作品六、期末專題

姓名：黃冠翔
學號：410978040

目標：

- 解決混合常態的參數估計問題（normal mixture data）。
- 學習繪製多變量函數的等高線圖。
- 計算限制式多變量函數的極值(最大值)。
- 計算最大概似估計(MLE)的問題。
- 學習多變量模擬資料的生成。

---

1. 混合常態參數估計 Normal Mixture

前面的範例應用最大概似估計法估計混合貝他（β）分配的參數。另一個常見的問題是混合常態的參數估計。即，

$$\max_{\Omega=\{\pi_1,\pi_2,\mu_1,\sigma_1^2,\mu_2,\sigma_2^2|\pi_1+\pi_2=1,\pi_1,\pi_2,\sigma_1^2,\sigma_2^2>0\}} L(\Omega)$$

其中對數聯合概似函數為

$$L(\Omega) = \ln \Pi_{i=1}^{N}(\pi_1 f(x_i|\mu_1,\sigma_1^2) + \pi_2 f(x_i|\mu_2,\sigma_2^2))$$

$$= \sum_{i=1}^{N} \ln(\pi_1 f(x_i|\mu_1,\sigma_1^2) + \pi_2 f(x_i|\mu_2,\sigma_2^2))$$

$f(x|\mu,\sigma^2)$ 為常態分配的機率密度函數。請模仿前範例的介紹，自行產生適用的資料並運用 $minimize$ 估計參數 $\Omega = \{\pi_1, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2\}$。環境設定如下:

自行設定資料生成的參數 $\Omega = \{\pi_1, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2\}$。

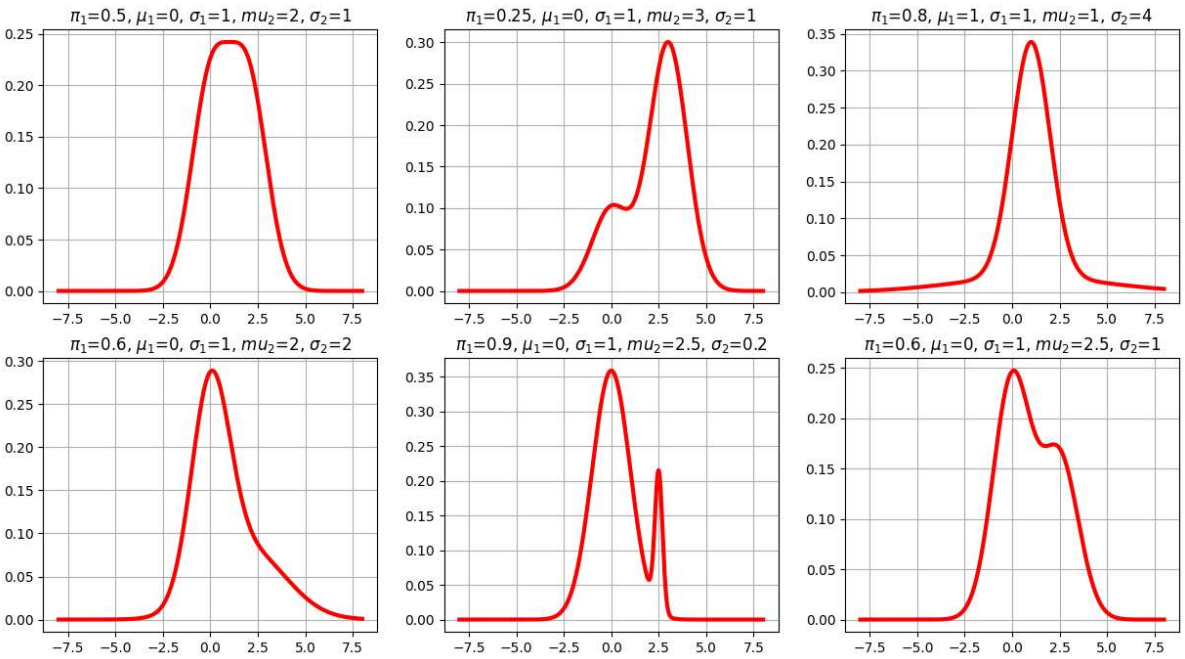
注意事項：

- 分別設定6種不同的參數並繪製混合常態PDF圖進行比較。
- 6張圖的參數由左至右、由上至下依序分別為

$$\begin{aligned}(\pi_1,\mu_1,\sigma_1,\mu_2,\sigma_2) &= (0.5,0,1,2,1)\\ &= (0.25,0,1,3,1)\\ &= (0.8,1,1,1,4)\\ &= (0.6,0,1,2,2)\\ &= (0.9,0,1,2.5,0.2)\\ &= (0.6,0,1,2.5,1)\end{aligned}$$

In [ ]:
```
import numpy as np
import scipy.optimize as opt
import matplotlib.pyplot as plt
from scipy.stats import norm, binom

# set up the parameters --------------
pi11, mu11, s11, mu21, s21 = 0.5, 0, 1, 2, 1
pi12, mu12, s12, mu22, s22 = 0.25, 0, 1, 3, 1
```

```
pi13, mu13, s13, mu23, s23 = 0.8, 1, 1, 1, 4
pi14, mu14, s14, mu24, s24 = 0.6, 0, 1, 2, 2
pi15, mu15, s15, mu25, s25 = 0.9, 0, 1, 2.5, 0.2
pi16, mu16, s16, mu26, s26 = 0.6, 0, 1, 2.5, 1
# draw the mixture pdf ---------------
f1 = lambda x: pi11 * norm.pdf(x, loc = mu11, scale = s11) + \
    (1-pi11) * norm.pdf(x, loc = mu21, scale = s21)
f2 = lambda x: pi12 * norm.pdf(x, loc = mu12, scale = s12) + \
    (1-pi12) * norm.pdf(x, loc = mu22, scale = s22)
f3 = lambda x: pi13 * norm.pdf(x, loc = mu13, scale = s13) + \
    (1-pi13) * norm.pdf(x, loc = mu23, scale = s23)
f4 = lambda x: pi14 * norm.pdf(x, loc = mu14, scale = s14) + \
    (1-pi14) * norm.pdf(x, loc = mu24, scale = s24)
f5 = lambda x: pi15 * norm.pdf(x, loc = mu15, scale = s15) + \
    (1-pi15) * norm.pdf(x, loc = mu25, scale = s25)
f6 = lambda x: pi16 * norm.pdf(x, loc = mu16, scale = s16) + \
    (1-pi16) * norm.pdf(x, loc = mu26, scale = s26)

x = np.linspace(-8, 8, 1000)
fig, ax = plt.subplots(2, 3, figsize = [15, 8])

plt.subplot(231)
ax[0, 0].plot(x, f1(x), color = 'r', linewidth = 3)
plt.grid(True)
ax[0, 0].set_title('$\pi_1$=0.5, $\mu_1$=0, $\sigma_1$=1, $mu_2$=2, $\sigma_2$=1')

plt.subplot(232)
ax[0, 1].plot(x, f2(x), color = 'r', linewidth = 3)
plt.grid(True)
ax[0, 1].set_title('$\pi_1$=0.25, $\mu_1$=0, $\sigma_1$=1, $mu_2$=3, $\sigma_2$=1')

plt.subplot(233)
ax[0, 2].plot(x, f3(x), color = 'r', linewidth = 3)
plt.grid(True)
ax[0, 2].set_title('$\pi_1$=0.8, $\mu_1$=1, $\sigma_1$=1, $mu_2$=1, $\sigma_2$=4')

plt.subplot(234)
ax[1, 0].plot(x, f4(x), color = 'r', linewidth = 3)
plt.grid(True)
ax[1, 0].set_title('$\pi_1$=0.6, $\mu_1$=0, $\sigma_1$=1, $mu_2$=2, $\sigma_2$=2')

plt.subplot(235)
ax[1, 1].plot(x, f5(x), color = 'r', linewidth = 3)
plt.grid(True)
ax[1, 1].set_title('$\pi_1$=0.9, $\mu_1$=0, $\sigma_1$=1, $mu_2$=2.5, $\sigma_2$=0.

plt.subplot(236)
ax[1, 2].plot(x, f6(x), color = 'r', linewidth = 3)
plt.grid(True)
ax[1, 2].set_title('$\pi_1$=0.6, $\mu_1$=0, $\sigma_1$=1, $mu_2$=2.5, $\sigma_2$=1'

plt.show()
```



討論：描述上述程式與執行結果，有甚麼值得說明或強調的。

- 當μ_1和μ_2較接近時，視覺上會好像只有一組，
- 當μ_1和μ_2差距較大時，視覺上能看出兩個常態混合。

注意事項：

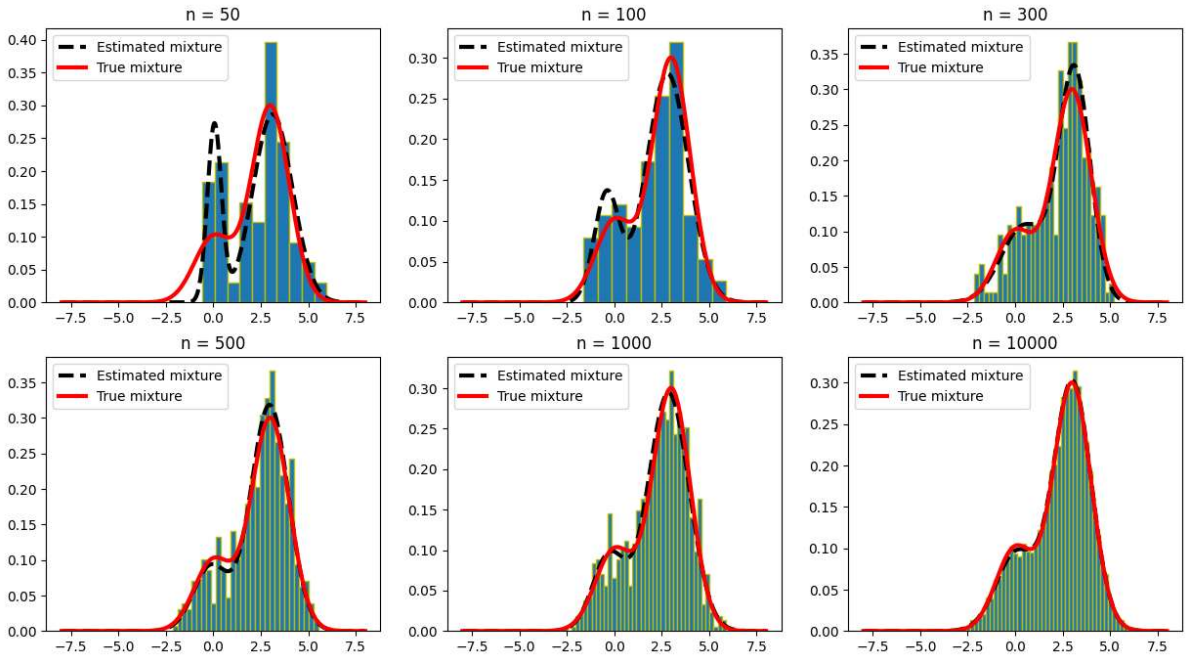
- 取(π_1, μ_1, σ_1, μ_2, σ_2) = (0.25, 0, 1, 3, 1)繪製混合常態的PDF圖。
- 總樣本數 $n = 50, 100, 300, 500, 1000, 10000$。藉以評估樣本數大小對估計值的影響。
- 根據上述參數條件，生成隨機樣本。
- 繪製隨機樣本的直方圖。
- 計算 MLE 的最大值問題（限制式條件）。
- 繪製真實的混合常態 PDF 與估計的混合常態 PDF。

In [ ]:
```
import numpy as np
import scipy.optimize as opt
import matplotlib.pyplot as plt
from scipy.stats import norm, binom

np.seterr(divide = 'ignore', invalid = 'ignore')
pi1, mu1, s1, mu2, s2 = 0.25, 0, 1, 3, 1

f = lambda x: pi1 * norm.pdf(x, loc = mu1, scale = s1) + \
    (1-pi1) * norm.pdf(x, loc = mu2, scale = s2)

x = np.linspace(-8, 8, 1000)

fig, ax = plt.subplots(2, 3, figsize = [15, 8])

plt.subplot(231)
N = 50
N1 = binom.rvs(N, pi1)
N2 = N - N1
sample = np.r_[norm.rvs(loc = mu1, scale = s1, size = N1),
norm.rvs(loc = mu2, scale = s2, size = N2)]
```

```
plt.hist(sample, 10, edgecolor = 'y', density = True)

L = lambda x : -np.sum(np.log(x[0] * norm.pdf(sample, loc = x[1], scale = x[2]) + \
    (1 - x[0]) * norm.pdf(sample, loc = x[3], scale = x[4])))

cons = []
bnds = [(0, 1), (-np.inf, np.inf), (0, np.inf), (-np.inf, np.inf), (0, np.inf)]
opts = dict(disp = True, maxiter = 1e4)
x0 = [0.3, 0.1, 1.2, 4, 1.2] # initial guess
res = opt.minimize(L, x0 = x0,
    bounds = bnds,
    constraints = cons,
    options = opts,
    tol = 1e-8)

f_hat = lambda x: res.x[0] * norm.pdf(x, loc = res.x[1], scale = res.x[2]) + \
    (1-res.x[0]) * norm.pdf(x, loc = res.x[3], scale = res.x[4])
plt.plot(x, f_hat(x), color = 'k', linestyle = '--', \
    linewidth = 3, label = 'Estimated mixture')
ax[0, 0].plot(x, f(x), color = 'r', linewidth = 3, label = 'True mixture')
ax[0, 0].set_title('n = 50')
plt.legend()

plt.subplot(232)
N = 100
N1 = binom.rvs(N, pi1)
N2 = N - N1
sample = np.r_[norm.rvs(loc = mu1, scale = s1, size = N1),
norm.rvs(loc = mu2, scale = s2, size = N2)]

plt.hist(sample, 10, edgecolor = 'y', density = True)

L = lambda x : -np.sum(np.log(x[0] * norm.pdf(sample, loc = x[1], scale = x[2]) + \
    (1 - x[0]) * norm.pdf(sample, loc = x[3], scale = x[4])))

cons = []
bnds = [(0, 1), (-np.inf, np.inf), (0, np.inf), (-np.inf, np.inf), (0, np.inf)]
opts = dict(disp = True, maxiter = 1e4)
x0 = [0.3, 0.1, 1.2, 4, 1.2] # initial guess
res = opt.minimize(L, x0 = x0,
    bounds = bnds,
    constraints = cons,
    options = opts,
    tol = 1e-8)

f_hat = lambda x: res.x[0] * norm.pdf(x, loc = res.x[1], scale = res.x[2]) + \
    (1-res.x[0]) * norm.pdf(x, loc = res.x[3], scale = res.x[4])
plt.plot(x, f_hat(x), color = 'k', linestyle = '--', \
    linewidth = 3, label = 'Estimated mixture')
ax[0, 1].plot(x, f(x), color = 'r', linewidth = 3, label = 'True mixture')
ax[0, 1].set_title('n = 100')
plt.legend()

plt.subplot(233)
N = 300
N1 = binom.rvs(N, pi1)
N2 = N - N1
sample = np.r_[norm.rvs(loc = mu1, scale = s1, size = N1),
norm.rvs(loc = mu2, scale = s2, size = N2)]

plt.hist(sample, 30, edgecolor = 'y', density = True)
```

```
L = lambda x : -np.sum(np.log(x[0] * norm.pdf(sample, loc = x[1], scale = x[2]) + \
    (1 - x[0]) * norm.pdf(sample, loc = x[3], scale = x[4])))

cons = []
bnds = [(0, 1), (-np.inf, np.inf), (0, np.inf), (-np.inf, np.inf), (0, np.inf)]
opts = dict(disp = True, maxiter = 1e4)
x0 = [0.3, 0.1, 1.2, 4, 1.2] # initial guess
res = opt.minimize(L, x0 = x0,
    bounds = bnds,
    constraints = cons,
    options = opts,
    tol = 1e-8)

f_hat = lambda x: res.x[0] * norm.pdf(x, loc = res.x[1], scale = res.x[2]) + \
    (1-res.x[0]) * norm.pdf(x, loc = res.x[3], scale = res.x[4])
plt.plot(x, f_hat(x), color = 'k', linestyle = '--', \
    linewidth = 3, label = 'Estimated mixture')
ax[0, 2].plot(x, f(x), color = 'r', linewidth = 3, label = 'True mixture')
ax[0, 2].set_title('n = 300')
plt.legend()

plt.subplot(234)
N = 500
N1 = binom.rvs(N, pi1)
N2 = N - N1
sample = np.r_[norm.rvs(loc = mu1, scale = s1, size = N1),
norm.rvs(loc = mu2, scale = s2, size = N2)]

plt.hist(sample, 30, edgecolor = 'y', density = True)

L = lambda x : -np.sum(np.log(x[0] * norm.pdf(sample, loc = x[1], scale = x[2]) + \
    (1 - x[0]) * norm.pdf(sample, loc = x[3], scale = x[4])))

cons = []
bnds = [(0, 1), (-np.inf, np.inf), (0, np.inf), (-np.inf, np.inf), (0, np.inf)]
opts = dict(disp = True, maxiter = 1e4)
x0 = [0.3, 0.1, 1.2, 4, 1.2] # initial guess
res = opt.minimize(L, x0 = x0,
    bounds = bnds,
    constraints = cons,
    options = opts,
    tol = 1e-8)

f_hat = lambda x: res.x[0] * norm.pdf(x, loc = res.x[1], scale = res.x[2]) + \
    (1-res.x[0]) * norm.pdf(x, loc = res.x[3], scale = res.x[4])
plt.plot(x, f_hat(x), color = 'k', linestyle = '--', \
    linewidth = 3, label = 'Estimated mixture')
ax[1, 0].plot(x, f(x), color = 'r', linewidth = 3, label = 'True mixture')
ax[1, 0].set_title('n = 500')
plt.legend()

plt.subplot(235)
N = 1000
N1 = binom.rvs(N, pi1)
N2 = N - N1
sample = np.r_[norm.rvs(loc = mu1, scale = s1, size = N1),
norm.rvs(loc = mu2, scale = s2, size = N2)]

plt.hist(sample, 40, edgecolor = 'y', density = True)
```

```
L = lambda x : -np.sum(np.log(x[0] * norm.pdf(sample, loc = x[1], scale = x[2]) + \
    (1 - x[0]) * norm.pdf(sample, loc = x[3], scale = x[4])))

cons = []
bnds = [(0, 1), (-np.inf, np.inf), (0, np.inf), (-np.inf, np.inf), (0, np.inf)]
opts = dict(disp = True, maxiter = 1e4)
x0 = [0.3, 0.1, 1.2, 4, 1.2] # initial guess
res = opt.minimize(L, x0 = x0,
    bounds = bnds,
    constraints = cons,
    options = opts,
    tol = 1e-8)

f_hat = lambda x: res.x[0] * norm.pdf(x, loc = res.x[1], scale = res.x[2]) + \
    (1-res.x[0]) * norm.pdf(x, loc = res.x[3], scale = res.x[4])
plt.plot(x, f_hat(x), color = 'k', linestyle = '--', \
    linewidth = 3, label = 'Estimated mixture')
ax[1, 1].plot(x, f(x), color = 'r', linewidth = 3, label = 'True mixture')
ax[1, 1].set_title('n = 1000')
plt.legend()

plt.subplot(236)
N = 10000
N1 = binom.rvs(N, pi1)
N2 = N - N1
sample = np.r_[norm.rvs(loc = mu1, scale = s1, size = N1),
norm.rvs(loc = mu2, scale = s2, size = N2)]

plt.hist(sample, 50, edgecolor = 'y', density = True)

L = lambda x : -np.sum(np.log(x[0] * norm.pdf(sample, loc = x[1], scale = x[2]) + \
    (1 - x[0]) * norm.pdf(sample, loc = x[3], scale = x[4])))

cons = []
bnds = [(0, 1), (-np.inf, np.inf), (0, np.inf), (-np.inf, np.inf), (0, np.inf)]
opts = dict(disp = True, maxiter = 1e4)
x0 = [0.3, 0.1, 1.2, 4, 1.2] # initial guess
res = opt.minimize(L, x0 = x0,
    bounds = bnds,
    constraints = cons,
    options = opts,
    tol = 1e-8)

f_hat = lambda x: res.x[0] * norm.pdf(x, loc = res.x[1], scale = res.x[2]) + \
    (1-res.x[0]) * norm.pdf(x, loc = res.x[3], scale = res.x[4])
plt.plot(x, f_hat(x), color = 'k', linestyle = '--', \
    linewidth = 3, label = 'Estimated mixture')
ax[1, 2].plot(x, f(x), color = 'r', linewidth = 3, label = 'True mixture')
ax[1, 2].set_title('n = 10000')
plt.legend()
plt.show()
```



討論：描述上述程式與執行結果，有甚麼值得說明或強調的。

- 隨著樣本數的增加，sample的直方圖、真實的混合常態和估計的混合常態愈趨近。
- 在$n = 50$ 時，真實的混合常態和估計的混合常態有顯著差異。
- 在$n = 300$ 時，真實的混合常態和估計的混合常態有較為接近。
- 在$n = 10000$ 時，真實的混合常態和估計的混合常態幾乎重疊。

---

2. 限制式條件的最大值問題 Constraint optimization

計算下列最大概似估計 MLE 問題的參數 α, β：

$$\max_{\alpha,\beta>0} \ln L(\alpha, \beta)$$

其中的聯合概似函數為

$$L(\alpha, \beta) = \prod_{i=1}^{n} f_t(v_i|\alpha, \beta) F_T(u_i|\alpha, \beta)^{-1}$$

where $f_t(v|\alpha, \beta) = \alpha\beta v^{\beta-1} exp(-\alpha v^{\beta})$

$$F_T(u|\alpha, \beta) = 1 - exp(-\alpha u^{\beta})$$

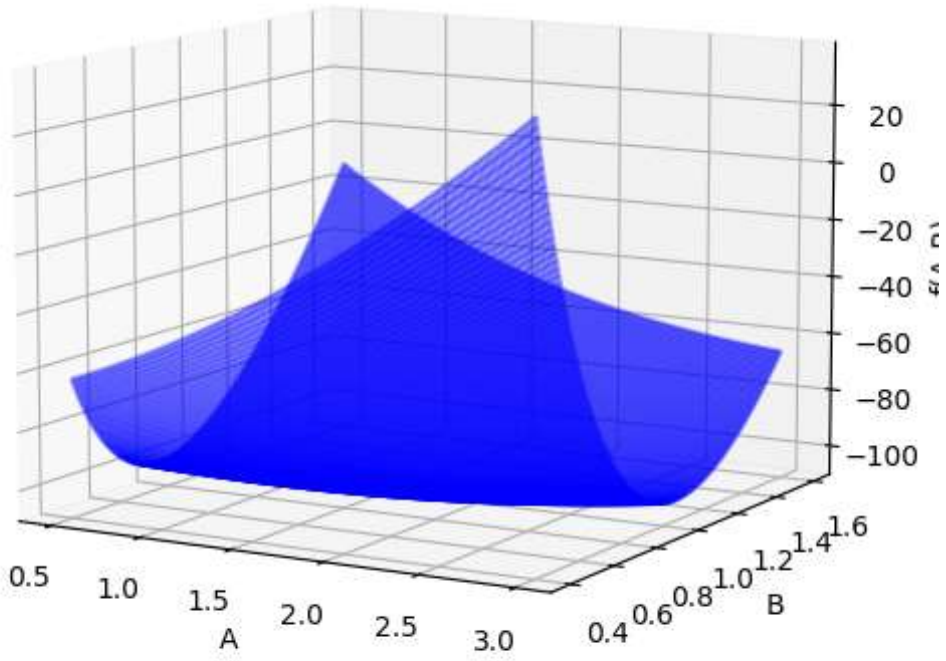
變數 u, v 的 n 個樣本已知並存在檔案 UV.txt

注意事項：

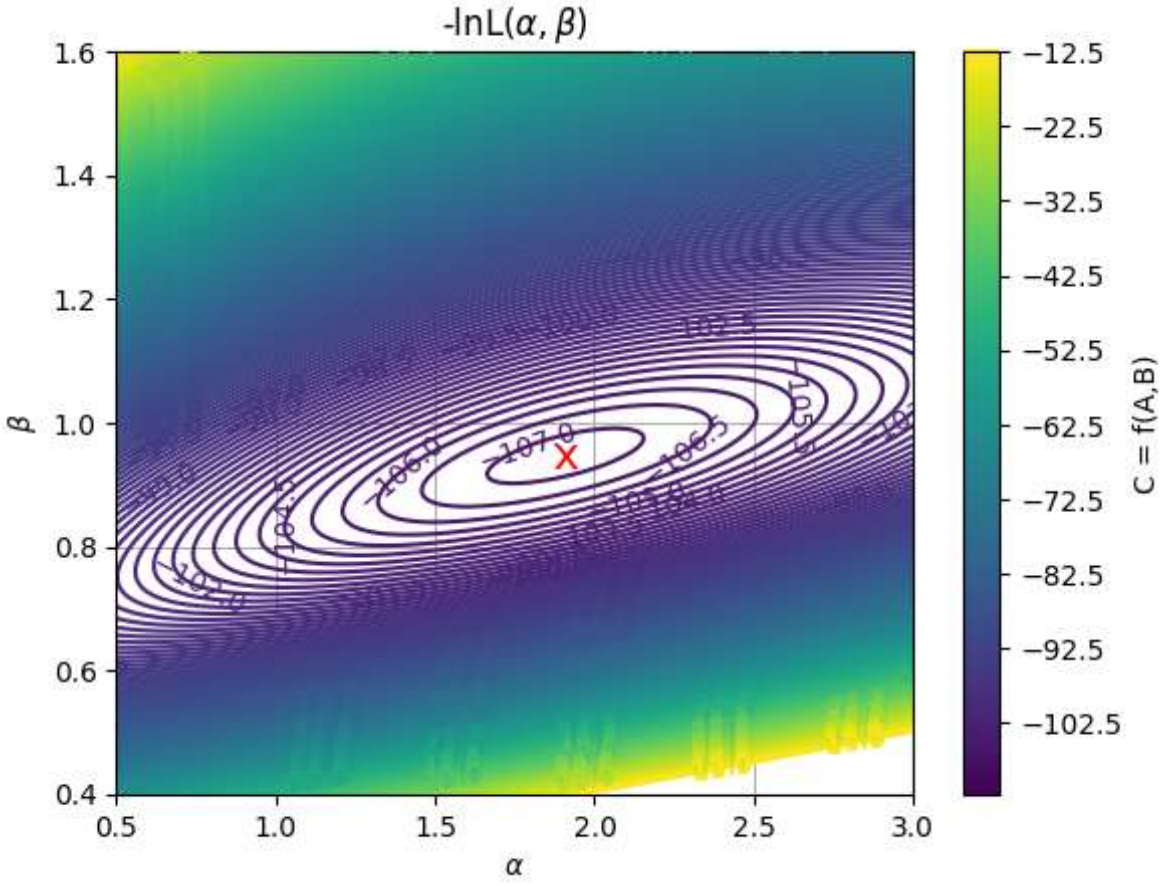
- 先下載資料檔並讀入資料。
- 目標函數 $\ln L(\alpha, \beta)$ 需要進一步推導到比較適合的樣子，也就是將 $\prod$ 透過 $\ln$ 換成 Σ。
- 利用推導到精簡的目標函數，繪製立體圖與等高線圖，並判斷出最大值的位置。
- 接著開始部署 minimize 的各項停止條件及計算。有了等高線圖的幫助，通常答案已經呼之欲出，計算的結果只是得到一組更明確的數據。如圖中紅色的 X。

In [ ]:
```
import numpy as np
import scipy.optimize as opt
import matplotlib.pyplot as plt
```

```
D = np.loadtxt('UV.txt', comments='%')
u, v = D[:, 0], D[:, 1]
n = D.shape[0]

np.seterr(divide = 'ignore', invalid = 'ignore')

def f(alpha):
    F = np.zeros([n, n])
    for i in range(n):
        F =  F + (-1) * (np.log(alpha[0]*alpha[1]) + (alpha[1]-1)*np.log(v[i]) - al
        np.log(1-np.exp(-alpha[0]*(u[i]**alpha[1]))))
    return F

alpha = np.linspace(0.5, 3, n)
beta = np.linspace(0.4, 1.6, n)
A, B = np.meshgrid(alpha, beta)
C = f([A, B])

fig = plt.figure(figsize=(9, 6))
ax = plt.axes(projection = '3d')
ax.plot_wireframe(A, B, C, color ='blue',
    alpha=0.3, rstride = 1, cstride = 1)
ax.set_xlabel('A'), ax.set_ylabel('B')
ax.set_zlabel('f(A,B)')
ax.view_init(10, -60)  #(elev=-165, azim=60)
plt.title('Wireframe (Mesh) Plot')
plt.show()

L = lambda x : -np.sum(np.log(x[0]*x[1]) + (x[1]-1)*np.log(v) - x[0]*(v**x[1]) - \
    np.log(1-np.exp(-x[0]*(u**x[1]))))
opts = dict(disp = True, maxiter = 1e4)
x0 = [1.9, 0.9] # initial guess
res = opt.minimize(L, x0 = x0,
    options = opts,
    tol = 1e-8)

levels = np.arange(-112, -12, 0.5) # levels of contour lines
contours = plt.contour(A, B, C, levels=levels) # check dir(contours)
# add function value on each line
plt.clabel(contours, inline = 0, fontsize = 10) # inline =1 or 0
cbar = plt.colorbar(contours)
plt.xlabel('$\\alpha$'), plt.ylabel('$\\beta$')
cbar.ax.set_ylabel('C = f(A,B)') # set colorbar label
plt.text(res.x[0], res.x[1], 'x', color ='red', fontsize = 16,
 horizontalalignment='center',
 verticalalignment='center')
plt.title('-lnL($\\alpha, \\beta$)')
plt.grid(True)
plt.show()
```




```
Optimization terminated successfully.
         Current function value: -107.208541
         Iterations: 9
         Function evaluations: 36
         Gradient evaluations: 12
```



討論：描述上述程式與執行結果，有甚麼值得說明或強調的。

- 從立體圖中可以發現，模型兩邊較高，愈往中間靠近便開始下降，直到中間的谷底。
- MLE求最大值時，必須先利用opt.minimize計算出最小值為-107.208541，而後加上負號。
- 繪製等高線圖可以明顯看出最大值位置，並利用plt.text標示其位置。
