Exercice 1

On centre tout d'abord la matrice X à l'aide de la matrice de centrage Q8, ça nous servira plus tard :

```
X = Q8 %*% X
```

On calcule la matrice des distances euclidiennes D2 :

```
D2 = as.matrix(dist(X))
D2 = D2^2
```

```
D2
```

```
##       1     2     3     4     5     6     7     8
## 1  0.00 37.25 67.25  1.00  1.00 55.25  1.25 58.25
## 2 37.25  0.00  4.50 48.25 31.25  2.50 36.50  2.50
## 3 67.25  4.50  0.00 81.25 58.25  1.00 65.00  1.00
## 4  1.00 48.25 81.25  0.00  2.00 67.25  1.25 72.25
## 5  1.00 31.25 58.25  2.00  0.00 46.25  0.25 51.25
## 6 55.25  2.50  1.00 67.25 46.25  0.00 52.00  2.00
## 7  1.25 36.50 65.00  1.25  0.25 52.00  0.00 58.00
## 8 58.25  2.50  1.00 72.25 51.25  2.00 58.00  0.00
```

Après avoir obtenu cette précieuse matrice, on peut calculer la matrice des produits scalaires de 2 façons :

```
W = X %*% t(X)
```

Mais normalement, on n'aurait pas accès à X directement, il faut la déduire de la matrice des distances euclidiennes :

```
W = (-1/2)*Q8 %*% D2 %*% Q8
```

Il reste à vérifier que W soit semi définie-positive, pour savoir si la matrice des distances était bien euclidienne :

```
L = eigen(W)$values
L
```

```
## [1]  1.114606e+02  1.758189e+00  1.100935e-14 -3.346521e-16 -9.500733e-16
## [6] -1.069879e-15 -1.845951e-15 -6.005347e-15
```

W est bien semi définie-positive, on va juste arrondir à zero les valeurs négatives très petites, et diagonaliser L :

```
L[L<0] = 0
L = diag(L)
L
```

```
##          [,1]     [,2]         [,3] [,4] [,5] [,6] [,7] [,8]
## [1,] 111.4606 0.000000 0.000000e+00    0    0    0    0    0
## [2,]   0.0000 1.758189 0.000000e+00    0    0    0    0    0
## [3,]   0.0000 0.000000 1.100935e-14    0    0    0    0    0
## [4,]   0.0000 0.000000 0.000000e+00    0    0    0    0    0
## [5,]   0.0000 0.000000 0.000000e+00    0    0    0    0    0
## [6,]   0.0000 0.000000 0.000000e+00    0    0    0    0    0
## [7,]   0.0000 0.000000 0.000000e+00    0    0    0    0    0
## [8,]   0.0000 0.000000 0.000000e+00    0    0    0    0    0
```

Matrice des vecteurs propres :

```
V = eigen(W)$vectors
```

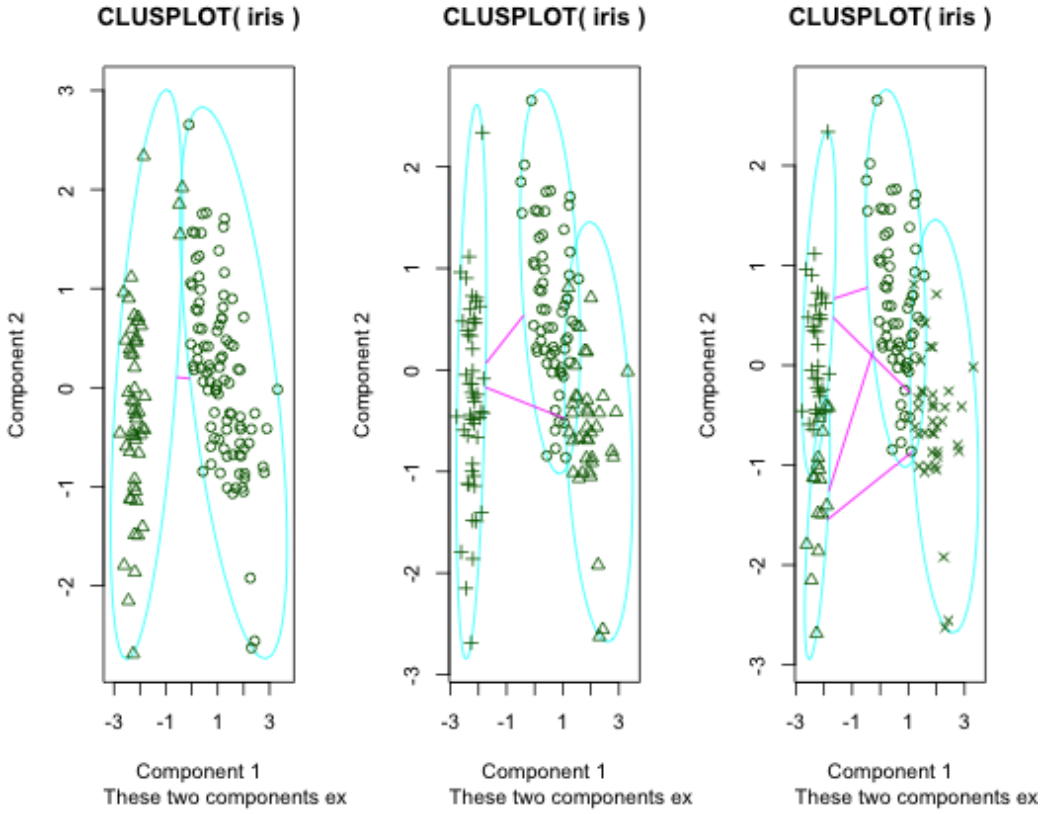
```
1. distX = as.matrix(X)
    distX = distX ^2

2. Méthode 1
  XC = scale(X, scale=T)
  W = XC\%*\%t(XC)

  Méthode 2
  QN = diag(nrow(X)) - matrix(1, nrow(X), nrow(X))/nrow(X)
  W = -1/2*QN\%*\%distX\%*\%QN

3. Pour vérifier si elle est définie semi-positive, il suffit de vérifier que les
eigen(W)

4. L = eigen(W)$values
  L = diag(nrow(X))*L
```

```
  V = eigen(W)$vectors

5. C = V\%*\%sqrt(L)
  pas oublier de retirer les NaN

  plot(C)
  idem à biplot(princomp(X))


m = as.vector(mutation)
b = cmdscale(mutation, 2, T)
c = as.vector(dist(b$points))
plot(b,c) problème mais on est pas loin
qualité à calculer avec les valeurs propres b[,1]$eigen etc... / sum

on refait de même avec cmdscale(mutation, 3, T) jusqu'à 5
```

Exercice 2

Iris

Question 1 - Différents nombres de partiton

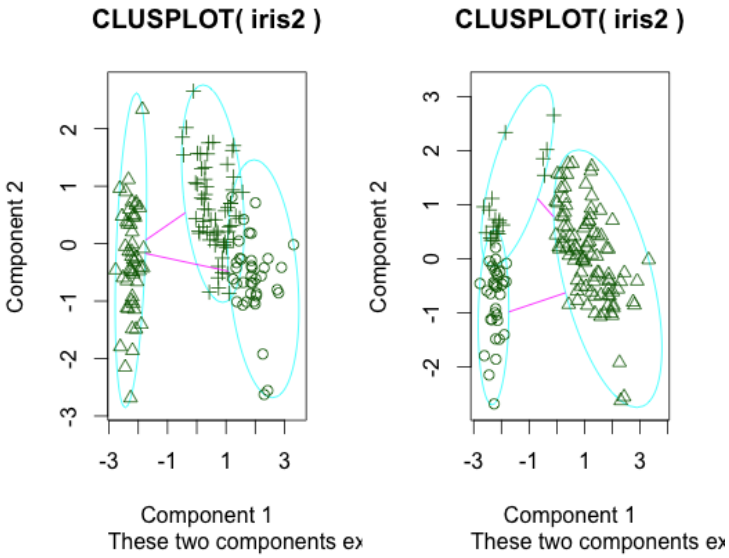


Premièrement, nous remarquons que les partitions n'ont pas toutes le même nombre d'éléments. Ensuite, elle varie suivant le nombre de partitions. En effet, nous pourrions penser qu'entre eux 3 et 4 partitions, l'ajout d'une partition subdiviserait une partition déjà existante. Comme le montre les graphes ci dessous cela n'est pas le cas. En effet les 3 partitions de droit pour K=4 ne sont pas pas contenus dans entièrement 2 partitions de K=3. Toutes les partitions sont redéfinis à chaque fois que nous augmontons le nombre. Nous savons qu'il existe 3 espèces distinctes d'Iris représentées la représentation avec K=3 représente assez fidélement les 3 espèces.

Question 2 - Stabilité des partitions

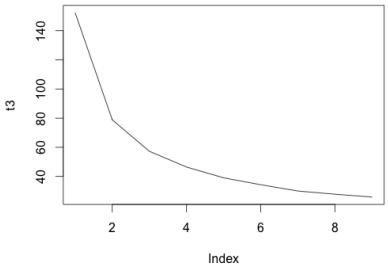
De plus, même pour un même K, dans notre cas k=3, les partitions peuvent changer. Ici, nous avons deux cas différents avec des inerties de

classes de 78.9 ou 143. Cela est du au choix aléatoire des centres au début de l'algorithme, nous tombons parfois sur un minima local. Cela une partition correspond à la classification selon les espéces il s'agit de celle avec le miminum d'inertie intra-classes



Question 3 - Nombre de partitions optimales

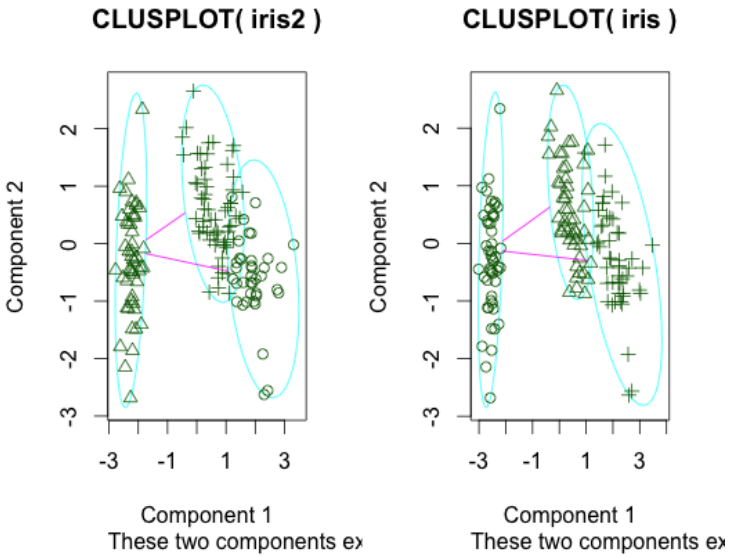
```
$test <- matrix(0, 9, 100)$
$for(j in 1:9){
  for(i in 1:100){
    test[j, i] = kmeans(iris, j+1)$tot.withinss
  }
}$
apply(test, 1, min)
```



La solution optimale semble être en 3 classes. Pourtant celle-ci n'est pas flagrante avec le tableau des minimums des inerties. La méthode du coude

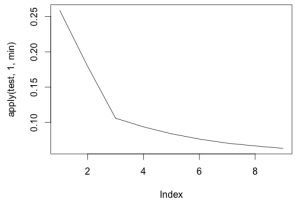
ne fonctionne pas très bien, elle ne fait pas apparaitre de coude. Le minimum d'inertie de fait que diminuer en fonction du nombre de classes. Une solution serait de pénaliser un grand nombre de classes par le nombre d'individus présents dans la classe.

Question 4 - Partitions réelles

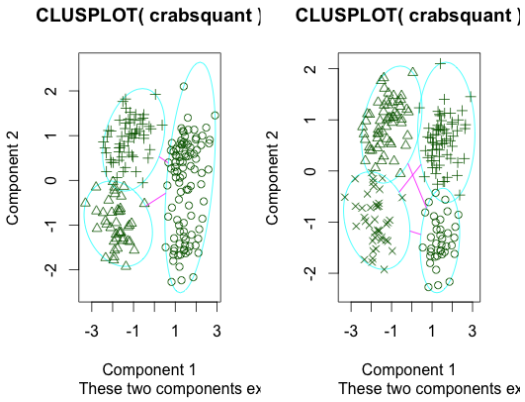


La première espèce (setosa) est bien différenciée suivant la méthode des centre mobiles alors que les espèces versicolor et verginia se chevauchent il est alors plus difficile pour l'algorithme de les différencer efficacement.

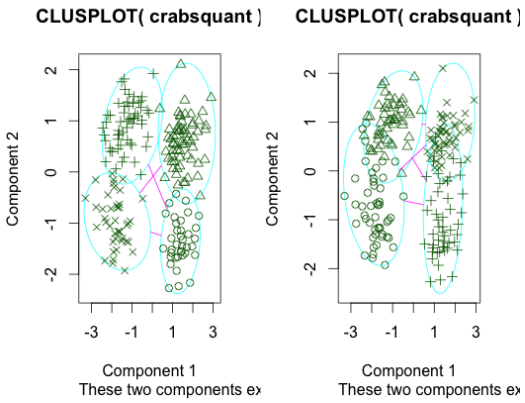
Crabs



Selon le graphe ci dessous de la méthode du coude, le nombre optimal de partition est 3. Le coude est bien visible pour les crabes. Cela est en contradiction avec la classification selon l'espèce et le sexe qui donne 4 classes diffèrentes.



J'ai donc effectué les 2 classifications avec 3 et 4 partitions.



Nous retrouvons à peu près les mêmes classes avec la partitions des centre mobiles ou suivant la classification suivant l'espèce et le sexe. Cela montre bien que les autres variables permettent de déterminer l'espèce et le sexe d'un crabe.

Mutations

```
res = kmeans(mutations2, 2)
plot(cmdscale(mutations), col=res$cluster)

Avec 3 vert au milieu des noirs

4 cluster seulement un point dans le dernier


tableau de contingence pour comparer les partitions table(res$cluster, res2$clus
```

Les