TME4- EM et le geyser Old Faithful

Partie obligatoire

Dans ce TME, l'objectif est de modéliser les éruptions du geyser **Old Faithful**. Il s'agit d'un geyser du parc national de Yellowstone. L'image ci-dessous, provenant de **wikipedia**, vous montre son aspect.

La durée des éruptions varie dans le temps ainsi que le délai entre deux éruptions consécutives. Le fichier **2015_tme4_faithful.txt** recense des couples (durée de l'éruption, délai jusqu'é la prochaine éruption). C'est en utilisant ce fichier que nous allons définir un modéle des éruptions du geyser.

1. Lecture des données

Téléchargez le fichier **2015_tme4_faithful.txt**. La fonction **read_file** : `string -> float np.2D-array` ci-dessous vous permettra de le lire et de sauvegarder ses données dans un tableau numpy é 2 dimensions : chaque ligne représente la durée d'une éruption ainsi que le délai jusqu'é la prochaine éruption : la premiére colonne du tableau contenant les durées et la deuxiéme les délais. Copiez-collez le code de cette fonction dans votre éditeur de code python.

```
import numpy as np
from math import *
from pylab import *

def read_file ( filename ):
    """
    Lit le fichier contenant les données du geyser Old Faithful
    """
    # lecture de l'en-téte
    infile = open ( filename, "r" )
    for ligne in infile:
        if ligne.find ( "eruptions waiting" ) != -1:
            break

    # ici, on a la liste des temps d'éruption et des délais d'irruptions
    data = []
    for ligne in infile:
        nb_ligne, eruption, waiting = [ float (x) for x in ligne.split () ]
        data.append ( eruption )
        data.append ( waiting )
    infile.close ()

    # transformation de la liste en tableau 2D
    data = np.asarray ( data )
    data.shape = ( int ( data.size / 2 ), 2 )

    return data

data = read_file ( "2015_tme4_faithful.txt" )
```

2. Loi normale bidimensionnelle

Vous l'avez compris, chaque donnée $\mathbf{x_i}$ de notre échantillon est en fait un couple, que nous noterons $\mathbf{x_i} = (x_i,z_i)$. Nous allons représenter la distribution de ces couples par des lois normales bidimensionnelles. On rappelle que la fonction de densité de la loi normale bidimensionnelle est :

$$f(x,z) = \frac{1}{2\pi\sigma_x\sigma_z\sqrt{1-\rho^2}}\exp\{-\frac{1}{2(1-\rho^2)}[(\frac{x-\mu_x}{\sigma_x})^2 - 2\rho\frac{(x-\mu_x)(z-\mu_z)}{\sigma_x\sigma_z} + (\frac{z-\mu_z}{\sigma_z})^2]\}$$

Ecrivez une fonction **normale_bidim** : `float x float x (float,float,float,float,float) -> float` qui, étant donné deux nombres réels x et z, ainsi que d'un quintuplet $(\mu_x, \mu_z, \sigma_x, \sigma_z, \rho)$ renvoie la valeur de la fonction de densité $f(x, z)$.

Vous pourrez tester votre fonction :

```
normale_bidim ( 1, 2, (1.0,2.0,3.0,4.0,0) )
```

```
0.013262911924324612
```

```
normale_bidim ( 1, 0, (1.0,2.0,1.0,2.0,0.7) )
```

```
0.041804799427614503
```

3. Visualisation de loi normale bidimensionnelle

Afin de bien comprendre la signification des paramétres σ_x, σ_z, ρ vous allez dessiner des isocontours de cette fonction de densité en utilisant la fonction **dessine_1_normale** (`params`) ci-dessous, que vous copierez-collerez dans votre éditeur de code python. Le paramétre `params` attendu est un quintuplet $(\mu_x, \mu_z, \sigma_x, \sigma_z, \rho)$ Vous pourrez tester cette fonction de la maniére suivante:

```
dessine_1_normale ( (-3.0,-5.0,3.0,2.0,0.7) )
```

puis

```
dessine_1_normale ( (-3.0,-5.0,3.0,2.0,0.2) )
```

Observez l'impact du paramétre ρ sur l'angle de rotation des ellipses.

```
import matplotlib.pyplot as plt

def dessine_1_normale ( params ):
    # récupération des paramétres
    mu_x, mu_z, sigma_x, sigma_z, rho = params

    # on détermine les coordonnées des coins de la figure
    x_min = mu_x - 2 * sigma_x
    x_max = mu_x + 2 * sigma_x
    z_min = mu_z - 2 * sigma_z
    z_max = mu_z + 2 * sigma_z

    # création de la grille
    x = np.linspace ( x_min, x_max, 100 )
    z = np.linspace ( z_min, z_max, 100 )
    X, Z = np.meshgrid(x, z)

    # calcul des normales
    norm = X.copy ()
    for i in range ( x.shape[0] ):
        for j in range ( z.shape[0] ):
            norm[i,j] = normale_bidim ( x[i], z[j], params )

    # affichage
    fig = plt.figure ()
    plt.contour ( X, Z, norm, cmap=cm.autumn )
    plt.show ()
```

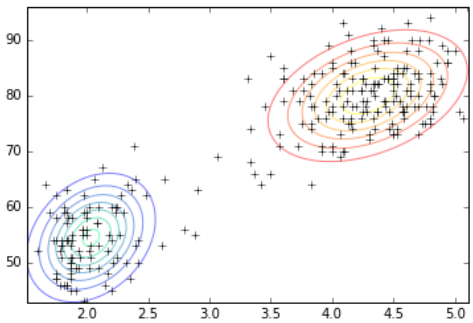
4. Visualisation des données du *Old Faithful*

Nous allons modéliser les éruptions du geyser Old Faithful gréce é une mixture de deux lois normales bidimensionnelles. Afin de comprendre pourquoi nous avons choisi ce modéle, visualisez les données en question utilisant le script ci-dessous. La fonction **dessine_normales** : `np.2D-array x (float,float,float,float,float) np.array x float np.array x float np.array x matplotlib.axes -> void` prend en argument le tableau *data* des données que vous avez lues é la question 1, un tableau *params* contenant les quintuplets des deux lois normales, un tableau *weights* de 2 éléments correspondant aux poids des lois normales, un tableau *bounds* définissant les coordonnées des coins de la figure (ce tableau est calculé par la fonction **find_bounds**). Enfin, le dernier paramétre, *ax*, sert é stocker le résultat de l'affichage. Il vous servira quand vous créerez une vidéo illustrant l'exécution de votre algorithme EM. La fin du script ci-dessous vous indique comment vous servir des fonctions **dessine_normales** et **find_bounds**. Exécutez-le.

```
def dessine_normales ( data, params, weights, bounds, ax ):
    # récupération des paramétres
    mu_x0, mu_z0, sigma_x0, sigma_z0, rho0 = params[0]
    mu_x1, mu_z1, sigma_x1, sigma_z1, rho1 = params[1]
```

```
    # on détermine les coordonnées des coins de la figure
    x_min = bounds[0]
    x_max = bounds[1]
    z_min = bounds[2]
    z_max = bounds[3]

    # création de la grille
    nb_x = nb_z = 100
    x = np.linspace ( x_min, x_max, nb_x )
    z = np.linspace ( z_min, z_max, nb_z )
    X, Z = np.meshgrid(x, z)

    # calcul des normales
    norm0 = np.zeros ( (nb_x,nb_z) )
    for j in range ( nb_z ):
        for i in range ( nb_x ):
            norm0[j,i] = normale_bidim ( x[i], z[j], params[0] )# * weights[0]
    norm1 = np.zeros ( (nb_x,nb_z) )
    for j in range ( nb_z ):
        for i in range ( nb_x ):
            norm1[j,i] = normale_bidim ( x[i], z[j], params[1] )# * weights[1]

    # affichages des normales et des points du dataset
    ax.contour ( X, Z, norm0, cmap=cm.winter, alpha = 0.5 )
    ax.contour ( X, Z, norm1, cmap=cm.autumn, alpha = 0.5 )
    for point in data:
        ax.plot ( point[0], point[1], 'k+' )


def find_bounds ( data, params ):
    # récupération des paramétres
    mu_x0, mu_z0, sigma_x0, sigma_z0, rho0 = params[0]
    mu_x1, mu_z1, sigma_x1, sigma_z1, rho1 = params[1]

    # calcul des coins
    x_min = min ( mu_x0 - 2 * sigma_x0, mu_x1 - 2 * sigma_x1, data[:,0].min() )
    x_max = max ( mu_x0 + 2 * sigma_x0, mu_x1 + 2 * sigma_x1, data[:,0].max() )
    z_min = min ( mu_z0 - 2 * sigma_z0, mu_z1 - 2 * sigma_z1, data[:,1].min() )
    z_max = max ( mu_z0 + 2 * sigma_z0, mu_z1 + 2 * sigma_z1, data[:,1].max() )

    return ( x_min, x_max, z_min, z_max )


# affichage des données : calcul des moyennes et variances des 2 colonnes
mean1 = data[:,0].mean ()
mean2 = data[:,1].mean ()
std1  = data[:,0].std ()
std2  = data[:,1].std ()

# les paramétres des 2 normales sont autour de ces moyennes
params = np.array ( [(mean1 - 0.2, mean2 - 1, std1, std2, 0),
                     (mean1 + 0.2, mean2 + 1, std1, std2, 0)] )
weights = np.array ( [0.4, 0.6] )
bounds = find_bounds ( data, params )

# affichage de la figure
fig = plt.figure ()
ax = fig.add_subplot(111)
dessine_normales ( data, params, weights, bounds, ax )
plt.show ()
```

Vous devriez observer un affichage similaire é celui-ci-dessous, qui montre bien que les données sont approximativement "réparties" selon deux gaussiennes.



5. EM : l'étape E

Nous allons maintenant écrire l'étape E de l'algorithme EM, qui va nous permettre d'estimer les paramétres de la mixture de lois normales bidimensionnelles. Ainsi, comme vu en cours, soit Y une variable aléatoire indiquant quelle classe/loi normale bidimensionnelle a généré le couple de données (x,z): $Y = y_0$ indique que le couple (x,z) a été généré par la premiére loi normale bidimensionnelle tandis que $Y = y_0$ indique que le couple (x,z) a été généré par la seconde loi normale bidimensionnelle. Par conséquent, si Θ^t représente l'ensemble des paramétres des deux lois normales:

$$\Theta^t = \{(\pi_0, \mu_{x0}, \mu_{z0}, \sigma_{x0}, \sigma_{z0}, \rho_0), (\pi_1, \mu_{x1}, \mu_{z1}, \sigma_{x1}, \sigma_{z1}, \rho_1)\}$$

L'étape E consiste é calculer, pour chaque couple de données (x_i, z_i) du tableau `data` des données du old faithful lues é la question 1, les probabilités $Q_i^{t+1}(y_0) = P(y_0|(x_i, z_i), \Theta^t)$ et $Q_i^{t+1}(y_1) = P(y_1|(x_i, z_i), \Theta^t)$. Cela correspond aux formules suivantes:

$$\alpha_0 = \pi_0 \frac{1}{2\pi\sigma_{x0}\sigma_{z0}\sqrt{1-\rho_0^2}} \text{é}$$

$$\exp\{-\frac{1}{2(1-\rho_0^2)}[(\frac{x_i-\mu_{x0}}{\sigma_{x0}})^2 - 2\rho_0\frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}} + (\frac{z_i-\mu_{z0}}{\sigma_{z0}})^2]\}$$

$$\alpha_1 = \pi_1 \frac{1}{2\pi\sigma_{x1}\sigma_{z1}\sqrt{1-\rho_1^2}} \text{é}$$

$$\exp\{-\frac{1}{2(1-\rho_1^2)}[(\frac{x_i-\mu_{x1}}{\sigma_{x1}})^2 - 2\rho_1\frac{(x_i-\mu_{x1})(z_i-\mu_{z1})}{\sigma_{x1}\sigma_{z1}} + (\frac{z_i-\mu_{z1}}{\sigma_{z1}})^2]\}$$

$$Q_i^{t+1}(y_0) = \frac{\alpha_0}{\alpha_0+\alpha_1} \qquad Q_i^{t+1}(y_1) = \frac{\alpha_1}{\alpha_0+\alpha_1}$$

Ecrivez une fonction **Q_i**: `float np.2D-array x float np.2D-array x float np.array -> float np.2D-array` qui, étant donné le tableau `data` des données du old faithful lues é la question 1, un tableau `current_params` é 2 dimensions contenant les 2 quintuplets de paramétres des lois normales (cf. la fin du script ci-dessus) et un tableau `current_weights` contenant les poids (π_0, π_1) des deux lois normales, renvoie un tableau é deux dimensions correspondant aux $Q_i^{t+1}(y_j) = P(y_j|(x_i, z_i), \Theta^t)$ oé j =0,1. Si `T` représente le tableau renvoyé par la fonction **Q_i**, alors `T[i,j]` contient le nombre $Q_i^{t+1}(y_j) = P(y_j|(x_i, z_i), \Theta^t)$ autrement dit `T` a 2 colonnes (correspondant aux 2 lois normales) et *n* lignes, oé *n* est le nombre de couples observés dans la base de données de la question 1.

Ainsi, l'exécution du code python ci-dessous:

```
#current_params = np.array ( [(mu_x, mu_z, sigma_x, sigma_z, rho),   # params 1ére loi normale
#                              (mu_x, mu_z, sigma_x, sigma_z, rho)] ) # params 2éme loi normale
current_params = np.array([[ 3.28778309, 69.89705882, 1.13927121, 13.56996002, 0. ],
                           [ 3.68778309, 71.89705882, 1.13927121, 13.56996002, 0. ]])

# current_weights = np.array ( [ pi_0, pi_1 ] )
current_weights = np.array ( [ 0.5, 0.5 ] )

T = Q_i ( data, current_params, current_weights )
```

produira un `T` de 272 lignes et 2 colonnes égal é:

```
array([[ 0.46939088,  0.53060912],
       [ 0.66899936,  0.33100064],
       [ 0.50349992,  0.49650008],
       [ 0.61489199,  0.38510801],
       [ 0.38336517,  0.61663483],
       [ 0.58880845,  0.41119155],
       [ 0.36370054,  0.63629946],
       [ 0.45319835,  0.54680165],
       [ 0.66597113,  0.33402887],
       ...................
       [ 0.3932781 ,  0.6067219 ],
       [ 0.42466754,  0.57533246],
       [ 0.66433838,  0.33566162],
       [ 0.3789881 ,  0.6210119 ],
       [ 0.68682391,  0.31317609],
       [ 0.41690852,  0.58309148]])
```

Et l'exécution du code python ci-dessous:

```
current_params = np.array([[ 3.2194684, 67.83748075, 1.16527301, 13.9245876,  0.9070348 ],
                           [ 3.75499261, 73.9440348, 1.04650191, 12.48307362, 0.88083712]])
current_weights = np.array ( [ 0.49896815, 0.50103185] )
T = Q_i ( data, current_params, current_weights )
```

produira un `T` de 272 lignes et 2 colonnes égal é:

```
array([[ 0.44352868,  0.55647132],
       [ 0.70465534,  0.29534466],
       [ 0.47692866,  0.52307134],
       [ 0.61531052,  0.38468948],
       [ 0.38502072,  0.61497928],
       [ 0.58493721,  0.41506279],
       [ 0.37621272,  0.62378728],
       [ 0.42185109,  0.57814891],
       [ 0.70665534,  0.29334466],
```

```
        [ 0.39346396,  0.60653604],
        ....................
        [ 0.73879683,  0.26120317],
        [ 0.62962181,  0.37037819],
        [ 0.36768604,  0.63231396],
        [ 0.41080893,  0.58919107],
        [ 0.70723862,  0.29276138],
        [ 0.38400503,  0.61599497],
        [ 0.74441719,  0.25558281],
        [ 0.39084155,  0.60915845]])
```

6. EM : l'étape M

Ecrivez maintenant l'étape M de l'algorithme EM, c'est-é-dire l'étape dans laquelle on met é jour les quintuplets de paramétres. Pour cela, écrivez une fonction **M_step** : `np.2D-array x np.2D-array x float np.2D-array x float np.array -> float np.2D-array x float np.array` qui, étant donné les données *data* lues é la question 1, un tableau *T* renvoyé par votre fonction **Q_i**, et les quintuplets et poids courants, renvoie un couple (new_params, new_weights) contenant un tableau é deux dimensions np.array([$[\mu_{x0}, \mu_{z0}, \sigma_{x0}, \sigma_{z0}, \rho_0]$,$[\mu_{x1}, \mu_{z1}, \sigma_{x1}, \sigma_{z1}, \rho_1]$]) avec les 2 nouveaux quintuplets et un tableau np.array ($[\pi_0, \pi_1]$) avec les deux nouveaux poids. Les formules pour mettre é jour ces quantités sont les suivantes (cf. les **démonstrations** tout en bas de l'énoncé du TME) :

$$\pi_0 = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0)}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)+Q_i^{t+1}(y_1)} \qquad \pi_1 = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_1)}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)+Q_i^{t+1}(y_1)}$$

$$\mu_{x0} = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0)x_i}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)} \qquad \mu_{x1} = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_1)x_i}{\sum_{i=1}^{n} Q_i^{t+1}(y_1)}$$

$$\mu_{z0} = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0)z_i}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)} \qquad \mu_{z1} = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_1)z_i}{\sum_{i=1}^{n} Q_i^{t+1}(y_1)}$$

$$\sigma_{x0} = \sqrt{\frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0)(x_i-\mu_{x0})^2}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)}} \qquad \sigma_{x1} = \sqrt{\frac{\sum_{i=1}^{n} Q_i^{t+1}(y_1)(x_i-\mu_{x1})^2}{\sum_{i=1}^{n} Q_i^{t+1}(y_1)}}$$

$$\sigma_{z0} = \sqrt{\frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0)(z_i-\mu_{z0})^2}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)}} \qquad \sigma_{z1} = \sqrt{\frac{\sum_{i=1}^{n} Q_i^{t+1}(y_1)(z_i-\mu_{z1})^2}{\sum_{i=1}^{n} Q_i^{t+1}(y_1)}}$$

$$\rho_0 = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0)\frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}}}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)} \qquad \rho_1 = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_1)\frac{(x_i-\mu_{x1})(z_i-\mu_{z1})}{\sigma_{x1}\sigma_{z1}}}{\sum_{i=1}^{n} Q_i^{t+1}(y_1)}$$

Attention: calculez bien ces formules du haut vers le bas, elles ne sont pas indépendantes (par exemple, la valeur de σ_{x0} dépend de celle de μ_{x0}).

Par exemple, le code suivant:

```
current_params = array([(2.51460515, 60.12832316, 0.90428702, 11.66108819, 0.86533355),
                        (4.2893485,  79.76680985, 0.52047055,  7.04450242, 0.58358284)])
current_weights = array([ 0.45165145,  0.54834855])
Q = Q_i ( data, current_params, current_weights )
M_step ( data, Q, current_params, current_weights )
```

retournera un couple:

```
(array([[ 2.33418412, 58.06784269, 0.74224878, 10.17591317, 0.82161824],
        [ 4.33880698, 80.36132657, 0.37819574,  5.71033527, 0.3008745 ]]),
 array( [ 0.42453067,  0.57546933 ] ) )
```

7. Algorithme EM : mise au point

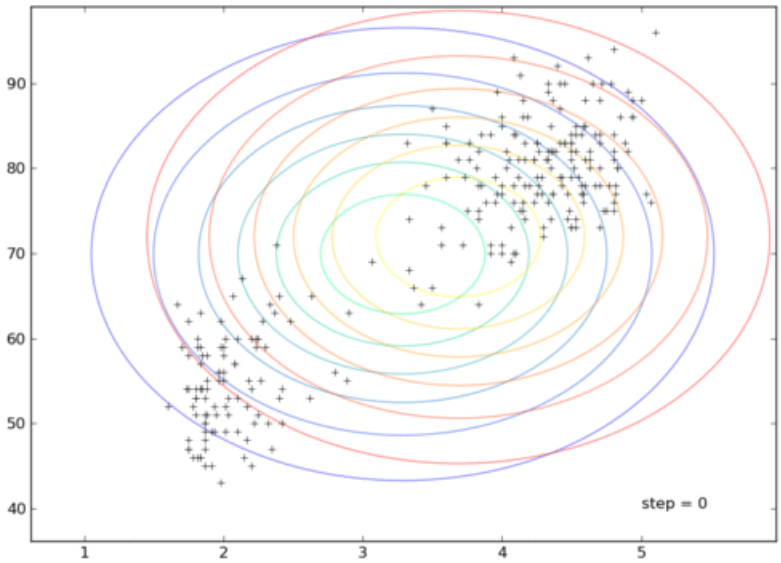
Ecrivez maintenant une premiére version de votre algorithme EM qui, étant donné le tableau de données lues é la question 1, exécute l'algorithme EM sur quelques itérations (3 ou 4). A chaque itération, utilisez la fonction **dessine_normales** afin de visualiser l'évolution des paramétres de vos normales. Pour initialiser EM, vous partirez des paramétres proposés dans la question 4 :

```
mean1 = data[:,0].mean ()
mean2 = data[:,1].mean ()
std1  = data[:,0].std ()
std2  = data[:,1].std ()
params = np.array ( [(mean1 - 0.2, mean2 - 1, std1, std2, 0),
                     (mean1 + 0.2, mean2 + 1, std1, std2, 0)] )
weights = np.array ( [ 0.5, 0.5 ] )
```

Partie optionnelle

8. Algorithme EM : version finale et animation

Maintenant que votre algorithme est au point, écrivez-en une nouvelle version qui ne crée plus des figures é chaque itération mais, plutét, qui sauvegarde dans une liste les couples (paramétres,poids) calculés. Votre fonction exécutera 20 pas de temps de l'algorithme EM et renverra cette liste (appelons-la res_EM). Le code ci-dessous permet alors de créer une animation des résultats obtenus : la fonction **find_video_bounds** calcule en effet la "bounding box" de la figure, et le code en dessous crée l'animation en question. Vous devriez obtenir une animation similaire é celle vue en cours, c'est-é-dire comme celle ci-dessous. Vous pouvez méme la sauvegarder dans une vidéo.




```
# calcul des bornes pour contenir toutes les lois normales calculées
def find_video_bounds ( data, res_EM ):
    bounds = np.asarray ( find_bounds ( data, res_EM[0][0] ) )
    for param in res_EM:
        new_bound = find_bounds ( data, param[0] )
        for i in [0,2]:
            bounds[i] = min ( bounds[i], new_bound[i] )
        for i in [1,3]:
            bounds[i] = max ( bounds[i], new_bound[i] )
    return bounds

bounds = find_video_bounds ( data, res_EM )


import matplotlib.animation as animation

# création de l'animation : tout d'abord on crée la figure qui sera animée
fig = plt.figure ()
ax = fig.gca (xlim=(bounds[0], bounds[1]), ylim=(bounds[2], bounds[3]))

# la fonction appelée é chaque pas de temps pour créer l'animation
def animate ( i ):
    ax.cla ()
    dessine_normales (data, res_EM[i][0], res_EM[i][1], bounds, ax)
    ax.text(5, 40, 'step = ' + str ( i ))
    print "step animate = %d" % ( i )

# exécution de l'animation
anim = animation.FuncAnimation(fig, animate,
                               frames = len ( res_EM ), interval=500 )
plt.show ()

# éventuellement, sauver l'animation dans une vidéo
# anim.save('old_faithful.avi', bitrate=4000)
```

Démonstration des formules de la question 6

On suppose ici que les couples de durée et de délai (x, z) suivent une mixture de lois normales bidimensionnelles. La distribution de probabilité que l'on cherche é estimer est donc :

$$P(x, z|\Theta) = \pi_0 \mathrm{N}\,(\mu_{x0}, \mu_{z0}, \sigma_{x0}, \sigma_{z0}, \rho_0)(x, z) + \pi_1 \mathrm{N}\,(\mu_{x1}, \mu_{z1}, \sigma_{x1}, \sigma_{z1}, \rho_1)(x, z)$$

On notera $f(\mu_x, \mu_z, \sigma_x, \sigma_z, \rho)$la fonction de densité d'une loi normale bidimensionnelle :

$$f_{\mu_x,\mu_z,\sigma_x,\sigma_z,\rho}(x,z) = \frac{1}{2\pi\sigma_x\sigma_z\sqrt{1-\rho^2}}$$
$$\exp\{-\frac{1}{2(1-\rho^2)}[(\frac{x-\mu_x}{\sigma_x})^2 - 2\rho\frac{(x-\mu_x)(z-\mu_z)}{\sigma_x\sigma_z} + (\frac{z-\mu_z}{\sigma_z})^2]\}.$$

Par conséquent, le logarithme de cette fonction est :

$$\log f_{\mu_x,\mu_z,\sigma_x,\sigma_z,\rho}(x,z) = -\log(2\pi) - \log(\sigma_x) - \log(\sigma_z) - \frac{1}{2}\log(1-\rho^2)$$
$$-\frac{1}{2(1-\rho^2)}[(\frac{x-\mu_x}{\sigma_x})^2 - 2\rho\frac{(x-\mu_x)(z-\mu_z)}{\sigma_x\sigma_z} + (\frac{z-\mu_z}{\sigma_z})^2].$$

L'étape M consiste é calculer :

$$\text{Argmax}_\Theta \log(L^{t+1}(\mathbf{x}^o,\Theta)) = \text{Argmax}_\Theta \sum_{i=1}^{n}\sum_{k=0}^{1} Q_i^{t+1}(y_k)\log(\frac{p(x_i,z_i,y_k|\Theta)}{Q_i^{t+1}(y_k)})$$

$$= \text{Argmax}_\Theta \sum_{i=1}^{n} Q_i^{t+1}(y_0)\log(\pi_0 f_{\mu_{x0},\mu_{z0},\sigma_{x0},\sigma_{z0},\rho_0}(x,z)) + Q_i^{t+1}(y_1)\log(\pi_1 f_{\mu_{x1},\mu_{z1},\sigma_{x1},\sigma_{z1},\rho_1}(x,z))$$

Donc, étant donné que $\pi_1 = 1-\pi_0$

$$\frac{\partial\log(L^{t+1}(\mathbf{x}^o,\Theta))}{\partial\pi_0} = \sum_{i=1}^{n} Q_i^{t+1}(y_0)\frac{1}{\pi_0} - \sum_{i=1}^{n} Q_i^{t+1}(y_1)\frac{1}{1-\pi_0} = 0$$

D'oé :

$$\pi_0 = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0)}{\sum_{i=1}^{n} Q_i^{t+1}(y_0) + Q_i^{t+1}(y_1)} \qquad (1)$$

et

$$\pi_1 = 1-\pi_0 = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_1)}{\sum_{i=1}^{n} Q_i^{t+1}(y_0) + Q_i^{t+1}(y_1)} \qquad (2)$$

Calculons maintenant les expressions des μ:

$$\frac{\partial\log(L^{t+1}(\mathbf{x}^o,\Theta))}{\partial\mu_0} = \frac{\partial}{\partial\mu_{x0}}\sum_{i=1}^{n} Q_i^{t+1}(y_0)(-\frac{1}{2(1-\rho_0^2)})[(\frac{x_i-\mu_{x0}}{\sigma_{x0}})^2 - 2\rho_0\frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}}]$$

$$\Longleftrightarrow \sum_{i=1}^{n} Q_i^{t+1}(y_0)(-\frac{1}{2(1-\rho_0^2)})[2(\frac{x_i-\mu_{x0}}{\sigma_{x0}^2}) - 2\rho_0\frac{z_i-\mu_{z0}}{\sigma_{x0}\sigma_{z0}}] = 0$$

ce qui est équivalent é :

$$\sum_{i=1}^{n} Q_i^{t+1}(y_0)[2(\frac{x_i-\mu_{x0}}{\sigma_{x0}}) - 2\rho_0\frac{z_i-\mu_{z0}}{\sigma_{z0}}] = 0 \qquad (3)$$

Par symmétrie, en dérivant par rapport é μ_{z0} on a également :

$$\sum_{i=1}^{n} Q_i^{t+1}(y_0)[2(\frac{z_i-\mu_{z1}}{\sigma_{z1}}) - 2\rho_0\frac{x_i-\mu_{x0}}{\sigma_{x0}}] = 0 \qquad (4)$$

Si l'on ajoute ρ_0 fois l'équation (4)é l'équation (3), on obtient :

$$\sum_{i=1}^{n} Q_i^{t+1}(y_0)\text{ é }2[\frac{1-\rho_0^2}{\sigma_{x0}}](\mu_{x0}-x_i) = 0$$

ce qui est équivalent é :

$$\mu_{x0} = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0) x_i}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)} \qquad (5)$$

Par symmétrie, on a :

$$\mu_{z0} = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0) z_i}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)} \qquad (6)$$

Calculons maintenant les expressions des σ et de ρ :

$$\frac{\partial \log(L^{t+1}(\mathbf{x}^o,\Theta))}{\partial \sigma_{x0}} = \frac{\partial}{\partial \sigma_{x0}} \sum_{i=1}^{n} Q_i^{t+1}(y_0) \{-\log(\sigma_{x0}) - \frac{1}{2(1-\rho_0^2)} [(\frac{x_i-\mu_{x0}}{\sigma_{x0}})^2 - 2\rho_0 \frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}}]\}$$

$$= \sum_{i=1}^{n} Q_i^{t+1}(y_0) \{-\frac{1}{\sigma_{x0}} - \frac{1}{2(1-\rho_0^2)} [-2\frac{(x_i-\mu_{x0})^2}{\sigma_{x0}^3} + 2\rho_0 \frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}^2}]\} = 0$$

$$\Longleftrightarrow \sum_{i=1}^{n} Q_i^{t+1}(y_0) \{-1 - \frac{1}{2(1-\rho_0^2)} [-2\frac{(x_i-\mu_{x0})^2}{\sigma_{x0}^2} + 2\rho_0 \frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}}]\} = 0$$

ce qui est équivalent é :

$$2(1-\rho_0^2)\sum_{i=1}^{n} Q_i^{t+1}(y_0) = \sum_{i=1}^{n} Q_i^{t+1}(y_0) \{[2\frac{(x_i-\mu_{x0})^2}{\sigma_{x0}^2} - 2\rho_0 \frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}}]\} \qquad (7)$$

Par symmétrie, quand on dérive par rapport é σ_{z0}, on obtient :

$$2(1-\rho_0^2)\sum_{i=1}^{n} Q_i^{t+1}(y_0) = \sum_{i=1}^{n} Q_i^{t+1}(y_0) \{[2\frac{(z_i-\mu_{z0})^2}{\sigma_{z0}^2} - 2\rho_0 \frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}}]\} \qquad (8)$$

En additionnant les 2 équations (7) et (8), on obtient:

$$4(1-\rho_0^2)\sum_{i=1}^{n} Q_i^{t+1}(y_0) = 2\sum_{i=1}^{n} Q_i^{t+1}(y_0) \{[\frac{(x_i-\mu_{x0})^2}{\sigma_{x0}^2} - 2\rho_0 \frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}} + \frac{(z_i-\mu_{z0})^2}{\sigma_{z0}^2}]\} \qquad (9$$

Enfin, dérivons la log-vraisemblance par rapport é ρ_0:

$$\frac{\partial \log(L^{t+1}(\mathbf{x}^o,\Theta))}{\partial \rho_0} = \frac{\partial}{\partial \rho_{x0}} \sum_{i=1}^{n} Q_i^{t+1}(y_0) \{-\frac{1}{2}\log(1-\rho_0^2)$$
$$- \frac{1}{2(1-\rho_0^2)} [\frac{(x_i-\mu_{x0})^2}{\sigma_{x0}^2} - 2\rho_0 \frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}} + \frac{(z_i-\mu_{z0})^2}{\sigma_{z0}^2}]\}$$

$$= \sum_{i=1}^{n} Q_i^{t+1}(y_0) \{\frac{\rho_0}{1-\rho_0^2}$$
$$- \frac{\rho_0}{(1-\rho_0^2)^2} [\frac{(x_i-\mu_{x0})^2}{\sigma_{x0}^2} - 2\rho_0 \frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}} + \frac{(z_i-\mu_{z0})^2}{\sigma_{z0}^2}]$$
$$+ \frac{1}{1-\rho_0^2} [\frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}}]\} = 0$$

$$\Longleftrightarrow \sum_{i=1}^{n} Q_i^{t+1}(y_0) \{\rho_0$$
$$- \frac{\rho_0}{(1-\rho_0^2)} [\frac{(x_i-\mu_{x0})^2}{\sigma_{x0}^2} - 2\rho_0 \frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}} + \frac{(z_i-\mu_{z0})^2}{\sigma_{z0}^2}]$$
$$+ [\frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}}]\} = 0$$

En remplaéant le 2éme terme par le membre gauche de l'équation (9), on obtient :

$$\sum_{i=1}^{n} Q_i^{t+1}(y_0)\,\{\rho_0 - 2\rho_0 + [\frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}}]\} = 0$$

Par conséquent,

$$\rho_0 = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0)\,\frac{(x_i-\mu_{x0})(z_i-\mu_{z0})}{\sigma_{x0}\sigma_{z0}}}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)} \qquad (10)$$

Appelons $\eta = \sum_{i=1}^{n} Q_i^{t+1}(y_0)(x_i-\mu_{x0})(z_i-\mu_{z0})$ Alors:

$$\rho_0 = \frac{\eta}{\sigma_{x0}\sigma_{z0}\sum_{i=1}^{n} Q_i^{t+1}(y_0)}$$

En remplaéant ρ_0 par son expression dans l'équation (8), on obtient:

$$2\left(1 - \left(\frac{1}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)}\right)^2 \frac{\eta^2}{\sigma_{x0}^2\sigma_{z0}^2}\right)\sum_{i=1}^{n} Q_i^{t+1}(y_0) =$$

$$2\left(\sum_{i=1}^{n} Q_i^{t+1}(y_0)\,\frac{(x_i-\mu_{x0})^2}{\sigma_{x0}^2}\right) - 2\,\frac{\eta^2}{\sigma_{x0}^2\sigma_{z0}^2\sum_{i=1}^{n} Q_i^{t+1}(y_0)}$$

ce qui est équivalent é

$$\sigma_{x0}^2\sum_{i=1}^{n} Q_i^{t+1}(y_0) = \sum_{i=1}^{n} Q_i^{t+1}(y_0)(x_i-\mu_{x0})^2.$$

D'oé

$$\sigma_{x0}^2 = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0)(x_i-\mu_{x0})^2}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)} \qquad (11)$$

Par symmétrie,

$$\sigma_{z0}^2 = \frac{\sum_{i=1}^{n} Q_i^{t+1}(y_0)(z_i-\mu_{z0})^2}{\sum_{i=1}^{n} Q_i^{t+1}(y_0)} \qquad (12)$$