TD3 EM

Nicolas Jouvin

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

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

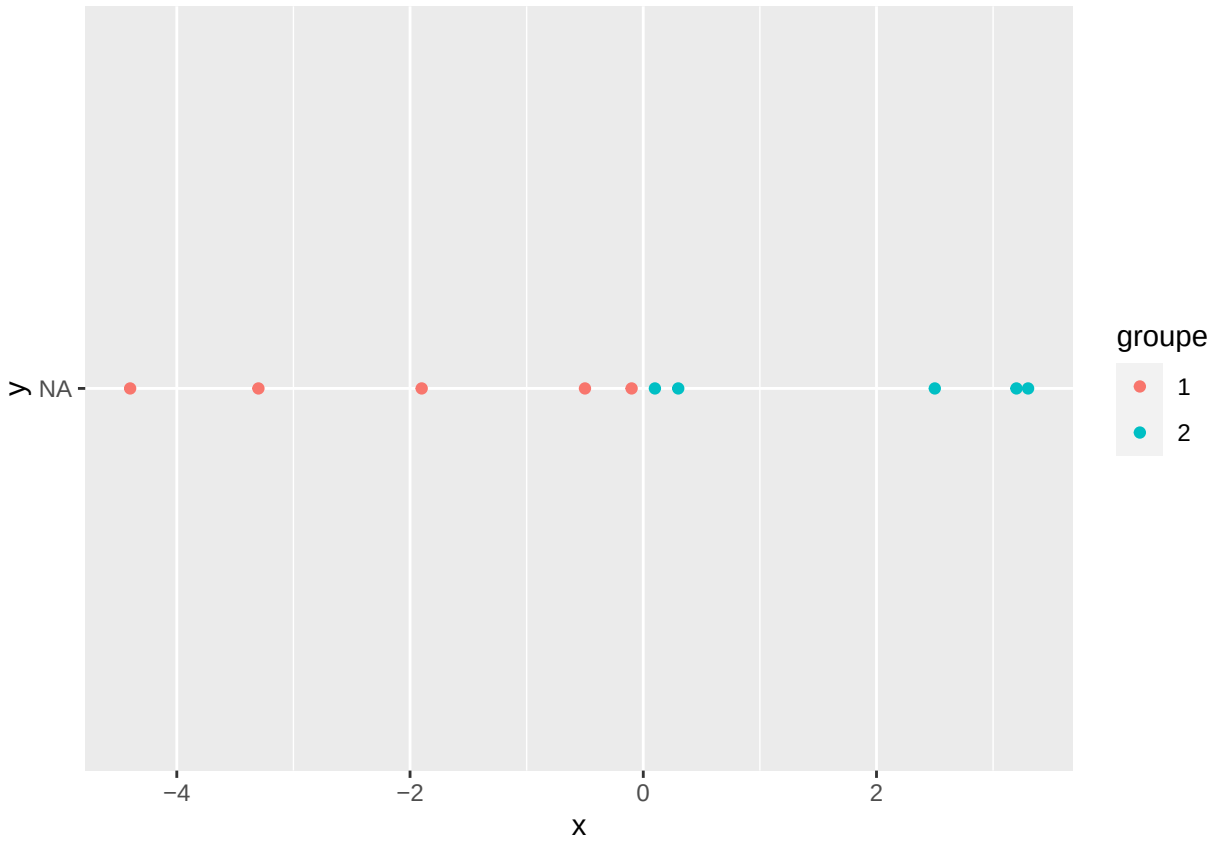
Création des données de l'exercice

On créé les données

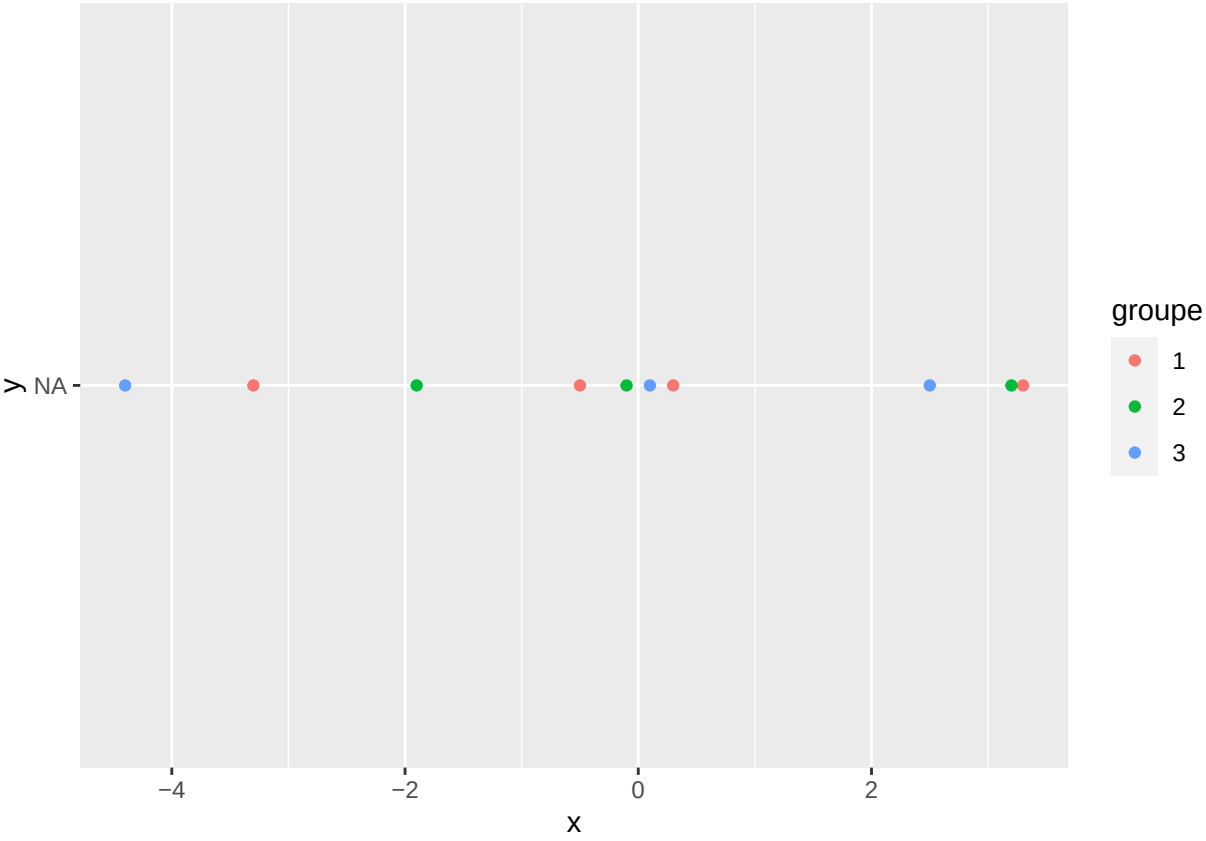
```
donnees<-matrix(c(-3.3,-4.4,-1.9,3.3,2.5,3.2,0.3,0.1,-0.1,-0.5, 1,1,1,2,2,2,2,2,1,1, 1,3,2,1,3,2,1,3,2,
donnees<-as.data.frame(donnees)
names(donnees)<-c("Var","partition1","partition2")
t(donnees)
```

```
##            [,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 $\sigma^{2(0)}$

```
initEM <- function(x, partition) {
  K = length(unique(partition))
  param = list()
  param$pi = table(partition) / length(partition)
  param$theta = list()
  param$theta$mu = rep(0, K)
  param$theta$sigma2 = rep(0, K)

  for (k in 1:K) {
    param$theta$mu[k] = mean(x[partition == k])
    nk = sum(partition == k)
    param$theta$sigma2[k] = ((nk -1) / nk) * var(x[partition == k])
  }

  return(param)
}
```

```r
param_0 = initEM(donnees$Var, donnees$partition1)
param_0
```

```
## $pi
## partition
##   1   2
## 0.5 0.5
##
## $theta
## $theta$mu
## [1] -2.04  1.88
##
## $theta$sigma2
## [1] 2.6624 1.9616
```

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 $cte_i = \log \sum_l \exp\{\log \tau_{il}\}$

```r
Estep <- function(x, param) {
  # Astuce : coder en log-space
  n = length(x)
  K = length(param$theta$mu)
  logtau = matrix(0, n, K)

  for (k in 1:K) {
    logtau[,k] = log(param$pi[k]) +
      dnorm(x=x, mean = param$theta$mu[k],
            sd = sqrt(param$theta$sigma2[k]),
            log = T) # use vectorization of dnorm()
  }

  tau = exp(logtau)
  tau = tau / rowSums(tau) # normalization

  return(tau)
}

tau_0 = Estep(donnees$Var, param_0)
tau_0
```

```
##              [,1]         [,2]
##  [1,] 0.998322097 0.0016779033
##  [2,] 0.999857197 0.0001428028
##  [3,] 0.970275611 0.0297243891
##  [4,] 0.006732798 0.9932672023
##  [5,] 0.019348146 0.9806518539
##  [6,] 0.007651619 0.9923483811
```

```
##  [7,] 0.367086378 0.6329136222
##  [8,] 0.448884498 0.5511155021
##  [9,] 0.534879918 0.4651200823
## [10,] 0.699664342 0.3003356584
```

Etape M

Coder

- une fonction `compute_PI(tau)` qui calcule $\hat{\pi}$.

```
compute_PI = function(tau) {
  n = dim(tau)[1]
  return(colSums(tau) / n)
}

compute_PI(tau_0)
```

```
## [1] 0.5052703 0.4947297
```

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

```
compute_mu = function(tau, x) {
    N = nrow(tau)
    K = ncol(tau)

    mu = rep(0, K)

    for(k in 1:K) {
      norm = 0
      for (i in 1:N) {
        norm = norm + tau[i, k]
        mu[k] = mu[k] + tau[i,k] * x[i]
      }
      mu[k] = mu[k] / norm
    }

    return(mu)
}

mu_1 = compute_mu(tau=tau_0, x=donnees$Var)
```

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

```
compute_sigma2 = function(tau, x, mu) {
    N = nrow(tau)
    K = ncol(tau)

    sigma2 = rep(0, K)

    for(k in 1:K) {
      norm = 0
      for (i in 1:N) {
        norm = norm + tau[i,k]
        sigma2[k] = sigma2[k] + tau[i,k] * (x[i] - mu[k])^2
      }
      sigma2[k] = sigma2[k] / norm
```

```
    }

    return(sigma2)
}

compute_sigma2(tau = tau_0, x=donnees$Var, mu = mu_1)
```

```
## [1] 3.094669 2.304496
```

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

```
Mstep = function(x, tau) {
  # list for storing the result
  param = list()
  param$theta = list()

  # compute pi_hat
  param$pi = compute_PI(tau)

  # compute mu_hat
  param$theta$mu = compute_mu(tau, x)

  # compute sigma² hat
  param$theta$sigma2 = compute_sigma2(tau, x, mu = param$theta$mu)

  return(param)
}

param_1 = Mstep(x=donnees$Var, tau = tau_0)
param_1
```

```
## $theta
## $theta$mu
## [1] -1.917902  1.797060
##
## $theta$sigma2
## [1] 3.094669 2.304496
##
##
## $pi
## [1] 0.5052703 0.4947297
```

Calcul de la vraisemblance marginale

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

```
compute_mixture_llhood = function(x, param) {
  N = length(x)
  K = length(param$theta$mu)

  # ---------- Brute force double loop
  # llhood = 0
  # for(i in 1:N) {
  #   temp = 0
```

```
  #   for(k in 1:K) {
      # temp = temp +
      #   param$pi[k] * dnorm(x=x[i],
      #                       mean = param$theta$mu[k],
      #                       sd = sqrt(param$theta$sigma2[k]),
      #                       log = F)
  #   }
  #   llhood = llhood + log(temp)
  # }


  # ------------ use vectorization of dnorm
  lhoods = matrix(0, N, K)
  for(k in 1:K) {
    lhoods[,k] = param$pi[k] *
      dnorm(x=x, mean = param$theta$mu[k],
            sd = sqrt(param$theta$sigma2[k]),
            log = F)
  }
  lhoods = sum(log(rowSums(lhoods)))

  return(sum(lhoods))
}

cat("Llhood à l\'init : ", compute_mixture_llhood(donnees$Var, param_0) , '\n')
```

```
## Llhood à l'init :  -23.15126
```

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

```
max.iter = 20
rtol = 1e-6
K = 2
partition_init = donnees$partition1
```

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

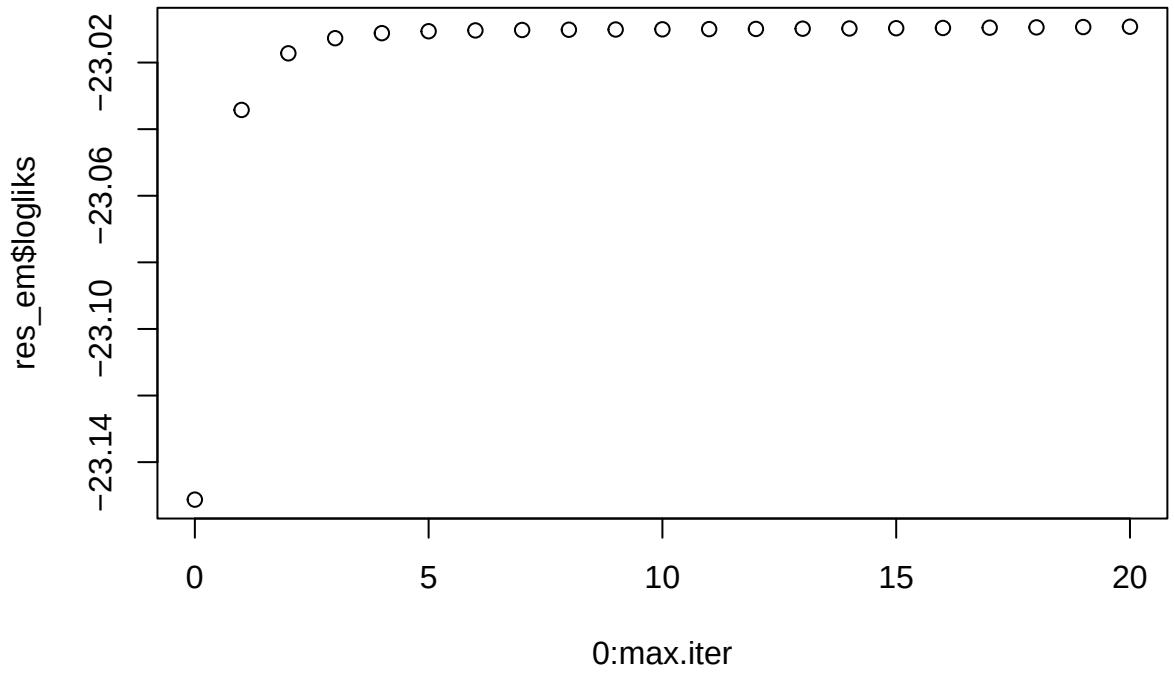
```
EMgauss1D <- function(x, K, partition_init, max.iter, rtol) {

  # sanity check
  if (length(unique(partition_init)) != K) {
    stop('The init partition must have the same number of clusters as K')
  }
```

```r
  # initialization
  param = initEM(x=x, partition=partition_init)
  logliks = rep(NA, max.iter+1) # store de loglikelihoods values
  logliks[1] = compute_mixture_llhood(x, param)
  cat("Llhood à l\'init ", logliks[1], '\n')

  for(ite in 1:max.iter) {
    old = compute_mixture_llhood(x, param)

    # E-step (use current param)
    tau = Estep(x=x, param=param)

    # M-step (update param)
    param = Mstep(x = x, tau = tau)

    # test convergence
    new = compute_mixture_llhood(x = x, param = param)
    logliks[ite+1] = new
    criterion = abs((new - old)/old)
    if(criterion < rtol) break;

    #sanity check that the likelihoods do not decrease
    stopifnot(new >= old)
    # affichage à chaque itération
    cat("Llhood à l\'étape ", ite,  " : ", new, '\n')
  }

  # compute tau one last time with the value of the final parameters
  tau = Estep(x, param)

  return(list(logliks=logliks, tau=tau, param=param))
}

res_em = EMgauss1D(x=donnees$Var, K=K, partition_init = partition_init,
                   max.iter = max.iter , rtol = rtol)
```

```
## Llhood à l'init  -23.15126
## Llhood à l'étape  1  :  -23.03423
## Llhood à l'étape  2  :  -23.01722
## Llhood à l'étape  3  :  -23.01268
## Llhood à l'étape  4  :  -23.01117
## Llhood à l'étape  5  :  -23.0106
## Llhood à l'étape  6  :  -23.01035
## Llhood à l'étape  7  :  -23.01022
## Llhood à l'étape  8  :  -23.01014
## Llhood à l'étape  9  :  -23.01008
## Llhood à l'étape  10  :  -23.01002
## Llhood à l'étape  11  :  -23.00996
## Llhood à l'étape  12  :  -23.00989
## Llhood à l'étape  13  :  -23.00983
## Llhood à l'étape  14  :  -23.00976
## Llhood à l'étape  15  :  -23.00969
## Llhood à l'étape  16  :  -23.00961
```

```
## Llhood à l'étape  17  :  -23.00952
## Llhood à l'étape  18  :  -23.00943
## Llhood à l'étape  19  :  -23.00934
## Llhood à l'étape  20  :  -23.00924
```

```
plot(0:max.iter, res_em$logliks)
```



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.

```
for(k in 1:K) {
  cat('----- Cluster ', k, ' ------ \n')
  cat('Pi chapeau : \n')
  print(res_em$param$pi[k])
  cat('mu chapeau \n')
  print(res_em$param$theta$mu[k])
  cat('Sigma chapeau \n')
  print(res_em$param$theta$sigma2[k])
}
```

```
## ----- Cluster  1  ------
## Pi chapeau :
## [1] 0.5216861
## mu chapeau
## [1] -1.757172
## Sigma chapeau
## [1] 3.63419
## ----- Cluster  2  ------
## Pi chapeau :
## [1] 0.4783139
## mu chapeau
## [1] 1.749253
```

```
## Sigma chapeau
## [1] 2.487324
```

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 $z_i, \forall i$:

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

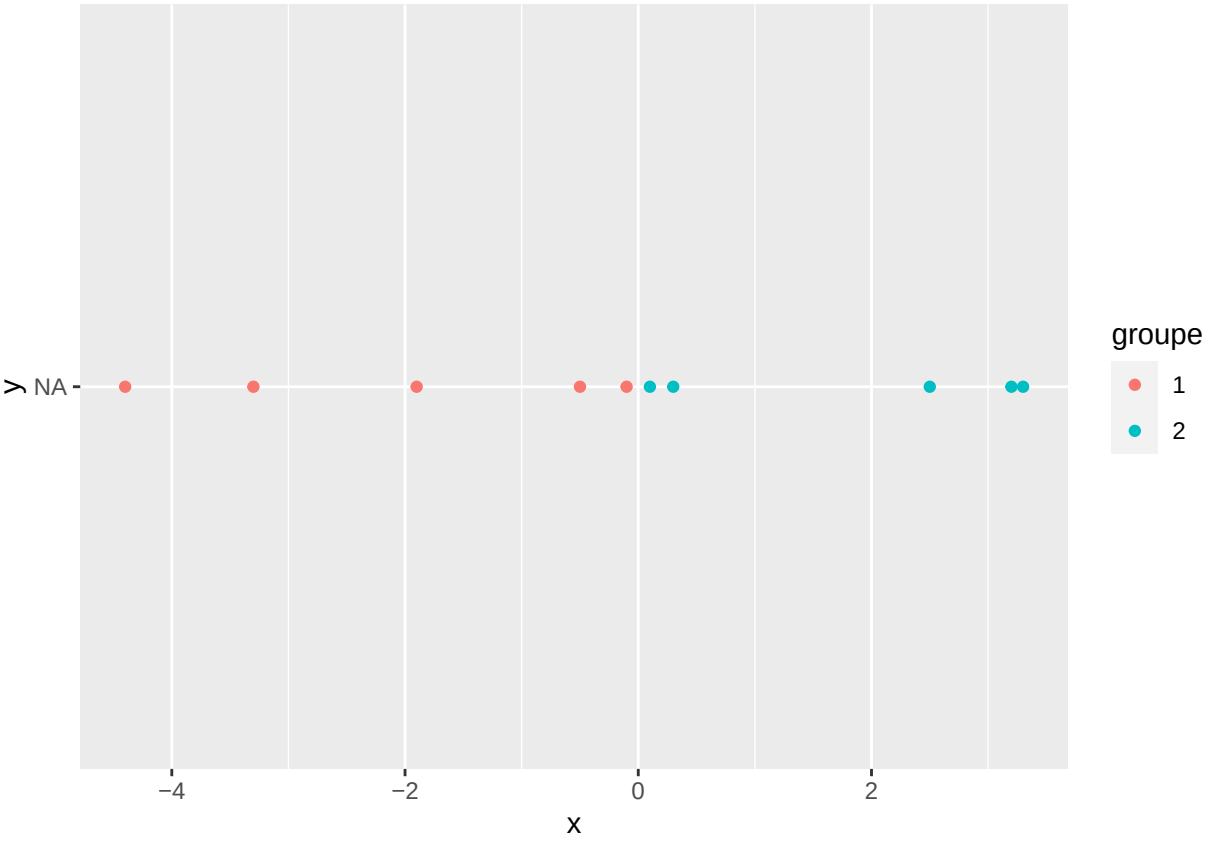
En fait cela revient à faire un argmax par ligne sur la matrix `res_em$tau`.

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

```
clustering = apply(res_em$tau, MARGIN = 1, which.max)
cat('Clustering final : ', clustering , '\n')
```

```
## Clustering final :  1 1 1 2 2 2 2 2 1 1
```

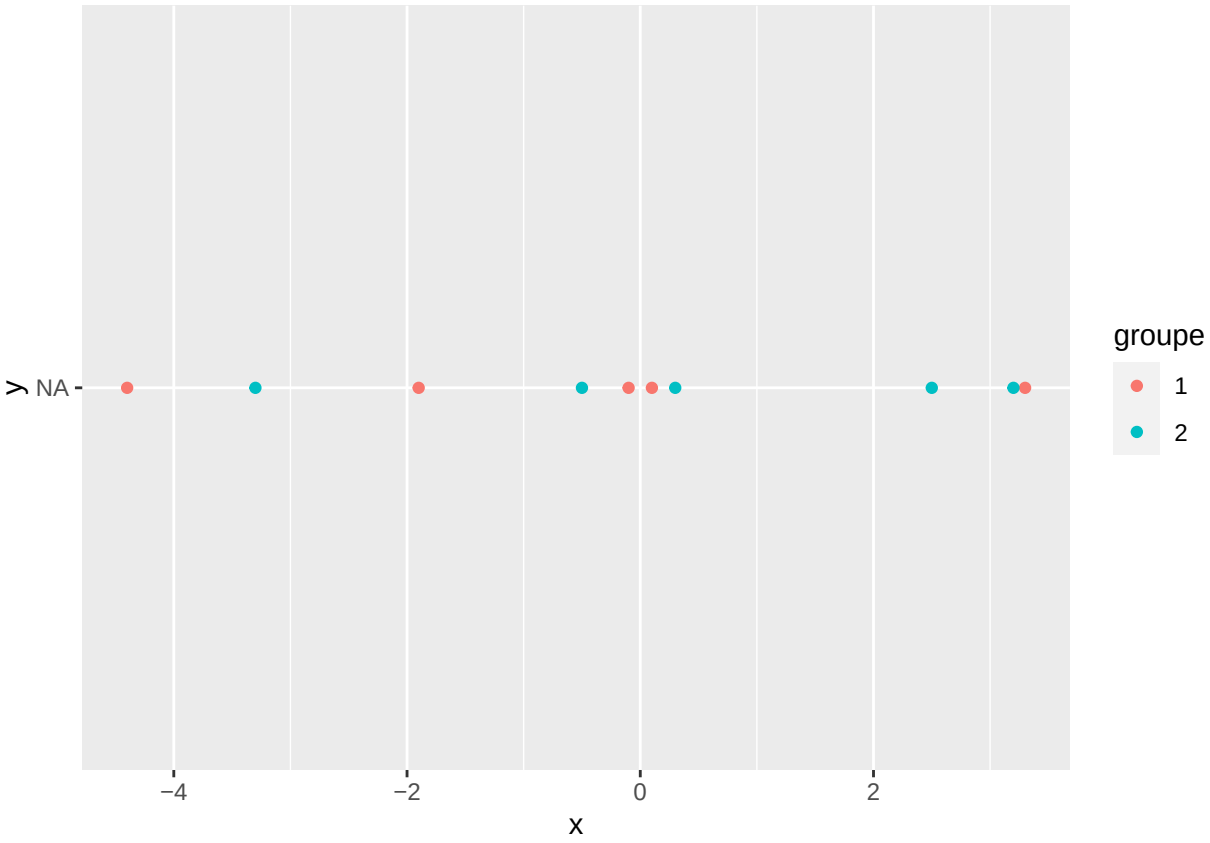
```
plot_data(x = donnees$Var, partition = clustering)
```



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)
```



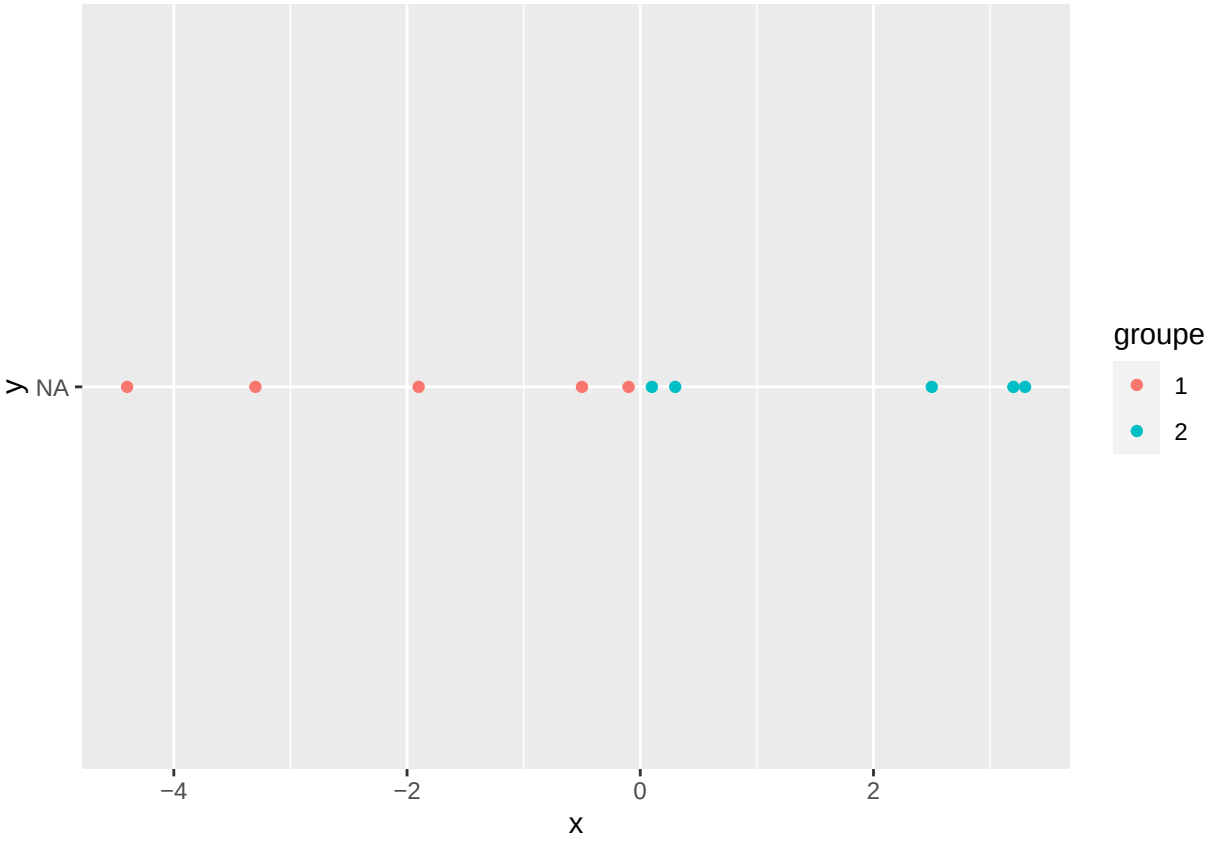

```
res_em_badinit = EMgauss1D(donnees$Var, K, partition_init, max.iter, rtol)
```

```
## Llhood à l'init  -23.26437
## Llhood à l'étape  1  :  -23.2629
## Llhood à l'étape  2  :  -23.26137
## Llhood à l'étape  3  :  -23.25963
## Llhood à l'étape  4  :  -23.25757
## Llhood à l'étape  5  :  -23.2551
## Llhood à l'étape  6  :  -23.25214
## Llhood à l'étape  7  :  -23.24854
## Llhood à l'étape  8  :  -23.24413
## Llhood à l'étape  9  :  -23.23868
## Llhood à l'étape  10  :  -23.2319
## Llhood à l'étape  11  :  -23.22345
## Llhood à l'étape  12  :  -23.21289
## Llhood à l'étape  13  :  -23.1998
## Llhood à l'étape  14  :  -23.18379
## Llhood à l'étape  15  :  -23.16473
## Llhood à l'étape  16  :  -23.14295
## Llhood à l'étape  17  :  -23.1194
## Llhood à l'étape  18  :  -23.09561
## Llhood à l'étape  19  :  -23.07332
## Llhood à l'étape  20  :  -23.054
```

```r
clustering = apply(res_em$tau, MARGIN = 1, which.max)
cat('Clustering final : ', clustering , '\n')
```

```
## Clustering final :  1 1 1 2 2 2 2 2 1 1
```

```r
plot_data(x = donnees$Var, partition = clustering)
```



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.