TP2

Exercice n°1 : Résistance au stress thermique chez Caenorhabditis elegans

On s'intéresse aux mécanismes cellulaires et moléculaires sous-jacents à l'élimination sélective des composants cellulaires par autophagie dans le cadre des processus de développement, du vieillissement et des réponses adaptatives à divers stress. Caenorhabditis elegans (*C. elegans* en abrégé) est un petit ver d'un millimètre environ, transparent et non parasitaire. C. elegans est un bon modèle pour analyser les processus autophagiques. Parce que la voie autophagique est hautement conservée chez les eucaryotes, les données obtenues chez la levure et le ver sont essentielles pour comprendre le rôle de l'autophagie chez l'homme.

Afin de comprendre l'impact de la température sur la croissance des organismes C. elegans, un laboratoire met au point une expérience. Ils mesurent la taille des vers soumis à température normale (20 degrés) et également la taille des vers soumis à température de 37 degrés. Les mesures sont effectuées par 4 techniciens différents (indiqués par les lettres A, B, C et D dans le jeu de données), le même jour et dans des conditions expérimentales identiques. Les longueurs de vers mesurées sont stockées dans la variable `length`.

Les données sont contenues dans le fichier `c.elegans.txt`.

(a) Charger les données contenues dans le fichier `c.elegeans.txt`. Décrire ces données.

(b) Proposer une modélisation appropriée de ces données.

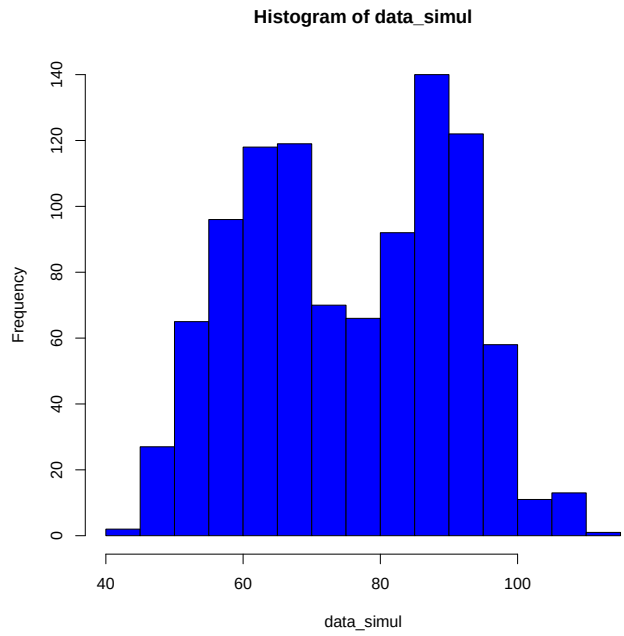
Exercice n°2 : Simulations de données sous un modèle de mélange et inférence

(a) Simuler 1000 observations issues d'un mélange de deux lois gaussiennes en proportion égale et de paramètres $\mu_1 = 88$, $\sigma_1^2 = 7.5^2$ pour la première composante et $\mu_2 = 62$ et $\sigma_1^2 = 7.5^2$ pour la deuxième composante. Stocker ces données dans un objet `data_simul.RData`.

Solution:

Corrigé en classe. Voici une version plus concise.

```
set.seed(123)
n=1000
mesclasses<- rbinom(n,1,0.5)
mu_vect <- c(88,62)
sigma_vect <- c(7.5,7.5)
mu_simul <- mu_vect[mesclasses+1]
sigma_simul <- sigma_vect[mesclasses+1]
data_simul <- rnorm(n,mean = mu_simul,sd = sigma_simul)
hist(data_simul,col="blue")
```




```r
save(data_simul,file="data_simul.RData")
```

(b) En utilisant le code R ci-dessous, comprendre le fonctionnement de l'algorithme EM pour l'estimation des paramètres du modèle de mélange.

```r
## init
load("data_simul.RData")
nbit <- 12
stock_value <- matrix(NA,ncol=7,nrow=nbit)
colnames(stock_value) <- c("pi1","pi2","m1","m2",
                           "sigma1","sigma2","loglik")
# initialisation aléatoire des paramètres du modèle
pi1 <- runif(1)
pi2 <- 1-pi1
mu1 <- runif(1,0,100)
mu2 <- runif(1,0,100)
sigma1 <- runif(1,0.5,10)
sigma2 <- runif(1,0.5,10)
get_loglike <- function(y){
  pi1*dnorm(y,mean=mu1,sd=sigma1)+pi2*dnorm(y,mean=mu2,sd=sigma2)
}
loglik <- sum(log(sapply(data_simul,get_loglike)))
loglik
stock_value[1,] <- c(pi1,pi2,mu1,mu2,sigma1,sigma2,loglik)
```

```r
for(i in 2:nbit){
  # Etape E
  get_tij <- function(y){
    numt11 <- (pi1*dnorm(y,mean=mu1,sd=sigma1))
    numt12 <- (pi2*dnorm(y,mean=mu2,sd=sigma2))
    denum <- numt11+numt12
    t11 <-  numt11/denum
    t12 <-  numt12/denum
    return(c(t11,t12))
```

```
  }
  tij_etapec <- t(sapply(data_simul, get_tij))

  ## Etape M
  pi1 <- mean(tij_etapec[,1])
  pi2 <- mean(tij_etapec[,2])
  mu1 <- sum(data_simul*tij_etapec[,1])/sum(tij_etapec[,1])
  mu2 <- sum(data_simul*tij_etapec[,2])/sum(tij_etapec[,2])
  sigma1 <- sqrt(sum((data_simul-mu1)^2*tij_etapec[,1])/sum(tij_etapec[,1]))
  sigma2 <- sqrt(sum((data_simul-mu2)^2*tij_etapec[,2])/sum(tij_etapec[,2]))
  get_loglike <- function(y){
    pi1*dnorm(y,mean=mu1,sd=sigma1)+pi2*dnorm(y,mean=mu2,sd=sigma2)
  }
  loglik <- sum(log(sapply(data_simul,get_loglike)))
  stock_value[i,] <- c(pi1,pi2,mu1,mu2,sigma1,sigma2,loglik)
}

head(stock_value)
```

(c) A l'aide des paramètres inférées, retrouver les affectations des observations simulées à la question précédente.

(d) Changer les paramètres d'initialisation de l'algo EM. Remarquer que la perf de l'algo est sensible aux valeurs initiales.

Solution:
On utilise le code suivant pour retrouver les classes d'affectation des observations : `apply(tij_etapec,1,`

```r
load("data_simul.RData")
nbit <- 12
stock_value <- matrix(NA,ncol=7,nrow=nbit)
colnames(stock_value) <- c("pi1","pi2","m1","m2",
                           "sigma1","sigma2","loglik")
# initialisation aléatoire des paramètres du modèle
pi1 <- runif(1)
pi2 <- 1-pi1
mu1 <- runif(1,0,100)
mu2 <- runif(1,0,100)
sigma1 <- runif(1,0.5,10)
sigma2 <- runif(1,0.5,10)
get_loglike <- function(y){
  pi1*dnorm(y,mean=mu1,sd=sigma1)+pi2*dnorm(y,mean=mu2,sd=sigma2)
}
loglik <- sum(log(sapply(data_simul,get_loglike)))
loglik
stock_value[1,] <- c(pi1,pi2,mu1,mu2,sigma1,sigma2,loglik)
for(i in 2:nbit){
  # Etape E
  get_tij <- function(y){
    numt11 <- (pi1*dnorm(y,mean=mu1,sd=sigma1))
    numt12 <- (pi2*dnorm(y,mean=mu2,sd=sigma2))
    denum <- numt11+numt12
    t11 <-  numt11/denum
    t12 <-  numt12/denum
    return(c(t11,t12))
  }
  tij_etapec <- t(sapply(data_simul, get_tij))
  ## affichage des classes "inférées" à chaque étape
  ## et comparaison avec les "vraies" classes simulées
  print(table(apply(tij_etapec,1,which.max),mesclasses))
  ## Etape M
  pi1 <- mean(tij_etapec[,1])
  pi2 <- mean(tij_etapec[,2])
  mu1 <- sum(data_simul*tij_etapec[,1])/sum(tij_etapec[,1])
  mu2 <- sum(data_simul*tij_etapec[,2])/sum(tij_etapec[,2])
  sigma1 <- sqrt(sum((data_simul-mu1)^2*tij_etapec[,1])/sum(tij_etapec[,1]))
  sigma2 <- sqrt(sum((data_simul-mu2)^2*tij_etapec[,2])/sum(tij_etapec[,2]))
  get_loglike <- function(y){
    pi1*dnorm(y,mean=mu1,sd=sigma1)+pi2*dnorm(y,mean=mu2,sd=sigma2)
  }
  loglik <- sum(log(sapply(data_simul,get_loglike)))
  stock_value[i,] <- c(pi1,pi2,mu1,mu2,sigma1,sigma2,loglik)
}

head(stock_value)
```

Exercice n°3 : Simulations de données sous un modèle de mélange

(a) Charger les données contenues dans le fichier dataMM.RData et effectuer un clustering des 600 observations à l'aide d'un modèle de mélange de lois gaussiennes en 2D à 2 classes, 3 classes et 4 classes à l'aide du package `mclust`.

(b) Choisir la forme de la matrice de variance covariance appropriée pour chaque composante du mélange (voir table 3 et figure 2 `https://journal.r-project.org/archive/2016/RJ-2016-021/RJ-2016-021.pdf`

Solution:

— Il faut d'abord choisir la forme des matrices de variance covariance possibles :

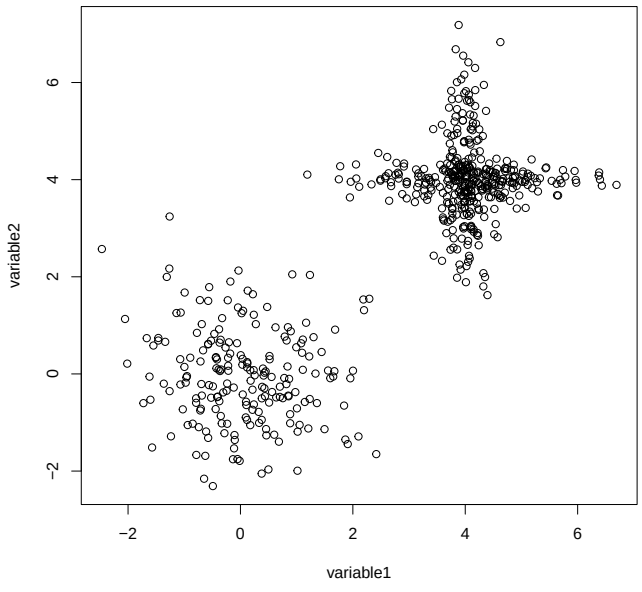
Modèle	$\boldsymbol{\Sigma}_k$	Distribution	Volume	Forme	Orientation
EII	$\lambda \boldsymbol{I}$	Spherical	Equal	Equal	—
VII	$\lambda_k \boldsymbol{I}$	Spherical	Variable	Equal	—
EEI	$\lambda \boldsymbol{A}$	Diagonal	Equal	Equal	Coordinate axes
VEI	$\lambda_k \boldsymbol{A}$	Diagonal	Variable	Equal	Coordinate axes
EVI	$\lambda \boldsymbol{A}_k$	Diagonal	Equal	Variable	Coordinate axes
VVI	$\lambda_k \boldsymbol{A}_k$	Diagonal	Variable	Variable	Coordinate axes
EEE	$\lambda \boldsymbol{D}\boldsymbol{A}\boldsymbol{D}^\top$	Ellipsoidal	Equal	Equal	Equal
EVE	$\lambda \boldsymbol{D}\boldsymbol{A}_k\boldsymbol{D}^\top$	Ellipsoidal	Equal	Variable	Equal
VEE	$\lambda_k \boldsymbol{D}\boldsymbol{A}\boldsymbol{D}^\top$	Ellipsoidal	Variable	Equal	Equal
VVE	$\lambda_k \boldsymbol{D}\boldsymbol{A}_k\boldsymbol{D}^\top$	Ellipsoidal	Variable	Variable	Equal
EEV	$\lambda \boldsymbol{D}_k\boldsymbol{A}\boldsymbol{D}_k^\top$	Ellipsoidal	Equal	Equal	Variable
VEV	$\lambda_k \boldsymbol{D}_k\boldsymbol{A}\boldsymbol{D}_k^\top$	Ellipsoidal	Variable	Equal	Variable
EVV	$\lambda \boldsymbol{D}_k\boldsymbol{A}_k\boldsymbol{D}_k^\top$	Ellipsoidal	Equal	Variable	Variable
VVV	$\lambda_k \boldsymbol{D}_k\boldsymbol{A}_k\boldsymbol{D}_k^\top$	Ellipsoidal	Variable	Variable	Variable

(c) Choisir le nombre de classes optimal à l'aide du critère BIC.

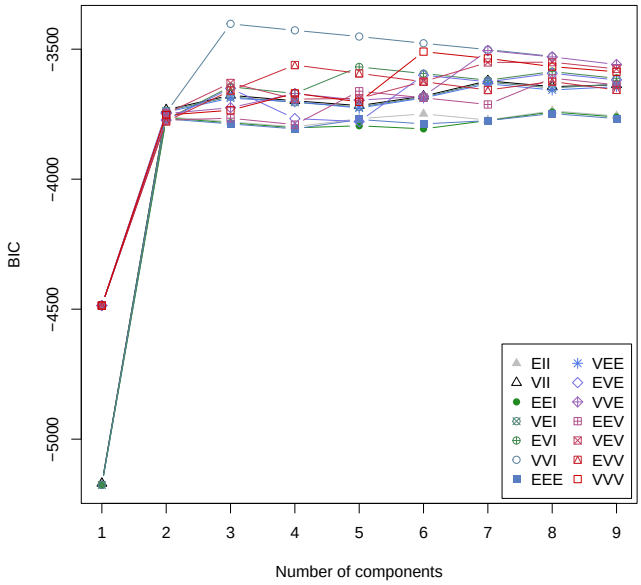
(d) Choisir le nombre de classes optimal à l'aide du critère ICL.

Solution:

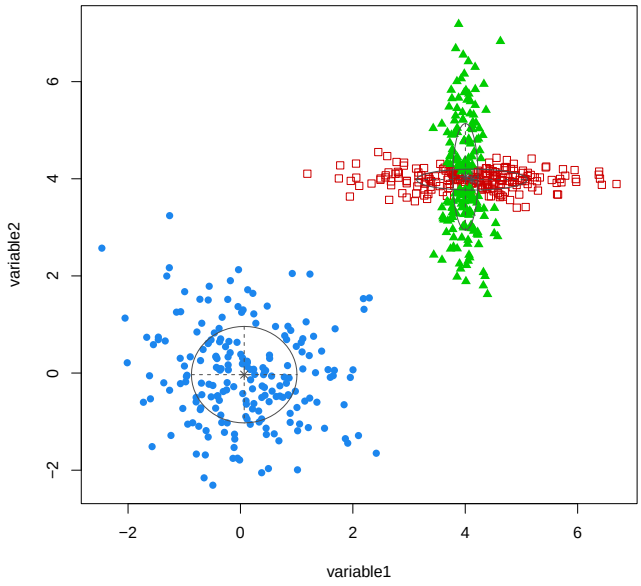
```
load("data/dataMM.RData")
plot(X)
```



```
library(mclust)
## Package 'mclust' version 6.0.1
## Type 'citation("mclust")' for citing this R package in publications.
resBIC <- mclustBIC(X)
plot(resBIC)
```

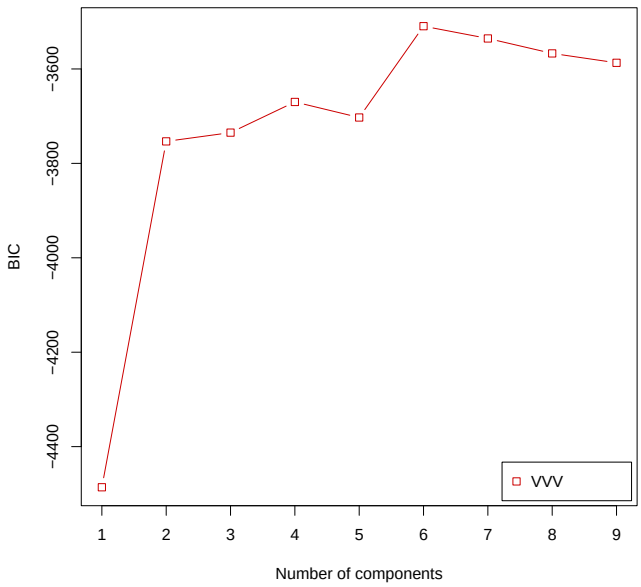



```
summary(resBIC)
## Best BIC values:
##              VVI,3       VVI,4       VVI,5
## BIC      -3402.653 -3427.47350 -3451.35522
## BIC diff     0.000   -24.82022   -48.70194
mod <- Mclust(X,x=resBIC)
plot(mod, what = "classification")
```

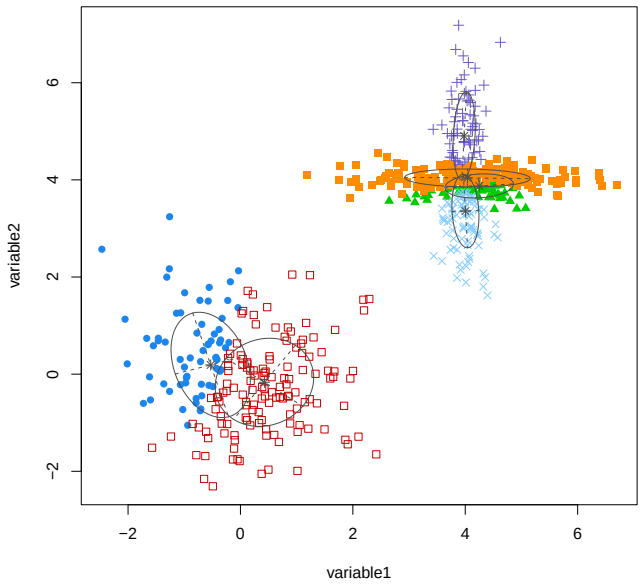


```r
## Le critère BIC sélectionne bien 3 clusters

resBIC <- mclustBIC(X,modelNames = c("VVV"))
plot(resBIC)
```

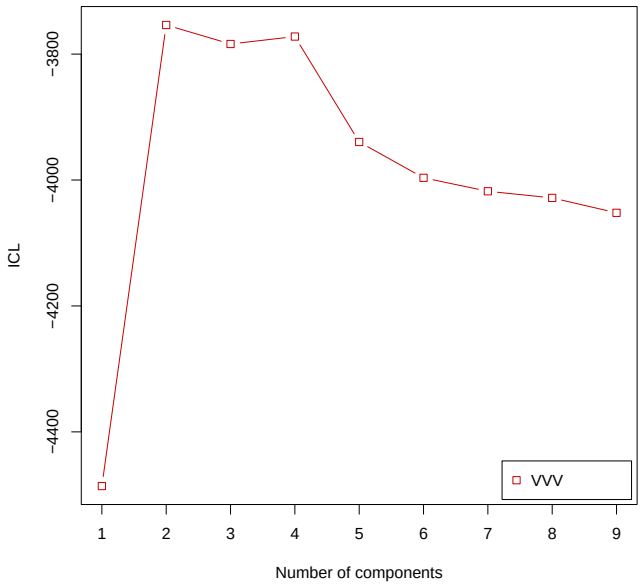



```r
summary(resBIC)
## Best BIC values:
##              VVV,6       VVV,7       VVV,8
## BIC      -3509.394 -3535.41490 -3567.05337
## BIC diff     0.000   -26.02109   -57.65957
mod <- Mclust(X,x=resBIC)
plot(mod, what = "classification")
```



```
## Si l'on impose des matrices "pleines",
## on surestime le nombre de clusters


resICL <- mclustICL(X,modelNames = c("VVV"))
plot(resICL)
```




```
summary(resICL)
## Best ICL values:
##             VVV,2       VVV,4       VVV,3
## ICL      -3753.79 -3772.14496 -3783.95557
## ICL diff     0.00   -18.35536   -30.16597
## le critère ICL choisit 2 clusters
```

Exercice n°4 : Simulations de modèle de mélange de lois de Poisson

(a) Simuler un modèle de mélange de trois lois de Poisson sous les paramètres de proportions suivant : $(p_1 = 0.4; p_2 = 0.1)$ et de paramètres de moyenne $(\lambda_1 = 0.5, \lambda_2 = 6, \lambda_3 = 15)$.

(b) Implémenter l'algorithme EM associé et retrouver les paramètres de simulations à partir d'un échantillon observé.

Exercice n°5 : Analyse de données réelles

(a) Récupérer les données "Levine_ 13dim.txt" à partir de cette page gitub `https://github.com/lmweber/benchmark-data-Levine-13-dim/blob/master/data/Levine_13dim.txt`.

(b) A l'aide de la description du jeu de données fournie sur la page github `https://github.com/lmweber/benchmark-data-Levine-13-dim`, comprendre le jeu de données. Visualiser graphiquement ces données.

(c) Effectuer une classification de ces données à l'aide d'un modèle de mélange. Justifier le choix de modélisation.