TD3 EM

Nicolas Jouvin

```
library(dplyr)
library(ggplot2)
library(knitr)
```

Exercice 6 : Algorithme EM pour les mélanges gaussiens en 1-D

Création des données de l'exercice

On créé les données

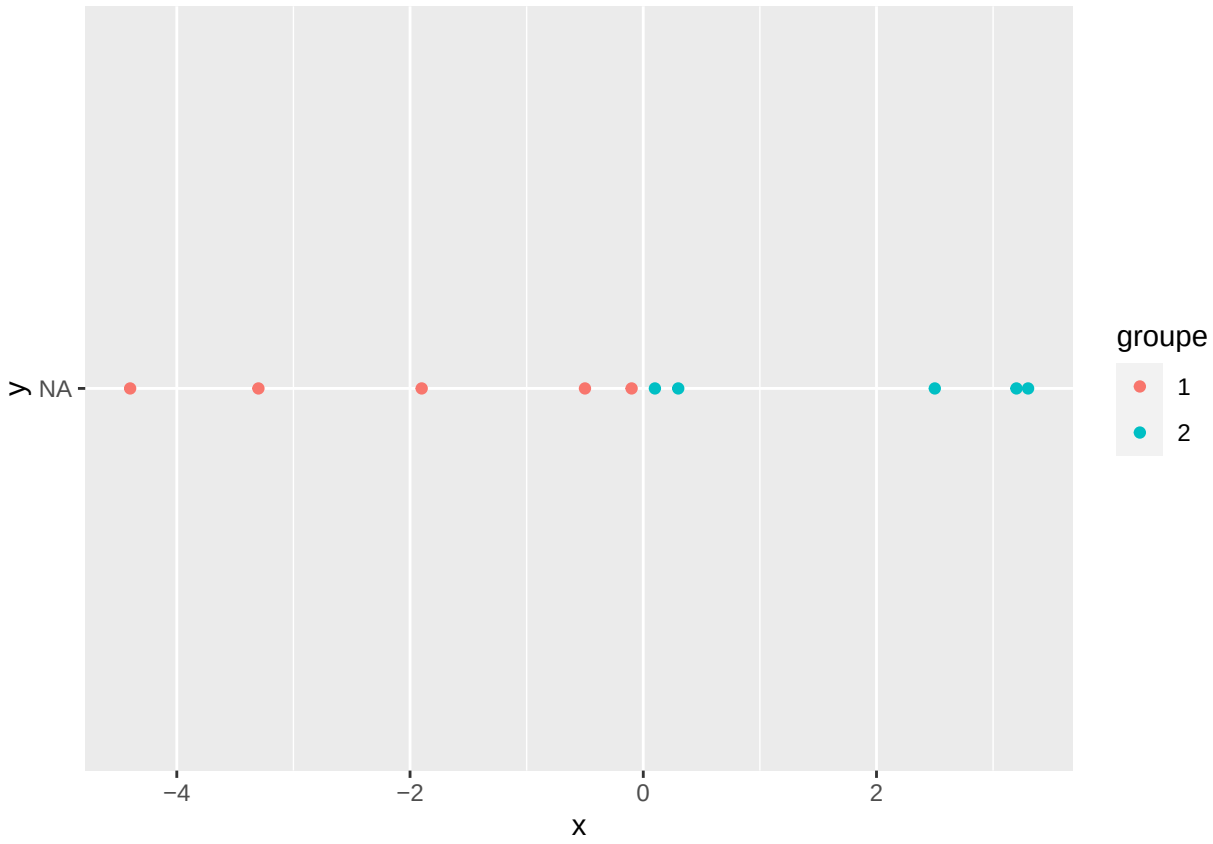
```
donnees<-matrix(c(-3.3,-4.4,-1.9,3.3,2.5,3.2,0.3,0.1,-0.1,-0.5, 1,1,1,2,2,2,2,2,1,1, 1,3,2,1,3,2,1,3,2,
donnees<-as.data.frame(donnees)
names(donnees)<-c("Var","partition1","partition2")
t(donnees)
```

```
##            [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## Var        -3.3 -4.4 -1.9  3.3  2.5  3.2  0.3  0.1 -0.1  -0.5
## partition1  1.0  1.0  1.0  2.0  2.0  2.0  2.0  2.0  1.0   1.0
## partition2  1.0  3.0  2.0  1.0  3.0  2.0  1.0  3.0  2.0   1.0
```

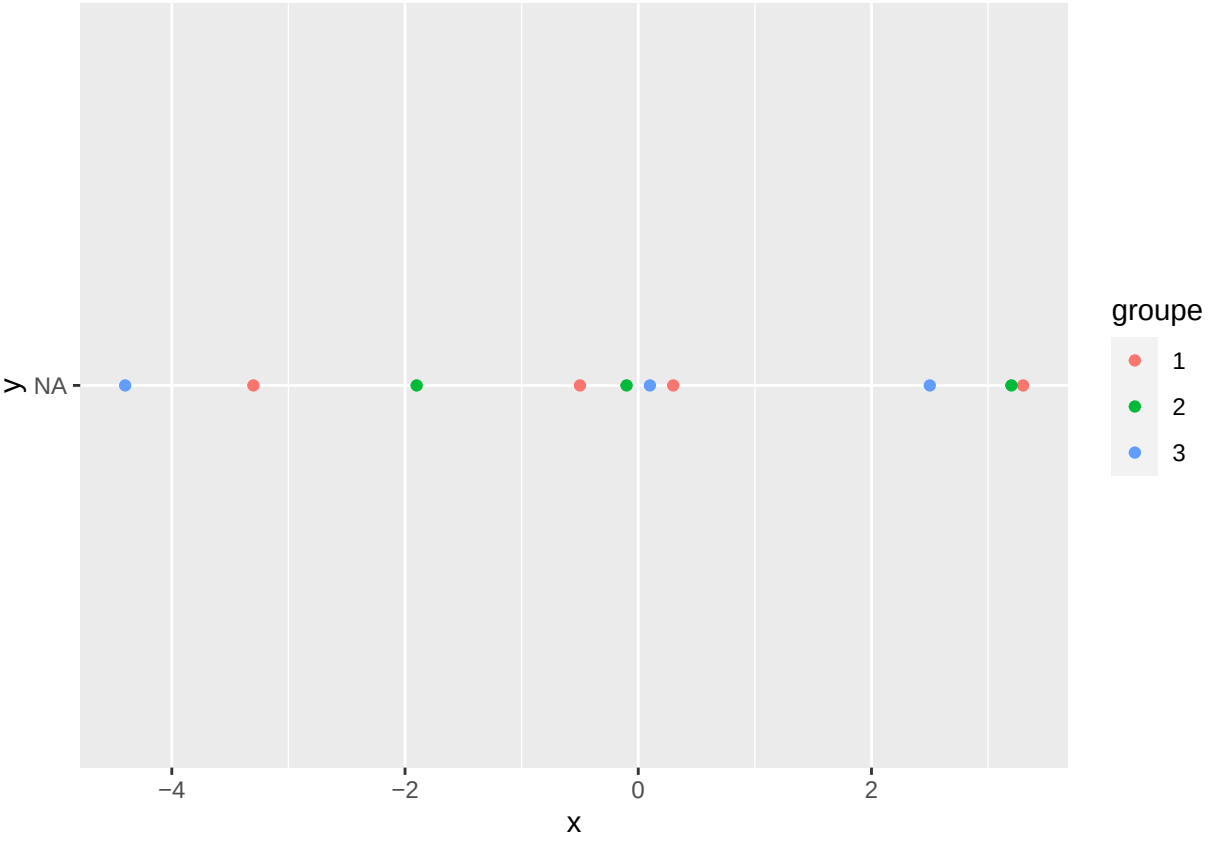
```
library(ggplot2)

plot_data <- function(x, partition) {
  # function that plot the 1D data vector x with color
  # according to ^argument partition
  # return : a ggplot graph
  df = data.frame(x = x, groupe = factor(partition))
  gg = ggplot(df) + geom_point(aes(x=x, y = NA, color=groupe))
  return(gg)
}

gg_part1 = plot_data(donnees$Var, donnees$partition1)
gg_part2 = plot_data(donnees$Var, donnees$partition2)
print(gg_part1)
```




```
print(gg_part2)
```



The second initialization do not seem really clever...

Question 1 : initialisation de l'algorithme

Coder la fonction `initEM(x, partition)` qui retourne une liste `param` avec slots

- `param$pi` : l'init $\pi^{(0)}$
- `param$theta` une liste avec slot
  - `param$theta$mu`: l'init $\mu^{(0)}$
  - `param$theta$sigma2` : l'init ${\sigma^2}^{(0)}$

Etape E

Coder une fonction `Estep(x, param)` qui calcule et renvoie les $\tau_{ik} \propto \pi_k \mathcal{N}(x_i \mid \mu_k, \sigma_k^2)$.

Astuce calculer en log-space pour mieux représenter les petites quantités ($e^{-1000} \approx 0$ tandis que $\log(e^{-1000}) = -1000$).

$$\log \tau_{ik} = \log \pi_k + \log \mathcal{N}(x_i \mid \mu_k, \sigma_k^2) - cte_i$$

La constante de normalisation peut-être calculée comme suit $cte_i = \log \sum_l \exp\{\log \tau_{il}\}$

Etape M

Coder

- une fonction `compute_PI(tau)` qui calcule $\hat{\pi}$.
- une fonction `compute_mu(tau, x)` qui calcule $\hat{\mu}_k$ pour tout k.

- une fonction `compute_sigma2(tau, mu, x)` qui calcule $\hat{\sigma}^2$.

Les compiler dans une fonction `Mstep(x, tau)` qui fait la M-step de l'algorithme EM.

Calcul de la vraisemblance marginale

Coder une fonction `compute_mixture_llhood = function(x, param)` qui retourn la log-vraisemblance marginale des observations : $\log p(x_1, \ldots, x_n \mid \theta, \pi) = \sum_i \log \sum_k \pi_k \mathcal{N}(x_i \mid \mu_k, \sigma_k^2)$.

Algo EM

Mettre toute ces fonctions ensemble dans une fonction `EMgauss1D(X, K, partition_init, max.iter, rtol)`. L'algorithme s'arrête après un nombre fixé `max.iter` d'itérations ou quand la différence relative entre les vraisemblances successive à t et $t+1$ est inférieur au seuil `rtol`.

La fonction retourne une liste avec slots

- `logliks` : la valeur successive des vraisemblance le long des itération (pour l'afficher dans un graphique par exemple)
- `param` : les paramètre finaux à la fin de l'algorithme
- `tau` : les probabilité a posteriori d'appartenir à chacuns des groupes. **Attention**, elles doivent calculées avec les valeurs des paramètres **finales**.

Tester votre fonction avec

Afficher l'évolution de la log-vraisemblance en fonctions des itérations de l'algorithme.

> **Note** On peut jouer avec le paramètre `max.iter` et `rtol` pour la convergence de la vraisemblance. Ne pas oublier que l'on converge uniquement vers un maximum **local** (en fait pire : un point selle) de cette dernière. On peut ensuite visualiser les estimateur des paramètre $(\hat{\pi_k}, \hat{\mu_k}, \hat{\Sigma_k})_k$

Afficher les paramètres estimés dans chacuns des clusters.

Clustering (partitionnement) à la fin de l'EM

On affecte les points selon leurs probabilité à posteriori après convergence de l'algorithme à l'itération (T). Cela revient à estimer $z_i, \forall i$:

$$\forall i, \quad \hat{z}_{ik^\star} = 1 \text{ où : } k^\star = \arg\max_{k=1,\ldots,K} p(z_{ik} = 1 \mid x_i, \theta^{(T)}) = \frac{\pi_k^{(T)} \mathcal{N}(x_i, \mu_k^{(T)}, \Sigma_k^{(T)})}{\sum_l \pi_l^{(T)} \mathcal{N}(x_i, \mu_l^{(T)}, \Sigma_l^{(T)})}$$

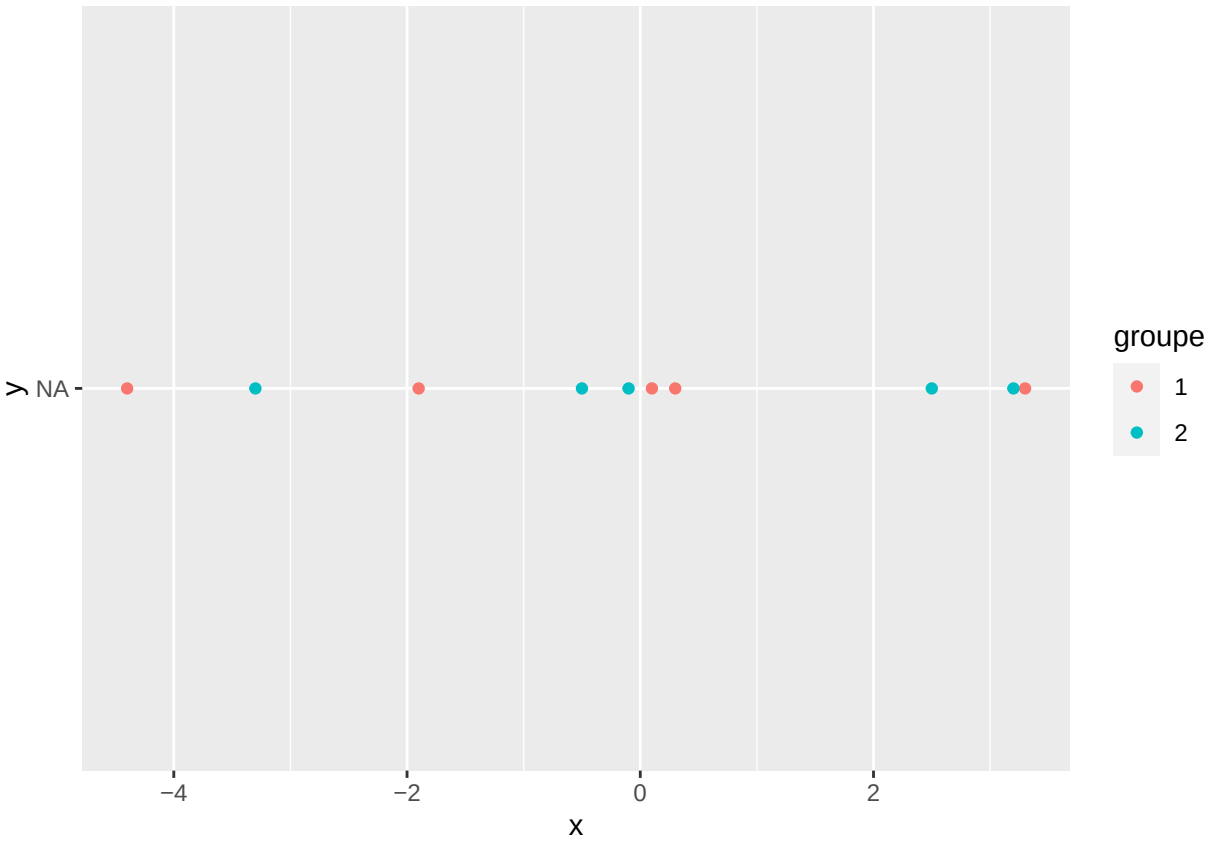
En fait cela revient à faire un argmax par ligne sur la matrix `res_em$tau`.

Calculer la partition obtenue grâce à cette méthode et faire un graphique avec les point coloré selon cette partition (utiliser `plot_data()`)

Impact de l'initialisation

Refaite un test avec une initialisation aléatoire (donc probablement mauvaise). La log vraisemblance va-t-elle augmenter à chaque itération dans ce cas de figure ? Qu'en est-il des paramètres estimés ?

```
partition_init = sample(1:K, size=10, replace=T)
plot_data(donnees$Var, partition_init)
```



Pour aller plus loin :

Relancer la procédure avec $K = 3$ en partant d'une partition initiale choisie au hasard ou celle de l'exercice au choix.