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 :

```r
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 :

```r
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 valeurs pr
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))
```

Exercice 2

```
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 3

library(cluster) clusplot

Iris

```
iris = iris[ ,1:4]
res2 = kmeans(iris, 2)
plot(iris, col= res2$cluster)
clusplot(iris, res2$cluster)i
res2 = kmeans(iris, 3)
> plot(iris, col= res2$cluster)

2 types différents : 143 ou 78.9 pour l'inertie des classes (tot.withinss)

$for(j in 2:10){
  for(i in 1:100){
    test[j, i] = kmeans(iris, j)$tot.withinss
  }
}$
apply(test, 2, min)
La solution qui apparait en 3 classes n'est pas flagrante avec le tableau des minimums des i
Une solution serait de pénaliser un grand nombre de classes par le nombre d'individus présen
```

Crabs

library(MASS)

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$cluster, res2$cluster)