IPST-CNAM 14 Janvier 2022
RCP208

TP 4
Estimation de densité par noyau
Correction

Question 1:
Correction :

Pour des échantillons plus restreints (`N` plus petit) la densité estimée s'écarte en général plus de la densité réélle, pour des échantillons plus larges (`N` plus grand) l'écart entre les deux densités est en général plus réduit.

Question 2 :
Correction :

Il suffit d'utiliser plus de lois lors de la génération de l'échantillon, en prenant soin de les « localiser » à des abscisses différentes (utiliser des valeurs différentes pour les espérances). Par ex. 3 lois :

```
X = np.concatenate((np.random.normal(0, 1, 0.3 * N),
                    np.random.normal(5, 1, 0.4 * N),
                    np.random.normal(8, 1, 0.3 * N)))[:, np.newaxis]
```

Plus le nombre de lois utilisé pour générer les données est élevé, plus la taille de l'échantillon doit être élevée pour que l'estimation reste de bonne qualité.

Question 3:
Correction :

Une valeur de `bandwidth` trop faible risque d'augmenter le nombre de « modes » (pics différents) dans l'estimation, surtout si l'échantillon est restreint. Une valeur trop élevée pour `bandwidth` produit un « lissage » excessiv, par ex. fusionne les « modes » estimés.

Question 4:
Correction :

Le type de noyau a un impact sur la forme de la densité estimée, mais l'impact sur la qualité d'estimation est plus faible que celui de la valeur de `bandwidth`.

Question 5:
Correction :

Nous pouvons tirer les mêmes conclusions que pour le cas unidimensionnel.

Question 6:

Correction :

Les données avaient été générées suivant une loi uniforme dans l'intervalle $[0,1)^2$ et la grille de visualisation définie en fonction de ce choix. Pour générer et visualiser les données à partir d'une somme de 2 lois normales il faut non seulement modifier la génération des données, mais aussi la définition de la grille de visualisation.

```
# générer l'échantillon
N = 100
kd1 = np.random.randn(N/2, 2) + (1.0, 1.0)
kd2 = np.random.randn(N/2, 2)
kd = np.concatenate((kd1, kd2))

# définir la grille pour la visualisation
grid_size = 100
Gx = np.arange(-1, 3, 4/grid_size)
Gy = np.arange(-1, 3, 4/grid_size)
Gx, Gy = np.meshgrid(Gx, Gy)

# définir la largeur de bande pour le noyau
bw = 0.2

# estimation, puis calcul densité sur la grille
kde4 = KernelDensity(kernel='gaussian', bandwidth=bw).fit(kd)
Z = np.exp(kde4.score_samples(np.hstack(((Gx.reshape(grid_size *
                                                     grid_size))[:,np.newaxis],
              (Gy.reshape(grid_size*grid_size)[:,np.newaxis])))))

# affichage
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(Gx, Gy, Z.reshape(grid_size,grid_size), rstride=1,
                    cstride=1, cmap=cm.coolwarm, linewidth=0, antialiased=True)
ax.scatter(kd[:,0], kd[:,1], -10)
plt.show()
```

Question 7:

Correction :

L'augmentation de la taille de l'échantillon avec une valeur inchangée pour `bw` produit une estimation plus lisse.

Question 8:

Correction :

Le programme suivant fait l'estimation sur les 2 premiers axes principaux :

```
# lecture des données
textures = np.loadtxt('texture.dat')

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(textures[:,:40])
texturesp = pca.transform(textures[:,:40])

# définir la grille pour la visualisation
grid_size = 100
mx = min(texturesp[:,0])
Mx = max(texturesp[:,0])
my = min(texturesp[:,1])
My = max(texturesp[:,1])
xstep = (Mx - mx) / grid_size
ystep = (My - my) / grid_size
Gx = np.arange(mx, Mx, xstep)
Gy = np.arange(my, My, ystep)
Gx, Gy = np.meshgrid(Gx, Gy)

# définir la largeur de bande pour le noyau
bw = 0.06
# estimation, puis calcul densité sur la grille
kdet = KernelDensity(kernel='gaussian', bandwidth=bw).fit(texturesp)
Z = np.exp(kdet.score_samples(np.hstack((
                    (Gx.reshape(grid_size*grid_size))[:,np.newaxis],
                    (Gy.reshape(grid_size*grid_size)[:,np.newaxis])))))

# affichage
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(Gx, Gy, Z.reshape(grid_size,grid_size), rstride=1,
                    cstride=1, cmap=cm.coolwarm, linewidth=0, antialiased=True)
ax.scatter(texturesp[:,0], texturesp[:,1], -0.5)
plt.show()
```

Question 9:

Correction :

Le programme suivant fait l'estimation sur les 2 premiers axes discriminants :

```
# lecture des données
textures = np.loadtxt('texture.dat')

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components=2)
lda.fit(textures[:,:40],textures[:,40])
texturest = lda.transform(textures[:,:40])

# définir la grille pour la visualisation
grid_size = 100
mx = min(texturest[:,0])
Mx = max(texturest[:,0])
my = min(texturest[:,1])
My = max(texturest[:,1])
xstep = (Mx - mx) / grid_size
ystep = (My - my) / grid_size
Gx = np.arange(mx, Mx, xstep)
Gy = np.arange(my, My, ystep)
Gx, Gy = np.meshgrid(Gx, Gy)

# définir la largeur de bande pour le noyau
bw = 0.25
# estimation, puis calcul densité sur la grille
kdet = KernelDensity(kernel='gaussian', bandwidth=bw).fit(texturest)
Z = np.exp(kdet.score_samples(np.hstack((
                    (Gx.reshape(grid_size*grid_size))[:,np.newaxis],
                    (Gy.reshape(grid_size*grid_size)[:,np.newaxis])))))

# affichage
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(Gx, Gy, Z.reshape(grid_size,grid_size), rstride=1,
                    cstride=1, cmap=cm.coolwarm, linewidth=0, antialiased=True)
ax.scatter(texturest[:,0], texturest[:,1], -0.05)
plt.show()
```