差，

$$\phi(y|\theta_k) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{(y-\mu_k)^2}{2\sigma_k^2}\right) \tag{2.2}$$

称为第 k 个高斯分布的密度函数。

2.3.1 EM 算法在高斯混合模型中的应用

EM 算法在 GMM 中的应用非常广泛，通过迭代更新高斯分布的参数（均值、方差和混合系数），从而对数据进行建模。

假设有一组数据 $y_1, y_2, \cdots, y_n$，服从高斯混合模型，即

$$P(y|\theta) = \sum_{k=1}^{K} \alpha_k \phi(y|\theta_k) \tag{2.3}$$

其中，$\theta = \{\alpha_k, \mu_k, \sigma_k^2\}_{k=1}^K$ 为模型参数。EM 算法估计高斯混合模型的参数的步骤如下：

1. 初始化参数 $\theta^{(0)} = \{\alpha_k^{(0)}, \mu_k^{(0)}, \sigma_k^{2(0)}\}_{k=1}^K$，计算对数似然函数 $L(\theta^{(0)})$。

2. E 步：对于每个数据点 y_i，计算其属于第 k 个高斯分量的后验概率

$$\gamma_{ik} = \frac{\alpha_k^{(t)} \phi(y_i|\theta_k^{(t)})}{\sum_{j=1}^{K} \alpha_j^{(t)} \phi(y_i|\theta_j^{(t)})} \tag{2.4}$$

3. M 步：基于 E 步计算的后验概率更新参数估计

$$\alpha_k^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} \gamma_{ik} \tag{2.5}$$

$$\mu_k^{(t+1)} = \frac{\sum_{i=1}^{n} \gamma_{ik} y_i}{\sum_{i=1}^{n} \gamma_{ik}} \tag{2.6}$$

$$\sigma_k^{2(t+1)} = \frac{\sum_{i=1}^{n} \gamma_{ik} (y_i - \mu_k^{(t+1)})^2}{\sum_{i=1}^{n} \gamma_{ik}} \tag{2.7}$$

4. 计算对数似然函数

$$L(\theta^{(t+1)}) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \alpha_k^{(t+1)} \phi(y_i|\theta_k^{(t+1)}) \right) \tag{2.8}$$

5. 检查收敛条件：$|L(\theta^{(t+1)}) - L(\theta^{(t)})| < \epsilon$，若满足则停止迭代，否则继续从 E 步开始下一轮迭代。

2.3.2 高斯混合模型案例

1. 参数设置

本案例使用如下参数设置：

- 真实混合系数：$\alpha = [0.3, 0.4, 0.3]$
- 真实均值：$\mu = [-2.0, 0.0, 4.0]$
- 真实标准差：$\sigma = [0.5, 0.8, 1.5]$
- 样本数量：$n = 1000$

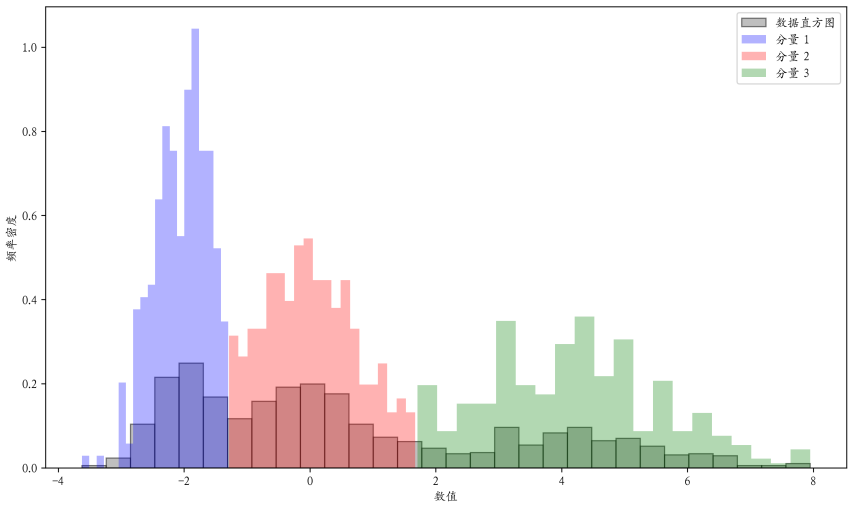

图 5: 数据分布与成分分解

根据上述参数生成数据，数据分布如图 (5) 所示。可以看到，数据呈现出三个高斯分布的混合特征。

2.EM 算法实现步骤

EM 算法通过迭代方式估计模型参数，包括以下步骤：

1. **数据生成**：根据指定的混合系数、均值和标准差生成 $n = 1000$ 个样本点

2. **参数初始化**：

   - 混合系数 $\alpha_k^{(0)} = 1/K$（初始时设每个组分权重相等）
   - 均值 $\mu_k^{(0)}$ 在数据范围内均匀分布，确保初始均值能够覆盖整个数据空间
   - 标准差 $\sigma_k^{(0)}$ 基于数据的整体标准差设定

3. **E 步（期望步）**：对每个数据点 y_i，计算其属于第 k 个高斯组分的后验概率（责任度）

$$\gamma_{ik} = \frac{\alpha_k^{(t)} \phi(y_i|\mu_k^{(t)}, \sigma_k^{2(t)})}{\sum_{j=1}^{K} \alpha_j^{(t)} \phi(y_i|\mu_j^{(t)}, \sigma_j^{2(t)})}$$

4. **M 步（最大化步）**：基于 E 步计算的后验概率更新参数估计

$$\begin{aligned}
\alpha_k^{(t+1)} &= \frac{1}{n} \sum_{i=1}^{n} \gamma_{ik} \\
\mu_k^{(t+1)} &= \frac{\sum_{i=1}^{n} \gamma_{ik} y_i}{\sum_{i=1}^{n} \gamma_{ik}} \\
\sigma_k^{2(t+1)} &= \frac{\sum_{i=1}^{n} \gamma_{ik} (y_i - \mu_k^{(t+1)})^2}{\sum_{i=1}^{n} \gamma_{ik}}
\end{aligned}$$

5. **计算对数似然**：评估模型拟合质量并检查收敛情况

$$L(\theta^{(t+1)}) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \alpha_k^{(t+1)} \phi(y_i|\theta_k^{(t+1)}) \right)$$

6. **收敛判断**：检查参数变化或对数似然函数变化是否小于预设阈值

   - 参数变化：$\|\theta^{(t+1)} - \theta^{(t)}\| < \epsilon_1$
   - 对数似然变化：$|L(\theta^{(t+1)}) - L(\theta^{(t)})| < \epsilon_2$

7. **重复迭代**：如未收敛，返回第 3 步继续迭代，直到满足收敛条件或达到最大迭代次数

2.3.3 结果分析与评估

经过多轮迭代，EM 算法成功估计出三个高斯分布的参数。最终估计结果与真实值的比较如下：

表 3: EM 算法多模型参数估计对比

参数	模型 1			模型 2			模型 3		
	真实值	估计值	误差	真实值	估计值	误差	真实值	估计值	误差
α	0.30	0.28	-0.02	0.40	0.41	0.01	0.30	0.30	0.00
μ	-2.00	-2.04	-0.04	0.00	-0.06	-0.06	4.00	4.09	0.09
σ	0.50	0.46	-0.04	0.80	0.84	0.04	1.50	1.48	-0.02

观察结果可以得出以下分析:

- 所有参数估计都非常接近真实值，表明 EM 算法在本案例中取得了良好的拟合效果
- 混合系数 (α) 的估计尤为准确，三个组分的误差均在 0.02 以内
- 均值参数 (μ) 的估计较为精确，最大偏差仅为 0.09（第三组分）
- 标准差 (σ) 的估计也很准确，最大偏差为 0.04（第一和第二组分）

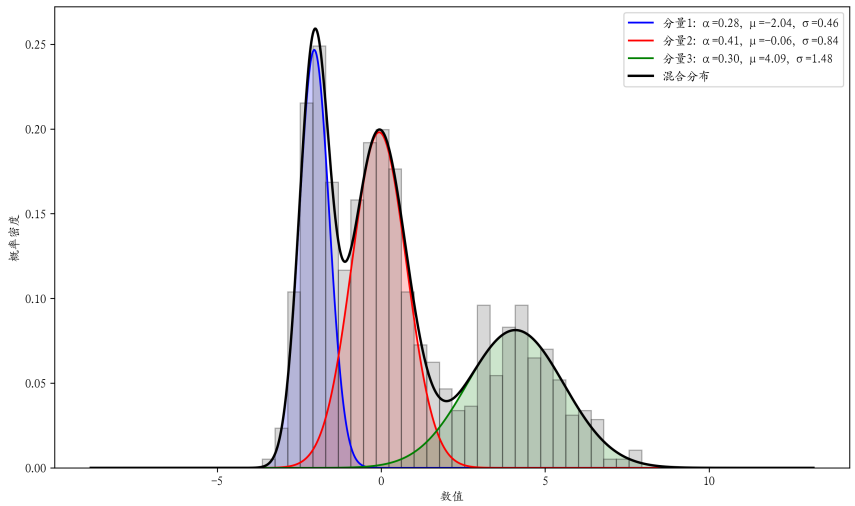

图 6: 估计的高斯混合模型概率密度 - 黑线为混合分布，彩色曲线为各组分分布

最终，EM 算法成功地对多模态数据进行了建模，识别出数据中的三个高斯分布组分。每个数据点被分配到最可能的高斯分量，这对数据聚类和密度估计具有重要意义。这种软聚类方法比 K-means 等硬聚类方法提供了更丰富的概率信息，能够更好地描述数据的内在结构。

通过可视化混合分布及其组分分布（如图6所示），我们可以直观地理解数据的多模态性质及 EM 算法的拟合效果。该案例充分展示了 EM 算法在处理含隐变量的复杂模型中的有效性和实用价值。

有关 GMM 的完整代码实现请参见附录A.3。

2.4 应用总结

EM 算法在实际应用中的效果非常显著，尤其是在处理不完全数据和隐变量模型时，能够提供准确且高效的参数估计。在硬币问题、桦尺蛾问题和高斯混合模型等应用中，EM 算法都展示了其强大的适应性和实用性。通过不断交替进行 E 步和 M 步，EM 算法可以逐步逼近最大似然估计，在许多领域中得到广泛应用。

小结

EM 算法通过期望（E 步）和最大化（M 步）交替进行，能够有效处理隐变量模型和缺失数据。通过实际应用案例，如硬币问题、桦尺蛾问题和高斯混合模型，EM 算法展示了其在实际问题中的强大能力，尤其是在多模态数据建模和基因频率估计等方面。

3 EM 算法的推广：GEM 算法

3.1 GEM 算法的背景与动机

EM 算法（期望最大化）在处理含有隐变量的最大似然估计问题时有着广泛应用，但在一些实际问题中，标准的 EM 算法在 M 步的求解上存在一定困难，尤其当 Q 函数无法通过解析方法直接求解时。这时，GEM 算法（Generalized EM Algorithm）应运而生，作为 EM 算法的一种推广形式，它通过放宽对 M 步的要求，允许在 M 步中仅要求 Q 函数增大，而不必求解其极大值。

标准 EM 算法要求 M 步找到使期望函数 $Q(\theta, \theta^{(i)})$ 最大化的参数 $\theta^{(i+1)}$，即：

$$\theta^{(i+1)} = \arg\max_{\theta} Q(\theta, \theta^{(i)}) \tag{3.1}$$

而在 GEM 算法中，M 步不要求 Q 函数的最大化，只需要找到一个使 Q 增大的 $\theta^{(i+1)}$，即：

$$Q(\theta^{(i+1)}, \theta^{(i)}) \geq Q(\theta^{(i)}, \theta^{(i)}) \tag{3.2}$$

这使得 GEM 算法具有更大的灵活性，可以在许多复杂的高维问题中应用，尤其是在 Q 函数无法解析最大化时。

3.2 GEM 算法的工作原理

GEM 算法与 EM 算法的最大区别在于 M 步的处理方式。具体来说，EM 算法要求 M 步通过优化 Q 函数来找到参数的极大似然估计，而 GEM 算法允许在 M 步中通过数值方法，找到一个能够增加 Q 函数的解，而不需要追求极大化。

- **E 步**：与 EM 算法相同，计算隐变量 Z 的后验分布，并计算 $Q(\theta, \theta^{(i)})$，该函数表

示在当前参数估计下对数似然的期望。

- **M 步**：不要求找到使 Q 最大化的参数，而是找到一个使得 Q 函数增大的解。这使得 M 步不再是严格的最优化问题，可以采用数值优化技术进行近似求解。

这种修改使得 GEM 算法能够灵活地处理一些复杂的、不能直接求解 Q 函数最大值的问题。与 EM 算法相比，GEM 算法具有更强的适应性，尤其在解决高维度问题时，它能避免陷入求解 Q 函数最大值的困难。

3.3 GEM 算法的迭代过程

GEM 算法的迭代过程遵循 EM 算法的基本框架，但放宽了 M 步的最大化要求。其主要步骤如下：

1. **E 步（期望步）**：给定当前的参数估计值 $\theta^{(i)}$，计算隐变量 Z 的后验分布，并计算辅助函数 $Q(\theta, \theta^{(i)})$，该函数表示对数似然函数在当前估计下的期望。

$$Q(\theta, \theta^{(i)}) = \mathbb{E}_{Z|Y,\theta^{(i)}}[\log P(Y, Z|\theta)] \tag{3.3}$$

2. **M 步（最大化步）**：不要求 $Q(\theta, \theta^{(i)})$ 达到最大值，而是寻找一个 $\theta^{(i+1)}$，使得 $Q(\theta, \theta^{(i)})$ 增大，即：

$$Q(\theta^{(i+1)}, \theta^{(i)}) \geq Q(\theta^{(i)}, \theta^{(i)}) \tag{3.4}$$

这一过程可以通过数值优化方法来实现，而不一定要求解析解。

3. **收敛判断**：在每一轮迭代中，计算参数变化是否小于预设的阈值 δ，或者 Q 函数的增量是否足够小：

$$\|\theta^{(i+1)} - \theta^{(i)}\| < \delta \quad 或 \quad |Q(\theta^{(i+1)}, \theta^{(i)}) - Q(\theta^{(i)}, \theta^{(i)})| < \delta_2 \tag{3.5}$$

当满足这些条件时，算法结束。

3.4 GEM 算法的优势与局限性

3.4.1 优势

- **灵活性**： GEM 算法通过放宽 M 步的最大化要求，使其在一些无法通过解析方法求解 Q 函数最大值的情况下，仍能进行有效的参数估计。
- **适应复杂问题**：当模型复杂，无法直接求解 Q 的极大值时，GEM 算法能够通过数值优化方法，灵活地处理这些问题。
- **提高计算效率**：在某些问题中，求解 Q 的最大值是非常计算密集型的，GEM 算法可以通过采用梯度下降等数值方法来近似求解，从而提高计算效率。

3.4.2 局限性

- **收敛速度较慢**：由于 GEM 算法不要求严格最大化 Q 函数，其收敛速度通常比 EM 算法慢，尤其是在没有显式最优化的情况下。
- **依赖数值优化方法**： GEM 算法通常需要使用数值优化方法，这可能会增加计算的复杂性，并且在某些高维度问题中可能会变得计算密集。
- **容易陷入局部最优**：由于 GEM 算法只要求 Q 增大，而不是极大化，可能导致算法仅收敛到局部最优解。

3.5 GEM 算法的应用场景

GEM 算法的灵活性使其在多个领域中得到了应用，特别是在无法解析求解的高维复杂模型中。以下是 GEM 算法的典型应用场景：

- **高斯混合模型 (GMM)**： GEM 算法用于高斯混合模型中的参数估计，尤其在模型的复杂度较高，无法解析求解 Q 函数时，使用数值方法进行近似优化。
- **隐马尔可夫模型 (HMM)**：在 HMM 中，当模型的转移概率或观测概率无法通过解析方法求解时，GEM 算法提供了有效的解决方案。
- **主题模型**：在自然语言处理领域，特别是 LDA（Latent Dirichlet Allocation）模型中，GEM 算法可以用于估计潜在主题的分布。
- **贝叶斯推断**：在一些复杂的贝叶斯推断问题中，GEM 算法通过数值优化方法估计后验分布，尤其适用于高维度和复杂的模型。

3.6 GEM 算法小结

GEM 算法作为 EM 算法的扩展，放宽了 M 步的优化要求，使得它在许多复杂的统计模型中具有重要应用。通过只要求 Q 函数增大而不是极大化，GEM 算法在高维问题和难以解析的模型中表现出色。然而，这种放宽条件也导致了其收敛速度较慢，且可能依赖数值优化方法。

小结

GEM 算法通过放宽 EM 算法 M 步的最大化要求，使其在处理高维度、复杂模型时更加灵活。虽然它的收敛速度较慢，且可能依赖数值优化方法，但它在许多应用中表现出了强大的适应性，特别是在隐马尔可夫模型、主题模型等复杂问题中，提供了一种有效的解决方案。

4 总结

本文详细介绍了 EM 算法及其应用，并探讨了其在各种实际问题中的有效性和局限性。EM 算法是一种处理含隐变量问题的经典方法，广泛应用于统计学、机器学习和数据科学领域。通过本文的探讨，可以清晰地了解到 EM 算法的理论框架、应用流程、以及不同场景下的具体应用。以下是本文的主要总结内容：

4.1 EM 算法的理论体系

EM 算法的核心思想是通过交替进行期望步（E 步）和最大化步（M 步）来逐步逼近最大似然估计。在 E 步，EM 算法估计隐变量的期望，而在 M 步则通过最大化期望函数来更新模型参数。通过这种交替的迭代过程，EM 算法能够有效处理含有隐变量的统计模型。

本文详细推导了 EM 算法的数学基础，使用了 Jensen 不等式来求得对数似然函数的下界，并通过最大化辅助函数 $Q(\theta, \theta^{(i)})$ 来更新参数。此外，EM 算法具有良好的收敛性，可以保证似然函数单调递增，但无法保证达到全局最优解，这也是 EM 算法的一大局限性。

4.2 EM 算法的应用与反思

EM 算法在多个实际问题中取得了显著成果。通过硬币问题、桦尺蛾基因频率估计以及高斯混合模型的应用，展示了 EM 算法在处理不完全数据和隐变量模型时的强大能力。在硬币问题中，EM 算法通过迭代计算隐变量的期望并更新参数估计，成功估计出了硬币正面出现的概率。在桦尺蛾基因频率估计中，EM 算法通过观测到的表现型数据推断了基因型的分布，为生物学研究提供了理论依据。而在高斯混合模型中，EM 算法则用于估计混合高斯分布的参数，从而对多模态数据进行建模。

然而，EM 算法也有其局限性。首先，EM 算法对初始值较为敏感，可能导致算法陷入局部最优解。其次，在处理高维数据时，EM 算法的计算效率可能成为瓶颈，尤其是在 M 步的最大化计算中，需要耗费大量的计算资源。未来的研究方向应关注如何提高 EM 算法的计算效率，并减少对初值的依赖。

4.3 EM 算法的拓展与未来发展

随着计算技术的不断进步，EM 算法在许多实际问题中的应用得到不断扩展。特别是在面对更复杂、更高维的模型时，EM 算法的推广版本，如 GEM 算法，能够有效地克服标准 EM 算法的不足。GEM 算法通过放宽 M 步的最大化要求，使其在无法解析求解 Q 函数的情况下，依然能够有效地进行参数估计。未来，GEM 算法的应用将会更加广泛，尤其是在隐马尔可夫模型、主题模型以及深度学习中的生成模型等复杂问题中。

此外，EM 算法与其他优化方法的结合，如变分推断、贝叶斯推断等，将进一步拓展 EM 算法的应用领域。大规模数据处理、并行计算和在线学习等新兴技术的应用，也为 EM 算法在大数据时代的实际应用提供了新的解决方案。

4.4 结论

EM 算法作为处理隐变量和缺失数据的经典方法，具有坚实的理论基础和广泛的应用前景。本文系统地总结了 EM 算法的理论推导、收敛性分析、应用流程及其在不同领域中的应用。随着算法的发展和计算技术的进步，EM 算法的变体和扩展将继续推动其在机器学习和数据科学中的应用，尤其是在处理复杂模型和大数据时，EM 算法仍将是不可或缺的重要工具。

参考文献

[1] 李航. 统计学习方法 [M]. 北京: 清华大学出版社, 2012.

[2] Dempster, A.P., Laird, N.M. and Rubin, D.B. Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society: Series B (Methodological), 39(1):1–38, 1977.

[3] Hastie, T., Tibshirani, R. and Friedman, J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York: Springer-Verlag, 2001.

[4] 茆诗松, 王静龙, 濮晓龙. 高等数理统计 [M]. 北京: 高等教育出版社, 1998.

[5] Wu, C.F.J. On the Convergence Properties of the EM Algorithm. The Annals of Statistics, 11(1):95–103, 1983.

[6] Neal, R.M., Hinton, G.E. and Jordan, M.I. A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants. In: Learning in Graphical Models, pp.355–368. Cambridge, MA: MIT Press, 1999.

A 附录

运行环境说明：

- Python 3.8.20
- VsCode 1.98.2

相关代码可前往 Gitee 仓库下载 (点击跳转)

A.1 硬币问题的 Python 实现

```python
import numpy as np
import random

random.seed(42)

def EM_algorithm(p, q, data, delta=1e-4, max_iter=100):
    n = len(data)
    for i in range(max_iter):
        # E-step
        p_old, q_old = p, q
        p_new = np.zeros(n)
        q_new = np.zeros(n)
        for j in range(n):
            p_new[j] = p**data[j].count('H') * (1-p)**data[j].count('T') / (p**data[j].count('H') * (1-p)**data[j].count('T') + q**data[j].count('H') * (1-q)**data[j].count('T'))
            q_new[j] = 1 - p_new[j]
        E_A_H = np.sum(p_new * np.array([data[j].count('H') for j in range(n)]))
        E_A_T = np.sum((p_new) * np.array([data[j].count('T') for j in range(n)]))
        E_B_H = np.sum(q_new * np.array([data[j].count('H') for j in range(n)]))
        E_B_T = np.sum((q_new) * np.array([data[j].count('T') for j in range(n)]))
        # M-step
        p = E_A_H / (E_A_H + E_A_T)
        q = E_B_H / (E_B_H + E_B_T)
        print('Iteration:', i+1)
        print('p =', p)
        print('q =', q)
        if np.abs(p-p_old) < delta and np.abs(q-q_old) < delta:
            break
    return p, q
```

```
29
30 data = [['H', 'T', 'T', 'T', 'H', 'H', 'T', 'H', 'T', 'H'],
31         ['H', 'H', 'H', 'H', 'T', 'H', 'H', 'H', 'H', 'H'],
32         ['H', 'T', 'H', 'H', 'H', 'H', 'H', 'T', 'H', 'H'],
33         ['H', 'T', 'H', 'T', 'T', 'T', 'H', 'H', 'T', 'T'],
34         ['T', 'H', 'H', 'H', 'T', 'H', 'H', 'H', 'T', 'H']]
35 p, q = EM_algorithm(0.6, 0.5, data)
36 print('result:')
37 print('\tp =', p)
38 print('\tq =', q)
```

A.2 桦尺蛾问题的 Python 实现

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  plt.rcParams['font.sans-serif'] = ['Kai']  # 设置中文字体
4  plt.rcParams['axes.unicode_minus'] = False  # 显示负号
5
6  def generate_data(n, pc=0.5, pi=0.3, pt=0.2):
7      """生成模拟观测数据
8
9      Args:
10         n (int): 数据样本数量
11         pc (float): C等位基因的频率
12         pi (float): I等位基因的频率
13         pt (float): T等位基因的频率
14
15     Returns:
16         list: 观测数据列表，元素为 'C', 'I' 或 'T'
17     """
18     # 计算各表型概率
19     prob_c = pc**2 + 2*pc*(pi + pt)  # CC, CI, CT
20     prob_i = pi**2 + 2*pi*pt         # II, IT
21     prob_t = pt**2                   # TT
22
23     # 生成数据
24     data = []
25     for _ in range(n):
26         r = np.random.rand()
27         if r < prob_c:
28             data.append('C')
29         elif r < prob_c + prob_i:
30             data.append('I')
31         else:
```

```
            data.append('T')
    return data

def em(data, p0=None, max_iter=100, tol=1e-6, verbose=True):
    """使用EM算法估计等位基因频率

    Args:
        data (list): 观测数据列表
        p0 (list): 初始参数 [pc, pi, pt]
        max_iter (int): 最大迭代次数
        tol (float): 收敛阈值
        verbose (bool): 是否打印迭代信息

    Returns:
        tuple: (估计的频率 [pc, pi, pt], 迭代历史)
    """
    # 参数初始化
    if p0 is None:
        p = np.random.dirichlet([1, 1, 1])
    else:
        p = np.array(p0) / sum(p0)

    history = {'pc': [p[0]], 'pi': [p[1]], 'pt': [p[2]]}
    for it in range(max_iter):
        # E步: 计算期望计数
        count_c, count_i, count_t = 0.0, 0.0, 0.0

        for pheno in data:
            pc, pi, pt = p
            if pheno == 'C':
                denom = pc**2 + 2*pc*(pi + pt)
                if denom <= 1e-8:
                    continue
                # 各基因型概率
                prob_cc = (pc**2) / denom
                prob_ci = (2*pc*pi) / denom
                prob_ct = (2*pc*pt) / denom
                # 累加期望计数
                count_c += 2*prob_cc + prob_ci + prob_ct
                count_i += prob_ci
                count_t += prob_ct

            elif pheno == 'I':
                denom = pi**2 + 2*pi*pt
                if denom <= 1e-8:
```

```
                    continue
                # 各基因型概率
                prob_ii = (pi**2) / denom
                prob_it = (2*pi*pt) / denom
                # 累加期望计数
                count_i += 2*prob_ii + prob_it
                count_t += prob_it

            else:  # T表型
                count_t += 2

        # M步：更新参数
        total = count_c + count_i + count_t
        if total == 0:
            new_p = np.array([0, 0, 0])
        else:
            new_p = np.array([count_c, count_i, count_t]) / total

        # 保存本次迭代结果
        history['pc'].append(new_p[0])
        history['pi'].append(new_p[1])
        history['pt'].append(new_p[2])

        # 检查收敛
        delta = np.linalg.norm(new_p - p)
        if verbose:
            print(f"Iteration {it+1}: C={new_p[0]:.4f}, I={new_p[1]:.4f}, T
    ={new_p[2]:.4f}, delta={delta:.6f}")
        if delta < tol:
            break
        p = new_p

    return p, history

def plot_convergence(history, true_values=None):
    """绘制迭代收敛过程

    Args:
        history (dict): 保存迭代历史的字典
        true_values (list): 真实参数值 [pc, pi, pt]
    """
    iterations = range(len(history['pc']))

    plt.figure(figsize=(10, 4))
    plt.plot(iterations, history['pc'], 'r-', label='C基因频率')
```

```
    plt.plot(iterations, history['pi'], 'g-', label='I基因频率')
    plt.plot(iterations, history['pt'], 'b-', label='T基因频率')

    if true_values is not None:
        plt.axhline(y=true_values[0], color='r', linestyle='--', alpha=0.5, label='真实C频率')
        plt.axhline(y=true_values[1], color='g', linestyle='--', alpha=0.5, label='真实I频率')
        plt.axhline(y=true_values[2], color='b', linestyle='--', alpha=0.5, label='真实T频率')

    plt.xlabel('迭代次数')
    plt.ylabel('基因频率')
    # plt.title('EM算法估计胡椒蛾基因频率的收敛过程')
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()

    # 保存图表
    plt.savefig('peppered_moths_convergence.png', dpi=300)
    plt.show()

# 生成模拟数据并估计参数
np.random.seed(42)
true_params = [0.5, 0.3, 0.2]  # 真实的基因频率
data = generate_data(10000, *true_params)

# 随机初始化参数
initial_params = np.random.dirichlet([1, 1, 1])
print(f"初始参数: C={initial_params[0]:.4f}, I={initial_params[1]:.4f}, T={initial_params[2]:.4f}")

# 运行EM算法
estimated_params, history = em(data, initial_params, max_iter=50, tol=1e-6)

print("\nFinal Estimates:")
print(f"C: {estimated_params[0]:.4f}, I: {estimated_params[1]:.4f}, T: {estimated_params[2]:.4f}")
print(f"真实值: C: {true_params[0]:.4f}, I: {true_params[1]:.4f}, T: {true_params[2]:.4f}")

# 绘制收敛过程
plot_convergence(history, true_params)

# 计算并显示误差
```

```
error = np.linalg.norm(estimated_params - true_params)
print(f"\n参数估计误差: {error:.6f}")
```

A.3 高斯混合模型的 Python 实现

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
plt.rcParams['font.sans-serif'] = ['Kai']  # 设置中文字体
plt.rcParams['axes.unicode_minus'] = False  # 显示负号

# 设置随机种子以确保结果可重现
np.random.seed(42)

# 生成高斯混合数据
n_samples = 1000

# 定义三个高斯分量的参数
mu1, mu2, mu3 = -2.0, 0.0, 4.0
sigma1, sigma2, sigma3 = 0.5, 0.8, 1.5

# 混合系数（权重）
true_alpha = [0.3, 0.4, 0.3]

# 生成数据
n1 = int(true_alpha[0] * n_samples)
n2 = int(true_alpha[1] * n_samples)
n3 = n_samples - n1 - n2

X1 = np.random.normal(mu1, sigma1, n1)
X2 = np.random.normal(mu2, sigma2, n2)
X3 = np.random.normal(mu3, sigma3, n3)
X = np.concatenate([X1, X2, X3])

# 打乱数据顺序
np.random.shuffle(X)

# 记录真实的类别标签
true_labels = np.concatenate([np.zeros(n1), np.ones(n2), 2*np.ones(n3)])
order = np.argsort(X)
true_labels = true_labels[order]
X = X[order]

# 高斯分布概率密度函数
```

```
def gaussian_pdf(x, mu, sigma):
    """
    计算高斯分布的概率密度
    """
    return norm.pdf(x, loc=mu, scale=sigma)

# EM算法实现
def EM_GMM(X, K, max_iter=100, tol=1e-6):
    """
    使用EM算法估计一维高斯混合模型的参数

    Args:
        X: 数据点 [n_samples]
        K: 高斯分量数量
        max_iter: 最大迭代次数
        tol: 收敛阈值

    Returns:
        alpha: 混合系数
        mu: 均值列表
        sigma: 标准差列表
        history: 迭代历史记录
    """
    n = len(X)  # 样本数

    # 随机初始化参数
    alpha = np.ones(K) / K  # 均匀分布的初始混合系数

    # 根据数据范围设置初始均值
    min_x = np.min(X)
    max_x = np.max(X)
    # 在数据范围内均匀分布初始均值
    mu = np.linspace(min_x, max_x, K)

    # 初始标准差 - 使用数据的标准差
    sigma = np.ones(K) * np.std(X) / np.sqrt(K)

    # 记录历史参数
    history = {
        'alpha': [alpha.copy()],
        'mu': [mu.copy()],
        'sigma': [sigma.copy()],
        'log_likelihood': []
    }

```

```
    for iteration in range(max_iter):
        # === E步 ===
        gamma = np.zeros((n, K))

        for k in range(K):
            gamma[:, k] = alpha[k] * gaussian_pdf(X, mu[k], sigma[k])

        row_sums = gamma.sum(axis=1)[:, np.newaxis]
        row_sums[row_sums == 0] = 1e-10
        gamma = gamma / row_sums

        # === M步 ===
        N_k = gamma.sum(axis=0)
        alpha_new = N_k / n

        mu_new = np.zeros(K)
        for k in range(K):
            mu_new[k] = np.sum(gamma[:, k] * X) / N_k[k]

        sigma_new = np.zeros(K)
        for k in range(K):
            diff = X - mu_new[k]
            sigma_new[k] = np.sqrt(np.sum(gamma[:, k] * diff**2) / N_k[k])

            sigma_new[k] = max(sigma_new[k], 1e-3)

        log_likelihood = 0
        for i in range(n):
            s = 0
            for k in range(K):
                s += alpha[k] * gaussian_pdf(X[i], mu[k], sigma[k])
            log_likelihood += np.log(s)

        # 保存历史
        history['alpha'].append(alpha_new.copy())
        history['mu'].append(mu_new.copy())
        history['sigma'].append(sigma_new.copy())
        history['log_likelihood'].append(log_likelihood)

        # 检查收敛
        if iteration > 0:
            ll_diff = abs(log_likelihood - history['log_likelihood'][-2])
            if ll_diff < tol:
                print(f"收敛于第 {iteration+1} 次迭代，对数似然差: {ll_diff
    :.8f}")
```

```
                break

        # 更新参数
        alpha = alpha_new
        mu = mu_new
        sigma = sigma_new

        if (iteration + 1) % 10 == 0:
            print(f"迭代 {iteration+1} - 对数似然: {log_likelihood:.4f}")

    if iteration == max_iter - 1:
        print(f"达到最大迭代次数 {max_iter}")

    return alpha, mu, sigma, history

# 可视化函数
def plot_gmm(X, alpha, mu, sigma, history):
    """
    可视化高斯混合模型
    """
    colors = ['blue', 'red', 'green']
    markers = ['o', 's', '^']

    # 计算数据点的后验概率
    n = len(X)
    gamma = np.zeros((n, len(mu)))
    for k in range(len(mu)):
        gamma[:, k] = alpha[k] * gaussian_pdf(X, mu[k], sigma[k])

    gamma = gamma / gamma.sum(axis=1)[:, np.newaxis]
    labels = np.argmax(gamma, axis=1)

    # 图1: 绘制直方图，不同颜色表示不同分量
    plt.figure(figsize=(10, 6))
    plt.hist(X, bins=30, density=True, alpha=0.5, color='gray',
             edgecolor='black', label='数据直方图')

    # 各分量的直方图
    for k in range(len(mu)):
        idx = labels == k
        plt.hist(X[idx], bins=20, density=True, alpha=0.3,
                 color=colors[k], label=f'分量 {k+1}')

    plt.xlabel('数值')
    plt.ylabel('频率密度')
```

```
    plt.legend()
    plt.tight_layout()
    plt.savefig('gmm_histogram.png', dpi=300, bbox_inches='tight')
    plt.show()

    # 图2: 概率密度函数
    plt.figure(figsize=(10, 6))
    x_grid = np.linspace(np.min(X) - 2*np.std(X), np.max(X) + 2*np.std(X), 1000)

    # 绘制各分量密度
    for k in range(len(mu)):
        density = gaussian_pdf(x_grid, mu[k], sigma[k])
        plt.plot(x_grid, alpha[k] * density, color=colors[k], linestyle='-',
                 label=f'分量{k+1}: α={alpha[k]:.2f}, μ={mu[k]:.2f}, σ={sigma[k]:.2f}')
        plt.fill_between(x_grid, alpha[k] * density, alpha=0.2, color=colors[k])

    # 绘制混合分布
    mixture_density = np.zeros_like(x_grid)
    for k in range(len(mu)):
        mixture_density += alpha[k] * gaussian_pdf(x_grid, mu[k], sigma[k])

    plt.plot(x_grid, mixture_density, color='black', linestyle='-', linewidth=2,
             label='混合分布')

    # 添加数据点分布
    plt.hist(X, bins=30, density=True, alpha=0.3, color='gray',
             edgecolor='black')

    plt.xlabel('数值')
    plt.ylabel('概率密度')
    plt.legend()
    plt.tight_layout()
    plt.savefig('gmm_density.png', dpi=300, bbox_inches='tight')
    plt.show()

alpha_est, mu_est, sigma_est, history = EM_GMM(X, K=3, max_iter=100, tol=1e-4)
print("\n估计的混合系数:", alpha_est)
print("\n估计的均值:", mu_est)
print("\n估计的标准差:", sigma_est)

```

```
print("\n真实参数:")
print("真实混合系数:", true_alpha)
print("真实均值:", [mu1, mu2, mu3])
print("真实标准差:", [sigma1, sigma2, sigma3])

# 可视化结果
plot_gmm(X, alpha_est, mu_est, sigma_est, history)
```