

SDS DM Programmation en R

Imad Bendimerad

11/16/2022

Installation et chargement des Packages

```
#install.packages("readxl")  
library(readxl)
```

Partie 1:

Jeu de données

Afin d'importer le sheet excel, on utilise `read_excel` en choisissant le calque "Body paramters" et on déclare le codage des NA qui est dans ce cas la chaîne de caractères "NA":

```
rm(list=ls())  
#setwd("~/Library/Mobile Documents/com~apple~CloudDocs/M2SDS")  
PhotoDRYAD <- read_excel("PhotoDRYAD.xlsx", sheet = "Body paramters", na = "NA")  
head(PhotoDRYAD)
```

```
## # A tibble: 6 x 10  
##   Sex ID      Age CaptivityWild Height Chest G~1 Neck ~2 Length Weight Foot ~3  
##   <dbl> <chr> <dbl> <chr>          <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1     1 AA      739 a              207      332     230    NA    2249    NA  
## 2     1 AA      749 a              207      341     232    NA    2136   110  
## 3     1 AA      741 a              208      332     232    NA    2120   116  
## 4     1 AA      744 a              208      340     230    NA    2138   110  
## 5     1 AA      747 a              208      342     235    NA    2290   111  
## 6     1 AA      735 a              210      323     227    203    2185   114  
## # ... with abbreviated variable names 1: 'Chest Girth',  
## # 2: 'Neck Circumference', 3: 'Foot size'
```

```
any(is.na(PhotoDRYAD$ID))
```

```
## [1] TRUE
```

```
which(is.na(PhotoDRYAD$ID))
```

```
## [1] 2237
```

Avant d'utiliser ce jeu de données, on remarque que les variables de la colonne ID ont une combinaison de 2 lettres, on vérifie alors si il y'a un ID qui se nomme "NA" et qui aurait pu être prit pour un NA dans la commande d'importation, et effectivement c'était le cas.

On remplace le NA par une chaine de caractères "NA" qui représente l'ID.

```
PhotoDRYAD[2237,2] <- "NA"
summary(PhotoDRYAD)
```

```
##      Sex      ID      Age      CaptivityWild
## Min.   :1.000 Length:2475 Min.    : 11.0 Length:2475
## 1st Qu.:1.000 Class :character 1st Qu.:100.0 Class :character
## Median :1.000 Mode  :character Median :179.0 Mode  :character
## Mean   :1.452      Mean   :289.1
## 3rd Qu.:2.000      3rd Qu.:490.0
## Max.   :2.000      Max.   :855.0
##
##      Height      Chest Girth      Neck Circumference      Length
## Min.   :122.0 Min.   :160.0 Min.    : 85.0 Min.    :122.0
## 1st Qu.:187.0 1st Qu.:272.0 1st Qu.:167.0 1st Qu.:188.0
## Median :206.0 Median :303.0 Median :190.0 Median :221.0
## Mean   :207.2 Mean   :303.8 Mean   :192.8 Mean   :226.6
## 3rd Qu.:224.0 3rd Qu.:335.0 3rd Qu.:216.0 3rd Qu.:262.0
## Max.   :279.0 Max.   :452.0 Max.   :305.0 Max.   :365.0
## NA's   :72     NA's   :44     NA's   :1325 NA's   :834
##      Weight      Foot size
## Min.   : 293 Min.    : 64.0
## 1st Qu.:1390 1st Qu.: 97.0
## Median :1935 Median :107.0
## Mean   :2019 Mean   :107.4
## 3rd Qu.:2532 3rd Qu.:117.0
## Max.   :4582 Max.    :375.0
##      NA's      :378
```

En appliquant le summary, on observe que les variables catégorielles Sex et ID et CaptivityWild ne sont pas dans le bon format et donc avant de manipuler ce jeu de données on commence par la rectification de cela:

```
Data <- PhotoDRYAD
Data$Sex <- as.factor(Data$Sex)
Data$ID <- as.factor(Data$ID)
Data$CaptivityWild <- as.factor(Data$CaptivityWild)
summary(Data)
```

```
## Sex      ID      Age      CaptivityWild      Height
## 1:1356 AJ      : 21 Min.    : 11.0 a:2010 Min.    :122.0
## 2:1119 AP      : 21 1st Qu.:100.0 b: 465 1st Qu.:187.0
##      AY      : 21 Median :179.0 Median :206.0
##      BI      : 21 Mean   :289.1 Mean   :207.2
##      BJ      : 21 3rd Qu.:490.0 3rd Qu.:224.0
##      BL      : 21 Max.   :855.0 Max.   :279.0
##      (Other):2349 NA's   :72
## Chest Girth Neck Circumference Length      Weight
## Min.   :160.0 Min.    : 85.0 Min.    :122.0 Min.    : 293
## 1st Qu.:272.0 1st Qu.:167.0 1st Qu.:188.0 1st Qu.:1390
```

```
## Median :303.0 Median :190.0 Median :221.0 Median :1935
## Mean :303.8 Mean :192.8 Mean :226.6 Mean :2019
## 3rd Qu.:335.0 3rd Qu.:216.0 3rd Qu.:262.0 3rd Qu.:2532
## Max. :452.0 Max. :305.0 Max. :365.0 Max. :4582
## NA's :44 NA's :1325 NA's :834
## Foot size
## Min. : 64.0
## 1st Qu.: 97.0
## Median :107.0
## Mean :107.4
## 3rd Qu.:117.0
## Max. :375.0
## NA's :378
```

1 Combien y a-t-il de lignes ? Combien d'animaux différents sont inclus ? Combien y en a-t-il de chaque sexe ?

```
dim(Data)
```

```
## [1] 2475 10
```

```
length(levels(Data$ID))
```

```
## [1] 400
```

```
summary(Data$Sex)
```

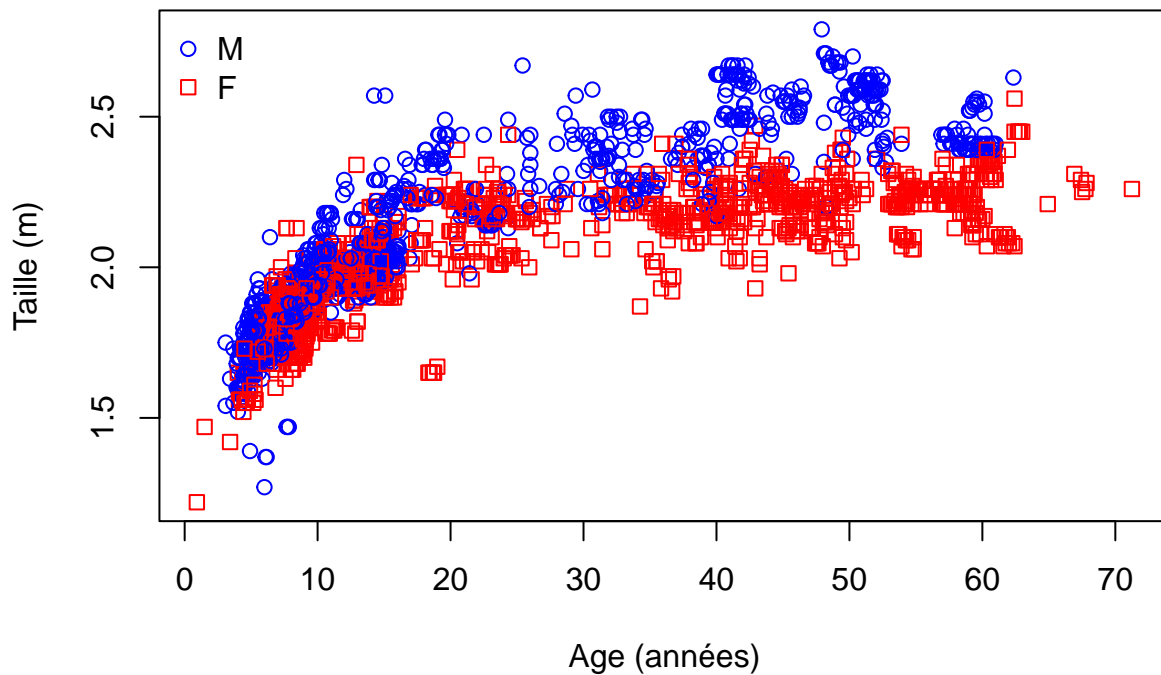
```
## 1 2
## 1356 1119
```

Le jeu de données se compose de :

- 2475 lignes (observations).
- 400 différents animaux.
- 1356 éléphantess et 1119 éléphants (indices de correspondance tirés de l'énoncé).

2 Faire un graphique montrant la relation entre l'âge en années et la taille (variable Height) en mètres, en distinguant les éléphantess des éléphants par des couleurs et des formes de points distinctes.

```
colorplot <- Data$Sex
colorplot <- ifelse(colorplot=="1","Red","Blue")
plot(Data$Age/12,Data$Height/100,col= colorplot, xlab="Age (années)",
      ylab="Taille (m)",pch=as.numeric(Data$Sex)-1)
legend("topleft", legend = c("M","F"), col = c("Blue","Red"), pch=c(1,0), bty = "n")
```



Graphique montrant la relation entre l'âge en années et la taille en mètres des éléphantesses et des éléphants

3 Quelle est la proportion de données manquantes pour chacune des variables ?

```
prop.na.Height <- sum(is.na(Data$Height)) / length(Data$Height)
prop.na.Age <- sum(is.na(Data$Age)) / length(Data$Age)
prop.na.CaptivityWild <- sum(is.na(Data$CaptivityWild)) / length(Data$CaptivityWild)
prop.na.ChestGirth <- sum(is.na(Data$`Chest Girth`)) / length(Data$`Chest Girth`)
prop.na.NeckCircumference <- sum(is.na(Data$`Neck Circumference`)) / length(Data$`Neck Circumference`)
prop.na.Length <- sum(is.na(Data$Length)) / length(Data$Length)
prop.na.Weight <- sum(is.na(Data$Weight)) / length(Data$Weight)
prop.na.Footsize <- sum(is.na(Data$`Foot size`)) / length(Data$`Foot size`)
prop.na <- list(prop.na.Height=prop.na.Height, prop.na.Age=prop.na.Age,
prop.na.CaptivityWild=prop.na.CaptivityWild, prop.na.ChestGirth=prop.na.ChestGirth,
prop.na.NeckCircumference=prop.na.NeckCircumference, prop.na.Length=prop.na.Length,
prop.na.Weight=prop.na.Weight, prop.na.Footsize=prop.na.Footsize)
prop.na
```

```
## $prop.na.Height
## [1] 0.02909091
##
## $prop.na.Age
## [1] 0
##
## $prop.na.CaptivityWild
```

```
## [1] 0
##
## $prop.na.ChestGirth
## [1] 0.01777778
##
## $prop.na.NeckCircumference
## [1] 0.5353535
##
## $prop.na.Length
## [1] 0.3369697
##
## $prop.na.Weight
## [1] 0
##
## $prop.na.Footsize
## [1] 0.1527273
```

La liste contient l'indicateur de la variable et la proportion des NA qu'elle contient.

4 Extraire un jeu de données ne conservant que les colonnes ID, Sex, Age, Height, Chest.Girth, Weight.

```
data4 <- Data[,c(2,1,3,5,6,9)]
head(data4)
```

```
## # A tibble: 6 x 6
##   ID      Sex      Age Height 'Chest Girth' Weight
##   <fct> <fct> <dbl> <dbl>      <dbl> <dbl>
## 1 AA      1      739    207      332    2249
## 2 AA      1      749    207      341    2136
## 3 AA      1      741    208      332    2120
## 4 AA      1      744    208      340    2138
## 5 AA      1      747    208      342    2290
## 6 AA      1      735    210      323    2185
```

Extraction des variables(colonnes) dans l'ordre voulu.

5 Ne conserver de data4 que les lignes où ne manque aucune donnée. Appelons le jeu de données obtenu data5. Combien a-t-il de lignes?

```
data5 <- na.omit(data4)
dim(data5)
```

```
## [1] 2374    6
```

```
dim(data4)[1]-dim(data5)[1]
```

```
## [1] 101
```

-On a 2374 lignes maintenant soit 101 observations supprimées.

J'ai utilisé le `na.omit` car la question précise: "les lignes où ne manque AUCUNE donnée".

si on avait eu d'autres variables qui contiennent aussi des NA pour plusieurs autres observations comme ce qu'on avait pour "Data" avec la variable "NeckCircumference" (question 3), il est préférable d'exclure les NA en précisant les variables avec ce code:

```
data[which(!is.na(colonne de la variable concernée par les NA) & !is.na(une autre colonne)),]
```

```
?cor
mxcor <- cor(data5[,3:6])
mxcor
```

5.1 la matrice des corrélations entre les variables Age, Height, Chest.Girth, Weight. :

```
##           Age      Height Chest Girth      Weight
## Age      1.0000000 0.7793432   0.7511265 0.7778077
## Height   0.7793432 1.0000000   0.9168284 0.9404766
## Chest Girth 0.7511265 0.9168284   1.0000000 0.9372308
## Weight    0.7778077 0.9404766   0.9372308 1.0000000
```

6 extraire du jeu de données data5 deux jeux de données où chaque individu n'apparaîtra qu'une fois; appelons les data6.y et data6.o :

Après avoir utilisé le `na.omit`, il y'aurait des individus "ID" qui n'auront plus aucune mesure et qu'on ne retrouve plus sur data 5, et pour s'assurer qu'on aura la bonne somme des individus on utilise le `droplevels` afin d'enlever les "levels" des IDs qui n'ont plus aucune observation

```
IDs <- levels(droplevels(data5$ID))
nIDs <- length(IDs)
nIDs
```

```
## [1] 393
```

```
y <- NULL
o <- NULL
lengthID <- 0

for (i in 1:nIDs) {
  y <- c(y, lengthID + which.min(data5$Age[data5$ID==IDs[i]]))
  lengthID <- lengthID + length(data5$Age[data5$ID==IDs[i]])
}
data6.y <- data5[y,]

lengthID = 0
for (i in 1:nIDs) {
  o[i] <- lengthID + which.max(data5$Age[data5$ID==IDs[i]])
  lengthID <- lengthID + length(data5$Age[data5$ID==IDs[i]])
}
data6.o <- data5[o,]

head(data6.y)
```

```
## # A tibble: 6 x 6
##   ID      Sex      Age Height 'Chest Girth' Weight
##   <fct> <fct> <dbl>  <dbl>      <dbl>  <dbl>
## 1 AA      1      732    211        343    2127
## 2 AB      1      647    224        356    2769
## 3 AC      1       59    167        248     983
## 4 AD      1     425    224        338    2538
## 5 AE      2     359    244        356    3254
## 6 AF      2     405    244        373    3378
```

```
head(data6.o)
```

```
## # A tibble: 6 x 6
##   ID      Sex      Age Height 'Chest Girth' Weight
##   <fct> <fct> <dbl>  <dbl>      <dbl>  <dbl>
## 1 AA      1      749    207        341    2136
## 2 AB      1      658    230        350    2780
## 3 AC      1       59    167        248     983
## 4 AD      1     436    209        335    2629
## 5 AE      2     370    240        352    3060
## 6 AF      2     416    246        369    3302
```

Ce petit algorithme que j'utilise pour répondre à cette question, fonctionne avec l'approche de parcourir ID par ID, en recherchant les numéros des ligne des observations avec l'âge le plus petit(y)/grand(o).

Quand le i est à 1, data5\$ID==IDs[i] mettra des trues que pour le premier individu, on prend les différents âges de ce même individu et on voit la position de l'âge le plus petit(y)/grand(o), puis on incrémente à chaque fois le nombre d'observations de l' "ID" actuel avec ceux des "ID"s d'avant afin de pouvoir se situer dans l'ID d'après et de continuer la boucle for.

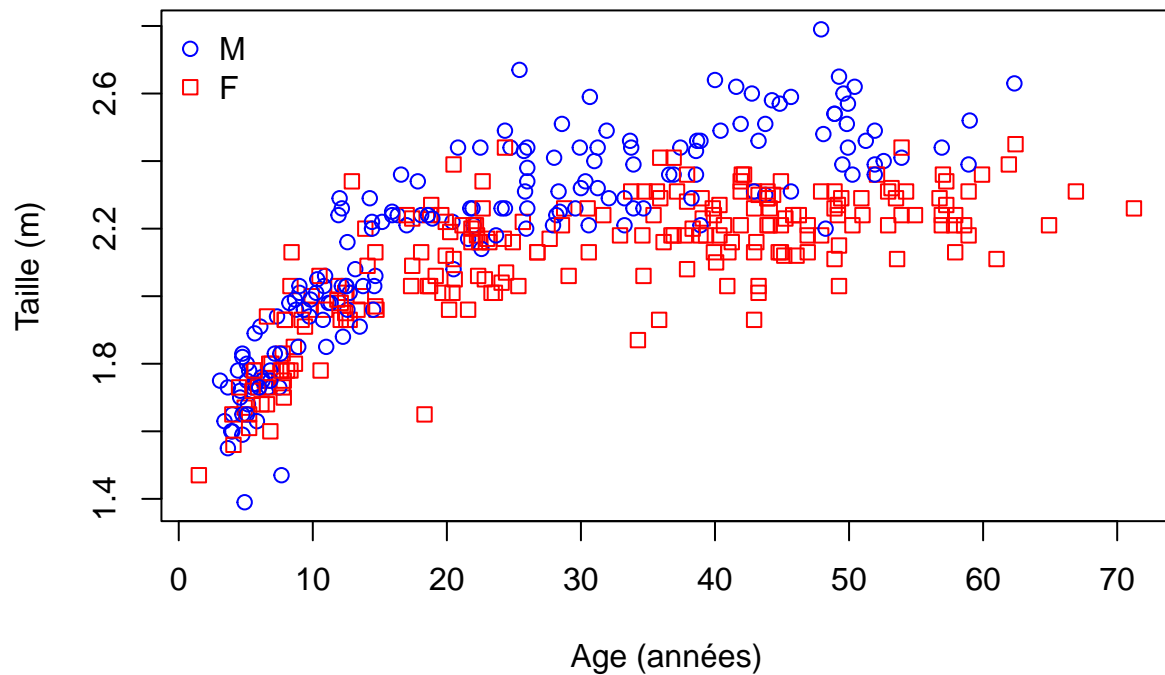
Afin de stocker l'ensemble des numéros de lignes dans un vecteur, j'ai utilisé 2 différentes méthodes, tout en favorisant la méthode qui n'utilise pas le c() récursif comme indiqué sur le cours de R par souci de performances.

La formation "Data Analysis for Life Sciences" de HarvardX, propose aussi l'utilisation du package "dplyr" afin de résoudre ce type de question d'extraction d'observations voulues.

6.1 Faire un graphique analogue à celui de la question 2 pour:

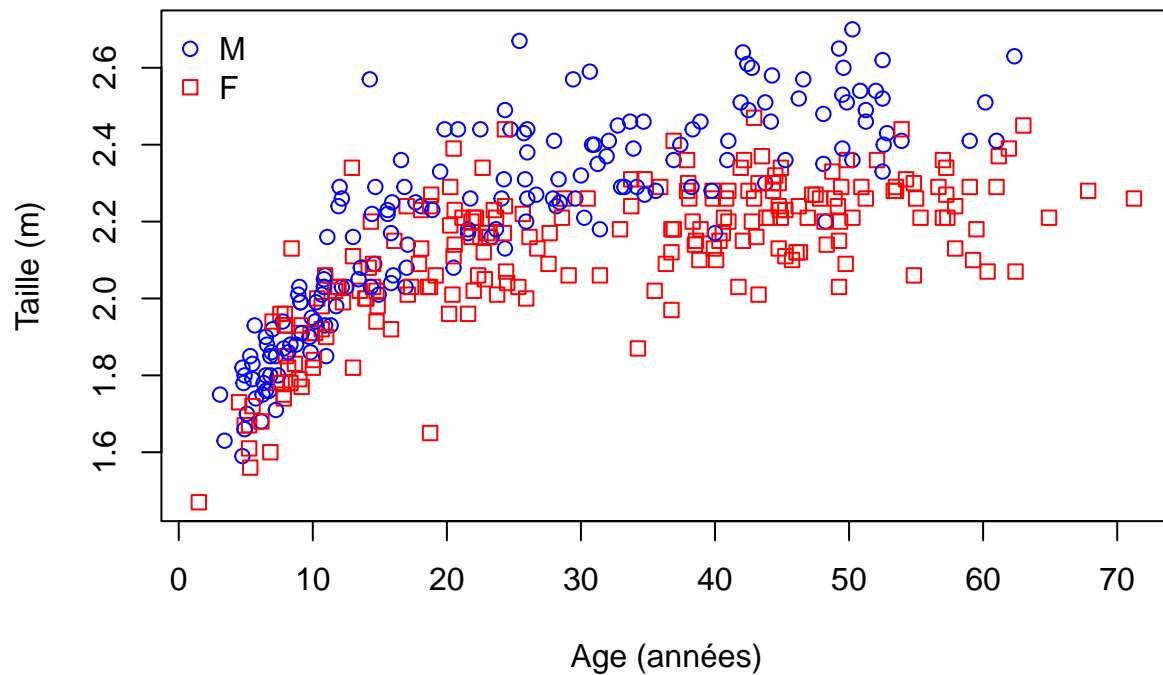
6.1.1 Graphique montrant la relation entre la taille en mètre et l'âge en années de l'observation où l'individu est le plus jeune pour chaque éléphant et éléphante data6.y

```
colorplot6.y <- data6.y$Sex
colorplot6.y <- ifelse(colorplot6.y=="1","Red","Blue")
plot(data6.y$Age/12,data6.y$Height/100,col= colorplot6.y, xlab="Age (années)",
      ylab="Taille (m)",pch=as.numeric(data6.y$Sex)-1)
legend("topleft", legend = c("M","F"), col = c("Blue","Red"), pch=c(1,0), bty = "n")
```



6.1.2 Graphique montrant la relation entre la taille en mètre et l'âge en années de l'observation où l'individu est le plus vieux pour chaque éléphant et éléphante data6.o

```
colorplot6.o <- data6.o$Sex
colorplot6.o <- ifelse(colorplot6.o=="1","Red","Blue")
plot(data6.o$Age/12,data6.o$Height/100,col= colorplot6.o, xlab="Age (années)",
      ylab="Taille (m)",pch=as.numeric(data6.o$Sex)-1)
legend("topleft", legend = c("M","F"), col = c("Blue","Red"), pch=c(1,0), bty = "n")
```

7 Matrices des corrélations entre les variables Age, Height, Chest.Girth, Weight

```
mcor6.y <- cor(data6.y[,3:6])
mcor6.y
```

7.1 Pour data6.y

```
##           Age      Height Chest Girth      Weight
## Age      1.0000000 0.7174606 0.7443640 0.7399276
## Height   0.7174606 1.0000000 0.9060852 0.9238656
## Chest Girth 0.7443640 0.9060852 1.0000000 0.9383781
## Weight    0.7399276 0.9238656 0.9383781 1.0000000
```

```
mcor6.o <- cor(data6.o[,3:6])
mcor6.o
```

7.2 Pour data6.o

```
##           Age      Height Chest Girth      Weight
```

```
## Age          1.0000000 0.6837245 0.7134047 0.7098509
## Height       0.6837245 1.0000000 0.8989707 0.9083056
## Chest Girth  0.7134047 0.8989707 1.0000000 0.9311755
## Weight       0.7098509 0.9083056 0.9311755 1.0000000
```

8 Extraire de data6.o un jeu de données data8 ne contenant que des sujets d'âge ≥ 35 ans. Combien y a-t-il de lignes ?

```
data8 <- subset(data6.o, Age/12 >= 35 )
head(data8)
```

```
## # A tibble: 6 x 6
##   ID      Sex    Age Height 'Chest Girth' Weight
##   <fct> <fct> <dbl>  <dbl>      <dbl>   <dbl>
## 1 AA      1      749    207          341    2136
## 2 AB      1      658    230          350    2780
## 3 AD      1      436    209          335    2629
## 4 AG      1      431    229          323    2202
## 5 AH      1      462    214          327    2125
## 6 AI      1      484    215          325    2274
```

```
dim(data8)
```

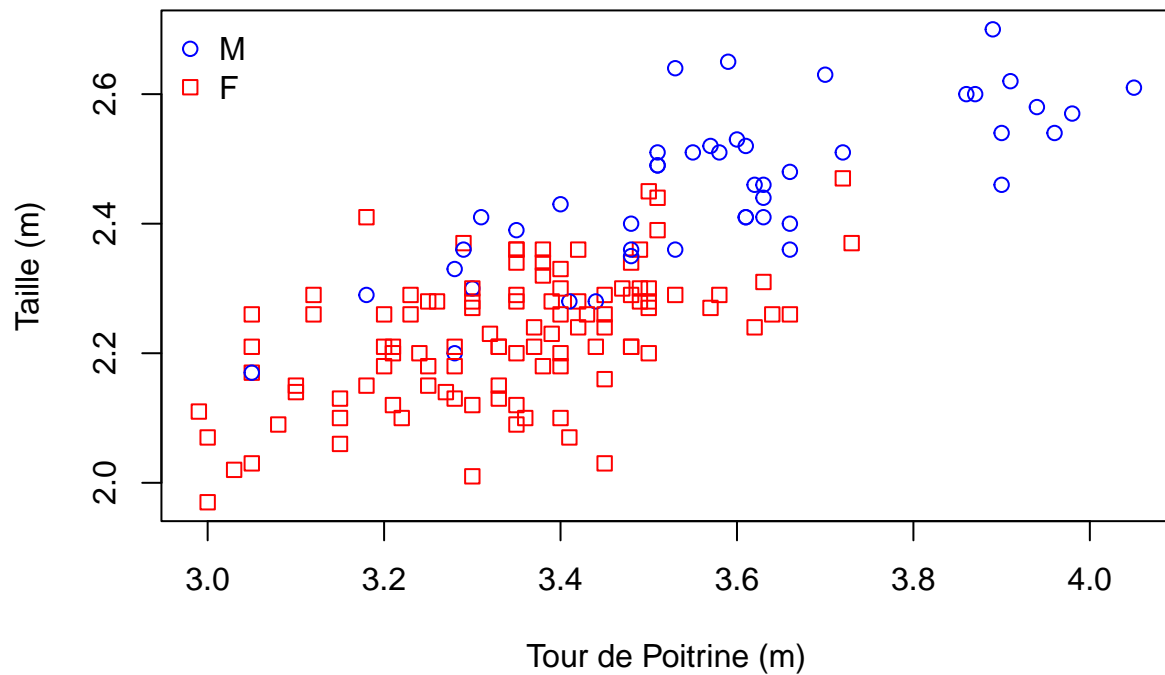
```
## [1] 151 6
```

On a 151 individus pour qui l'âge de la mesure où ils/elles étaient les plus vieux était supérieur ou égale à 35ans.

8.1.1 Graphique montrant la relation entre la taille en mètre et le tour de poitrine en mètre des éléphants et des éléphantes :

•

```
colorplot8 <- data8$Sex
colorplot8 <- ifelse(colorplot8=="1","Red","Blue")
plot(data8$`Chest Girth`/100,data8$Height/100,col= colorplot8, xlab="Tour de Poitrine (m)",
      ylab="Taille (m)",pch=as.numeric(data8$Sex)-1)
legend("topleft", legend = c("M","F"), col = c("Blue","Red"), pch=c(1,0), bty = "n")
```

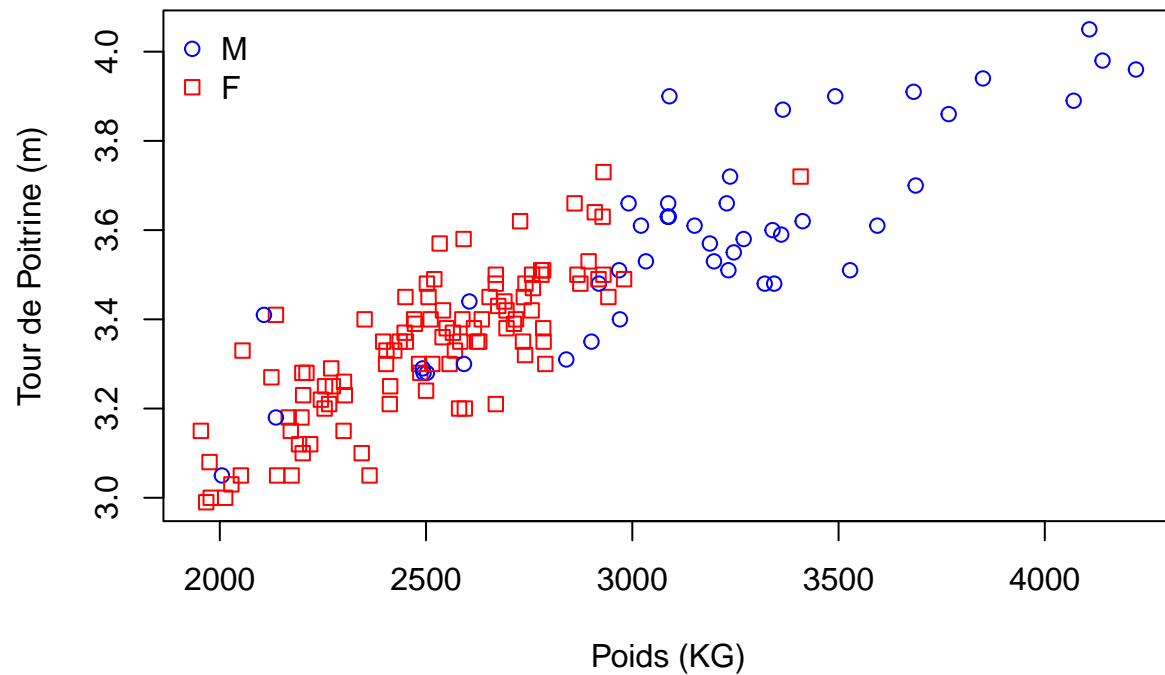


Graphique montrant la relation entre la taille en mètre et le tour de poitrine en mètre des éléphants et des éléphantess

8.1.2 Graphique montrant la relation entre le tour de poitrine en mètre et le poids en KG des éléphants et des éléphantess :

•

```
plot(data8$Weight,data8$`Chest Girth`/100,col= colorplot8, xlab="Poids (KG)",
      ylab="Tour de Poitrine (m)",pch=as.numeric(data8$Sex)-1)
legend("topleft", legend = c("M","F"), col = c("Blue","Red"), pch=c(1,0), bty = "n")
```

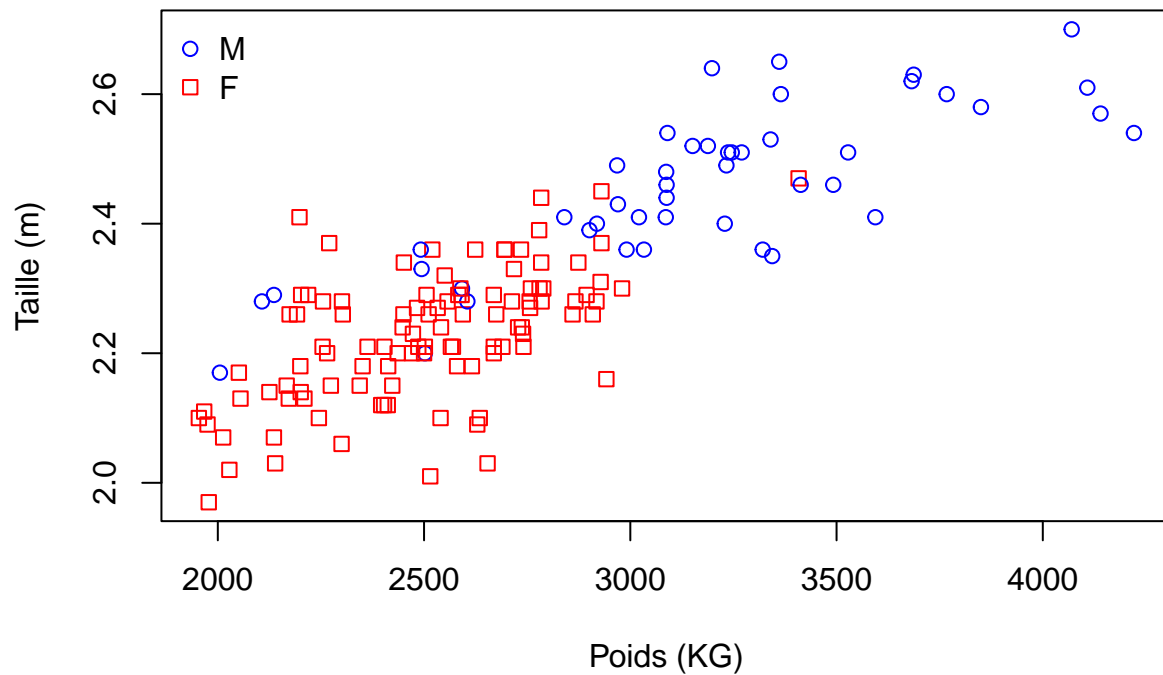


Graphique montrant la relation entre la le tour de poitrine en mètre et le poids en KG des éléphants et des éléphantess

8.1.3 Graphique montrant la relation entre la taille en mètre et le poids en KG des éléphants et des éléphantess :

•

```
plot(data8$Weight,data8$Height/100,col= colorplot8, xlab="Poids (KG)",
      ylab="Taille (m)",pch=as.numeric(data8$Sex)-1)
legend("topleft", legend = c("M","F"), col = c("Blue","Red"), pch=c(1,0), bty = "n")
```



Graphique montrant la relation entre la taille en mètre et le poids en KG des éléphants et des éléphantesses

- Sur les 3 graphiques, on observe une distinction des observations pour les éléphants par rapport aux éléphantesses.

9 l'algorithme Kmeans.

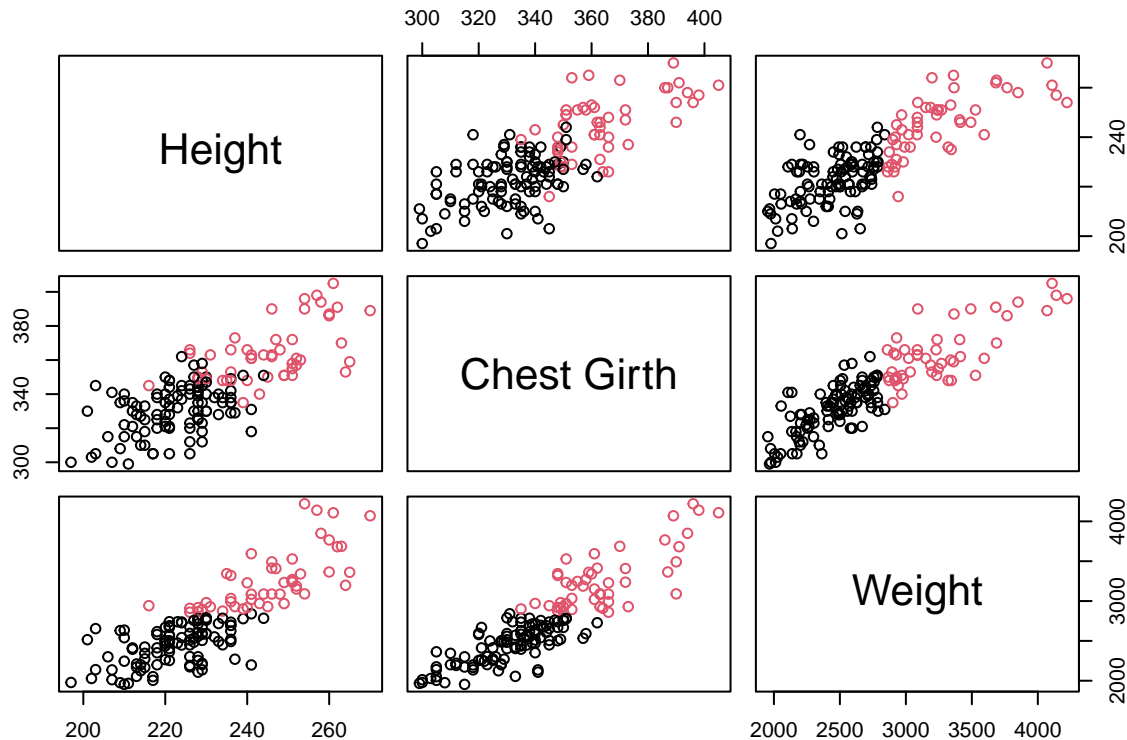
9.1

•

```
?kmeans
head(data8)
```

```
## # A tibble: 6 x 6
##   ID    Sex    Age Height 'Chest Girth' Weight
##   <fct> <fct> <dbl>  <dbl>      <dbl>   <dbl>
## 1 AA     1     749    207        341    2136
## 2 AB     1     658    230        350    2780
## 3 AD     1     436    209        335    2629
## 4 AG     1     431    229        323    2202
## 5 AH     1     462    214        327    2125
## 6 AI     1     484    215        325    2274
```

```
kmeans8 <- kmeans(data8[,4:6],2)
plot(data8[,4:6], col = kmeans8$cluster)
```



En visualisant les clusters créés entre les 3 variables, on voit une claire ressemblance aux graphiques de la question d'avant, ce qui nous ramène à mettre l'hypothèse que les clusters que l'algorithme kmeans a trouvé pour data8 correspondent au sexe des éléphants.

```
summary(kmeans8)
```

```
##           Length Class  Mode
## cluster      151    -none- numeric
## centers         6    -none- numeric
## totss          1    -none- numeric
## withinss       2    -none- numeric
## tot.withinss   1    -none- numeric
## betweenss      1    -none- numeric
## size           2    -none- numeric
## iter           1    -none- numeric
## ifault         1    -none- numeric
```

```
head(kmeans8)
```

```
## $cluster
```

```
##      [1] 1 1 1 1 1 1 2 1 1 2 2 2 1 2 2 2 1 1 1 1 2 1 2 1 1 2 2 2 2 2 1 2 1 2 2 1
##     [38] 1 1 1 2 1 1 2 1 2 2 1 1 1 2 2 2 2 1 1 1 1 2 2 1 1 2 1 2 1 1 2 1 1 2 2 1 2
##     [75] 1 2 1 1 1 2 1 1 1 2 1 1 1 1 1 2 2 2 1 1 1 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2
##    [112] 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2
##    [149] 2 1 1
##
## $centers
##      Height Chest Girth  Weight
## 1 222.3333    330.8039 2447.588
## 2 245.3469    364.2653 3259.531
##
## $totss
## [1] 34139923
##
## $withinss
## [1] 5772129 6492451
##
## $tot.withinss
## [1] 12264580
##
## $betweeness
## [1] 21875343
```

```
prop_mauvais_cluster <- ifelse((sum(kmeans8$cluster != data8$Sex)/length(data8$Sex))<0.5,
(sum(kmeans8$cluster !=data8$Sex)/length(data8$Sex)),
(sum(kmeans8$cluster == data8$Sex)
/length(data8$Sex)))
prop_mauvais_cluster
```

```
## [1] 0.1390728
```

En explorant le résultat de kmeans, on voit que les clusters sont codés en 1 et 2, ce qui nous arrange du fait que le codage de la variable \$Sex prend aussi 1 et 2, le problème c'est que kmeans peut donner le 1 pour le cluster en bas à gauche sur les graphique et en le relancant, l'algorithme peut donner au meme cluster le code 2, l'important c'est que les individus qui sont ensembles dans un cluster restent dans le meme cluster meme si le codage change, afin de remedier à cet aléas, on peut soit fixer la graine du generateur avec set.seed, soit faire une condition avec ifelse pour que si le taux de mauvais classement dépasse 50 %, cela le considerera comme taux de bon classement pour le codage inverse, ainsi on attribue le sexe au cluster qui le correspond le plus à chaque fois, ainsi notre taux de mauvais classement sera stable.

9.2 Kmeans sur les variables centrées réduites

```
•

data8_CR <- data8
data8_CR[,4:6] <- scale(data8[,4:6])
kmeans8_CR <- kmeans(data8_CR[,4:6],2)

prop_mauvais_cluster_CR <- ifelse((sum(kmeans8_CR$cluster != data8$Sex
)/length(data8$Sex))<0.5,(sum(kmeans8_CR$cluster != data8$Sex)/length(data8$Sex))
,(sum(kmeans8_CR$cluster == data8$Sex)/length(data8$Sex)))
prop_mauvais_cluster_CR
```

```
## [1] 0.1192053
```

```
prop_mauvais_cluster - prop_mauvais_cluster_CR
```

```
## [1] 0.01986755
```

Effectivement, cela améliore le résultat de presque de 2%.

Partie 2: Estimation du Déséquilibre de Liaison avec l'algorithme EM

1 E Step

On mets en place une fonction qui prend comme entrée un vecteur f de 4 frequences (fAB, fAb, faB, fab) et qui calcule la valeur tau pour nous:

```
E.step <- function(f) {  
  a <- 2*f[1]*f[4];  
  b <- 2*f[1]*f[4]+2*f[2]*f[3]  
  tau <- a/b  
  return(tau)  
}  
E.step(c(0.25, 0.25, 0.25, 0.25))
```

```
## [1] 0.5
```

```
E.step(c(0.10, 0.20, 0.30, 0.40))
```

```
## [1] 0.4
```

2 M Step

On met en place une fonction qui prend comme entrée une valeur de tau et une matrice d'effectif génotypiques x de dimensions 3x3 et nous donne comme sortie les frequences (fAB, fAb, faB, fab). L'objectif est de ne rien fixer afin d'utiliser cette fonction avec n'importe quelle matrice x et n'importe quelle valeur de tau spécifiée:

```
M.step<- function(tau, x) {  
  a <- (1/(2*sum(x)))*(2*x[1,1]+x[1,2]+x[2,1]+tau*x[2,2])  
  b <- (1/(2*sum(x)))*(2*x[1,3]+x[1,2]+x[2,3]+(1-tau)*x[2,2])  
  c <- (1/(2*sum(x)))*(2*x[3,1]+x[2,1]+x[3,2]+(1-tau)*x[2,2])  
  d <- (1/(2*sum(x)))*(2*x[3,3]+x[3,2]+x[2,3]+tau*x[2,2])  
  
  f <- c(a,b,c,d)  
  names(f) <- c("AA", "Ab", "aB", "ab")  
  return(f)  
}  
X1 <- matrix(c(91, 50, 3, 74, 103, 15, 7, 31, 26), nrow = 3)  
M.step(0.5,X1)
```

```
##          AA          Ab          aB          ab  
## 0.446875 0.213125 0.153125 0.186875
```



```
M.step(0.719, X1)
```

```
##           AA           Ab           aB           ab
## 0.4750713 0.1849288 0.1249288 0.2150713
```

3 Calcule du déséquilibre gamétique D

```
DL<- function(f) {
  a <- f[1]+f[2];
  b <- f[1]+f[3]
  return(f[1]-a*b)
}
```

```
DL( c(0.25, 0.25, 0.25, 0.25) )
```

```
## [1] 0
```

```
DL( c(0.10, 0.20, 0.30, 0.40) )
```

```
## [1] -0.02
```

4 Algorithme E M avec n.iter fois répétition des étapes

4.1 Méthode détaillée On initie l'algorithme avec un tau de 0.5.

•

```
EMO <- function(x, n.iter = 10) {
  tau <- 0.5

  for (i in 1:n.iter) {
    a <- (1/(2*sum(x)))*(2*x[1,1]+x[1,2]+x[2,1]+tau*x[2,2])
    b <- (1/(2*sum(x)))*(2*x[1,3]+x[1,2]+x[2,3]+(1-tau)*x[2,2])
    c <- (1/(2*sum(x)))*(2*x[3,1]+x[2,1]+x[3,2]+(1-tau)*x[2,2])
    d <- (1/(2*sum(x)))*(2*x[3,3]+x[3,2]+x[2,3]+tau*x[2,2])

    f <- c(a,b,c,d)

    g <- 2*f[1]*f[4]
    h <- 2*f[1]*f[4]+2*f[2]*f[3]
    tau <- (g/h)

  }
  a <- f[1]+f[2];
  b <- f[1]+f[3]
  D <- (f[1]-a*b)

  names(f) <- c("AA", "Ab", "aB", "ab")
}
```

```

l <- list(f=f,D=D)
return(l)
}
EMO(X1)

```

```

## $f
##      AA      Ab      aB      ab
## 0.494352 0.165648 0.105648 0.234352
##
## $D
## [1] 0.09835195

```

```
EMO(X1, 1e3)
```

```

## $f
##      AA      Ab      aB      ab
## 0.4943544 0.1656456 0.1056456 0.2343544
##
## $D
## [1] 0.09835444

```

4.2 Méthode avec utilisation des fonctions créées avant:

-

```

EMO <- function(x, n.iter = 10) {
  tau <- 0.5
  for (i in 1:n.iter) {
    f <- M.step(tau,x)
    tau <- E.step(f)
  }
  D <- DL(f)
  l <- list(f=f,D=D)
  return(l)
}
EMO(X1)

```

```

## $f
##      AA      Ab      aB      ab
## 0.494352 0.165648 0.105648 0.234352
##
## $D
##      AA
## 0.09835195

```

```
EMO(X1, 1e3)
```

```

## $f
##      AA      Ab      aB      ab
## 0.4943544 0.1656456 0.1056456 0.2343544

```

```
##
## $D
##      AA
## 0.09835444
```

5 Algorithme E M qui s'arrete une fois la difference entre le dernier tau et celui d'avant atteint une valeur seuil petite définie sans dépasser un nombre n.iter fois d'itération des E et M steps

l'utilisation du debugonce (EM) est conseillée

5.1 Méthode détaillée On initie l'algorithme avec un tau de 0.5.

La différence (diff) entre le dernier tau et celui d'avant est initiée avec un 0.5 du fait que tau a été initié avec 0.5 et que à chaque itération sa valeur diminue et donc tau est la plus grande valeur que ça peut prendre.

On utilise la boucle while car on ne connaît pas le nombre d'itérations au préalable, on définit la condition pour rester dans cette boucle, qui est le fait que la différence entre le dernier tau obtenu après une itération de M et E steps et le tau de l'itération d'avant soit plus grand que l'eps défini et que le nombre d'itérations soit inférieur à n.iter défini aussi.

C'est possible de remplacer le "l" de la liste avec celui qui est en commentaire apres si on veut avoir dans les résultats la valeur de la différence afin de s'assurer qu'elle a atteint une valeur inférieure ou égale à l'eps pour sortir de la boucle

```
EM <- function(x,eps = 1e-6, n.iter = 100) {
  tau <- 0.5
  tau_vect <- tau
  diff <- tau
  i <- 1
  while (diff > eps & i < n.iter) {
    a <- (1/(2*sum(x)))*(2*x[1,1]+x[1,2]+x[2,1]+tau*x[2,2])
    b <- (1/(2*sum(x)))*(2*x[1,3]+x[1,2]+x[2,3]+(1-tau)*x[2,2])
    c <- (1/(2*sum(x)))*(2*x[3,1]+x[2,1]+x[3,2]+(1-tau)*x[2,2])
    d <- (1/(2*sum(x)))*(2*x[3,3]+x[3,2]+x[2,3]+tau*x[2,2])
    f <- c(a,b,c,d)

    g <- 2*f[1]*f[4]
    h <- 2*f[1]*f[4]+2*f[2]*f[3]
    tau <- (g/h)
    #tau_vect <- c(tau_vect,tau)
    tau_vect[length(tau_vect)+1] <- tau
    diff <- (tau_vect[length(tau_vect)]-tau_vect[length(tau_vect)-1])
    i <- i+1
  }
  a <- (1/(2*sum(x)))*(2*x[1,1]+x[1,2]+x[2,1]+tau*x[2,2])
  b <- (1/(2*sum(x)))*(2*x[1,3]+x[1,2]+x[2,3]+(1-tau)*x[2,2])
  c <- (1/(2*sum(x)))*(2*x[3,1]+x[2,1]+x[3,2]+(1-tau)*x[2,2])
  d <- (1/(2*sum(x)))*(2*x[3,3]+x[3,2]+x[2,3]+tau*x[2,2])

  f <- c(a,b,c,d)

  a <- f[1]+f[2];
  b <- f[1]+f[3]
  D <- (f[1]-a*b)
```

```

names(f) <- c("AA", "Ab", "aB", "ab")
l <- list(f=f, D=D, n.iter=i)
#l <- list(f=f, D=D, n.iter=i, diff=diff)
return(l)
}
#debugonce(EM)
EM(X1)

```

```

## $f
##      AA      Ab      aB      ab
## 0.4943544 0.1656456 0.1056456 0.2343544
##
## $D
## [1] 0.09835441
##
## $n.iter
## [1] 14

```

```
EM(X1, 0)
```

```

## $f
##      AA      Ab      aB      ab
## 0.4943544 0.1656456 0.1056456 0.2343544
##
## $D
## [1] 0.09835444
##
## $n.iter
## [1] 33

```

5.1.1 tester la partie n.iter de l'EM algorithm Afin de tester que la limite n.iter fonctionne, on peut reprendre l'exemple EM(X1, 0) qui a eu besoin de n.iter=33 pour atteindre une différence de 0 et on fixe n.iter = 30.

```
EM(X1, 0, 30)
```

```

## $f
##      AA      Ab      aB      ab
## 0.4943544 0.1656456 0.1056456 0.2343544
##
## $D
## [1] 0.09835444
##
## $n.iter
## [1] 30

```

l'algorithme s'arrete bien a 30 itérations.

5.2 Méthode avec utilisation des fonctions créées dans les questions précédentes:

-

```

EM <- function(x,eps = 1e-6, n.iter = 100) {
  tau <- 0.5
  tau_vect <- tau
  diff <- tau
  i <- 1
  while (diff > eps & i < n.iter) {
    f <- M.step(tau,x)
    tau <- E.step(f)
    tau_vect[length(tau_vect)+1] <- tau
    diff <- (tau_vect[length(tau_vect)]-tau_vect[length(tau_vect)-1])
    i <- i+1
  }
  f <- M.step(tau,x)
  D <- DL(f)
  l <- list(f=f,D=D,n.iter=i)
  return(l)
}
EM(X1)

```

```

## $f
##      AA      Ab      aB      ab
## 0.4943544 0.1656456 0.1056456 0.2343544
##
## $D
##      AA
## 0.09835441
##
## $n.iter
## [1] 14

```

```
EM(X1, 0)
```

```

## $f
##      AA      Ab      aB      ab
## 0.4943544 0.1656456 0.1056456 0.2343544
##
## $D
##      AA
## 0.09835444
##
## $n.iter
## [1] 33

```

```
EM(X1, 0, 30)
```

```

## $f
##      AA      Ab      aB      ab
## 0.4943544 0.1656456 0.1056456 0.2343544
##
## $D
##      AA
## 0.09835444

```

```
##  
## $n.iter  
## [1] 30
```