Exercice 1

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

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

On calcule la matrice des distances euclidiennes D2 :

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

```
##       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 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
```

```
##        [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]
## [1,]  13.50  -8.78 -16.56  16.25  11.07 -13.81  12.44 -14.12
## [2,]  -8.78   6.19  11.16 -11.03  -7.71   8.91  -8.84  10.10
## [3,] -16.56  11.16  20.63 -20.31 -14.00  16.88 -15.87  18.07
## [4,]  16.25 -11.03 -20.31  20.00  13.82 -16.56  15.69 -17.87
## [5,]  11.07  -7.71 -14.00  13.82   9.63 -11.25  11.00 -12.56
## [6,] -13.81   8.91  16.88 -16.56 -11.25  14.13 -12.62  14.32
## [7,]  12.44  -8.84 -15.87  15.69  11.00 -12.62  12.63 -14.43
## [8,] -14.12  10.10  18.07 -17.87 -12.56  14.32 -14.43  16.50
```

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

```
eigen(W/8)$values
```

```
## [1]  1.393257e+01  2.197736e-01  1.376168e-15 -4.183151e-17 -1.187592e-16
## [6] -1.337349e-16 -2.307438e-16 -7.506684e-16
```

W est bien semi définie-positive, car ses valeurs propres sont positives ou nulles. Nous avons mis à zéro les valeurs négatives (erreurs d'arrondis de calcul du processeur), et diagonalisé les valeurs propres.

Matrice L des valeurs propres :

```
##          [,1]      [,2]          [,3] [,4] [,5] [,6] [,7] [,8]
## [1,] 13.93257 0.0000000 0.000000e+00    0    0    0    0    0
## [2,]  0.00000 0.2197736 0.000000e+00    0    0    0    0    0
## [3,]  0.00000 0.0000000 1.376168e-15    0    0    0    0    0
## [4,]  0.00000 0.0000000 0.000000e+00    0    0    0    0    0
## [5,]  0.00000 0.0000000 0.000000e+00    0    0    0    0    0
## [6,]  0.00000 0.0000000 0.000000e+00    0    0    0    0    0
## [7,]  0.00000 0.0000000 0.000000e+00    0    0    0    0    0
## [8,]  0.00000 0.0000000 0.000000e+00    0    0    0    0    0
```

Ensuite nous avons calculé la matrice des vecteurs propres, et l'avons normée correctement :

```
V = eigen(W)$vectors
V = V*sqrt(8)  # on norme les vectors propres au sens de (1/n)*I
```

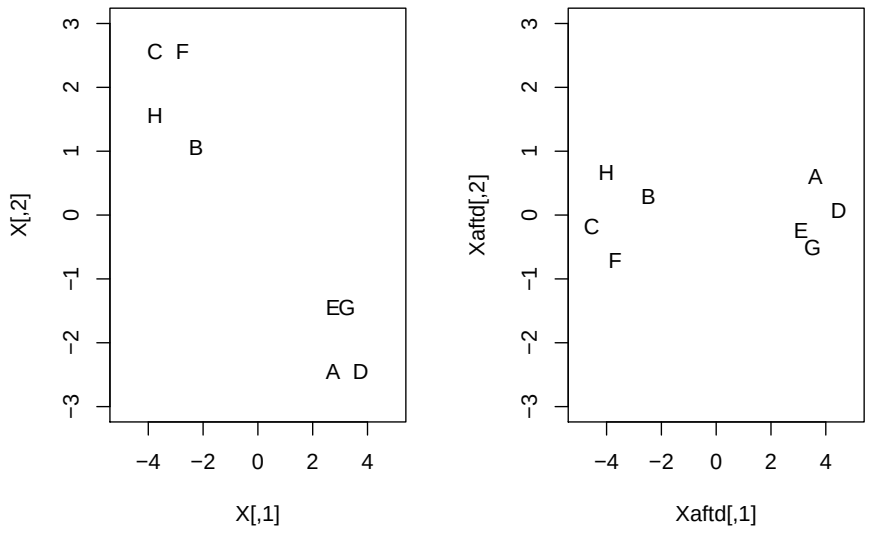
```
round(V,2)
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]
## [1,]  0.97  1.29  0.00  0.00  0.00  0.00  0.00  2.32
## [2,] -0.66  0.63  1.57 -1.70 -0.47 -1.08 -0.65 -0.07
## [3,] -1.22 -0.38 -0.92  0.83  0.02 -1.15 -1.73  0.72
## [4,]  1.20  0.16 -0.93 -1.25  0.85  0.62 -1.63 -0.59
## [5,]  0.83 -0.52 -0.67 -0.26 -2.54 -0.13 -0.21 -0.06
## [6,] -0.99 -1.51 -0.23 -1.30  0.07  1.15  0.29  1.25
## [7,]  0.94 -1.09 -0.54 -0.62  0.75 -1.94  0.95  0.21
## [8,] -1.07  1.43 -1.74 -0.83 -0.16 -0.20  0.95 -0.34
```

Cela nous permet d'en déduire une version approximée par AFTD de X :

```
Xaftd = V %*% sqrt(L)
```

Voici une comparaison graphique de X et Xaftd :



On constate que les résultats sont très proches (aux dissimilarités isométriques près).

Voici une fonction qui fait l'AFTD d'un tableau de distance, et fait un plot du résultat :

```
AFTD = function(D){
    D = as.matrix(D)
    D = D^2
    Q = diag(nrow(D))-matrix(1, nrow(D), nrow(D))/nrow(D)
    W = (-1/2)*Q%*%D%*%Q
    V = eigen(W/nrow(W))$vectors
    V = V*sqrt(nrow(D))
    L = eigen(W/nrow(W))$values
    L[L<0] = 0
    L = diag(L)

    C = V%*%sqrt(L)
    plot(C)
}
```

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 préser
```

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)
```