Statistique en grande dimension

Laurent Rouvière

13 décembre 2023

Table des matières

Présentation

Ce tutoriel présente quelques exercices d'application du cours `Modèle linéaire en grande dimension`. On pourra trouver

- les supports de cours associés à ce tutoriel ainsi que les données utilisées à l'adresse suivante https://lrouviere.github.io/page_perso/grande_dim.html ;
- le tutoriel sans les corrections à l'url https://lrouviere.github.io/TUTO_GRANDE_DIM/
- le tutoriel avec les corrigés (à certains moment) à l'url https://lrouviere.github.io/TUTO_GRANDE_DIM/correction.

Il est recommandé d'utiliser **mozilla firefox** pour lire le tutoriel.

Des connaissances de base en R et en statistique (modèles de régression) 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.

partie I

Supervisée

1 Les problèmes de 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'estimateur 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

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

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 voisins 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)
   k.opt
   ```

   ```
   [1] 31
   ```

   ```
   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
```

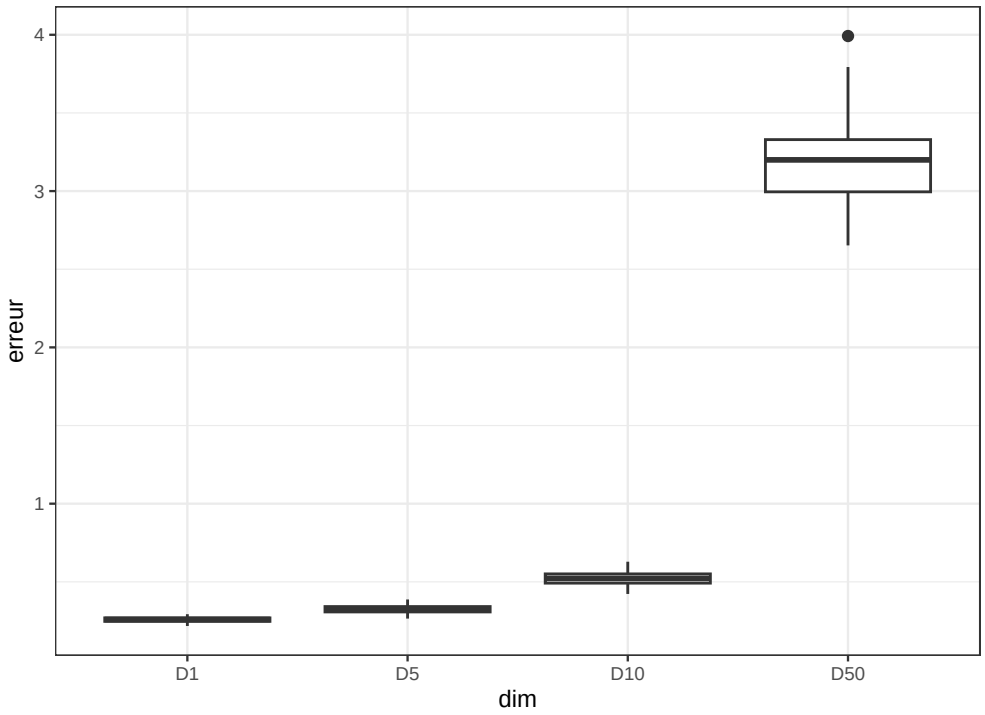
```
df |> summarise_all(mean)
```

```
       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
```

```r
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()
```



4. 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 linaire suivant

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

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 met en forme les résultats

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

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

- *en étudiant le biais et la variance*

```
df |> summarise_all(mean)
```

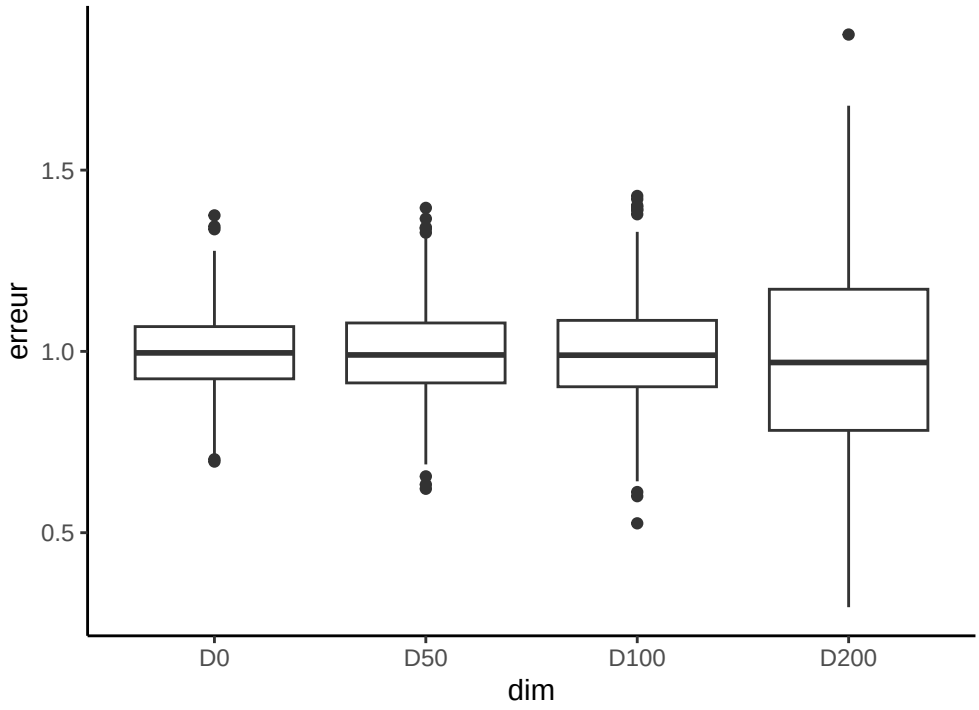
```
        D0       D50      D100     D200
1 0.992891 0.9960811 0.9959025 0.98173
```

```
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)**.** Cet exercice est fortement inspiré de Giraud (2015). 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

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

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

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

car

$$\mathbf{E}\left[(U'-U)^4\right] = \int_0^1 \int_0^1 (x-y)^4 \,\mathrm{d}x\mathrm{d}y = \frac{1}{15}$$

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 $\|\,.\,\|$ la norme Euclidienne dans $\mathbb{R}^d$. On définit l'estimateur localement constant de m en $x \in \mathbb{R}^d$ par :

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

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 $\widehat{m}(x)$.

   En annulant la dérivée par rapport à a, on obtient

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

2. Montrer que

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

   et

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

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 $i = 1, \ldots, n$

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

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

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

où γ_d désigne le volume de la boule unité en dimension d. On a donc

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

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

On déduit

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

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

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

2 Régression sur composantes

Les performances des estimateurs classiques (MCO) des paramètres du modèle linéaire

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

peuvent se dégrader lorsque la dimension d est grande ou en présence de dépendance linéaire entre les variables explicatives. Les régressions sur composantes consistent à trouver de nouvelles composantes $Z_k, j = k, \ldots, q$ avec $q \leq p$ qui s'écrivent le plus souvent comme des combinaisons linéaires des X_j dans l'idée de diminuer le nombre de paramètres du modèle ou la dépendance entre les covariables. Il existe plusieurs façons de construire ces composantes, dans cette partie nous proposons :

- la **régression sous composantes principales (PCR)** : il s'agit de faire simplement une ACP sur la matrice des variables explicatives ;
- la **régression partial least square (PLS)** qui fait intervenir la variable cible dans la construction des composantes.

Nous commençons par un bref rappel sur la sélection de variables.

2.1 Sélection de variables

On considère le jeu de données `ozone.txt` où on cherche à expliquer la concentration maximale en ozone relevée sur une journée (variable `maxO3`) 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
```

1. Ajuster un modèle linéaire avec `lm` et analyser la pertinence des variables explicatives dans le modèle.

```r
lin.complet <- lm(maxO3~.,data=ozone)
summary(lin.complet)
```

```
Call:
lm(formula = maxO3 ~ ., data = ozone)

Residuals:
    Min      1Q  Median      3Q     Max
-51.814  -8.695  -1.020   7.891  40.046

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 16.26536   15.94398   1.020   0.3102
T9           0.03917    1.16496   0.034   0.9732
T12          1.97257    1.47570   1.337   0.1844
T15          0.45031    1.18707   0.379   0.7053
Ne9         -2.10975    0.95985  -2.198   0.0303 *
Ne12        -0.60559    1.42634  -0.425   0.6721
Ne15        -0.01718    1.03589  -0.017   0.9868
Vx9          0.48261    0.98762   0.489   0.6262
Vx12         0.51379    1.24717   0.412   0.6813
Vx15         0.72662    0.95198   0.763   0.4471
maxO3v       0.34438    0.06699   5.141 1.42e-06 ***
ventNord     0.53956    6.69459   0.081   0.9359
ventOuest    5.53632    8.24792   0.671   0.5037
ventSud      5.42028    7.16180   0.757   0.4510
pluieSec     3.24713    3.48251   0.932   0.3534
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 14.51 on 97 degrees of freedom
Multiple R-squared:  0.7686,    Adjusted R-squared:  0.7352
F-statistic: 23.01 on 14 and 97 DF,  p-value: < 2.2e-16
```

```r
anova(lin.complet)
```

```
Analysis of Variance Table

Response: maxO3
          Df Sum Sq Mean Sq  F value    Pr(>F)
T9         1  43138   43138 205.0160 < 2.2e-16 ***
T12        1  11125   11125  52.8706 9.165e-11 ***
T15        1    876     876   4.1619 0.0440614 *
Ne9        1   3244    3244  15.4170 0.0001613 ***
Ne12       1    232     232   1.1035 0.2961089
Ne15       1      5       5   0.0248 0.8752847
Vx9        1   2217    2217  10.5355 0.0016079 **
Vx12       1      1       1   0.0049 0.9443039
Vx15       1     67      67   0.3186 0.5737491
maxO3v     1   6460    6460  30.6993 2.584e-07 ***
vent       3    234      78   0.3709 0.7741473
pluie      1    183     183   0.8694 0.3534399
Residuals 97  20410     210
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Il semble que quelques variables ne sont pas nécessaires dans le modèle.

2. Expliquer les sorties de la commande

```r
library(leaps)
mod.sel <- regsubsets(maxO3~.,data=ozone,nvmax=14)
summary(mod.sel)
```

```
Subset selection object
Call: regsubsets.formula(maxO3 ~ ., data = ozone, nvmax = 14)
14 Variables  (and intercept)
          Forced in Forced out
T9            FALSE      FALSE
T12           FALSE      FALSE
T15           FALSE      FALSE
Ne9           FALSE      FALSE
Ne12          FALSE      FALSE
```

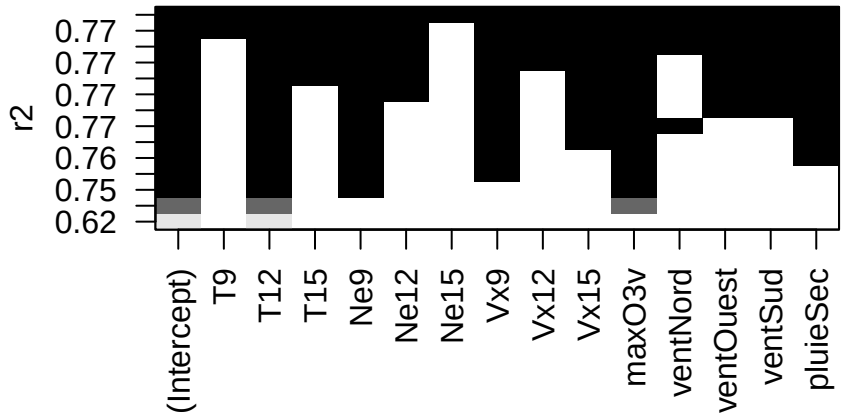
```
Ne15          FALSE      FALSE
Vx9           FALSE      FALSE
Vx12          FALSE      FALSE
Vx15          FALSE      FALSE
maxO3v        FALSE      FALSE
ventNord      FALSE      FALSE
ventOuest     FALSE      FALSE
ventSud       FALSE      FALSE
pluieSec      FALSE      FALSE
1 subsets of each size up to 14
Selection Algorithm: exhaustive
          T9  T12 T15 Ne9 Ne12 Ne15 Vx9 Vx12 Vx15 maxO3v ventNord ventOuest
1  ( 1 )  " " "*" " " " " " "  " "  " " " "  " "  " "     " "       " "
2  ( 1 )  " " "*" " " " " " "  " "  " " " "  " "  "*"     " "       " "
3  ( 1 )  " " "*" " " "*" " "  " "  " " " "  " "  "*"     " "       " "
4  ( 1 )  " " "*" " " "*" " "  " "  "*" " "  " "  "*"     " "       " "
5  ( 1 )  " " "*" " " "*" " "  " "  "*" " "  " "  "*"     " "       " "
6  ( 1 )  " " "*" " " "*" " "  " "  "*" " "  "*"  "*"     " "       " "
7  ( 1 )  " " "*" " " "*" " "  " "  "*" " "  "*"  "*"     "*"       " "
8  ( 1 )  " " "*" " " "*" " "  " "  "*" " "  "*"  "*"     " "       "*"
9  ( 1 )  " " "*" " " "*" "*"  " "  "*" " "  "*"  "*"     " "       "*"
10  ( 1 ) " " "*" "*" "*" "*"  " "  "*" " "  "*"  "*"     " "       "*"
11  ( 1 ) " " "*" "*" "*" "*"  " "  "*" "*"  "*"  "*"     " "       "*"
12  ( 1 ) " " "*" "*" "*" "*"  " "  "*" "*"  "*"  "*"     "*"       "*"
13  ( 1 ) "*" "*" "*" "*" "*"  " "  "*" "*"  "*"  "*"     "*"       "*"
14  ( 1 ) "*" "*" "*" "*" "*"  "*"  "*" "*"  "*"  "*"     "*"       "*"
          ventSud pluieSec
1  ( 1 )  " "     " "
2  ( 1 )  " "     " "
3  ( 1 )  " "     " "
4  ( 1 )  " "     " "
5  ( 1 )  " "     "*"
6  ( 1 )  " "     "*"
7  ( 1 )  " "     "*"
8  ( 1 )  "*"     "*"
9  ( 1 )  "*"     "*"
10  ( 1 ) "*"     "*"
11  ( 1 ) "*"     "*"
12  ( 1 ) "*"     "*"
13  ( 1 ) "*"     "*"
14  ( 1 ) "*"     "*"
```

On obtient une table avec des étoiles qui permettent de visualiser les meilleurs modèles

à $1, 2, \ldots, 8$ *variables au sens du* R^2.

3. Sélectionner le meilleur modèle au sens du R^2. Que remarquez-vous ?

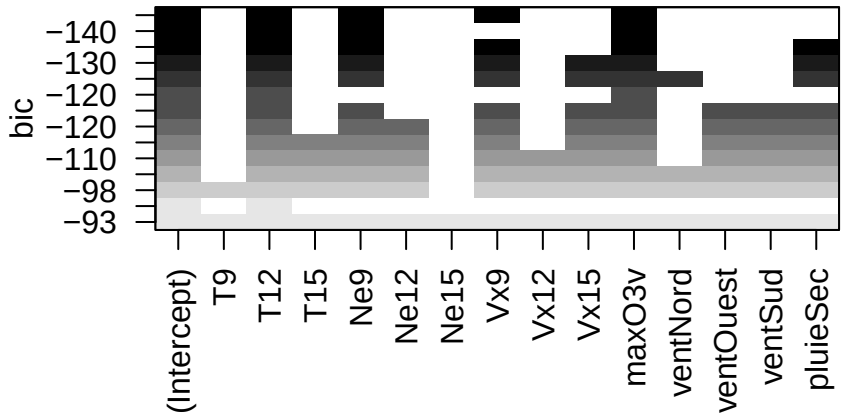
```
plot(mod.sel,scale="r2")
```



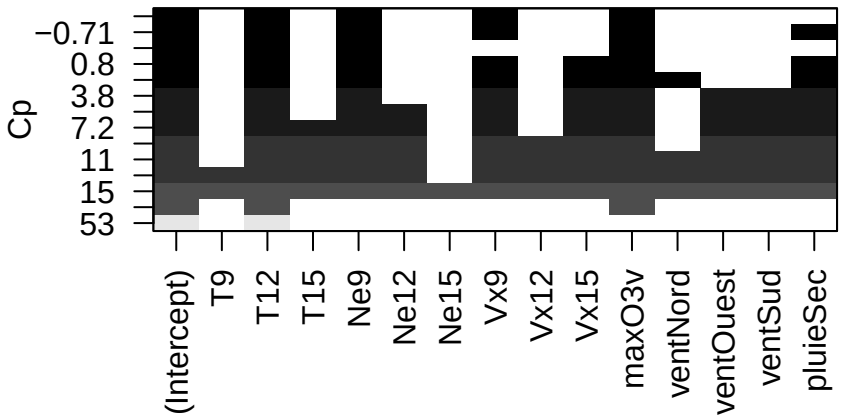
Le meilleur modèle est le modèle complet. C'est logique puisque le R^2 *va toujours privilégier le modèle le plus complexe, c'est un critère* `d'ajustement`.

4. Faire de même pour le C_p et le BIC. Que remarquez-vous pour les variables explicatives qualitatives ?

```
plot(mod.sel,scale="bic")
```




```
plot(mod.sel,scale="Cp")
```



Ces critères choisissent ici le même modèle, avec 4 variables. On remarque que les variables qualitatives ne sont ***pas réellement traitées comme des variables*** *: une modalité est égale à une variable. Par conséquent, cette procédure ne permet pas vraiment de sélectionner des variables qualitatives.*

5. Comparer cette méthode avec des modèles sélectionnées par la fonction `step` ou la fonction `bestglm` du package `bestglm`.

- *La fonction* ***step*** *permet de faire de la sélection* ***pas à pas****. Par exemple, pour une procédure* ***descendante*** *avec le critère AIC on utilisera :*

```
mod.step <- step(lin.complet,direction="backward",trace=0)
mod.step
```

La fonction ***bestglm*** *permet quant à elle de faire des sélections exhaustive ou pas à pas, on peut l'utiliser pour tous les* ***glm****. Attention les variables qualitatives doivent être des facteurs et la variable à expliquer doit être positionnée en dernière colonne pour cette fonction.*

```
ozone1 <- ozone |> mutate(vent=as.factor(vent),pluie=as.factor(pluie)) |>
  select(-maxO3,everything())
library(bestglm)
model.bglm <- bestglm(ozone1,IC="BIC")
model.bglm$BestModel |> summary()
```

```
Call:
lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),
    drop = FALSE], y = y))
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-52.396  -8.377  -1.086   7.951  40.933

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 12.63131   11.00088   1.148 0.253443
T12          2.76409    0.47450   5.825 6.07e-08 ***
Ne9         -2.51540    0.67585  -3.722 0.000317 ***
Vx9          1.29286    0.60218   2.147 0.034055 *
maxO3v       0.35483    0.05789   6.130 1.50e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14 on 107 degrees of freedom
Multiple R-squared:  0.7622,    Adjusted R-squared:  0.7533
F-statistic: 85.75 on 4 and 107 DF,  p-value: < 2.2e-16
```

2.2 Régression sur composantes principales (méthodo)

L'algorithme **PCP** est une méthode de réduction de dimension, elle consiste à faire un modèle linéaire **MCO** sur les premiers axes de l'**ACP**. On désigne par

- $\mathbb{X}$ la matrice qui contient les valeurs des variables explicatives que l'on suppose centrée réduite.
- $Z_1, \dots, Z_p$ les axes de l'ACP qui s'écrivent comme des combinaisons linéaires des variables explicatives : $Z_j = w_j^t X$.

L'algorithme **PCR** consiste à choisir un nombre de composantes m et à faire une régression MCO sur les m premiers axes de l'ACP :

$$Y = \alpha_0 + \alpha_1 Z_1 + \cdots + \alpha_m Z_m + \varepsilon.$$

Si on désigne par

- $x \in \mathbb{R}^d$ une nouvelle observation que l'on a centrée réduite également;
- $z_1, \dots, z_M$ les coordonnées de x dans la base définie par les m premiers axes de l'ACP ($z_j = w_j^t x$)

l'algorithme **PCR** reverra la prévision

$$\widehat{m}_{\mathrm{PCR}}(x) = \hat{\alpha}_0 + \hat{\alpha}_1 z_1 + \cdots + \hat{\alpha}_m z_m.$$

Cette prévision peut s'écrire également comme une combinaison linéaire des variables explicatives (centrées réduites ou non) :

$$\widehat{m}_{\text{PCR}}(x) = \hat{\gamma}_0 + \hat{\gamma}_1\tilde{x}_1 + \cdots + \hat{\gamma}_p\tilde{x}_p = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p,$$

$\tilde{x}_j$ désignant l'observation brute (non centrée réduite).

L'exercice suivant revient sur cet algorithme et notamment sur le lien entre ces différents paramètres.

Exercice 2.1 (Régression PCR avec R)**.** On considère le jeu de données **Hitters** dans lequel on souhaite expliquer la variable **Salary** par les autres variables du jeu de données. Pour simplifier le problème, on supprime les individus qui possèdent des données manquantes (il ne faut pas faire ça normalement !).

```
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]
   ```

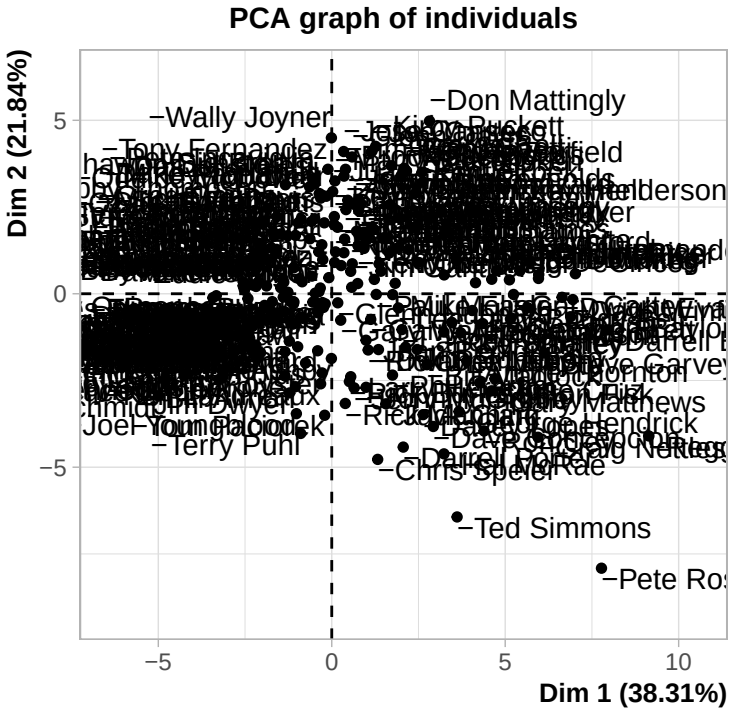
2. Calculer la matrice **Xcr** qui correspond à la matrice **X** centrée réduite. On pourra utiliser la fonction `scale`.

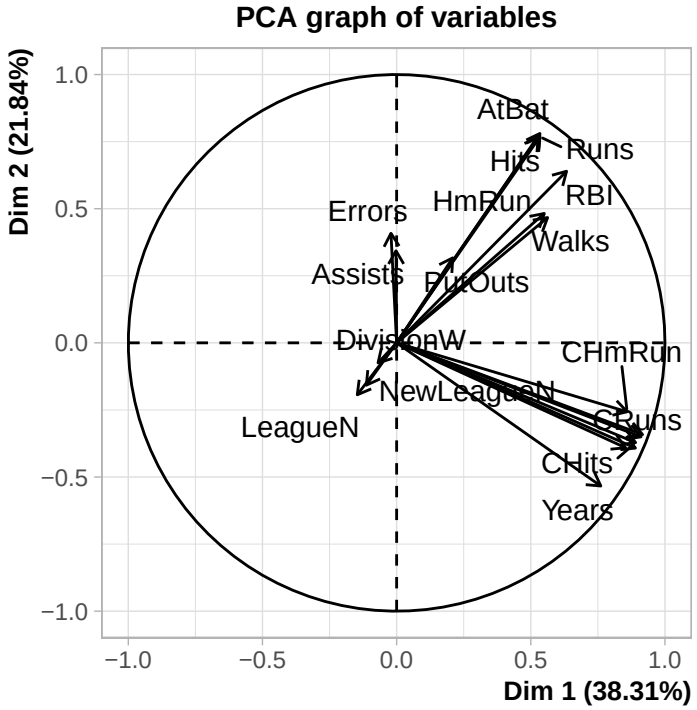
   ```
   Xcr <- scale(X)
   Xbar  <- apply(X,2,mean)
   stdX <- apply(X,2,sd)
   ```

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)
   ```





4. Récupérer les coordonnées des individus sur les 5 premiers axes de l'ACP (variables Z

dans le cours).

```
CC <- acp.hit$ind$coord
```

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

```
donnees <- cbind.data.frame(CC,Salary=Hitters$Salary)
mod <- lm(Salary~.,data=donnees)
alpha <- coef(mod)
alpha
```

```
(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 obtient ici les estimateurs des* $\alpha, j = 1, \dots, 5$.
- *on peut aussi tout faire "à la main" (sans utiliser* ***PCA****)*

```
acp.main <- eigen(t(Xcr)%*%Xcr)
U <- acp.main$vectors
CC <- Xcr%*%(-U[,1:5])
D <- cbind.data.frame(CC,Salary=Hitters$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 (w_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*

$$\hat{\beta}_0 = \bar{\mathbb{Y}} \quad et \quad \hat{\beta}_j = \sum_{k=1}^{m} \hat{\alpha}_k w_{kj}.$$

```
W <- acp.hit$svd$V
V <- t(W)
beta0.cr <- mean(Hitters$Salary)
beta.cr <- as.vector(alpha[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 :*

$$\hat{\gamma}_0 = \hat{\beta}_0 - \sum_{j=1}^{p} \hat{\beta}_j \mu_j \quad et \quad \hat{\gamma}_j = \frac{\hat{\beta}_j}{\sigma_j}.$$

```
gamma0 <- beta0.cr-sum(beta.cr*Xbar/stdX)
gamma <- beta.cr/stdX
gamma0
```

```
[1] -58.32022
```

```
gamma
```

```
          [,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
```

On remarque que la fonction ***PCR*** *renvoie les coefficients par rapport aux variables initiales centrées réduites. Cela fait du sens car il est dans ce cas possible de comparer les valeurs des estimateurs pour tenter d'interpréter le modèle. C'est beaucoup plus difficile à faire avec les coefficients des axes de l'ACP ou des variables intiales. Il est également important de noter que, contrairement aux estimateurs MCO du modèle linéaire Gaussien, on n'a pas d'information précise sur la loi des estimateurs, il n'est donc pas possible (ou pas facile) de faire des tests ou de calculer des intervalles de confiance.*

8. On considère les individus suivants

```r
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*** *:*

```r
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 :*

$$\widehat{m}_{PCR}(x) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p.$$

```
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 :*

$$\widehat{m}_{PCR}(x) = \hat{\gamma} + \hat{\gamma}_1 \tilde{x}_1 + \cdots + \hat{\gamma}_p \tilde{x}_p.$$

```
gamma0+gamma %*% t(as.matrix(X[c(1,100,80),]))
```

```
     -Alan Ashby -Hubie Brooks -George Bell
[1,]    495.0068      577.9581     822.0296
```

Exercice 2.2 (Composantes PCR)**.** On rappelle que les poids w_k des composantes principales s'obtiennent en résolvant le problème :

$$\max_{w \in \mathbb{R}^d} \mathbf{V}(\mathbb{X}w)$$

$$\text{sous les contraintes } \|w\| = 1, w^t \mathbb{X}^t \mathbb{X} w_\ell = 0, \ell = 1, \ldots, k-1.$$

1. Montrer w_1 est un vecteur propre associé à la plus grande valeur propre de $\mathbb{X}^t\mathbb{X}$.

On écrit le Lagrangien

$$L(w, \lambda) = w^t \mathbb{X}^t \mathbb{X} w - \lambda(w^t w - 1).$$

et on le dérive par rapport à w :

$$\frac{\partial L}{\partial w}(w,\lambda) = 2\mathbb{X}^t\mathbb{X}w - 2\lambda w.$$

En annulant cette dérivée, on déduit que w_1 est un vecteur propre de $\mathbb{X}^t\mathbb{X}$. De plus, si w est vecteur propre unitaire de $\mathbb{X}^t\mathbb{X}$ associé à la valeur propre λ on a $\mathbf{V}(\mathbb{X}w) = \lambda$. On déduit que w_1 est un vecteur propre associé à la plus grande valeur propre de $\mathbb{X}^t\mathbb{X}$.

2. Calculer w_2.

On écrit le Lagrangien

$$L(w,\lambda,\mu) = w^t\mathbb{X}^t\mathbb{X}w - \lambda(w^tw - 1) - \mu w^t\mathbb{X}^t\mathbb{X}w_1$$

et on calcule les dérivées partielles :

$$\frac{\partial L}{\partial w}(w,\lambda,\mu) = 2\mathbb{X}^t\mathbb{X}w - 2\lambda w - \mu\mathbb{X}^t\mathbb{X}w_1.$$

$$\frac{\partial L}{\partial \lambda}(w,\lambda,\mu) = w^tw - 1 \quad \text{et} \quad \frac{\partial L}{\partial \mu}(w,\lambda,\mu) = -w^t\mathbb{X}^t\mathbb{X}w_1.$$

En multipliant la première dérivée partielle par w_1^t et en utilisant le fait que W_1 est un vecteur propre de $\mathbb{X}^t\mathbb{X}$, on déduit que $\mu = 0$. Par conséquent, w_2 est un vecteur propre associé à la deuxième plus grande valeur propre de $\mathbb{X}^t\mathbb{X}$.

2.3 Régression PLS : méthodo

La régression **PLS** propose de construire également de nouvelles composantes comme des combinaisons linéaires des variables explicatives. Comme pour l'algorithme **PCR**, les composantes sont calculées les unes après les autres et orthogonales entre elles. La principale différence et qu'on ne cherche pas les composantes qui maximisent la variabilités des observations projetées, mais les composantes qui maximisent la colinéarité avec la cible. L'algorithme est expliqué dans l'exercice suivant.

Exercice 2.3 (Calcul des composantes PLS)**.** On reprend les notations du cours : $\mathbb{Y}$ désigne le vecteur de la variable à expliquer et $\mathbb{X}$ la matrice qui contient les observations des variables explicatives. On la suppose toujours centrée réduite.

1. On pose $\mathbb{Y}^{(1)} = \mathbb{Y}$ et $\mathbb{X}^{(1)} = \mathbb{X}$. On cherche $Z_1 = w_1^tX^{(1)}$ qui maximise

$$\langle \mathbb{X}^{(1)}w_1, \mathbb{Y}^{(1)}\rangle \quad \text{sous la contrainte} \quad \|w\|^2 = 1.$$

Cela revient à cherche la combinaison linéaire des colonnes de $\mathbb{X}^{(1)}$ la plus corrélée à $\mathbb{Y}^{(1)}$. Calculer cette première composante.

On écrit le lagrangien

$$L(x, \lambda) = {\mathbb{Y}^{(1)}}^t \mathbb{X}^{(1)} w_1 - \frac{1}{2}\lambda(\|w_1\|^2 - 1)$$

En dérivant par rapport à w et λ on obtient les équations

$$\begin{cases} {\mathbb{X}^{(1)}}^t \mathbb{Y}^{(1)} - \lambda w_1 = 0 \\ \|w_1\|^2 = 1 \end{cases}$$

La solution est donnée par

$$w_1 = \frac{{\mathbb{X}^{(1)}}^t \mathbb{Y}^{(1)}}{\|{\mathbb{X}^{(1)}}^t \mathbb{Y}^{(1)}\|}.$$

2. On pose $Z_1 = w_1^t X^{(1)}$ et $\mathbb{Z}_1 = \mathbb{X}^{(1)} w_1$. On considère le modèle de régression linéaire

$$Y^{(1)} = \alpha_0 + \alpha_1 Z_1 + \varepsilon.$$

Exprimer les estimateurs MCO de $\alpha = (\alpha_0, \alpha_1)$ en fonction de $\mathbb{Z}^{(1)}$ et $\mathbb{Y}^{(1)}$.

On déduit

$$\hat{\alpha}_0 = \bar{\mathbb{Y}}^{(1)} - \hat{\alpha}_1 \bar{\mathbb{Z}}_1 = \bar{\mathbb{Y}}^{(1)}$$

car $\bar{\mathbb{Z}}_1 = 0$ puisque $\mathbb{X}^{(1)}$ est centrée. Le second estimateur s'obtient par

$$\hat{\alpha}_1 = \frac{\langle \mathbb{Z}_1, \mathbb{Y}^{(1)} \rangle}{\langle \mathbb{Z}_1, \mathbb{Z}_1 \rangle}.$$

3. On passe maintenant à la deuxième composante. On cherche à expliquer la partie résiduelle

$$\mathbb{Y}^{(2)} = P_{Z_1^\perp}(\mathbb{Y}^{(1)}) = \hat{\varepsilon}_1 = \mathbb{Y}^{(1)} - \hat{\mathbb{Y}}^{(1)}$$

par la "meilleure" combinaison linéaire orthogonale à Z_1. On orthogonalise chaque $\tilde{\mathbb{X}}_j^{(1)}$ par rapport à $\mathbb{Z}_1$:

$$\mathbb{X}_j^{(2)} = P_{\mathbb{Z}_1^\perp}(\mathbb{X}_j^{(1)}) = (\mathrm{Id} - P_{\mathbb{Z}_1})(\mathbb{X}_j^{(1)}) = \mathbb{X}_j^{(1)} - \frac{\langle \mathbb{Z}_1, \mathbb{X}_j^{(1)} \rangle}{\langle \mathbb{Z}_1, \mathbb{Z}_1 \rangle} \mathbb{Z}_1.$$

et on déduit w_2 comme w_1 : $w_2 = \tilde{\mathbb{X}}^{(2)'} \mathbb{Y}^{(2)}$. On considère ensuite le modèle $Y^{(2)} = \alpha_2 Z_2 + \varepsilon$. Exprimer l'estimateur des MCO de α_2 en fonction de $\mathbb{Z}_2 = \mathbb{X}^{(2)} w_2$ et $\mathbb{Y}$.

On a

$$\hat{\alpha}_2 = \frac{\langle \mathbb{Z}_2, \mathbb{Y}^{(2)} \rangle}{\langle \mathbb{Z}_2, \mathbb{Z}_2 \rangle} = \frac{\langle \mathbb{Z}_2, \mathbb{Y} - \hat{\mathbb{Y}}^{(1)} \rangle}{\langle \mathbb{Z}_2, \mathbb{Z}_2 \rangle} = \frac{\langle \mathbb{Z}_2, \mathbb{Y} \rangle}{\langle \mathbb{Z}_2, \mathbb{Z}_2 \rangle}$$

car $\widehat{\mathbb{Y}}^{(1)} = \widehat{\alpha}_0 + \widehat{\alpha}_1 \mathbb{Z}_1$ est orthogonal à $\mathbb{Z}_2$.

Exercice 2.4 (Régression PLS sur R)**.** On considère les mêmes données que précédemment.

1. A l'aide du vecteur $\mathbb{Y}$ (*Salary*) et de la matrice des $\mathbb{X}$ centrées réduites calculées dans l'Exercice 2.1, calculer la première composante **PLS** $\mathbb{Z}_1$.

```
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
```

2.4 Comparaison : PCR vs PLS.

1. Séparer le jeu de données (`Hitters` toujours) 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)
ncomp.pcr
```

```
[1] 4
```

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

```
[1] 3
```

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 à l'aide d'une validation croisée 10 blocs.

   Attention *il ne s'agit pas ici de sélectionner les nombres de composantes par validation croisée. On veut comparer :*

   - *l'algorithme* ***PCR*** *qui sélectionne le nombre de composantes par validation croisée à*
   - *l'algorithme* ***PLS*** *qui sélectionne le nombre de composantes par validation croisée.*

   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
```

   Puis on fait la validation croisée (en sélectionnant le nombre de composantes par validation croisée) à chaque étape :

```
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 <- plsr(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 <- 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.

3 Régressions pénalisées (ou sous contraintes)

Nous considérons toujours le modèle linéaire

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

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 (variable `maxO3`) 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
   ```

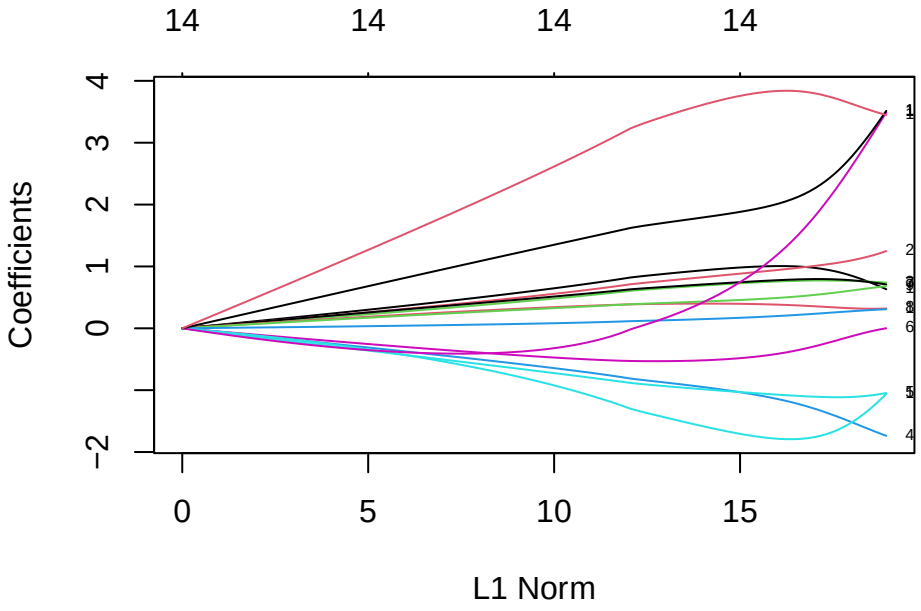
   On peut récupérer les valeurs de ***`beta`*** *associées à chaque valeur de la grille avec*

   ```
   mod.R$beta[,1]
   ```

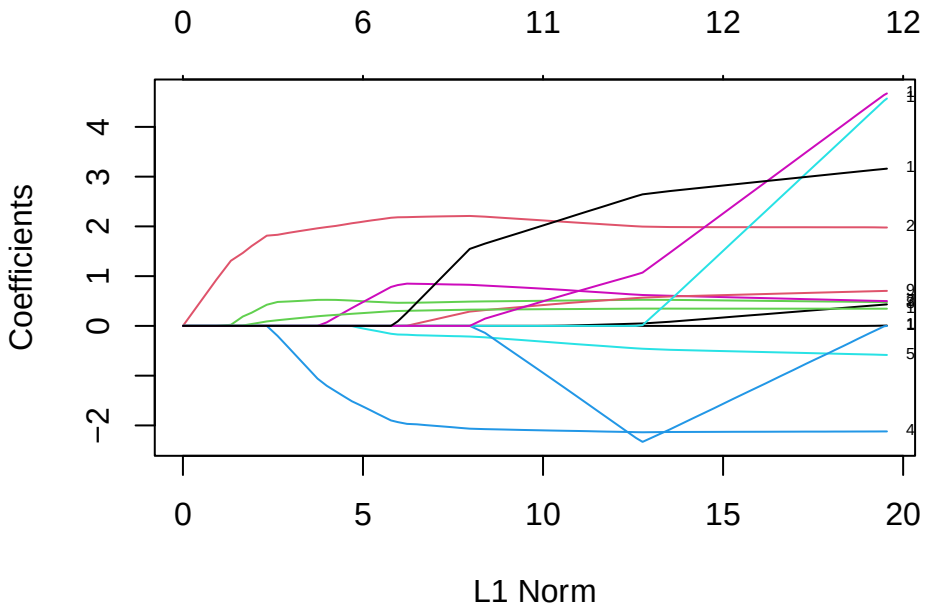
```
          T9           T12           T15           Ne9          Ne12
6.376767e-36  5.523924e-36  4.867402e-36 -6.821464e-36 -7.994984e-36
         Ne15           Vx9          Vx12          Vx15         maxO3v
-5.839057e-36  5.706014e-36  4.387350e-36  3.970583e-36  6.892387e-37
     ventNord     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`.

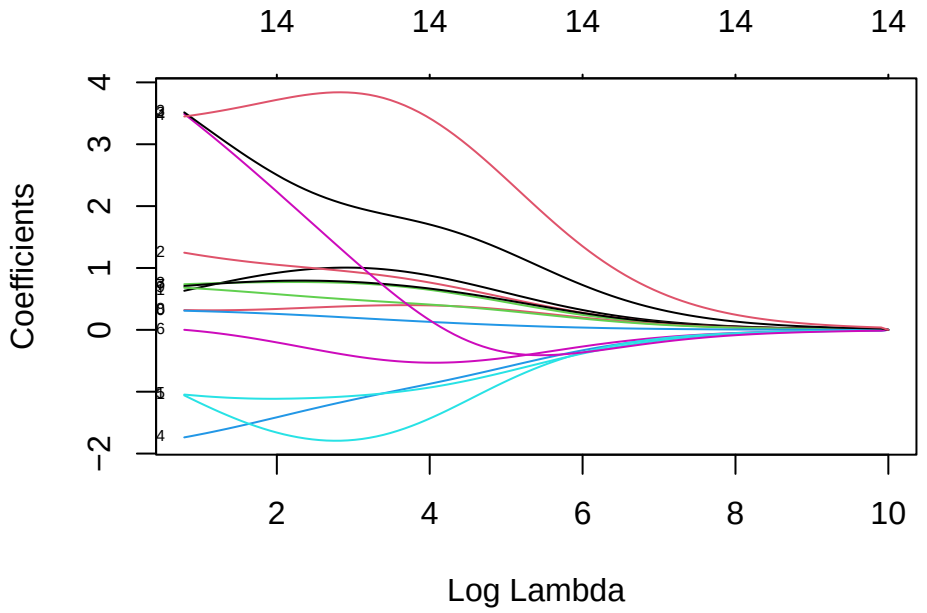
```
plot(mod.R,label=TRUE)
```



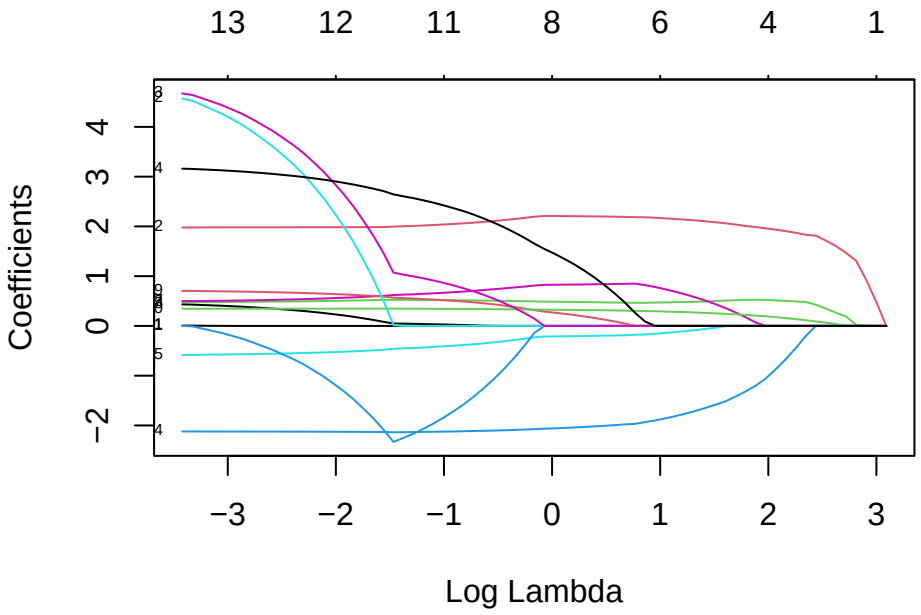

```
plot(mod.L,label=TRUE)
```




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



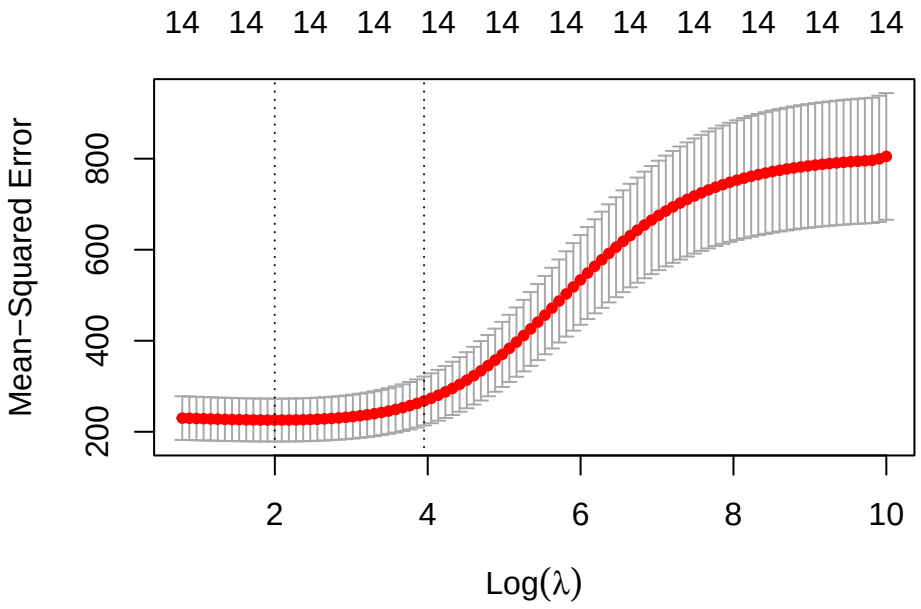

```
plot(mod.L,xvar="lambda",label=TRUE)
```



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 `lambda` *(échelle logarithmique). Deux traites verticaux sont représentés :*

- *celui de gauche correspond à la valeur de 'lambda' qui minimise l'erreur quadratique ;*
- *celui de droite correspond à la plus grande valeur de 'lambda' 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 `lambda` *entre les deux traits verticaux. Si on veut diminuer la complexité du modèle on choisira la valeur de droite. On peut obtenir ces deux valeurs avec*

```
ridgeCV$lambda.min
```

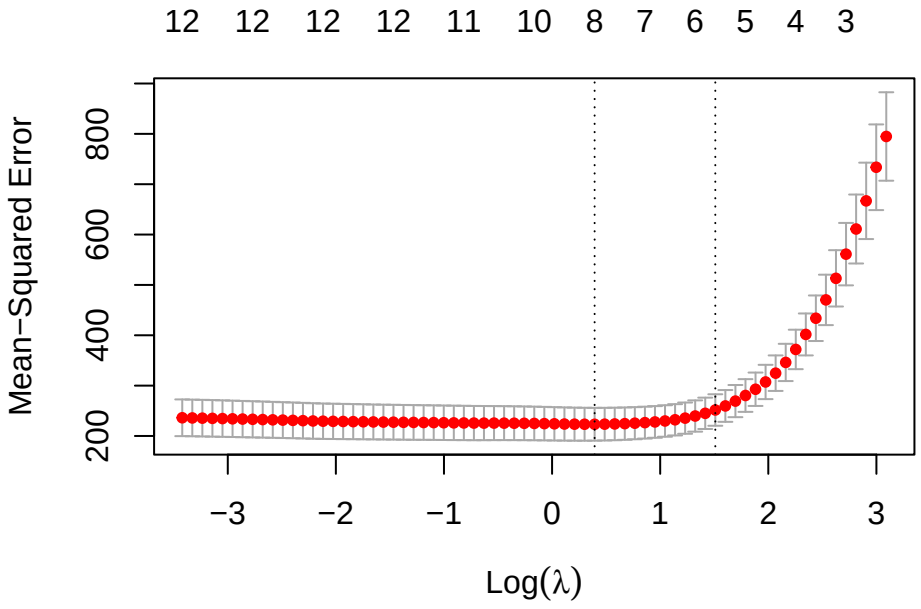
```
[1] 7.375962
```

```
ridgeCV$lambda.1se
```

```
[1] 52.03595
```

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, par exemple les 25ème et 50ème individus du jeu de données. Calculer les valeurs prédites pour ces deux individus.

   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")
```

```
         lambda.min
20010723   89.39929
20010724   96.81660
```

```
predict(ridgeCV,newx = ozone.X[50:51,],s="lambda.1se")
```

```
         lambda.1se
20010723   93.42750
20010724   96.08038
```

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

```
predict(lassoCV,newx = ozone.X[50:51,],s="lambda.min")
```

```
         lambda.min
20010723   87.19354
20010724   97.97744
```

```
predict(lassoCV,newx = ozone.X[50:51,],s="lambda.1se")
```

```
         lambda.1se
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 :

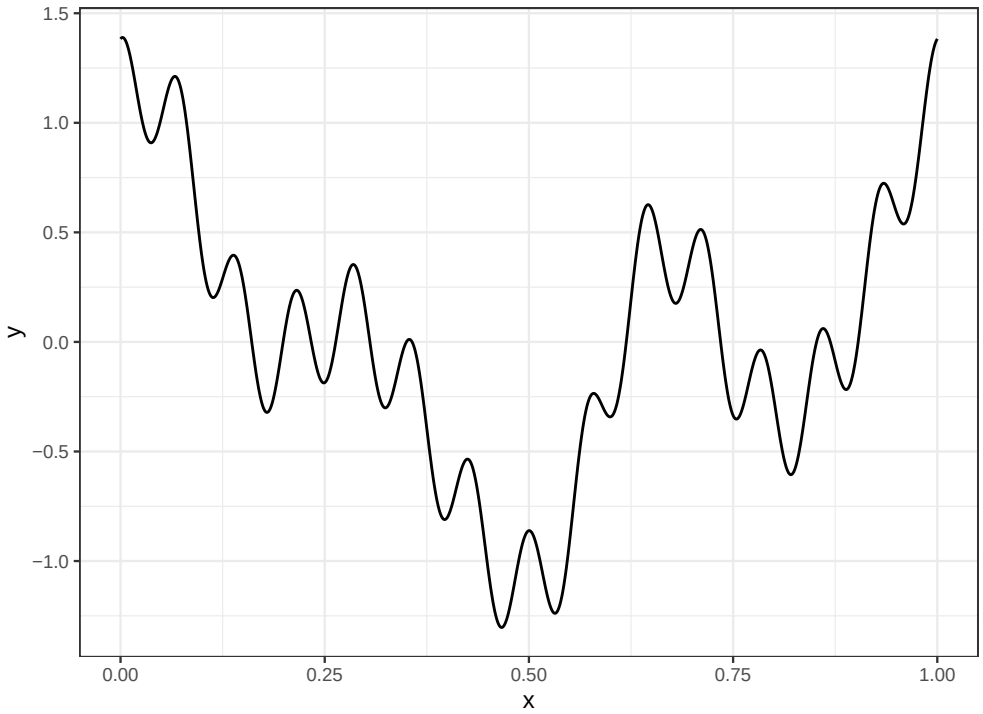
```
ozone2.X <- model.matrix(maxO3~.^2,data=ozone1)[,-1]
dim(ozone2.X)
```

```
[1] 1366  253
```

3.2 Reconstruction d'un signal

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

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

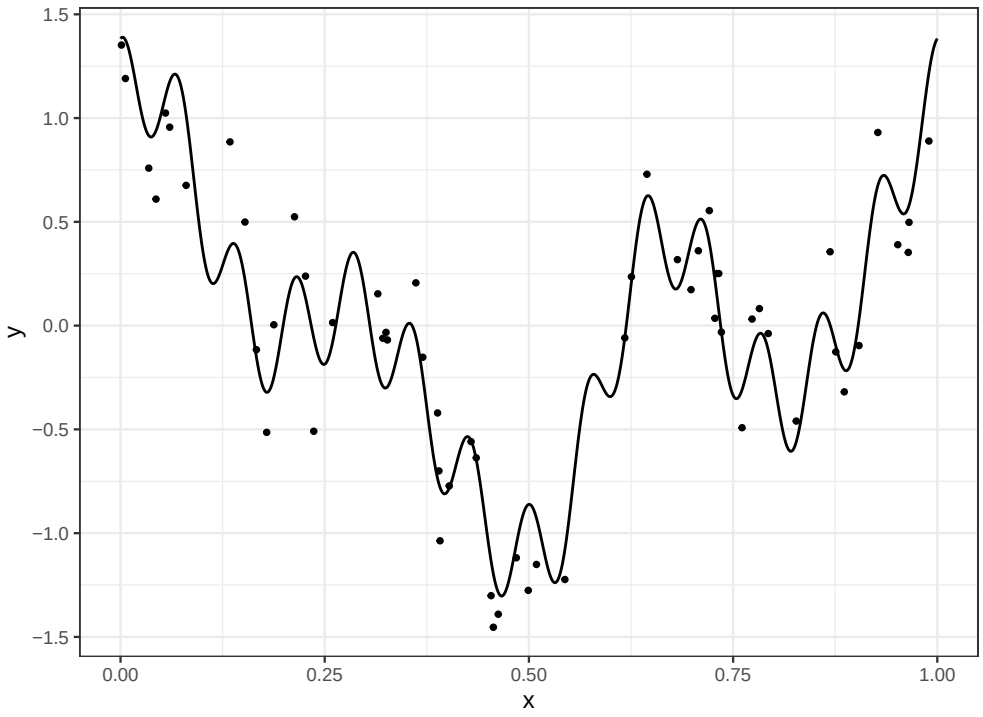


Plaçons nous dans le cas où on ne dispose que d'une version bruitée de ce signal. La courbe n'est pas observée mais on dispose d'un échantillon $(x_i, y_i), i = 1, \dots, n$ généré selon le modèle

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

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**

$$y_i = \sum_{j=1}^{K} \beta_j \psi_j(x_i) + \varepsilon_i. \tag{3.1}$$

Le problème est toujours d'estimer les paramètres β_j mais les variables sont maintenant définies par les élements 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)
}
```

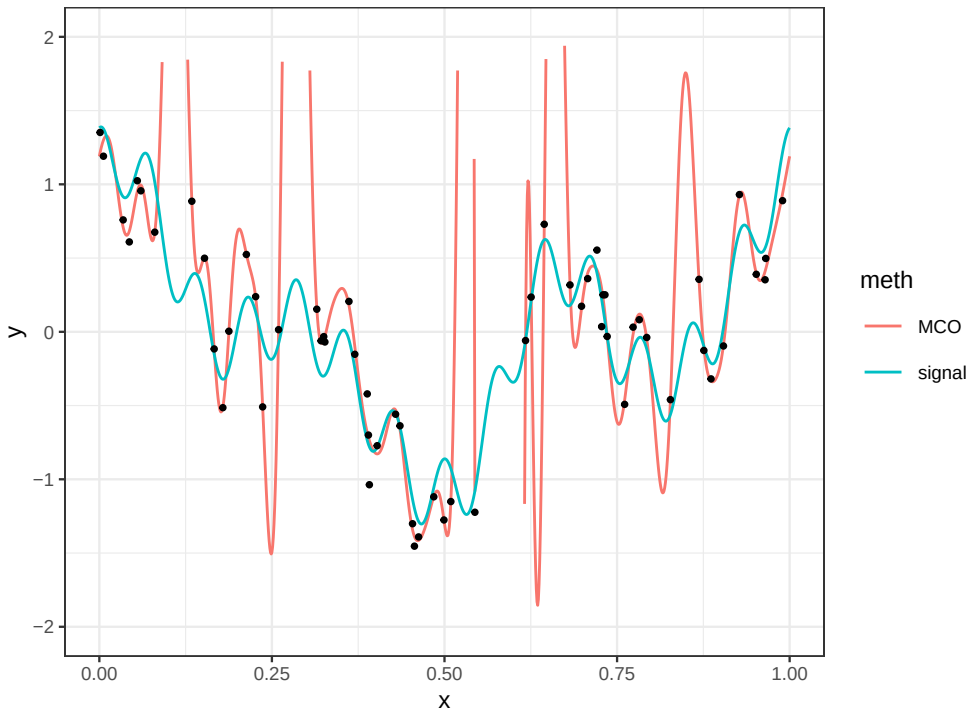
2. On fixe `K=25`. Calculer les estimateurs des moindres carrés du modèle (Équation 3.1).

   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))
```

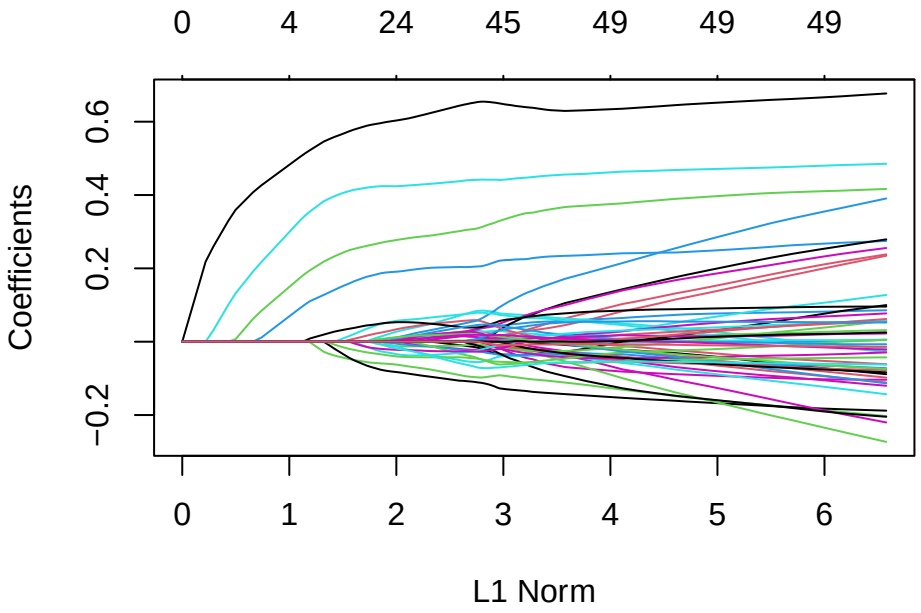


Le signal estimé a tendance à surajuster les données. Cela vient du fait que on estime 51 paramètres avec seulement 60 observations.

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

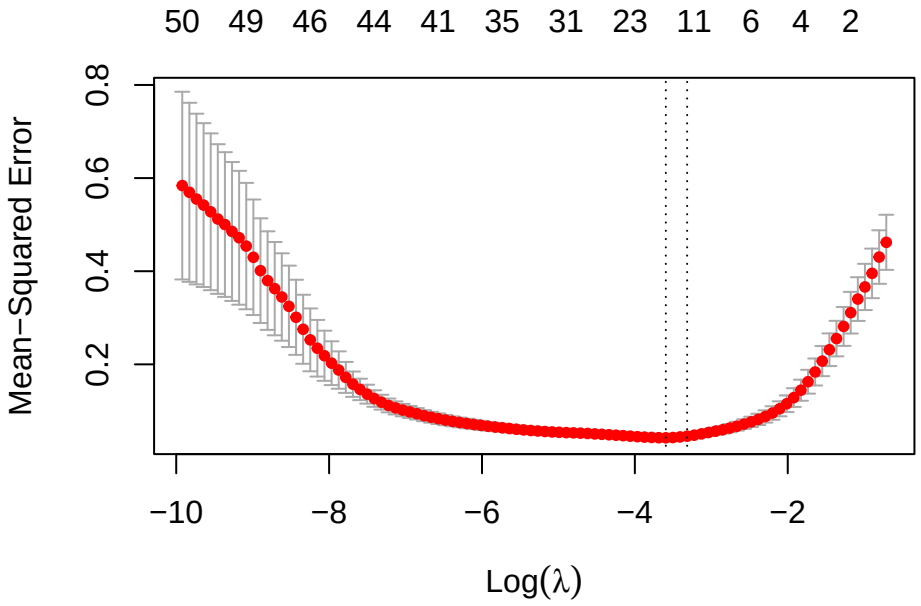
   On regarde tout d'abord le ***chemin de régularisation*** *des estimateurs* ***lasso***

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



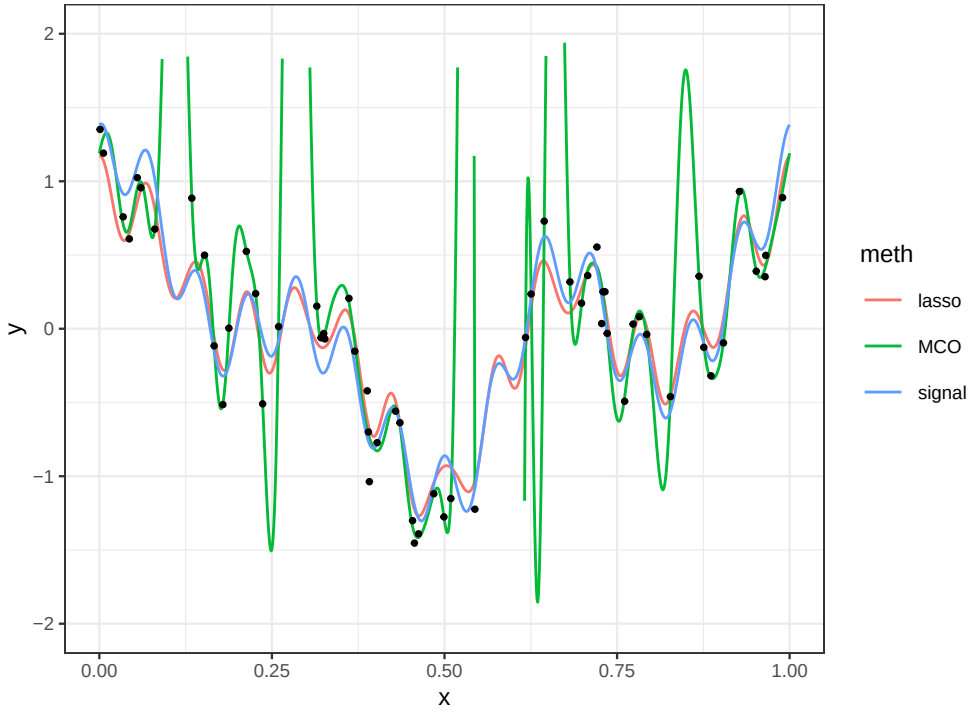
Il semble que quelques coefficients quittent la valeur 0 bien avant les autres. On effectue maintenant la validation croisée pour sélectionner le paramètre λ.

```
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
```

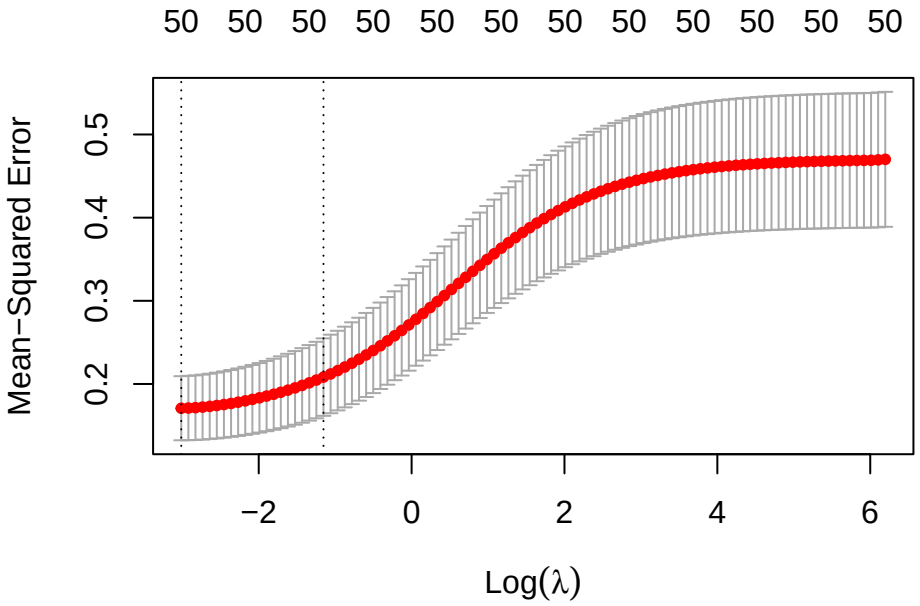
```
[1]  1  2  4  5  6  8 21 28 30 36 37 38 40
```

6. Ajouter les signaux ajustés par les algorithme Ridge, PCR et PLS. Comparer les performances.

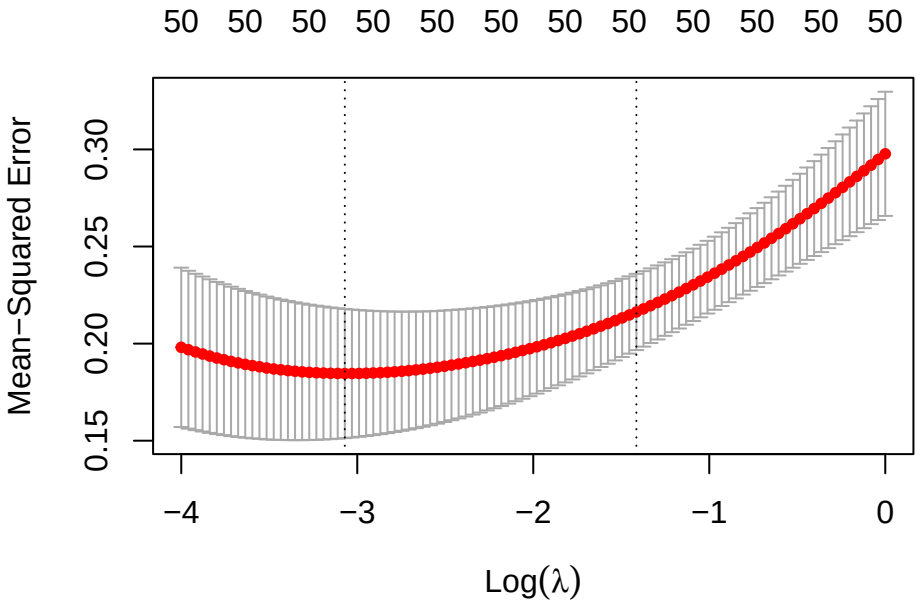
   - *On commence par la* ***régression ridge*** *:*

```
set.seed(1234)
ridge.cv <- cv.glmnet(X.25,D25$Y,alpha=0)
```

```r
plot(ridge.cv) # on est en bord de grille
```




```r
ridge.cv <- cv.glmnet(X.25,D25$Y,alpha=0,lambda = exp(seq(-4,0,length=100)))
plot(ridge.cv)
```



```
prev.ridge <- as.vector(predict(ridge.cv,newx=as.matrix(S25)))
```

- *On effectue la PCR :*

```
library(pls)
pcr.fit <- pcr(Y~.,data=D25,validation="CV")
ncomp.pcr <- which.min(pcr.fit$validation$PRESS)
ncomp.pcr
```

```
[1] 39
```

```
prev.pcr <- predict(pcr.fit,newdata=S25,ncomp=ncomp.pcr)
```

- *Puis la PLS :*

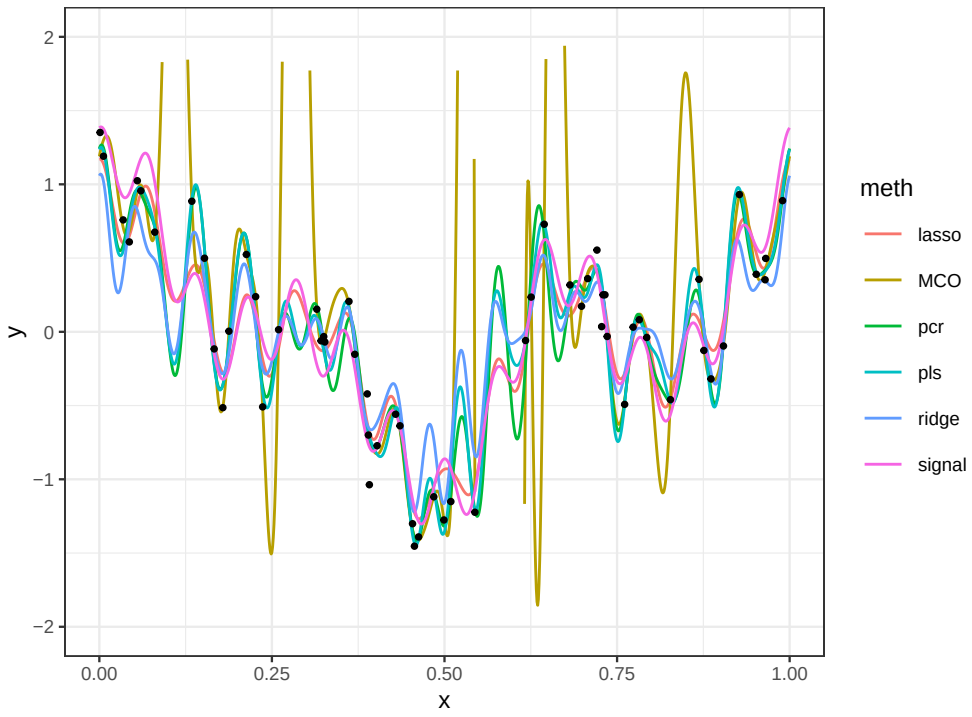
```
pls.fit <- plsr(Y~.,data=D25,validation="CV")
ncomp.pls <- which.min(pls.fit$validation$PRESS)
ncomp.pls
```

```
[1] 5
```

```
prev.pls <- predict(pls.fit,newdata=S25,ncomp=ncomp.pls)
```

- *On trace les signaux :*

```
signal1 <- signal1 |> mutate(ridge=prev.ridge,
                             pcr=prev.pcr,
                             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 (puisqu'on connait la vraie courbe)

```
signal1 |> summarise_at(-(1:2),~mean((.-signal)^2)) |> round(3)
```

```
# A tibble: 1 x 5
    MCO lasso ridge   pcr   pls
  <dbl> <dbl> <dbl> <dbl> <dbl>
1  598. 0.014 0.089  0.07 0.067
```

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_delim("data/ad_data.txt",delim=",",
                      col_names = FALSE,na=c("?"),trim_ws = TRUE) |>
  rename("Y"=last_col()) |>
  mutate(Y=fct(Y))
```

La variable à expliquer est

```r
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 :

```r
sum(is.na(ad.data))
```

```
[1] 2729
```

```r
var.na <- apply(is.na(ad.data),2,any)
names(ad.data)[var.na]
```

```
[1] "X1" "X2" "X3" "X4"
```

```r
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.

```r
ad.data1 <- ad.data[,var.na==FALSE]
dim(ad.data1)
```

```
[1] 3279 1555
```

```r
sum(is.na(ad.data1))
```

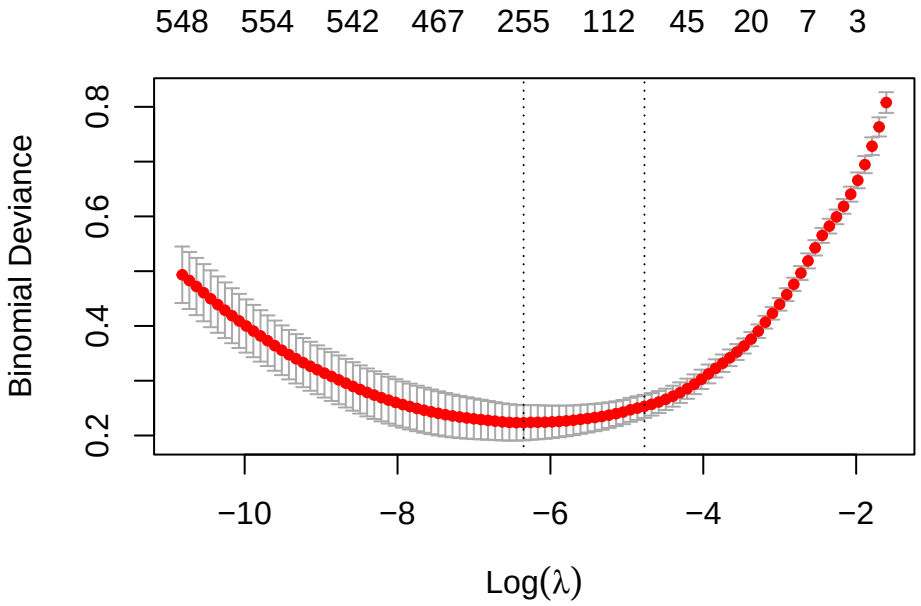
```
[1] 0
```

*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 la syntaxe de l'algorithme **lasso**. Comme pour la régression, on utilise la fonction **cv.glmnet**, il faut simplement 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.*

```
set.seed(5678)
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){
  print(k)
  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")))
}
```

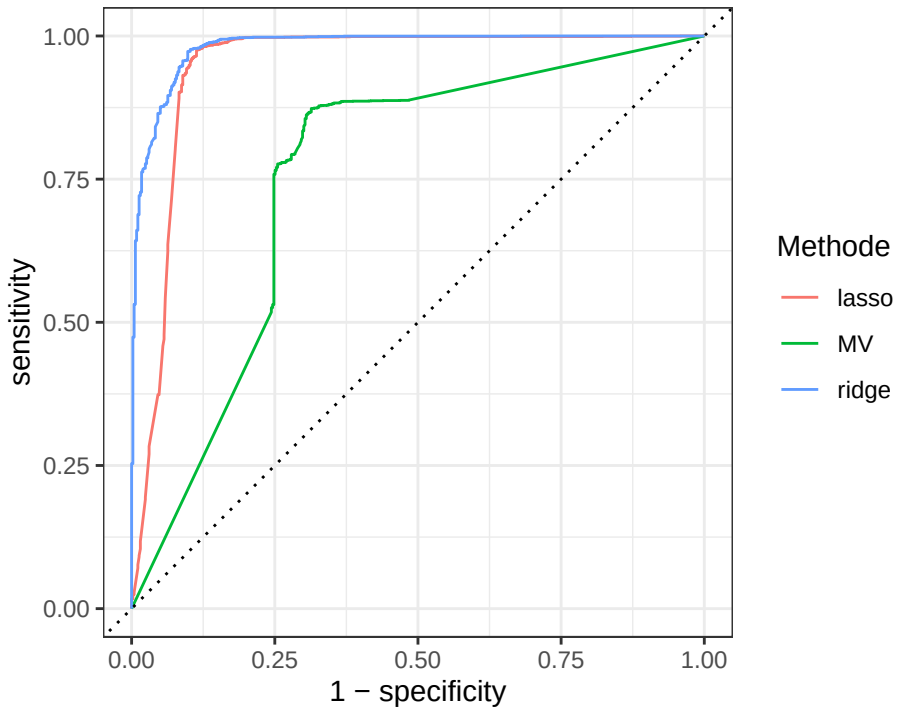
Le tibble `score` *contient, pour chaque individu, les prévisions des probabilités a posteriori*

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

On peut déduire de ce tableau les critères souhaités :

- *les courbes ROC:*

```
library(tidymodels)
score1 <- score |>
  mutate(obs=ad.data1$Y) |>
  pivot_longer(-obs,names_to="Methode",values_to="score")
score1 |> group_by(Methode)|>
  roc_curve(obs,score,event_level = "second") |> autoplot()
```



- *les AUC:*

```
score1 |> group_by(Methode) |>
  roc_auc(obs,score,event_level = "second") |>
  arrange(desc(.estimate))
```

```
# A tibble: 3 x 4
  Methode .metric .estimator .estimate
  <chr>   <chr>   <chr>          <dbl>
1 ridge   roc_auc binary         0.981
2 lasso   roc_auc binary         0.945
3 MV      roc_auc binary         0.756
```

- *les accuracy :*

```
score1 |> mutate(prev=fct_recode(as.character(round(score)),"ad."="0","nonad."="1")) |>
  group_by(Methode) |>
  accuracy(obs,prev) |>
  arrange(desc(.estimate))
```

```
# A tibble: 3 x 4
  Methode .metric  .estimator .estimate
```

```
  <chr>   <chr>    <chr>          <dbl>
1 lasso   accuracy binary         0.970
2 ridge   accuracy binary         0.970
3 MV      accuracy binary         0.847
```

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

On peut également faire le travail dans un environnement ***tidymodels*** *:*

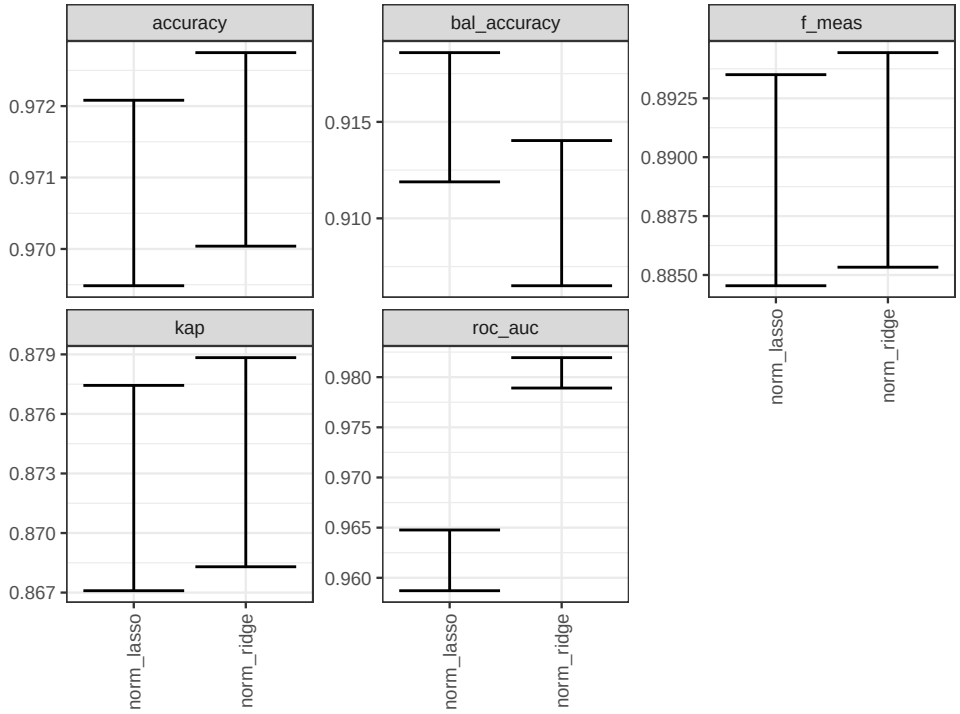
```
# 1 recette pour normaliser
rec_norm <-
  recipe(Y ~ ., data = ad.data1) |>
  step_normalize()

# Definition du lasso et du ridge
lasso_spec <-
  logistic_reg(penalty=tune(),mixture=1) |>
  set_mode("classification") |>
  set_engine("glmnet")

ridge_spec <-
  logistic_reg(penalty=tune(),mixture=0) |>
  set_mode("classification") |>
  set_engine("glmnet")

MV_spec <-
  logistic_reg(penalty=NULL,mixture=NULL) |>
  set_mode("classification") |>
  set_engine("glm")

# agregation des algorithme
wflow <- workflow_set(
  preproc = list(norm=rec_norm),
  models=list(lasso=lasso_spec,
              ridge=ridge_spec)
              #MV_spec)
  )

# definition des blocs
set.seed(12345)
blocs <- vfold_cv(ad.data1,v=10,repeats = 5)

results <- wflow |>
  workflow_map(fn="tune_grid",
               resamples=blocs,
               grid=50,
```

```
                metrics=metric_set(kap,f_meas,bal_accuracy,accuracy,roc_auc),
                seed = 321,
                control=control_grid(event_level = "first",save_pred = FALSE))
```

```
#On peut visualiser les résultats avec :
results |>
  rank_results(select_best = TRUE) |>
  select(wflow_id,.metric,mean) |>
  mutate(mean=round(mean,3)) |>
  pivot_wider(names_from = .metric,values_from = mean)
```

```
# A tibble: 2 x 6
  wflow_id   accuracy bal_accuracy f_meas   kap roc_auc
  <chr>         <dbl>        <dbl>  <dbl> <dbl>   <dbl>
1 norm_ridge    0.971        0.91   0.89  0.874   0.98
2 norm_lasso    0.971        0.915  0.889 0.872   0.962
```

```
results |>
  rank_results(select_best = TRUE) |>
  ggplot()+
  aes(x=wflow_id,ymin=mean-std_err,ymax=mean+std_err,y=mean)+
  geom_errorbar()+
  facet_wrap(~.metric,scales = "free_y")+
  scale_x_discrete(guide = guide_axis(angle = 90),name=NULL)+ylab("")
```



On peut maintenant choisir l'algorithme, par exemple ***ridge*** *:*

```
(best_result <-
   results |>
   extract_workflow_set_result("norm_ridge") |>
   select_best(metric="roc_auc"))
```

```
# A tibble: 1 x 2
  penalty .config
    <dbl> <chr>
1  0.0457 Preprocessor1_Model44
```

```
(final <-
    results |>
    extract_workflow("norm_ridge") |>
    finalize_workflow(best_result) |>
    fit(data=ad.data1))
```

```
== Workflow [trained] ==========================================================
Preprocessor: Recipe
```

```
Model: logistic_reg()

-- Preprocessor ----------------------------------------------------------------
1 Recipe Step

* step_normalize()

-- Model ------------------------------------------------------------------------

Call:  glmnet::glmnet(x = maybe_matrix(x), y = y, family = "binomial",      alpha = ~0)

      Df  %Dev  Lambda
1   1554  0.00 200.700
2   1554  2.26 182.800
3   1554  2.48 166.600
4   1554  2.71 151.800
5   1554  2.97 138.300
6   1554  3.24 126.000
7   1554  3.55 114.800
8   1554  3.88 104.600
9   1554  4.24  95.340
10  1554  4.63  86.870
11  1554  5.05  79.150
12  1554  5.51  72.120
13  1554  6.00  65.710
14  1554  6.52  59.870
15  1554  7.10  54.550
16  1554  7.71  49.710
17  1554  8.38  45.290
18  1554  9.09  41.270
19  1554  9.85  37.600
20  1554 10.66  34.260
21  1554 11.51  31.220
22  1554 12.42  28.440
23  1554 13.39  25.920
24  1554 14.40  23.620
25  1554 15.45  21.520
26  1554 16.56  19.610
27  1554 17.71  17.860
28  1554 18.90  16.280
29  1554 20.13  14.830
30  1554 21.39  13.510
31  1554 22.69  12.310
```

```
32  1554 24.01  11.220
33  1554 25.36  10.220
34  1554 26.72   9.314
35  1554 28.10   8.487
36  1554 29.49   7.733
37  1554 30.89   7.046
38  1554 32.29   6.420
39  1554 33.70   5.850
40  1554 35.11   5.330
41  1554 36.51   4.857
42  1554 37.91   4.425
43  1554 39.30   4.032
44  1554 40.69   3.674
45  1554 42.07   3.347
46  1554 43.43   3.050

...
and 54 more lines.
```

3.4 Exercices

Exercice 3.1 (Estimateurs ridge pour le modèle linéaire)**.** On considère le modèle de régression

$$Y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i$$

où les ε_i sont i.i.d de loi $\mathcal{N}(0, \sigma^2)$. Pour $\lambda \geq 0$, on note $\hat{\beta}_R(\lambda)$ l'estimateur ridge défini par

$$\hat{\beta}_R(\lambda) = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2.$$

1. Exprimer $\hat{\beta}_R(\lambda)$ en fonction de $\mathbb{X}$, $\mathbb{Y}$ et λ.

   Le critère à minimiser se réécrit

   $$\mathcal{C}(\beta) = (\mathbb{Y} - \mathbb{X}\beta)^t(\mathbb{Y} - \mathbb{X}\beta) + \lambda\beta^t\beta.$$

   L'estimateur ridge est donc solution de

   $$-2\mathbb{X}^t\mathbb{Y} + 2\mathbb{X}^t\mathbb{X}\beta + 2\lambda\beta = 0,$$

   d'où

   $$\hat{\beta}_R(\lambda) = (\mathbb{X}^t\mathbb{X} + \lambda I)^{-1}\mathbb{X}^t\mathbb{Y}.$$

2. Étudier le biais et la variance de $\widehat{\beta}_R(\lambda)$ en fonction de λ. On pourra également faire la comparaison avec l'estimateur des MCO.

   Comme $\mathbb{Y} = \mathbb{X}\beta + \varepsilon$, on obtient

   $$\begin{aligned}\mathbf{E}[\widehat{\beta}_R(\lambda)] - \beta &= (\mathbb{X}^t\mathbb{X} + \lambda I)^{-1}\mathbb{X}^t\mathbb{X}\beta - \beta \\ &= \left[(\mathbb{X}^t\mathbb{X} + \lambda I)^{-1}(\mathbb{X}^t\mathbb{X} - (\mathbb{X}^t\mathbb{X} + \lambda I))\right]\beta \\ &= -\lambda(\mathbb{X}^t\mathbb{X} + \lambda I)^{-1}\beta.\end{aligned}$$

   De même, on obtient pour la variance

   $$\mathbf{V}(\widehat{\beta}_R(\lambda)) = \sigma^2(\mathbb{X}^t\mathbb{X} + \lambda\mathbb{I})^{-1}\mathbb{X}^t\mathbb{X}(\mathbb{X}^t\mathbb{X} + \lambda\mathbb{I})^{-1}.$$

   La variance diminue lorsque λ augmente, mais on remarque une augmentation du bais par rapport à l'estimateur des moindres carrés (et réciproquement lorsque λ diminue).

3. On suppose que la matrice $\mathbb{X}$ est orthogonale. Exprimer les estimateurs $\widehat{\beta}_{R,j}(\lambda)$ en fonction des estimateurs des MCO $\widehat{\beta}_j, j = 1, \dots, p$. Interpréter.

   Si $\mathbb{X}$ est orthogonale, alors

   $$\widehat{\beta}_R(\lambda) = \frac{1}{1+\lambda}\mathbb{X}^t\mathbb{Y} = \frac{\widehat{\beta}_{MCO}}{1+\lambda}.$$

Exercice 3.2 (Estimateurs lasso dans le cas orthogonal)**.** Cet exercice est inspiré de Giraud (2015). On rappelle qu'une fonction $F : \mathbb{R}^n \to \mathbb{R}$ est convexe si $\forall x, y \in \mathbb{R}^n$, $\forall \lambda \in [0,1]$ on a

$$F(\lambda x + (1-\lambda)y) \leq \lambda F(x) + (1-\lambda)F(y).$$

On définit la sous-différentielle d'une fonction convexe F par

$$\partial F(x) = \{w \in \mathbb{R}^n : F(y) \geq F(x) + \langle w, y - x\rangle \text{ pour tout } y \in \mathbb{R}^n\}.$$

On admettra que les minima d'une fonction convexe $F : \mathbb{R}^n \to \mathbb{R}$ sont caractérisés par

$$x^\star \in \operatorname*{argmin}_{x\in\mathbb{R}^n} F(x) \iff 0 \in \partial F(x^\star)$$

et que $\partial F(x) = \{\nabla F(x)\}$ lorsque F est différentiable en x.

1. Montrer que pour $x \in \mathbb{R}$

   $$\partial|x| = \begin{cases} \text{signe}(x) & \text{si } x \neq 0 \\ [-1;1] & \text{sinon,} \end{cases}$$

   où $\text{signe}(x) = \mathbf{1}_{x>0} - \mathbf{1}_{x\leq 0}$.

$x \mapsto |x|$ est dérivable partout sauf en 0 donc $\partial|x| = \text{signe}(x)1$ si $x \neq 0$. De plus, si $x = 0$

$$\partial|x| = \{w \in \mathbb{R} : |y| \geq \langle w, y\rangle \ \forall y \in \mathbb{R}\} = \{w \in \mathbb{R} : |y| \geq wy \ \forall y \in \mathbb{R}\} = [-1, 1].$$

2. Soit $x \in \mathbb{R}^n$.

   a. Montrer que
   $$\partial\|x\|_1 = \{w \in \mathbb{R}^n : \langle w, x\rangle = \|x\|_1 \text{ et } \|w\|_\infty \leq 1\}.$$
   On pourra utiliser que pour tout p, q tels que $1/p + 1/q = 1$ on a
   $$\|x\|_p = \sup\left\{\langle w, x\rangle : \|w\|_q \leq 1\right\}.$$

   On montre la double inclusion. Soit w tel que $\langle w, x\rangle = \|x\|_1$ et $\|w\|_\infty = 1$. On a $\forall y \in \mathbb{R}^n$,
   $$\|y\|_1 \geq \langle w, y\rangle = \langle w, y - x + x\rangle = \|x\|_1 + \langle w, y - x\rangle.$$
   Donc $w \in \partial\|x\|_1$. Inversement, soit $w \in \partial\|x\|_1$. Par définition
   $$\partial\|x\|_1 = \{w \in \mathbb{R}^n : \|y\|_1 \geq \langle w, y - x\rangle + \|x\|_1 \ \forall y \in \mathbb{R}^n\}.$$
   Pour $y = 0$ et $y = 2x$, on a donc
   $$\|x\|_1 \leq \langle w, x\rangle \quad \text{et} \quad 2\|x\|_1 \geq \langle w, x\rangle + \|x\|_1$$
   d'où $\|x\|_1 = \langle x, w\rangle = \sum_i w_i x_i$. De plus en posant $\tilde{w} = (0, \dots, 0, \text{signe}(w_i), 0, \dots, 0)$ où la coordonnée non nulle correspond au $\max_i(|w_i|)$ on a $\|w\|_\infty = \langle w, \tilde{w}\rangle$ et $\|\tilde{w}\|_\infty = \|\tilde{w}\|_1 = 1$. De plus
   $$\|\tilde{w}\|_1 \geq \|x\|_1 + \langle w, \tilde{w} - x\rangle = \|w\|_\infty \quad \Longrightarrow \|w\|_\infty \leq \|\tilde{w}\|_1 = 1.$$

   b. En déduire
   $$\partial\|x\|_1 = \{w \in \mathbb{R}^n : w_j = \text{signe}(x_j) \text{ si } x_j \neq 0, w_j \in [-1, 1] \text{ si } x_j = 0\}.$$

   On a
   $$\begin{aligned}\partial\|x\|_1 &= \{w \in \mathbb{R}^n : \langle w, x\rangle = \|x\|_1 \text{ et } \|w\|_\infty \leq 1\} \\ &= \{w \in \mathbb{R}^n : \sum_{i=1}^n (w_i x_i - |x_i|) = 0 \text{ et } \|w\|_\infty \leq 1\}.\end{aligned}$$
   Or si $\|w\|_\infty \leq 1$ alors $w_i x_i - |x_i| \leq 0 \ \forall i = 1, \dots, n$. Donc
   $$\begin{aligned}\partial\|x\|_1 &= \{w \in \mathbb{R}^n : (w_i x_i - |x_i|) = 0, i = 1, \dots, n \text{ et } \|w\|_\infty \leq 1\} \\ &= \{w \in \mathbb{R}^n : w_j = \text{signe}(x_j)1 \text{ si } x_j \neq 0, w_j \in [-1, 1] \text{ si } x_j = 0\}.\end{aligned}$$

3. Étant données n observations $(x_i, y_i), i = 1, \dots, n$ telles que $x_i \in \mathbb{R}^p$ et $y_i \in \mathbb{R}$ on rappelle que l'estimateur lasso $\hat{\beta}(\lambda)$ est construit en minimisant

$$\mathcal{L}(\beta) = \|Y - \mathbb{X}\beta\|_2^2 + \lambda\|\beta\|_1. \tag{3.2}$$

On admettra que la sous-différentielle $\partial\mathcal{L}(\beta)$ est donnée par

$$\partial\mathcal{L}(\beta) = \left\{-2\mathbb{X}^t(Y - \mathbb{X}\beta) + \lambda z : z \in \partial\|\beta\|_1\right\}.$$

Montrer que $\hat{\beta}(\lambda)$ vérifie

$$\mathbb{X}^t\mathbb{X}\hat{\beta}(\lambda) = \mathbb{X}^tY - \frac{\lambda}{2}\hat{z}$$

où $\hat{z} \in \mathbb{R}^p$ vérifie

$$\hat{z}_j \begin{cases} = \text{signe}(\hat{\beta}_j(\lambda)) & \text{si } \hat{\beta}_j(\lambda) \neq 0 \\ \in [-1; 1] & \text{sinon.} \end{cases}$$

D'après les indications, on a $0 \in \partial\mathcal{L}(\hat{\beta}(\lambda))$. Donc il existe $\hat{z} \in \partial\|\hat{\beta}(\lambda)\|_1$ tel que

$$-2\mathbb{X}^t(Y - \mathbb{X}\hat{\beta}(\lambda)) + \lambda\hat{z} = 0 \quad \Longleftrightarrow \quad \mathbb{X}^t\mathbb{X}\hat{\beta}(\lambda) = \mathbb{X}^tY - \frac{\lambda}{2}\hat{z}.$$

4. On suppose maintenant que la matrice $\mathbb{X}$ est orthogonale.

   a. Montrer que

   $$\text{signe}(\hat{\beta}_j(\lambda)) = \text{signe}(\mathbb{X}_j^tY) \quad \text{lorsque } \hat{\beta}_j(\lambda) \neq 0$$

   et $\hat{\beta}_j(\lambda) = 0$ si et seulement si $|\mathbb{X}_j^tY| \leq \lambda/2$.

   $\mathbb{X}$ étant orthogonale, on a pour $\hat{\beta}_j(\lambda) \neq 0$

   $$\hat{\beta}_j(\lambda) + \frac{\lambda}{2}\text{signe}(\hat{\beta}_j(\lambda)) = \hat{\beta}_j(\lambda)\left(1 + \frac{\lambda}{2|\hat{\beta}_j(\lambda)|}\right) = \mathbb{X}_j^tY,$$

   donc $\hat{\beta}_j(\lambda)$ est du signe de $\mathbb{X}_j^tY$. De plus si $\hat{\beta}_j(\lambda) = 0$ alors $\mathbb{X}_j^tY = \frac{\lambda}{2}\hat{z}_j$ avec $\hat{z}_j \in [-1, 1]$. Donc

   $$|\mathbb{X}_j^tY| = \left|\frac{\lambda}{2}\hat{z}_j\right| \leq \frac{\lambda}{2}.$$

   A l'inverse si $|\mathbb{X}_j^tY| \leq \lambda/2$ et si $\hat{\beta}_j(\lambda) \neq 0$ alors

   $$\left|\hat{\beta}_j(\lambda)\left(1 + \frac{\lambda}{2|\hat{\beta}_j(\lambda)|}\right)\right| = |\hat{\beta}_j(\lambda)| + \frac{\lambda}{2} = |\mathbb{X}_j^tY| \leq \frac{\lambda}{2}.$$

   Donc $\hat{\beta}_j(\lambda) = 0$.

b. En déduire

$$\hat{\beta}_j(\lambda) = \mathbb{X}_j^t Y \left(1 - \frac{\lambda}{2|\mathbb{X}_j^t Y|}\right)_+, \quad j = 1, \dots, p$$

où $(x)_+ = \max(x, 0)$. Interpréter ce résultat.

On obtient donc

$$\hat{\beta}_j(\lambda) = \mathbb{X}_j^t Y - \frac{\lambda}{2} \frac{\mathbb{X}_j^t Y}{|\mathbb{X}_j^t Y|} = \mathbb{X}_j^t Y \left(1 - \frac{\lambda}{2|\mathbb{X}_j^t Y|}\right)$$

si $\mathbb{X}_j^t Y \geq \frac{\lambda}{2}$ et $\hat{\beta}_j(\lambda) = 0$ sinon. D'où

$$\hat{\beta}_j(\lambda) = \mathbb{X}_j^t Y \left(1 - \frac{\lambda}{2|\mathbb{X}_j^t Y|}\right)_+, \quad j = 1, \dots, d.$$

Exercice 3.3 (Unicité de l'estimateur lasso)**.** Cet exercice est inspiré de Giraud (2015). Soit $\hat{\beta}^1(\lambda)$ et $\hat{\beta}^2(\lambda)$ deux solutions qui minimisent l'Équation 3.2. Soit $\hat{\beta} = (\hat{\beta}^1(\lambda) + \hat{\beta}^2(\lambda))/2$.

1. Montrer que si $\mathbb{X}\hat{\beta}^1(\lambda) \neq \mathbb{X}\hat{\beta}^2(\lambda)$ alors

$$\|\mathbb{Y} - \mathbb{X}\hat{\beta}\|_2^2 + \lambda\|\hat{\beta}\|_1 < \frac{1}{2}\left(\|\mathbb{Y} - \mathbb{X}\hat{\beta}^1(\lambda)\|_2^2 + \lambda\|\hat{\beta}^1(\lambda)\|_1 + \|\mathbb{Y} - \mathbb{X}\hat{\beta}^2(\lambda)\|_2^2 + \lambda\|\hat{\beta}^2(\lambda)\|_1\right).$$

On pourra utiliser la convexité (forte) de $x \mapsto \|x\|_2^2$.

On a

$$\begin{aligned}\|\mathbb{Y} - \mathbb{X}\hat{\beta}\|_2^2 + \lambda\|\hat{\beta}\|_1 =& \left\|\frac{1}{2}(\mathbb{Y} - \mathbb{X}\hat{\beta}^1(\lambda)) + \frac{1}{2}(\mathbb{Y} - \mathbb{X}\hat{\beta}^2(\lambda))\right\|_2^2 + \lambda\left\|\frac{1}{2}(\hat{\beta}^1(\lambda) + \hat{\beta}^2(\lambda))\right\|_1 \\ <& \frac{1}{2}\left\|\mathbb{Y} - \mathbb{X}\hat{\beta}^1(\lambda))\right\|_2^2 + \frac{1}{2}\left\|\mathbb{Y} - \mathbb{X}\hat{\beta}^2(\lambda))\right\|_2^2 + \frac{1}{2}\lambda\|\hat{\beta}^1(\lambda)\|_1 + \frac{1}{2}\lambda\|\hat{\beta}^2(\lambda)\|_1\end{aligned}$$

en utilisant la stricte convexité de $x \mapsto \|x\|_2^2$ et l'inégalité triangulaire.

2. En déduire que $\mathbb{X}\hat{\beta}^1(\lambda) = \mathbb{X}\hat{\beta}^2(\lambda)$.

Donc si $\mathbb{X}\hat{\beta}^1(\lambda) \neq \mathbb{X}\hat{\beta}^2(\lambda)$ alors

$$\|\mathbb{Y} - \mathbb{X}\hat{\beta}\|_2^2 + \lambda\|\hat{\beta}\|_1 < \|\mathbb{Y} - \mathbb{X}\hat{\beta}^1(\lambda)\|_2^2 + \lambda\|\hat{\beta}^1(\lambda)\|_1$$

ce qui est impossible par définition de $\hat{\beta}^1(\lambda)$.

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

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

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 $\bar{\mathbb{Y}}$ le vecteur des y_i et $\mathbb{X}_k$ le vecteur contenant les observations de la variable k pour $k = 1, \dots, d$. L'algorithme se résume ainsi

1. Initialisation : $\hat{\alpha} = \bar{\mathbb{Y}}$, $\hat{g}_k(x_k) = \bar{\mathbb{X}}_k$.
2. Pour $k = 1, \dots, d$:
   - $\mathbb{Y}^{(k)} = \mathbb{Y} - \hat{\alpha} - \sum_{j \neq k} \hat{g}_j(\mathbb{X}_j)$ (résidus partiels)
   - $\hat{g}_k$: lissage non paramétrique de $\mathbb{Y}^{(k)}$ sur $\mathbb{X}_k$.
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

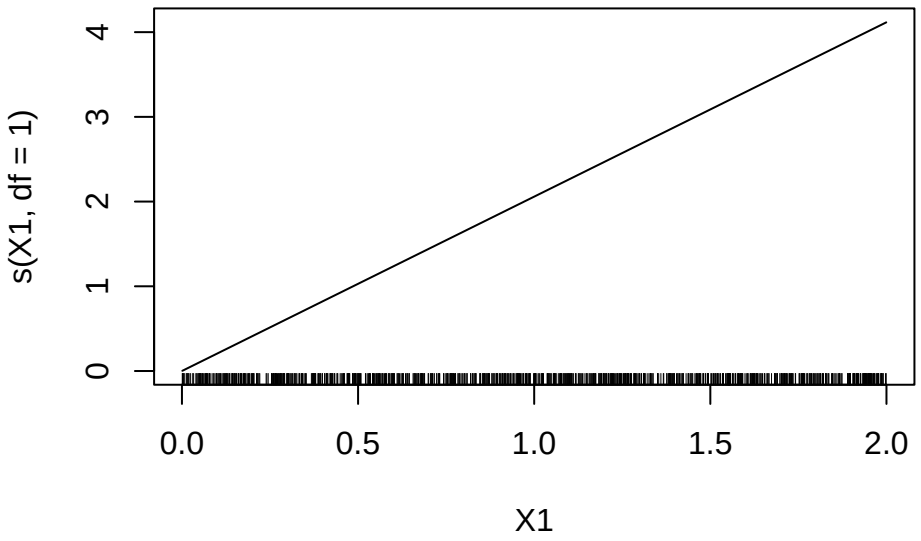
   Il s'agit d'un modèle additif

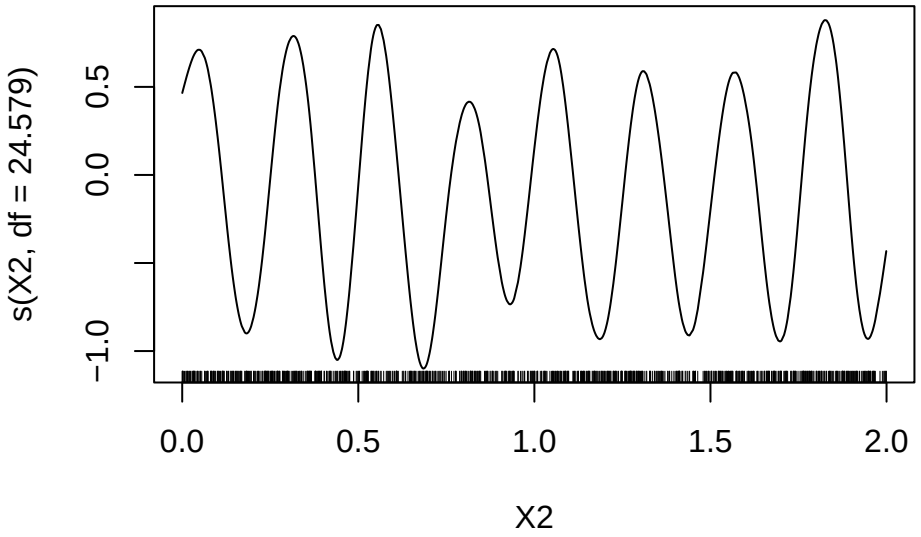
   $$Y = 2X_1 + \sin(8\pi X_2) + \varepsilon$$

   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)
```

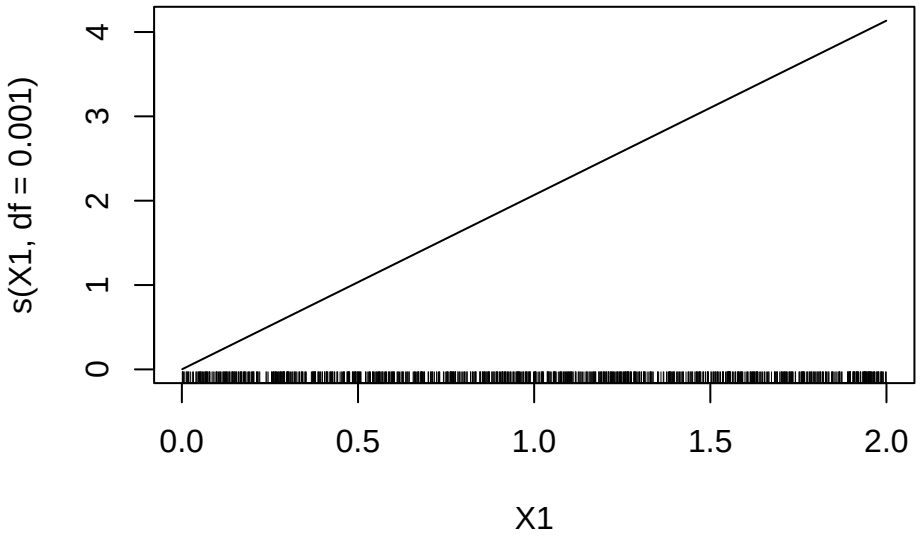


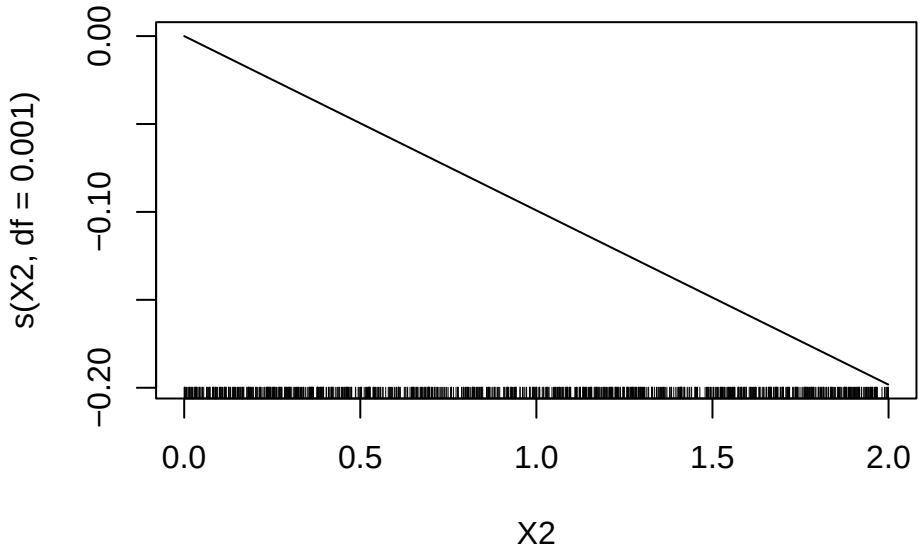


3. Faire varier les degrés de liberté, interpréter.

   On prend d'abord peu de degrés de liberté.

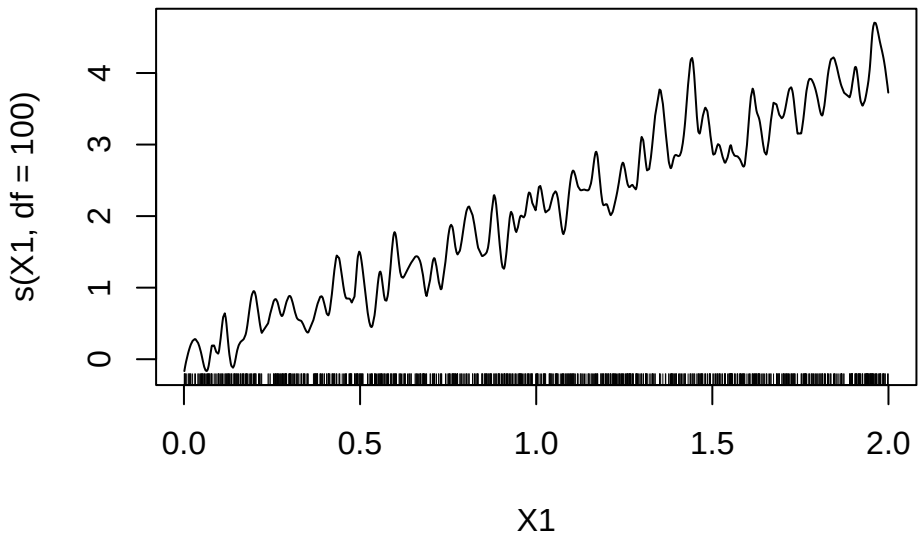
```
model2 <- gam(Y~s(X1,df=0.001)+s(X2,df=0.001)-1,data=donnees)
plot(model2)
```

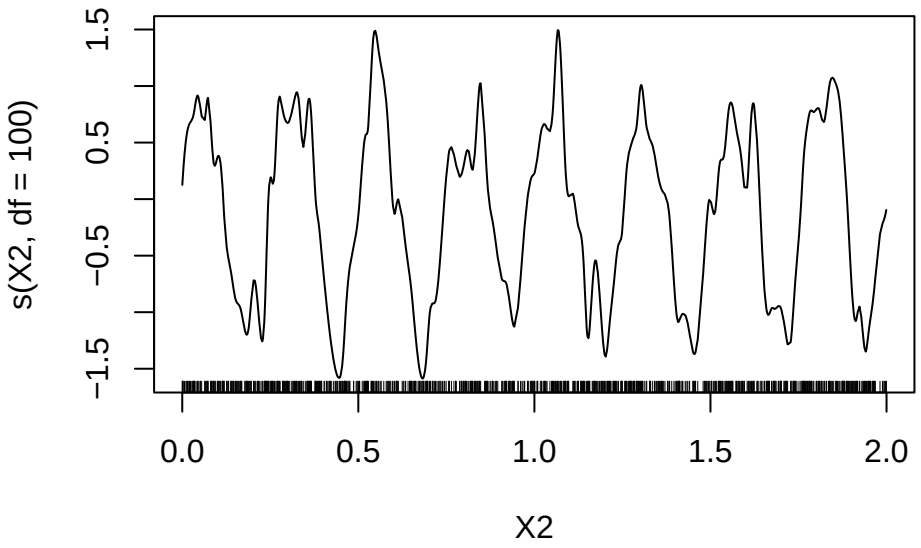




Le sinus n'est pas bien estimé. On prend maintenant un grand nombre de degrés de liberté.

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

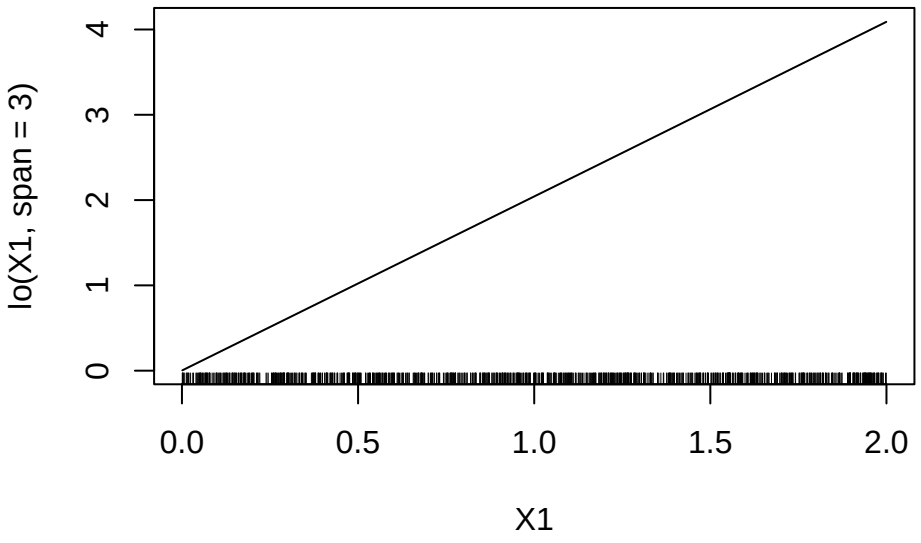


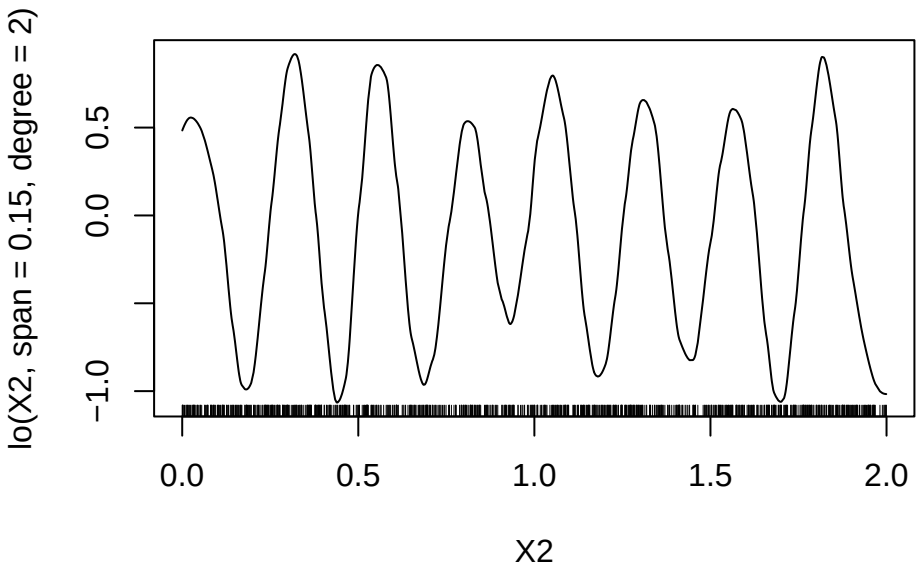


Le modèle est trop flexible, risque de sur-ajustement.

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`.

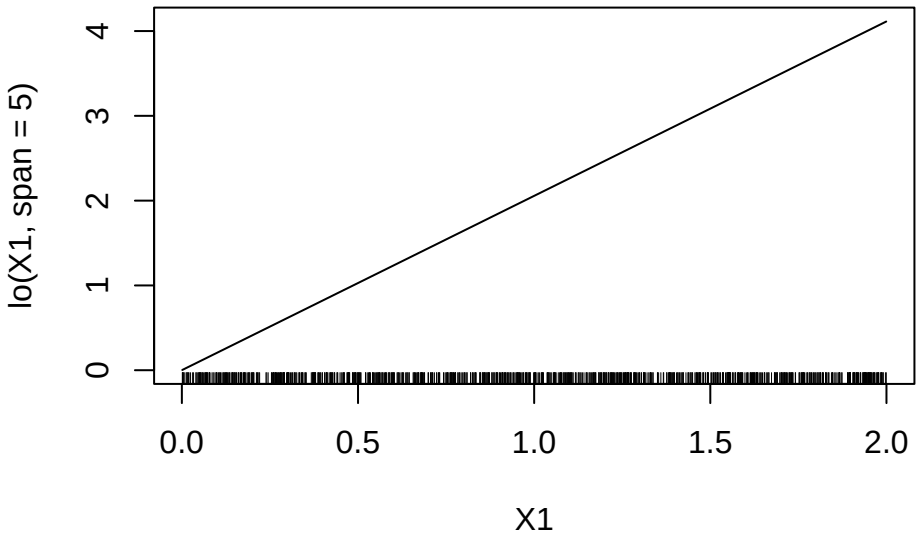
```
model4 <- gam(Y~lo(X1,span=3)+lo(X2,span=0.15,degree=2)-1,data=donnees)
plot(model4)
```

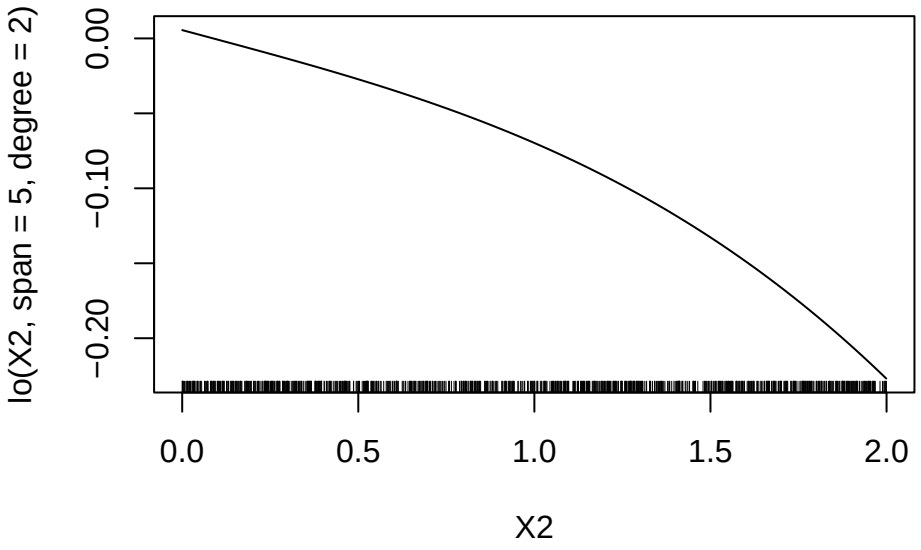




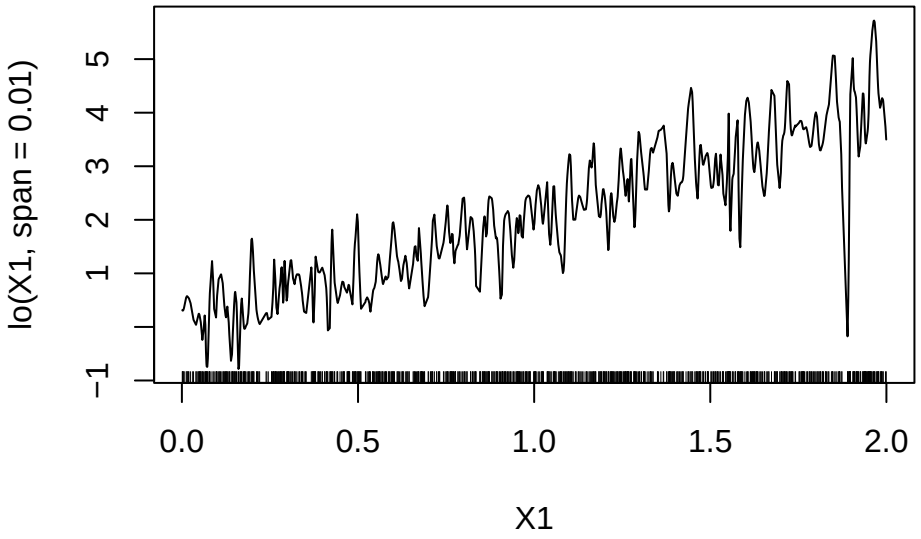
On fait varier ***span*** *:*

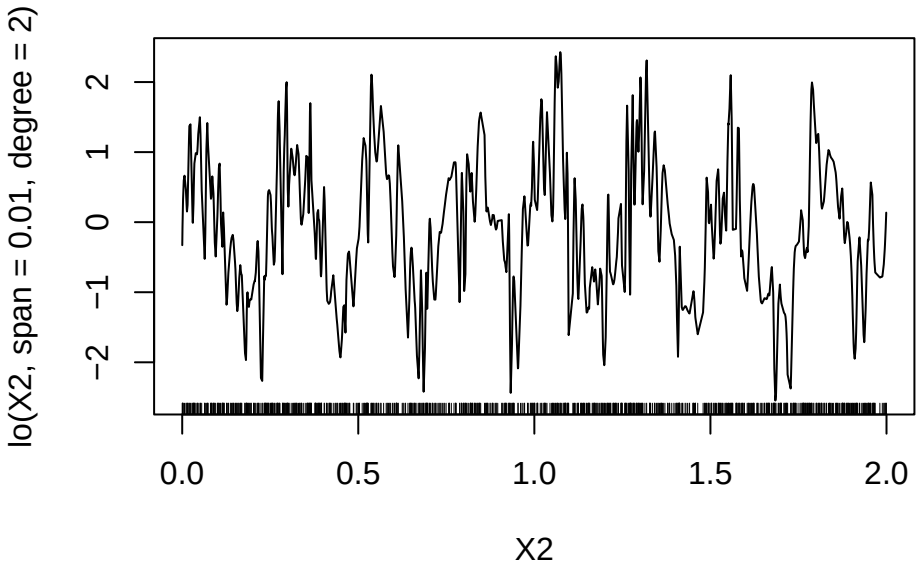
```
model5 <- gam(Y~lo(X1,span=5)+lo(X2,span=5,degree=2)-1,data=donnees)
plot(model5)
```






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

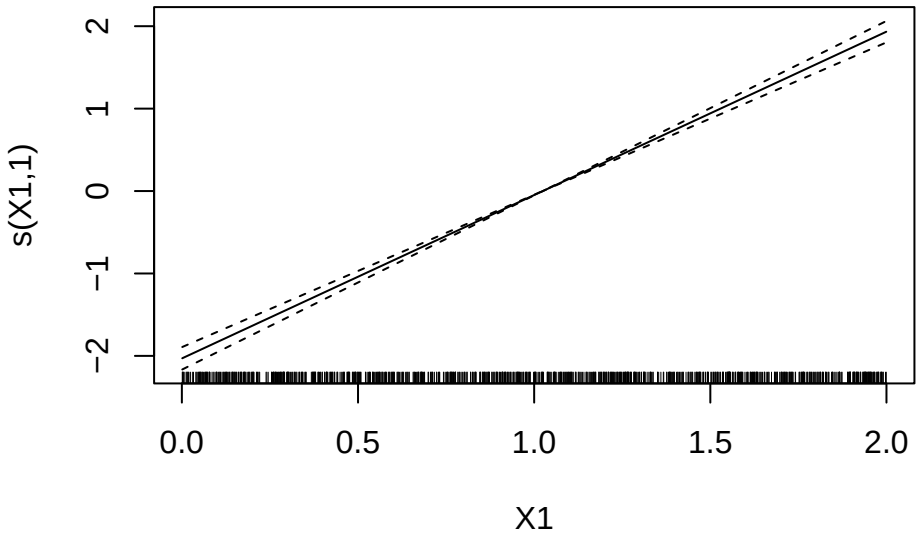


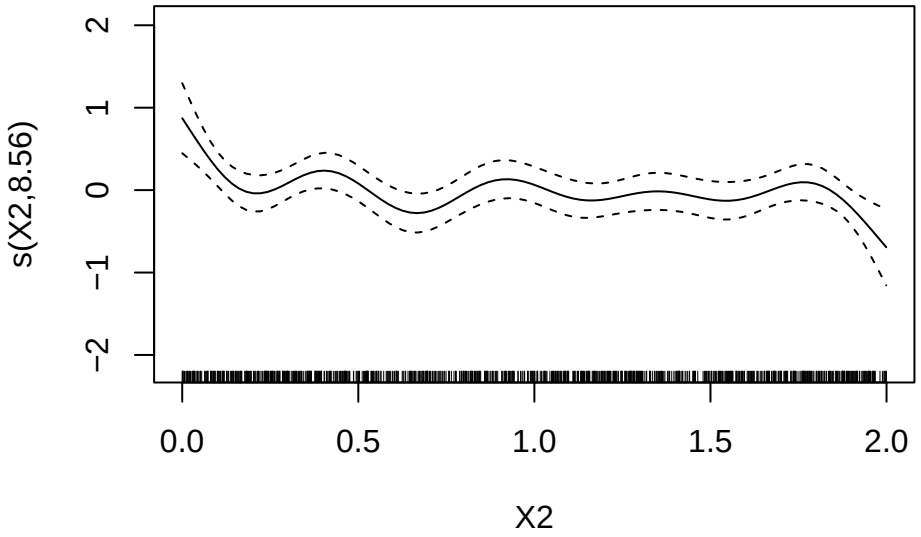


On a les mêmes remarques que pour les splines.

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)
```

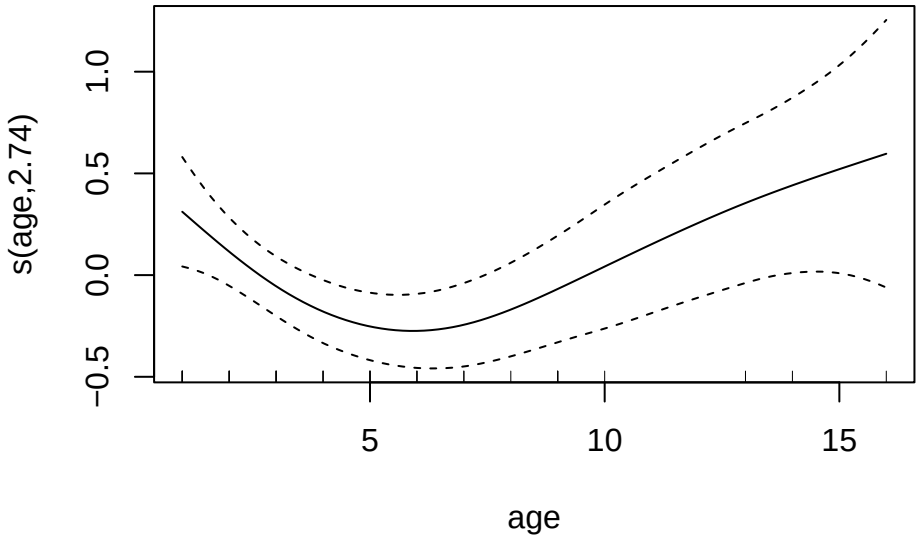
```
Call:
glm(formula = etat ~ age, family = binomial, data = panne)

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 (vieillissement 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)

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" !

partie II

Non supervisée

5 Rappels sur le k-means et la CAH

Ces méthodes sont certainement les deux algorithmes les plus utilisés en apprentissage non supervisé.

L'algorithme des k-means propose de trouver un représentant pour chaque classe, appelé centroïde, en minimisant :

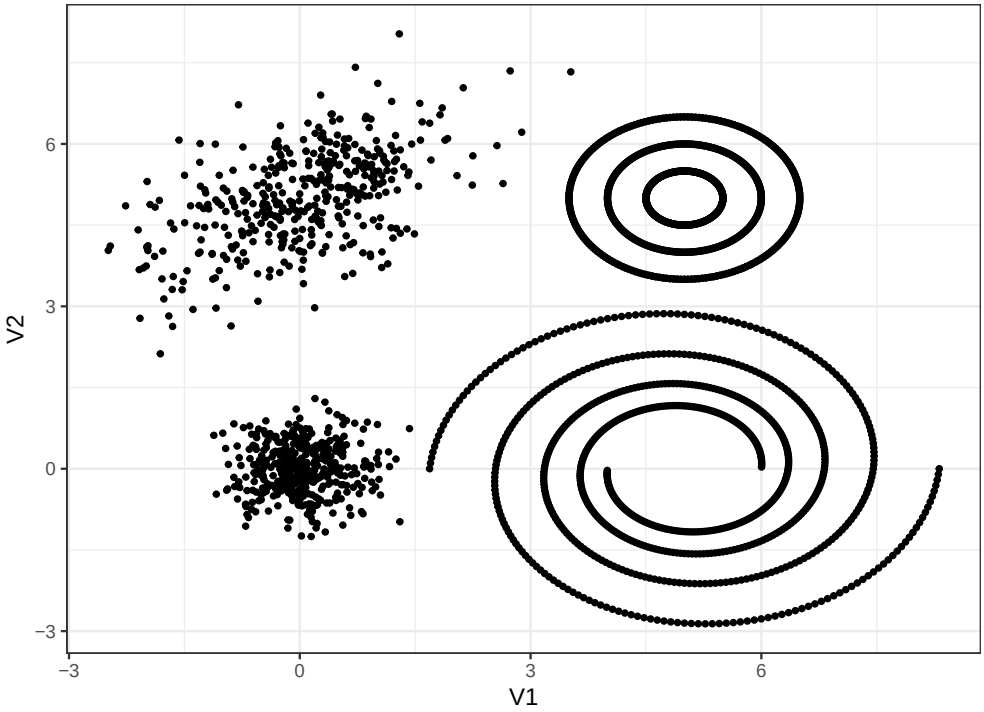
$$\frac{1}{n}\sum_{i=1}^{n}\min_{j=1,\dots,K}\|x_i - c_j\|^2.$$

Plusieurs types d'algorithmes peuvent être utiliser pour trouver des solutions (locales) à ce problème. Une fois la solution obtenue, les clusters s'obtiennent en affectant chaque observation à son centroïde le plus proche.

Une **CAH** va quant à elle définir des clusters de façon récursive en agrégeant à chaqué étape les deux clusters les plus proches au sens d'une mesure de proximité à définir.

Exercice 5.1 (kmeans et CAH sur R)**.** On considère les données

```
tbl <- read_delim("data/donclassif.txt",delim = ";")
ggplot(tbl)+aes(x=V1,y=V2)+geom_point()
```



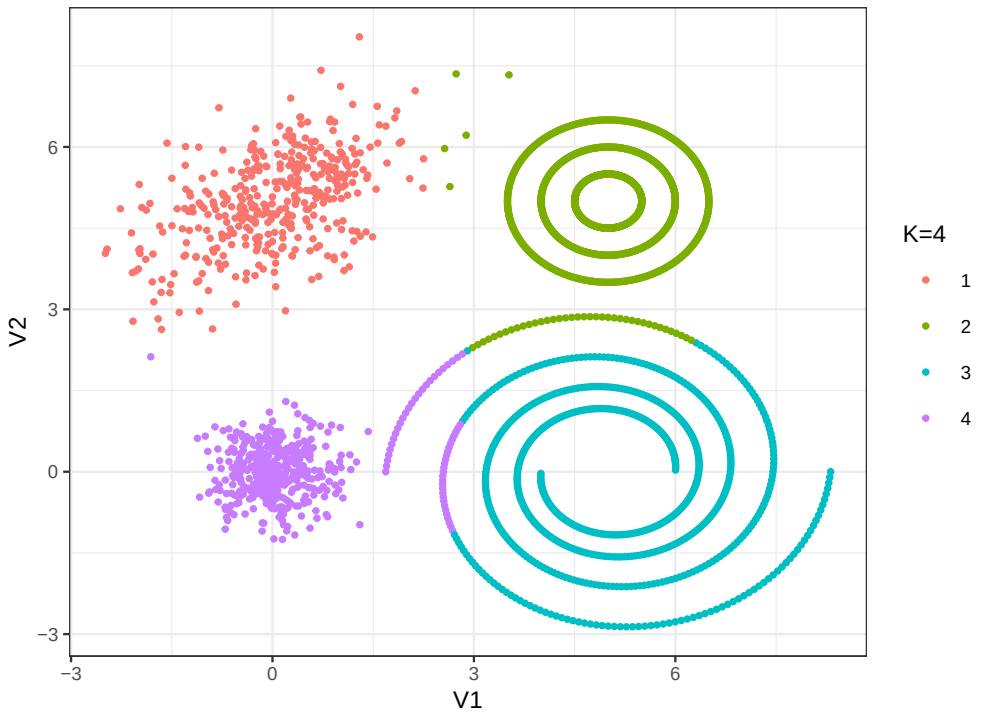
1. Discuter du nombre de clusters pour ce jeu de données.

   Le choix du nombre de groupes est toujours une question difficile. On peut dire à minima qu'il y a 4 groupes : un dans chaque coin. Ensuite, il semble éventuellement possible de scinder les groupes de droite en sous-groupes pour arriver au total à 9-10 groupes.

2. Tester différents algorithmes k-means, visualiser les résultats et discuter de la capacité de cet algorithme à identifier les différentes structures géométriques des données.

   Commençons par un k-means à 4 groupes :

```
res4 <- kmeans(tbl,centers = 4,nstart = 100)
tbl1 <- tbl |> mutate(`K=4`=res4$cluster)
ggplot(tbl1)+aes(x=V1,y=V2,color=as.factor(`K=4`))+
  geom_point()+labs(color="K=4")
```



On passe à plus de groupes

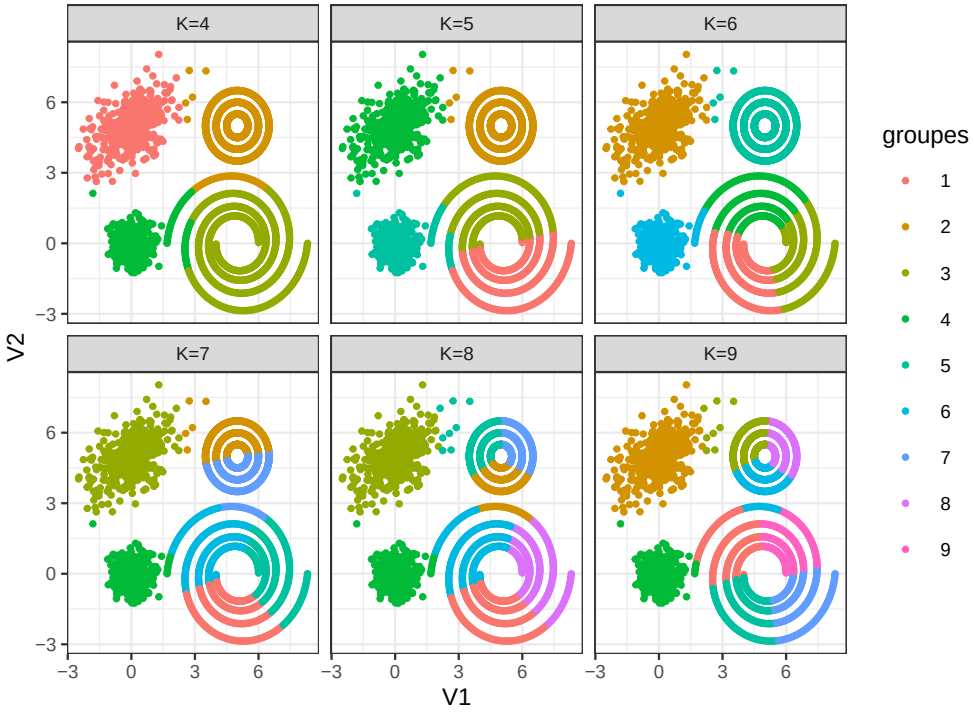
```
nom <- paste("K=",5:9,sep="")
mat <- matrix(0,ncol=5,nrow=nrow(tbl))
k <- 5:9
for (j in 1:5){
  res <- kmeans(tbl,centers = k[j],nstart = 100)
  mat[,j] <- res$cluster
}
mat1 <- as_tibble(mat)
names(mat1) <- nom
(tbl2 <- tbl1 |> bind_cols(mat1))
```

```
# A tibble: 2,800 x 8
      V1     V2 `K=4` `K=5` `K=6` `K=7` `K=8` `K=9`
   <dbl>  <dbl> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  6.00 0.0315     3     1     3     5     8     9
 2  6.00 0.0632     3     1     3     5     8     9
 3  6.00 0.0950     3     1     3     5     8     9
 4  6.00 0.127      3     1     3     5     8     9
 5  6.00 0.159      3     1     3     5     8     9
 6  6.00 0.191      3     1     3     5     8     9
 7  6.00 0.223      3     1     3     5     8     9
```

```
 8  5.99 0.255      3     3     3     5     8     9
 9  5.99 0.287      3     3     3     5     8     9
10  5.98 0.318      3     3     3     5     8     9
# i 2,790 more rows
```

On visualise les résultats :

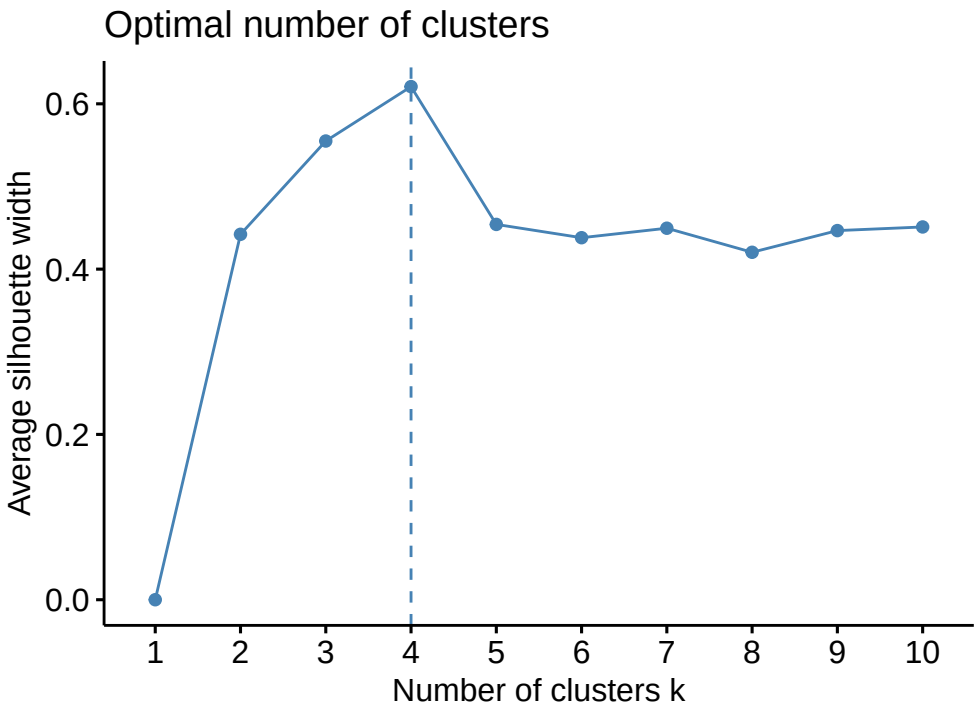
```
tbl3 <- tbl2 |>
  pivot_longer(-c(V1,V2),names_to = "Nb_clust",values_to="groupes") |>
  mutate(groupes=as.factor(groupes))
ggplot(tbl3)+aes(x=V1,y=V2,color=groupes)+
  geom_point()+facet_wrap(~Nb_clust)
```



Sans surprise, le k-means ne parvient pas à scinder les spirales en bas à droite et les cercles concentriques en haut à droite.

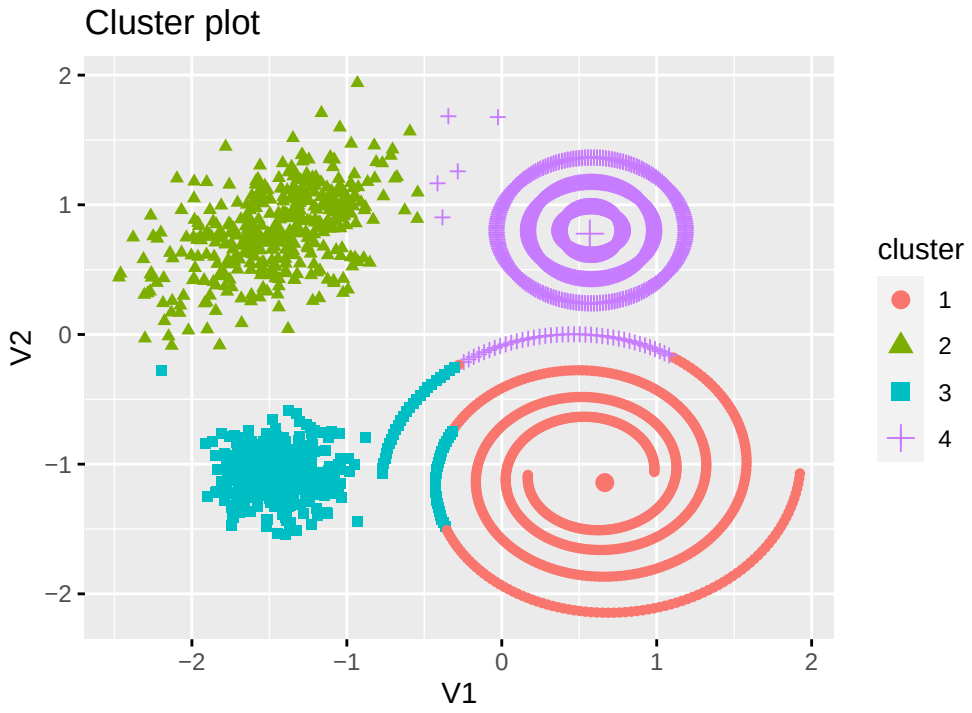
3. Sélectionner le nombre de classes à l'aide du coefficient de silhouette. On pourra utiliser la fonction `fviz_nbclust` du package **factoextra**. Visualiser les clusters avec `fviz_cluster`.

```
library(factoextra)
fviz_nbclust(tbl, kmeans, method = "silhouette")
```



On choisit 4 groupes que l'on visualise :

```
res4 <- kmeans(tbl,centers = 4,nstart = 100)
fviz_cluster(res4,tbl,ellipse = FALSE,labelsize = 0)
```

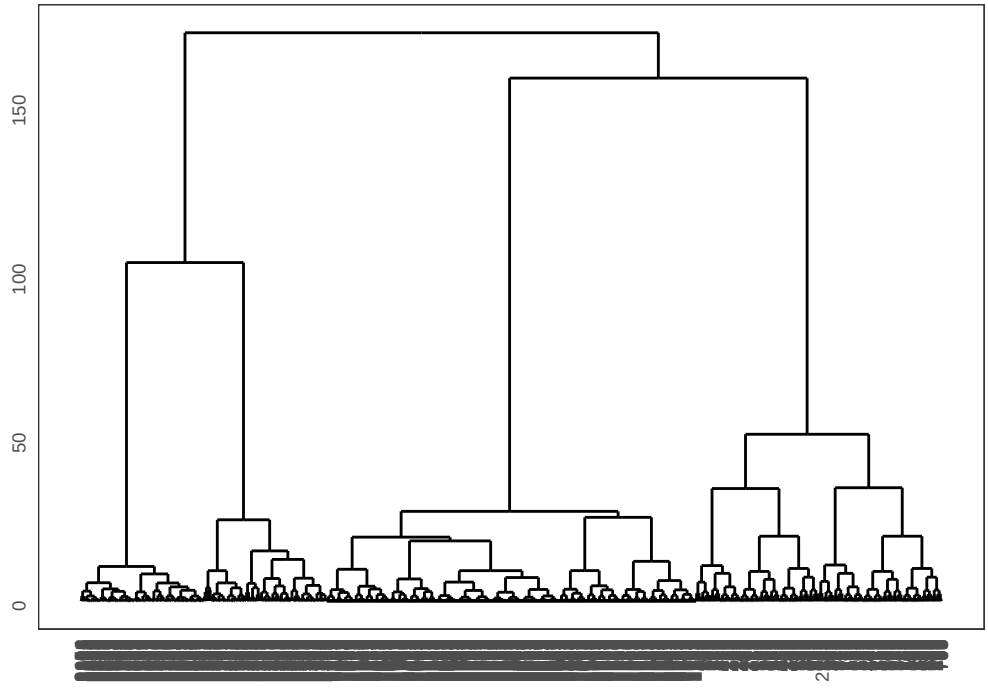


4. Faire le même travail avec la classification ascendente hiérarchique. On commencera par comparer les différentes méthodes d'agglomération en fonction du nombre de clusters, afin d'en déduire une stratégie efficace permettant notamment d'identifier les spirales et les cercles concentriques.

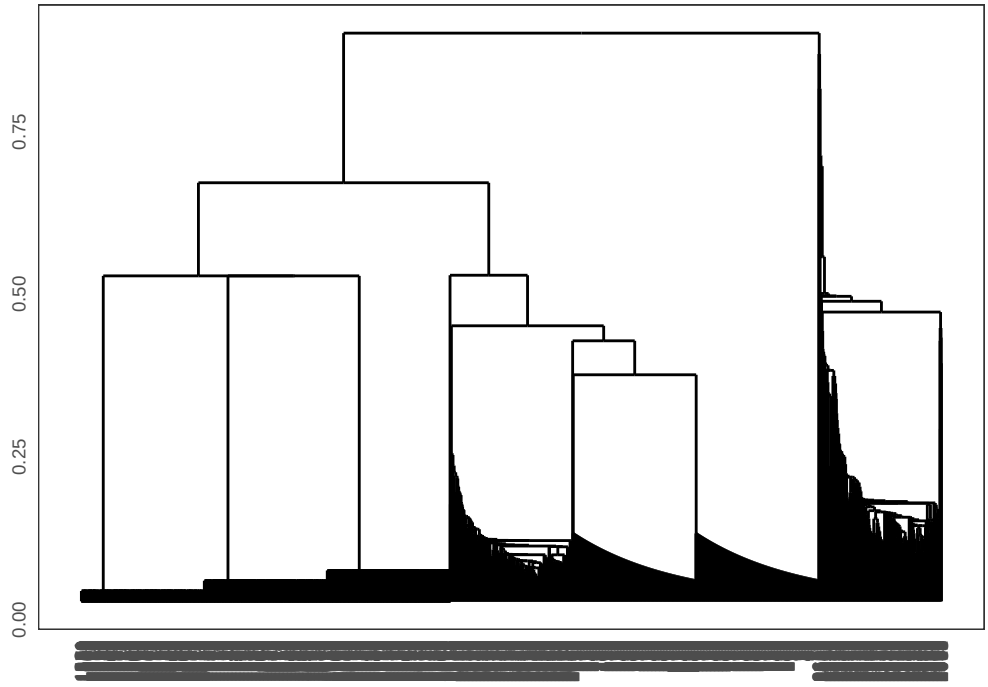
   On commence par calculer la ***matrice de distances*** *et on visualise les* ***dendrogrammes***

```
DD <- dist(tbl)
ward <- hclust(DD,method="ward.D2")
single <- hclust(DD,method="single")
complete <- hclust(DD,method="complete")
average <- hclust(DD,method="average")
```

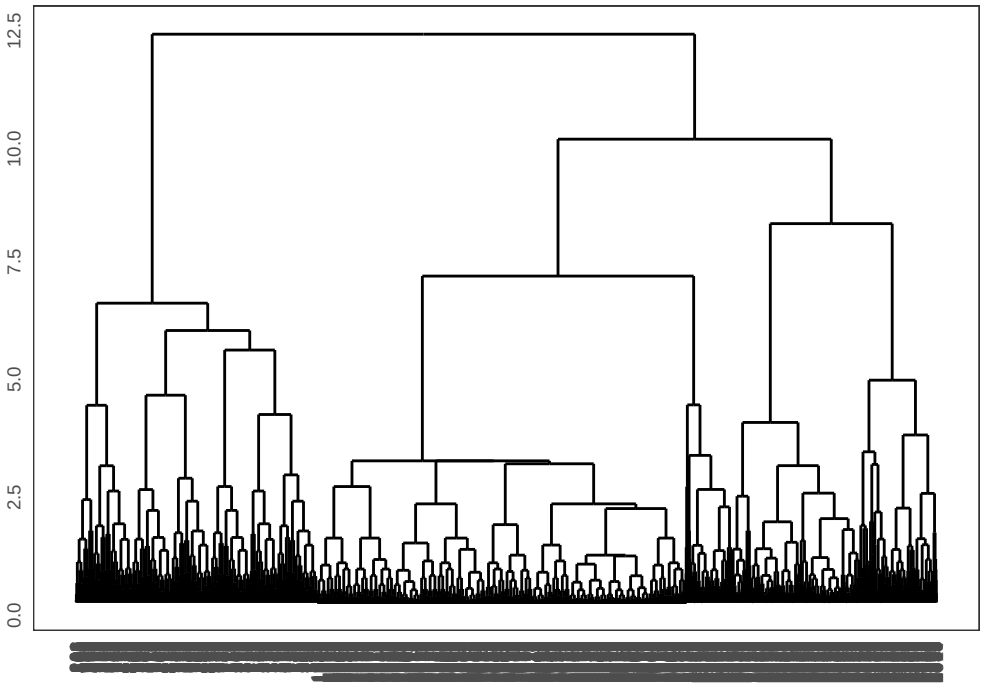
```
library(ggdendro)
ggdendrogram(ward)
```



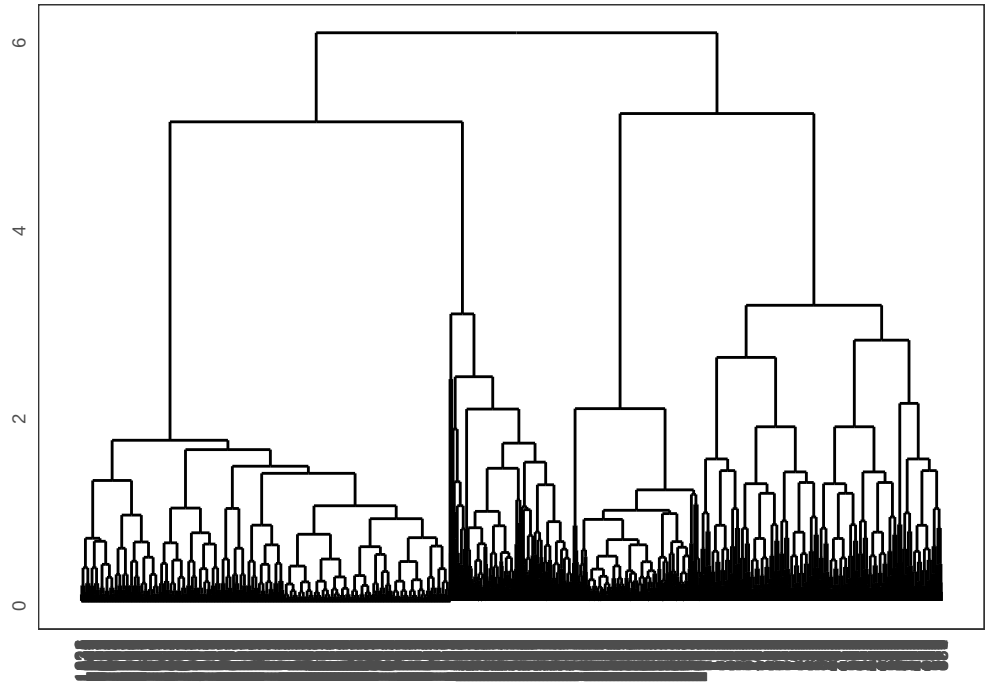

```
ggdendrogram(single)
```



```
ggdendrogram(complete)
```

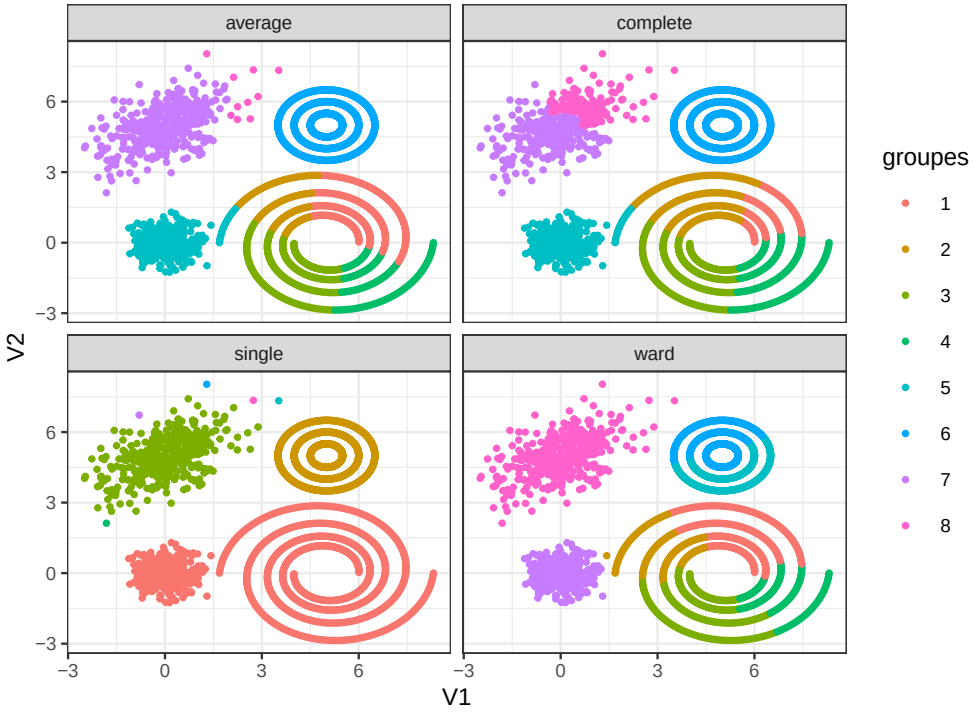



```
ggdendrogram(average)
```



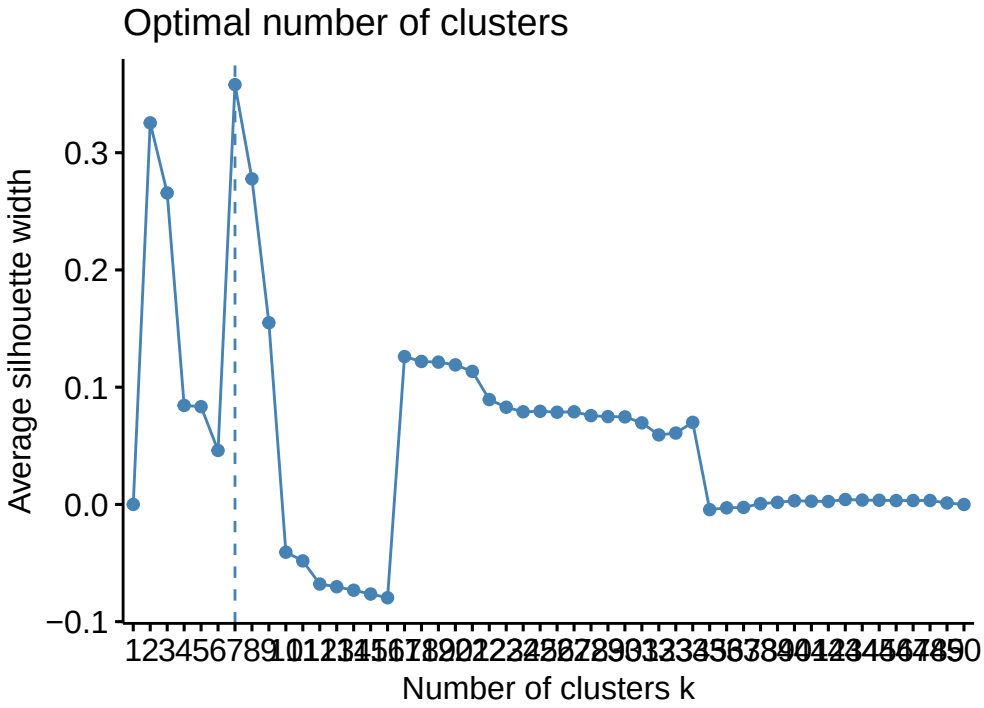
On regarde maintenant ce qu'il se passe pour un nombre de classes fixé, par exemple 8.

```
ward8 <- cutree(ward,k = 8)
single8 <- cutree(single,k = 8)
complete8 <- cutree(complete,k = 8)
average8 <- cutree(average,k = 8)
tbl_cah <- tbl |> mutate(
  ward = ward8,
  single = single8,
  complete = complete8,
  average = average8)
tbl1_cah <- tbl_cah |>
  pivot_longer(-c(V1, V2), names_to = "Nb_clust",
               values_to = "groupes") |>
  mutate(groupes = as.factor(groupes))
ggplot(tbl1_cah) + aes(x = V1, y = V2,
                       color = groupes) +
  geom_point() + facet_wrap(~ Nb_clust)
```



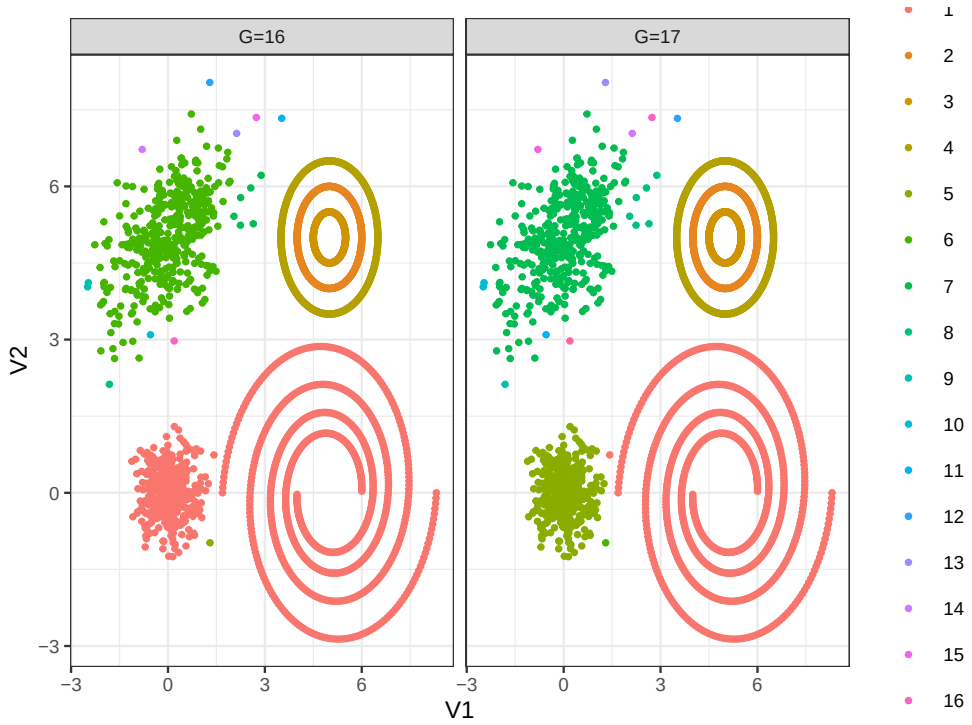
Ici encore il est difficile d'identifier les cercles concentriques et la spirale. Pour y parvenir, on propose de regarder un peu plus en détails le lien simple :

```
fviz_nbclust(tbl,
             hcut,
             method = "silhouette",
             hc_method="single",
             k.max=50)
```



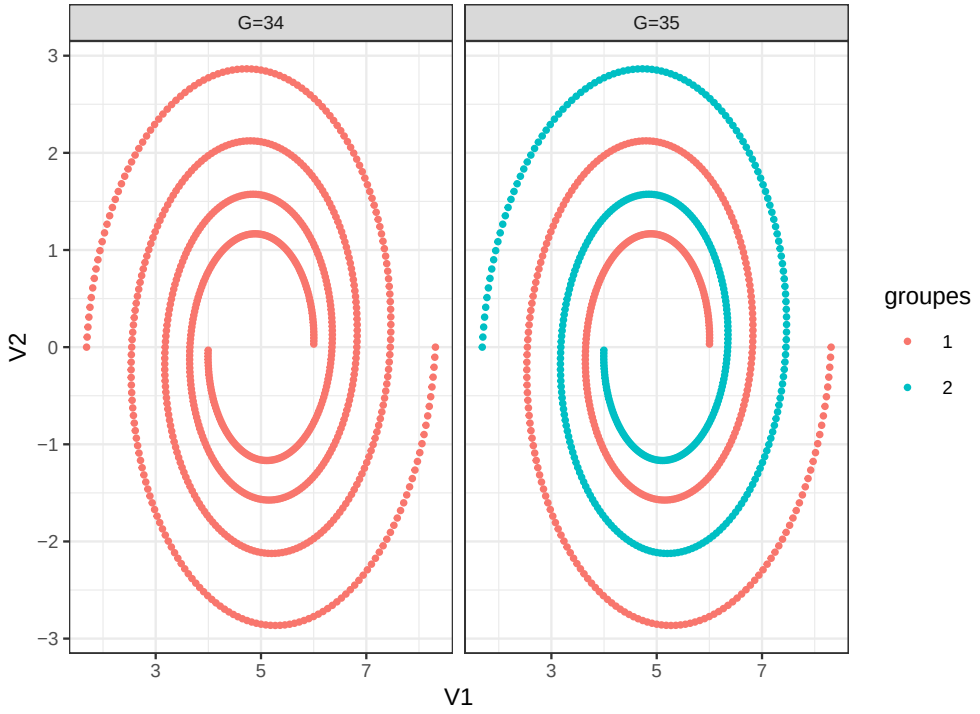
La silouhette augmente à nouveau quand on passe à 17 groupes : cela correspond à la division des groupes du bas :

```
single16 <- cutree(single,k = 16)
single17 <- cutree(single,k = 17)
tbl |> mutate(`G=16`=single16,`G=17`=single17) |>
  pivot_longer(-c(V1, V2), names_to = "Nb_clust",
               values_to = "groupes") |>
  mutate(groupes = as.factor(groupes)) |>
  ggplot() + aes(x = V1, y = V2,color = groupes) +
  geom_point() + facet_wrap(~ Nb_clust)
```



On a également une cassure lorsqu'on passe à 35 groupes avec ici une baisse de la silouhette :

```
single34 <- cutree(single,k = 34)
single35 <- cutree(single,k = 35)
tbl |> mutate(`G=34`=single34,`G=35`=single35) |>
  filter(`G=35`<=2) |>
  pivot_longer(-c(V1, V2), names_to = "Nb_clust",
               values_to = "groupes") |>
  mutate(groupes = as.factor(groupes)) |>
  ggplot() + aes(x = V1, y = V2,color = groupes) +
  geom_point() + facet_wrap(~ Nb_clust)
```

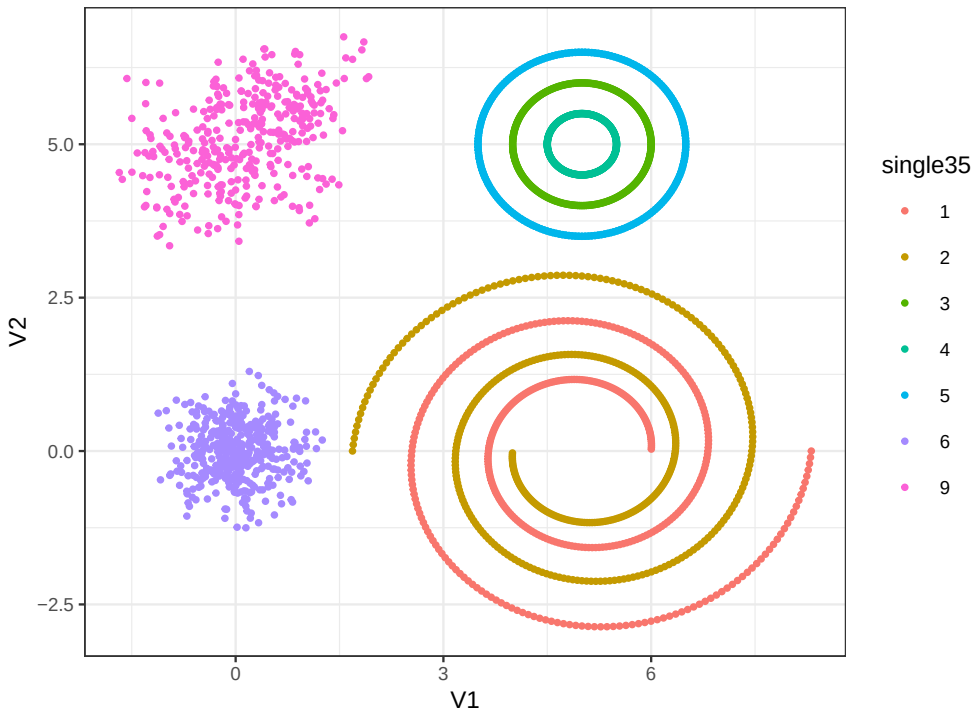


Elle provient de la séparation des spirales. L'identification des spirales produit donc une baisse de la silouhette. Cela s'explique par le fait que ce coefficient n'est pas adapté à ce type de structures géométriques (il va privilégier des clusters à géométrie sphérique). On termine en visualisant la classification à 35 groupes, en ne conservant que les "gros" groupes :

```
table(single35)
```

```
single35
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
400 400 400 400 400 398   1   1 349   1   1   1   4  20   2   1   1   2   1   1
 21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  1   1   1   1   1   1   1   1   2   1   1   1   1   1   1
```

```
keep_groupe <- which(table(single35)>=100)
tbl |>
  mutate(single35=single35) |>
  filter(single35 %in% keep_groupe) |>
  mutate(single35=as.factor(single35)) |>
  ggplot()+aes(x = V1, y = V2,color = single35) +
  geom_point()
```



Exercice 5.2 (CAH sur un gros jeu de données)**.** On reprend le même jeu de données mais avec plus d'individus :

```
tbl <- read_delim("data/donclassif2.txt",delim = ";")
dim(tbl)
```

```
[1] 70000     2
```

1. Que se passe t-il lorsque vous faites une CAH ?

   ```
   DD <- dist(tbl)
   ```

   ```
   Error: vector memory exhausted (limit reached?)
   ```

   Le nombre d'individus est trop important pour calculer la matrice des distances. Il est bien connu qu'on ne peut pas faire une CAH lorsque n est (trop) grand.

2. Proposer une solution pour faire quand même la CAH.

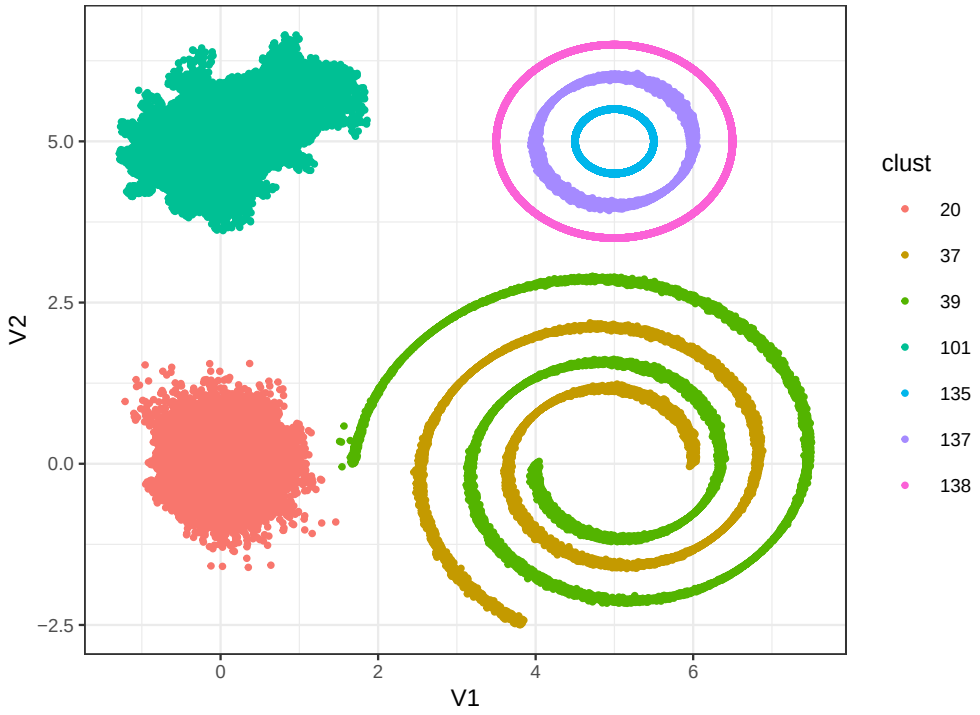
   La solution classique consiste à faire une ***classification mixte*** *:*

   - *Faire un k-means avec un nombre de groupes conséquent ;*

- *Faire le CAH sur les centroïdes du k-means (en prenant en compte la taille des clusters). Sur*

Sur ***R*** *on peut faire cela avec la fonction* `HCPC` *du package* `FactoMineR`*.*

```
library(FactoMineR)
classif <- HCPC(tbl,kk=2000,nb.clust = 150,
                method="single",
                description = FALSE,graph = FALSE)
keep_groupe <- which(table(classif$data.clust$clust)>1000)
as_tibble(classif$data.clust) |>
  filter(clust%in%keep_groupe) |> ggplot() +
  aes(x=V1,y=V2,color=clust)+geom_point()
```



6 Dbscan et clustering spectral

6.1 L'algorithme DBSCAN

L'algorithme DBSCAN (Density Based Spatial Clustering of Applications with Noise, Ester et al. (1996)) fait partie des méthodes basées sur la densité : les clusters correspondent à des zones de fortes densité séparées par des zones où la densité est plus faible. Ces zones sont définies par deux types de points :

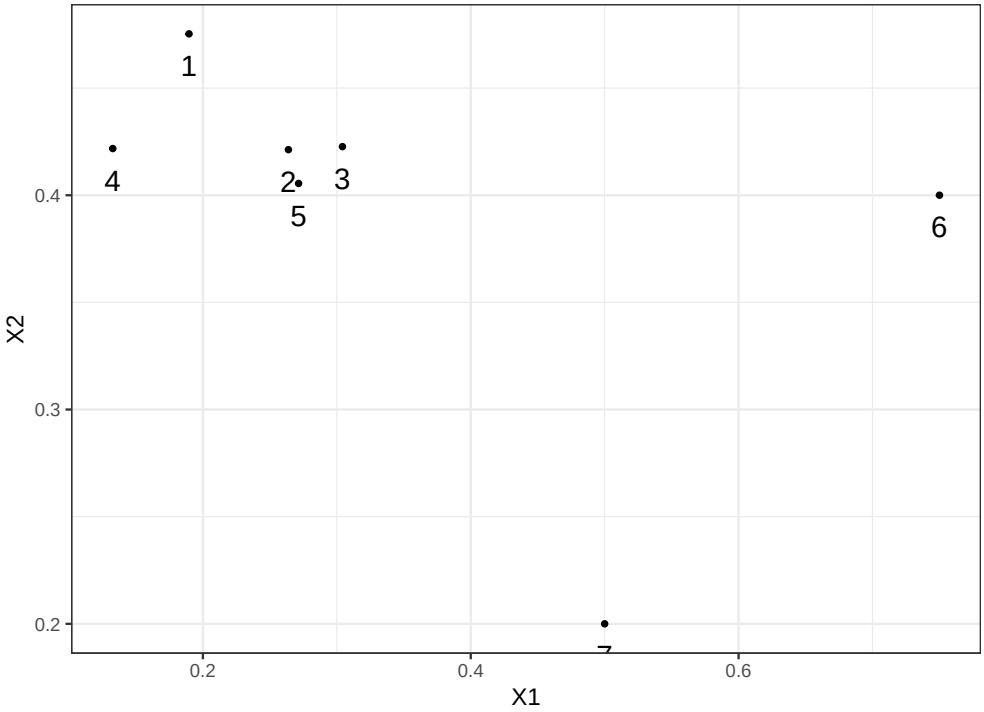
- les **noyaux** : des points qui contiennent plus de `minPts` points à une distance inférieure à `epsilon` ;
- les **points de bordure** : les points qui vont se situés en bordure des clusters ou qui sont isolés.

On charge le package

```
library(dbscan)
```

Exercice 6.1 (Noyaux et points de bordure)**.** On considère le "nuage" de points suivant :

```
set.seed(1234)
tbl <- tibble(X1=c(rnorm(5,mean=0.25,sd=0.05),0.75,0.5),
              X2=c(rnorm(5,mean=0.45,sd=0.05),0.4,0.2),
              label=as.character(1:7))


ggplot(tbl)+aes(x=X1,y=X2)+geom_point()+
  geom_text(aes(label=label),vjust=2)
```



1. On fixe `eps=0.13` et `minPts=4`. À l'aide de calculs simples, identifier les noyaux et points de bordure.

   Il suffit de calculer les distances entre individus et de compter le nombre de points à une distance inférieurs de ***`eps`*** *de chaque individu.*

```
dist(tbl[,1:2])
```

```
          1         2         3         4         5         6
2 0.0918129
3 0.1260868 0.0403751
4 0.0781420 0.1311574 0.1715093
5 0.1075430 0.0174946 0.0369921 0.1396929
6 0.5653904 0.4865933 0.4463539 0.6176689 0.4785753
7 0.4148624 0.3235954 0.2964966 0.4290494 0.3073463 0.3201562
```

   On déduit que 1, 2, 3 et 5 sont des noyaux, 4, 6 et 7 sont des points de bordure.

2. Retrouver ces résultats à l'aide de la fonction `is.corepoint`.

```
is.corepoint(tbl[,1:2],eps = 0.13,minPts = 4)
```

```
[1]  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
```

3. Effectuer l'algorithme **dbscan** avec ces valeurs de paramètre et interpréter.

```
(db <- dbscan(tbl[,1:2],eps = 0.13,minPts = 4))
```

```
DBSCAN clustering for 7 objects.
Parameters: eps = 0.13, minPts = 4
Using euclidean distances and borderpoints = TRUE
The clustering contains 1 cluster(s) and 2 noise points.

0 1
2 5

Available fields: cluster, eps, minPts, dist, borderPoints
```

```
db$cluster
```

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

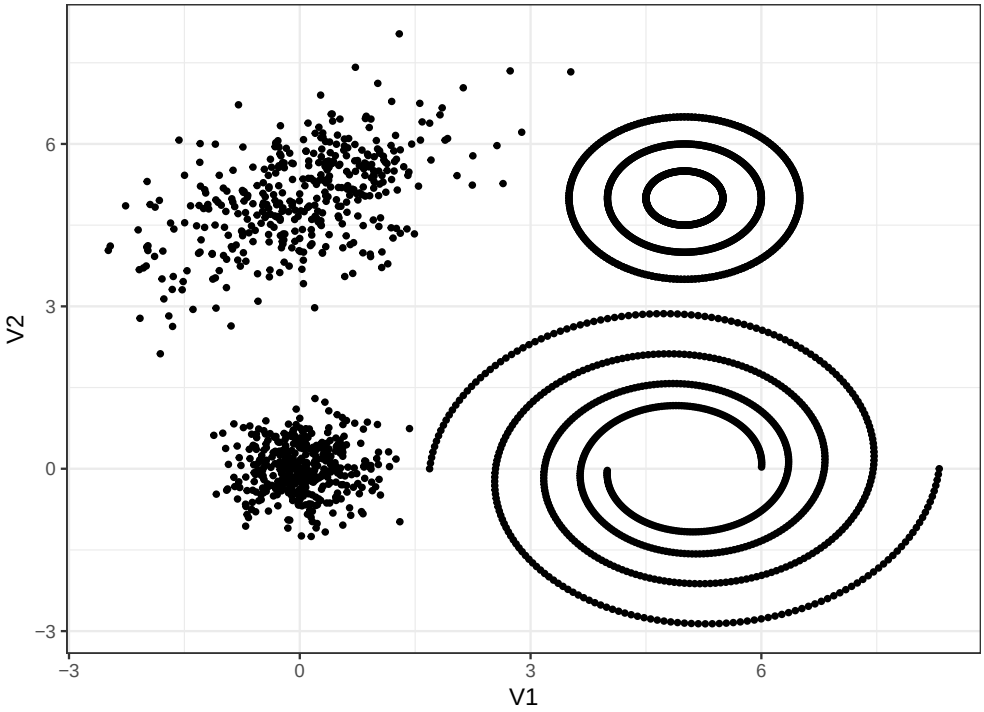
L'algorithme a identifié 1 cluster de 5 points et 2 outliers (6 et 7).

4. Est-ce que des points de bordure peuvent être affectés à des cluster ? Justifier.

   Oui ! On voit par exemple que 4 est dans le cluster. Ce point est en effet ***connecté*** *aux 4 autres. On voit par exemple qu'il est connecté avec 2 car 4 et 2 sont tous les deux accessibles depuis 1.*

Exercice 6.2 (Calibration de dbscan)**.** On reprend les données de l'Exercice 5.1 :

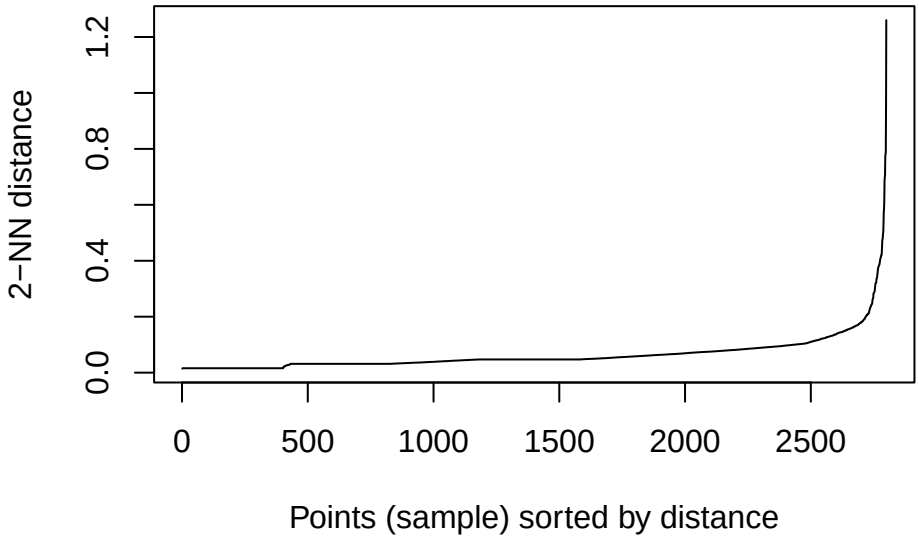
```
tbl <- read_delim("data/donclassif.txt",delim = ";")
ggplot(tbl)+aes(x=V1,y=V2)+geom_point()
```



En utilisant la stratégie proposée dans l'aide de `dbscan`, calibrer l'algorithme pour essayer d'identifier au mieux les différents clusters. On pourra envisager une deuxième étape pour affecter les petits clusters aux gros…

On fixe `minPts=3` et on regarde la distance entre chaque point et son deuxième plus proche voisin.

```
kNNdistplot(tbl,k=2)
```

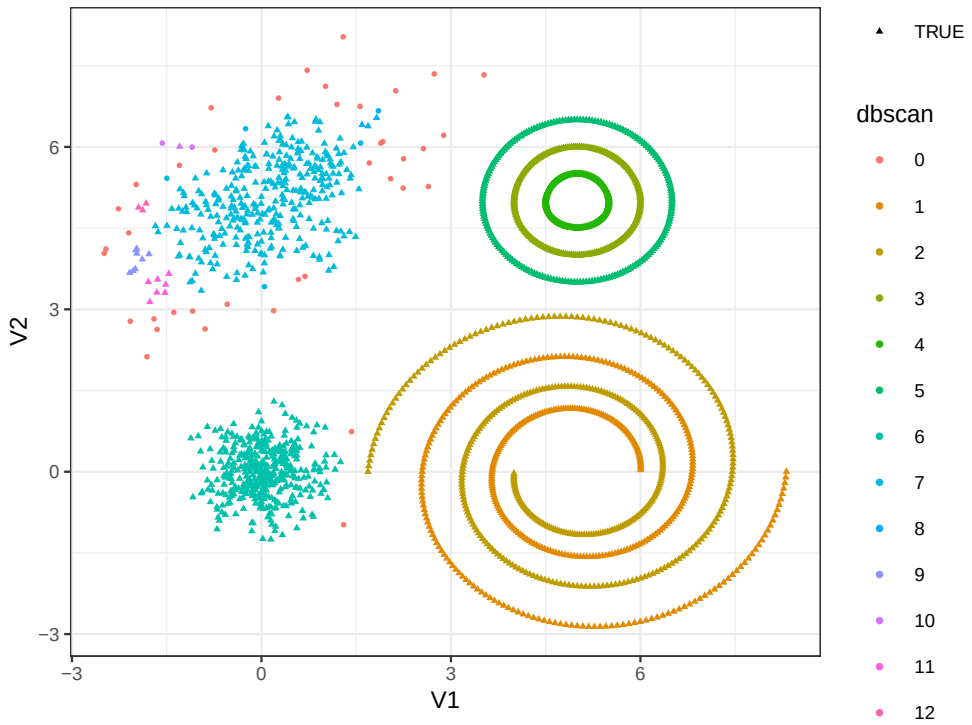


On choisit `eps=0.3` *et on visualise les résultats :*

```
(db1 <- dbscan(tbl,eps=0.3,minPts = 3))
```

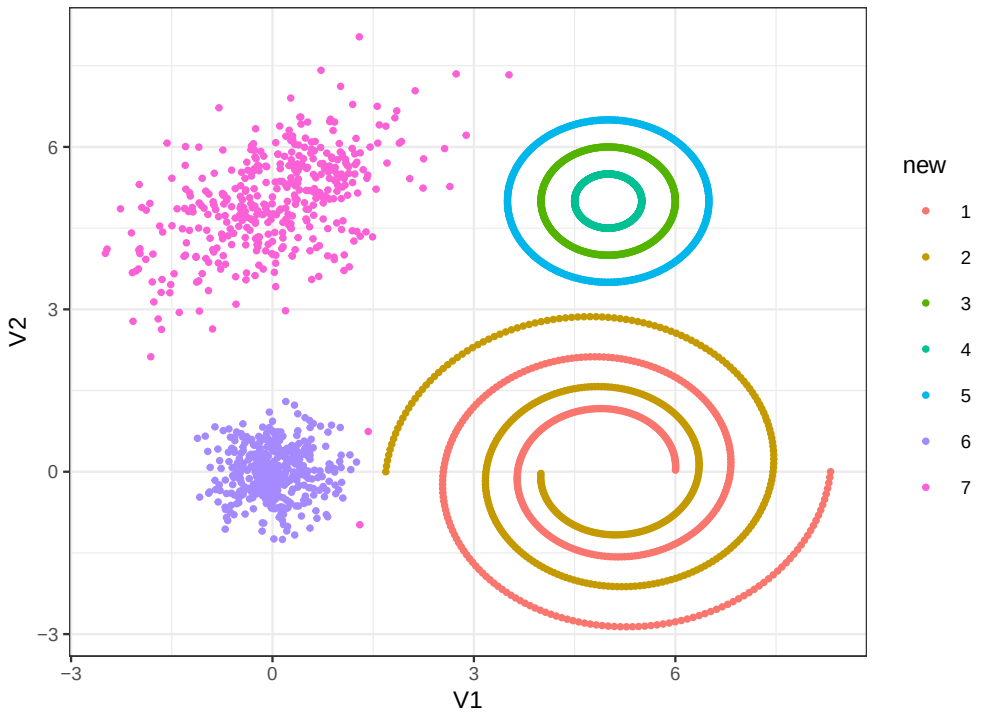
```
DBSCAN clustering for 2800 objects.
Parameters: eps = 0.3, minPts = 3
Using euclidean distances and borderpoints = TRUE
The clustering contains 12 cluster(s) and 39 noise points.

  0   1   2   3   4   5   6   7   8   9  10  11  12
 39 400 400 400 400 400 398 338   4   8   3   7   3

Available fields: cluster, eps, minPts, dist, borderPoints
```

```
noyau <- is.corepoint(tbl,eps=0.3,minPts = 3)
tbl_db <- tbl |> mutate(dbscan=as.factor(db1$cluster),noyau=noyau)
ggplot(tbl_db)+aes(x=V1,y=V2,color=dbscan,shape=noyau)+geom_point()
```



On remarque que les groupes 8 à 12 ont des effectifs très faibles. On propose de les affecter au "gros" groupe le plus proche en utilisant la distance du min *:*

```
library(LearnClust)
D <- rep(0,7)
tbl_db <- tbl_db |> mutate(new=dbscan)
for (i in c(0,8:12)){
  tbl1 <- tbl_db |> filter(dbscan %in% i) |> select(1:2) |> as.matrix()
  for (j in 1:7){
    tbl2 <- tbl_db |> filter(dbscan %in% j) |> select(1:2) |> as.matrix()
    D[j] <- clusterDistance(tbl1,tbl2,'MIN','MAN')
    tbl_db$new[tbl_db$dbscan==i] <- which.min(D)
  }
}
ggplot(tbl_db)+aes(x=V1,y=V2,color=new)+geom_point()
```



6.2 Clustering spectral

Le *clustering spectral* est un algorithme de classification non supervisé qui permet de définir des clusters de nœuds sur des graphes ou d'individus pour des données `individus/variables`. L'algorithme est basé sur la décomposition spectrale du Laplacien (normalisé) d'une matrice de similarité, il est résumé ci-dessous :

Entrées :

- tableau de données $n \times p$
- K un noyau
- k le nombre de clusters.

1. Calculer la matrice de **similarités** W sur les données en utilisant le **noyau** K
2. Calculer le **Laplacien normalisé** L_{norm} à partir de W.
3. Calculer les k **premiers vecteurs propres** $u_1, \dots, u_k$ de L_{norm}. On note U la matrice $n \times k$ qui les contient.
4. Calculer la matrice T en **normalisant les lignes** de U : $t_{ij} = u_{ij}/(\sum_\ell u_{i\ell}^2)^{1/2}$.
5. Faire un k-**means** avec les points $y_i, i = 1, \dots, n$ (i-*me* ligne de T) $\Longrightarrow A_1, \dots, A_k$.

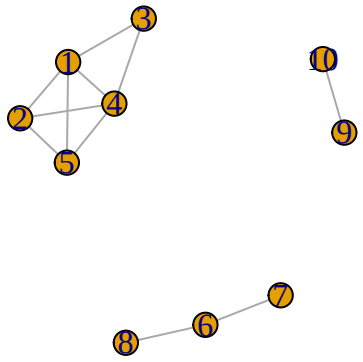
Sortie : clusters $C_1, \dots, C_k$ avec

$$C_j = \{i | y_i \in A_j\}.$$

L'objet de ce chapitre est de travailler sur cet algorithme en le programmant, puis en utilisant la fonction **specc** du package `kernlab`.

On crée tout d'abord un graphe avec trois composantes connexes : on utilise la commande **sample_gnp()** qui permet de créer un graphe selon le modèle d'Erdos-Renyi.

```
library(igraph)
set.seed(1)
n1 <- 5
n2 <- 3
n3 <- 2
n <- n1+n2+n3
# il faut prendre des grandes valeurs de p sinon on risque d'avoir des sous-graphes non connexes
p1 <- 0.85
p2 <- 0.75
p3 <- 0.7
G1 <- sample_gnp(n1,p1)
G2 <- sample_gnp(n2,p2)
G3 <- sample_gnp(n3,p3)
G <- G1 + G2 + G3 # il cree un graphe avec ces 3 sous-graphes
plot(G)
```



On vérifie le nombre de composantes connexes

```
components(G)$no
```

```
[1] 3
```

Exercice 6.3 (Laplacien non normalisé)**.**

1. Calculer la matrice d'adjacence de **G** et en déduire le Laplacien normalisé. On pourra utiliser la fonction `as_adj`.

```
A <- as_adj(G,sparse=F)
D <- diag(rowSums(A))
D_moins1_2 <- diag(1/sqrt(diag(D)))
LN <- diag(n) - D_moins1_2 %*% A %*%D_moins1_2
```

2. Retrouver ce Laplacien avec la fonction **laplacian_matrix**.

```
LN <- laplacian_matrix(G,norm=TRUE,sparse=F)
```

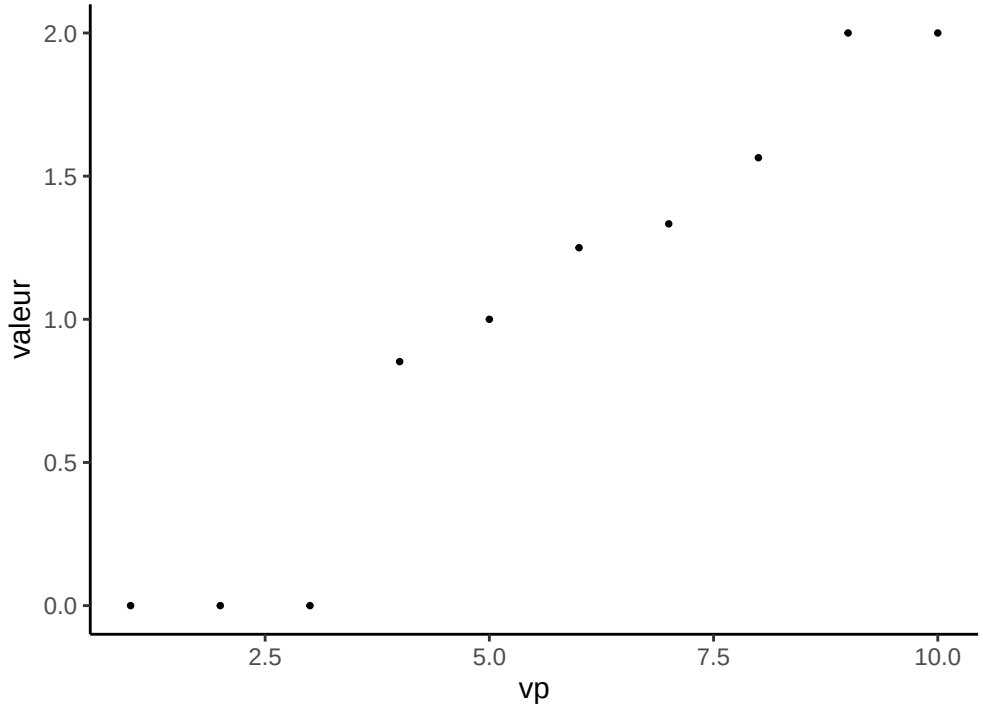
3. Calculer les valeurs propres et représenter les sur un graphe. Que remarquez-vous ?

```
specN <- eigen(LN)
specN$values
```

```
[1] 2.000000e+00 2.000000e+00 1.564333e+00 1.333333e+00 1.250000e+00
[6] 1.000000e+00 8.523332e-01 1.110223e-15 1.110223e-15 8.881784e-16
```

On observe un trou spectral entre les valeurs propres 3 et 4. Conformément à la théorie, l'ordre de multiplicité de la valeur propre 0 est égal au nombre de composantes connexes du graphe.

```
dfN <- tibble(vp=1:length(specN$values),valeur=rev(specN$values))
ggplot(dfN)+aes(x=vp,y=valeur)+geom_point()+theme_classic()
```



4. Obtenir les trois vecteurs propres associés à la valeur propre nulle. Commenter.

   On calcule les 3 vecteurs propres :

```
U <- specN$vectors[,n:(n-2)]
U
```

```
            [,1]      [,2]      [,3]
 [1,] -0.5000000 0.0000000 0.0000000
 [2,] -0.4330127 0.0000000 0.0000000
 [3,] -0.3535534 0.0000000 0.0000000
 [4,] -0.5000000 0.0000000 0.0000000
 [5,] -0.4330127 0.0000000 0.0000000
 [6,]  0.0000000 0.7071068 0.0000000
 [7,]  0.0000000 0.5000000 0.0000000
 [8,]  0.0000000 0.5000000 0.0000000
 [9,]  0.0000000 0.0000000 0.7071068
[10,]  0.0000000 0.0000000 0.7071068
```

5. Normaliser ces vecteurs. On pourra utiliser la fonction

```
normalize <- function(x){
  return(x/sqrt(sum(x^2)))
}
```

On normalise à l'aide de la fonction ***normalize*** *:*

```
U.norm <- t(apply(U,1,normalize))
U.norm
```

```
      [,1] [,2] [,3]
 [1,]   -1    0    0
 [2,]   -1    0    0
 [3,]   -1    0    0
 [4,]   -1    0    0
 [5,]   -1    0    0
 [6,]    0    1    0
 [7,]    0    1    0
 [8,]    0    1    0
 [9,]    0    0    1
[10,]    0    0    1
```

6. Terminer l'algorithme avec le k-means.

```
res <- kmeans(U.norm,3,nstart=100)
res$cluster
```

```
 [1] 1 1 1 1 1 2 2 2 3 3
```

Exercice 6.4 (Clustering spectral avec specc)**.** On reprend les données de l'Exercice 5.1 :

```
tbl <- read_delim("data/donclassif.txt",delim = ";")
```

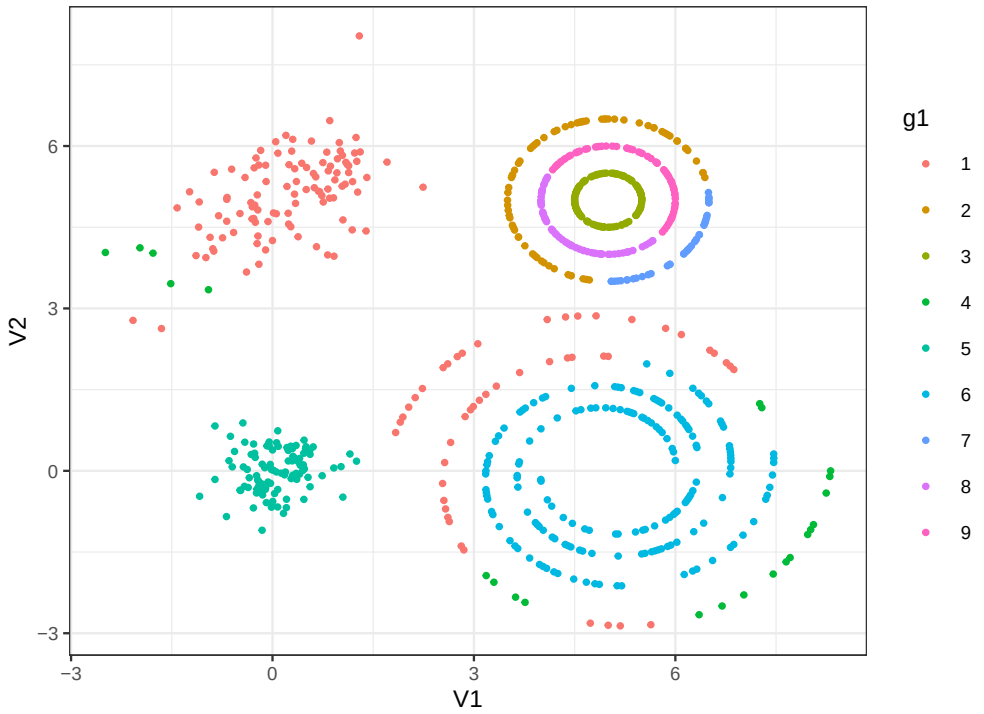
Faire le "meilleur" clustering spectral, on pourra utiliser plusieurs noyaux et plusieurs valeurs de paramètres.

1. On commencera par lancer la procédure sur un sous-échantillon de taille 1000
2. On proposera ensuite une autre solution qui permettra de traiter tous les observations.

On commence en tirant un sous-échantillon et un choisit 9 groupes.

```
set.seed(1234)
perm <- sample(nrow(tbl))
tbl1 <- tbl[perm[1:800],]
```

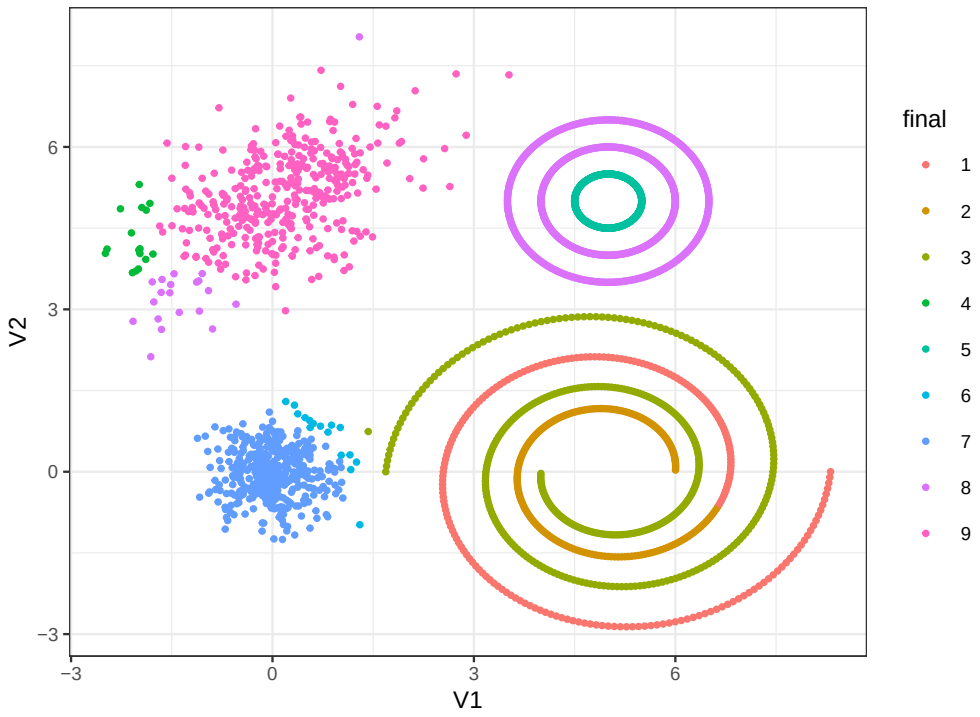
```
library(kernlab)
g1 <- specc(as.matrix(tbl1),centers=9,kpar=list(sigma=100))
tbl2 <- tbl1 |> mutate(g1=as.factor(g1))
ggplot(tbl2)+aes(x=V1,y=V2,color=g1)+geom_point()
```



Pour pouvoir appliquer l'algorithme à toutes les données on

1. *fait un k-means avec 1000 groupes*
2. *fait le clustering spectral sur les centroïdes*
3. *affecte tous les individus en fonction du k-means initial.*

```
e1 <- kmeans(tbl,centers=1000,nstart = 100,iter.max = 100)
centres <- e1$centers
g1 <- e1$cluster
e2 <- specc(centres,centers=9,kernel="rbfdot",
            kpar=list(sigma=100))
g2 <- e2[g1]
tbl1 <- tbl |> mutate(KM=g1,final=as.factor(g2))
ggplot(tbl1)+aes(x=V1,y=V2,color=final)+geom_point()
```


6
3
0
−3
V2
−3
0
3
6
V1
final
1
2
3
4
5
6
7
8
9

Références

Ester, M., H. P. Kriegel, J. Sander, et X. Xu. 1996. « A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise ». In.

Giraud, C. 2015. *Introduction to High-Dimensional Statistics*. CRC Press.