

# Régression en grande dimension

Laurent Rouvière

2020-06-14

## Table des matières

<b>Présentation</b>	<b>1</b>
<b>1 Introduction à la grande dimension</b>	<b>1</b>
1.1 Fléau de la dimension pour les plus proches voisins . . . . .	1
1.2 Influence de la dimension dans le modèle linéaire . . . . .	4
1.3 Exercices . . . . .	6
<b>2 Régression sur composantes</b>	<b>8</b>
2.1 Régression sur composantes principales (méthodo) . . . . .	8
2.2 Régression PLS : méthodo . . . . .	12
2.3 Comparaison : PCR vs PLS. . . . .	14

## Présentation

Ce tutoriel présente quelques exercices d'application du cours **Modèle linéaire en grande dimension**. On pourra trouver les documents de cours ainsi que les données utilisées à l'adresse suivante [https://lrouviere.github.io/stat\\_grand\\_dim/](https://lrouviere.github.io/stat_grand_dim/). Des connaissances de base en R sont nécessaires. Le tutoriel se structure en 4 parties :

- **Fléau de la dimension** : identification du problème de la dimension pour le problème de régression ;
- **Régression sur composantes** : présentation des algorithmes **PCR** et **PLS** ;
- **Régression pénalisées** : régularisation à l'aide de pénalités de type **Ridge/Lasso**
- **Modèle additif** : conservation de la structure additive du modèle linéaire mais modélisation non paramétrique des composantes.

## 1 Introduction à la grande dimension

Nous proposons ici d'illustrer le problème de la grande dimension en régression. On commencera par étudier, à l'aide de simulation, ce problème pour l'estimateurs des  $k$  plus proches voisins, puis pour les estimateurs des moindres carrés dans le modèle linéaire. Quelques exercices sont ensuite proposées pour calculer les vitesses de convergence de ces estimateurs dans des modèles simples.

### 1.1 Fléau de la dimension pour les plus proches voisins

La fonction suivante permet de générer un échantillon d'apprentissage et un échantillon test selon le modèle

$$Y = X_1^2 + \dots + X_p^2 + \varepsilon$$

où les  $X_j$  sont uniformes i.i.d de loi uniforme sur  $[0, 1]$  et le bruit  $\varepsilon$  suit une loi  $\mathcal{N}(0, 0.5^2)$ .

```

> simu <- function(napp=300,ntest=500,p=3,graine=1234){
+   set.seed(graine)
+   n <- napp+ntest
+   X <- matrix(runif(n*p),ncol=p)
+   Y <- apply(X^2,1,sum)+rnorm(n,sd=0.5)
+   Yapp <- Y[1:napp]
+   Ytest <- Y[-(1:napp)]
+   Xapp <- data.frame(X[1:napp,])
+   Xtest <- data.frame(X[-(1:napp),])
+   return(list(Xapp=Xapp,Yapp=Yapp,Xtest=Xtest,Ytest=Ytest))
+ }
> df <- simu(napp=300,ntest=500,p=3,graine=1234)

```

La fonction **knn.reg** du package **FNN** permet de construire des estimateurs des  $k$  plus proches voisins en régression. On peut par exemple faire du 3 plus proches voisin avec

```

> library(FNN)
> mod3ppv <- knn.reg(train=df$Xapp,y=df$Yapp,k=3)

```

Parmi toutes les sorties proposées par cette fonction on a notamment

```

> mod3ppv$PRESS
## [1] 98.98178

```

qui renvoie la somme des carrés des erreurs de prévision par validation croisée Leave-One-Out (LOO). On peut ainsi obtenir l'erreur quadratique moyenne par LOO

```

> mod3ppv$PRESS/max(c(nrow(df$Xapp),1))
## [1] 0.3299393

```

1. Construire la fonction **sel.k** qui admet en entrée :

- une grille de valeurs possibles de plus proches voisins (un vecteur).
- une matrice **Xapp** de dimension  $n \times p$  qui contient les valeurs variables explicatives.
- un vecteur **Yapp** de dimension  $n$  qui contient les valeurs de la variable à expliquer

et qui renvoie en sortie la valeur de  $k$  dans la grille qui minimise l'erreur LOO présentée ci-dessus.

```

> sel.k <- function(K_cand=seq(1,50,by=5),Xapp,Yapp){
+   ind <- 1
+   err <- rep(0,length(K_cand))
+   for (k in K_cand){
+     modkppv <- knn.reg(train=Xapp,y=Yapp,k=k)
+     err[ind] <- modkppv$PRESS/max(c(nrow(Xapp),1))
+     ind <- ind+1
+   }
+   return(K_cand[which.min(err)])
+ }

```

Une fois la fonction créée, on peut calculer l'erreur de l'estimateur sélectionné sur un échantillon test avec

```

> k.opt <- sel.k(seq(1,50,by=5),df$Xapp,df$Yapp)
> prev <- knn.reg(train=df$Xapp,y=df$Yapp,test=df$Xtest,k=k.opt)$pred
> mean((prev-df$Ytest)^2)
## [1] 0.283869

```

2. On souhaite comparer les erreurs des règles des  $k$  plus proches voisins en fonction de la dimension. On considère 4 dimensions collectées dans le vecteur DIM et la grille de valeurs de  $k$  suivantes :

```
> DIM <- c(1,5,10,50)
> K_cand <- seq(1,50,by=5)
```

Pour chaque valeur de dimension répéter  $B = 100$  fois :

- simuler un échantillon d'apprentissage de taille 300 et test de taille 500
- calculer la valeur optimale de  $k$  dans **K\_cand** grâce à **sel.k**
- calculer l'erreur de l'estimateur sélectionné sur un échantillon test.

On pourra stocker les résultats dans une matrice de dimension  $B \times 4$ .

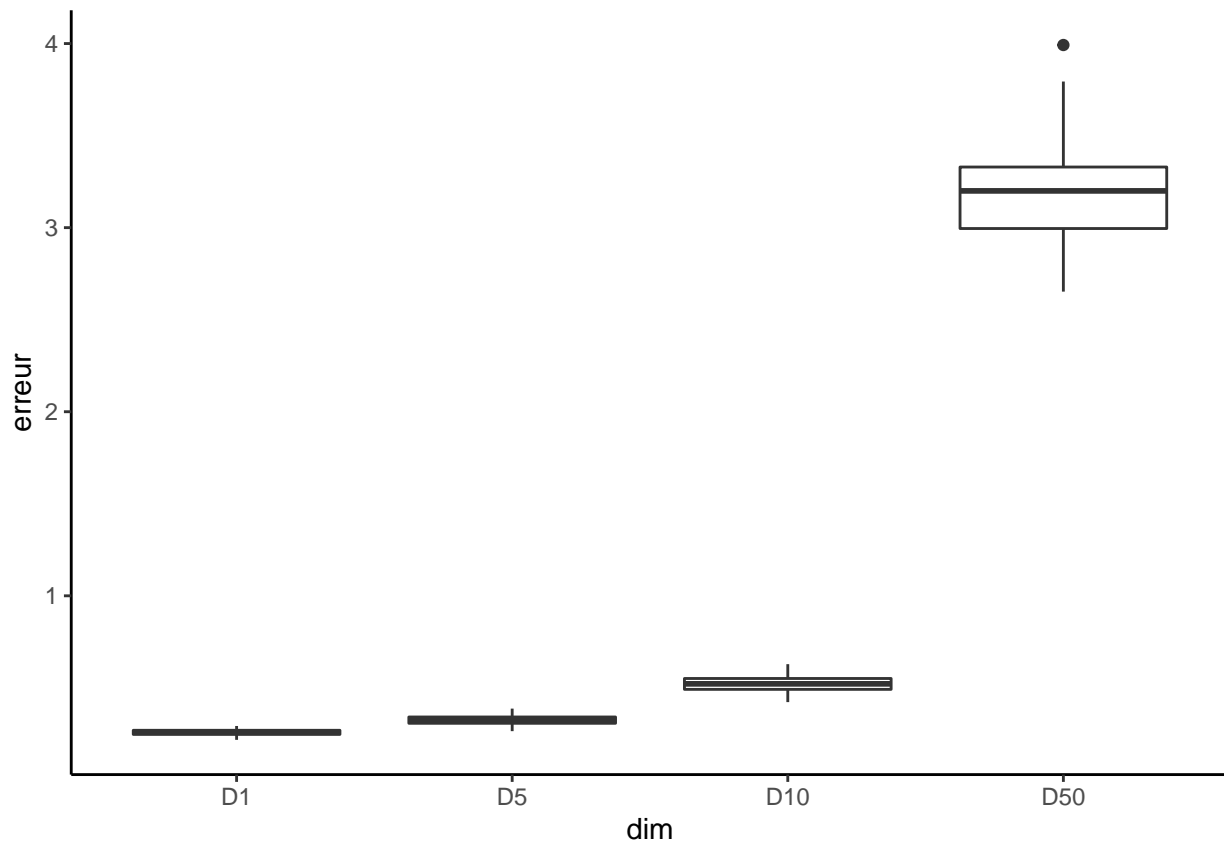
```
> B <- 100
> mat.err <- matrix(0,ncol=length(DIM),nrow=B)
> for (p in 1:length(DIM)){
+   for (i in 1:B){
+     df <- simu(napp=300,ntest=500,p=DIM[p],graine=1234*p+2*i)
+     k.opt <- sel.k(K_cand,df$Xapp,df$Yapp)
+     prev <- knn.reg(train=df$Xapp,y=df$Yapp,test=df$Xtest,k=k.opt)$pred
+     mat.err[i,p] <- mean((prev-df$Ytest)^2)
+   }
+ }
```

3. A l'aide d'indicateurs numériques et de boxplots, comparer la distribution des erreurs en fonction de la dimension.

```
> df <- data.frame(mat.err)
> nom.dim <- paste("D",DIM,sep="")
> names(df) <- nom.dim
```

```
> df %>% summarise_all(mean)
##           D1           D5           D10          D50
## 1 0.258003 0.3243574 0.52247 3.191055
> df %>% summarise_all(var)
##           D1           D5           D10          D50
## 1 0.0002556399 0.0005417109 0.001857967 0.06749414
```

```
> df1 <- pivot_longer(df,cols=everything(),names_to="dim",values_to="erreur")
> df1 <- df1 %>% mutate(dim=fct_relevel(dim,nom.dim))
> ggplot(df1)+aes(x=dim,y=erreur)+geom_boxplot()+theme_classic()
```



### 3. Conclure

Les estimateurs sont moins précis lorsque la dimension augmente. C'est le **fléau de la dimension**.

## 1.2 Influence de la dimension dans le modèle linéaire

En vous basant sur l'exercice précédent, proposer une illustration qui peut mettre en évidence la précision d'estimation dans le modèle linéaire en fonction de la dimension. On pourra par exemple considérer le modèle linéaire suivant

$$Y = X_1 + 0X_2 + \dots + 0X_p + \varepsilon$$

et étudier la performance de l'estimateur MCO du coefficient de  $X_1$  pour différentes valeurs de  $p$ . Par exemple avec  $p$  dans le vecteur

```
> DIM <- c(0,50,100,200)
```

Les données pourront être générées avec la fonction suivante

```
> n <- 250
> p <- 1000
> X <- matrix(runif(n*p), ncol=p)
> simu.lin <- function(X,graine){
+   set.seed(graine)
+   Y <- X[,1] + rnorm(nrow(X), sd=0.5)
+   df <- data.frame(Y,X)
+   return(df)
+ }
```

On s'intéresse à la distribution de  $\hat{\beta}_1$  en fonction de la dimension. Pour ce faire, on calcule un grand nombre d'estimateurs de  $\hat{\beta}_1$  pour différentes valeurs de  $p$ .

```

> B <- 500
> matbeta1 <- matrix(0,nrow=B,ncol=length(DIM))
> for (i in 1:B){
+   dftot <- simu.lin(X,i+1)
+   for (p in 1:length(DIM)){
+     dfp <- dftot[, (1:(2+DIM[p]))]
+     mod <- lm(Y~.,data=dfp)
+     matbeta1[i,p] <- coef(mod)[2]
+   }
+ }

```

On compare, pour chaque dimension considérée, les distributions de  $\hat{\beta}_1$  :

```

> df <- data.frame(matbeta1)
> nom.dim <- paste("D",DIM,sep="")
> names(df) <- nom.dim

```

— en étudiant le biais et la variance

```

> df %>% summarise_all(mean)
##           D0           D50           D100           D200
## 1 0.992891 0.9960811 0.9959025 0.98173
> df %>% summarise_all(var)
##           D0           D50           D100           D200
## 1 0.01266578 0.016072 0.02023046 0.06939837

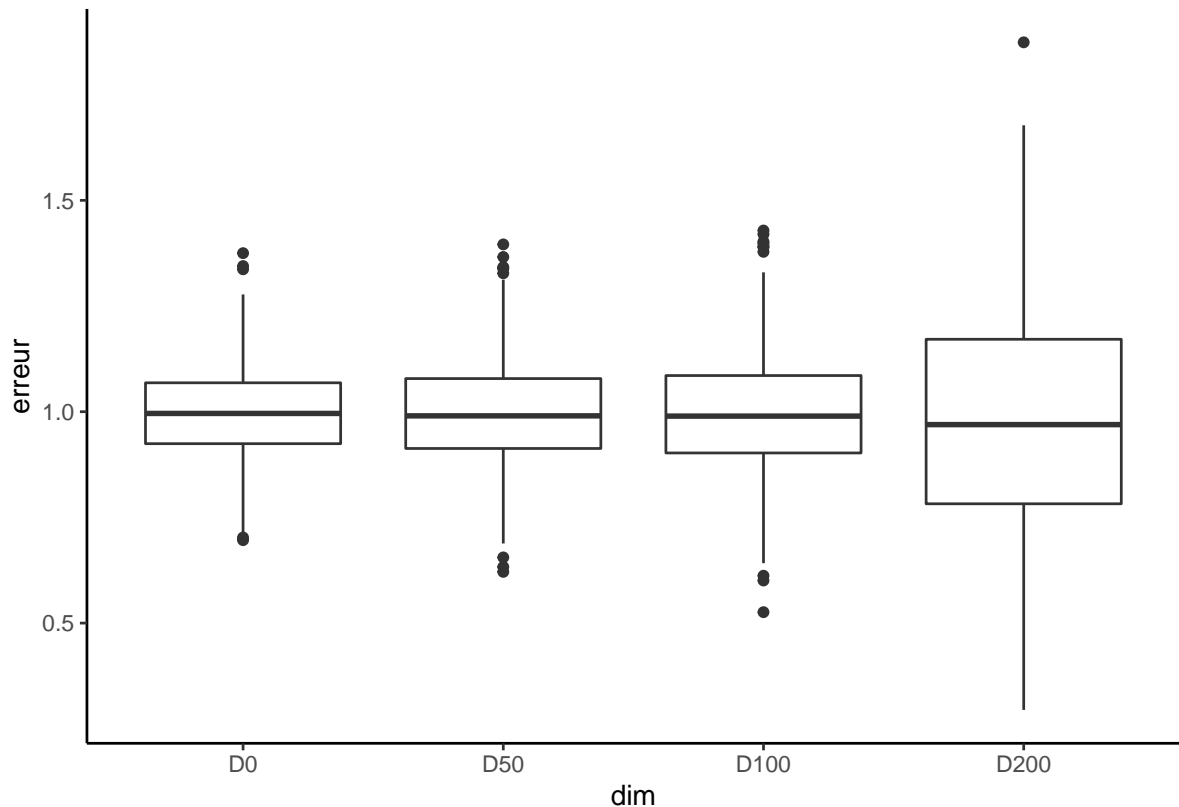
```

— en visualisant la distribution avec un boxplot

```

> df1 <- gather(df,key="dim",value="erreur")
> df1 <- df1 %>% mutate(dim=fct_relevel(dim,nom.dim))
> ggplot(df1)+aes(x=dim,y=erreur)+geom_boxplot()+theme_classic()

```



On retrouve bien que la dimension impacte notamment la variance des estimateurs.

### 1.3 Exercices

**Exercice 1.1** (Distances entre deux points, voir [gir15]. )

Soit  $X^{(1)} = (X_1^{(1)}, \dots, X_p^{(1)})$  et  $X^{(2)} = (X_1^{(2)}, \dots, X_p^{(2)})$  deux variables aléatoires indépendantes de loi uniforme sur l'hypercube  $[0, 1]^p$ . Montrer que

$$\mathbf{E}[\|X^{(1)} - X^{(2)}\|^2] = \frac{p}{6} \quad \text{et} \quad \sigma[\|X^{(1)} - X^{(2)}\|^2] \approx 0.2\sqrt{p}.$$

Soit  $U$  et  $U'$  deux variables aléatoires indépendantes de loi uniforme sur  $[0, 1]$ . On a

$$\mathbf{E}[\|X^{(1)} - X^{(2)}\|^2] = \sum_{k=1}^p \mathbf{E}[(X_k^{(1)} - X_k^{(2)})^2] = p\mathbf{E}[(U - U')^2] = p(2\mathbf{E}[U^2] - 2\mathbf{E}[U]^2) = \frac{p}{6}$$

car  $\mathbf{E}[U^2] = 1/3$  et  $\mathbf{E}[U] = 1/2$ . De même

$$\sigma[\|X^{(1)} - X^{(2)}\|^2] = \sqrt{\sum_{k=1}^p \mathbf{V}[(X_k^{(1)} - X_k^{(2)})^2]} = \sqrt{p\mathbf{V}[(U' - U)^2]} \approx 0.2\sqrt{p}$$

car

$$\mathbf{E}[(U' - U)^4] = \int_0^1 \int_0^1 (x - y)^4 dx dy = \frac{1}{15}$$

et donc  $\mathbf{V}[(U' - U)^2] = 1/15 - 1/36 \approx 0.04$ .

**Exercice 1.2** (Vitesse de convergence pour l'estimateur à noyau).

On considère le modèle de régression

$$Y_i = m(x_i) + \varepsilon_i, \quad i = 1, \dots, n$$

où  $x_1, \dots, x_n \in \mathbb{R}^d$  sont déterministes et  $\varepsilon_1, \dots, \varepsilon_n$  sont des variables i.i.d. d'espérance nulle et de variance  $\sigma^2 < +\infty$ . On désigne par  $\|\cdot\|$  la norme Euclidienne dans  $\mathbb{R}^d$ . On définit l'estimateur localement constant de  $m$  en  $x \in \mathbb{R}^d$  par :

$$\hat{m}(x) = \operatorname{argmin}_{a \in \mathbb{R}} \sum_{i=1}^n (Y_i - a)^2 K\left(\frac{\|x_i - x\|}{h}\right)$$

où  $h > 0$  et pour  $u \in \mathbb{R}$ ,  $K(u) = \mathbf{1}_{[0,1]}(u)$ . On suppose que  $\sum_{i=1}^n K\left(\frac{\|x_i - x\|}{h}\right) > 0$ .

1. Donner la forme explicite de  $\hat{m}(x)$ .

En annulant la dérivée par rapport à  $a$ , on obtient

$$\hat{m}(x) = \frac{\sum_{i=1}^n Y_i K\left(\frac{\|x_i - x\|}{h}\right)}{\sum_{i=1}^n K\left(\frac{\|x_i - x\|}{h}\right)}.$$

2. Montrer que

$$\mathbf{V}[\hat{m}(x)] = \frac{\sigma^2}{\sum_{i=1}^n K\left(\frac{\|x_i - x\|}{h}\right)}$$

et

$$\mathbf{E}[\hat{m}(x)] - m(x) = \frac{\sum_{i=1}^n (m(x_i) - m(x)) K\left(\frac{\|x_i - x\|}{h}\right)}{\sum_{i=1}^n K\left(\frac{\|x_i - x\|}{h}\right)}.$$

Ces propriétés se déduisent directement en remarquant que  $\mathbf{V}[Y_i] = \sigma^2$  et  $\mathbf{E}[Y_i] = m(x_i)$ .

3. On suppose maintenant que  $m$  est Lipschitzienne de constante  $L$ , c'est-à-dire que  $\forall (x_1, x_2) \in \mathbb{R}^d \times \mathbb{R}^d$

$$|m(x_1) - m(x_2)| \leq L\|x_1 - x_2\|.$$

Montrer que

$$|\text{biais}[\hat{m}(x)]| \leq Lh.$$

On a  $|m(x_i) - m(x)| \leq L\|x_i - x\|$ . Or

$$K\left(\frac{\|x_i - x\|}{h}\right)$$

est non nul si et seulement si  $\|x_i - x\| \leq h$ . Donc pour tout  $i = 1, \dots, n$

$$L\|x_i - x\|K\left(\frac{\|x_i - x\|}{h}\right) \leq LhK\left(\frac{\|x_i - x\|}{h}\right).$$

D'où le résultat.

4. On suppose de plus qu'il existe une constante  $C_1$  telle que

$$C_1 \leq \frac{\sum_{i=1}^n \mathbf{1}_{B_h}(x_i - x)}{n\text{Vol}(B_h)},$$

où  $B_h = \{u \in \mathbb{R}^d : \|u\| \leq h\}$  est la boule de rayon  $h$  dans  $\mathbb{R}^d$  et  $\text{Vol}(A)$  désigne le volume d'un ensemble  $A \subset \mathbb{R}^d$ . Montrer que

$$\mathbf{V}[\hat{m}(x)] \leq \frac{C_2\sigma^2}{nh^d},$$

où  $C_2$  est une constante dépendant de  $C_1$  et  $d$  à préciser.

On a

$$\mathbf{V}[\hat{m}(x)] = \frac{\sigma^2}{\sum_{i=1}^n K\left(\frac{\|x_i - x\|}{h}\right)} = \frac{\sigma^2}{\sum_{i=1}^n \mathbf{1}_{B_h}(x_i - x)}.$$

Or

$$\sum_{i=1}^n \mathbf{1}_{B_h}(x_i - x) \geq C_1 n \text{Vol}(B_h) \geq C_1 \gamma_d n h^d$$

où  $\gamma_d$  désigne le volume de la boule unité en dimension  $d$ . On a donc

$$\mathbf{V}[\hat{m}(x)] \leq \frac{\sigma^2}{C_1 \gamma_d n h^d}.$$

5. Déduire des questions précédentes un majorant de l'erreur quadratique moyenne de  $\hat{m}(x)$ .

On déduit

$$\mathbf{E}[(\hat{m}(x) - m(x))^2] \leq L^2 h^2 + \frac{C_2 \sigma^2}{n h^d}.$$

6. Calculer  $h_{\text{opt}}$  la valeur de  $h$  qui minimise ce majorant. Que vaut ce majorant lorsque  $h = h_{\text{opt}}$ ? Comment varie cette vitesse lorsque  $d$  augmente? Interpréter.

Soit  $M(h)$  le majorant. On a

$$M(h)' = 2hL^2 - \frac{C_2 \sigma^2 d}{n} h^{-d-1}.$$

La dérivée s'annule pour

$$h_{\text{opt}} = \frac{2L^2}{C_2 \sigma^2 d} n^{-\frac{1}{d+2}}.$$

Lorsque  $h = h_{\text{opt}}$  l'erreur quadratique vérifie

$$\mathbf{E}[(\hat{m}(x) - m(x))^2] = O\left(n^{-\frac{2}{d+2}}\right).$$

## 2 Régression sur composantes

### 2.1 Régression sur composantes principales (méthodo)

On considère le jeu de données **Hitters** dans lequel on souhaite expliquer la variable **Salary** par les autres variables du jeu de données. On supprime les individus qui possèdent des données manquantes.

```
> library(ISLR)
> Hitters <- na.omit(Hitters)
```

1. Parmi les variables explicatives, certaines sont qualitatives. Expliquer comment, à l'aide de la fonction **model.matrix** on peut utiliser ces variables dans un modèle linéaire. On appellera **X** la matrice des variables explicatives construites avec cette variable.

*Comme pour le modèle linéaire, on utilise des contraintes identifiantes. Cela revient à prendre une modalité de référence et à coder les autres modalités par 0-1.*

```
> X <- model.matrix(Salary~.,data=Hitters)[-1]
```

2. Calculer la matrice **Xcr** qui correspond à la matrice **X** centrée réduite. On pourra utiliser la fonction **scale**.

```
> Xcr <- scale(X)
> Xbar <- apply(X,2,mean)
> stdX <- apply(X,2,sd)
```

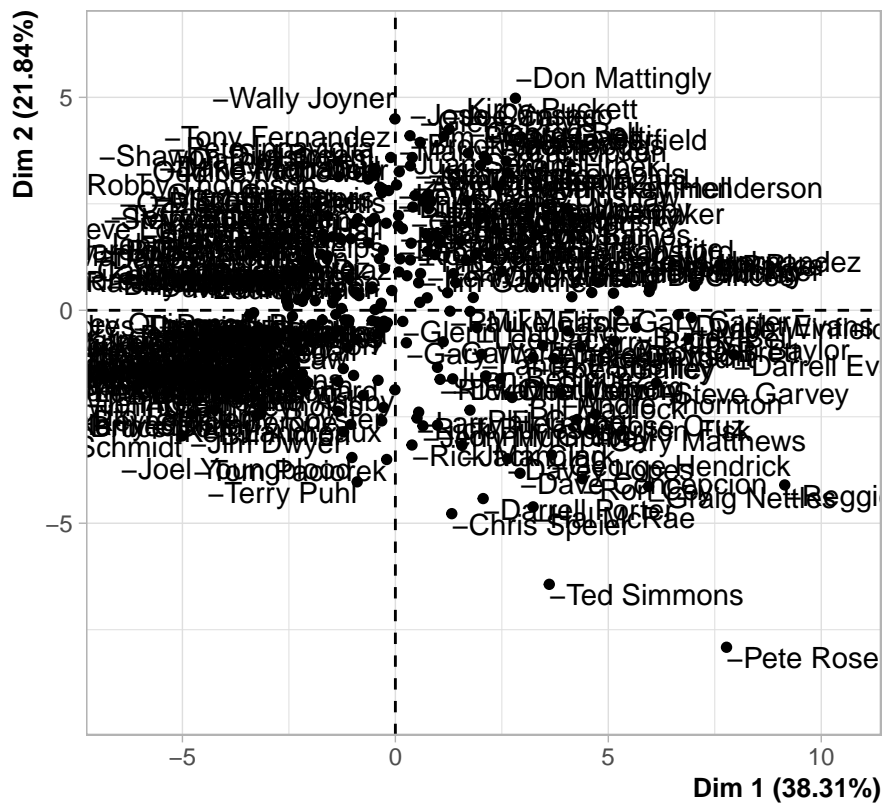
3. A l'aide de la fonction **PCA** du package **FactoMineR**, effectuer l'ACP du tableau **Xcr** avec l'option **scale.unit=FALSE**.

*On utilise ici **scale.unit=FALSE** car les données sont déjà centrées-réduites. Ça nous permet de contrôler cette étape.*

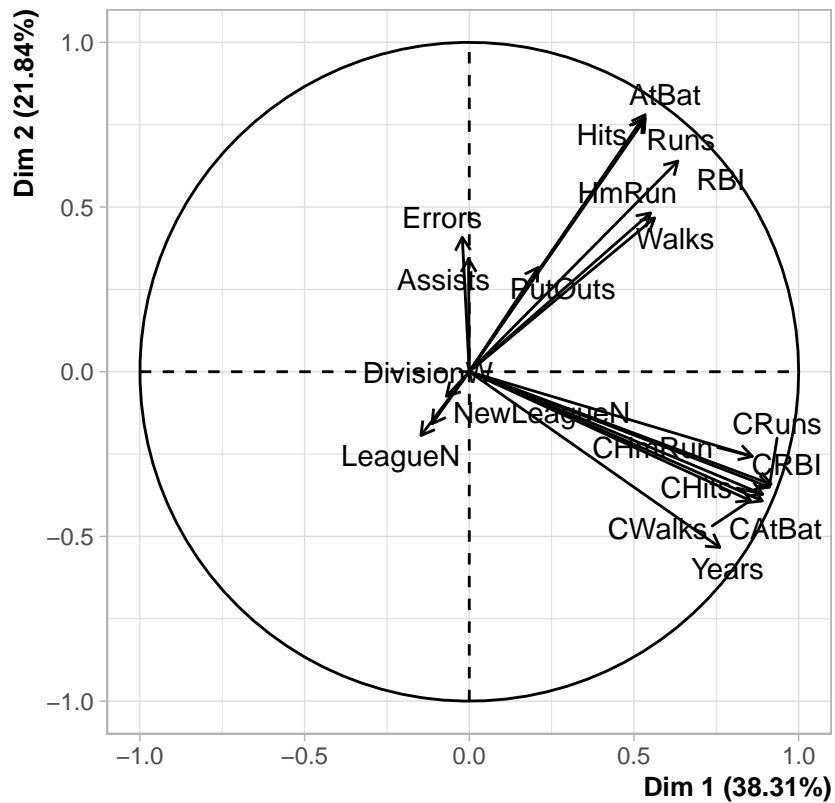
```
> library(FactoMineR)
> acp.hit <- PCA(Xcr,scale.unit=FALSE,graph=TRUE)
```



PCA graph of individuals



PCA graph of variables



4. Récupérer les coordonnées des individus sur les 5 premiers axes de l'ACP (variables  $Z$  dans le cours).

```
> Z <- acp.hit$ind$coord
```

5. Effectuer la régression linéaire sur les 5 premières composantes principales et calculer les estimateurs des MCO ( $\hat{\theta}_k, k = 1, \dots, 5$  dans le cours).

```
> donnees <- cbind.data.frame(Z,Salary=Hitters$Salary)
> mod <- lm(Salary~.,data=donnees)
> theta <- coef(mod)
> theta
## (Intercept)      Dim.1      Dim.2      Dim.3      Dim.4      Dim.5
##  535.92588  106.57139  21.64469  24.34057  37.05637 -58.52540
```

*Remarque : on peut aussi tout faire "à la main" (sans utiliser PCA)*

```
> acp.main <- eigen(t(Xcr)%*%Xcr)
> U <- acp.main$eigenvectors
> CC <- Xcr%*%(-U[,1:5])
> D <- cbind.data.frame(CC,Salary=Hitters$Salary)
> modS <- lm(Salary~.,data=D)
> coefS <- modS$coefficients
> coef(modS)
## (Intercept)      `1`      `2`      `3`      `4`      `5`
##  535.92588  106.57139  21.64469  24.34057  37.05637 -58.52540
```

6. En déduire les estimateurs dans l'espace des données initiales pour les données centrées réduites, puis pour les données brutes. On pourra récupérer les vecteurs propres de l'ACP ( $u_k$  dans le cours) dans la sortie **svd** de la fonction **PCA**

— Pour les données centrées-réduites, les coefficients s'obtiennent avec les formules vues en cours

$$\hat{\beta}_0 = \bar{y} \quad \text{et} \quad \hat{\beta}_j = \hat{\theta}' v_j.$$

```
> U <- acp.hit$svd$V
> V <- t(U)
> beta0.cr <- mean(Hitters$Salary)
> beta.cr <- as.vector(theta[2:6])%*%V
> beta.cr
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]
## [1,] 28.76604 30.44702 25.8445 33.00088 33.81997 35.08779 22.35103 29.01477
##           [,9]      [,10]      [,11]      [,12]      [,13]      [,14]      [,15]      [,16]
## [1,] 29.78584 30.00201 32.06912 31.11231 31.48735 19.439 -63.20387 17.36044
##           [,17]      [,18]      [,19]
## [1,] -5.523264 -6.044002 21.74267
```

— Pour les données brutes, on utilise les formules :

$$\hat{\beta}_0 = \bar{y} - \sum_{j=1}^p \hat{\theta}' v_j \frac{\bar{x}_j}{\sigma_{x_j}} \quad \text{et} \quad \hat{\beta}_j = \frac{\hat{\theta}' v_j}{\sigma_{x_j}}, j = 1, \dots, p.$$

```
> beta0 <- beta0.cr - sum(beta.cr*Xbar/stdX)
> beta <- beta.cr/stdX
> beta0
## [1] -58.32022
> beta
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]
## [1,] 0.1952793 0.6747214 2.95126 1.292134 1.306662 1.615605 4.662667 0.01268914
```

```
##           [,9]      [,10]      [,11]      [,12]      [,13]      [,14]      [,15]
## [1,] 0.04595165 0.3649987 0.09682748 0.09621344 0.119245 38.86728 -126.19
##           [,16]      [,17]      [,18]      [,19]
## [1,] 0.06201606 -0.03807032 -0.9148466 43.51629
```

7. Retrouver les estimateurs dans l'espace des données initiales pour les données centrées réduites à l'aide de la fonction `pcr` du package `pls`.

```
> library(pls)
> pcr.fit <- pcr(Salary~.,data=Hitters,scale=TRUE,ncomp=19)
> coefficients(pcr.fit,ncomp=5)
## , , 5 comps
##
##           Salary
## AtBat      28.766042
## Hits       30.447021
## HmRun       25.844498
## Runs        33.000876
## RBI         33.819966
## Walks       35.087794
## Years       22.351033
## CAtBat      29.014768
## CHits       29.785842
## CHmRun      30.002014
## CRuns       32.069124
## CRBI        31.112315
## CWalks      31.487349
## LeagueN     19.438996
## DivisionW   -63.203872
## PutOuts     17.360440
## Assists     -5.523264
## Errors      -6.044002
## NewLeagueN  21.742668
```

8. On considère les individus suivants

```
> df.new <- Hitters[c(1,100,80),]
```

Calculer de 3 façons différentes les valeurs de salaire prédites par la régression sur 5 composantes principales.

— Approche classique : on utilise `predict.pcr` :

```
> predict(pcr.fit,newdata=df.new,ncomp=5)
## , , 5 comps
##
##           Salary
## -Alan Ashby    495.0068
## -Hubie Brooks 577.9581
## -George Bell  822.0296
```

— On considère les valeurs centrées réduites et on utilise :

$$\hat{y} = \bar{y} + \hat{\theta}' v_1 \tilde{x}_1 + \cdots + \hat{\theta}' v_p \tilde{x}_p$$

```
> t(as.matrix(coefficients(pcr.fit,ncomp=5))) %*%
+   t(as.matrix(Xcr[c(1,100,80),]))+mean(Hitters$Salary)
##           -Alan Ashby -Hubie Brooks -George Bell
```

```
## [1,] 495.0068 577.9581 822.0296
> #ou
> beta0.cr+beta.cr%*%t(as.matrix(Xcr[c(1,100,80),]))
## -Alan Ashby -Hubie Brooks -George Bell
## [1,] 495.0068 577.9581 822.0296
```

— On considère les données brutes et on utilise :

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p$$

```
> beta0+beta %*% t(as.matrix(X[c(1,100,80),]))
## -Alan Ashby -Hubie Brooks -George Bell
## [1,] 495.0068 577.9581 822.0296
```

## 2.2 Régression PLS : méthode

On considère les mêmes données que précédemment.

1. A l'aide du vecteur  $Y$  (*Salary*) et de la matrice des  $X$  centrées réduites calculées dans l'exercice précédent, calculer la première composante **PLS**  $Z_1$ .

```
> Y <- as.vector(Hitters$Salary)
> w1 <- t(Xcr)%*%Y
> w1
## [1,]
## AtBat 46659.1995
## Hits 51848.3247
## HmRun 40543.5500
## Runs 49624.3823
## RBI 53122.7240
## Walks 52462.0450
## Years 47354.8899
## CAtBat 62185.5603
## CHits 64877.3193
## CHmRun 62043.1671
## CRuns 66504.6198
## CRBI 67011.4288
## CWalks 57893.5821
## LeagueN -1688.0134
## DivisionW -22753.8726
## PutOuts 35514.7030
## Assists 3006.3756
## Errors -638.3256
## NewLeagueN -335.0136
> Z1 <- Xcr%*%w1
```

2. En déduire le coefficient associé à cette première composante en considérant le modèle

$$Y = \alpha_1 Z_1 + \varepsilon.$$

```
> df <- data.frame(Z1,Y)
> mod1 <- lm(Y~Z1-1,data=df)
> alpha1 <- coef(mod1)
> alpha1
## Z1
## 0.0005367014
```

3. En déduire les coefficients en fonction des variables initiales (centrées réduites) de la régression PLS à une composante

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon.$$

```
> alpha1*w1
##           [,1]
## AtBat      25.0420570
## Hits       27.8270677
## HmRun      21.7597795
## Runs       26.6334747
## RBI        28.5110396
## Walks      28.1564522
## Years      25.4154350
## CAtBat     33.3750764
## CHits      34.8197471
## CHmRun     33.2986538
## CRuns      35.6931216
## CRBI       35.9651267
## CWalks     31.0715657
## LeagueN    -0.9059591
## DivisionW  -12.2120349
## PutOuts    19.0607903
## Assists     1.6135259
## Errors     -0.3425902
## NewLeagueN -0.1798022
```

4. Retrouver ces coefficients en utilisant la fonction `pls`.

```
> pls.fit <- pls(Salary~.,data=Hitters,scale=TRUE)
> coefficients(pls.fit,ncomp = 1)
## , , 1 comps
##
##           Salary
## AtBat      25.0420570
## Hits       27.8270677
## HmRun      21.7597795
## Runs       26.6334747
## RBI        28.5110396
## Walks      28.1564522
## Years      25.4154350
## CAtBat     33.3750764
## CHits      34.8197471
## CHmRun     33.2986538
## CRuns      35.6931216
## CRBI       35.9651267
## CWalks     31.0715657
## LeagueN    -0.9059591
## DivisionW  -12.2120349
## PutOuts    19.0607903
## Assists     1.6135259
## Errors     -0.3425902
## NewLeagueN -0.1798022
```

## 2.3 Comparaison : PCR vs PLS.

1. Séparer le jeu de données en un échantillon d'apprentissage de taille 200 et un échantillon test de taille 63.

```
> set.seed(1234)
> perm <- sample(nrow(Hitters))
> dapp <- Hitters[perm[1:200],]
> dtest <- Hitters[perm[201:nrow(Hitters)],]
```

2. Avec les données d'apprentissage uniquement construire les régressions PCR et PLS. On choisira les nombres de composantes par validation croisée.

```
> choix.pcr <- pcr(Salary~.,data=dapp,validation="CV")
> ncomp.pcr <- which.min(choix.pcr$validation$PRESS)
```

```
> choix.pls <- pls(Salary~.,data=dapp,validation="CV")
> ncomp.pls <- which.min(choix.pls$validation$PRESS)
```

3. Comparer les deux méthodes en utilisant l'échantillon de validation. On pourra également utiliser un modèle linéaire classique.

```
> mod.lin <- lm(Salary~.,data=dapp)
```

```
> prev <- data.frame(
+   lin=predict(mod.lin,newdata=dtest),
+   pcr=as.vector(predict(choix.pcr,newdata = dtest,ncomp=ncomp.pcr)),
+   pls=as.vector(predict(choix.pls,newdata = dtest,ncomp=ncomp.pls)),
+   obs=dtest$Salary
+ )
```

```
> prev %>% summarize_at(1:3,~(mean((.-obs)^2))) %>% sqrt()
##      lin      pcr      pls
## 1 334.8819 348.3943 342.7771
```

4. Comparer ces méthodes en faisant une validation croisée 10 blocs.

*On définit d'abord les 10 blocs pour la validation croisée.*

```
> set.seed(1234)
> bloc <- sample(1:10,nrow(Hitters),replace=TRUE)
> table(bloc)
## bloc
##  1  2  3  4  5  6  7  8  9 10
## 19 22 31 29 28 39 19 26 25 25
```

```
> set.seed(4321)
> prev <- data.frame(matrix(0,nrow=nrow(Hitters),ncol=3))
> names(prev) <- c("lin","PCR","PLS")
> for (k in 1:10){
+   # print(k)
+   ind.test <- bloc==k
+   dapp <- Hitters[!ind.test,]
+   dtest <- Hitters[ind.test,]
+   choix.pcr <- pcr(Salary~.,data=dapp,validation="CV")
+   ncomp.pcr <- which.min(choix.pcr$validation$PRESS)
+   choix.pls <- pls(Salary~.,data=dapp,validation="CV")
+   ncomp.pls <- which.min(choix.pls$validation$PRESS)
+   mod.lin <- lm(Salary~.,data=dapp)
+   prev[ind.test,] <- data.frame(
+     lin=predict(mod.lin,newdata=dtest),
+     PCR=as.vector(predict(choix.pcr,newdata = dtest,ncomp=ncomp.pcr)),
```

```
+ PLS=as.vector(predict(choix.pls,newdata = dtest,ncomp=ncomp.pls)))
+ }

> prev %>% mutate(obs=Hitters$Salary) %>% summarize_at(1:3,~(mean((.-obs)^2))) %>% sqrt()
##          lin          PCR          PLS
## 1 340.0631 343.8019 350.6712
```

On compare à un modèle qui prédit toujours la moyenne :

```
> var(Hitters$Salary) %>% sqrt()
## [1] 451.1187
```

On peut retenir l'analyse en considérant toutes les interactions d'ordre 2 :

```
> set.seed(54321)
> prev1 <- data.frame(matrix(0,nrow=nrow(Hitters),ncol=3))
> names(prev1) <- c("lin","PCR","PLS")
> for (k in 1:10){
+ # print(k)
+ ind.test <- bloc==k
+ dapp <- Hitters[!ind.test,]
+ dtest <- Hitters[ind.test,]
+ choix.pcr <- pcr(Salary~.^2,data=dapp,validation="CV")
+ ncomp.pcr <- which.min(choix.pcr$validation$PRESS)
+ choix.pls <- plsr(Salary~.^2,data=dapp,validation="CV")
+ ncomp.pls <- which.min(choix.pls$validation$PRESS)
+ mod.lin <- lm(Salary~.^2,data=dapp)
+ prev1[ind.test,] <- data.frame(
+ lin=predict(mod.lin,newdata=dtest),
+ PCR=as.vector(predict(choix.pcr,newdata = dtest,ncomp=ncomp.pcr)),
+ PLS=as.vector(predict(choix.pls,newdata = dtest,ncomp=ncomp.pls)))
+ }
```

On obtient les performances suivantes :

```
> prev1 %>% mutate(obs=Hitters$Salary) %>% summarize_at(1:3,~(mean((.-obs)^2))) %>% sqrt()
##          lin          PCR          PLS
## 1 1494.847 330.0474 349.1116
```

On mesure bien l'intérêt de réduire la dimension dans ce nouveau contexte.