# TP R 4: Introduction aux statistiques exploratoires

Cours de Programmation

Vitorio Perduca, Master 1 Mathématiques et Applications

UFR Math-Info, Université Paris Descartes, septembre 2017

## Table des matières



## 1. Analyse en composantes principales (ACP)

L'analyse en composantes principales (PCA en anglais) est une méthode de l'analyse des données qui permet de réduire la dimension d'un jeu de données se trouvant dans un espace défini par des variables corrélées tout en gardant le plus possible de variabilité et en minimisant la distorsion. Cela est réalisé en trouvant une nouvelle base orthonormale dont les axes (appelés composantes principales) ne sont pas corrélés et sont ordonnés de façon à que les $M$ premiers axes maximisent la *variabilité* de la projection des données dans l'espace de dimension $M$ correspondant. On peut montrer que cette nouvelle représentation minimise aussi la *distorsion* (une mesure de distance) avec les données originales. L'ACP est utilisée pour mieux connaître les données et peut aider à identifier des structures latentes.

### 1.1 Rappels

On rappelle rapidement les propriétés principale de l'ACP, pour plus de détails voir par exemple le livre de C. Bishop *Pattern recognition and machine learning* (pages 561-565).

**Notations :**

- $\mathbf{x}_n$ : vecteur d'observations, pour $n = 1, \dots, N$. Chaque observation est constituée des valeurs de $D$ variables, donc $\mathbf{x}_n$ est un vecteur dans un espace vectoriel euclidien de dimension $D$.
- $\bar{\mathbf{x}}$ : vecteur des moyennes, une pour chaque dimension (le centre de gravité).

— $\mathbf{u}_1$ : première composante principale définie comme un vecteur t.q. la projection des données sur $u_1$ a variance $V_1$ maximale.
— $\mathbf{u}_1, \ldots, \mathbf{u}_D$ : toutes les composantes principales ordonnées de façon à que $V_1 \geq V_2 \geq \ldots \geq V_D$ où $V_j$ est la variance de la projection des données sur l'axe $\mathbf{u}_j$. Pour tout $M = 1, \ldots, D$, les composantes principales sont définies t.q. $\mathbf{u}_1, \ldots, \mathbf{u}_M$ constituent une base orthonormale du sous espace qui maximise la *variabilité* $V_1 + \ldots + V_M$.
— $\lambda_1, \ldots, \lambda_D$ : valeurs propres de la matrice de covariance des données.

**Proprietés :**

— $\mathbf{u}_1, \ldots, \mathbf{u}_D$ sont les vecteurs propres de la matrice de covariance des données et $V_j = \lambda_j$, pour $j = 1, \ldots, D$. Les composantes des vecteurs propres dans la base canonique sont parfois appelées les *loadings* (mais attention : dans certains contextes les loadings sont les vecteurs propres multipliés par les racines des valeurs propres).
— La matrice de covariance de $\mathbf{u}_1, \ldots, \mathbf{u}_D$ est la matrice diagonale avec les valeurs propres sur la diagonale.
— La projection de $\mathbf{x}_n$ dans le sous espace généré par $\mathbf{u}_1, \ldots, \mathbf{u}_M$ a pour composantes, dans cette même base, les scalaires $\mathbf{u}_j^T \mathbf{x}_n$, $j = 1, \ldots, M$, ce qu'on appelle parfois les *scores*.
— Pour tout $M = 1, \ldots, D$, on considère la *reconstruction* de $\mathbf{x}_n$ :

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} (\mathbf{x}_n^T \mathbf{u}_i)\mathbf{u}_i + \sum_{j=M+1}^{D} (\bar{\mathbf{x}}^T \mathbf{u}_j)\mathbf{u}_j.$$

Les composantes principales ont la propriété de minimiser la *distorsion* : en effet on peut montrer que

$$J_M := \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{x}_n - \tilde{\mathbf{x}}_n||_2^2 = \sum_{j=M+1}^{D} \lambda_j.$$

est minimale.

## 1.2 La fonction `prcomp()`

Plusieurs fonctions existent en `R` pour l'ACP, ici on se concentrera sur la fonction de base `prcomp()`. On travaillera sur les quatre variables quantitatives du jeu de données `iris` :

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
mydata <- iris[,1:4]
```

`prcomp()` fait l'ACP en réalisant la *décomposition en valeurs singulières (SVD)* de la matrice des données. On peut montrer que la SVD de la matrice des données **centrées** donne les valeurs et les vecteur propres de façon plus efficace que la diagonalisation directe de la matrice de covariance des données originales. En général on recommande de **centrer et réduire** toutes les variables avant de procéder à l'ACP. Le centrage à pour effet d'avoir $\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} (\mathbf{x}_n^T \mathbf{u}_i)\mathbf{u}_i$.

```r
pca <- prcomp(mydata,center=TRUE,scale.=TRUE)
#center=TRUE (défaut) et scale.=TRUE centrent et réduisent chaque variable
#Attention! Par défaut scale.=FALSE
pca
```

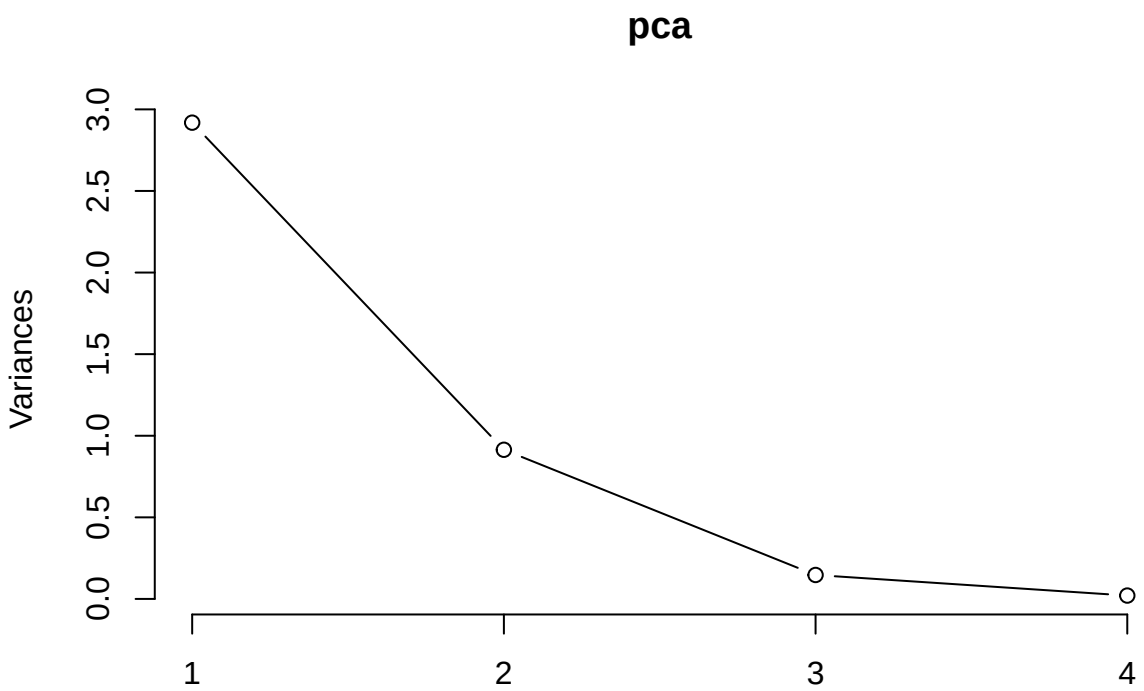
```
## Standard deviations (1, .., p=4):
## [1] 1.7083611 0.9560494 0.3830886 0.1439265
##
## Rotation (n x k) = (4 x 4):
##                     PC1         PC2        PC3        PC4
## Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
## Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
## Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
## Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
```

Variances $V_1, \ldots, V_4$ (c'est à dire les valeurs propres $\lambda_1, \ldots \lambda_4$) :

```r
pca$sdev^2
```

```
## [1] 2.91849782 0.91403047 0.14675688 0.02071484
```

```r
plot(pca,type='l')
```




Proportion de variabilité expliquée par les composantes principales :

```r
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4
```

```
## Standard deviation     1.7084 0.9560 0.38309 0.14393
## Proportion of Variance 0.7296 0.2285 0.03669 0.00518
## Cumulative Proportion  0.7296 0.9581 0.99482 1.00000
```

Ainsi on voit que les deux premières composantes principales expliquent plus de 95% de la variabilité totale.

Composantes principales (colonnes avec les composantes des vecteurs propres dans la base des variables initiales) :

```
pca$rotation
```

```
##                     PC1         PC2        PC3        PC4
## Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
## Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
## Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
## Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
```

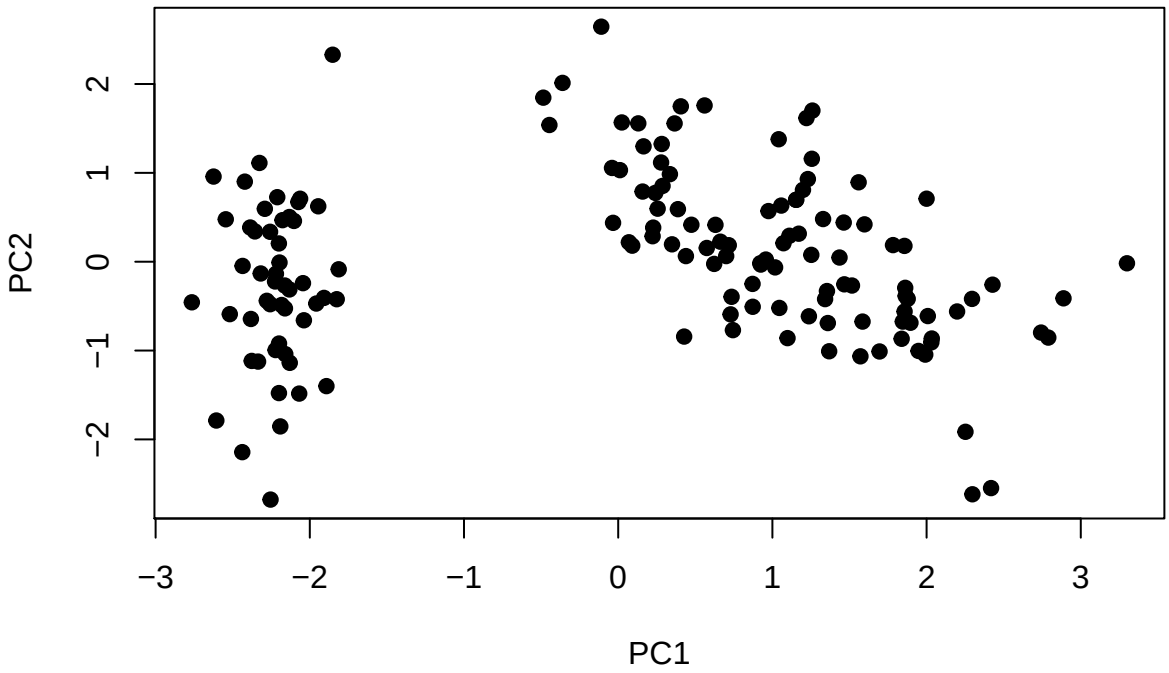
Composantes des données (centrées par défaut) dans la base des composantes principales :

```
head(pca$x)
```

```
##            PC1        PC2         PC3          PC4
## [1,] -2.257141 -0.4784238  0.12727962  0.024087508
## [2,] -2.074013  0.6718827  0.23382552  0.102662845
## [3,] -2.356335  0.3407664 -0.04405390  0.028282305
## [4,] -2.291707  0.5953999 -0.09098530 -0.065735340
## [5,] -2.381863 -0.6446757 -0.01568565 -0.035802870
## [6,] -2.068701 -1.4842053 -0.02687825  0.006586116
```

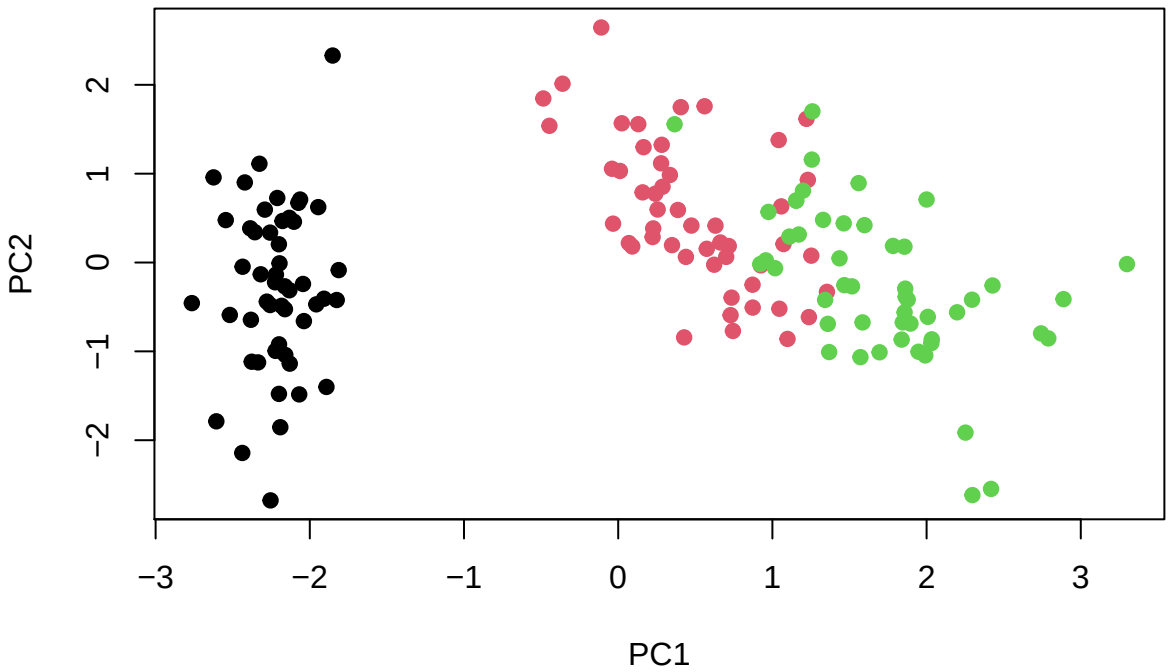
Représentation des projections des données centrées dans le premier plan principale :

```
plot(pca$x[,1],pca$x[,2], pch=19,
     xlab='PC1',
     ylab='PC2')
```




On voit bien que les deux premières composantes dévoilent la présence de deux-trois sous groupes. Ceux là correspondent aux modalités de la cinquième variable du jeu de données :

```
plot(pca$x[,1],pca$x[,2], pch=19,
     xlab='PC1',
     ylab='PC2',
     col=iris[,5])
```




Nous n'approfondissons pas ici la construction d'autres outils graphiques utilisé dans l'ACP (notamment le biplot et le cercle de corrélation) et des autres fonctions disponibles (en particulier `princomp()` et `PCA()`).

# 2. Classification ascendante hiérarchique (CAH)

## 2.1 Rappels

Le but de la classification ascendant hiérarchique (en anglais Hierarchical cluster analysis - HCA) est de répartir les observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$ dans un certain nombre de classes. La méthode se base sur

— une mesure de dissimilarité entre observations, par exemple la distance euclidienne

$$d(\mathbf{x}_i, \mathbf{x}_j) = ||\mathbf{x}_i - \mathbf{x}_j||_2.$$

— une mesure de dissimilarité entre classes, par exemple la distance de Ward

$$diss(C_i, C_j) = \frac{n_i \cdot n_j}{n_i + n_j} d(\mathbf{G}_i, \mathbf{G}_j)$$

où $\mathbf{G}_i$ et $n_i$ sont le centré de gravité et la taille de la classe $C_i$.

L'algorithme est dit ascendant parce que :

— A l'étape initiale chaque observation forme une classe.
— On calcule ensuite la dissimilarité deux à deux entre observations ; les deux observations les plus proches sont réunis dans une classe.
— A l'étape $j$-ème, les dissimilarités entre classes sont calculées deux à deux ; les deux classes les plus proches sont réunis dans une classe.
— On réitère, jusqu'à quand il reste une seule classe.

Pourquoi utilise t-on la distance de Ward ? A chaque fusion l'inertie intra-classes augmente (les classes sont moins homogènes) et l'inertie inter-classes diminue. On cherche donc à fusionner deux classes de façon à que le gain d'inertie intra-classes soit minimal, ou, de façon équivalente, à que la perte d'inertie inter-classes soit minimale. On peut montrer que la perte d'inertie inter-classes obtenue en fusionnant $C_i, C_j$ est égale à la distance de Ward entre $C_i$ et $C_j$.

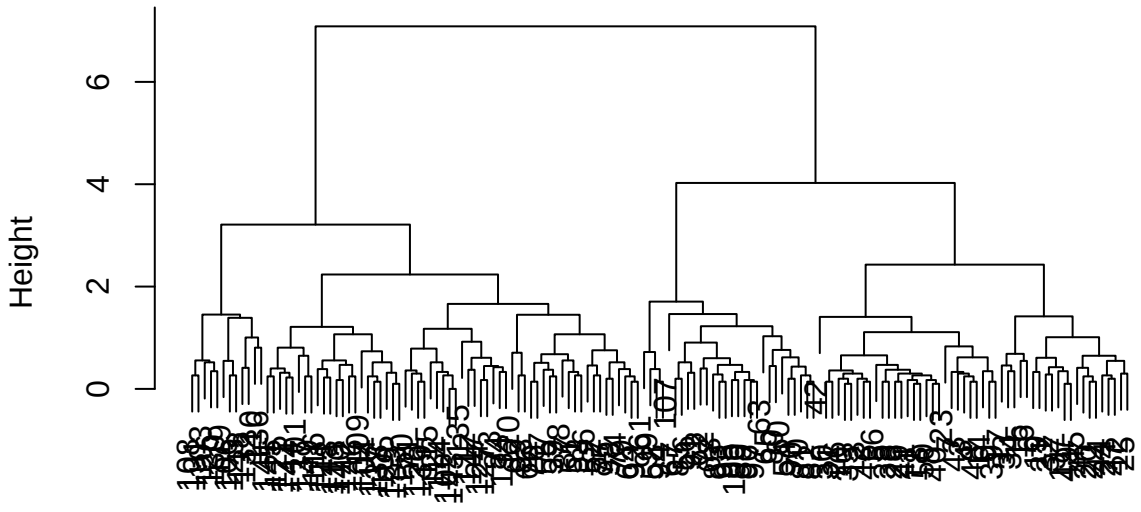
L'historique de l'algorithme, avec toutes les fusions opérées, est représenté par un arbre appelé *dendogramme*. En coupant l'arbre à une hauteur donnée, on obtient un regroupement en un certain nombre de classes (les branches de l'arbre).

Pour plus de détails, voir le poly de E. Lebarbier et T. Mary-Huard *Classification non supervisée* (pages 11-16).

## 2.2 Les fonctions `hclust()` et `cutree()`

On considère les variables quantitatives dans `iris`. On commence par calculer la distance deux à deux des observations ; on donne ensuite le résultat en argument à la fonction `hclust()` :

```
d = dist(iris[-5]) #distance euclidienne par défault
h = hclust(d) #dissimilarité de Ward par défaut
plot(h)
```




La matrice `h$merge` contient l'historique des fusions. Par exemple, dans la première étape les observations `102` e `143` ont fusionnées et à l'étape 23 l'observation `50` a fusionné avec la classe qui s'est formé dans l'étape 2 :

```
h$merge[1,]
```

```
## [1] -102 -143
```

```
h$merge[23,]
```

```
## [1] -50   2
```

Le vecteur `h$height` contient la valeur de la dissimilarité minimale à chaque étape. Par exemple, dans la première étape les deux observations fusionnées étaient à distance 0 et à l'étape 23 les deux classes qui ont fusionnées était à distance 0.17 :

```
h$height[1]
```

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

```
h$height[23]
```

```
## [1] 0.1732051
```

La fonction `cutree()` permet d'obtenir la classification en $k = 1, \ldots, n$ groupes à partir du dendogramme :

```
(z = cutree(h, k=3))
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 2 3 2 3 2 3 3 3 3 2 3 2 3 3 2 3 2 3 2 2
##  [75] 2 2 2 2 2 3 3 3 3 2 3 2 2 2 3 3 3 2 3 3 3 3 2 3 3 2 2 2 2 2 2 3 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2
```

Cette classification arrive à isoler toutes (et seulement) les plantes de l'espèce `setosa` dans la classe 1 et presque toutes les plantes `virginica` dans la classe 2, mais met dans le classe 2 aussi des plantes `versicolor`. Une troisième classe avec les restantes plantes `versicolor` est aussi constituée :

```
table(z,iris[,5])
```

```
##
## z   setosa versicolor virginica
##   1     50          0         0
##   2      0         23        49
##   3      0         27         1
```

# 3. Exercices

## 3.1 ACP et diagonalisation de la matrice de covariance

Dans cet exercice vous serez amenés à faire une ACP en calculant directement les valeurs et vecteurs propres de la matrice de covariance des données. Par défaut, la fonction `prcomp()` fait l'ACP en réalisant la SVD de la matrice des données *centrées*, ce qui est plus efficace que diagonaliser la matrice de covariance des données originales. On travaille sur les données `iris` (variables quantitatives uniquement). On note $\mathbf{X}$ la matrice avec les observations $\mathbf{x}_n^T$ sur les lignes [1] et les variables par colonnes.

1. Calculer la matrice de covariance $\mathbf{S}$ de $\mathbf{X}$ .
2. Calculer valeurs et vecteurs propres de $\mathbf{S}$. Prendre des vecteurs propres qui forment une base orthonormale.
3. Faire l'ACP à l'aide de `prcomp()` avec le paramètres par défaut `center=TRUE, scale.=FALSE`. Comparer les composantes principales et les variances avec les résultats précédents. Comment expliquez-vous la différence observée ?
4. Un vecteur propre est bien évidemment défini à moins d'un scalaire. Contrairement à `prcomp()`, `eigen()` ne rend pas forcement des vecteurs propres qui définissent une rotation : le vérifier en calculant le déterminant de la matrice des vecteurs propres. Changer le signe des vecteurs propres de façon à avoir exactement les vecteurs trouvés par `prcomp()`.
5. Calculer les composantes de chaque observation *centrée* dans la base des vecteurs propres. Indication : dans la base des vecteurs propres $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4$ on a

$$\mathbf{x}_n - \bar{\mathbf{x}} = (\mathbf{u}_1^T(\mathbf{x}_n - \bar{\mathbf{x}}))\mathbf{u}_1 + (\mathbf{u}_2^T(\mathbf{x}_n - \bar{\mathbf{x}}))\mathbf{u}_2 + (\mathbf{u}_3^T(\mathbf{x}_n - \bar{\mathbf{x}}))\mathbf{u}_3 + (\mathbf{u}_4^T(\mathbf{x}_n - \bar{\mathbf{x}}))\mathbf{u}_4$$

   Comparer les résultats avec ceux de `prcomp()`.
6. Vérifier dans cet exemple la formule qui lie la distorsion à la somme des valeurs propres (prendre $M = 2$).

On remarque, en passant, que ne pas centrer les données dans `prcomp()` avec le paramètre `center=FALSE`, a pour résultat de trouver les même valeurs et vecteurs propres qu'on trouve si on diagonalise $\mathbf{X}^T\mathbf{X}$, où $\mathbf{X}$ est la matrice des données originales, ce qui n'est pas la matrice de covariance.

1. Par définition $\mathbf{x}_n$ est un vecteur colonne