EM 算法

一 实验要求

一个袋子中三种硬币的混合比例为：s1, s2 与 1-s1-s2 (0<=si<=1), 三种硬币掷出正面的概率分别为：p, q, r。（1）自己指定系数 s1, s2, p, q, r，生成 N 个投掷硬币的结果（由 01 构成的序列，其中 1 为正面，0 为反面），利用 EM 算法来对参数进行估计并与预先假定的参数进行比较。

二 EM 算法及公式推导

EM 算法适用于求解含有隐变量的变量估计问题。EM 算法主要分为两个步骤：E-step 和 M-step。E-step 对变量求估计，再通过 M-step 求极大似然，不断迭代直至收敛到局部最优。

对于三种硬币问题，E-step 可求解每个样本的后验概率 π_i。每个样本 x_i 的概率 $p(x_i)$ 可以表示为

$$\begin{aligned} p(x_i) &= \sum_{z=1}^{3} p(x \mid z) \times p(z) \\ &= p^{x_i}(1-p)^{1-x_i} s_1 + q^{x_i}(1-q)^{1-x_i} s_2 + r^{x_i}(1-r)^{1-x_i}(1-s_1-s_2) \end{aligned} \tag{1}$$

其中，z 表示硬币的种类。则该样本的后验概率可以分别表示为

$$\begin{aligned} \pi_1(x_i) &= p(z=1 \mid x_i) = \frac{p(x_i \mid z=1) \times p(z=1)}{p(x_i)} = \frac{p^{x_i}(1-p)^{1-x_i} s_1}{p(x_i)} \\ \pi_2(x_i) &= p(z=2 \mid x_i) = \frac{p(x_i \mid z=2) \times p(z=2)}{p(x_i)} = \frac{q^{x_i}(1-q)^{1-x_i} s_2}{p(x_i)} \\ \pi_3(x_i) &= p(z=3 \mid x_i) = \frac{p(x_i \mid z=3) \times p(z=3)}{p(x_i)} = \frac{r^{x_i}(1-r)^{1-x_i}(1-s_1-s_2)}{p(x_i)} \end{aligned} \tag{2}$$

随后进行 M-step，求解在该迭代步下的参数的极大似然估计，

$$s_1 = \frac{\sum_i \pi_1(x_i)}{N}, s_2 = \frac{\sum_i \pi_2(x_i)}{N} \tag{3}$$

$$p = \frac{\sum_i \pi_1(x_i) x_i}{N s_1}, q = \frac{\sum_i \pi_2(x_i) x_i}{N s_2}, \hat{r} = \frac{\sum_i \pi_3(x_i) x_i}{N(1-s_1-s_2)} \tag{4}$$

三 实验结果

表 1 仿真初始值、迭代初始值以及迭代收敛值

参数	值	迭代初始值	迭代收敛值	收敛后参数对似然函数的导数
s1	0.2	0.2	0.19	0
s2	0.2	0.3	0.30	0
p	0.5	0.5	0.55	0
q	0.6	0.7	0.74	0
r	0.8	0.7	0.74	0

表 2 仿真初始值、迭代初始值以及迭代收敛值

参数	值	迭代初始值	迭代收敛值	收敛后参数对似然函数的导数
s1	0.3	0.4	0.41	0
s2	0.6	0.5	0.48	0
p	0.4	0.5	0.42	0
q	0.6	0.7	0.63	0
r	0.3	0.4	0.33	0

两次实验的结果如上表所示，发现经过 EM 算法迭代后并没有收敛到样本分布的实际参数值。通过计算收敛后参数对似然函数的导数发现已经达到了局部收敛点。说明似然函数是一个非凸函数，有多个局部最优解，并且由于 EM 算法并没有办法确定收敛点是否为全局最优解，所以通过 EM 算法对三种硬币的隐变量模型的参数估计并不准确。

四 实验代码

```
## EM
import numpy as np


def main():
    s1, s2, p, q, r = 0.2, 0.2, 0.5, 0.6, 0.8
```

```
        total_num = 1000
        head_num = int(total_num * (s1*p + s2*q + (1-s1-s2)*r))
        tail_num = total_num - head_num
        samples = np.zeros(total_num)
        samples[:head_num] = 1
        iterations = 6000

        e_s1, e_s2, e_p, e_q, e_r = 0.5, 0.5, 0.7, 0.7, 0.7
        for i in range(iterations):
            p_samples = np.power(e_p, samples)*np.power(1-e_p, 1-samples)*e_s1 + \
                        np.power(e_q, samples)*np.power(1-e_q, 1-samples)*e_s2 + \
                        np.power(e_r, samples)*np.power(1-e_r, 1-samples)*(1-e_s1-e_s2)
            pi1 = np.power(e_p, samples)*np.power(1-e_p, 1-samples)*e_s1 / p_samples
            pi2 = np.power(e_q, samples)*np.power(1-e_q, 1-samples)*e_s2 / p_samples
            pi3 = np.power(e_r, samples)*np.power(1-e_r, 1-samples)*(1-e_s1-e_s2) / p_samples
            e_s1, e_s2  = pi1.sum()/total_num, pi2.sum()/total_num
            e_p, e_q, e_r = np.sum(pi1*samples)/pi1.sum(), np.sum(pi2*samples)/pi2.sum(), np.sum(pi3*samples)/pi3.sum()
        print(e_s1, e_s2, e_p, e_q, e_r)

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