

TD2 - The EM algorithm for Gaussian mixtures

Nicolas Jouvin

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

Maths part

Recall the Gaussian mixture model where $X = \{x_i\}_{i=1}^n \subset \mathbb{R}$ are **i.i.d** with p.d.f.

$$p_\theta(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x \mid \mu_k, \sigma_k^2) \quad (1)$$

The model parameters are $\theta = \{\pi_k, \mu_k, \sigma_k^2\}$, with $\sum_l \pi_l = 1$.

This model can be interpreted as a **latent variable model** with unobserved latent variables $Z = \{z_i\}_{i=1}^n$, where $z_i \in \{0, 1\}^K$ is a binary vector encoding for the cluster assignment of x_i .

The complete data are $\{x_i, z_i\}_{i=1}^n$.

i Questions (on paper)

1. Write the complete-data log-likelihood of the model

$$\log p_\theta(X, Z)$$

2. Derive the maximum **complete** likelihood estimators π_k^C, μ_k^C and $\sigma_k^{2,C}$ solution of

$$\arg \max_{\theta} \log p_\theta(X, Z)$$

3. **Bonus** redo the same calculations in dimension d . *Hint:* For A a positive matrix, $\nabla_A \log \det(A) = A^{-1}$.

4. Let $Z = \{z_i\}_i \sim q$, with q any distribution on $\{0, 1\}^{K^n}$. Denote $q_{ik} = q(z_{ik} = 1)$. Use the preceding questions to give the solution of

$$\arg \max_{\theta} \mathbb{E}_{Z \sim q} [\log p_{\theta}(X, Z)]$$

in terms of q_{ik} .

Programmation part : implementing your own EM in 1-d

About this exercise

The goal of this practical exercise is to implement your own EM algorithm. You will be guided step-by-step throughout the question.

Hint: I strongly advise against using ChatGPT or any equivalent LLM to “help” you. One needs to make mistake to learn the caveats of coding such algorithms !

We will apply it on the following synthetic dataset 1-dimension.

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

	[,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

Introductory question

How do you interpret the two following graphic: does one partition seems better than the other ?

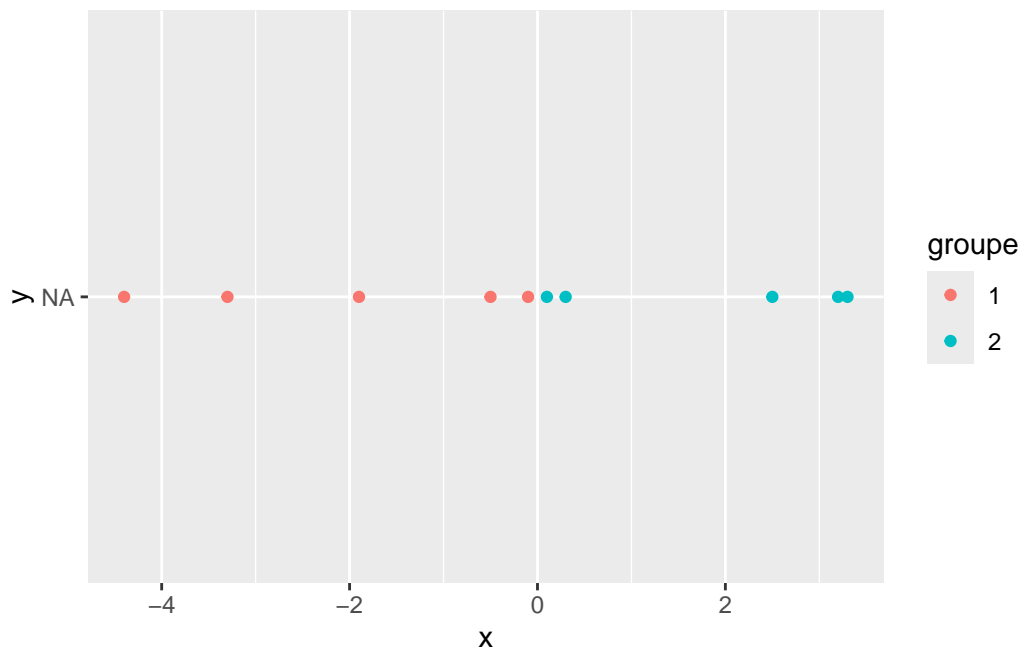
```

library(ggplot2)

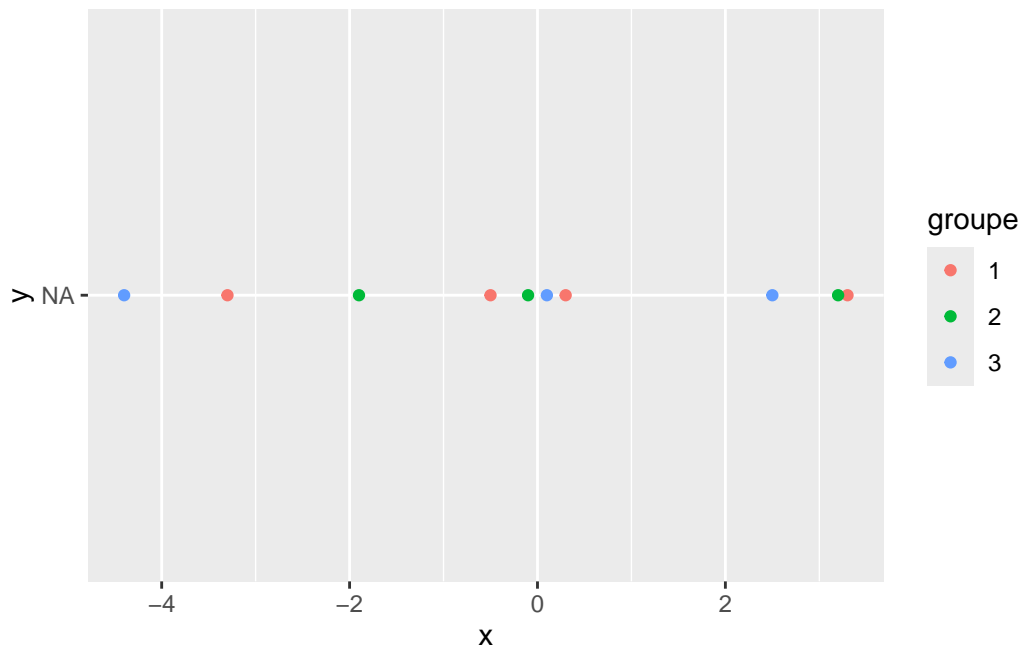
plot_data <- function(x, partition) {
  # function that plot the 1D data vector x with color
  # according to an argument partition
  #
  # return: a ggplot graph
  df = data.frame(x = x, groupe = factor(partition))
  gg = ggplot(df) + geom_point(aes(x=x, y = NA, color=groupe))
  return(gg)
}

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

```



```
print(gg_part2)
```



The second initialization do not seem really clever...

i 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)
}

param_0 = initEM(data$Var, data$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

```

i Question 2: Etape E

Coder une fonction `Estep(x, param)` qui calcule et renvoie les $\tau_{ik} \propto \pi_k \mathcal{N}(x_i | \mu_k, \sigma_k^2)$. **Astuce** calculer en log-space pour mieux représenter les petites quantités. Le problème vient que, numériquement, $\log e^{-1000} \approx \log 0 = -\infty$ tandis que mathématiquement $\log(e^{-1000}) = -1000$. Evidemment le -1000 est arbitraire ici, et ce qu'on appelle l'*underflow* dépend de votre précision de nombre flottant (nombre de bits: 32, 64, 128 ?). Pour contourner ce problème, remarquons que

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

La constante de normalisation peut-être calculée comme suit

1. *Méthode naïve (instable)*: $cte_i = \log \sum_l \exp\{\log \tau_{il}\}$
2. *LogSumExp trick*: $cte_i = m_i + \log \sum_l \exp\{\log \tau_{il} - m_i\}$ avec $m_i := \max_l \log \tau_{il}$. This ensures an $\exp(0)$ somewhere in the sum, hence avoiding numerical underflows.

```

logsumexp <- function(logx) {
  # compute \log(\sum exp(logx)) by rescaling it by m = \max(logx)
  # indeed : \log(\sum exp(logx)) = m + \log(\sum exp(logx - m))
  # This ensures an exp(0) somewhere in the sum => avoid
  # numerical underflows.
  m = max(logx)
  return(m + log(sum(exp(logx - m))))
}

```

```

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()
  }

  # normalize in log-space with LogSumExp trick
  logtau = logtau - apply(logtau, 1, logsumexp)

  # Compute tau
  tau = exp(logtau)

  # sanity check that tau_i are normalized (sum to 1)
  stopifnot(all.equal(rowSums(tau), rep(1, n)))
  return(tau)
}

```

```

tau_0 = Estep(data$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
```

i Question 3: étape M

En utilisant les formules données dans les slides, 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 assembler dans une fonction `Mstep(x, tau)` qui fait la M-step de l'algorithme EM.

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

```
compute_PI(tau_0)
```

```
[1] 0.5052703 0.4947297
```

```
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=data$Var)

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=data$Var, mu = mu_1)

```

[1] 3.094669 2.304496

```

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 sigma^2 hat
  param$theta$sigma2 = compute_sigma2(tau, x, mu = param$theta$mu)

```



```

    return(param)
  }

param_1 = Mstep(x=data$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

```

i Question 4: calcul de la vraisemblance marginale

Coder une fonction `compute_mixture_llhood = fonction(x, param)` qui retourne la log-vraisemblance marginale des observations :

$$\log p(x_1, \dots, x_n | \theta, \pi) = \sum_i \log \sum_k \pi_k \mathcal{N}(x_i | \mu_k, \sigma_k^2)$$

Astuce: utiliser la vectorisation de `dnorm()` pour éviter une boucle sur les observations.

```

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

  # /\ This method should not be used (too slow)
  # ----- 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(data$Var, param_0) , '\n')

```

Llhood à l'init : -23.15126

i Question 4 (BONUS): vérifier l'égalité de l'ELBO

D'après le cours, on sait que l'ELBO $\mathcal{L}(q, \theta) = \mathbb{E}_q[\log p_\theta(X, Z)] + \mathcal{H}(q)$ est égale à la vraisemblance marginale ssi

$$q = p_\theta(Z | X).$$

Le vérifier numériquement sur cet exemple.

i Question 5: algorithme 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` :

$$\left| \frac{\log p_{\theta^{(t+1)}}(X) - \log p_{\theta^{(t)}}(X)}{\log p_{\theta^{(t)}}(X)} \right| < rtol.$$

La fonction devra retourner une liste avec les 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 les hyper-paramètres suivants:

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

```
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')
  }

  # 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=data$Var, K=K, partition_init = partition_init,
                  max.iter = max.iter , rtol = rtol)

```

```

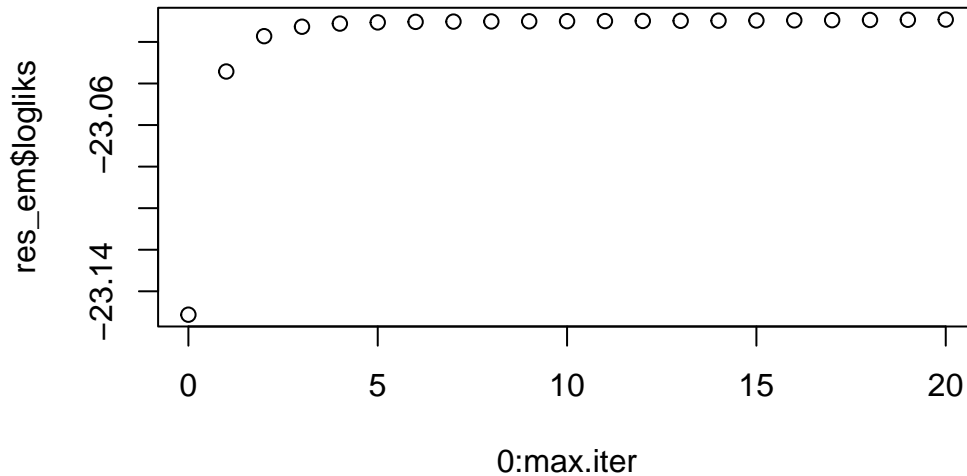
Llhood à l'init -23.15126
Llhood à l'étape 1 : -23.03423
Llhood à l'étape 2 : -23.01722
Llhood à l'étape 3 : -23.01268
Llhood à l'étape 4 : -23.01117
Llhood à l'étape 5 : -23.0106
Llhood à l'étape 6 : -23.01035
Llhood à l'étape 7 : -23.01022
Llhood à l'étape 8 : -23.01014
Llhood à l'étape 9 : -23.01008
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 $(\hat{\pi}_k, \hat{\mu}_k, \hat{\Sigma}_k)_k$

i Question 7: affichage et diagnostic

1. Afficher l'évolution de la log-vraisemblance en fonctions des itérations de l'algorithme.
2. Afficher les paramètres estimés dans chacun 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 $z_i, \forall i$:

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

Dans notre code, cela revient à faire un argmax par ligne sur la matrix `res_em$tau`.

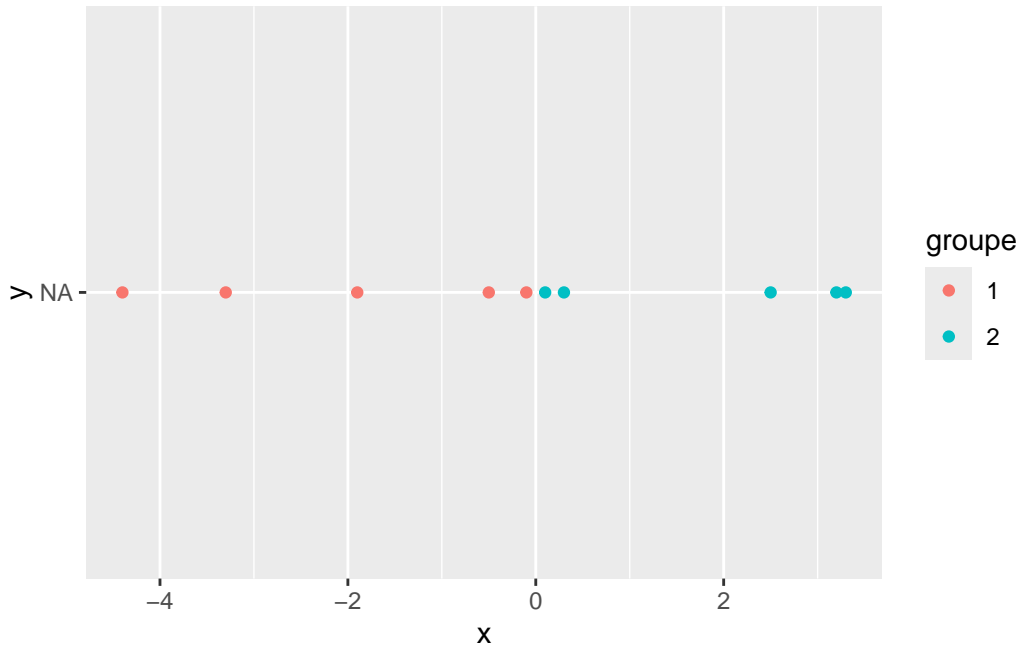
i Question 8: visualisation

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
```

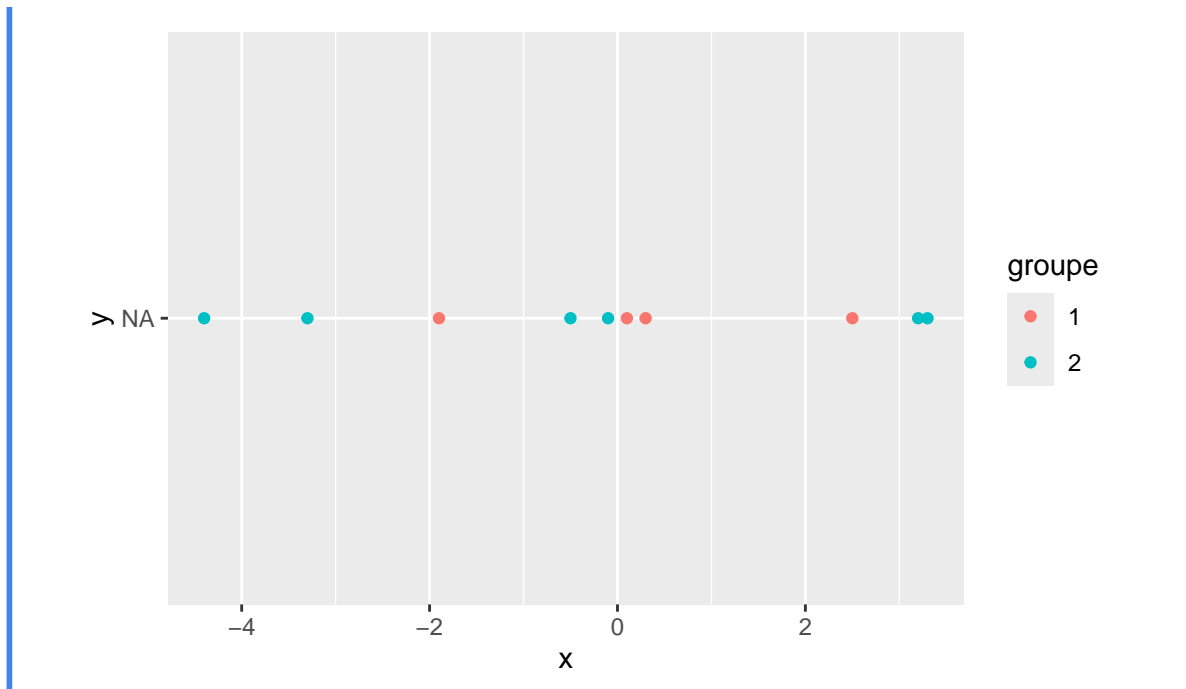
```
plot_data(x = data$Var, partition = clustering)
```



i Question 9: Impact de l'initialisation

Refaites 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(data$Var, partition_init)
```



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

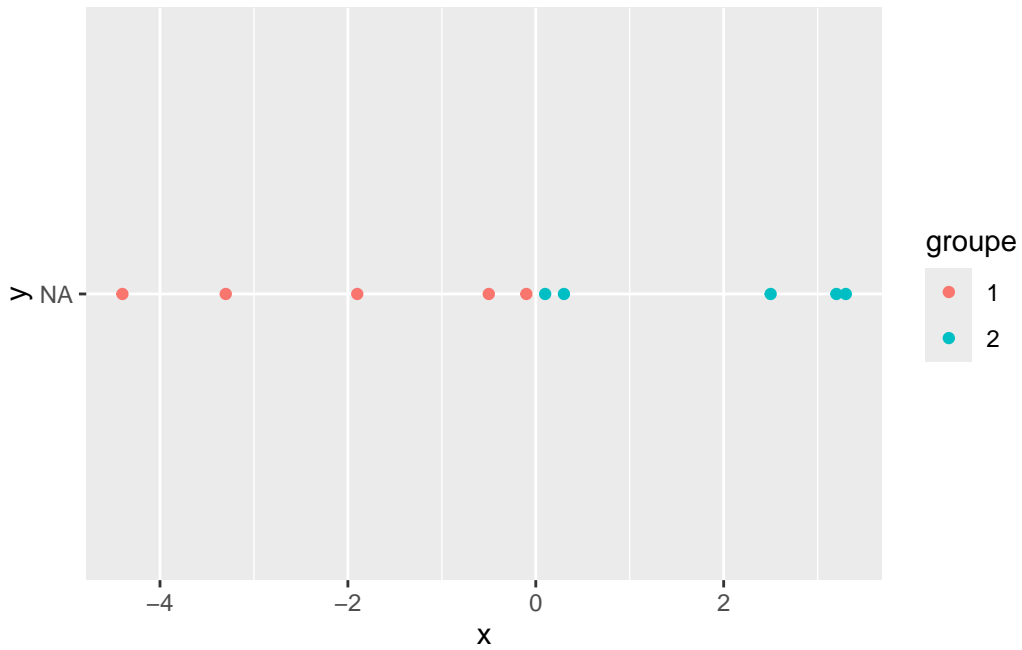
```
Llhood à l'init -23.62138
Llhood à l'étape 1 : -23.40042
Llhood à l'étape 2 : -23.29649
Llhood à l'étape 3 : -23.25763
Llhood à l'étape 4 : -23.24178
Llhood à l'étape 5 : -23.23096
Llhood à l'étape 6 : -23.21947
Llhood à l'étape 7 : -23.20547
Llhood à l'étape 8 : -23.18835
Llhood à l'étape 9 : -23.16816
Llhood à l'étape 10 : -23.14561
Llhood à l'étape 11 : -23.12196
Llhood à l'étape 12 : -23.09857
Llhood à l'étape 13 : -23.07649
Llhood à l'étape 14 : -23.05625
Llhood à l'étape 15 : -23.03806
Llhood à l'étape 16 : -23.02184
Llhood à l'étape 17 : -23.00732
Llhood à l'étape 18 : -22.99404
Llhood à l'étape 19 : -22.98127
```


Llhood à l'étape 20 : -22.9679

```
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 = data$Var, partition = clustering)
```



i 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.