統計計算與模擬(EM)

目標 ：以 EM 演算法將 Old Faithful Geyser Dataset 分群並估計母體參數

資料介紹 ：

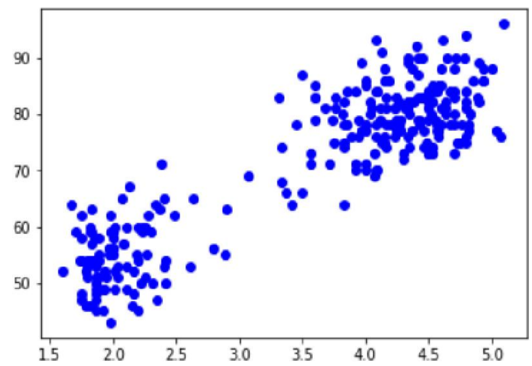
Old Faithful Geyser Dataset 具 272 筆間歇泉資料，共有兩個變數 1. Eruptions: 噴發時間(秒) 2. Waiting: 噴發間隔時間(分鐘)，從散佈圖可看出資料大致上可分為兩群，因此可以決定 k=2，並假設資料服從二元常態分配，在此假設之下，執行 EM 演算法分群並估計參數。

```
faithful.head(10)
```

	Unnamed: 0	eruptions	waiting
0	1	3.600	79
1	2	1.800	54
2	3	3.333	74
3	4	2.283	62
4	5	4.533	85
5	6	2.883	55
6	7	4.700	88
7	8	3.600	85
8	9	1.950	51
9	10	4.350	85

```
plt.plot(faithful['eruptions'],faithful['waiting'],'bo')
```

[<matplotlib.lines.Line2D at 0x1f268874080>]



EM (Expectation Maximization) 簡介 ：

EM 是透過迭代找到一組分群及參數可使概似函數(likelihood function)最大化的演算法，已知我們手中有一筆觀測資料$x = (x_1, x_2, \dots, x_n)$，生成一筆資料$y = (y_1, y_2, \dots, y_n); \left(其中 y_i = j, j = 1, \dots, k, k 為總群數\right)$，$y_i$又稱作 “class label” ，可視作每個樣本的標籤，告訴我們這筆樣本x_i屬於第j類分配，再給定群數k及起始參數$\theta^{(0)}$，過程如下：

首先以觀測資料x及起始參數$\theta^{(0)}$估計 label $y^{(1)}$，這步驟稱作“E-Step”(分群)

再根據$y^{(1)}$更新參數$\theta^{(1)}$，這步驟稱作 “M-Step”

重覆 E-Step 及 M-Step，直至概似函數收斂

即是：以參數分群 → 以分群估計參數 → 以參數分群 → ... 反覆迭代

執行 EM 演算法：

在已知群數 k=2、分配假設為二元常態的情況下，設定參數

$\theta = (\alpha_1, \alpha_2, \mu_1, \Sigma_1, \mu_2, \Sigma_2)$;其中$(\alpha_1, \alpha_2)$為兩群出現的比率，首先第一步就是計算

在給定$(x, \theta^{(0)})$條件下，$y_i = j$ 機率$P\left(y_i = j \middle| x_i, \theta^{(0)}\right) =$

$$\frac{P(x_i\ ,y_i=j|\theta^{(0)})}{P(x_i|\theta^{(0)}\ )} = \frac{\alpha_j^{(0)}\phi(x_i|\mu_j^{(0)},\Sigma_j^{(0)})}{\alpha_1^{(0)}\phi\left(x_i|\mu_1^{(0)},\Sigma_1^{(0)}\right)+\alpha_2^{(0)}\phi\left(x_i|\mu_2^{(0)},\Sigma_2^{(0)}\right)}$$

(其中$\phi(x_i|\mu_1, \Sigma_1), \phi(x_i|\mu_2, \Sigma_2)$皆為二元常態分配的$pdf$)

```
#define a function of 2-dim normal distribution
def BN(X,MU,COV):
    y=np.exp(-((((X-MU).T).dot(np.linalg.inv(COV))).dot(X-MU))/2)/(2*np.pi*(np.sqrt(np.linalg.det(COV))))
    return y

#defind a function to compute label probability p(yj|xi,Sita)
def PY(X,Sita):
    y=[]
    for i in range(len(X)):
        y.append([(Sita[0]*BN(X[i],Sita[2],Sita[3]))/(Sita[0]*BN(X[i],Sita[2],Sita[3])+Sita[1]*BN(X[i],Sita[4],Sita[5])),
                  Sita[1]*BN(X[i],Sita[4],Sita[5])/(Sita[0]*BN(X[i],Sita[2],Sita[3])+Sita[1]*BN(X[i],Sita[4],Sita[5])) ])
    y=np.array(y)
    return y
```

函數 BN(x_i, μ, Σ)是二元常態分配的pdf，而函數 PY(x, θ)的結果會是

$((P(y_1 = 1|x_1, \theta), P(y_1 = 2|x_1, \theta)), \ldots, (P(y_{27\ 2} = 1|x_{27\ 2}\theta), P(y_{27\ 2} = 2|x_{27\ 2}\theta)))$的陣列

由於 EM 演算法的目的是求出可使概似函數最大化的參數，因此在迭代之前須

計算概似函數，若概似函數$L\left(\theta^{(t)}\middle|x, y^{(t)}\right) < L\left(\theta^{(t-1)}\middle|x, y^{(t-1)}\right)$，則停止迭代。

因為概似函數$L\left(\theta^{(t)}\middle|x, y^{(t)}\right)$在樣本數多的情況下會太接近 0，不好比較，因此

在這裡將概似函數取對數：

$$Q\left(\theta \middle| \theta^{(t)}\right) = E_Y(\log(P(x, y|\theta)|x, \theta^{(t)})$$
$$= \sum_{i=1}^{27} \sum_{j=1}^{2} \log\left(\alpha_j \phi\left(x_i \middle| \mu_j, \Sigma_j\right)\right) P\left(y_i = j \middle| x_i, \theta^{(t)}\right)$$

```
#define a likelihood function of mixture 2-dim normal distribution
def Q(X,Sita):
    y=0
    for i in range(len(X)):
        y+=np.log(Sita[0]*BN(X[i],Sita[2],Sita[3]))*PY(X,Sita)[i,0]+np.log(Sita[1]*BN(X[i],Sita[4],Sita[5]))*PY(X,Sita)[i,1]
    return y
```

建立完概似函數後，建立每次迭代各參數$\theta = (\alpha_1, \alpha_2, \mu_1, \Sigma_1, \mu_2, \Sigma_2)$的估計量

根據弱大數法則，$\frac{1}{n}\sum_{i=1}^{n} x_i \xrightarrow{P} E(X)$，

因此如果我們要估計第 j 分配在母體中的比率α_j，可用：

$\alpha_j^{(t+1)} = \frac{\#\ (y_i = j)}{n}$ (hard assigned)

在這裡提一下 EM 演算法在迭代過程中的分群是採用 soft assigned

soft assigned 並不會指定x_i屬於特定的哪個分配，而是根據x_i屬於這個分配的機率是多少來給予加權。而 hard assigned 就是直接指定x_i屬於哪個分配，K-means 就是採用這樣的方法分群

兩者比較如下圖:

	EM		K-Means	
	$y_i = 1$	$y_i = 2$	$y_i = 1$	$y_i = 2$
x_1	0.7	0.3	1	0
x_2	0.9	0.1	1	0
x_3	0.2	0.8	0	1
x_4	0.6	0.4	1	0
x_5	0.3	0.7	0	1

EM 會用 soft assigned 的原因在於好的分群要有精確的分配參數資訊，而好的參數估計又需要好的分群，可以得知在迭代過程中，我們其實並不知道我們的分群是否正確，若以錯誤的分群來估計參數，顯然會得到錯的估計值，如果又用這錯誤的估計值再去更新分群，最後演算法完成的結果肯定差強人意，hard assigned 就很可能造就這樣的結果，指定x_i屬於其中一個分配，就一定得承擔x_i屬於另一個分配的風險，相反地，soft assigned 有將x_i同時屬於兩個分配的可能考慮進加權當中，得到錯誤分群的可能就會大幅降低

根據 soft assigned，參數的估計量為：

$$\alpha_j^{(t+1)} = \frac{\#\ (y_i = j)}{n} \approx \frac{1}{n}\sum_{i=1}^{n} P(y_i = j|x_i, \theta^{(t)})$$

$$\mu_j^{(t+1)} = \frac{\sum_{\#(y_i=j)} x_i}{\#(y_i = j)} \approx \frac{\sum_{i=1}^{n} x_i P(y_i = j|x_i, \theta^{(t)})}{\sum_{i=1}^{n} P(y_i = j|x_i, \theta^{(t)})}$$

$$\Sigma_j^{(t+1)} = \frac{\sum_{\#(y_i=j)}(x_i - \hat{\mu})(x_i - \hat{\mu})^T}{\#(y_i=j)} \approx \frac{\sum_{i=1}^{n}(x_i - \mu^{(t)})(x_i - \mu^{(t)})^T P(y_i = j|x_i, \theta^{(t)})}{\sum_{i=1}^{n} P(y_i = j|x_i, \theta^{(t)})}$$

有了估計量即可建立 EM 演算法：

```
#define a function of EM algorithm(data,initial parameter)
def EM(X,Sita):
    i=0
    sita0=[0.5,0.5,np.array([1,30]),np.array([[0.07,0.5],[0.5,34.0]]),np.array([3.0,50.0]),np.array([[0.17,0.9],[0.9,35.0]])]
    sita1=Sita.copy()
    while Q(X,sita1)>Q(X,sita0):
        sita0=sita1.copy()
        sita1[0:2]=[np.mean(PY(X,sita0)[:,0]),np.mean(PY(X,sita0)[:,1])]
        sita1[2]=(X.T).dot(PY(X,sita0)[:,0])/(np.sum(PY(X,sita0)[:,0]))
        cov1=np.array([[0.0,0.0],[0.0,0.0]])
        for j in range(len(X)):cov1+=(np.outer((X[j]-sita0[2]),X[j]-sita0[2]))*(PY(X,sita0)[j,0])
        sita1[3]=cov1/(np.sum(PY(X,sita0)[:,0]))
        sita1[4]=(X.T).dot(PY(X,sita0)[:,1])/(np.sum(PY(X,sita0)[:,1]))
        cov2=np.array([[0.0,0.0],[0.0,0.0]])
        for k in range(len(X)):cov2+=(np.outer((X[k]-sita0[4]),X[k]-sita0[4]))*(PY(X,sita0)[k,1])
        sita1[5]=cov2/(np.sum(PY(X,sita0)[:,1]))
        i+=1
    print('(a1,a2,mu1,cov1,mu2,cov2)=',sita0,'iteration=',i)
```

接下來給定參數起始值$\theta^{(0)}$，第一組起始值是以”eruptions”=3 切為兩個分群，並用 python 套件 numpy 中的函數直接進行估計，可以想這一組起始值已經很接近 MLE

```
x=[]
for i in range(len(faithful)):x.append((faithful['eruptions'][i],faithful['waiting'][i]))
x=np.array(x)

#use the lists z1、z2 to compute initial value
#from the plot we guess we can divide the data into two group of
# normal distribution by ['eruptions']=3
z1=[]
z2=[]
for i in range(len(x)):
    if x[i][0]<=3:z1.append(x[i])
    else : z2.append(x[i])

z1=np.array(z1)
z2=np.array(z2)

#define a variable of initial parameter from estimation (a1,a2,mu1,cov1,mu2,cov2)
sita=[len(z1)/(len(z1)+len(z2)),len(z2)/(len(z1)+len(z2)),
      np.array([np.mean(z1[:,0]),np.mean(z1[:,1])]),np.cov(z1[:,0],z1[:,1]),
      np.array([np.mean(z2[:,0]),np.mean(z2[:,1])]),np.cov(z2[:,0],z2[:,1])]
```

出來的結果$\theta^{(0)} = (\alpha_1^{(0)} = 0.35661764705882354, \alpha_2^{(0)} = 0.6433823529411765$

$\mu_1^{(0)} = [2.03813402 \quad 54.49484536]^T, \Sigma_1^{(0)} = \begin{bmatrix} 0.07121718 & 0.45226632 \\ 0.45226632 & 34.10674399 \end{bmatrix},$

$\mu_2^{(0)} = [4.29130286 \quad 79.98857143]^T, \Sigma_2^{(0)} = \begin{bmatrix} 0.16879903 & 0.9180667 \\ 0.9180667 & 35.93090312 \end{bmatrix})$

將第一組起始值執行 EM 演算法的結果如下，演算法只迭代一次即停止，結果

也與起始值沒有太大的差異

```
EM(x,sita)
```

```
(a1,a2,mu1,cov1,mu2,cov2)= [0.3560462082867129, 0.6439537917132871, array([ 2.03681388, 54.48286414]), array([[ 0.06951069,  0.
43881671],
       [ 0.43881671, 33.72337668]]), array([ 4.29003333, 79.97257302]), array([[ 0.16950149,  0.93470502],
       [ 0.93470502, 35.98036659]])] iteration= 1
```

接下來將第一組稍作改良，使它稍微偏離中心參數，並執行 EM 演算法

$\theta_2^{(0)} = (\alpha_1^{(0)} = 0.5, \alpha_2^{(0)} = 0.5, \quad \mu_1^{(0)} = [1 \quad 40]^T$

$, \Sigma_1^{(0)} = \begin{bmatrix} 0.07 & 0.5 \\ 0.5 & 34 \end{bmatrix}, \quad \mu_2^{(0)} = [3 \quad 60]^T, \Sigma_2^{(0)} = \begin{bmatrix} 0.17 & 0.9 \\ 0.9 & 35 \end{bmatrix})$

```
#from the other initial paramater to estimate sita
sita_2=[0.5,0.5,np.array([1,40]),np.array([[0.07,0.5],[0.5,34.0]]),np.array([3.0,60.0]),np.array([[0.17,0.9],[0.9,35.0]])]
EM(x,sita_2)
```

```
(a1,a2,mu1,cov1,mu2,cov2)= [0.35587285710570676, 0.6441271428942933, array([ 2.03638845, 54.47851638]), array([[ 0.06916767,
0.43516762],
       [ 0.43516762, 33.69728207]]), array([ 4.28966197, 79.96811517]), array([[ 0.16996844,  0.94060932],
       [ 0.94060932, 36.04621132]])] iteration= 31
```

這筆較差的起始值迭代了 31 次才停止，但最後的結果與另一筆起始值的結果幾乎相同，直到小數點三位後才開始出現分歧，代表這個 EM 演算法的模型並沒有建立錯誤。這筆間歇性噴泉資料的分群工作到這邊告一段落，最後附上分群散佈圖，可看到我們用 EM 演算法清楚地將資料分成兩個群集。

```
# use the estimate to cluster the data to two different distribution
sita_estimate=[0.35587285710570676, 0.6441271428942933, np.array([ 2.03638845, 54.47851638]), np.array([[ 0.06916767,  0.4351676
       [ 0.43516762, 33.69728207]]), np.array([ 4.28966197, 79.96811517]), np.array([[ 0.16996844,  0.94060932],
       [ 0.94060932, 36.04621132]])]
x1=[]
x2=[]
Py=PY(x,sita_estimate)
for i in range(len(x)):
    if Py[i,0]>Py[i,1]:x1.append(x[i])
    else :x2.append(x[i])
x1=np.array(x1)
x2=np.array(x2)
plt.plot(x1[:,0],x1[:,1],'ro')
plt.plot(x2[:,0],x2[:,1],'bo')
```

```
[<matplotlib.lines.Line2D at 0x1f269a953c8>]
```