

Régression en grande dimension

Laurent Rouvière

2020-06-15

Table des matières

Présentation	1
1 Introduction à la grande dimension	1
1.1 Fléau de la dimension pour les plus proches voisins	2
1.2 Influence de la dimension dans le modèle linéaire	4
1.3 Exercices	6
2 Régression sur composantes	8
2.1 Régression sur composantes principales (méthodo)	8
2.2 Régression PLS : méthodo	12
2.3 Comparaison : PCR vs PLS.	14
3 Régressions pénalisées (ou sous contraintes)	15
3.1 Ridge et lasso avec glmnet	16
3.2 Reconstruction d'un signal	21
3.3 Régression logistique pénalisée	28
4 Modèle additif	32
4.1 Pseudo backfitting	32
4.2 Modèle GAM	33
4.3 Régression logistique additive	40

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/. 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** ;
- **Régressions 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 ε 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 $B \times 4$.

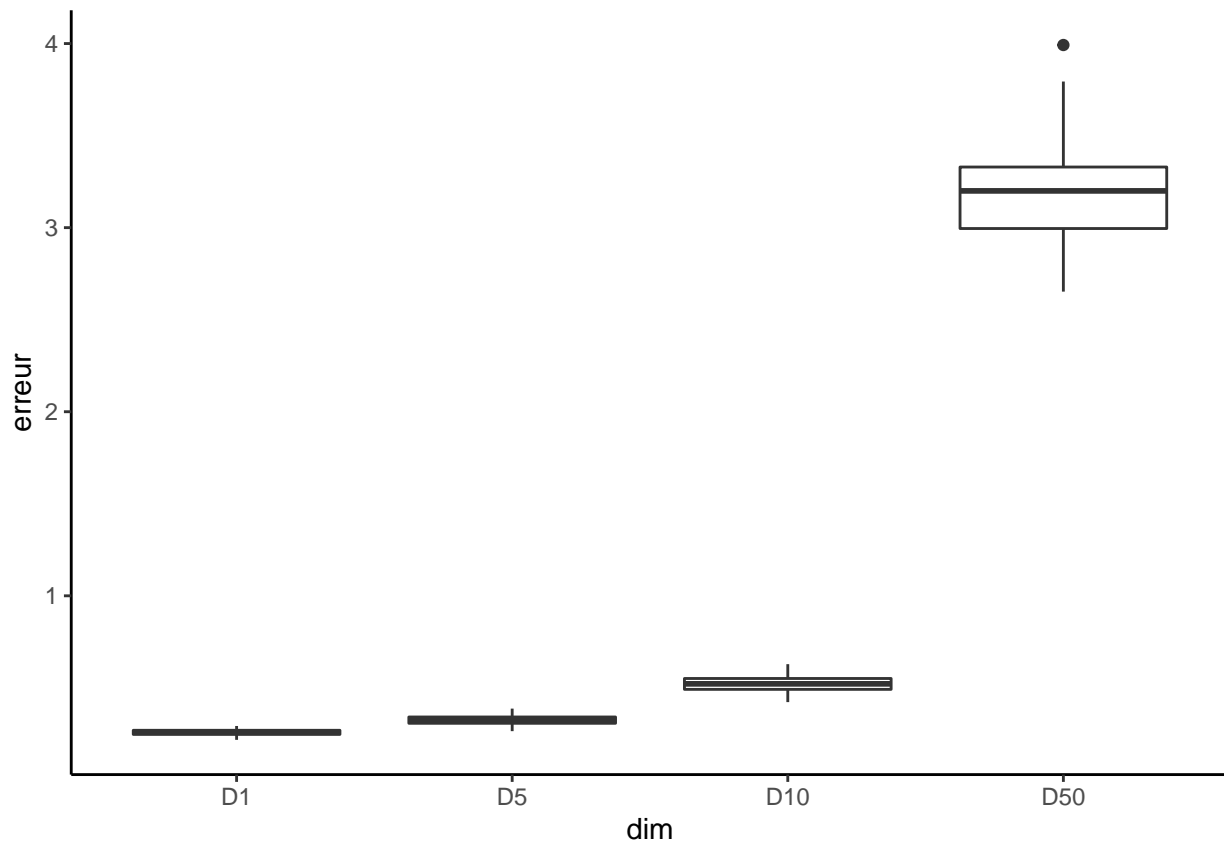
```
> B <- 100
> mat.err <- matrix(0,ncol=length(DIM),nrow=B)
> for (p in 1:length(DIM)){
+   for (i in 1:B){
+     df <- simu(napp=300,ntest=500,p=DIM[p],graine=1234*p+2*i)
+     k.opt <- sel.k(K_cand,df$Xapp,df$Yapp)
+     prev <- knn.reg(train=df$Xapp,y=df$Yapp,test=df$Xtest,k=k.opt)$pred
+     mat.err[i,p]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

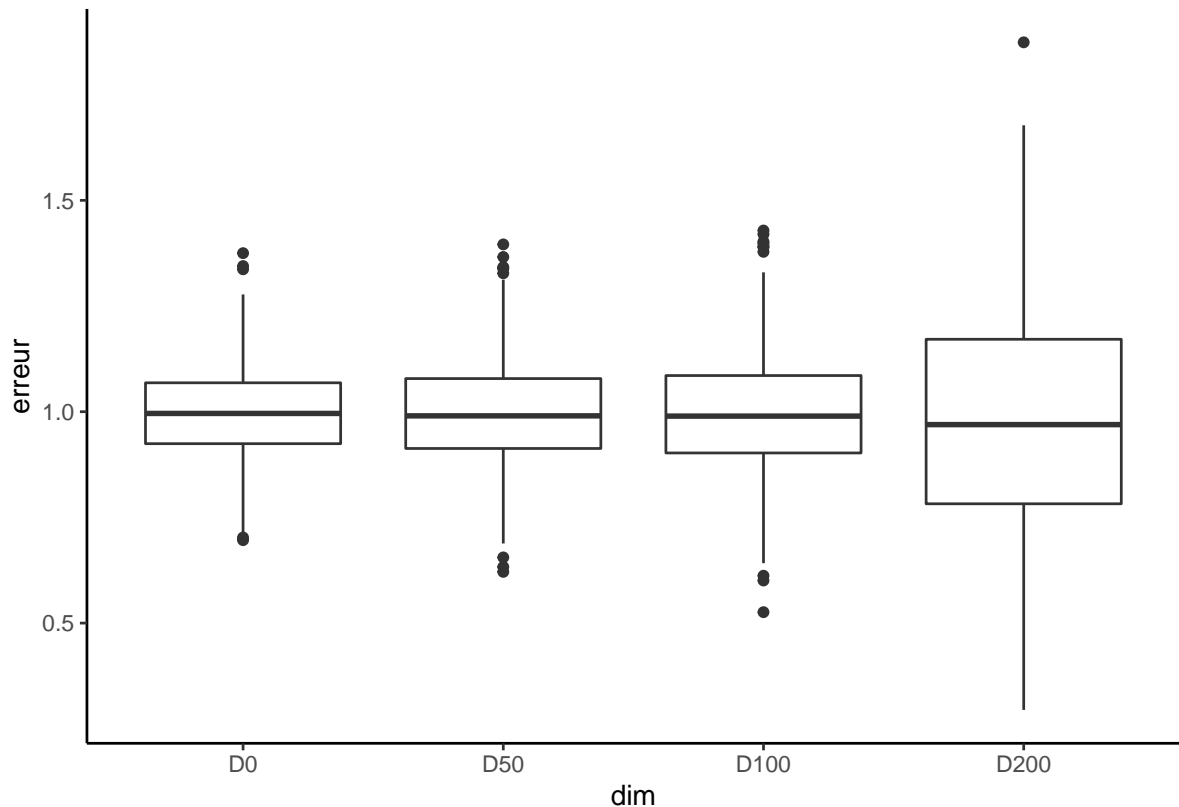
```

— 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 γ_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_{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**.

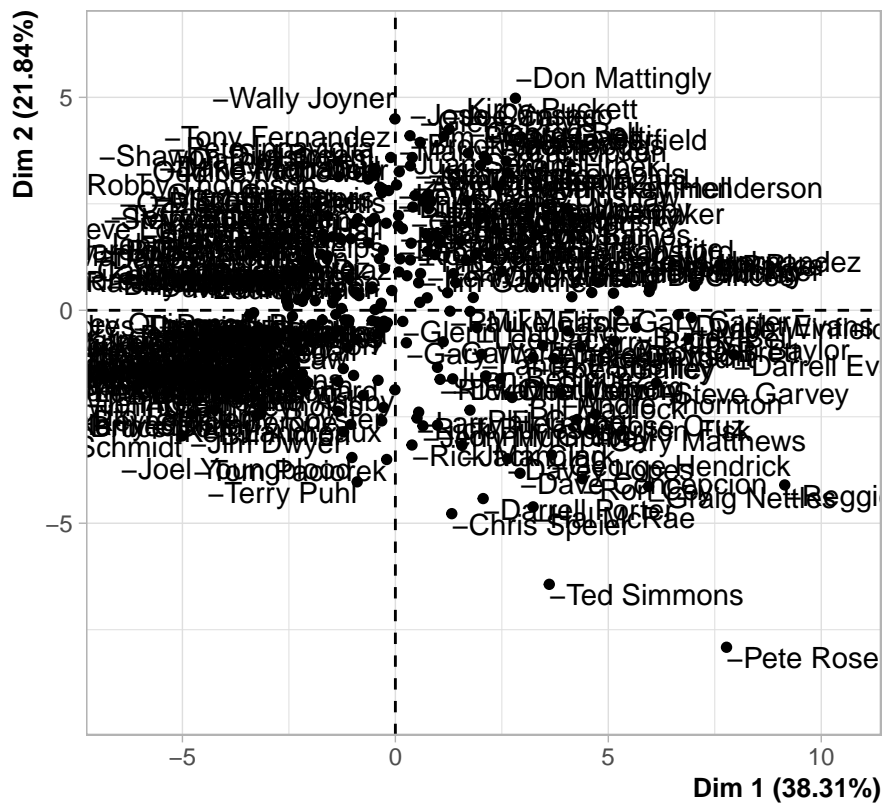
```
> Xcr <- scale(X)
> Xbar <- apply(X,2,mean)
> stdX <- apply(X,2,sd)
```

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

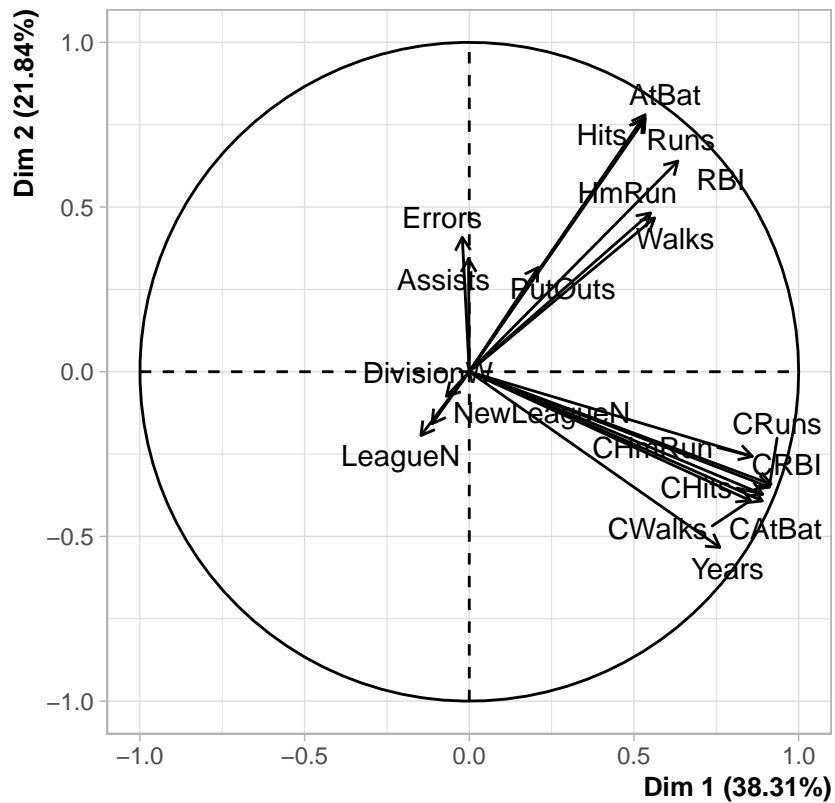
*On utilise ici **scale.unit=FALSE** car les données sont déjà centrées-réduites. Ça nous permet de contrôler cette étape.*

```
> 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 `plsr`.

```
> pls.fit <- plsr(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 retenter 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 <- pls(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.

3 Régressions pénalisées (ou sous contraintes)

Nous considérons toujours le modèle linéaire

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_d X_d + \varepsilon$$

Lorsque d est grand ou que les variables sont linéairement dépendantes, les estimateurs des moindres carrées peuvent être mis en défaut. Les méthodes pénalisées ou sous contraintes consistent alors à restreindre l'espace sur lequel on minimise ce critère. On va alors chercher le vecteur β qui minimise

$$\sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^d x_{ij} \beta_j \right)^2 \quad \text{sous la contrainte} \quad \sum_{j=1}^d \beta_j^2 \leq t$$

ou de façon équivalente (dans le sens où il existe une équivalence entre t et λ)

$$\sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^d x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^d \beta_j^2.$$

Les estimateurs obtenus sont les estimateurs **ridge**. Les estimateurs **lasso** s'obtiennent en remplaçant la contrainte ou la pénalité par une norme 1 ($\sum_{j=1}^d |\beta_j|$). Nous présentons dans cette partie les étapes principales qui permettent de faire ce type de régression avec **R**. Le package le plus souvent utilisé est **glmnet**.

3.1 Ridge et lasso avec glmnet

On considère le jeu de données `ozone.txt` où on cherche à expliquer la concentration maximale en ozone relevée sur une journée par d'autres variables essentiellement météorologiques.

```
> ozone <- read.table("data/ozone.txt")
> head(ozone)
##           maxO3   T9  T12  T15 Ne9 Ne12 Ne15      Vx9      Vx12      Vx15 maxO3v
## 20010601      87 15.6 18.5 18.4   4   4   8  0.6946 -1.7101 -0.6946      84
## 20010602      82 17.0 18.4 17.7   5   5   7 -4.3301 -4.0000 -3.0000      87
## 20010603      92 15.3 17.6 19.5   2   5   4  2.9544  1.8794  0.5209      82
## 20010604     114 16.2 19.7 22.5   1   1   0  0.9848  0.3473 -0.1736      92
## 20010605      94 17.4 20.5 20.4   8   8   7 -0.5000 -2.9544 -4.3301     114
## 20010606      80 17.7 19.8 18.3   6   6   7 -5.6382 -5.0000 -6.0000      94
##           vent pluie
## 20010601  Nord   Sec
## 20010602  Nord   Sec
## 20010603   Est   Sec
## 20010604  Nord   Sec
## 20010605 Ouest   Sec
## 20010606 Ouest Pluie
```

Contrairement à la plupart des autres package **R** qui permettent de faire de l'apprentissage, le package **glmnet** n'autorise pas l'utilisation de **formules** : il faut spécifier explicitement la matrice des X et le vecteur des Y . On peut obtenir la matrice des X et notamment le codage des variables qualitatives avec la fonction `model.matrix` :

```
> ozone.X <- model.matrix(maxO3~.,data=ozone)[,-1]
> ozone.Y <- ozone$maxO3
```

1. Charger le package **glmnet** et à l'aide de la fonction **glmnet** calculer les estimateurs **ridge** et **lasso**.

```
> library(glmnet)
> mod.R <- glmnet(ozone.X,ozone.Y,alpha=0)
> mod.L <- glmnet(ozone.X,ozone.Y,alpha=1)
```

2. Analyser les sorties qui se trouvent dans les arguments **lambda** et **beta** de **glmnet**.

La fonction **glmnet** calcule tous les estimateurs pour une grille de valeurs de **lambda** spécifiée ici :

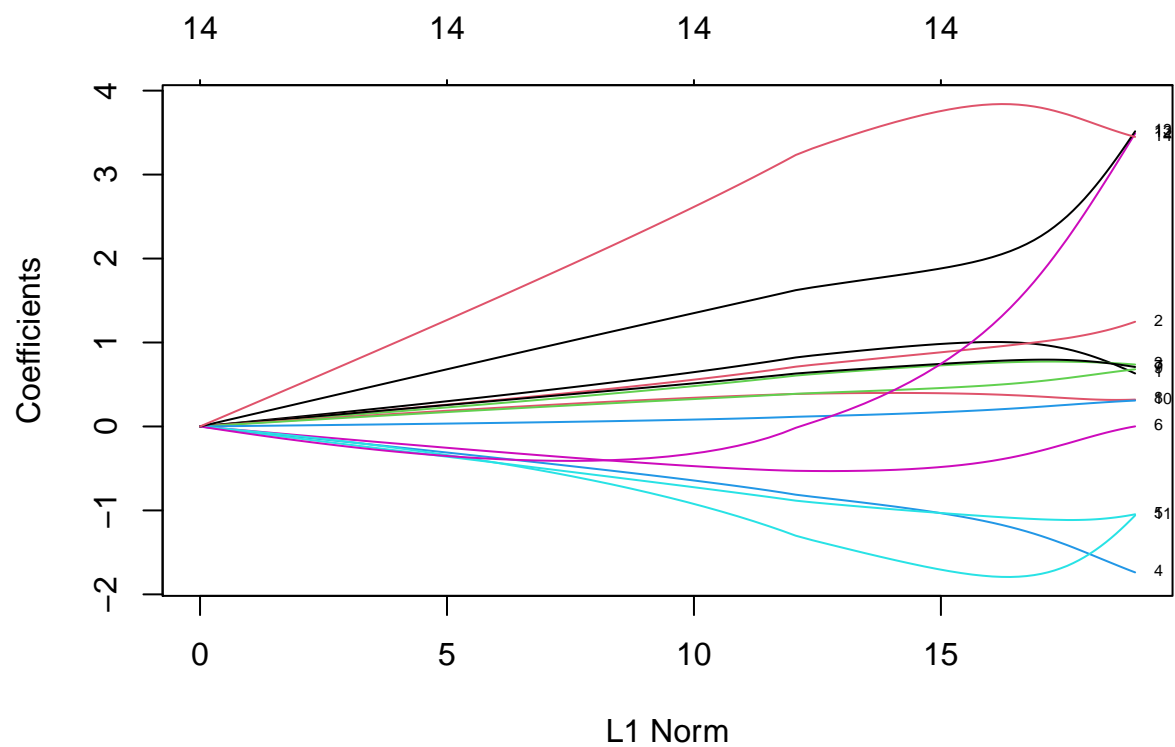
```
> mod.R$lambda %>% head()
## [1] 22007.27 20052.20 18270.82 16647.69 15168.76 13821.21
```

On peut récupérer les valeurs de **beta** associées à chaque valeur de la grille avec

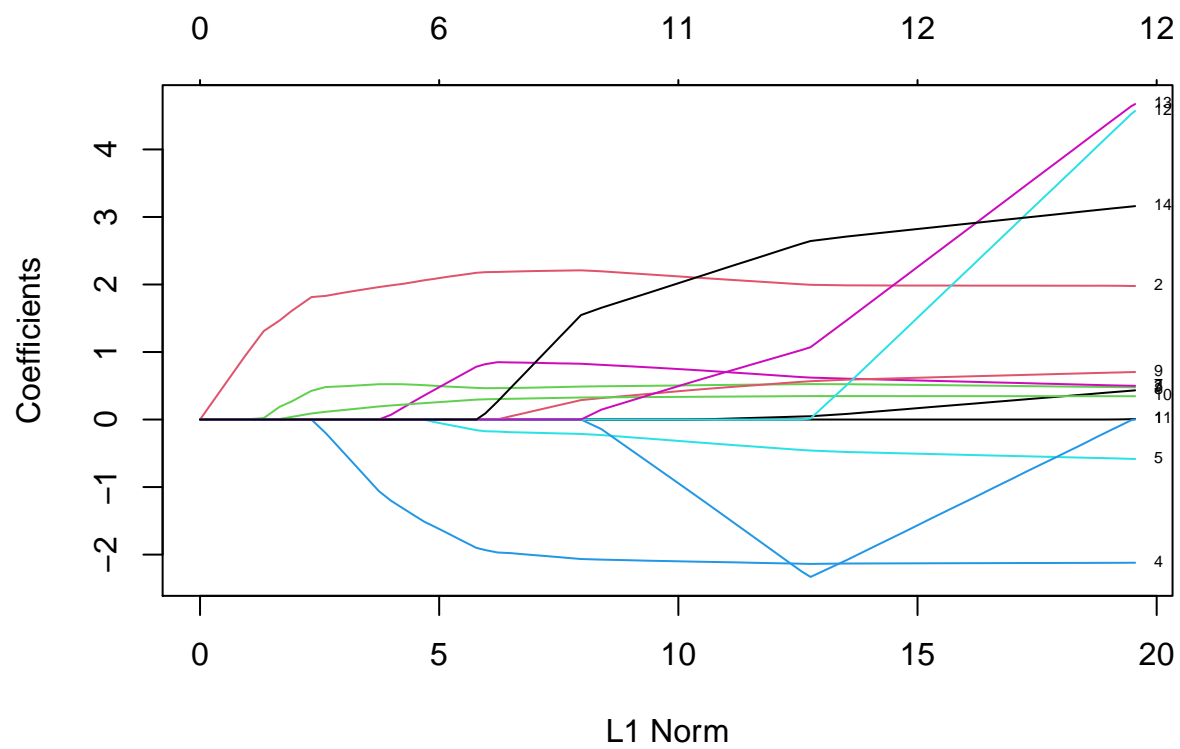
```
> mod.R$beta[,1]
##           T9           T12           T15           Ne9           Ne12
## 6.376767e-36 5.523924e-36 4.867402e-36 -6.821464e-36 -7.994984e-36
##           Ne15           Vx9           Vx12           Vx15           maxO3v
## -5.839057e-36 5.706014e-36 4.387350e-36 3.970583e-36 6.892387e-37
##           ventNord      ventOuest      ventSud      pluieSec
## -5.830507e-36 -1.022483e-35 1.519222e-35 2.772246e-35
```

3. Visualiser les chemins de régularisation des estimateurs **ridge** et **lasso**. On pourra utiliser la fonction `plot`.

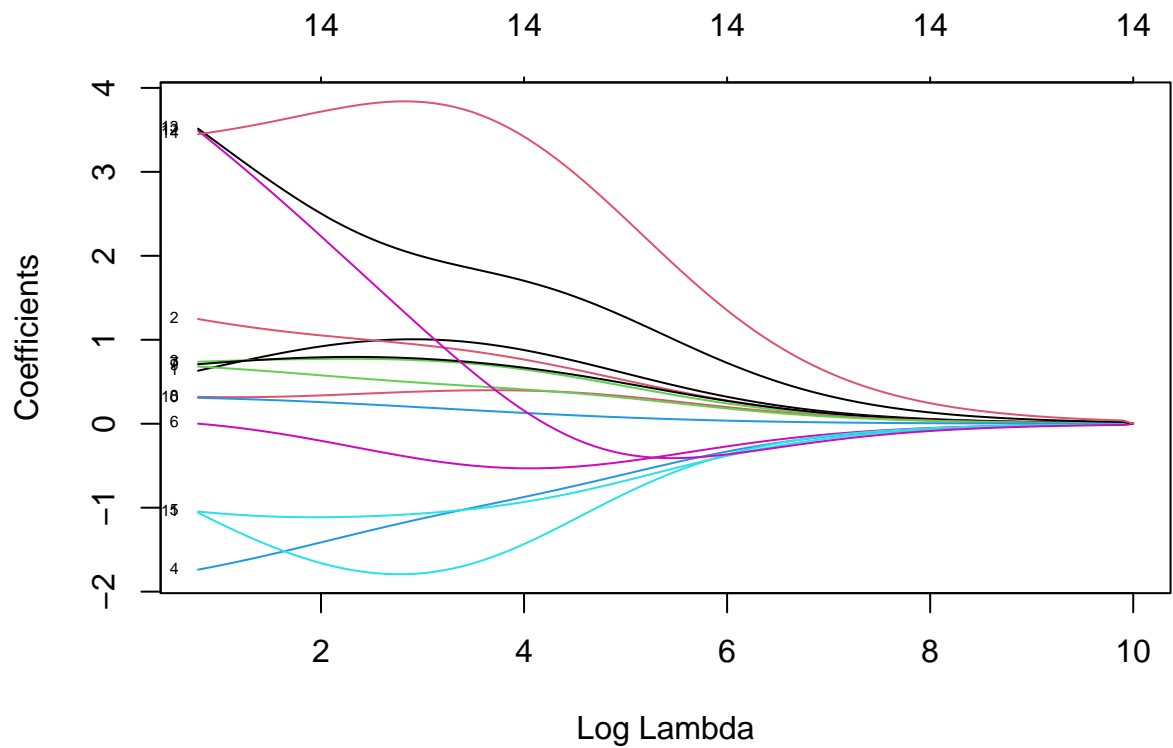

```
> plot(mod.R, label=TRUE)
```



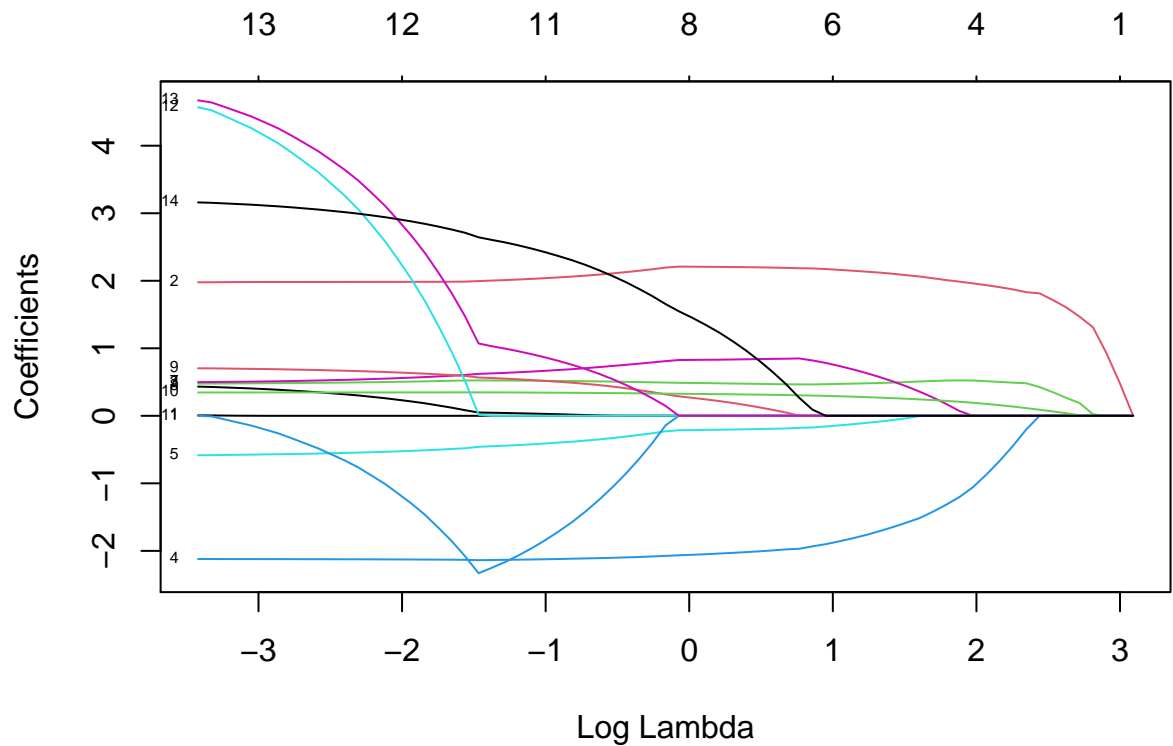
```
> plot(mod.L, label=TRUE)
```



```
> plot(mod.R, xvar="lambda", label=TRUE)
```



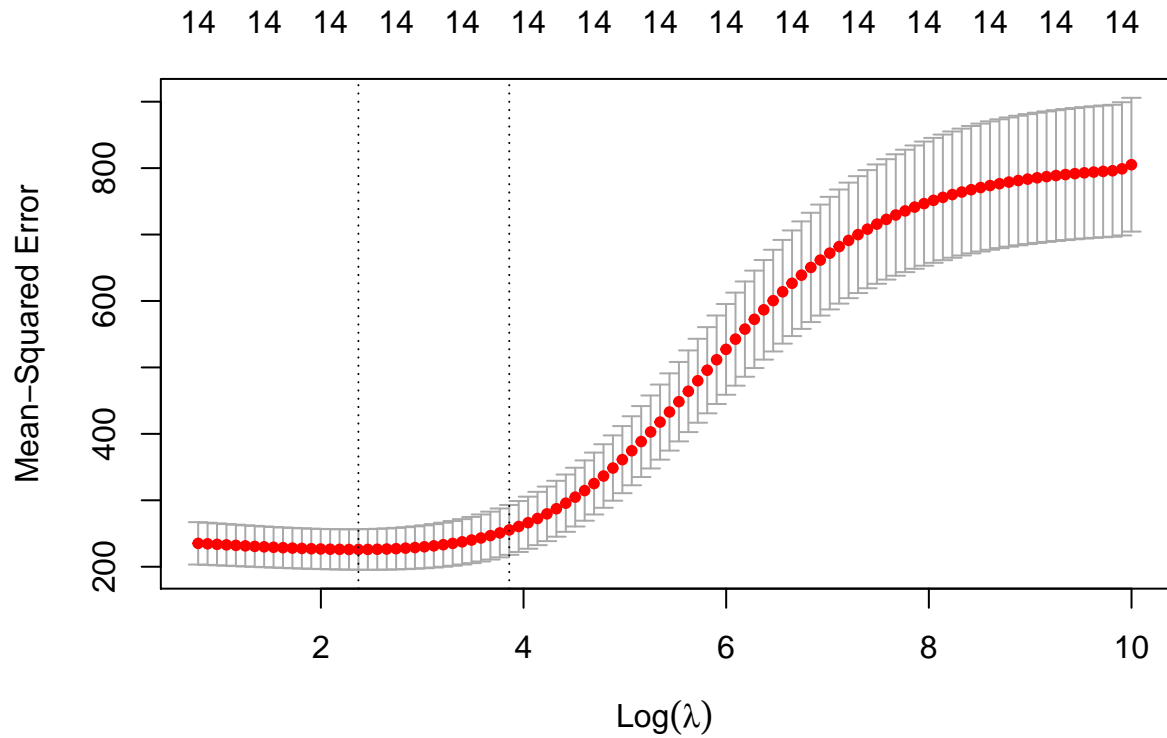
```
> plot(mod.L, xvar="lambda", label=TRUE)
```



4. Sélectionner les paramètres de régularisation à l'aide de la fonction `cv.glmnet`. On pourra notamment faire un `plot` de l'objet et expliquer le graphe obtenu.

Commençons par *ridge* :

```
> ridgeCV <- cv.glmnet(ozone.X, ozone.Y, alpha=0)
> plot(ridgeCV)
```



On visualise les erreurs quadratiques calculées par validation croisée 10 blocs en fonction de λ (échelle logarithmique). Deux traits verticaux sont représentés :

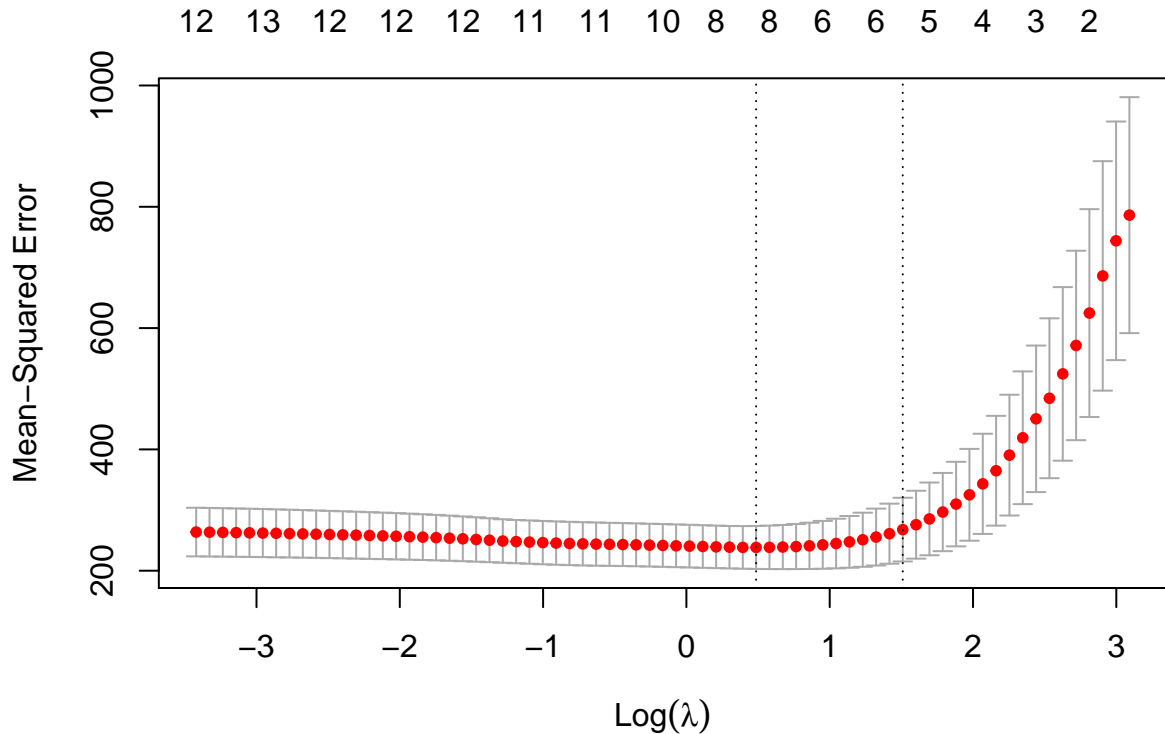
- celui de gauche correspond à la valeur de λ qui minimise l'erreur quadratique ;
- celui de droite correspond à la plus grande valeur de λ telle que l'erreur ne dépasse pas l'erreur minimale + 1 écart-type estimé de cette erreur.

D'un point de vu pratique, cela signifie que l'utilisateur peut choisir n'importe quelle valeur de λ entre les deux traits verticaux. Si on veut diminuer la complexité du modèle on choisit la valeur de droite. On peut obtenir ces deux valeurs avec

```
> ridgeCV$lambda.min
## [1] 10.70126
> ridgeCV$lambda.1se
## [1] 47.41322
```

On peut faire de même pour le **lasso** :

```
> lassoCV <- cv.glmnet(ozone.X, ozone.Y, alpha=1)
> plot(lassoCV)
```



5. On souhaite prédire la variable cible pour de nouveaux individus. Prenons par exemple les 25ème et 50ème individus du jeu de données. Calculer les valeurs prédites.

Une première approche pourrait consister à réajuster le modèle sur toutes les données pour la valeur de *lambda* sélectionnée. Cette étape est en réalité déjà effectuée par la fonction *cv.glmnet*. Il suffit par conséquent d'appliquer la fonction *predict* à l'objet obtenu avec *cv.glmnet* en spécifiant la valeur de *lambda* souhaitée. Par exemple pour *ridge* :

```
> predict(ridgeCV,newx = ozone.X[50:51,],s="lambda.min")
##           1
## 20010723 90.34787
## 20010724 96.71932
> predict(ridgeCV,newx = ozone.X[50:51,],s="lambda.1se")
##           1
## 20010723 93.33611
## 20010724 96.14918
```

On peut faire de même pour le *lasso* :

```
> predict(lassoCV,newx = ozone.X[50:51,],s="lambda.min")
##           1
## 20010723 87.19995
## 20010724 97.82825
> predict(lassoCV,newx = ozone.X[50:51,],s="lambda.1se")
##           1
## 20010723 87.40631
## 20010724 95.85602
```

6. A l'aide d'une validation croisée, comparer les performances des estimateurs **MCO**, **ridge** et **lasso**. On pourra utiliser les données *ozone_complet.txt* qui contiennent plus d'individus et de variables.

```
> ozone1 <- read.table("data/ozone_complet.txt",sep=";") %>% na.omit()
> ozone1.X <- model.matrix(maxO3~.,data=ozone1)[,-1]
> ozone1.Y <- ozone1$maxO3
```

On crée une fonction qui calcule les erreurs quadratiques par validations croisée des 3 procédures d'estimation.

```

> cv.ridge.lasso <- function(data,form){
+   set.seed(1234)
+   data.X <- model.matrix(form,data=data)[,-1]
+   data.Y <- data$maxO3
+   blocs <- caret::createFolds(1:nrow(data),k=10)
+   prev <- matrix(0,ncol=3,nrow=nrow(data)) %>% as.data.frame()
+   names(prev) <- c("lin","ridge","lasso")
+   for (k in 1:10){
+     app <- data[-blocs[[k]],]
+     test <- data[blocs[[k]],]
+     app.X <- data.X[-blocs[[k]],]
+     app.Y <- data.Y[-blocs[[k]]]
+     test.X <- data.X[blocs[[k]],]
+     test.Y <- data.Y[blocs[[k]]]
+     ridge <- cv.glmnet(app.X,app.Y,alpha=0)
+     lasso <- cv.glmnet(app.X,app.Y,alpha=1)
+     lin <- lm(form,data=app)
+     prev[blocs[[k]],] <- tibble(lin=predict(lin,newdata=test),
+                               ridge=as.vector(predict(ridge,newx=test.X)),
+                               lasso=as.vector(predict(lasso,newx=test.X)))
+   }
+   err <- prev %>% mutate(obs=data$maxO3) %>% summarise_at(1:3,~mean((obs-. )^2))
+   return(err)
+ }

```

```

> cv.ridge.lasso(ozone1,form=formula(maxO3~.))
##      lin      ridge      lasso
## 1 184.3755 192.4984 191.5436

```

On remarque que les approches régularisées n'apportent rien par rapport aux estimateurs MCO ici. Ceci peut s'expliquer par le fait que le nombre de variables n'est pas très important.

7. Refaire la question précédente en considérant toutes les interactions d'ordre 2.

```

> cv.ridge.lasso(ozone1,form=formula(maxO3~.^2))
##      lin      ridge      lasso
## 1 185.0517 168.7122 166.0982

```

Les méthodes régularisées permettent ici de diminuer les erreurs quadratiques de manière intéressante. Cela vient certainement du fait du nombre de variables explicatives qui est beaucoup plus important lorsqu'on prend en compte toutes les interactions d'ordre 2, nous en avons en effet 253 :

```

> ozone2.X <- model.matrix(maxO3~.^2,data=ozone1)[,-1]
> dim(ozone2.X)
## [1] 1366 253

```

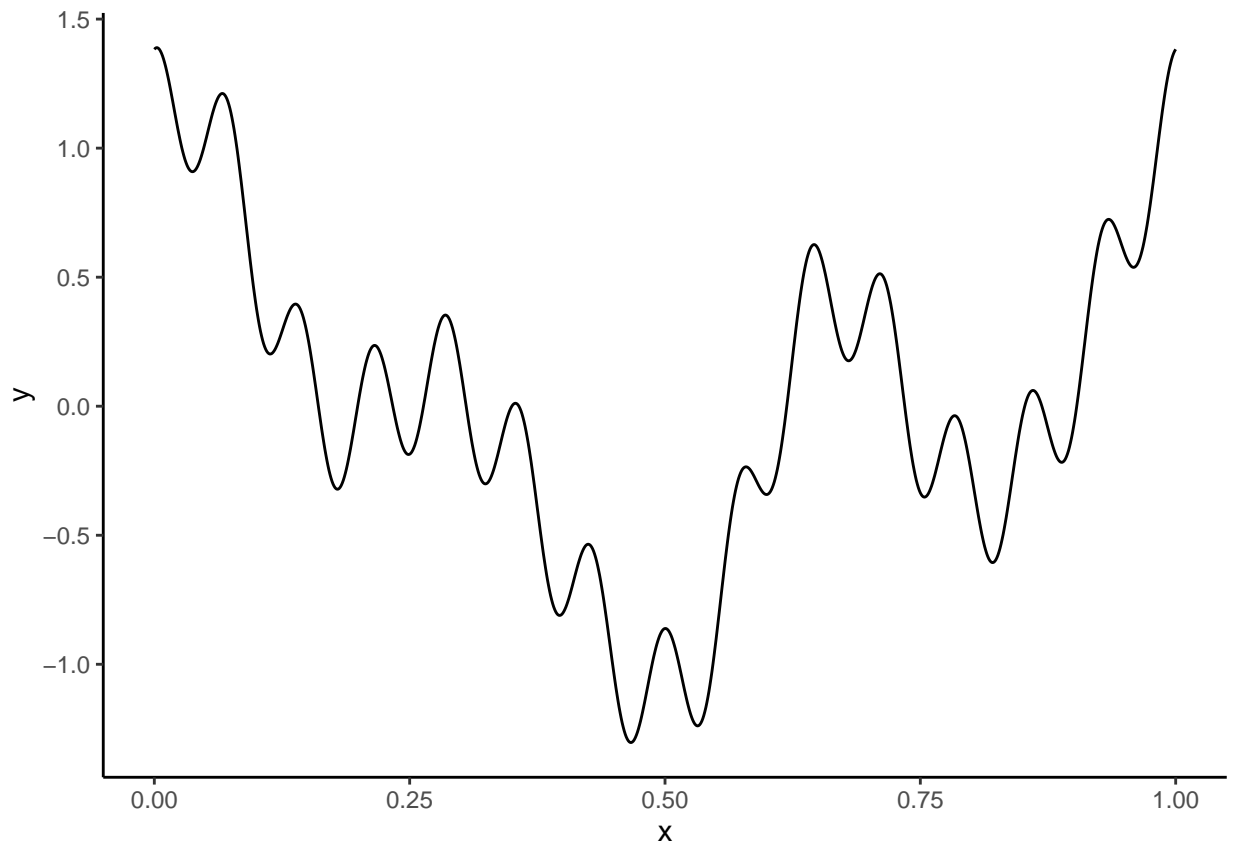
3.2 Reconstruction d'un signal

Le fichier `signal.csv` contient un signal que l'on peut représenter par une fonction $m : \mathbb{R} \rightarrow \mathbb{R}$. On le visualise

```

> signal <- read_csv("data/signal.csv")
> ggplot(signal)+aes(x=x,y=y)+geom_line()

```

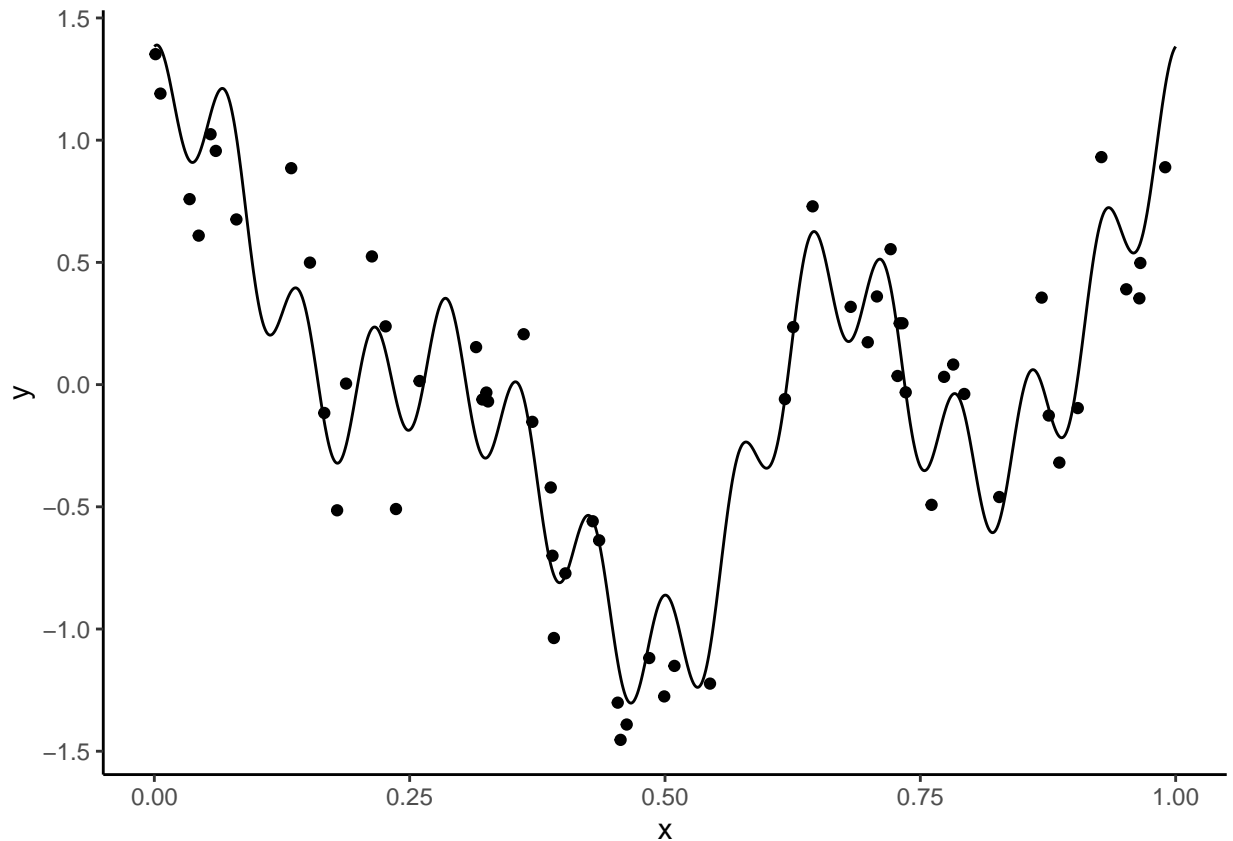


Plaçons nous dans le cas où on ne dispose que d'une version bruitée de ce signal. La courbe n'est pas observée mais on dispose d'un échantillon $(x_i, y_i), i = 1, \dots, n$ généré selon le modèle

$$y_i = m(x_i) + \varepsilon_i.$$

Le fichier `ech_signal.csv` contient $n = 60$ observations issues de ce modèle. On représente les données et la courbe

```
> donnees <- read_csv("data/ech_signal.csv")
> ggplot(signal)+aes(x=x,y=y)+geom_line()+
+   geom_point(data=donnees,aes(x=X,y=Y))
```



Nous cherchons dans cette partie à reconstruire le signal à partir de l'échantillon. Bien entendu, vu la forme du signal, un modèle linéaire de la forme

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

n'est pas approprié. De nombreuses approches en **traitement du signal** proposent d'utiliser une **base** ou **dictionnaire** représentée par une collection de fonctions $\{\psi_j(x)\}_{j=1,\dots,K}$ et de décomposer le signal dans cette base :

$$m(x) \approx \sum_{j=1}^K \beta_j \psi_j(x).$$

Pour un dictionnaire donné, on peut alors considérer un **modèle linéaire**

$$y_i = \sum_{j=1}^K \beta_j \psi_j(x) + \varepsilon_i. \quad (1)$$

Le problème est toujours d'estimer les paramètres β_j mais les variables sont maintenant définies par les éléments du dictionnaire. Il existe différents types de dictionnaire, dans cet exercice nous proposons de considérer la base de Fourier définie par

$$\psi_0(x) = 1, \quad \psi_{2j-1}(x) = \cos(2j\pi x) \quad \text{et} \quad \psi_{2j}(x) = \sin(2j\pi x), \quad j = 1, \dots, K.$$

1. Écrire une fonction **R** qui admet en entrée :
 - une grille de valeurs de **x** (un vecteur)

— une valeur de K (un entier positif)

et qui renvoie en sortie une matrice qui contiennent les valeurs du dictionnaire pour chaque valeur de x . Cette matrice devra donc contenir $2K$ colonnes et le nombre de lignes sera égal à la longueur du vecteur x .

```
> mat.dict <- function(K,x){  
+   res <- matrix(0,nrow=length(x),ncol=2*K) %>% as_tibble()  
+   for (j in 1:K){  
+     res[,2*j-1] <- cos(2*j*pi*x)  
+     res[,2*j] <- sin(2*j*pi*x)  
+   }  
+   return(res)  
+ }
```

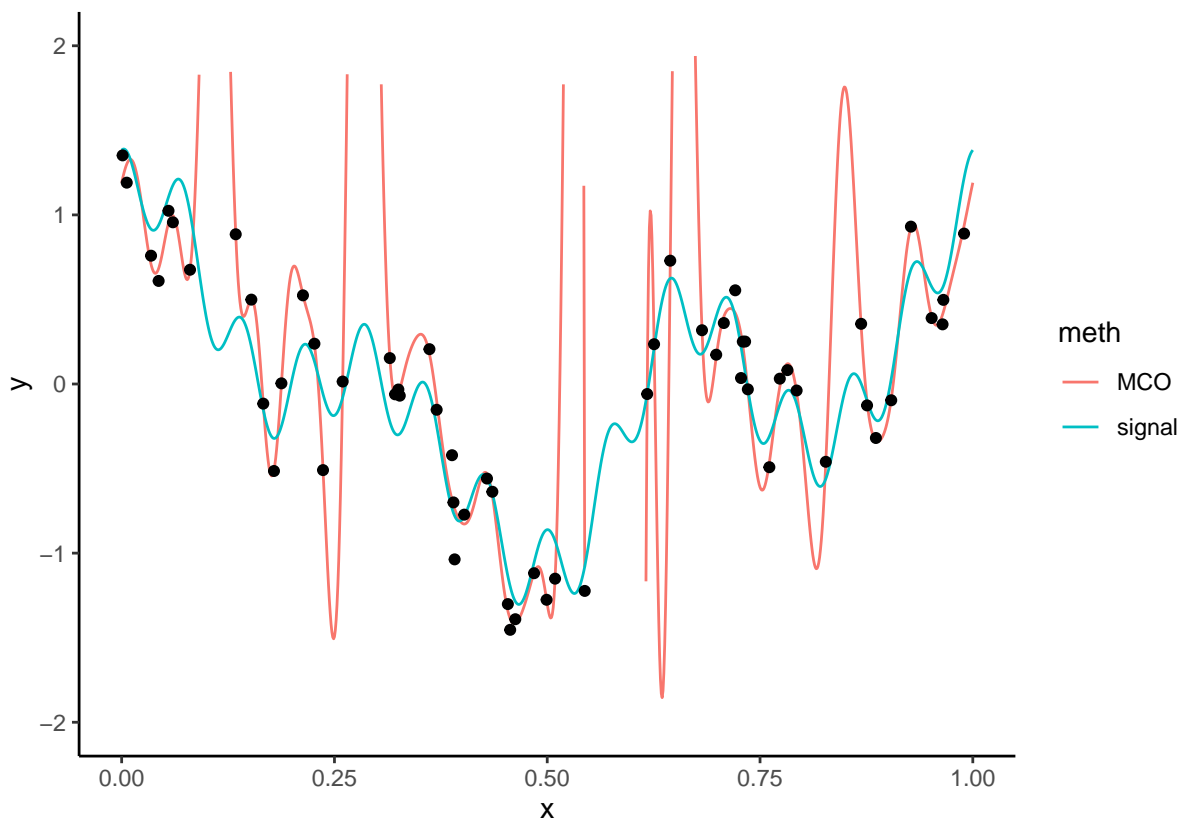
2. On fixe $K=25$. Calculer les estimateurs des moindres carrés du modèle (1).

Il suffit d'ajuster le modèle linéaire où les variables explicatives sont données par le dictionnaire :

```
> D25 <- mat.dict(25,donnees$X) %>% mutate(Y=donnees$Y)  
> mod.lin <- lm(Y~.,data=D25)
```

3. Représenter le signal estimé. Commenter le graphe.

```
> S25 <- mat.dict(25,signal$x)  
> prev.MCO <- predict(mod.lin,newdata = S25)  
> signal1 <- signal %>% mutate(MCO=prev.MCO) %>% rename(signal=y)  
> signal2 <- signal1 %>% pivot_longer(-x,names_to="meth",values_to="y")  
> ggplot(signal2)+aes(x=x,y=y)+geom_line(aes(color=meth))+  
+   scale_y_continuous(limits = c(-2,2))+geom_point(data=donnees,aes(x=X,y=Y))
```

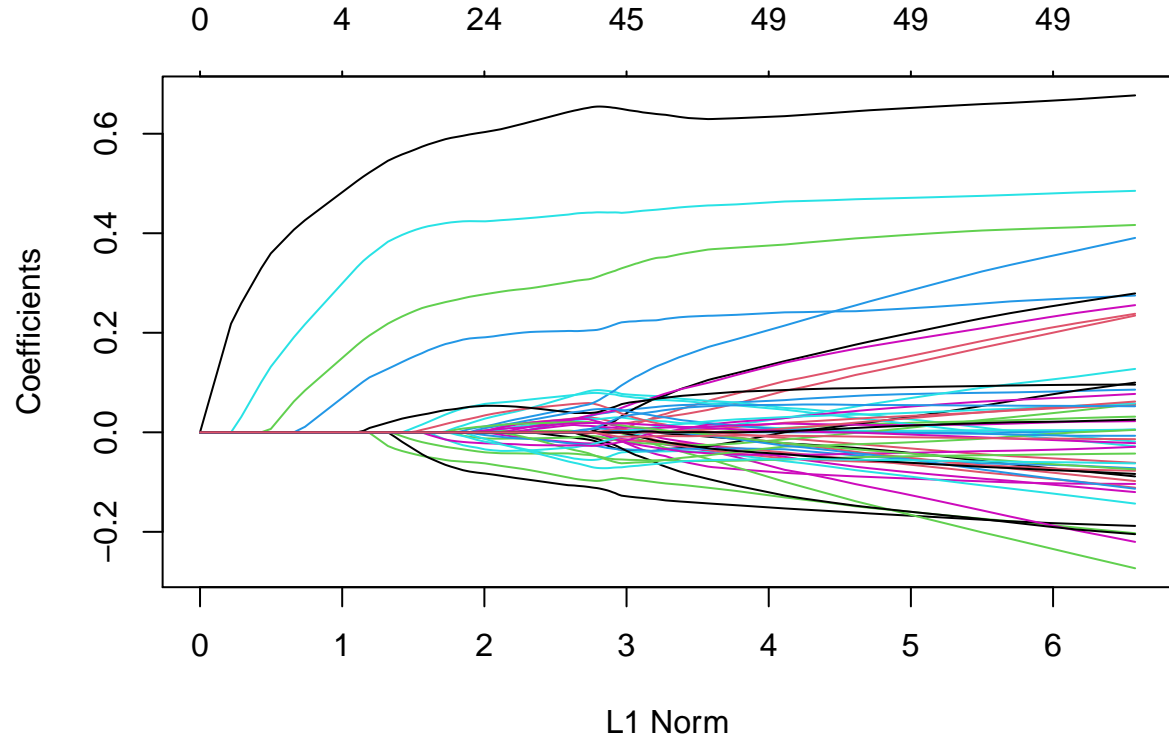


Le signal estimé a tendance à surajuster les données. Cela vient du fait que on estime 51 paramètres avec seulement 60 observations.

4. Calculer les estimateurs **lasso** et représenter le signal issu de ces estimateurs.

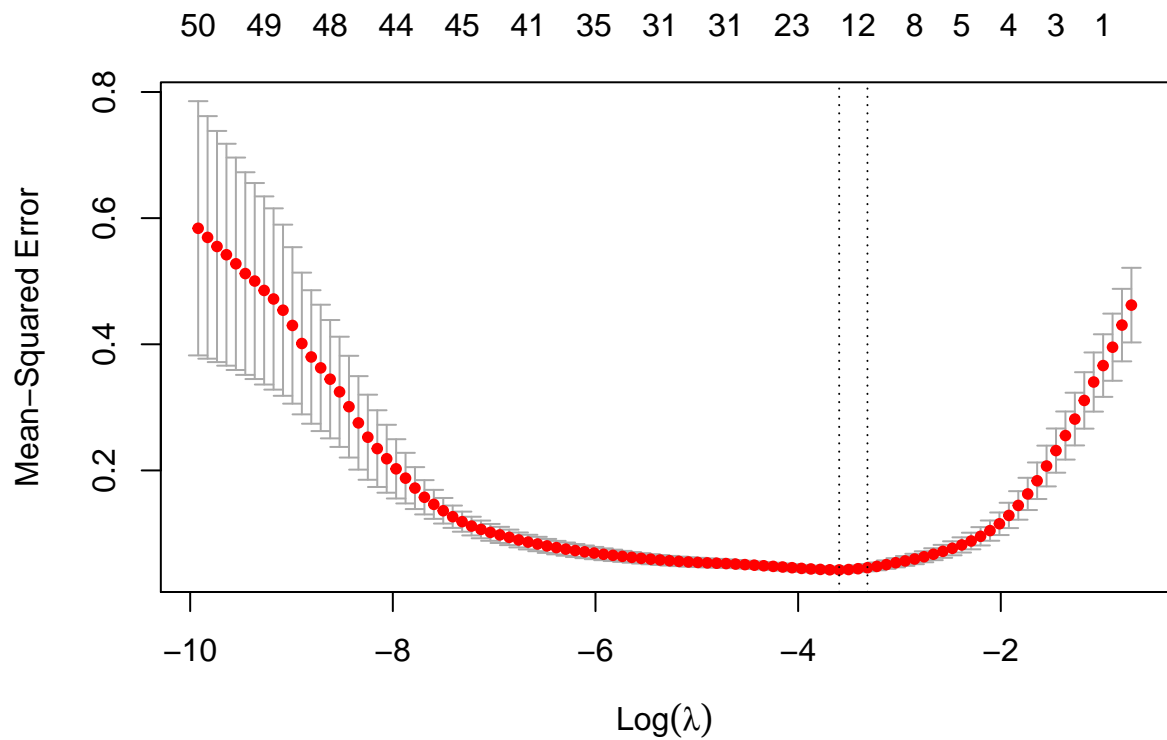
*On regarde tout d'abord le chemin de régularisation des estimateurs **lasso***

```
> X.25 <- model.matrix(Y~.,data=D25)[,-1]
> lasso1 <- glmnet(X.25,D25$Y,alpha=1)
> plot(lasso1)
```



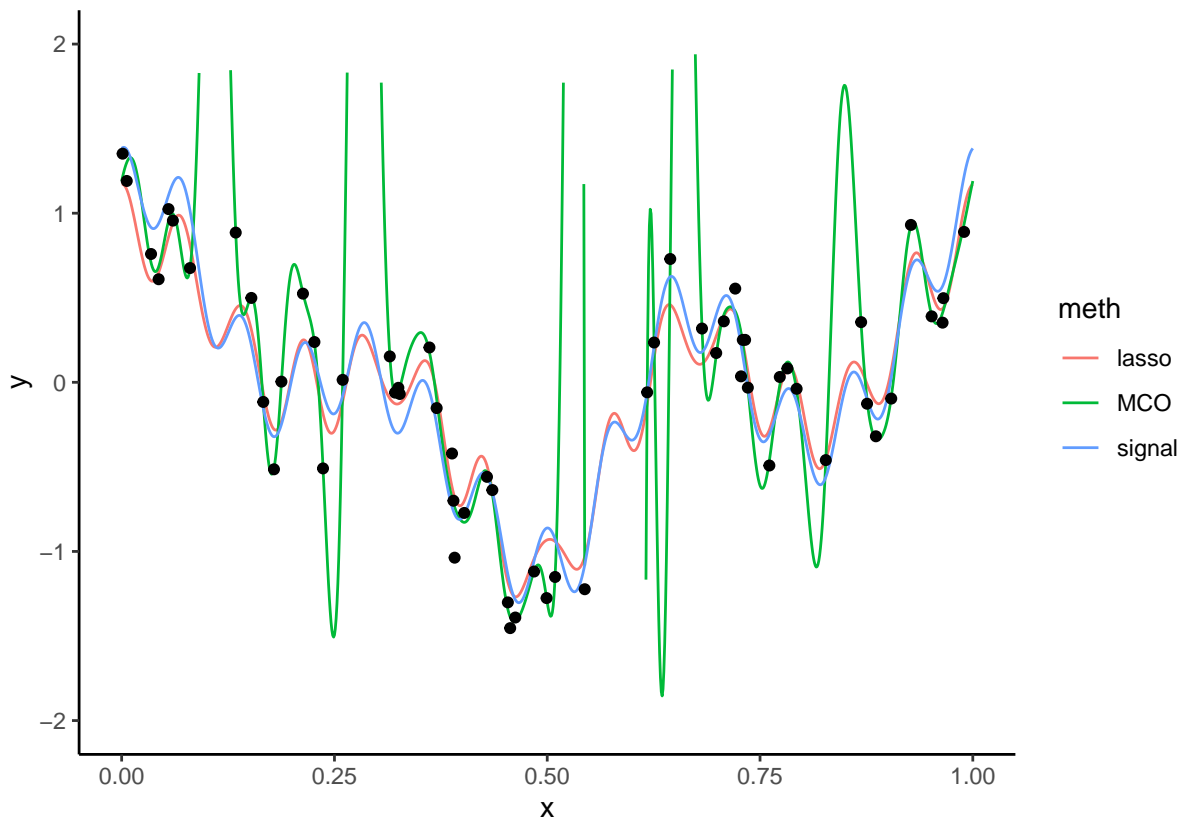
Il semble que quelques coefficients quittent la valeur 0 bien avant les autres. On effectue maintenant la validation croisée pour sélectionner le paramètre λ .

```
> lasso.cv <- cv.glmnet(X.25,D25$Y,alpha=1)
> plot(lasso.cv)
```



On calcule les prévisions et on trace le signal.

```
> prev.lasso <- as.vector(predict(lasso.cv,newx=as.matrix(S25)))
> signal1$lasso <- prev.lasso
> signal2 <- signal1 %>% pivot_longer(-x,names_to="meth",values_to="y")
> ggplot(signal2)+aes(x=x,y=y)+geom_line(aes(color=meth))+
+   scale_y_continuous(limits = c(-2,2))+geom_point(data=donnees,aes(x=X,y=Y))
```



L'algorithme **lasso** a permis de corriger le problème de sur-apprentissage.

5. Identifier les coefficients lasso sélectionnés qui ne sont pas nuls.

```
> v.sel <- which(coef(lasso.cv)!=0)
> v.sel
## [1] 1 2 4 5 6 8 21 28 30 36 37 38 40
```

6. Ajouter les signaux ajustés par les algorithmes PCR et PLS.

— On effectue la PCR :

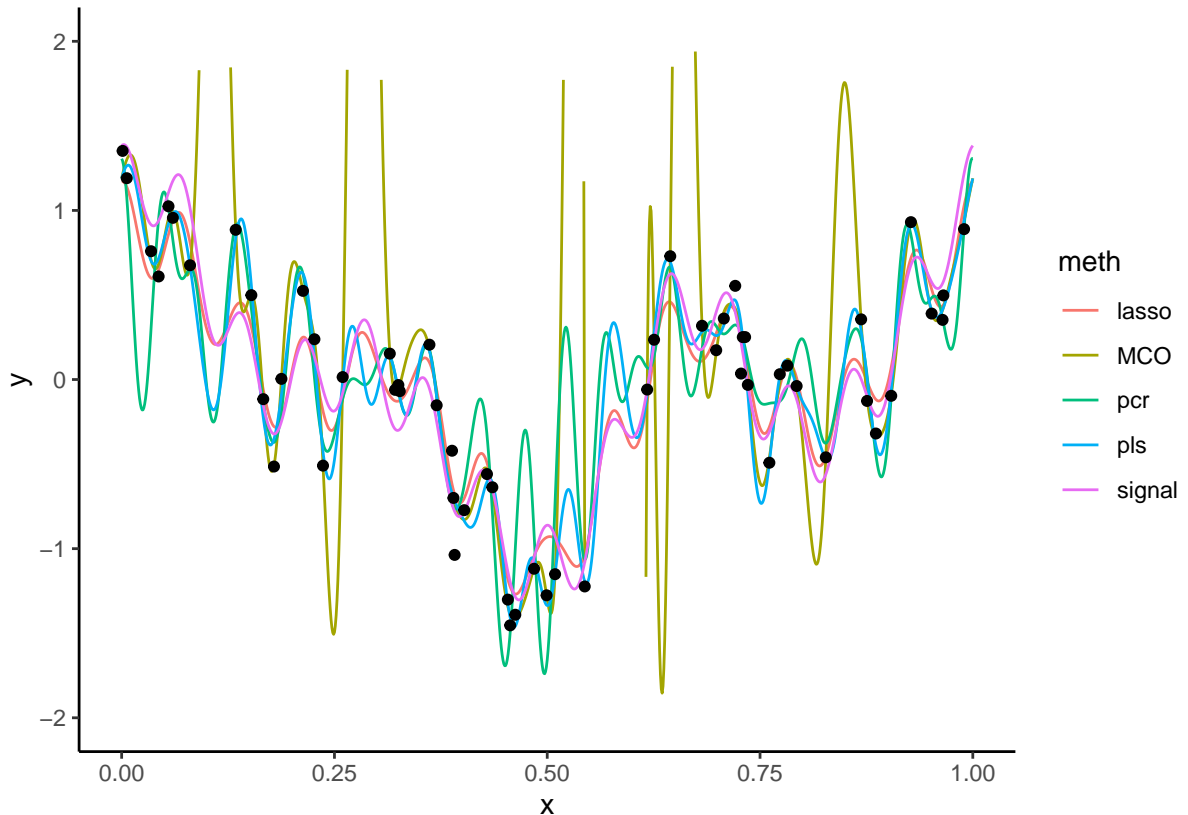
```
> pcr.fit <- pcr(Y~.,data=D25,validation="CV")
> ncomp.pcr <- which.min(pcr.fit$validation$PRESS)
> ncomp.pcr
## [1] 33
> prev.pcr <- predict(pcr.fit,newdata=S25,ncomp=ncomp.pcr)
```

— Puis la PLS :

```
> pls.fit <- plsr(Y~.,data=D25,validation="CV")
> ncomp.pls <- which.min(pls.fit$validation$PRESS)
> ncomp.pls
## [1] 7
> prev.pls <- predict(pls.fit,newdata=S25,ncomp=ncomp.pls)
```

— On trace les signaux :

```
> signal1$pcr <- prev.pcr
> signal1$pls <- prev.pls
> signal2 <- signal1 %>% pivot_longer(-x,names_to="meth",values_to="y")
> ggplot(signal2)+aes(x=x,y=y)+geom_line(aes(color=meth))+
+   scale_y_continuous(limits = c(-2,2))+geom_point(data=donnees,aes(x=X,y=Y))
```



On peut également obtenir les erreurs quadratiques (puisque l'on connaît la vraie courbe)

```
> signal1 %>% summarise_at(-(1:2), ~mean((.-signal)^2)) %>%
+   sort() %>% round(3)
## # A tibble: 1 x 4
##   lasso pls pcr MCO
##   <dbl> <dbl> <dbl> <dbl>
## 1 0.014 0.055 0.152 598.
```

3.3 Régression logistique pénalisée

On considère le jeu de données sur la détection d'images publicitaires disponible ici <https://archive.ics.uci.edu/ml/datasets/internet+advertisements>.

```
> ad.data <- read.table("data/ad_data.txt", header=FALSE, sep="," , dec=".", na.strings = "?", strip.white = " ")
> names(ad.data)[ncol(ad.data)] <- "Y"
> ad.data$Y <- as.factor(ad.data$Y)
```

La variable à expliquer est

```
> summary(ad.data$Y)
##   ad. nonad.
##   459   2820
```

Cette variable est binaire. On considère une régression logistique pour expliquer cette variable. Le nombre de variables explicatives étant important, comparer les algorithmes du maximum de vraisemblance aux algorithmes de type **ridge/lasso** en faisant une validation croisée 10 blocs. On pourra utiliser comme critère de comparaison l'erreur de classification, la courbe ROC et l'AUC. Il faudra également prendre des décisions pertinentes vis-à-vis des données manquantes...

On commence par regarder les données manquantes :

```
> sum(is.na(ad.data))
## [1] 2729
> var.na <- apply(is.na(ad.data),2,any)
> names(ad.data)[var.na]
## [1] "V1" "V2" "V3" "V4"
> ind.na <- apply(is.na(ad.data),1,any)
> sum(ind.na)
## [1] 920
```

On remarque que 920 individus ont au moins une donnée manquante alors que seules les 4 premières variables ont des données manquantes, on choisit donc de supprimer ces 4 variables.

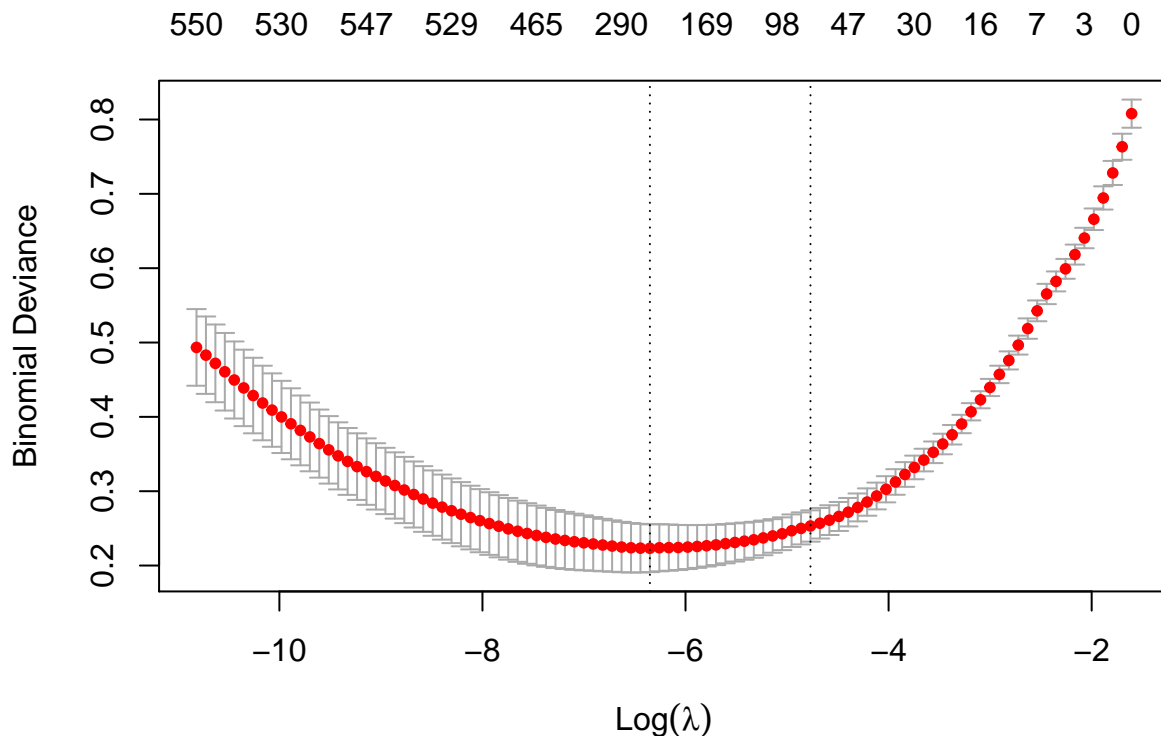
```
> ad.data1 <- ad.data[,var.na==FALSE]
> dim(ad.data1)
## [1] 3279 1555
> sum(is.na(ad.data1))
## [1] 0
```

On construit les matrices des variables explicatives pour les méthodes lasso et ridge (`glmnet` veut les variables explicatives sous forme de matrices).

```
> X.ad <- model.matrix(Y~.,data=ad.data1)[,-1]
> Y.ad <- ad.data1$Y
```

Avant de faire la validation croisée, nous présentons juste comment faire l'algorithme lasso. Comme pour la régression, on utilise la fonction `cv.glmnet`, il faut juste ajouter l'argument `family="binomial"` :

```
> set.seed(1234)
> lasso.cv <- cv.glmnet(X.ad,Y.ad,family="binomial",alpha=1)
> plot(lasso.cv)
```



Par défaut le critère utilisé pour la classification binaire est celui de la *déviance*. On peut utiliser d'autres

critères comme l'erreur de classification ou l'auc en modifiant l'argument `type.measure`. On gardera la déviance dans la suite. On peut maintenant faire la validation croisée 10 blocs pour calculer les prévisions des 3 algorithmes.

```
> blocs <- caret::createFolds(1:nrow(ad.data1),k=10)
> score <- matrix(0,ncol=3,nrow=nrow(ad.data1)) %>% as.data.frame()
> names(score) <- c("MV","ridge","lasso")
> for (k in 1:10){
+   app <- ad.data1[-blocs[[k]],]
+   test <- ad.data1[blocs[[k]],]
+   app.X <- X.ad[-blocs[[k]],]
+   app.Y <- Y.ad[-blocs[[k]]]
+   test.X <- X.ad[blocs[[k]],]
+   test.Y <- Y.ad[blocs[[k]]]
+   ridge <- cv.glmnet(app.X,app.Y,family="binomial",alpha=0)
+   lasso <- cv.glmnet(app.X,app.Y,family="binomial",alpha=1)
+   MV <- glm(Y~.,data=app,family="binomial")
+   score[blocs[[k]],] <- tibble(MV=predict(MV,newdata=test,type="response"),
+                               ridge=as.vector(predict(ridge,newx=test.X,type="response")),
+                               lasso=as.vector(predict(lasso,newx=test.X,type="response")))
+ }
```

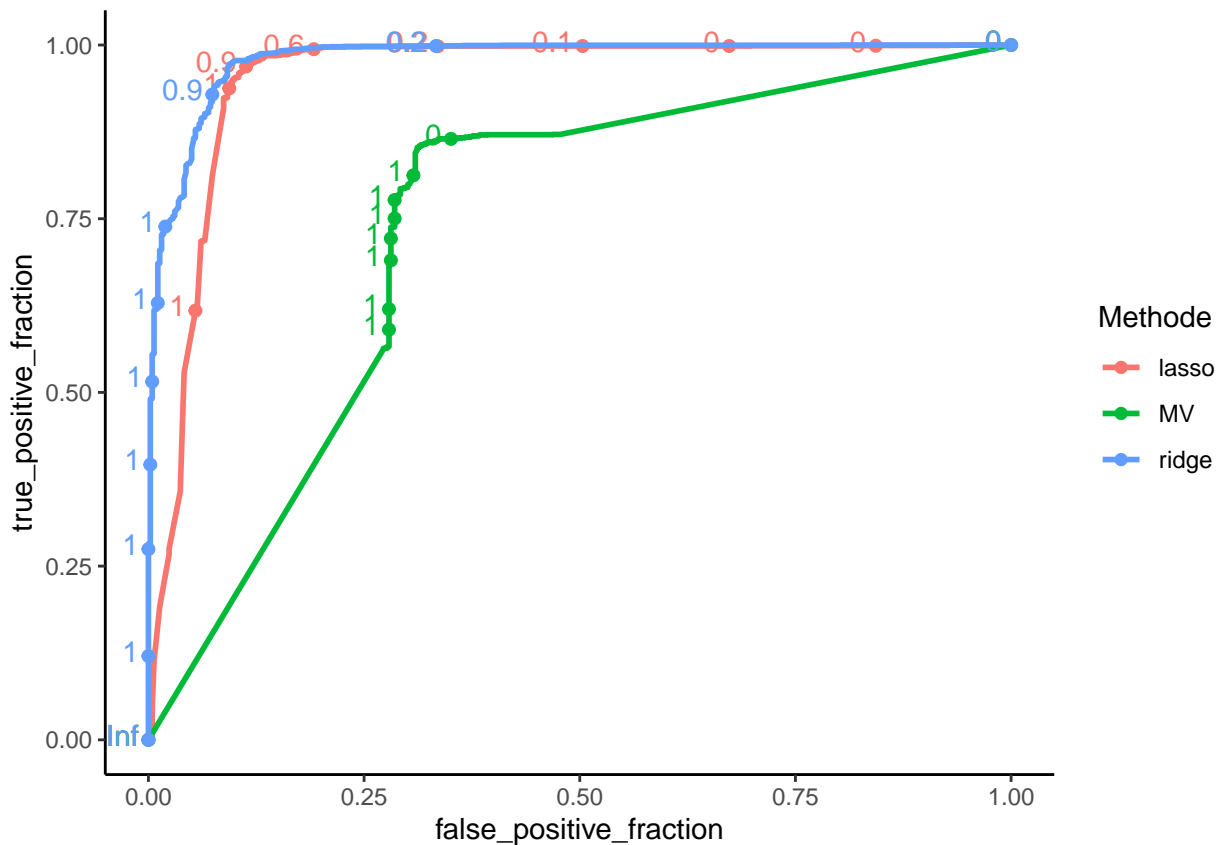
Le `tibble score` contient, pour chaque individu, les prévisions eds probabilités a posteriori

$$P(Y = \text{nonad.} | X = x_i), \quad i = 1, \dots, n.$$

On peut déduire de ce tableau les critères souhaités :

— les courbes ROC :

```
> score1 <- score %>%
+   mutate(obs=fct_recode(ad.data1$Y,"0"="ad.", "1"="nonad.")) %>%
+   pivot_longer(-obs,names_to="Methode",values_to="score")
> ggplot(score1)+aes(m=score,d=as.numeric(obs),color=Methode)+plotROC::geom_roc()
```



— les AUC :

```
> score1 %>% group_by(Methode) %>%
+   summarize(AUC=round(as.numeric(pROC::auc(obs,score)),3)) %>%
+   arrange(desc(AUC))
## # A tibble: 3 x 2
##   Methode   AUC
##   <chr>   <dbl>
## 1 ridge   0.98
## 2 lasso   0.952
## 3 MV      0.739
```

— les erreurs de classification :

```
> score1 %>% mutate(prev=round(score),err=prev!=obs) %>%
+   group_by(Methode) %>% summarize(Err_classif=round(mean(err),3)) %>%
+   arrange(Err_classif)
## # A tibble: 3 x 2
##   Methode Err_classif
##   <chr>         <dbl>
## 1 lasso         0.031
## 2 ridge         0.031
## 3 MV           0.166
```

On remarque que les méthodes pénalisées sont nettement meilleures que l'approche classique par maximum de vraisemblance sur cet exemple.

4 Modèle additif

Le modèle additif (modèle GAM) peut être vu comme un compromis entre une modélisation linéaire et non paramétrique de la fonction de régression. Il suppose que cette fonction s'écrit

$$m(x) = m(x_1, \dots, x_d) = \alpha + g_1(x_1) + \dots + g_d(x_d).$$

4.1 Pseudo backfitting

L'algorithme du backfitting est souvent utilisé pour estimer les composantes du modèle additif. Etant donné un échantillon $(x_i, y_i), i = 1, \dots, n$ on note \bar{Y} le vecteur des y_i et \bar{X}_k le vecteur contenant les observations de la variable k pour $k = 1, \dots, d$. L'algorithme se résume ainsi

1. Initialisation : $\hat{\alpha} = \bar{Y}, \hat{g}_k(x_k) = \bar{X}_k$.
2. Pour $k = 1, \dots, d$:
 - $\bar{Y}^{(k)} = \bar{Y} - \hat{\alpha} - \sum_{j \neq k} \hat{g}_j(\bar{X}_j)$ (résidus partiels)
 - \hat{g}_k : lissage non paramétrique de $\bar{Y}^{(k)}$ sur \bar{X}_k .
3. Répéter l'étape précédente tant que les \hat{g}_k changent.

On propose dans cette partie d'utiliser cet algorithme pour estimer les paramètres du modèle linéaire en remplaçant le lissage non paramétrique par un estimateur MCO. On considère le modèle de régression linéaire

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$$

avec X_1 et X_2 de lois uniformes sur $[0, 1]$ et ε de loi $\mathcal{N}(0, 1)$ (ε est indépendante de $(X_1, X_2)'$).

1. Générer un échantillon (x_i, y_i) de taille $n = 300$ selon le modèle ci-dessus pour $\beta_0 = 1, \beta_1 = 3, \beta_2 = 5$.

```
> set.seed(1234)
> n <- 300
> X1<-runif(n)
> X2<-runif(n)
> bruit<-rnorm(n)
> Y<-1+3*X1+5*X2+bruit
> donnees<-data.frame(Y,X1,X2)
```

2. Créer une fonction **R** qui admet en entrée un jeu de données et qui fournit en sortie les estimateurs par la méthode du backfitting.

```
> pseudo_back <- function(df,eps=0.00001){
+   mat.X <- model.matrix(Y~.,data=df)
+   beta_i <- rep(0,ncol(mat.X))
+   beta <- rep(1,ncol(mat.X))
+   while (min(abs(beta_i-beta))>eps){
+     beta_i <- beta
+     for (k in 1:ncol(mat.X)){
+       Yk <- Y-mat.X[,-k]%*(beta[-k])
+       dfk <- data.frame(Yk=Yk,Xk=mat.X[,k])
+       beta[k]<-coef(lm(Yk~Xk-1,data=dfk))
+     }
+   }
+   return(beta)
+ }
```

3. En déduire les estimateurs backfitting pour le problème considéré.

```
> pseudo_back(donnees)
## [1] 1.021341 2.864543 4.980367
```


4. Comparer aux estimateurs **MCO**.

```
> lm(Y~.,data=donnees)
##
## Call:
## lm(formula = Y ~ ., data = donnees)
##
## Coefficients:
## (Intercept)          X1          X2
##      1.021      2.865      4.981
```

On obtient les mêmes estimateurs.

4.2 Modèle GAM

On considère les données générées selon

```
> n <- 1000
> set.seed(1465)
> X1 <- 2*runif(n)
> X2 <- 2*runif(n)
> bruit <- rnorm(n)
> Y <- 2*X1+sin(8*pi*X2)+bruit
> donnees<-data.frame(Y,X1,X2)
```

1. Écrire le modèle

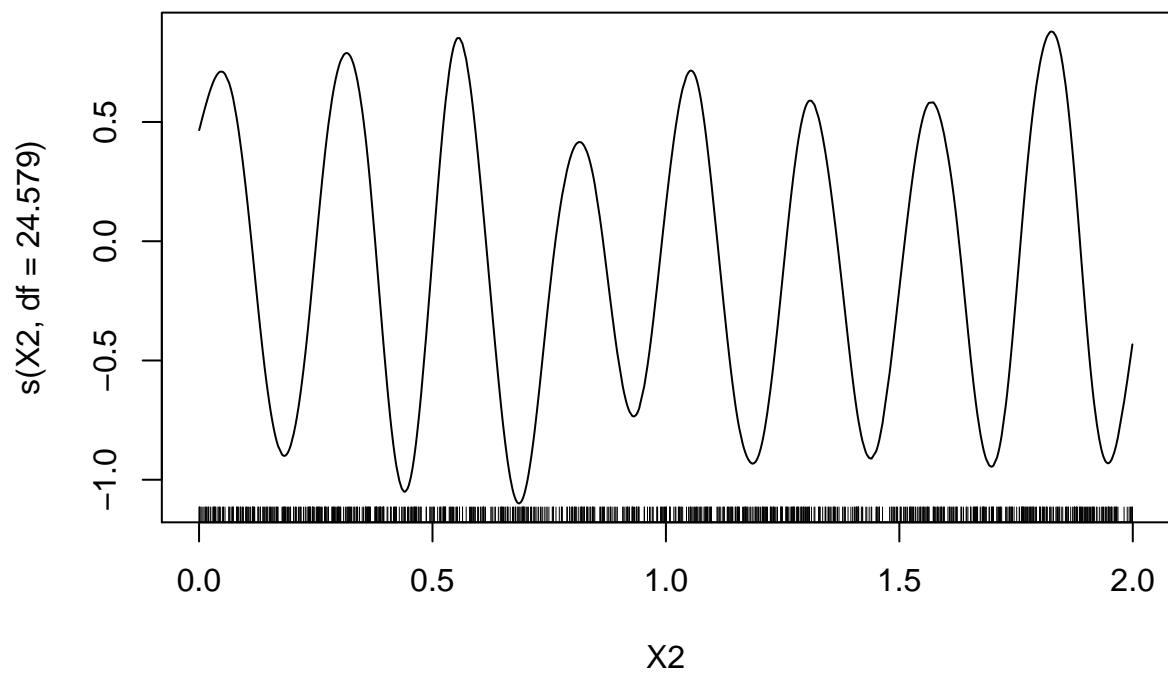
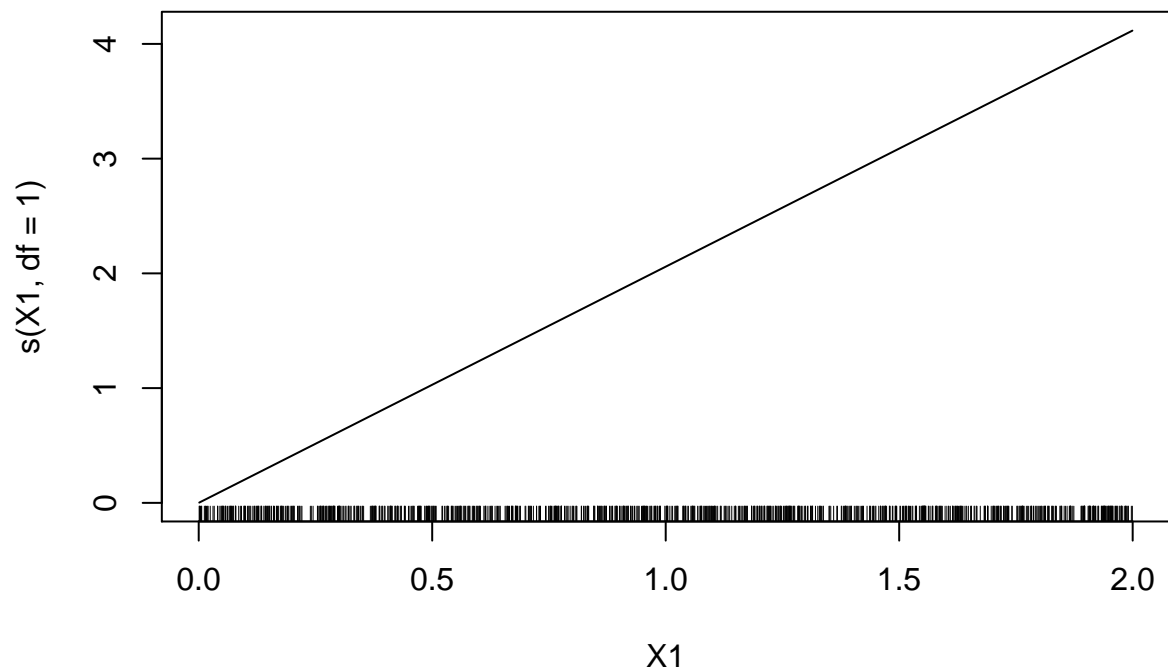
Il s'agit d'un modèle additif

$$Y = 2X_1 + \sin(8\pi X_2) + \varepsilon$$

où X_1 et X_2 sont uniformes sur $[0, 1]$ et ε suit une $\mathcal{N}(0, 1)$.

2. A l'aide du package **gam** visualiser les estimateurs des composantes additives du modèle. On utilisera tout d'abord un lissage par **spline** avec 1 ddl pour la première composante et 24.579 ddl pour la seconde.

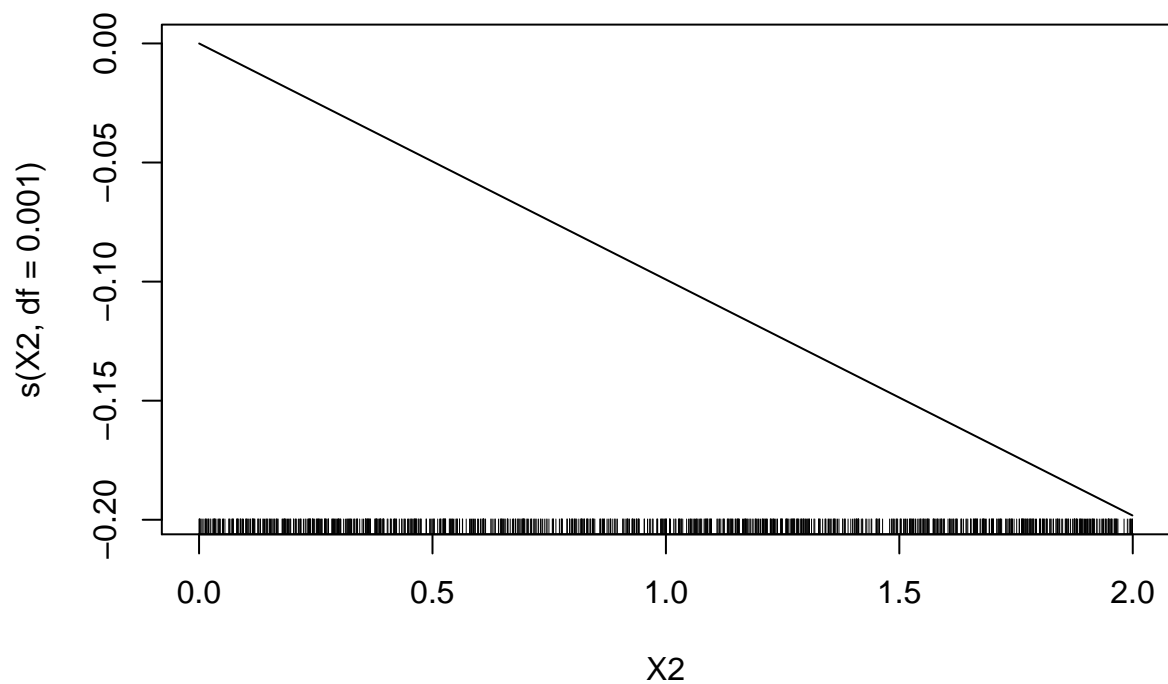
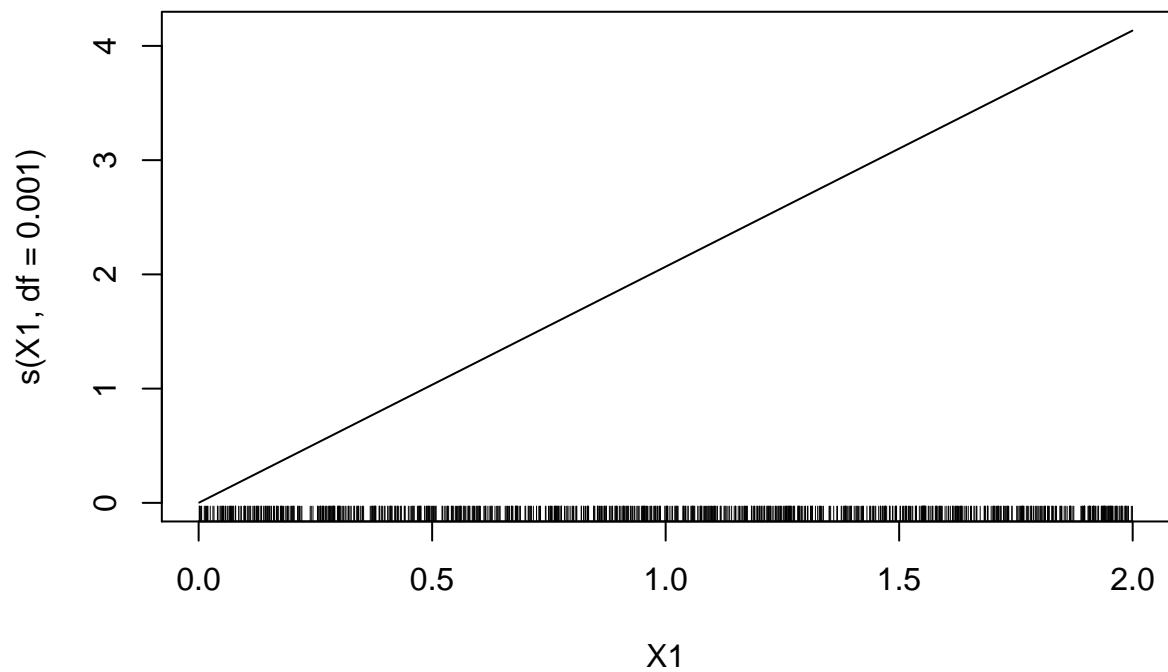
```
> library(gam)
> model1 <- gam(Y~s(X1,df=1)+s(X2,df=24.579)-1,data=donnees)
> plot(model1)
```



3. Faire varier les degrés de liberté, interpréter.

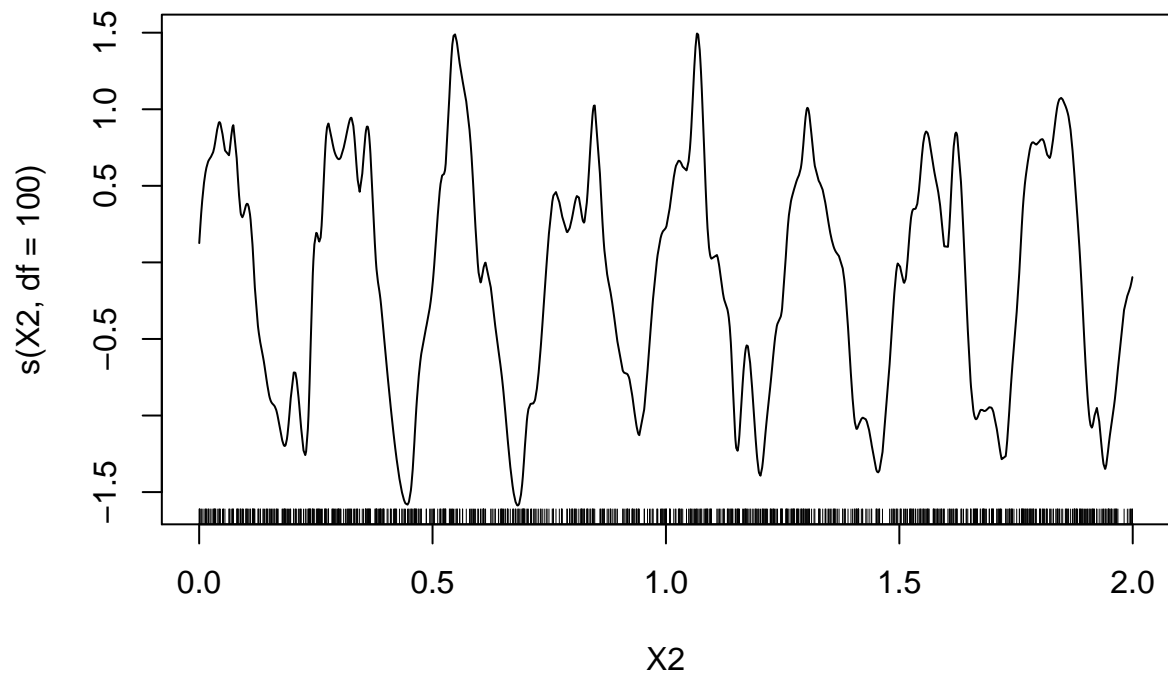
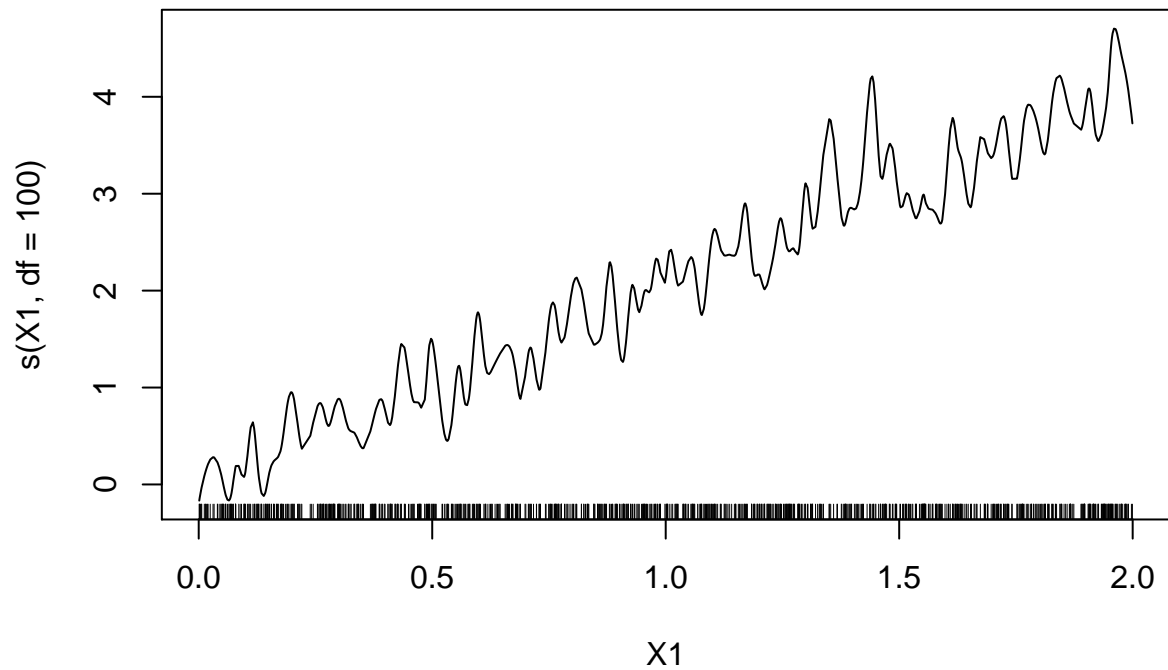
On prend d'abord peu de degrés de liberté.

```
> model2 <- gam(Y~s(X1,df=0.001)+s(X2,df=0.001)-1,data=donnees)
> plot(model2)
```



Le sinus n'est pas bien estimé. On prend maintenant un grand nombre de degrés de liberté.

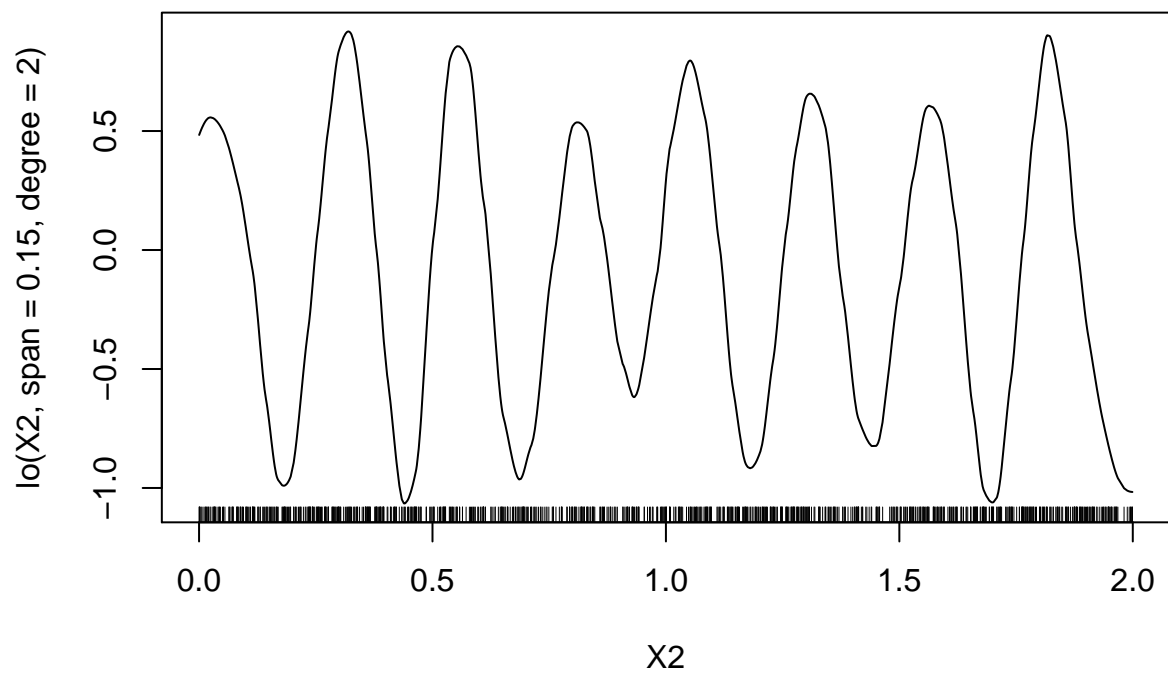
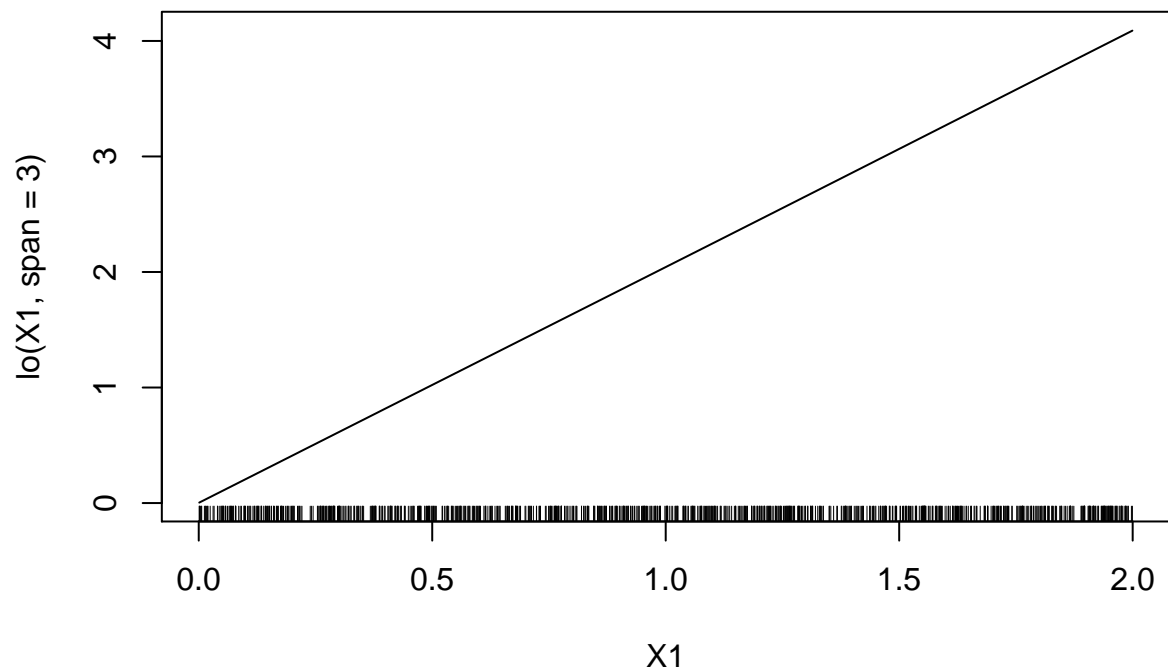
```
> model2 <- gam(Y~s(X1,df=100)+s(X2,df=100)-1,data=donnees)
> plot(model2)
```



Le modèle est trop flexible, risque de sur-ajustement.

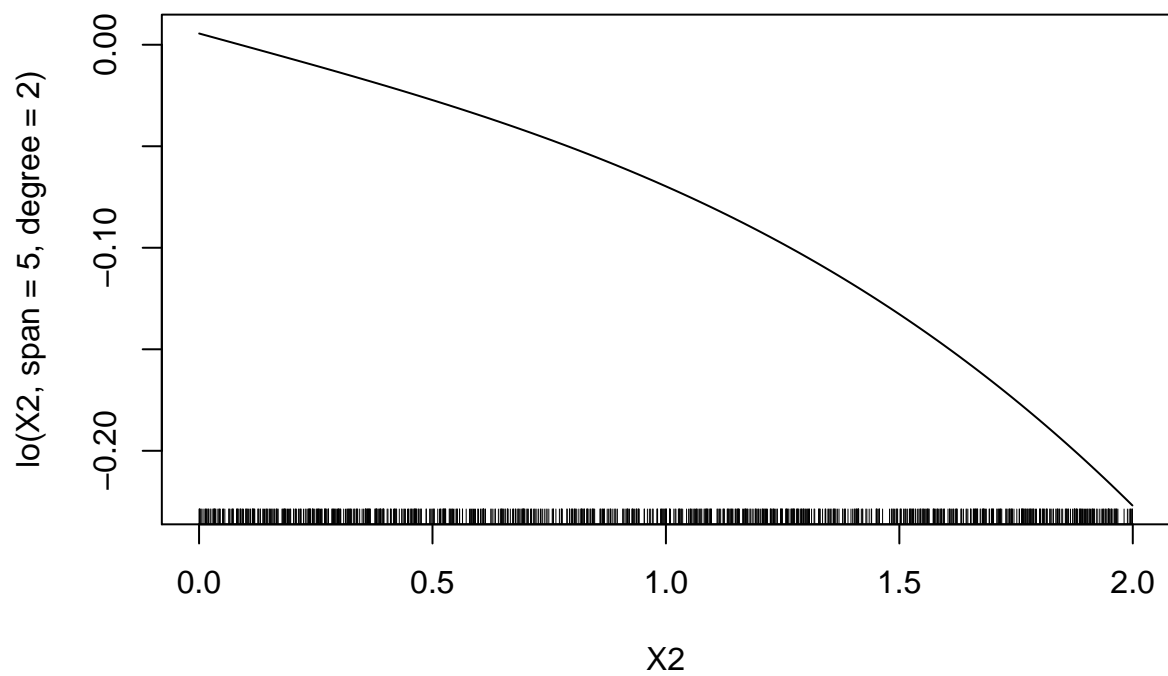
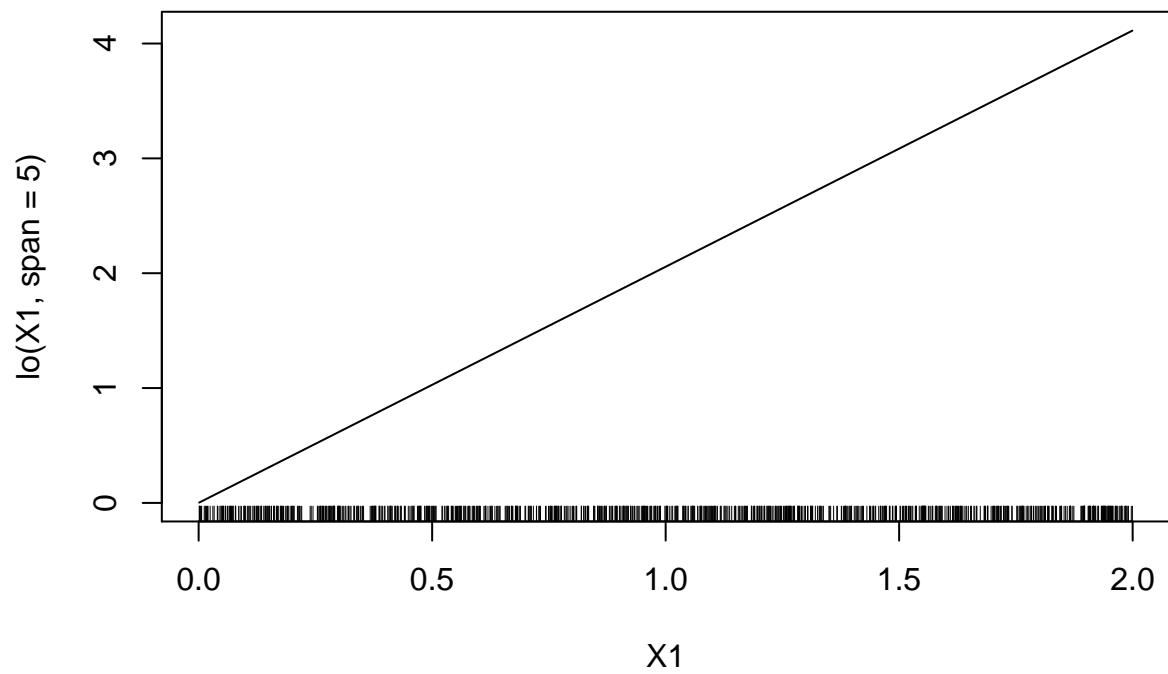
4. Faire le même travail avec le lisseur **loess**. On commencera avec **degree=2** et **span=0.15** puis on fera varier le paramètre **span**.

```
> model4 <- gam(Y~lo(X1,span=3)+lo(X2,span=0.15,degree=2)-1,data=donnees)
> plot(model4)
```

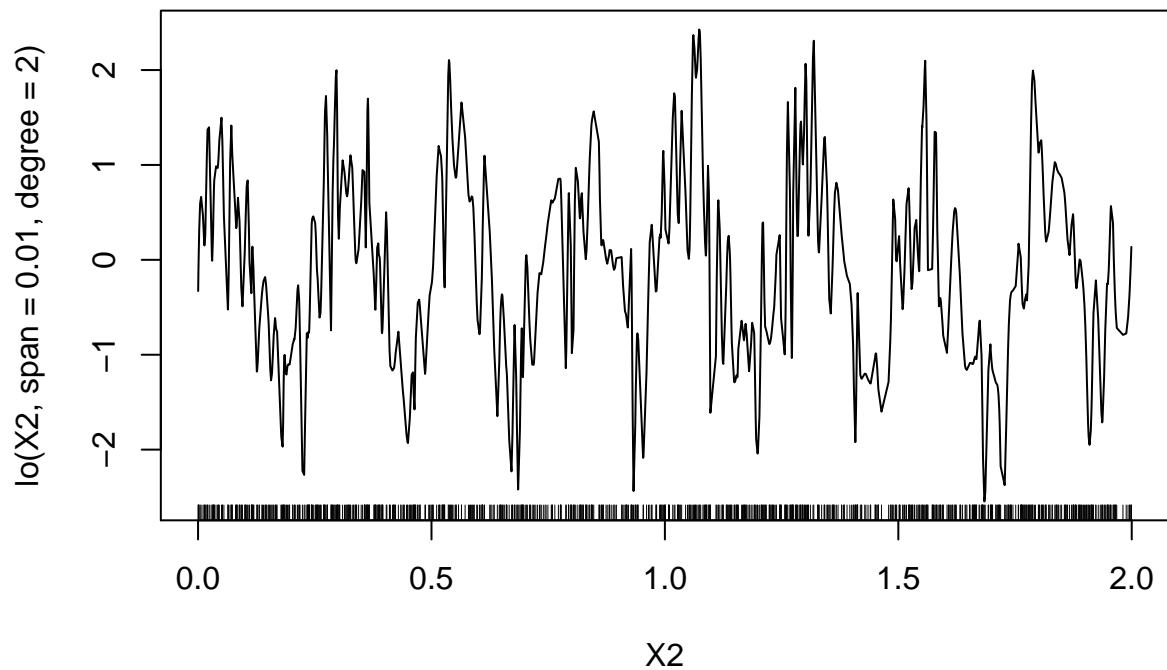
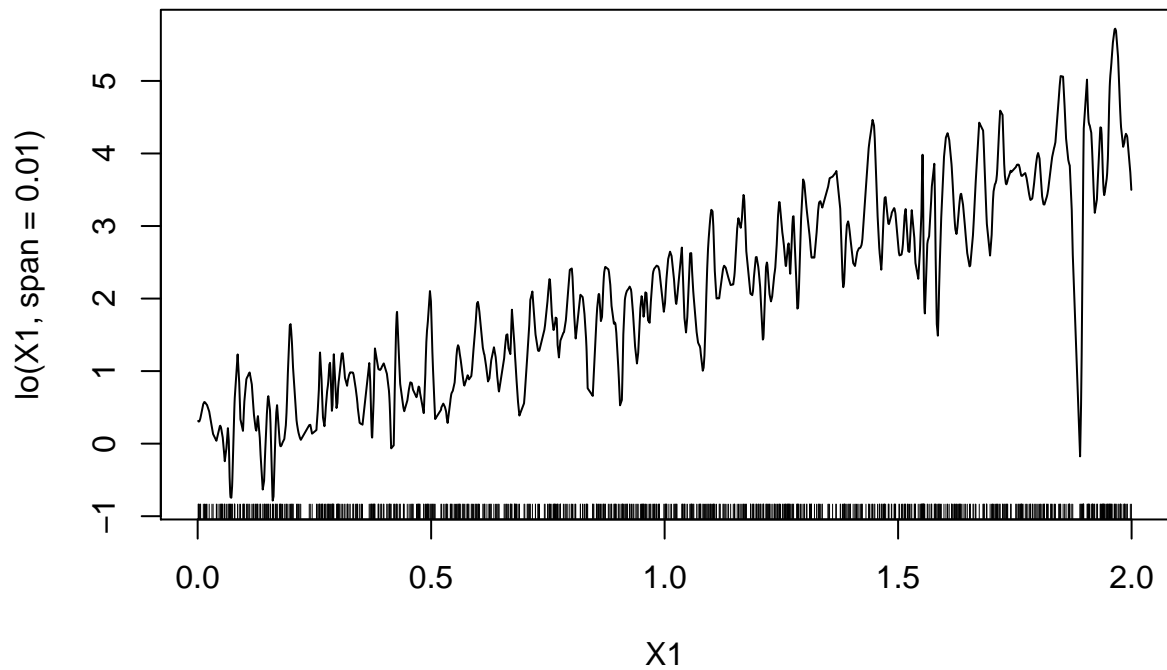


On fait varier `span` :

```
> model5 <- gam(Y~lo(X1,span=5)+lo(X2,span=5,degree=2)-1,data=donnees)
> plot(model5)
```



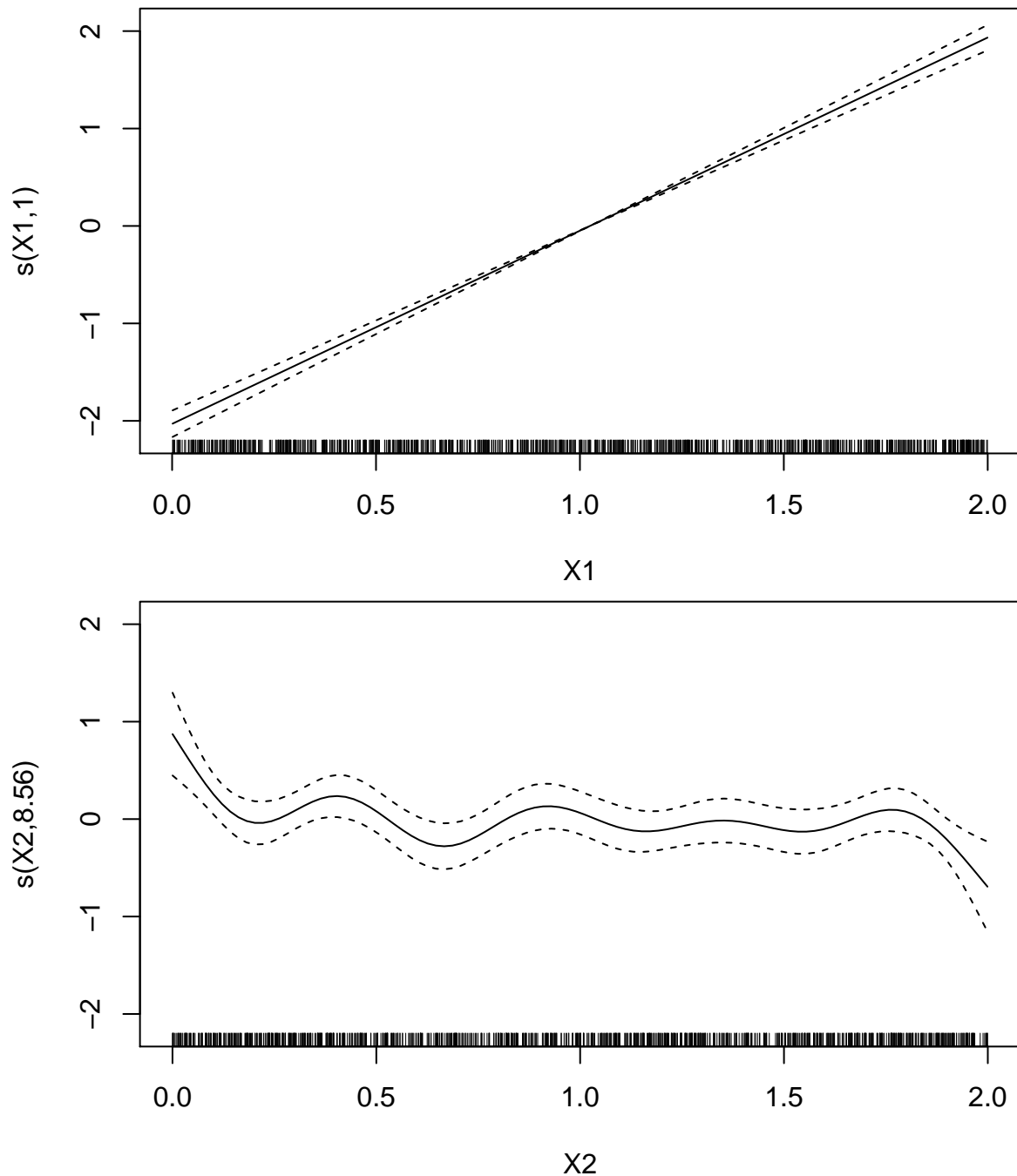
```
> model6 <- gam(Y~lo(X1,span=0.01)+lo(X2,span=0.01,degree=2)-1,data=donnees)
> plot(model6)
```



On a les mêmes remarques que pour les splines.

5. Estimer le degrés de liberté avec la fonction `gam` du package `mgcv` (Il n'est pas nécessaire de charger le package pour éviter les conflits).

```
> mod.mgcv <- mgcv::gam(Y~s(X1)+s(X2),data=donnees)
> plot(mod.mgcv)
```



4.3 Régression logistique additive

On considère le jeu de données **panne.txt** qui recense des pannes de machine (**etat=1**) en fonction de leur âge et de leur marque.

1. Faire une régression logistique permettant d'expliquer la variable **etat** par la variable **age** uniquement. Critiquer le modèle.

```
> panne <- read.table("data/panne.txt", header=TRUE)
> mod1 <- glm(etat~age, data=panne, family=binomial)
> summary(mod1)
##
```

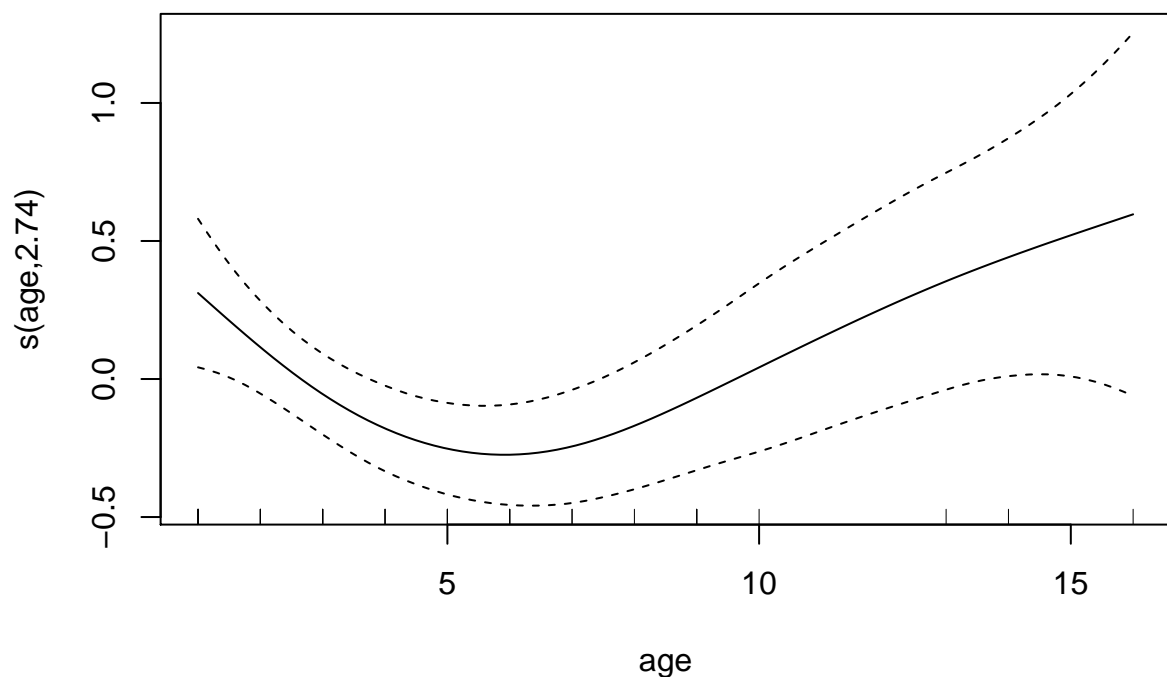


```
## Call:
## glm(formula = etat ~ age, family = binomial, data = panne)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.253  -1.199   1.015   1.183   1.210
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.10748    0.59864  -0.180   0.858
## age          0.03141    0.09117   0.345   0.730
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 45.717  on 32  degrees of freedom
## Residual deviance: 45.598  on 31  degrees of freedom
## AIC: 49.598
##
## Number of Fisher Scoring iterations: 3
```

Le modèle n'est pas pertinent. On accepte la nullité du coefficient *age*, ce qui signifie que le modèle constant est meilleur que le modèle avec la variable *age*.

2. Ajuster un modèle additif, toujours avec uniquement la variable *age*.

```
> mod.panne <- mgcv::gam(etat~s(age),data=panne)
> plot(mod.panne)
```



3. En utilisant le modèle additif, proposer un nouveau modèle logistique plus pertinent.

Il semble que l'âge agisse de façon quadratique. Cela peut s'expliquer par le fait que les pannes interviennent souvent au début (phase de rodage) et à la fin (vieillesse de la machine).

```
> mod2 <- glm(etat~age+I(age^2),data=panne,family=binomial)
> summary(mod2)
##
```

```
## Call:
## glm(formula = etat ~ age + I(age^2), family = binomial, data = panne)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.54043  -0.74739   0.00033   0.64877   1.88091
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  4.18501     1.73860   2.407  0.01608 *
## age         -2.03343     0.77401  -2.627  0.00861 **
## I(age^2)      0.17601     0.07044   2.499  0.01247 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 45.717  on 32  degrees of freedom
## Residual deviance: 31.279  on 30  degrees of freedom
## AIC: 37.279
##
## Number of Fisher Scoring iterations: 6
```

On remarque ici que l'âge devient "significatif"!