Stephen_COHEN

February 27, 2024

1 TP Bayes

1.1 Stephen Cohen

1.1.1 1 : Préliminaires

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import sqrtm
from numpy.linalg import eig
import scipy
import scipy.io
import math
from numpy.linalg import det
```

1.1.2 2 : Génération d'une variable aléatoire

[ ]:
```
def gen_gauss(N):
    vect_gauss=2*np.random.randn(N)+3
    return vect_gauss

def moy(data):
    return data.mean()
def var(data):
    return data.var()


#plt.plot([i for i in range(1,1000)],[moy(gen_gauss(i)) for i in range(1,1000)])
#plt.plot([i for i in range(1,1000)],[var(gen_gauss(i)) for i in range(1,1000)])
#plt.show()
```

1.1.3 3 : Génération de vecteurs aléatoires gaussiens

1)

[ ]:
```
N=100
m=[4,9]
s=np.array([[1,0],[0,np.sqrt(6)]])
```

```
def gauss_rand_partiel(n,m):
    return np.random.randn(2,n)+ np.array([[m[0] for _ in range(n)],[m[1] for _
 ↪in range(n)]])

q1 = gauss_rand_partiel(N,m)

print(q1)
```

```
[[ 3.11072136  3.15683439  4.12223897  3.06907968  1.93850823  3.14237557
   2.90985387  2.27413145  5.65890361  5.25375829  3.13253102  3.73747045
   3.06874275  3.4760019   3.69379002  3.96821729  5.37367993  5.2653482
   3.70941577  4.21289411  3.48079594  3.19903245  3.25053884  3.39696787
   3.82413977  3.5380806   3.78939143  4.14448169  2.88098106  2.61678676
   3.32550782  2.98624596  2.62403519  3.41917036  5.01398398  3.20082226
   4.82124579  2.80002354  3.32722254  4.38248734  3.96569052  4.9605267
   3.4812828   3.5705585   5.6571163   3.54662622  4.11739789  4.52103942
   2.15969057  4.94537534  3.19949751  5.17794307  4.44510401  3.28928401
   3.35356686  4.39131795  4.19719171  3.92677169  4.76639657  5.83548855
   5.37145894  4.3519223   3.68359471  2.27314893  3.87581768  3.18929245
   4.47652529  2.93677327  5.47155197  6.41214222  3.99008404  5.66403668
   2.48488176  4.70186663  2.60452943  3.76717811  2.01284858  5.25350042
   3.65071064  5.85441372  3.5874363   3.65071833  4.26986813  2.71134474
   4.04062647  4.62532531  3.7026254   4.49425921  5.41586652  4.56006422
   3.79623333  4.30672645  4.41954716  3.91241416  3.6668253   5.99686796
   3.28707832  3.17421206  3.2875595   4.35113653]
 [ 8.996049   10.72222446  9.64976132  8.51854841  7.79787127  8.8846224
   8.45120888  7.49033493  9.83916928  9.70856898  8.75693286 11.26179975
   9.51855206  9.60735163  9.12660316 10.06800382  9.70681467 10.45545272
   9.7153812   8.78499405  9.70473383  7.28606131  8.11498014  7.21062071
  10.20809019  8.74840783  8.12920057  9.1627786   8.49004717  8.56826691
   9.80709131  9.00562835  9.20177658  8.09121873 10.26539432 10.11746239
   9.05875912  8.89734599  8.25202998  9.00213231  7.93839032  9.0049822
   9.07181963  9.37968217  6.24777459  9.73673658  9.28002995  8.94800917
   8.48752689 10.0180041   8.51838757  7.89344498  8.9568468   8.4553518
   9.65926085  9.81517087  9.58617525 10.32708145  7.87012732  8.38101776
   9.61147994  9.18892762 10.22065437  8.15333032  9.82540328  5.82380527
   8.80057087  7.82384168  7.47209221  9.92739978  9.03328021  8.07017112
   8.72005201  9.86399115 10.33490177 10.08668297  8.95547903 12.30599696
   7.63489763  9.45927115  8.4142445  10.31087247  9.14969751  9.01191005
   9.38234586  7.97269202  7.46199193  7.89211931  8.02174112  9.04495887
   7.25889614  7.3695161   9.35823328  8.61217082  8.6847677   8.92468507
   9.55291756  9.299535    9.50271399  9.25534556]]
```

2)

[ ]:
```
def gauss_rand_full(n,m,s):
    c=np.dot(s,np.random.randn(2,n))+ np.array([[m[0] for _ in range(n)],[m[1]
 ↪for _ in range(n)]])
```

```
    moy=np.mean(c,axis=1)
    print("la moyenne est impirique est de " + str(moy) )
    print("la variance est de " + str(np.cov(c)))
    #print(c)
    return c

q2 = gauss_rand_full(N,m,s)
```

```
la moyenne est impirique est de [4.00239476 9.23696233]
la variance est de [[1.10261006 0.20932281]
 [0.20932281 4.93690516]]
```

3) En prenant Sigma = R (R transposée) alors U=R convient d'après le cours

4) On a tg(2*alpha)=2*S_12/(S_11 - S_22) d'où alpha = (1/2)*tan^-1(2*S_12/(S_11 - S_22))

[ ]:
```
s2 = sqrtm((np.array([[2,2],[2,5]])))
m2 =[0,0]

q4 = gauss_rand_full(N,m2,s2)

alpha = (1/2)*math.atan(2*2/(2 - 5))%(2*math.pi)
print("L'orientation de l'ellipsoïde de Mahalanobis est de : " +str(alpha) + " en radian")

V = np.array([[math.cos(alpha), -math.sin(alpha)],[math.sin(alpha), math.cos(alpha)]])
vp = np.linalg.eigvals(np.dot(s2,s2))

s2_ = np.dot(np.dot(V,np.diag([vp[0],vp[1]])),np.transpose(V))

print(s2_)
```

```
la moyenne est impirique est de [0.14366404 0.15323303]
la variance est de [[1.8445415  1.58907793]
 [1.58907793 3.64684732]]
L'orientation de l'ellipsoïde de Mahalanobis est de : 5.81953769817878 en radian
[[2. 2.]
 [2. 5.]]
```

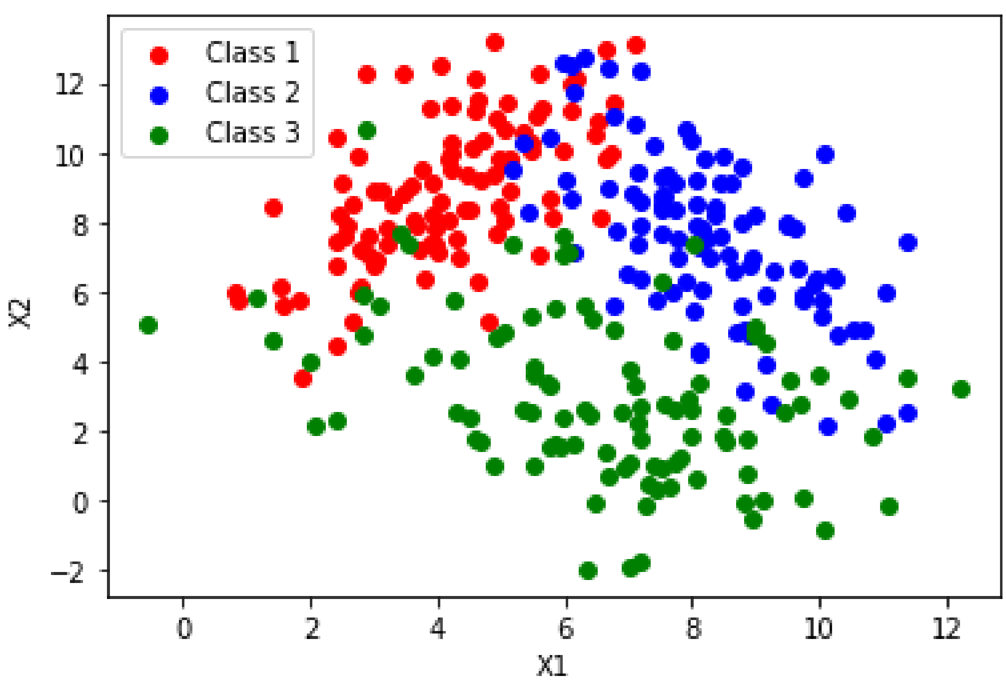
Ainsi la formule fonctionne bien

5)

[ ]:
```
m_1= [4,9]
m_2= [8.5,7.5]
m_3= [6,3.5]
```

```
s_1=sqrtm(np.array([[2,2],[2,5]]))
s_2=sqrtm(np.array([[2,-2],[-2,5]]))
s_3=sqrtm(np.array([[7,-4],[-4,7]]))

x1=gauss_rand_full(N,m_1,s_1)
x2=gauss_rand_full(N,m_2,s_2)
x3=gauss_rand_full(N,m_3,s_3)

plt.scatter(x1[0], x1[1], color='red', label='Class 1')
plt.scatter(x2[0], x2[1], color='blue', label='Class 2')
plt.scatter(x3[0], x3[1], color='green', label='Class 3')

plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.show()
```

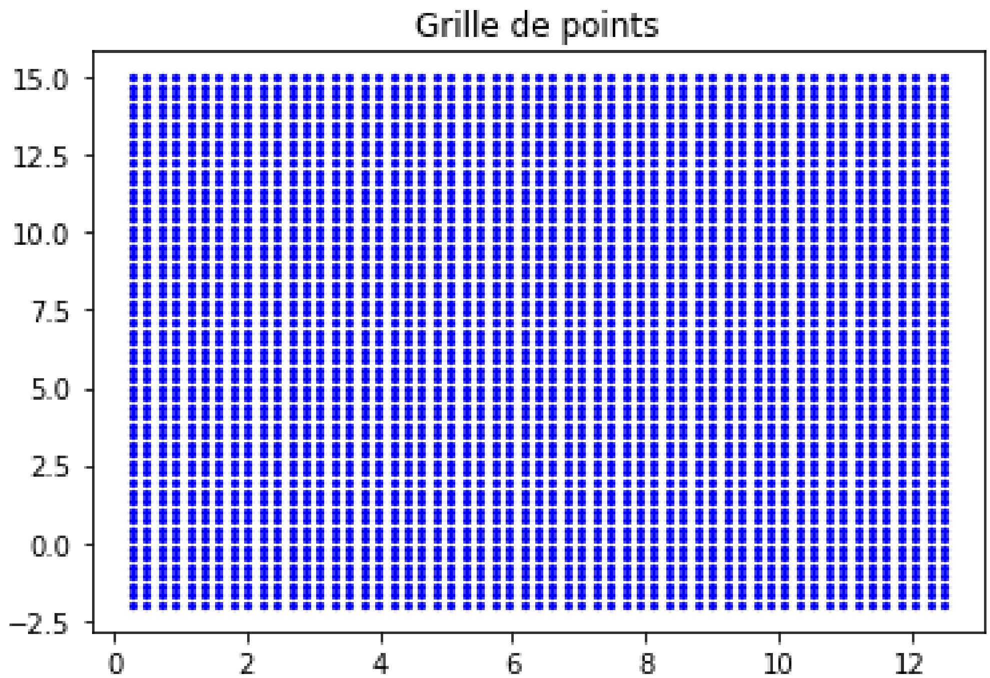
```
la moyenne est impirique est de [4.13784536 8.92508308]
la variance est de [[2.00401581 1.78514714]
 [1.78514714 4.33921416]]
la moyenne est impirique est de [8.29023635 7.57896082]
la variance est de [[ 2.05208019 -2.04189812]
 [-2.04189812  5.45671298]]
la moyenne est impirique est de [6.5878996  2.93266587]
la variance est de [[ 5.62551819 -2.21367308]
 [-2.21367308  5.52173647]]
```



1.1.4 4 : Courbes d'équidensité

1)

[ ]:
```
x_i = np.linspace(0.27,12.5, 57)
y_j = np.linspace(-2,15, 57)

X_grid, Y_grid = np.meshgrid(x_i, y_j)

plt.figure()
plt.scatter(X_grid, Y_grid, color='blue', s=5)
plt.title('Grille de points')
plt.show()
```



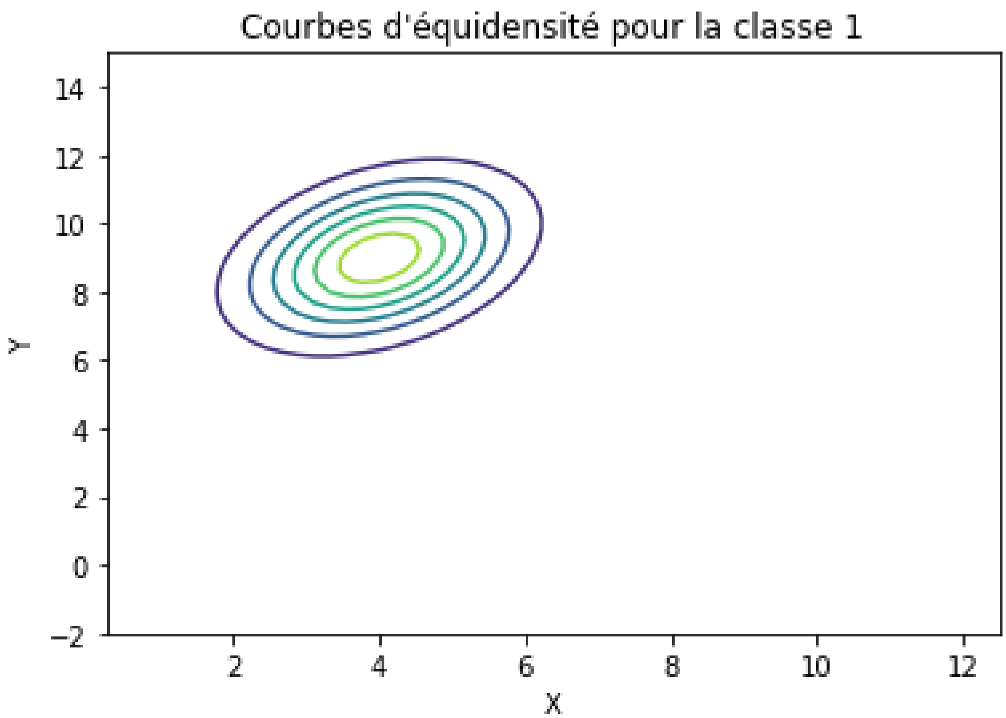
2)

[ ]:
```
m_1 = np.array([4, 9])
s_1 = sqrtm(np.array([[2, 2], [2, 5]]))
dens1 = np.zeros((57, 57))
for i in range(57):
    for j in range(57):
        point = np.array([X_grid[i, j], Y_grid[i, j]])
        diff = point - m_1
        exponent = -0.5 * np.dot(np.dot(diff.T, np.linalg.inv(s_1)), diff)
```

```
        dens1[i, j] = (1 / (2 * np.pi * np.sqrt(np.linalg.det(s_1)))) * np.
 ↪exp(exponent)

#print(dens1)
```

3)

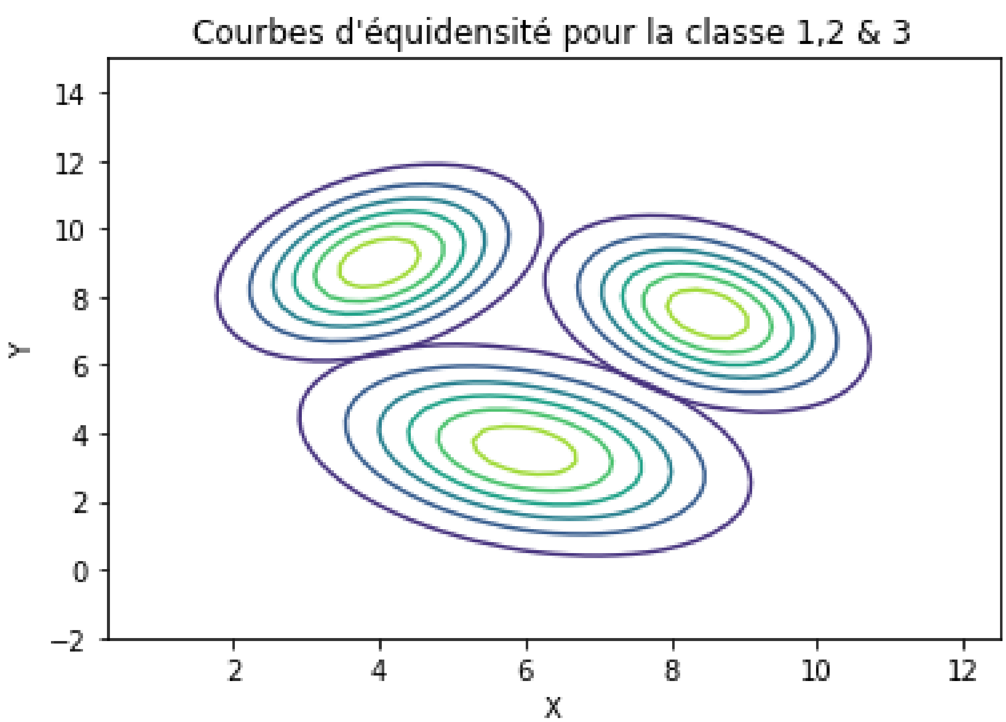
[ ]:
```
plt.figure()
plt.contour(X_grid, Y_grid, dens1)
plt.title('Courbes d\'équidensité pour la classe 1')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



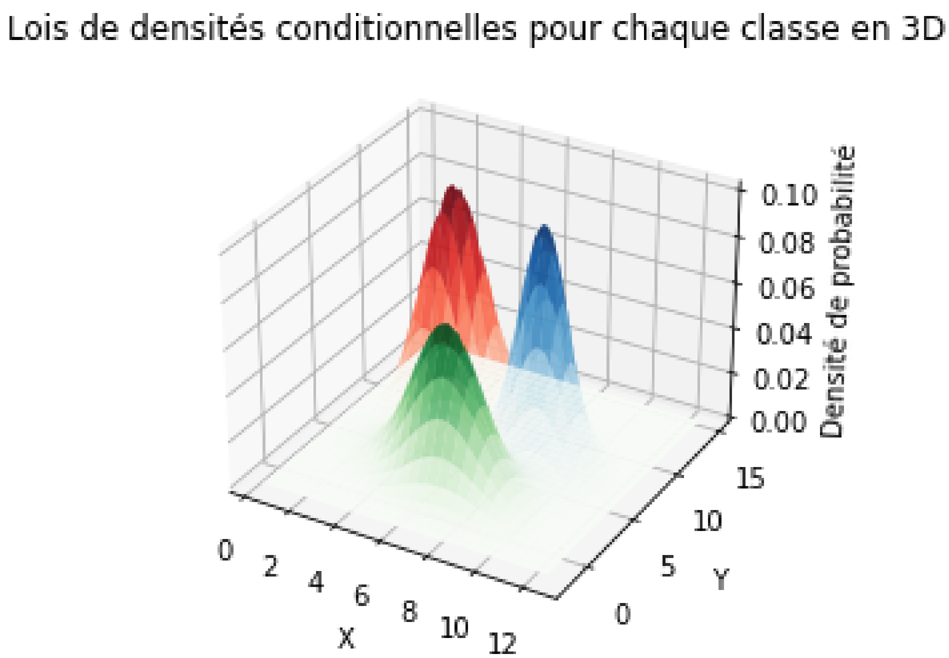
4)

[ ]:
```
##Pour la classe 2

m_2 = np.array([8.5,7.5])
s_2 = sqrtm(np.array([[2,-2],[-2,5]]))
dens2 = np.zeros((57, 57))
for i in range(57):
    for j in range(57):
        point = np.array([X_grid[i, j], Y_grid[i, j]])
```

```
        diff = point - m_2
        exponent = -0.5 * np.dot(np.dot(diff.T, np.linalg.inv(s_2)), diff)
        dens2[i, j] = (1 / (2 * np.pi * np.sqrt(np.linalg.det(s_1)))) * np.
 ↪exp(exponent)

##Pour la classe 3

m_3 = np.array([6,3.5])
s_3 = sqrtm(np.array([[7,-4],[-4,7]]))
dens3 = np.zeros((57, 57))
for i in range(57):
    for j in range(57):
        point = np.array([X_grid[i, j], Y_grid[i, j]])
        diff = point - m_3
        exponent = -0.5 * np.dot(np.dot(diff.T, np.linalg.inv(s_3)), diff)
        dens3[i, j] = (1 / (2 * np.pi * np.sqrt(np.linalg.det(s_3)))) * np.
 ↪exp(exponent)

plt.figure()
plt.contour(X_grid, Y_grid, dens1)
plt.contour(X_grid, Y_grid, dens2)
plt.contour(X_grid, Y_grid, dens3)
plt.title('Courbes d\'équidensité pour la classe 1,2 & 3')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



[ ]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')


ax.plot_surface(X_grid, Y_grid, dens1, cmap='Reds', alpha=0.7, label='Classe 1')
ax.plot_surface(X_grid, Y_grid, dens2, cmap='Blues', alpha=0.7, label='Classe␣
↪2')
ax.plot_surface(X_grid, Y_grid, dens3, cmap='Greens', alpha=0.7, label='Classe␣
↪3')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Densité de probabilité')
ax.set_title('Lois de densités conditionnelles pour chaque classe en 3D')


plt.show()
```



On observe que pour chacune des classes, les amplitudes maximales sont d'environ 0,08 et ont lieu sur les points correspondants à la moyenne de chacune des densité. C'est bien ce que l'on attend puisque ce sont des lois normales donc la densité est la plus forte au niveau de la moyenne. ### 5 : Visulalisation des frontières

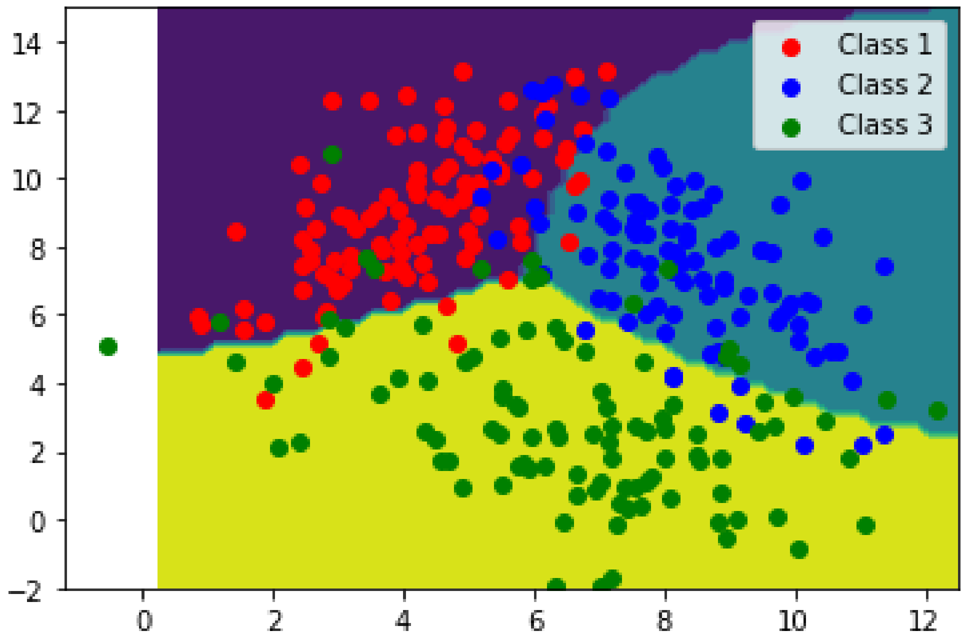
1)

```
[ ]: def classes(d1,d2,d3,n=57):
         z=np.zeros((n,n))
         for i in range(n):
             for j in range(n):
                 a = [d1[i,j],d2[i,j],d3[i,j]]
                 z[i,j]=a.index(max(a))
                 z[i,j]+=1
         return z

     print(classes(dens1,dens2,dens3))
```

```
[[3. 3. 3. … 3. 3. 3.]
 [3. 3. 3. … 3. 3. 3.]
 [3. 3. 3. … 3. 3. 3.]
 …
 [1. 1. 1. … 2. 2. 2.]
 [1. 1. 1. … 2. 2. 2.]
 [1. 1. 1. … 1. 2. 2.]]
```

2)

```
[ ]: z=classes(dens1,dens2,dens3)

     plt.contourf(X_grid,Y_grid,z)
     plt.scatter(x1[0], x1[1], color='red', label='Class 1')
     plt.scatter(x2[0], x2[1], color='blue', label='Class 2')
     plt.scatter(x3[0], x3[1], color='green', label='Class 3')
     plt.legend()
     plt.show()
```



En violet, il s'agit de la zone de l'espace liée la classe 1. En jaune, la classe 3. En bleu, la classe 2.

6 : Application

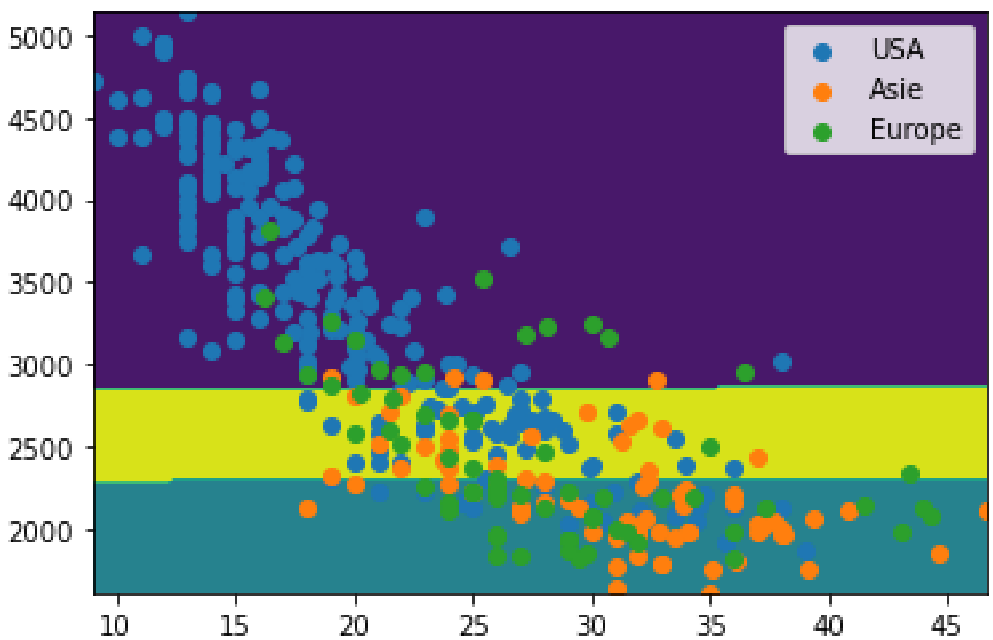
[ ]:
```
data = scipy.io.loadmat('voitures.mat')
cars = data['cars']



MPG = cars[:, 0]
weight = cars[:, 4]
continent = cars[:, -1]

usa=cars[continent==1]
asie=cars[continent==3]
europe=cars[continent==2]

m_USA = np.mean(usa[:,[0,4]],axis=0)
m_Asie = np.mean(asie[:,[0,4]],axis=0)
m_Europe = np.mean(europe[:,[0,4]],axis=0)

s_USA= np.cov(usa[:,[0,4]].T)
s_Asie= np.cov(asie[:,[0,4]].T)
s_Europe= np.cov(europe[:,[0,4]].T)

print(s_USA)
```

```
[[ 4.14785473e+01 -4.33566974e+03]
```

```
[-4.33566974e+03  6.32576357e+05]]
```

Maintenant on va refaire exactement la même chose que dans les questions précédentes.

```
[ ]: nb_point = 200
mpg_min, mpg_max= np.min(MPG),np.max(MPG)
A=np.linspace(mpg_min, mpg_max, 200)

weight_min, weight_max = np.min(weight), np.max(weight)
B=np.linspace(weight_min,weight_max,200)

A_grid,B_grid = np.meshgrid(A,B)

def densite(n,m,s,a,b):
    sqrt= sqrtm(s)
    dens = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            point = np.array([a[i, j], b[i, j]])
            diff = point - m
            exponent = -0.5 * np.dot(np.dot(diff.T, np.linalg.inv(sqrt)), diff)
            dens[i, j] = (1 / (2 * np.pi * np.sqrt(np.linalg.det(sqrt)))) * np.
 ↪exp(exponent)
    return dens

dens_usa=densite(nb_point,m_USA,s_USA,A_grid,B_grid)
dens_asie=densite(nb_point,m_Asie,s_Asie,A_grid,B_grid)
dens_europe=densite(nb_point,m_Europe,s_Europe,A_grid,B_grid)

z_car=classes(dens_usa,dens_asie,dens_europe,n=nb_point)

plt.contourf(A_grid,B_grid,z_car)
plt.scatter(usa[:,[0,4]][:,0],usa[:,[0,4]][:,1], label='USA')
plt.scatter(asie[:,[0,4]][:,0],asie[:,[0,4]][:,1], label='Asie')
plt.scatter(europe[:,[0,4]][:,0],europe[:,[0,4]][:,1], label='Europe')
plt.legend()
plt.show()
```


USA
Asie
Europe
5000
4500
4000
3500
3000
2500
2000
10
15
20
25
30
35
40
45