Programmation des algorithmes des moindres carrés, RANSAC et K-means

Ahmed OSMAN

Exercice 1 - Moindres carrés

Question 1 :

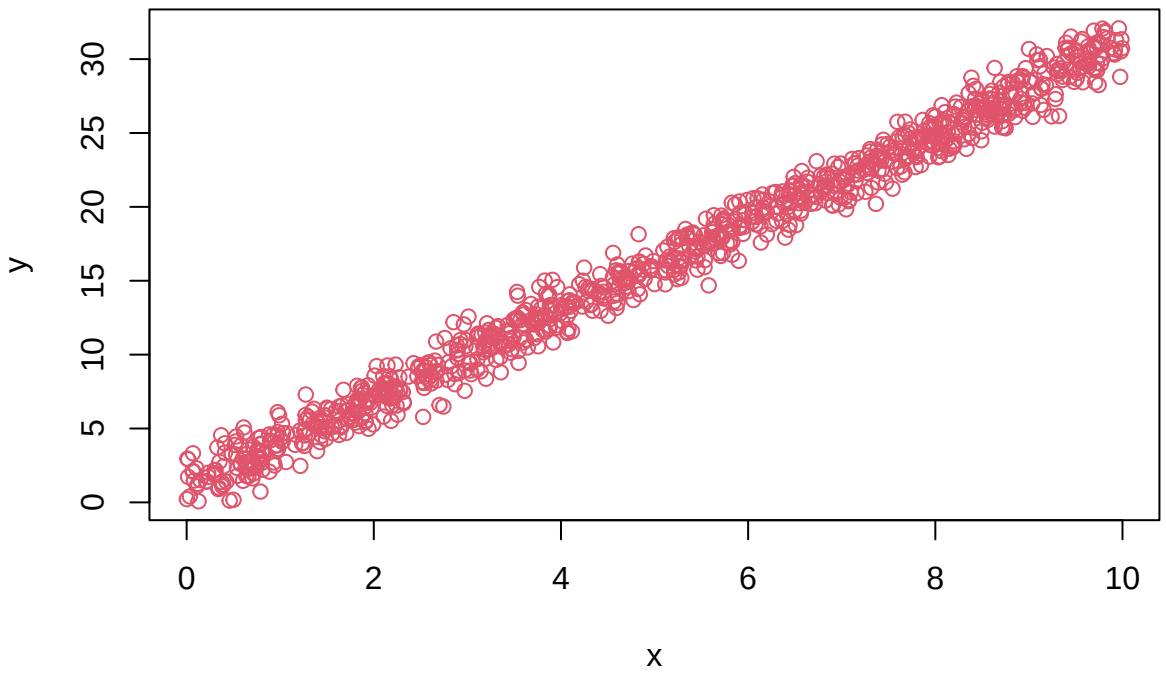
Création des vecteurs x et y.

```
# pour x
N <- 1000
x <- runif(N, 0, 10)

# pour y
a <- 3
b <- 1
eps <- rnorm(N, 0, 1)
y <- a*x + b + eps
```

Question 2 :

Graphique représentant le Nuage de points de la question précédente.

```
plot(x, y,
     main = "Nuage de points",
     col = 2)
```



Question 3 :

Soit S(a, b) la somme suivante :

$$S(a,b) = \sum_{i=1}^{n} |ax_i + b - y_i|^2$$

En dérivant la somme S par rapport aux variables a et b, on obtient les dérivées partielles suivantes :

$$\begin{cases} \frac{\partial S(a,b)}{\partial a} = \sum_{i=1}^{n} 2|x_i|\,|ax_i + b - y_i| \\ \frac{\partial S(a,b)}{\partial b} = \sum_{i=1}^{n} 2\,|ax_i + b - y_i| \end{cases}$$

Pour trouver le min de l'équation S(a, b) on cherche ses points critiques, donc la jacobienne de S doit être égale à 0, alors :

$$\begin{cases} \frac{\partial S(a,b)}{\partial a} = 0 \\ \frac{\partial S(a,b)}{\partial b} = 0 \end{cases} \Leftrightarrow \begin{cases} \sum_{i=1}^{n} |x_i|\,|ax_i + b - y_i| = 0 \\ \sum_{i=1}^{n} |ax_i + b - y_i| = 0 \end{cases}$$

En développant les deux équations obtenues, on obtient le système suivant :

$$\begin{cases} \sum_{i=1}^{n} ax_i^2 + \sum_{i=1}^{n} bx_i = \sum_{i=1}^{n} x_i y_i, \\ \sum_{i=1}^{n} ax_i + \sum_{i=1}^{n} b = \sum_{i=1}^{n} y_i \end{cases}$$

Résolution du système

De la seconde équation on cherche l'expression de b :

$$b = \frac{1}{n}\left(\sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i\right) = \bar{y} - a\bar{x}$$

et en remplaçant l'expression obtenue de la variable b dans la première équation on obtient l'expression de la variable a. On fait pareil pour b après avoir trouvé a.

on obtient donc :

$$a = \frac{\sum_{i=1}^{n} x_i y_i - \bar{y}\sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} x_i^2 - \bar{x}\sum_{i=1}^{n} x_i}$$

et

$$b = \bar{y} - a\bar{x}$$

Question 4 :

Fonction moindrescarres(x, y) qui prend en entrée x et y deux vecteurs de même taille et qui renvoie une liste contenant la pente "a" et l'intercept "b".

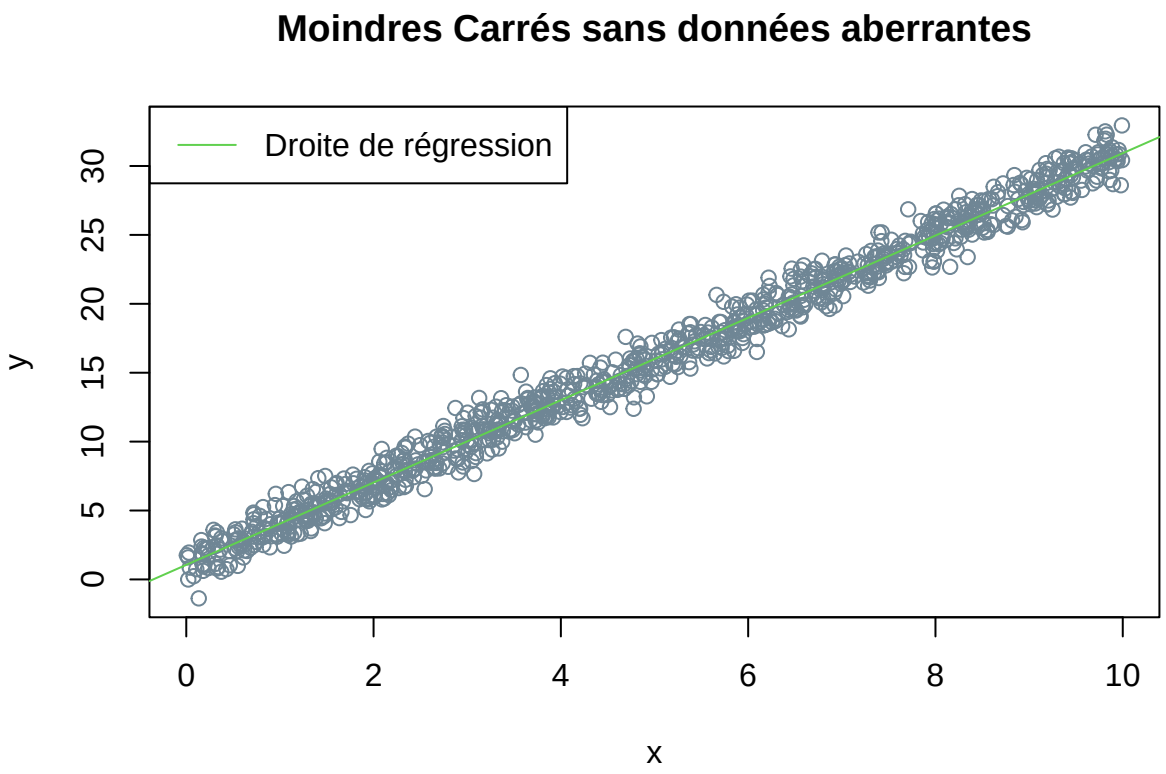
On applique les formules obtenues aux questions précédentes.

```
moindrescarres <- function(x, y) {
  # vecteur x * y
  xy <- x*y

  # vecteur y * xbar ( avec xbar la moyenne empirique des xi)
  y_xbar <- (1/N)*sum(y)*sum(x)

  # vecteur x * xbar
  x_xbar <- (1/N)*sum(x)*sum(x)

  # c.f. formule de la question (3)
  a <- (sum(x*y) - y_xbar)/(sum(x*x) - x_xbar)

  b <- (1/N)*sum(y) - a*(1/N)*sum(x)

  return(list(a = a, b = b))
}
```

Question 5)

```
# De la question (1) :
# pour x
N <- 1000
x <- runif(N, 0, 10)

# pour y
a <- 3
b <- 1
eps <- rnorm(N, 0, 1)
y <- a*x + b + eps
```

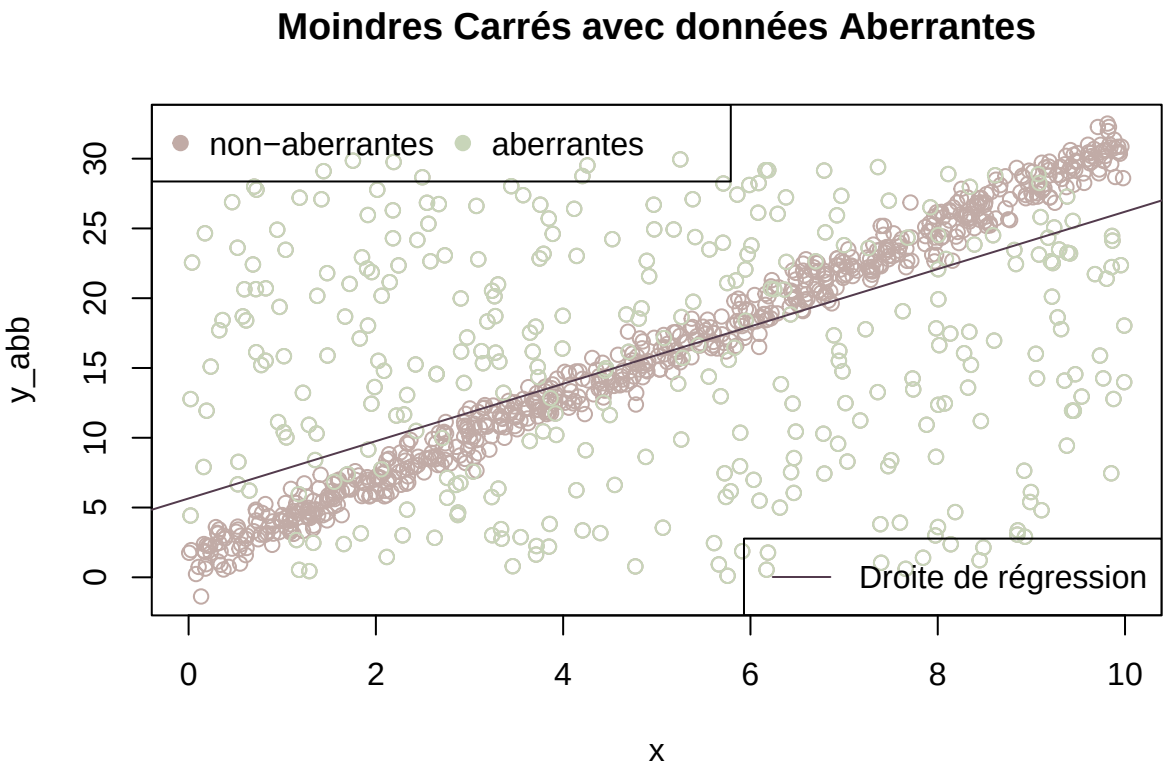
```r
# Moindres Carrés
a_1 <- moindrescarres(x, y)$a
b_1 <- moindrescarres(x, y)$b

# Nuage de points des données
plot(x, y,
     main = "Moindres Carrés sans données aberrantes",
     col = '#6F8695')

# droite de régression
abline(b_1, a_1,
       col = 3)
legend("topleft",
       legend = "Droite de régression",
       col = 3,
       lty = 1)
```



Question 6)

```r
# données aberrantes
z <- runif(N/3, 0, 30)  # ici N/3 = 333

# on enlève les 333 premiers réels du vecteur y
partie_y <- y[-(1:length(z))]

# et on rajoute au vecteur y les données abberantes
```

```r
y_abb <- c(z, partie_y)

# coeff. des Moindres Carrés avec données aberrantes
a_mc <- moindrescarres(x, y_abb)$a
b_mc <- moindrescarres(x, y_abb)$b

# nuage de points des données
plot(x, y_abb,
     main = "Moindres Carrés avec données Aberrantes",
     col = "#C1ABA6")

# représentation graphique des données aberrantes
points(x[c(1:length(z))], z, col = "#C8D5B9")
legend("topleft",
       legend = c("non-aberrantes", "aberrantes"),
       col = c("#C1ABA6", "#C8D5B9"),
       pch = 19,
       horiz = T)

# représentation graphique de la droite de régression
abline(b_mc, a_mc,
       col = "#533B4D")
legend("bottomright",
       legend = c("Droite de régression"),
       col = "#533B4D",
       lty = 1)
```



Que se passe t-il ?

On voit bien que la droite de régression est décentré de la partie dense du nuage de points à cause des

données aberrantes (les points dispersés), contrairement au graphique précédent où il n'y avait pas de données aberrantes.

Conclusion :

On conclut que les données aberrantes ont un effet sur la droite de régression des moindres carrés. Les solutions des moindres carrés manquent de robustesse par rapport aux valeurs aberrantes c'est pour cela qu'on remarque une déviation dans la droite de régression.

Exercice 2 - RANSAC (Random Sample Consensus)

Principe du RANSAC

- On commence par tirer au hasard deux couples (xi, yi) et (xj , yj) des données.
- On calcule (a, b) les paramètres de la droite passant par ces deux couples de points.
- On cherche C : l'ensemble Consensus

$$C = \{(x_k, y_k) : |ax_k + b - y_k| < \alpha\}$$

- On répète toutes ces opérations M fois, et on conserve le maximum du Consensus

Question 1)

Fonction droite(x1, y1, x2, y2) qui calcule les paramètres de la droite y = ax + b

```
droite <- function(x1, y1, x2, y2) {
  # pente
  a <- (y2-y1)/(x2-x1)

  # intercept "y1 = a.x1 + b"
  b <- y1 - a*x1

  return(list(a = a, b = b))
}
```

Question 2)

Fonction ransac qui prend en entrée x et y deux vecteurs de même taille, un seuil alpha, M le nombre d'itérations et P un autre seuil et qui renvoie une liste contenant l'ensemble consensus, la pente "a" et l'intercept "b".

```
ransac <- function(x, y, alpha, M, P) {

  # initialisation
  beta <- 0

  # vecteurs qui contiendront les coefficients ak, bk
  a <- c()
  b <- c()

  # ensemble Consensus (liste de taille M)
```

```
  C <- vector(mode = "list", length = M)

  # initialisation des vecteurs qui contiendrons les résultats souhaités
  Consensus <- c(NA)
  a_cons <- c(NA)
  b_cons <- c(NA)

  for (k in 1:M) {
    # vecteurs aléatoires
    l_et_j <- sample(1:length(x), 2)
    l <- l_et_j[1]
    j <- l_et_j[2]

    xl <- x[l]
    yl <- y[l]

    xj <- x[j]
    yj <- y[j]

    # coefficients de la droite de régression
    coeff <- droite(xl, yl, xj, yj)
    a[k] <- coeff$a
    b[k] <- coeff$b

    # on remplit notre ensemble C
    Ck <- c()
    for (i in 1:length(x)) {
      if (abs(a[k]*x[i] + b[k] - y[i]) <= alpha) {
        Ck <- append(Ck, c(x[i], y[i]))
      }
    }
    C[k] <- list(Ck)

    # on regarde si Card(Ck) > max(P, beta)
    if (length(C[[k]]) > max(P, beta)) {
      Consensus <- C[[k]]
      beta <- length(C[[k]])
      a_cons <- a[k]
      b_cons <- b[k]
    }
  }
  return (list(Consensus = Consensus, a = a_cons, b = b_cons))
}

r <- ransac(x, y = y_abb, 1, 10^3, 100)

print("Les 20 premières valeurs du Consensus")
```

```
## [1] "Les 20 premières valeurs du Consensus"
```

```
print(head(r$Consensus, 20))
```

```
##  [1]  7.260767 23.598286  2.708520  9.978370  3.918100 11.733974  3.865291
```

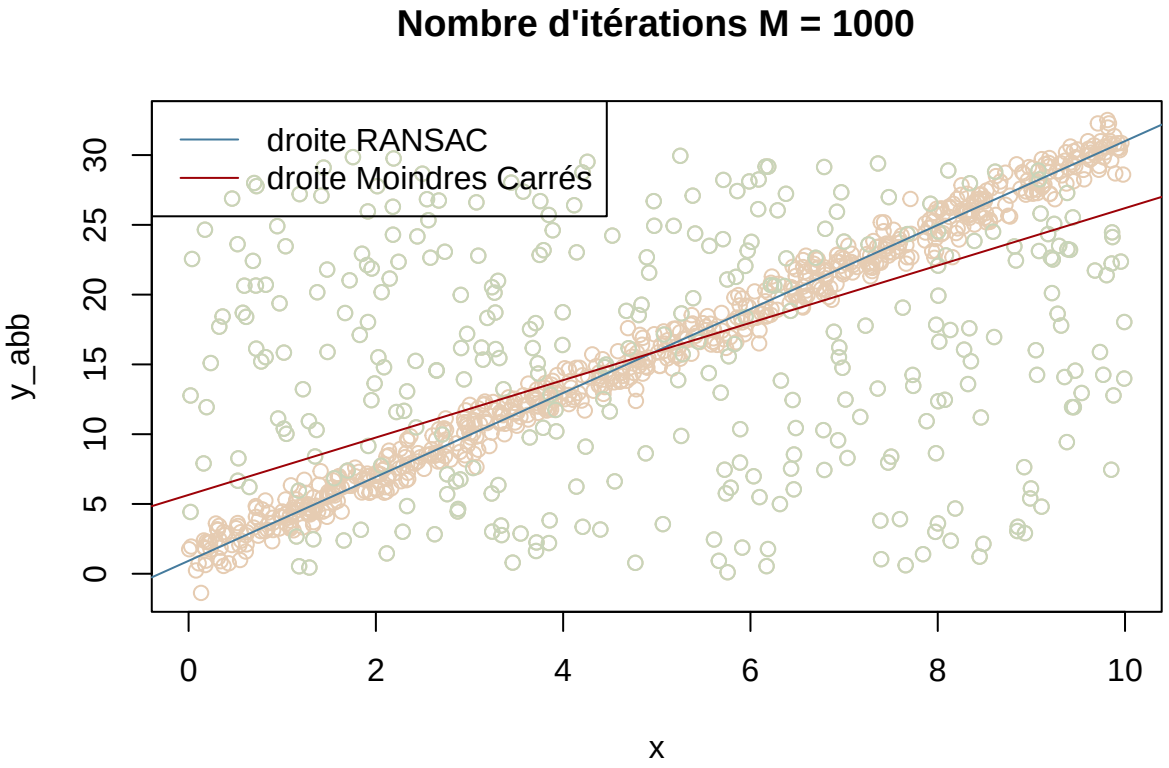
```
##  [8] 12.841115  8.004584 24.526016  6.254691 20.638925  5.444190 16.628645
## [15]  5.948530 18.392868  4.458274 14.989164  4.447419 14.804275
```

```r
# la liste contenant la pente et l'intercept
print(r[-1])
```

```
## $a
## [1] 3.007722
##
## $b
## [1] 0.9239956
```

Question 3)

```r
plot(x, y = y_abb,
     main = "Nombre d'itérations M = 1000",
     col = "#e6ccb2")
# représentation graphique des données aberrantes
points(x[c(1:length(z))], z, col = "#C8D5B9")

# droite de régression de l'algorithme RANSAC
abline(r$b, r$a,
       col = "#457b9d")

# droite de régression des moindres carrés
abline(b_mc, a_mc,
       col = "#9d0208")

legend("topleft",
       legend = c("droite RANSAC", "droite Moindres Carrés"),
       col = c("#457b9d", "#9d0208"),
       lty = 1)
```



D'après le graphique ci-dessus, on remarque que la droite de régression de l'algorithme RANSAC est plus centré aux données non aberrantes par rapport à la droite des moindres carrés.

Conclusion :

On ne voit pas l'effet des données aberrantes sur la droite de l'algorithme RANSAC, les données aberrantes sont presque négligeable devant cet algorithme. On conclut que l'algorithme du RANSAC est plus efficace (plus précis) que l'algorithme des moindres carrés.

Question 4)

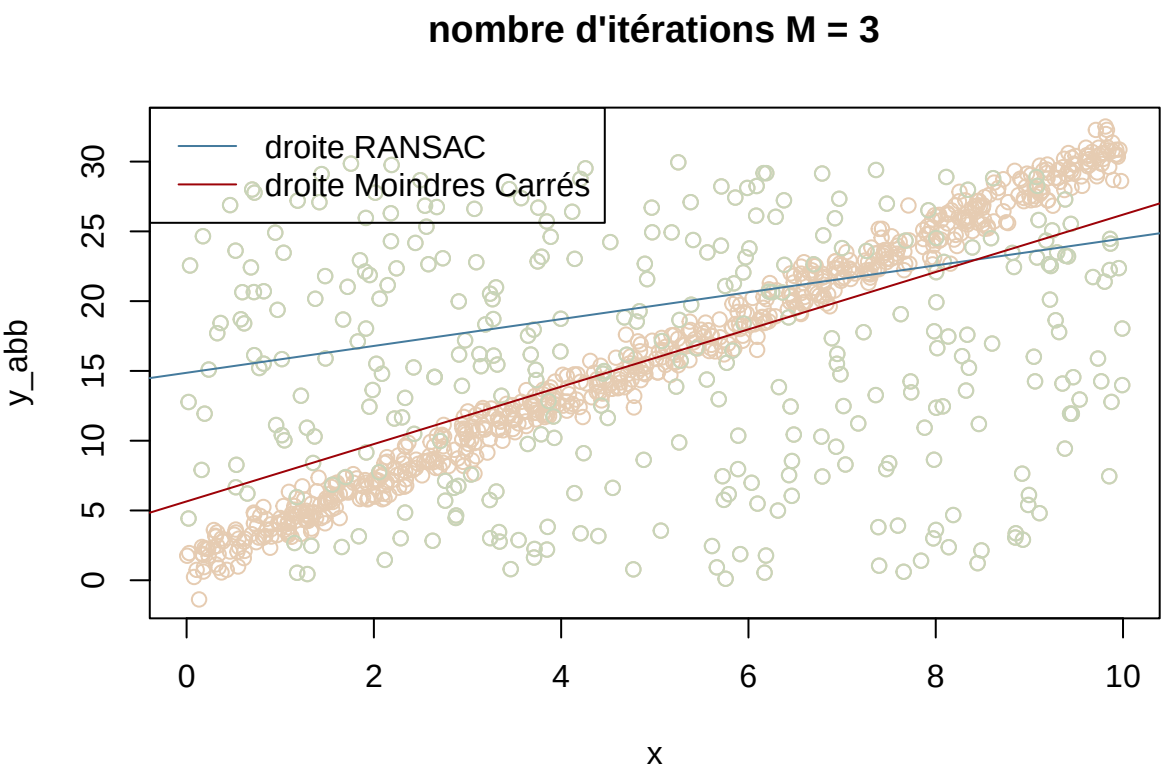
Stabilité de l'algorithme

- Pour le nombre d'itérations M

On a vu quand le nombre d'itérations était grand (M = 1000) dans la question précédente.

```
# --- Pour M petit (M = 3)
r_petit <- ransac(x, y = y_abb, 1, M = 3, 100)

plot(x, y_abb,
     col = "#e6ccb2",
     main = "nombre d'itérations M = 3")
# représentation graphique des données aberrantes
points(x[c(1:length(z))], z, col = "#C8D5B9")

# droite de régression de l'algorithme RANSAC
abline(r_petit$b, r_petit$a,
       col = "#457b9d")
```

```
# droite de régression des moindres carrés
abline(b_mc, a_mc,
       col = "#9d0208")

legend("topleft",
       legend = c("droite RANSAC", "droite Moindres Carrés"),
       col = c("#457b9d", "#9d0208"),
       lty = 1)
```



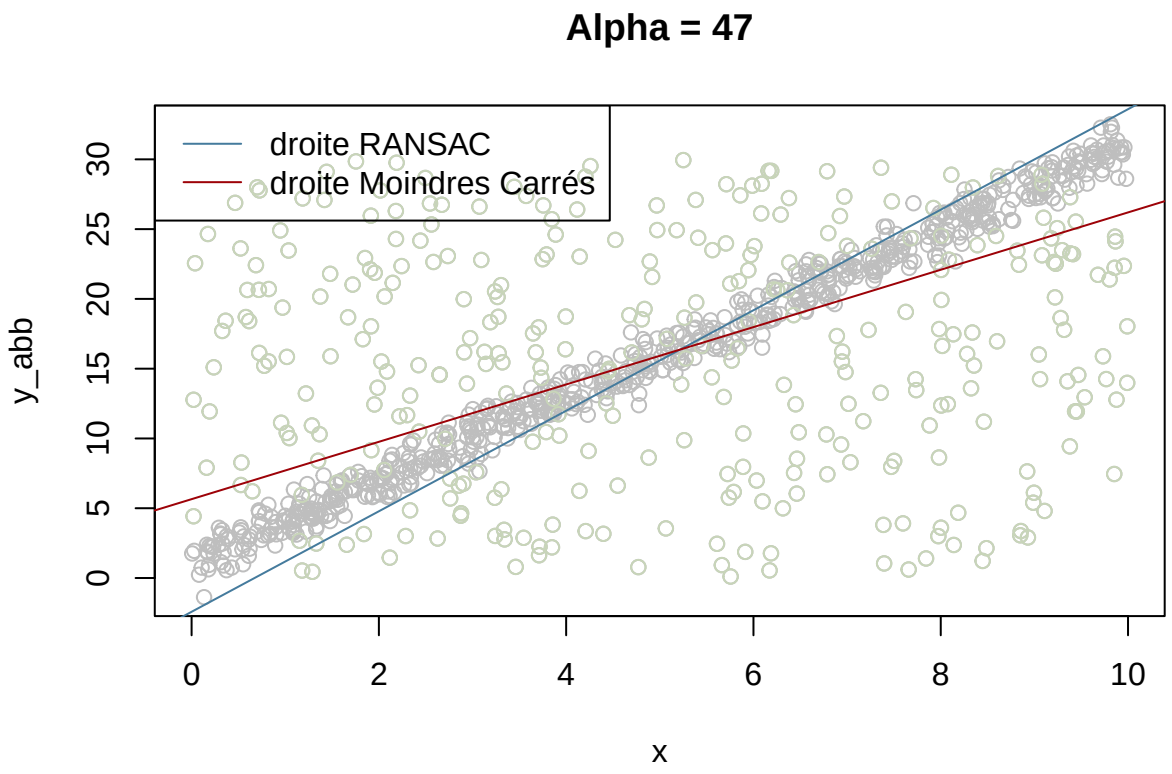
On voit que la droite de RANSAC n'est pas exactement centré aux données non aberrantes comme précédemment, on voit donc l'effet des données aberrantes sur la droite de plus la droite n'est pas stable en re-éxécutant le programme.

- Pour le seuil alpha.

Pour alpha grand :

```
# --- Pour alpha grand
r_alpha <- ransac(x, y = y_abb, alpha = 47, 100,  100)

plot(x, y_abb,
     col = "grey",
     main = "Alpha = 47")
# représentation graphique des données aberrantes
points(x[c(1:length(z))], z, col = "#C8D5B9")

# droite de régression de l'algorithme RANSAC
abline(r_alpha$b, r_alpha$a,
```

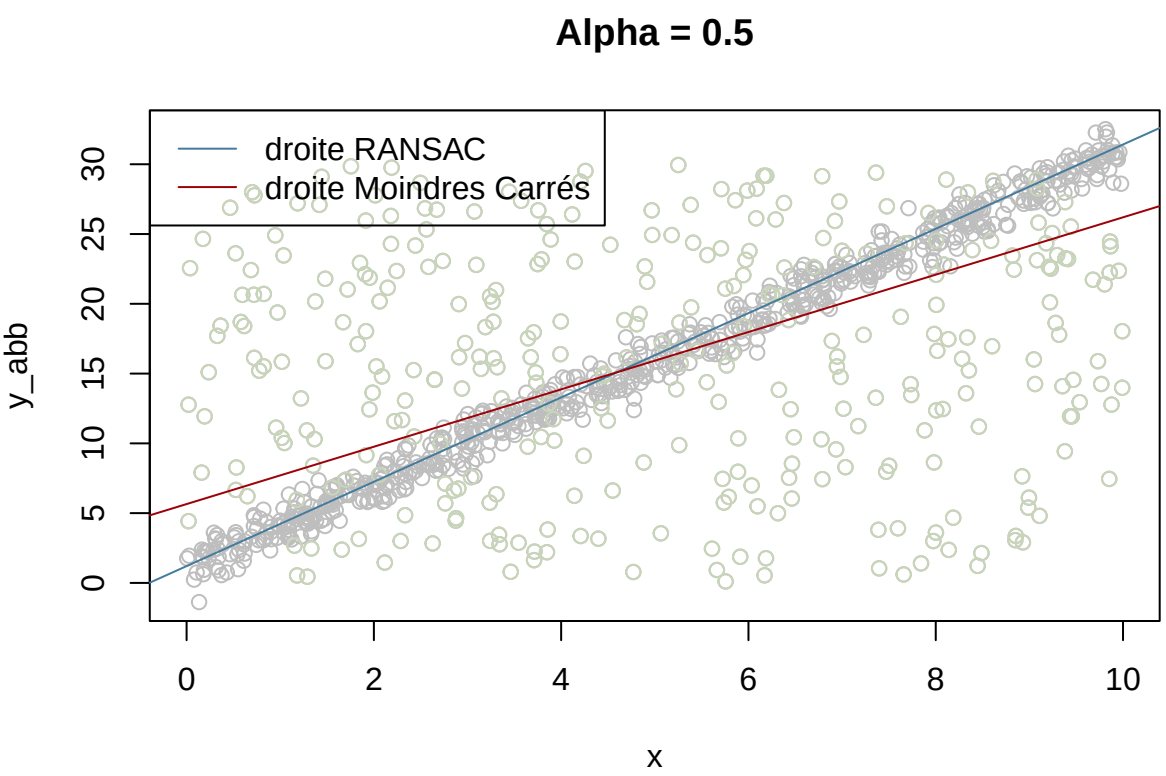
```
       col = "#457b9d")

# droite de régression des moindres carrés
abline(b_mc, a_mc,
       col = "#9d0208")

legend("topleft",
       legend = c("droite RANSAC", "droite Moindres Carrés"),
       col = c("#457b9d", "#9d0208"),
       lty = 1)
```



Pour alpha petit :

```
# --- Pour alpha petit (alpha = 0.5)
r_alpha <- ransac(x, y = y_abb, alpha = 0.5, 100,  100)

plot(x, y_abb,
     col = "grey",
     main = "Alpha = 0.5")
# représentation graphique des données aberrantes
points(x[c(1:length(z))], z, col = "#C8D5B9")

# droite de régression de l'algorithme RANSAC
abline(r_alpha$b, r_alpha$a,
       col = "#457b9d")

# droite de régression des moindres carrés
abline(b_mc, a_mc,
       col = "#9d0208")
```

```
legend("topleft",
       legend = c("droite RANSAC", "droite Moindres Carrés"),
       col = c("#457b9d", "#9d0208"),
       lty = 1)
```



Pour Alpha = 37

On voit bien que la droite de RANSAC s'éloigne de la partie dense du nuage de points.

Pour Alpha = 0.5

On voit bient que la droite de RANSAC se rapproche de la partie dense du nuage de points.

Conclusion :

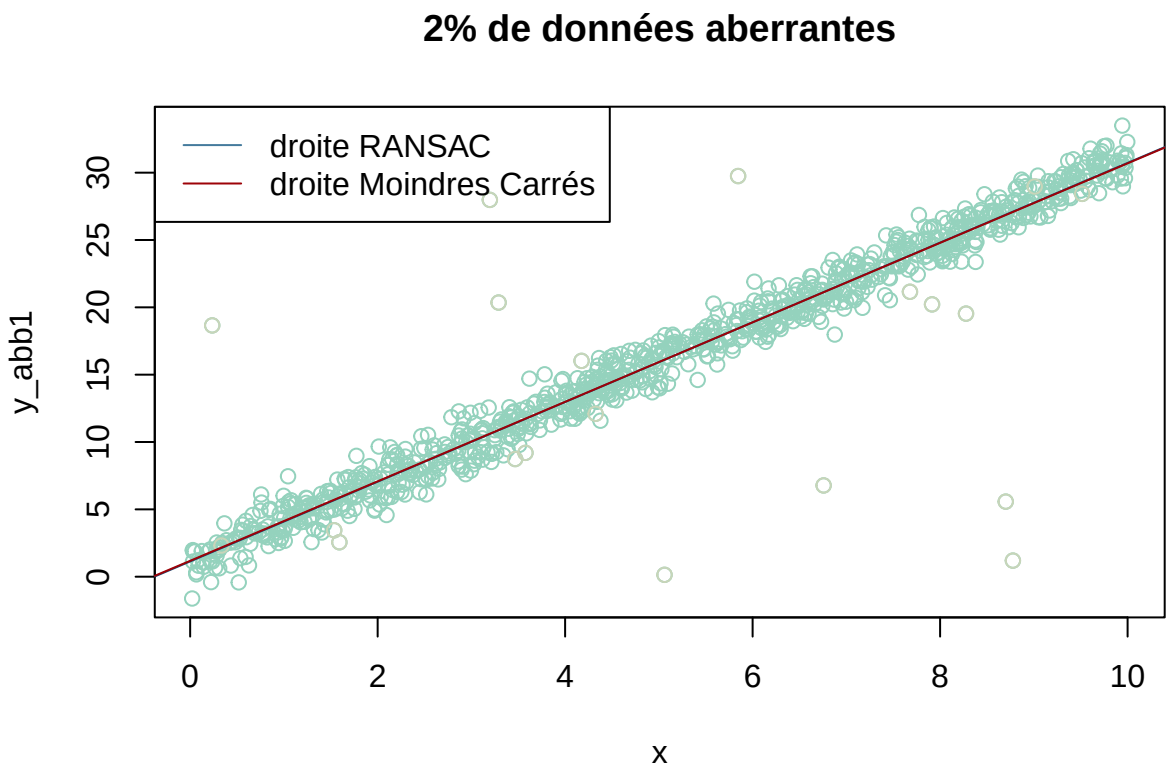
On conclut que, plus alpha devient petit plus la droite de régression se rapproche de la partie dense de nuage de points (des données non aberrantes).

- Pour une proportion p de données aberrantes

Proportion 2% de données aberrantes v.s. Proportion 40% de données aberrantes

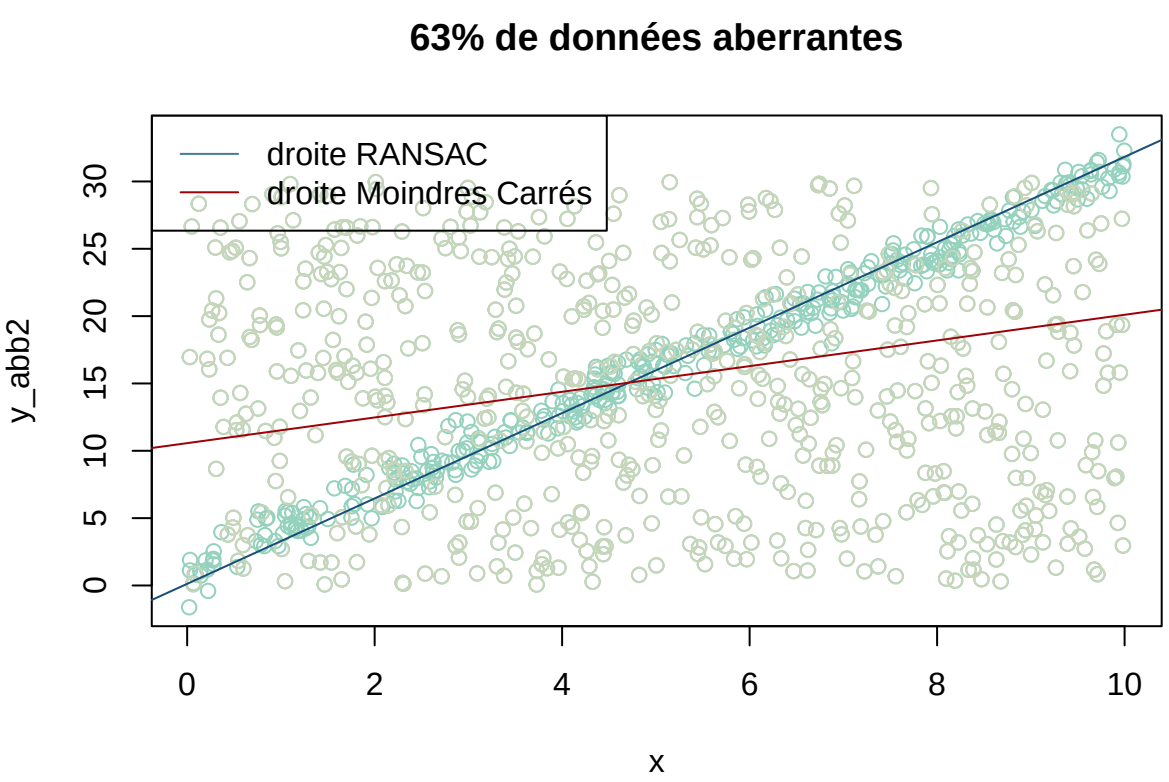
```
# données aberrantes P = 2%
x <- runif(1000, 0, 10)

# pour y
eps <- rnorm(N, 0, 1)
y <- 3*x + 1 + eps

# ----- Proportion 2% -----
```

```r
abb1 <- runif(20, 0, 30)
y_abb1 <- y[-(1:20)]
y_abb1 <- c(abb1, y_abb1)

plot(x, y_abb1,
     main = "2% de données aberrantes",
     col = "#94d2bd")
# représentation graphique des données aberrantes
points(x[c(1:length(abb1))], abb1, col = "#C8D5B9")

r_prop1 <- ransac(x, y = y_abb1, 1, 100, 100)
# droite de régression de l'algorithme RANSAC
abline(r_prop1$b, r_prop1$a,
       col = "#184e77")

# droite de régression des moindres carrés
abline(moindrescarres(x, y_abb1)$b, moindrescarres(x, y_abb1)$a,
       col = "#9d0208")

legend("topleft",
       legend = c("droite RANSAC", "droite Moindres Carrés"),
       col = c("#457b9d", "#9d0208"),
       lty = 1)
```




```r
# ----- Proportion 63% -----

abb2 <- runif(630, 0, 30)
y_abb2 <- y[-(1:630)]
```

```r
y_abb2 <- c(abb2, y_abb2)

plot(x, y_abb2,
     main = "63% de données aberrantes",
     col = "#94d2bd")
# représentation graphique des données aberrantes
points(x[c(1:length(abb2))], abb2, col = "#C8D5B9")

r_prop2 <- ransac(x, y = y_abb2, 1, 100, 100)
# droite de régression de l'algorithme RANSAC
abline(r_prop2$b, r_prop2$a,
       col = "#184e77")

# droite de régression des moindres carrés
abline(moindrescarres(x, y_abb2)$b, moindrescarres(x, y_abb2)$a,
       col = "#9d0208")

legend("topleft",
       legend = c("droite RANSAC", "droite Moindres Carrés"),
       col = c("#457b9d", "#9d0208"),
       lty = 1)
```



Proportion 2% :

On remarque que les deux droites sont confondues. (On ne voit pas la différence des deux algorithmes)

Proportion 63% :

On remarque qu'il y a une très grande différence entre les deux droites des deux algorithmes. On voit très bien que l'algorithme RANSAC est beacoup plus précis dans ce cas.

Donc plus la proption des données aberrantes augmente plus l'efficacité de l'algorithme des moindres carrés diminue.

Conclusion

L'algorithme RANSAC devient plus efficace quand :

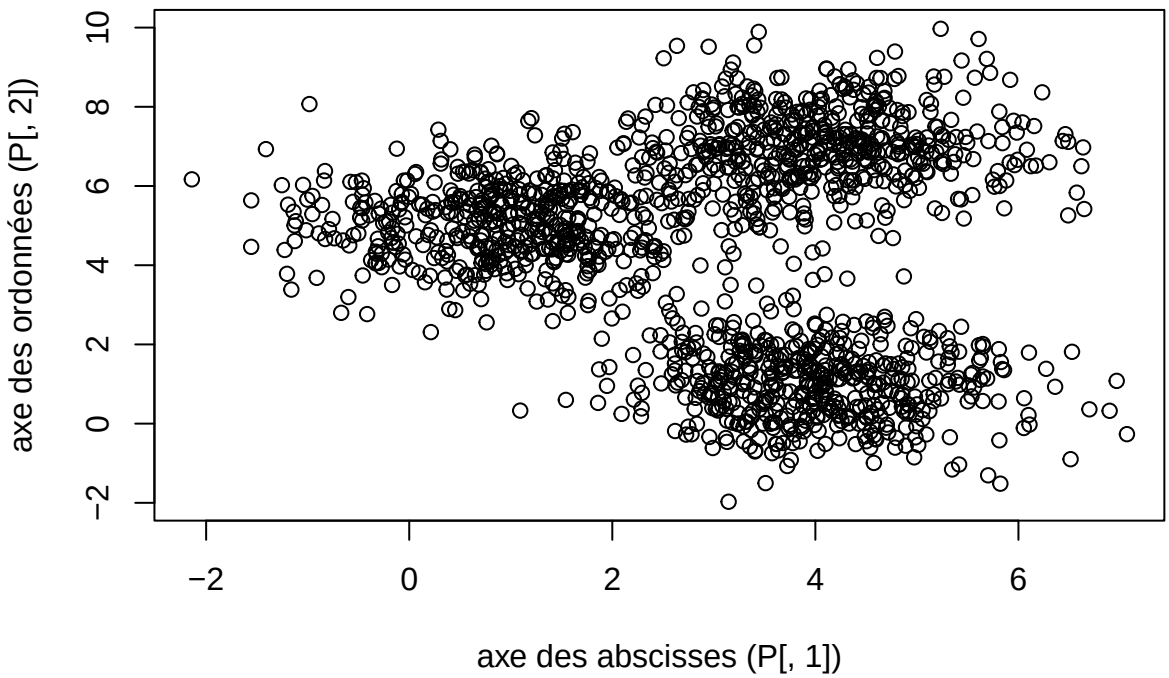
- le nombre d'itérations M augmente
- le seuil alpha diminue

De plus quand la propotion des données aberrantes augmente l'algorithme RANSAC devient plus efficace par rapport à l'algorithme des moindres carrés.

Exercice 3 - Clustering et K-moyennes

Question 1)

```
u <- cbind(1 + rnorm(500), 5 + rnorm(500))
v <- cbind(4 + rnorm(500), 1 + rnorm(500))
w <- cbind(4 + rnorm(500), 7 + rnorm(500))
P <- rbind(u, v, w)
plot(P,
     xlab = "axe des abscisses (P[, 1])",
     ylab = "axe des ordonnées (P[, 2])")
```



Pour u, v et w : u, v et w sont égales à `cbind(r1 + rnorm(500), r2 + rnorm(500))` avec r1 et r2 deux réels.

- `rnorm(500)` : choisit 500 valeurs aléatoire de la distribution d'une loi normale centré réduite, on obtient donc un vecteur de taille 500.

- `r1+rnorm(500) "ou (r2+rnorm(500))"` : rajoute la valeur r1 *"(r2)"* à chaque coordonnées du vecteur obtenu de rnorm(500). La taille des vecteurs restent inchangées.
- `cbind(r1 + rnorm(500), r2 + rnorm(500))` : fusionne les deux vecteurs `r1+rnorm(500)` et `r2+rnorm(500)` qui sont de tailles 500 pour former une matrice de dimension 500*2 (500 lignes et 2 colonnes).

Donc `cbind()` est une fonction qui fusionne horizontalement n vecteurs de même taille pour obtenir une matrice dont le nombre de lignes est égale aux nombres de coordonnées du vecteur et le nombre de colonne est égale à n (le nombre de vecteurs fusionnés).

Donc u, v et w sont des matrices de dimensions 500*2.

Pour P :

- `rbind(u, v, w)` : fusionne verticalement les trois matrices u, v et w pour former une matrice de dimension (500+500+500)*2 (1500 lignes et 2 colonnes).

Donc `rbind()` est une fonction qui fusionne verticalement n vecteurs de même taille pour obtenir une matrice dont le nombre de lignes est égale à n (le nombre de vecteurs fusionnés) et le nombre de colonne est égale aux nombres de coordonnées du vecteur.

`P = rbind(u, v, w)` donc P est une matrice de dimension 1500*2

- `plot(P)` : Affiche le graphique de P, c.à.d affiche P[, 2] *(axe des ordonnées y)* en fonction de P[, 1] *(axe des abscisses x).*

Question 2)

$$\sum_{k=1}^{K} \sum_{P_i \in\ G_k} \|P_i - \mu_k\|_2^2 = \sum_{k=1}^{K} \sum_{P_i \in G_k} \|(x_i, y_i) - \mu_k\|_2^2$$

est équivalent à :

$$\sum_{k=1}^{K} \sum_{P_i \in G_k} (x_i - \mu_{k_x})^2 + (y_i - \mu_{k_y})^2$$

Étapes

1- **Étape 1 :**

On initialise les moyennes mu[[k]] en choisissant 3 points aléatoires différents des données P.

2- **Étape 2 :**

On mesure la distance entre chaque point i de P avec chaque mu[[k]].

3- **Étape 3 :**

On rajoute chaque point i de P dans le groupe cluster[k] si la distance entre le point i de P et mu[[k]] est plus petite que la distance de ce point avec les m[[j]] avec j différent k.

4- **Étape 4 :**

On calcule la moyenne de chaque groupe cluster (Gi).

5- **Étape 5 :**

On répète les étapes précédentes mais en modifiant les moyennes mu qui prendront les valeurs des nouvelles moyennes de chaque cluster (Gi) obtenues.

6- **Étape 6 :**

On s'arrête lorsque les groupes cluster (Gi) restent inchangés. c.à.d lorque les groupes cluster à l'itération i sont égaux aux groupes cluster à l'itération i+1. Si c'est le cas alors les moyennes aussi restent inchangés.

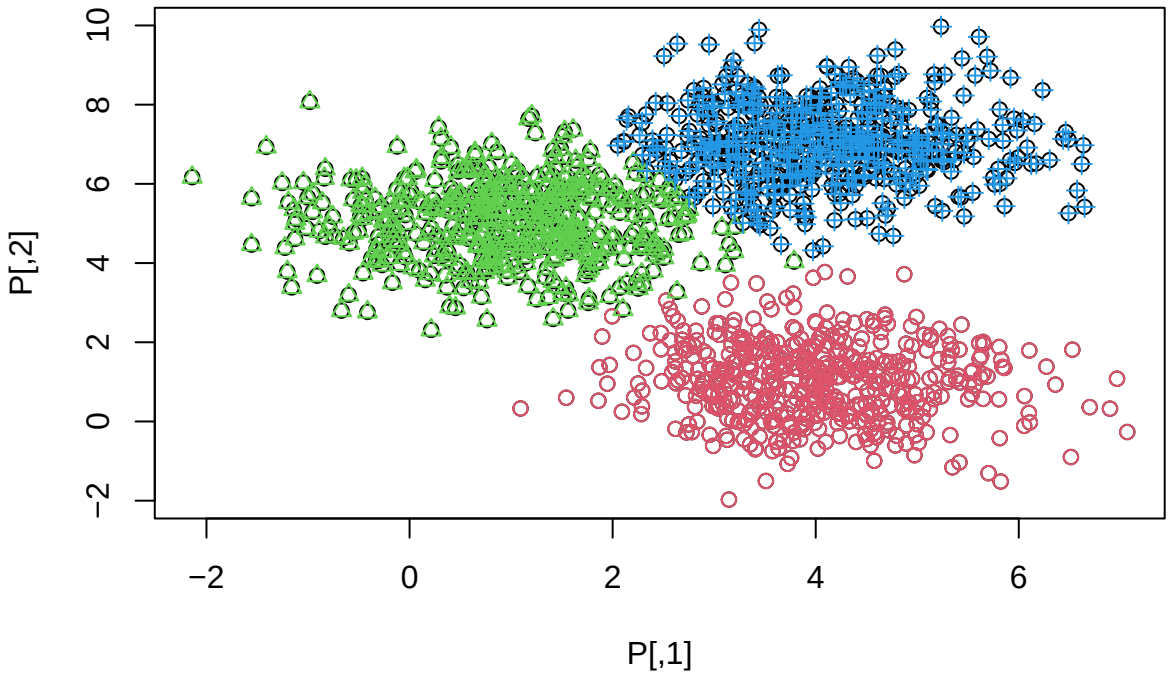
```
kmeans <- function(P, K) {

  # Étape 1. Initialisation
  mu <- vector(mode = "list", length = K)
  s <- sample(1:nrow(P), K)

  for(index in 1:K) {
    mu[[index]] <- P[s[index], ]
  }

  count <- 0

  # Tant que les K clusters sont différents des K anciens clusters
  while (count != K) {
    # Initialisation des Groupes Gi
    cluster <- vector(mode = "list", length = K)

    # Étape 2. Mesurer la distance
    # pour chaque point i on mesure la distance entre ce point et mu[[k]]
    for (i in 1:nrow(P)) {
      distance <- c()
      for (k in 1:K) {
        norme <-(P[i, 1] - mu[[k]][1])^2 + (P[i, 2] - mu[[k]][2])^2
        distance <- c(distance, norme)
      }

      # Étape 3. Ajouter les points dans le groupe
      # on s'interesse à la distance la plus petite entre chaque point i et mu[[k]]
      # pour cela on enregistre la postion du min.
      # EX. si la position vaut 1 alors le point i de P appartiendra au Groupe G1
      pos_min_distance <- which(distance == min(distance))

      cluster[[pos_min_distance]] <- rbind(cluster[[pos_min_distance]], P[i, ])
    }

    # on crée old_mu qui contiendra l'ancienne liste des moyennes des cluster
    # dans les prochaines itérations.
    old_mu <- mu

    # on initialise une variable qui va compter le nombre de fois quand l'ancien
    # cluster sera égale au nouveau.
    count <- 0
    for (j in 1:K) {
      # Étape 4. Calculs des moyennes
      # On calcule la moyenne pour chaque groupe Gi (chaque groupe de cluster)

      # on calcule la moyenne pour les x et les y de chaque Gi (cluster)
```

```
      x_mean <- mean(cluster[[j]][, 1])
      y_mean <- mean(cluster[[j]][, 2])

      # mise à jour de mu (moyenne des Gi)
      mu[[j]] <- c(x_mean, y_mean)

      # on teste si l'ancienne moyenne est égale à la nouvelle moyenne
      # si c'est le cas on a forcément les même groupes Gi, on arrêtra aiinsi
      # l'algorithme.
      if (mu[[j]][1] == old_mu[[j]][1] & mu[[j]][2] == old_mu[[j]][2])
        count <- count + 1
    }

    # Étape 5.
    # On répète les même étapes mais avec la moyenne des cluster et non pas avec
    # les mu[[k]] qu'on avait choisit aléatoirement.

    # Étape 6.
    # On s'arrête lorsque les groupes cluster (Gi) restent inchangés.
    # donc quand les moyennes restent inchangés
    # donc lorsque la variable count vaudra K.

    # On initialise une variable count qui va nous indiquer si la moyenne de
    # l'ancienne Gi est égale à la moyenne de la nouvelle Gi.
    # Si c'est le cas on incrémente count, donc une fois qu'on a count = K alors
    # c.à.d que pour toutes les nouvelles moyennes de Gi pour i allant de 1 à K
    # sont égales aux anciennes moyenne. Donc tout les nouveaux Gi sont égales
    # aux anciens Gi. Fin de L'algorithme.

  }
  return(cluster)
}
```

Question 3)

Application de l'algoritme Kmeans.

```
K = 3
kmoyennes <- kmeans(P, K)
plot(P)

for (k in 1:K) {
  points(kmoyennes[[k]][, 1], kmoyennes[[k]][, 2], col = k+1, pch = k)
}
```



Si le schéma ci-dessus n'est pas très clair (à cause des symboles différents) on peut enlever l'argument `pch = k`. On obtient donc un graphique de la forme suivante :

```
plot(P)
for (k in 1:K) {
  points(kmoyennes[[k]][, 1], kmoyennes[[k]][, 2], col = k+1)
}
```