Introduction au Machine Learning

Pierre-Edouard Portier

Mars 2022

Table des matières

1 Introduction **7**
- 1.1 Contexte . . . 7
- 1.2 Objectifs . . . 7
- 1.3 Ajustement de courbe . . . 8
- 1.4 Interpolation polynomiale sur un jeu de données synthétique . . . 9
- 1.5 Annexe code source . . . 11

2 Méthode des moindres carrés **13**
- 2.1 Espace de fonctions . . . 13
- 2.2 Expression matricielle . . . 13
- 2.3 Méthode des moindres carrés appliquée à la régression polynomiale . . . 14
- 2.4 Annexe code source . . . 15

3 Bréviaire proba/stat **17**
- 3.1 Probabilités . . . 17
  - 3.1.1 Espérance . . . 17
  - 3.1.2 Variance . . . 18
- 3.2 Distribution normale . . . 18
- 3.3 Statistiques . . . 18
- 3.4 Théorème Central Limite (CLT) . . . 19
- 3.5 Intervalles de confiance . . . 19
- 3.6 Tests d'hypothèses . . . 19
- 3.7 Application aux coefficients d'un modèle linéaire . . . 20
- 3.8 Indicateur R^2 de la qualité d'une régression linéaire . . . 21
- 3.9 Mesure des colinéarités par le facteur d'inflation de la variance . . . 21
- 3.10 Effet levier des observations sur les prédictions . . . 21

4 Application au jeu de données `abalone` **23**
- 4.1 Récupération du jeu de données . . . 23
- 4.2 Préparation du jeu de données . . . 24
- 4.3 Modèle linéaire . . . 25
- 4.4 Analyse graphique du modèle linéaire . . . 26
- 4.5 Interprétation du modèle linéaire . . . 30

5 Régularisation de Tikhonov **31**
- 5.1 Problèmes linéaires mal posés . . . 31

5.2 Ajout de contraintes de régularité . . . 31
5.3 Standardisation . . . 32
5.4 Visulation de l'effet sur les paramètres de différents niveaux de régularisation . . . 33
5.5 Régularisation et complexité . . . 34
5.6 Annexe code source . . . 35

6 Validation croisée 37
6.1 Principe de la validation croisée . . . 37
6.2 Application de la validation croisée à la régularisation de Tikhonov . . . 38
6.3 Annexe code source . . . 40

7 Présentation de la décomposition en valeurs singulières (SVD) 43
7.1 Contexte . . . 43
7.2 Recherche d'un sous-espace qui minimise les carrés des écarts à l'espace d'origine . . . 43
7.3 Illustrations du SVD avec la compression d'une image . . . 46
7.4 Annexe code source . . . 49

8 SVD et Analyse en Composantes Principales 51
8.1 Projection sur les axes principaux . . . 51
8.1.1 SVD . . . 51
8.1.2 Facteurs . . . 51
8.1.3 Contributions des observations . . . 51
8.1.4 Contributions des axes . . . 52
8.1.5 Contribution des variables aux axes principaux . . . 52
8.2 Implémentation . . . 52
8.3 Exemple . . . 54

9 Matrice définie non négative 59

10 Optimisation sous contraintes et multiplicateurs de Lagrange 63

11 Dérivation du SVD 65

12 Plus grande valeur propre et puissance itérée 69
12.1 Décomposition en valeurs propres d'une matrice carrée . . . 69
12.2 Plus grande valeur propre et méthode de la puissance itérée . . . 70

13 Méthode de la puissance itérée et SVD 73

14 Projecteurs orthogonaux et SVD 77
14.1 Projecteur . . . 77
14.2 Projecteur orthogonal . . . 79
14.3 Projection sur une base quelconque . . . 81

15 Factorisation QR 83
15.1 Factorisation QR . . . 83
15.2 Orthogonalisation de Gram-Schmidt . . . 84
15.3 Algorithme de Gram-Schmidt modifié . . . 84
15.3.1 Dérivation de l'algorithme de Gram-Schmidt modifié . . . 84

15.3.2 Test de l'algorithme . . . 85
15.3.3 Comparaison de la stabilité numérique des deux versions de l'algorithme de Gram-Schmidt . . . 86
15.4 Autres algorithmes . . . 87
15.5 Décomposition QR pour résoudre un système linéaire au sens des moindres carrés . 87
15.6 Annexe code source . . . 87

16 Puissance itérée par bloc et SVD 89
16.1 Principe et application à la décomposition en valeurs propres . . . 89
16.2 Application à la décomposition en valeurs singulières . . . 90

17 Géométrie de la régression ridge et SVD 91
17.1 Coefficients de la régression ridge en fonction du SVD . . . 91
17.2 Régression ridge et géométrie . . . 95
17.3 Annexe code source . . . 95

18 Validation croisée un contre tous 99
18.1 Annexe code source . . . 104

19 Biais et variance d'un estimateur 107
19.1 Biais et variance pour l'estimateur d'un paramètre d'un modèle paramétrique . . . 107
19.2 Estimateurs par maximum de vraissemblance . . . 107
19.2.1 Exemple d'une loi normale . . . 108
19.3 Analyse biais-variance pour la régression . . . 112

20 Biais et variance des paramètres d'une régression ridge 115
20.1 Biais des coefficients d'un modèle de régression ridge . . . 115
20.2 Variance des coefficients d'un modèle de régression ridge . . . 116
20.2.1 Variance des coefficients d'un modèle de régression linéaire . . . 116
20.2.2 Relation linéaire entre $\hat{\boldsymbol{\beta}}_\lambda$ et $\hat{\boldsymbol{\beta}}$ . . . 118
20.2.3 Variance des coefficients d'un modèle de régression linéaire régularisée . . . 119

21 Régression ridge à noyau 121
21.1 Noyau gaussien . . . 121
21.2 Régression ridge à noyau . . . 123
21.3 Calcul des paramètres de la régression ridge à noyau . . . 125
21.4 LOOCV pour le choix de l'hyper-paramètre λ . . . 125
21.5 Choix de l'hyper-paramètre σ^2 . . . 127
21.6 Exemple sur un jeu de données synthétique . . . 127
21.7 Annexe code source . . . 128

22 Approximation de Nyström 131
22.1 Approximation de Nyström d'un noyau . . . 131
22.2 Calcul de l'approximation de Nyström . . . 132
22.3 Régression ridge à noyau et approximation de Nyström . . . 133

23 TP - Régression ridge et dilemme biais-variance - Sujet 135
23.1 Générer un jeu de données synthétique . . . 135

23.2 Calculer les coefficients d'une régression ridge . . . 135
23.3 Espérance de l'erreur de prédiction . . . 136

24 TP - Régression ridge et dilemme biais-variance - Correction 137
24.1 Générer un jeu de données synthétique . . . 137
24.2 Calculer les coefficients d'une régression ridge . . . 137
24.3 Espérance de l'erreur de prédiction . . . 139

25 TP - Projection aléatoire - Sujet 143
25.1 Générer un jeu de données synthétique avec la fonction `sinc` . . . 143
25.2 Régression ridge après projections non-linéaires aléatoires . . . 143
25.3 Jeu de données `housing` . . . 144

26 Projection aléatoire - `sinc` - Correction 145
26.1 Générer un jeu de données synthétique avec la fonction `sinc` . . . 145
26.2 Régression ridge après projections non-linéaires aléatoires . . . 146

27 TP - Projection aléatoire - `housing` PCA - Correction 149
27.1 Présentation du jeu de données `housing` . . . 149
27.2 Prétraitements . . . 150
27.3 Analyse exploratoire par PCA . . . 151
27.3.1 Première analyse . . . 151
27.3.2 Seconde analyse . . . 153
27.3.3 Troisième analyse . . . 156
27.3.4 Quatrième analyse . . . 158

28 TP - Projection aléatoire - `housing` ELM - Correction 161
28.1 Créations des jeux d'entraînement et de test . . . 161
28.2 Modèle ELM . . . 162
28.3 Prédictions sur le jeu de test . . . 164

29 TP - Régression ridge à noyau - Sujet 165
29.1 Analyse de l'effet du paramètre σ^2 pour un noyau gaussien . . . 165
29.2 Régression ridge à noyau sur un petite exemple synthétique . . . 166

30 TP - Régression ridge à noyau - Correction 167
30.1 Analyse de l'effet du paramètre σ^2 pour un noyau gaussien . . . 167
30.2 Régression ridge à noyau sur un petite exemple synthétique . . . 169

31 Références 173

Chapitre 1

Introduction

1.1 Contexte

A l'occasion des trois premières révolutions industrielles, des tâches, auparavant réservées au travail manuel de l'Homme, ont été automatisées. Il semble envisageable d'associer au tournant du 21ème siècle une quatrième révolution portée par l'automatisation de la capacité à prédire, essentielle pour le processus de prise de décision dans les secteurs de l'industrie, du commerce et des services.

Cette transformation se fonde sur des évolutions scientifiques et techniques majeures. Elle est ainsi associée à une discipline, le machine learning ou apprentissage automatique de modèles prédictifs par extrapolation à partir de données générées par des processus physiques, numériques ou biologiques. Ces développements algorithmiques, en particulier la redécouverte des réseaux de neurones profonds, ont révélé sous un nouveau jour leur potentiel autour des années 2010 grâce, d'une part, à la création de jeux de données volumineux dans des domaines variés comme la reconnaissance de la parole, la vision par ordinateur, les données multimedia, le traitement de la langue naturelle, la robotique, les véhicules autonomes... et, grâce d'autre part, à une croissance rapide des capacités de calcul et de stockage aux coûts toujours plus abordables.

Par ailleurs, cette automatisation des prédictions s'accompagne d'un renouveau des formes de jugement dans les processus de prise de décision avec un couplage de plus en plus fin entre d'un côté, des experts humains et de l'autre, des chaînes de traitement automatique des données qui aboutissent sur la mise en production d'algorithmes prédictifs. Pour la réussite de cette intégration, les compétences de l'ingénieur informaticien sont essentielles. Ce dernier, puisqu'il comprend en profondeur le fonctionnement et les limites des algorithmes qu'il déploie, est capable d'en mesurer les risques et les biais pour éclairer le jugement de ceux qui utiliseront ses réalisations logicielles pour prendre des décisions.

Ainsi, ce cours introductif à l'apprentissage automatique a pour objectif d'offrir des connaissances fondamentales et des compétences pratiques qui aideront l'ingénieur à tenir ce rôle essentiel.

1.2 Objectifs

La discipline de l'apprentissage automatique, ou machine learning, élabore des algorithmes et des méthodes pour découvrir des régularités dans des données multidimensionnelles afin, entre autres,

d'automatiser la prédiction. Elle peut se subdiviser en trois catégories. D'abord, l'apprentissage non (ou semi) supervisé qui s'attache à découvrir des structures dans les données non étiquetées à travers des approches comme le clustering, la réduction dimensionnelle, les modèles génératifs... Ensuite, l'apprentissage par renforcement, dans le cadre duquel un agent interagit avec son environnement en adaptant son comportement pour maximiser une fonction de récompense. Enfin, l'apprentissage supervisé, qui fait l'objet de ce module, a quant à lui pour objectif d'apprendre à prédire l'association entre un objet décrit selon plusieurs dimensions et une étiquette.

Par exemple, il peut s'agir d'associer aux quartiers d'une ville le prix médian d'un logement. Dans ce cas, un quartier peut être décrit par la proportion de zones résidentielles, le taux de criminalité, le nombre moyen de pièces par habitat, etc. Ici, nous faisons face à un problème dit de « régression » où la valeur à prédire, autrement l'étiquette associée à chaque observation, est continue. Lorsque la variable à prédire est discrète, il s'agit d'un problème dit de « classification », comme détecter un objet dans une image ou décider si une transaction bancaire risque d'être une fraude. Nous considérerons ces deux catégories de problèmes.

Ainsi, ce cours a pour objectif d'introduire quelques concepts fondamentaux de l'apprentissage supervisé et de montrer leurs interconnexions variées dans le cadre de développements algorithmiques qui permettent d'analyser des jeux de données dans une visée avant tout prédictive. Ainsi, les propositions théoriques mèneront à l'écritures de programmes qui implémentent ou utilisent quelques modèles essentiels de l'apprentissage supervisé.

Pour faciliter l'acquisition des connaissances, le cours est accompagné de notebooks manipulables, rédigés dans le langage de programmation R. Ces mises en pratique systématiques doivent permettre de faire le lien entre des concepts fondamentaux et leur application dans des projets d'analyse de données.

1.3 Ajustement de courbe

$\mathbf{x}^{(1)} \dots \mathbf{x}^{(n)}$ sont des vecteurs de $\mathbb{R}^p$ associés aux valeurs, aussi appelées étiquettes, $y^{(1)} \dots y^{(n)}$ de $\mathbb{R}$. Nous cherchons une fonction $f(\mathbf{x}) : \mathbb{R}^p \to \mathbb{R}$ qui modélise la relation entre les observations $\mathbf{x}$ et les étiquettes y.

La fonction f peut avoir une forme paramétrique, comme par exemple :

$$f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

Si $n = p + 1$, les paramètres $\beta_0, \beta_1, \dots \beta_p$ sont solution d'un système linéaire :

$$\begin{cases} y^{(1)} & = \beta_0 + \beta_1 x_1^{(1)} + \beta_2 x_2^{(1)} + \cdots + \beta_p x_p^{(1)} \\ y^{(2)} & = \beta_0 + \beta_1 x_1^{(2)} + \beta_2 x_2^{(2)} + \cdots + \beta_p x_p^{(2)} \\ \dots & = \dots \\ y^{(n)} & = \beta_0 + \beta_1 x_1^{(P)} + \beta_2 x_2^{(P)} + \cdots + \beta_p x_p^{(n)} \end{cases}$$

Ce système s'écrit également sous forme matricielle :

$$\begin{pmatrix} 1 & x_1^{(1)} & \dots & x_p^{(1)} \\ 1 & x_1^{(2)} & \dots & x_p^{(2)} \\ \dots & \dots & \dots & \dots \\ 1 & x_1^{(n)} & \dots & x_p^{(n)} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_p \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(n)} \end{pmatrix}$$

Chaque ligne i de la matrice du terme de gauche de l'égalité ci-dessus est le vecteur ligne $\mathbf{x}^{(i)T}$ avec l'addition d'un premier terme constant qui correspond au paramètre β_0. En nommant cette matrice $\mathbf{X}^T$, le système linéaire ci-dessus s'écrit :

$$\mathbf{X}^T\boldsymbol{\beta} = \mathbf{y}$$

Soit le cas particulier où x est un scalaire et f est un polynôme de degré p :

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_p x^p$$

Avec $n = p + 1$ observations et les étiquettes associées $\left(x^{(k)}, y^{(k)}\right)$, les coefficients de ce polynôme sont solution d'un système linéaire :

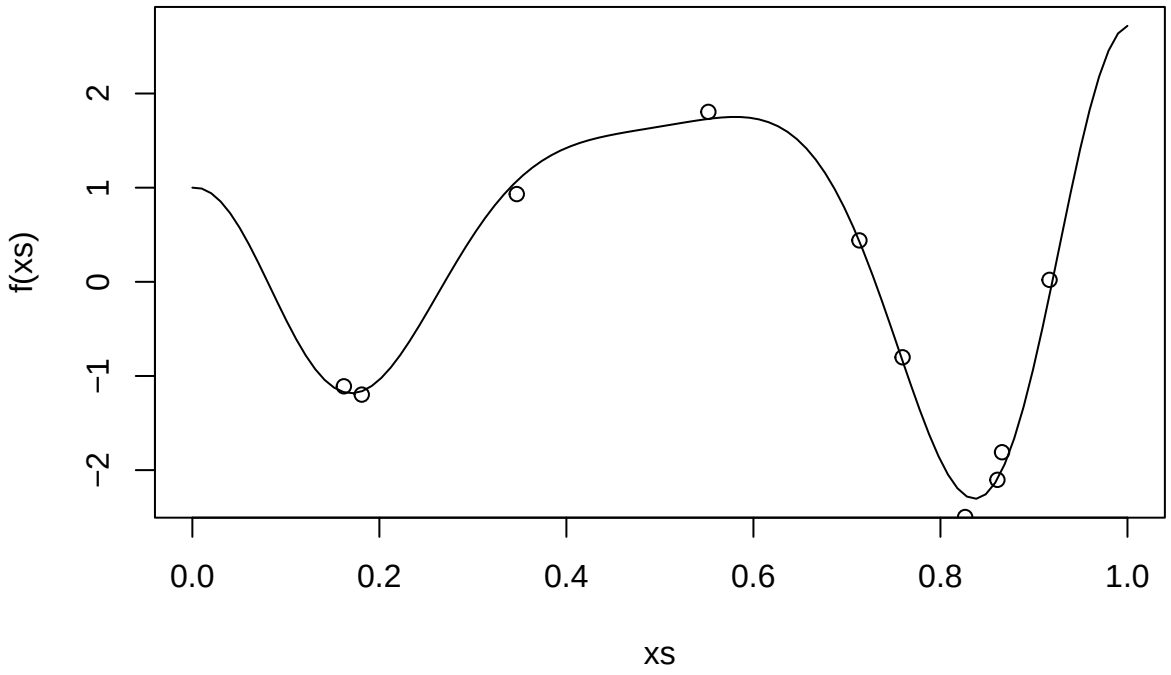
$$\begin{pmatrix} 1 & x^{(1)} & (x^{(1)})^2 & \dots & (x^{(1)})^p \\ 1 & x^{(2)} & (x^{(2)})^2 & \dots & (x^{(2)})^p \\ \dots & \dots & \dots & \dots & \\ 1 & x^{(n)} & (x^{(n)})^2 & \dots & (x^{(n)})^p \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_p \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(n)} \end{pmatrix}$$

La matrice du terme de gauche de l'égalité ci-dessus est traditionnellement appelée "matrice de Vandermonde".

1.4 Interpolation polynomiale sur un jeu de données synthétique

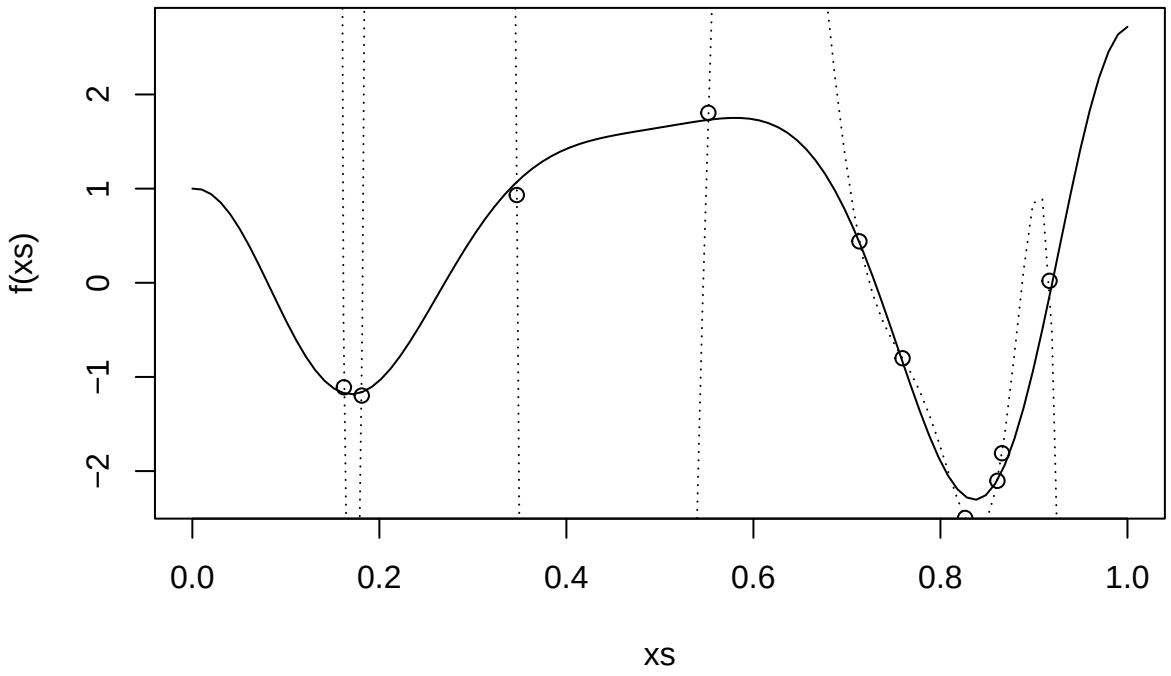
Soit un exemple de fonction non-linéaire, $f(x) = e^x \times cos(2\pi \times sin(\pi x))$, utilisée pour générer un jeu de données synthétique.

```
set.seed(1123)
# Image par f d'un échantillon uniforme sur l'intervalle [0,1], avec ajout d'un
# bruit gaussien de moyenne nulle et d'écart type 0.2
data = gendat(10,0.2)
plt(data,f)
```



Sur cet exemple, nous résolvons le système linéaire de Vandermonde pour découvrir un polynôme qui passe par chaque point du jeu de données.

```
# Résolution du système linéaire correspondant à la matrice de Vandermonde.
# coef contient les coefficients d'un polynôme qui passe par chaque point du jeu
# de données.
coef = polyreg1(data)
plt(data,f)
pltpoly(coef)
```



Il est improbable que ce polynôme, passant exactement par chaque observation, puisse offrir de bonnes capacités prédictives. Vérifier par exemple que, sur notre exemple synthétique, pour cinq

points générés à partir de la fonction f et avec l'ajout d'un bruit gaussien (par exemple d'écart type 0.2), le polynôme découvert, de degré quatre, peut être très éloigné de la fonction génératrice. C'est un exemple du phénomène de sur-apprentissage. Pour limiter ce problème, nous cherchons à découvrir un polynôme de degré plus faible. Il ne passera pas exactement par toutes les observations, mais il prédira probablement mieux les étiquettes associées à de nouvelles observations.

1.5 Annexe code source

```
# ML 1 01 Intro

# Exemple de fonction non-linéaire pour générer un jeu de données artificiel.
f <- function(x) {exp(x) * cos(2*pi*sin(pi*x))}

# Génération d'un jeu de données synthétique qui est l'image par f d'un
# échantillon uniforme de l'intervalle [0,1] auquel nous ajoutons un bruit
# gaussien de moyenne nulle et d'écart type sd.
gendat <- function(n, sd) {
  # n: nombre d'observations à générer
  # sd: écart type d'un bruit gaussien de moyenne nulle
  X = runif(n)
  Y = f(X) + rnorm(n, mean=0, sd=sd)
  dim(X) <- c(n,1) # en général chaque observation est décrite par plusieurs
  # variables et X est une matrice avec autant de lignes que
  # d'observations et autant de colonnes que de variables. Sur
  # notre exemple, chaque observation n'est décrite que par une
  # seule variable.
  list(X = X, Y = Y)
}

# Affichage simultanée du jeu de données bruité et de la courbe de la fonction
# utilisée pour le générer.
plt <- function(data, f, ...) {
  xs = seq(0,1,length.out=100)
  plot(xs, f(xs), type="l", ...)
  points(data$X, data$Y)
}

# Résolution du système linéaire correspondant à la matrice de Vandermonde.
# Autrement dit, découverte d'un polynôme qui passe par chaque point.
polyreg1 <- function(data) {
  xs <- c(data$X) # on transforme la matrice X, de dimension Nx1 sur notre
                  # exemple, en un vecteur
  vandermonde = outer(xs, 0:(length(xs)-1), "^")
  solve(vandermonde, data$Y)
}
```

```
# Evaluation d'un polynôme en un point.
polyeval <- function(coef,x) {
  powers = 0:(length(coef)-1)
  f = function(y) { sum(coef * y^powers) }
  sapply(x,f)
}

# Affichage de la courbe d'un polynôme défini par ses coefficients.
pltpoly <- function(coef) {
  xs = seq(0,1,length.out=100)
  lines(xs, polyeval(coef,xs), lty="dotted")
}
```

Chapitre 2

Méthode des moindres carrés

2.1 Espace de fonctions

Soit un espace vectoriel composé de fonctions. Une base de cet espace est un ensemble de fonctions $(f_1, f_2, \dots f_p)$ tel que toute fonction de l'espace s'exprime comme combinaison linéaire des fonctions de base.

$$f(\mathbf{x}) = \beta_1 f_1(\mathbf{x}) + \beta_2 f_2(\mathbf{x}) + \cdots + \beta_p f_p(\mathbf{x})$$

Pour un jeu de données $\left\{\mathbf{x^{(k)}}, y^{(k)}\right\}_{k=1}^{n}$ de taille $n = p$, les coefficients β_i sont solution d'un système linéaire.

$$\begin{pmatrix} f_1(\mathbf{x^{(1)}}) & f_2(\mathbf{x^{(1)}}) & \dots & f_p(\mathbf{x^{(1)}}) \\ f_1(\mathbf{x^{(2)}}) & f_2(\mathbf{x^{(2)}}) & \dots & f_p(\mathbf{x^{(2)}}) \\ \dots & \dots & \dots & \dots \\ f_1(\mathbf{x^{(n)}}) & f_2(\mathbf{x^{(n)}}) & \dots & f_p(\mathbf{x^{(n)}}) \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \dots \\ \beta_p \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(n)} \end{pmatrix}$$

Nous notons ce système linéaire $\mathbf{X}\boldsymbol{\beta} = \mathbf{y}$.

2.2 Expression matricielle

Le système linéaire $\mathbf{X}\boldsymbol{\beta} = \mathbf{y}$ avec $\mathbf{X} \in \mathbb{R}^{n \times p}$ n'a pas de solution quand le nombre d'observations dépasse le nombre de fonctions de base (c'est-à-dire, $n > p$). Une approche possible est alors de chercher une approximation $\mathbf{X}\boldsymbol{\beta} \approx \mathbf{y}$ qui minimise la somme des carrés des erreurs : $\|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2$.

$$
\begin{aligned}
& \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2 \\
= & \{\|\boldsymbol{\beta}\|_2 = \sqrt{\boldsymbol{\beta} \cdot \boldsymbol{\beta}}\} \\
& (\mathbf{X}\boldsymbol{\beta} - \mathbf{y}) \cdot (\mathbf{X}\boldsymbol{\beta} - \mathbf{y}) \\
= & \{\text{Par définition du produit scalaire euclidien}\} \\
& (\mathbf{X}\boldsymbol{\beta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{y}) \\
= & \{\text{propriété de la transposition}\} \\
& \left(\boldsymbol{\beta}^T\mathbf{X}^T - \mathbf{y}^T\right) (\mathbf{X}\boldsymbol{\beta} - \mathbf{y}) \\
= & \{\text{multiplication}\} \\
& \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\boldsymbol{\beta} + \mathbf{y}^T\mathbf{y} \\
= & \{\mathbf{y}^T\mathbf{X}\boldsymbol{\beta} \text{ étant une valeur scalaire, } \mathbf{y}^T\mathbf{X}\boldsymbol{\beta} = \left(\mathbf{y}^T\mathbf{X}\boldsymbol{\beta}\right)^T = \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y}\} \\
& \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - 2\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} + \mathbf{y}^T\mathbf{y}
\end{aligned}
$$

Cette dernière expression quadratique en $\boldsymbol{\beta}$ correspond à une surface convexe. Donc son minimum peut être calculé en annulant sa dérivée (penser à une courbe $y = a + bx + cx^2$ dont l'unique extremum est atteint lorsque la pente est nulle).

$$
\begin{aligned}
& \mathbf{0} = 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - 2\mathbf{X}^T\mathbf{y} \\
= & \\
& \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}
\end{aligned}
$$

Ainsi, quand $n > p$, la solution approximée $\hat{\boldsymbol{\beta}}$, telle que $\mathbf{X}\hat{\boldsymbol{\beta}} \approx \mathbf{y}$ par minimisation de la somme des carrés des erreurs, est la solution du système linéaire suivant où $\mathbf{X}^T\mathbf{X}$ est appelée la matrice de Gram.

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}$$

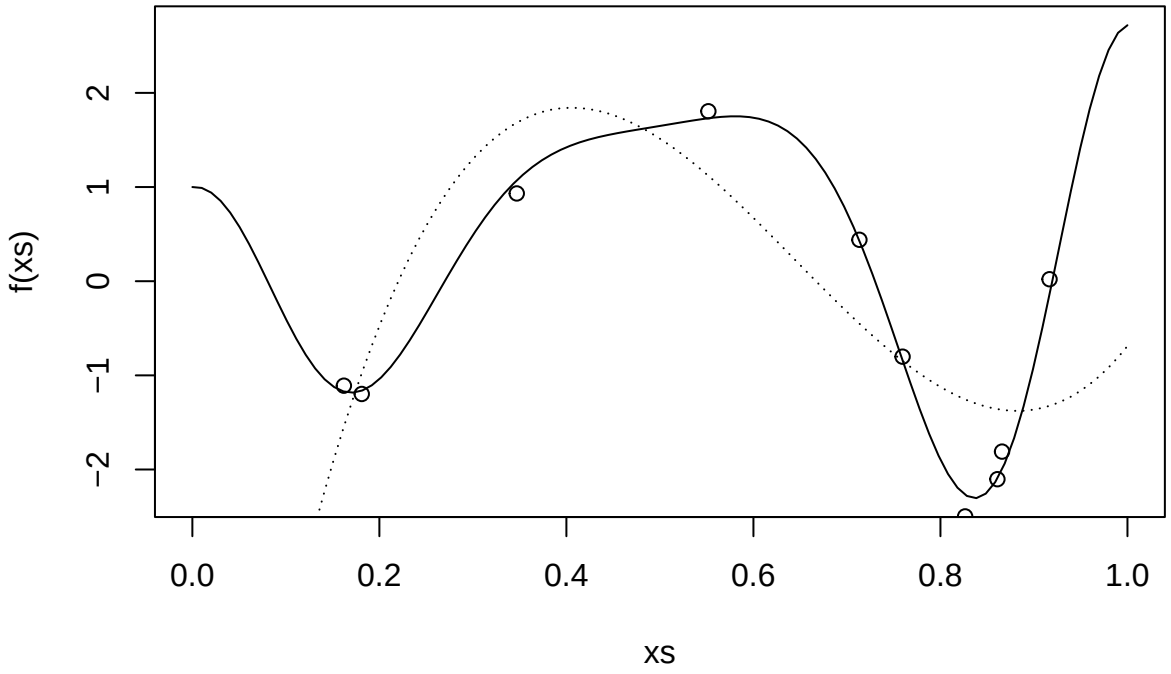
2.3 Méthode des moindres carrés appliquée à la régression polynomiale

Pour un polynôme de degré $p-1$, les fonctions de base mentionnées ci-dessus sont : $f_1(x) = 1$, $f_2(x) = x$, $f_3(x) = x^2$,..., $f_p(x) = x^{p-1}$. Elles permettent de définir la matrice des données $\mathbf{X}$ et la matrice de Gram $\mathbf{X}^T\mathbf{X}$.

Nous reprenons l'exemple synthétique du précédent chapitre et nous résolvons le système linéaire correspondant à la matrice de Gram pour un polynôme de degré fixé.

```
set.seed(1123)
# Image par f d'un échantillon uniforme sur l'intervalle [0,1], avec ajout d'un
# bruit gaussien de moyenne nulle et d'écart type 0.2
```

```
data = gendat(10,0.2)
coef = polyreg2(data,3)
plt(data,f)
pltpoly(coef)
```



Ce polynôme de degré trois modélise mieux la fonction génératrice inconnue que celui de degré quatre qui ne commettait aucune erreur sur les données observées.

2.4 Annexe code source

```
# 02 Moindres carrés
# Résolution du système linéaire correspondant à la matrice de Gram pour un
# polynôme de degré fixé.
polyreg2 <- function(data, degre) {
  xs <- c(data$X)
  A = outer(xs, 0:degre, "^")
  gram = t(A) %*% A
  solve(gram, as.vector(t(A) %*% data$Y))
}
```

Chapitre 3

Bréviaire proba/stat

3.1 Probabilités

3.1.1 Espérance

$$
\begin{aligned}
E[X] &= \lim_{n\to\infty} \frac{X_1 + \cdots + X_n}{n} \\
E[X] &= \sum_{c\in A} cP(X=c) \quad X \text{ var aléatoire discrète de support } A \\
E[U+V] &= E[U] + E[V] \\
E[aU] &= aE[U] \\
E[g(X)] &= \sum_{c\in A} g(c)P(X=c) \\
E[UV] &= E[U]E[V] \quad \text{si } U \text{ et } V \text{ sont indépendantes}
\end{aligned}
$$

3.1.2 Variance

$$Var[U] = E[(U - EU)^2]$$
$$Var[U] = E[U^2] - (EU)^2$$
$$g(c) = E[(U - c)^2] \quad \text{Erreur de l'estimation de } U \text{ par } c.\ c = E[x] \text{ minimise } g(c).$$
$$Var[cU] = c^2 Var[U]$$
$$Var[U + d] = Var[U]$$
$$P(|X - \mu| \geq c\sigma) \leq \frac{1}{c^2} \quad \text{Inégalité de Chebychev pour } X \text{ var aléatoire de moyenne } \mu \text{ et variance } \sigma^2.$$
$$\text{coef. var.} = \frac{\sqrt{Var[X]}}{E[X]} \quad \text{L'amplitude de } Var \text{ dépend de celle de } E. \text{ Le coef de variation est sans dime}$$
$$Cov[U, V] = E[(U - EU)(V - EV)]$$
$$Cov[U, V] = E[UV] - E[U]E[V]$$
$$Var[aU + bV] = a^2 Var[U] + b^2 Var[V] + 2abCov[U, V]$$
$$\rho(U, V) = \frac{Cov[U, V]}{\sqrt{Var[U]}\sqrt{Var[V]}},\ -1 \leq \rho(U, V) \leq 1 \quad \text{Corrélation}$$

3.2 Distribution normale

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} exp\left[-0.5\left(\frac{t - \mu}{\sigma}\right)^2\right], -\infty < t < +\infty$$
$$X \sim \mathcal{N}(\mu, \sigma^2) \quad X \text{ est distribué selon } \mathcal{N}(\mu, \sigma^2)$$
$$\text{Si } X_i \sim \mathcal{N}(\mu_i, \sigma_i^2) \text{ et } Y = a_1 X_1 + \cdots + a_k X_k \text{ alors } Y \sim \mathcal{N}\left(\sum_{i=1}^{k} a_i \mu_i, \sum_{i=1}^{k} a_i^2 \sigma_i^2\right)$$
$$\text{Si } X \sim \mathcal{N}(\mu, \sigma^2) \text{ alors } Z \sim \mathcal{N}(0, 1) \text{ avec } Z = \frac{X - \mu}{\sigma}$$

3.3 Statistiques

Pour des X_i indépendantes identiquement distribuées, de moyenne μ et de variance σ^2 :

$$\bar{X} = \frac{X_1 + \cdots + X_n}{n} \quad \text{Estimateur de la moyenne de la population sur un échantillon.}$$
$$E[\bar{X}] = \mu \quad \text{Estimateur non biaisé de l'espérance.}$$
$$Var[\bar{X}] = \frac{1}{n}\sigma^2$$
$$s^2 = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \bar{X}\right)^2 = \frac{1}{n}\sum_{i=1}^{n} X_i^2 - \bar{X}^2$$
$$E[s^2] = \frac{n - 1}{n}\sigma^2 \quad \text{Estimateur biaisé de la variance.}$$
$$\sigma/\sqrt{n} = \text{écart type de } \bar{X}, \text{ aussi appelé erreur standard}$$

3.4 Théorème Central Limite (CLT)

Si $X_1, \ldots, X_n$ sont des variables aléatoires indépendantes qui suivent la même distribution de moyenne m et de variance v^2, alors $T = X_1 + \cdots + X_n$ suit approximativement $\mathcal{N}(nm, nv^2)$. L'approximation est en pratique bonne à partir d'environ $n = 25$.

Ainsi, $Z = \frac{\bar{X}-\mu}{\sigma/\sqrt{n}}$ suit approximativement $\mathcal{N}(0,1)$.

Notons $\Phi()$ la fonction de distribution cumulative (cdf) de $\mathcal{N}(0,1)$.

$$P\left(|\bar{X} - \mu| < 3\sigma/\sqrt{n}\right) = P(|Z| < 3) \approx 1 - 2\Phi(-3) = 1 - 0.0027 = 0.9973$$

Avec $\Phi(-3)$ donné par `pnorm(-3) =` 0.0013499.

L'approximation de Chebychev donnerait :

$$P\left(|\bar{X} - \mu| \geq 3\sigma/\sqrt{n}\right) \leq \frac{1}{3^2}$$

Soit :

$$P\left(|\bar{X} - \mu| < 3\sigma/\sqrt{n}\right) \geq \frac{8}{9} = 0.8889$$

3.5 Intervalles de confiance

C'est sur la base du CLT que nous déterminons des intervalles de confiance.

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}} \text{ suit approximativement } \mathcal{N}(0,1)$$
$$\Rightarrow \{qnorm(0.0025) = -1.96\}$$
$$0.95 \approx P\left(-1.96 < \frac{\bar{X} - \mu}{\sigma/\sqrt{n}} < 1.96\right)$$
$$= \{\text{Arithmétique}\}$$
$$0.95 \approx P\left(\bar{X} - 1.96\frac{\sigma}{\sqrt{n}} < \mu < \bar{X} + 1.96\frac{\sigma}{\sqrt{n}}\right)$$

Dans cette formule, il est juste de remplacer l'écart-type σ, inconnu, par son estimation s.

3.6 Tests d'hypothèses

— Faire une hypothèse : $H_0 : \theta = c$
— Calculer $Z = \frac{\hat{\theta}-c}{\text{s.e.}(\hat{\theta})}$
— Rejeter H_0 à un niveau de risque $\alpha = 0.005$ si $|Z| >= 1.96$
— La p-valeur est le plus faible niveau de risque auquel l'hypothèse serait rejetée.

3.7 Application aux coefficients d'un modèle linéaire

Sous hypothèse d'un modèle génératif linéaire :

$$y_i = \mathbf{X}\boldsymbol{\beta} + \epsilon_i \quad , \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2) \ , \ Cov\left(\mathbf{y}|\mathbf{X}\right) = \sigma^2\mathbf{I}$$

Nous avons montré que :

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Nous calculons l'espérance et la variance de cet estimateur $\hat{\boldsymbol{\beta}}$.

$$\begin{aligned}
& E\left[\hat{\boldsymbol{\beta}}\right] \\
= & \{\text{Estimation de } \boldsymbol{\beta}, \text{ voir ci-dessus.}\} \\
& E\left[\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}\right] \\
= & \{\text{Hypothèse : } \mathbf{X} \text{ n'est pas une variable aléatoire. Linéarité de } E.\} \\
& \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T E\left[\mathbf{y}\right] \\
= & \{\text{Hypothèse du modèle génératif linéaire, voir ci-dessus.}\} \\
& \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \\
= & \{\text{Algèbre linéaire.}\} \\
& \boldsymbol{\beta}
\end{aligned}$$

$$\begin{aligned}
& Cov\left[\hat{\boldsymbol{\beta}}\right] \\
= & \{\text{Estimation de } \boldsymbol{\beta}, \text{ voir ci-dessus.}\} \\
& Cov\left[\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}\right] \\
= & \{\text{Propriétés de la variance.}\} \\
& \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T Cov\left[\mathbf{y}\right]\left(\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\right)^T \\
= & \{\text{Hypothèse du modèle génératif linéaire, voir ci-dessus.}\} \\
& \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\sigma^2\mathbf{I}\left(\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\right)^T \\
= & \{\text{Algèbre linéaire.}\} \\
& \sigma^2\left(\mathbf{X}^T\mathbf{X}\right)^{-1}
\end{aligned}$$

Par ailleurs,

$$\sigma^2 = Var[\mathbf{y}] = E\left[\left(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right)^2\right]$$

Donc, nous avons l'estimateur :

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{\beta}_0 - \hat{\beta}_1 X_1^{(i)} - \cdots - \hat{\beta}_p X_p^{(i)}\right)^2$$

Ainsi, un estimateur de la covariance est :

$$\hat{Cov}[\hat{\boldsymbol{\beta}}] = s^2 \left(\mathbf{X}^T\mathbf{X}\right)^{-1}$$

Les éléments diagonaux sont les carrés des erreurs standards pour les coefficients $\hat{\beta}_i$.

3.8 Indicateur R^2 de la qualité d'une régression linéaire

L'indicateur R^2 est le carré de la corrélation estimée entre $\hat{y}_i$ et y_i. Nous pouvons également dire que c'est le pourcentage de la variance de y_i expliquée par $\mathbf{x_i}$. Il y a une version ajustée de cet estimateur qui est sinon naturellement biaisé vers le haut.

3.9 Mesure des colinéarités par le facteur d'inflation de la variance

Pour mesurer les colinéarités entre la variable $\mathbf{x_1}$ et les variables $\mathbf{x_2}, \ldots, \mathbf{x_p}$, nous pouvons apprendre un modèle de la forme :

$$\mathbf{x_1} = \beta_0 + \beta_2\mathbf{x_2} + \cdots + \beta_p\mathbf{x_p} + \epsilon$$

Noton R_1^2 le carré de la corrélation estimée entre $\hat{\mathbf{x}}_1$ et $\mathbf{x_1}$. Nous pouvons alors calculer le facteur d'inflation de la variance :

$$VIF_1 = \frac{1}{1 - R_1^2}$$

En pratique, quand $VIF(\beta_i) > 10$, la colinéarité est importante.

3.10 Effet levier des observations sur les prédictions

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$$
$$= \{\text{Définition de } \hat{\boldsymbol{\beta}}\}$$
$$\hat{\mathbf{y}} = \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$
$$= \{\text{Introduction de } H\}$$
$$\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}$$

H est la *matrice chapeau* ou *matrice de projection*. Elle projette la cible y_i sur la prédiction $\hat{y}_i$.

Les h_{ii} sont appelés les effets levier. h_{ii} quantifie l'impact de y_i sur $\hat{y}_i$.

$$0 \leq h_{ii} \leq 1 \quad , \quad \sum_{i=1}^{n} h_{ii} = p$$

La valeur levier moyenne est p/n. Un effet levier supérieur à $2p/n$ peut indiquer une observation hors norme.

Chapitre 4

Application au jeu de données `abalone`

4.1 Récupération du jeu de données

Téléchargeons le jeu de données `abalone` depuis le répertoire de l'Université de Californie Irvine (University of California Irvine, UCI, machine learning repository).

Variable	Type	Unité	Commentaire
Sex	nominal	-	M, F, and I (infant)
Length	continuous	mm	Longest shell measurement
Diameter	continuous	mm	perpendicular to length
Height	continuous	mm	with meat in shell
Whole weight	continuous	grams	whole abalone
Shucked weight	continuous	grams	weight of meat
Viscera weight	continuous	grams	gut weight (after bleeding)
Shell weight	continuous	grams	after being dried
Rings	integer	-	+1.5 gives the age in years

```
abalone.cols = c("sex", "length", "diameter", "height", "whole.wt",
                 "shucked.wt", "viscera.wt", "shell.wt", "rings")

url <- 'http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data'
abalone <- read.table(url, sep=",", row.names=NULL, col.names=abalone.cols,
                      nrows=4177)
```

```
str(abalone)
```

```
## 'data.frame':    4177 obs. of  9 variables:
##  $ sex       : chr  "M" "M" "F" "M" ...
##  $ length    : num  0.455 0.35 0.53 0.44 0.33 0.425 0.53 0.545 0.475 0.55 ...
##  $ diameter  : num  0.365 0.265 0.42 0.365 0.255 0.3 0.415 0.425 0.37 0.44 ...
##  $ height    : num  0.095 0.09 0.135 0.125 0.08 0.095 0.15 0.125 0.125 0.15 ...
##  $ whole.wt  : num  0.514 0.226 0.677 0.516 0.205 ...
##  $ shucked.wt: num  0.2245 0.0995 0.2565 0.2155 0.0895 ...
```

```
##  $ viscera.wt: num  0.101 0.0485 0.1415 0.114 0.0395 ...
##  $ shell.wt  : num  0.15 0.07 0.21 0.155 0.055 0.12 0.33 0.26 0.165 0.32 ...
##  $ rings     : int  15 7 9 10 7 8 20 16 9 19 ...
```

4.2 Préparation du jeu de données

En cherchant des valeurs aberrantes, nous remarquons que, pour deux observations, la variable `height` a une valeur nulle.

```
table(abalone$height)
```

```
##
##     0  0.01 0.015  0.02 0.025  0.03 0.035  0.04 0.045  0.05 0.055  0.06 0.065
##     2     1     2     2     5     6     6    13    11    18    25    26    39
##  0.07 0.075  0.08 0.085  0.09 0.095   0.1 0.105  0.11 0.115  0.12 0.125  0.13
##    47    61    76    74   124    91   145   114   135   133   169   202   169
## 0.135  0.14 0.145  0.15 0.155  0.16 0.165  0.17 0.175  0.18 0.185  0.19 0.195
##   189   220   182   267   217   205   193   160   211   131   103   103    78
##   0.2 0.205  0.21 0.215  0.22 0.225  0.23 0.235  0.24  0.25 0.515  1.13
##    68    45    23    31    17    13    10     6     4     3     1     1
```

Nous regardons les valeurs des autres attributs pour ces deux observations.

```
abalone[abalone$height==0,]
```

```
##      sex length diameter height whole.wt shucked.wt viscera.wt shell.wt rings
## 1258   I  0.430     0.34      0    0.428     0.2065     0.0860   0.1150     8
## 3997   I  0.315     0.23      0    0.134     0.0575     0.0285   0.3505     6
```

Nous supprimons ces observations.

```
abalone <- subset(abalone, height!=0)
```

Nous affichons les corrélations entre variables.

```
print(as.matrix(cor(na.omit(abalone[,-1]))), digits=2)
```

```
##            length diameter height whole.wt shucked.wt viscera.wt shell.wt rings
## length       1.00     0.99   0.83     0.93       0.90       0.90     0.90  0.56
## diameter     0.99     1.00   0.83     0.93       0.89       0.90     0.91  0.57
## height       0.83     0.83   1.00     0.82       0.78       0.80     0.82  0.56
## whole.wt     0.93     0.93   0.82     1.00       0.97       0.97     0.96  0.54
## shucked.wt   0.90     0.89   0.78     0.97       1.00       0.93     0.88  0.42
## viscera.wt   0.90     0.90   0.80     0.97       0.93       1.00     0.91  0.50
## shell.wt     0.90     0.91   0.82     0.96       0.88       0.91     1.00  0.63
## rings        0.56     0.57   0.56     0.54       0.42       0.50     0.63  1.00
```

Les variables sont très corrélées. Mesurons les colinéarités par les facteurs d'inflation de la variance.

```
library(car)
```

```
## Warning: le package 'car' a été compilé avec la version R 4.1.1
## Le chargement a nécessité le package : carData
## Warning: le package 'carData' a été compilé avec la version R 4.1.1
```

```
vif(lm(rings ~ ., data = abalone))
```

```
##                   GVIF Df GVIF^(1/(2*Df))
## sex           1.542203  2        1.114385
## length       40.914963  1        6.396481
## diameter     42.377129  1        6.509772
## height        3.602519  1        1.898030
## whole.wt    111.397350  1       10.554494
## shucked.wt   28.811125  1        5.367600
## viscera.wt   17.488895  1        4.181973
## shell.wt     21.851561  1        4.674565
```

Parmi les variables corrélées, nous proposons de conserver celles avec les scores VIF les plus faibles (c'est-à-dire celles dont la variance s'explique le moins par la variance d'autres variables).

4.3 Modèle linéaire

Nous calculons un modèle linéaire.

```
abalone_lm <- lm(rings ~ length + height + viscera.wt + sex, data = abalone)
```

Observons les résultats statistiques proposés par défaut.

```
summary(abalone_lm)
```

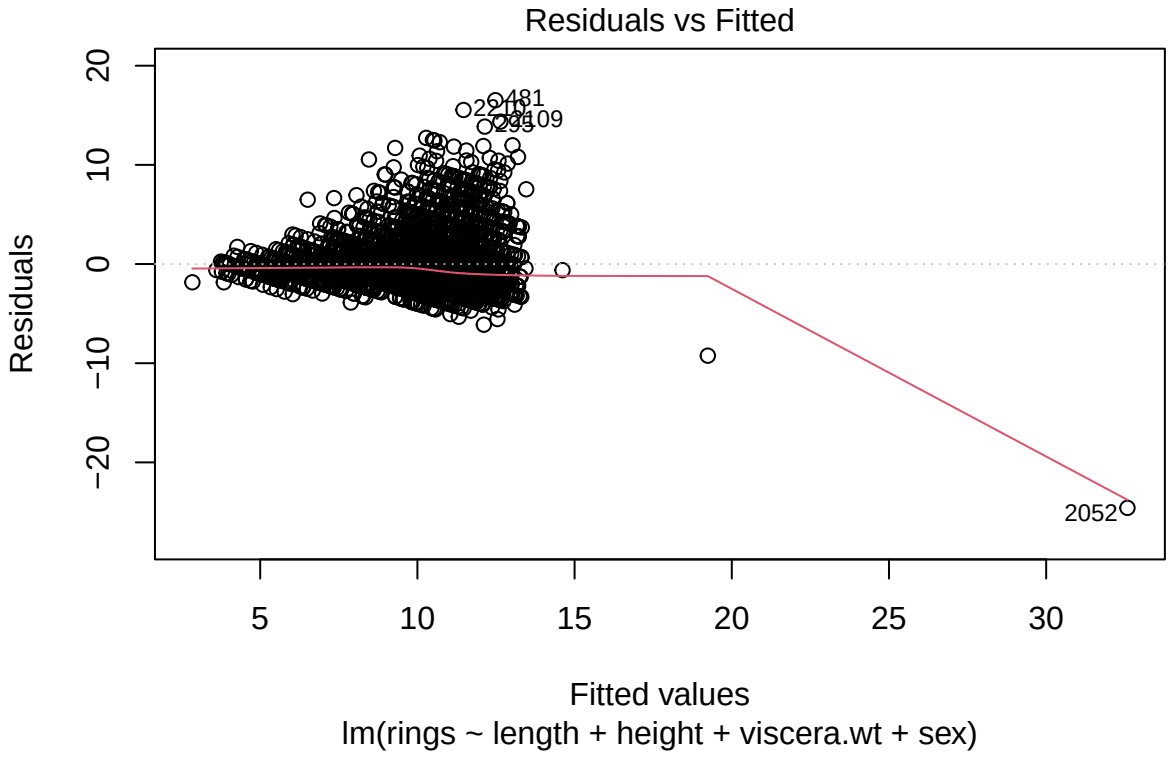
```
##
## Call:
## lm(formula = rings ~ length + height + viscera.wt + sex, data = abalone)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24.5855  -1.5960  -0.5637   0.8411  16.5186
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.21887    0.31096  10.351  < 2e-16 ***
## length       8.96888    0.85290  10.516  < 2e-16 ***
## height      22.76892    1.75676  12.961  < 2e-16 ***
## viscera.wt  -3.81953    0.87602  -4.360 1.33e-05 ***
## sexI        -1.27520    0.11903 -10.713  < 2e-16 ***
## sexM        -0.17192    0.09751  -1.763   0.0779 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 2.58 on 4169 degrees of freedom
## Multiple R-squared:  0.3606, Adjusted R-squared:  0.3598
## F-statistic: 470.2 on 5 and 4169 DF,  p-value: < 2.2e-16
```

4.4 Analyse graphique du modèle linéaire

Affichons les résidus pour chaque observation.

```
plot(abalone_lm, which = 1, id.n = 5)
```



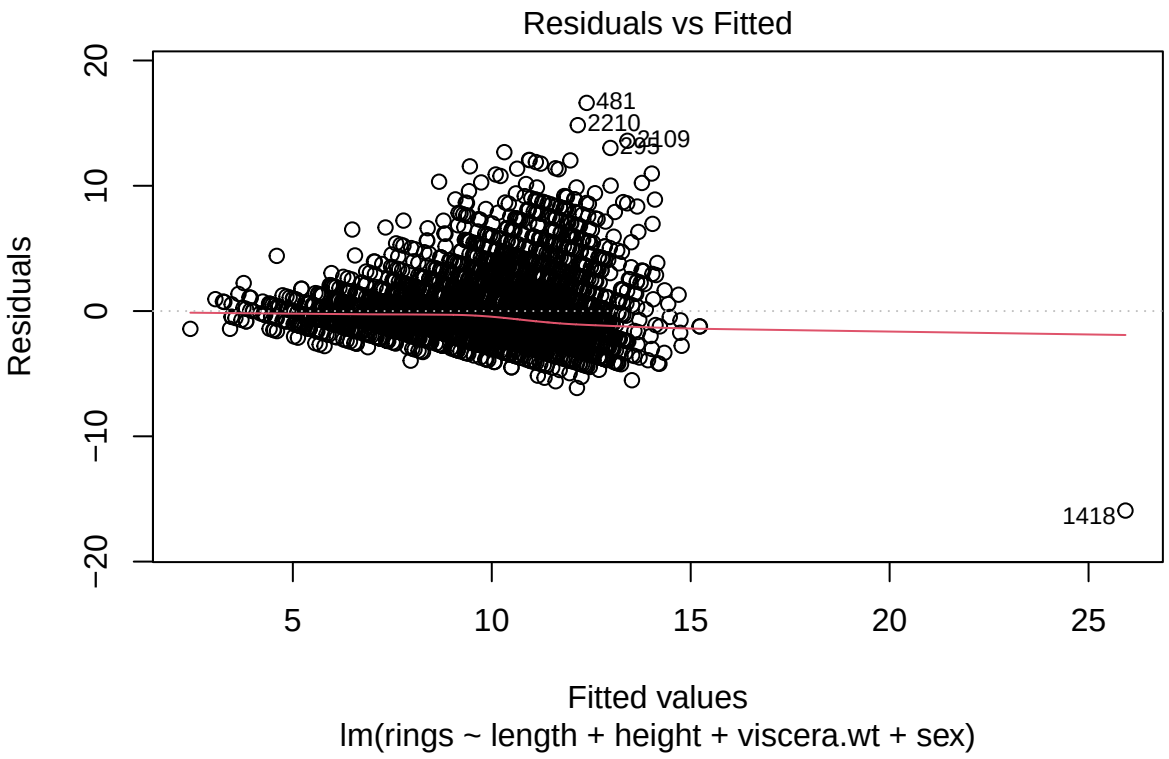
L'observation d'identifiant 2052 semble anormale. Nous observons que sa valeur `height` est anormalement élevée : 1.13.

```
abalone <- abalone[!(row.names(abalone) %in% c("2052")),]
abalone_lm <- lm(rings ~ length + height + viscera.wt + sex, data = abalone)
summary(abalone_lm)
```

```
##
## Call:
## lm(formula = rings ~ length + height + viscera.wt + sex, data = abalone)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.9277  -1.5686  -0.5256   0.8584  16.6153
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.81286    0.30602   9.192  < 2e-16 ***
```

```
## length       4.67928    0.89457   5.231 1.77e-07 ***
## height      44.50444    2.36542  18.815  < 2e-16 ***
## viscera.wt  -6.02489    0.87354  -6.897 6.11e-12 ***
## sexI        -1.18179    0.11677 -10.120  < 2e-16 ***
## sexM        -0.17276    0.09549  -1.809   0.0705 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.526 on 4168 degrees of freedom
## Multiple R-squared:  0.3869, Adjusted R-squared:  0.3862
## F-statistic:   526 on 5 and 4168 DF,  p-value: < 2.2e-16
```

```
plot(abalone_lm, which = 1, id.n = 5)
```



La variance des résidus n'est pas homogène. Plus le nombre d'anneaux est grand, plus la variance des résidus est grande. Ainsi, testons si un transformation logarithmique peut être significative.

```
abalone_lm <- lm(log(rings) ~ length + height + viscera.wt + sex, data = abalone)
summary(abalone_lm)
```

```
##
## Call:
## lm(formula = log(rings) ~ length + height + viscera.wt + sex,
##     data = abalone)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.41451 -0.14919 -0.03028  0.11883  0.85263
```

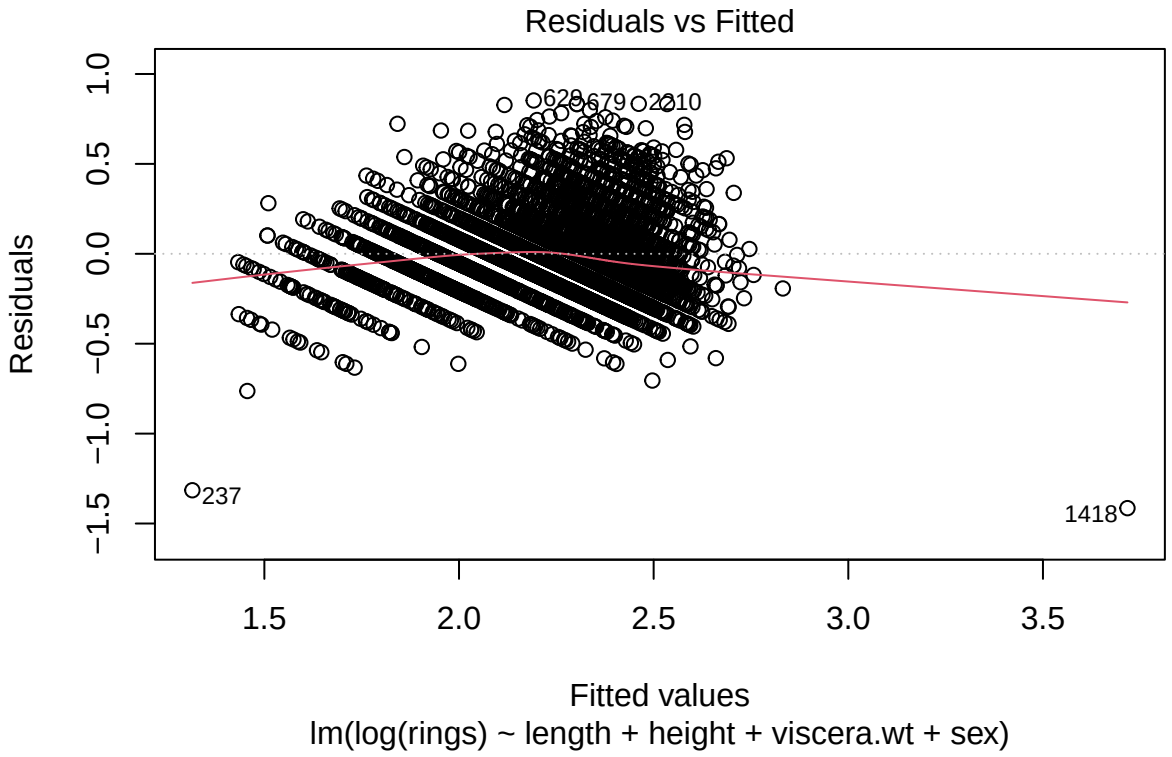
```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.315388   0.027301  48.181   <2e-16 ***
## length       1.108950   0.079806  13.896   <2e-16 ***
## height       4.094791   0.211022  19.405   <2e-16 ***
## viscera.wt  -0.978545   0.077930 -12.557   <2e-16 ***
## sexI        -0.123957   0.010418 -11.899   <2e-16 ***
## sexM        -0.012857   0.008519  -1.509    0.131
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2254 on 4168 degrees of freedom
## Multiple R-squared:  0.5031, Adjusted R-squared:  0.5025
## F-statistic: 844.1 on 5 and 4168 DF,  p-value: < 2.2e-16
```

Avec les intervalles de confiance.

```
confint(abalone_lm)
```

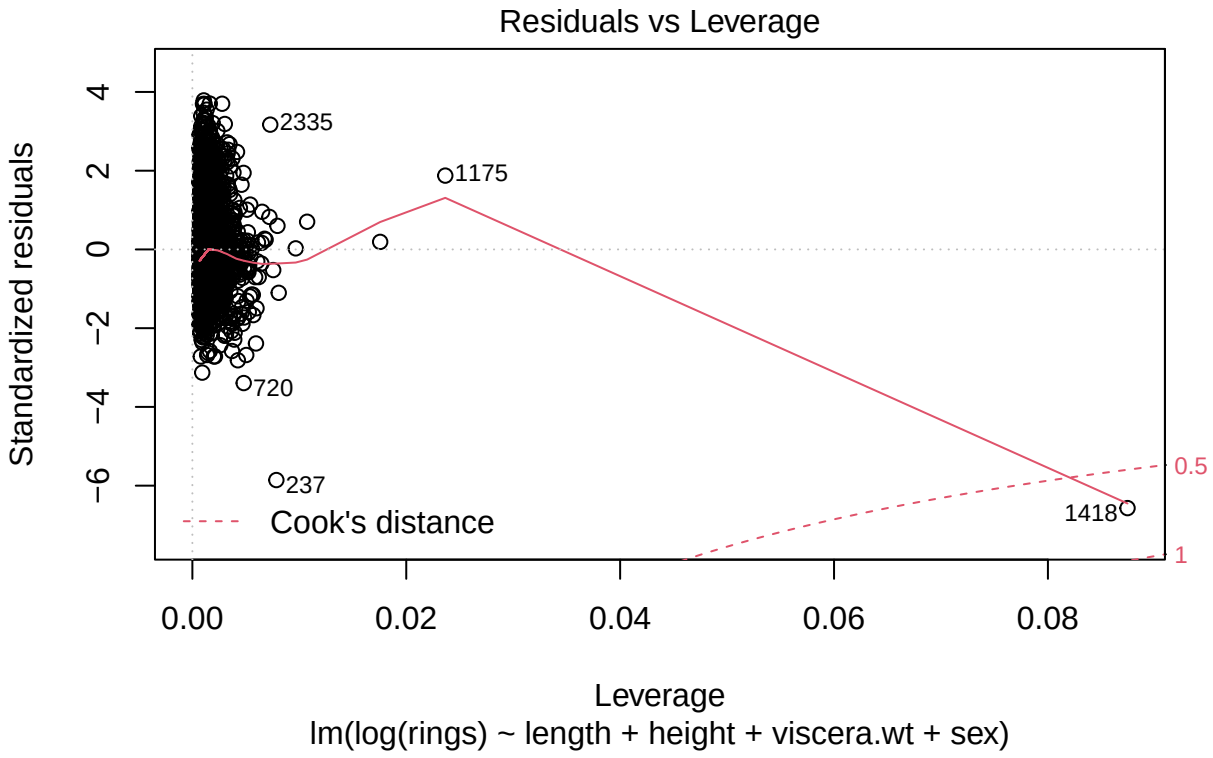
```
##                   2.5 %       97.5 %
## (Intercept)  1.26186386  1.368912268
## length       0.95248780  1.265412626
## height       3.68107529  4.508506578
## viscera.wt  -1.13132959 -0.825761408
## sexI        -0.14438146 -0.103533504
## sexM        -0.02955801  0.003843852
```

```
plot(abalone_lm, which = 1, id.n = 5)
```



Observons les résidus en fonction des effets levier.

```
plot(abalone_lm, which = 5, id.n = 5)
```



4.5 Interprétation du modèle linéaire

Une augmentation de `length` de $1mm$ augmente la prédiction de 1.1 anneaux avec un intervalle de confiance à 95% qui vaut $[0.95, 1.27]$. De même pour les autres variables continues.

Un passage de la variable `sex` de la catégorie de référence F à la catégorie I entraîne une diminution du nombre d'anneaux prédits de 0.12 avec un intervalle de confiance à 95% qui vaut $[-0.14, -0.10]$.

Chapitre 5

Régularisation de Tikhonov

5.1 Problèmes linéaires mal posés

Avec moins d'observations que de fonctions de base ($n < p$), le système $\mathbf{X}\boldsymbol{\beta} = \mathbf{y}$ ne possède pas de solution unique. Même quand $n \geq p$, le système linéaire peut posséder une solution approchée préférable à la solution optimale. C'est en particulier vrai quand plusieurs observations sont très proches à un coefficient multiplicatif près (on parle de colinéarités). Par exemple, soit le système linéaire suivant :

$$\begin{pmatrix} 1 & 1 \\ 1 & 1.00001 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0.99 \end{pmatrix}$$

Sa solution est $\boldsymbol{\beta}^T = (1001, -1000)$. Cependant, la solution approchée $\boldsymbol{\beta}^T = (0.5, 0.5)$ semble préférable. En effet, la solution optimale a peu de chance de bien s'adapter à de nouvelles observations (par exemple, l'observation $(1, 2)$ serait projetée sur l'étiquette -999).

5.2 Ajout de contraintes de régularité

Ainsi, lorsqu'il faut choisir entre plusieurs solutions, il peut être efficace d'exprimer une préférence envers celles dont les coefficients (ou paramètres) ont de faibles valeurs. Cela consiste par exemple à minimiser $|\beta_1| + |\beta_2| + \ldots$ (aussi noté $\|\boldsymbol{\beta}\|_1$, la "norme 1") ou encore $\beta_1^2 + \beta_2^2 + \ldots$ (aussi noté $\|\boldsymbol{\beta}\|_2^2$, le carré de la "norme 2"). Dans ce dernier cas, il s'agit de résoudre un nouveau problème de minimisation :

$$\min_{\boldsymbol{\beta}} \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_2^2$$
$$avec \quad 0 \leq \lambda$$

C'est encore un problème de minimisation quadratique en $\boldsymbol{\beta}$ dont le minimum se découvre par annulation de la dérivée.

$$\begin{aligned}
&\mathbf{0} = 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - 2\mathbf{X}^T\mathbf{y} + 2\lambda\boldsymbol{\beta} \\
&= \\
&\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_{n\times n}\right)\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}
\end{aligned}$$

En pratique, il s'agit donc d'ajouter une petite valeur positive λ aux éléments de la diagonale de la matrice de Gram. Cette approche porte plusieurs noms dont "régularisation de Tikhonov" ou "régression Ridge".

5.3 Standardisation

La régression Ridge contraint les coefficients de la régression linéaire en les contractant vers 0. L'importance de la contraction est proportionnelle au carré des valeurs des coefficients. Ainsi, il est important que les domaines des différentes variables partage une même échelle. Sinon, l'intensité de l'influence d'une variable sur la régularisation dépendrait de son échelle.

Avant d'inférer les valeurs des paramètres du modèle linéaire, nous standardisons les variables pour qu'elles soient toutes sur une échelle comparable. Chaque variable $\mathbf{x_j}$ est centrée par soustraction de sa moyenne $\bar{x_j}$ puis standardisée par division par son écart type σ_j. Notons $\mathbf{z_j}$ la version standardisée de la variable $\mathbf{x_j}$.

$$z_{ij} = \frac{x_{ij} - \bar{x_j}}{\sigma_j}$$

Nous pouvons également centrer ou standardiser la cible. Supposons que nous choisissions de seulement centrer la cible. Nous exprimons la relation entre les paramètres inférés sur les variables standardisées et leurs valeurs sur l'échelle des variables non transformées.

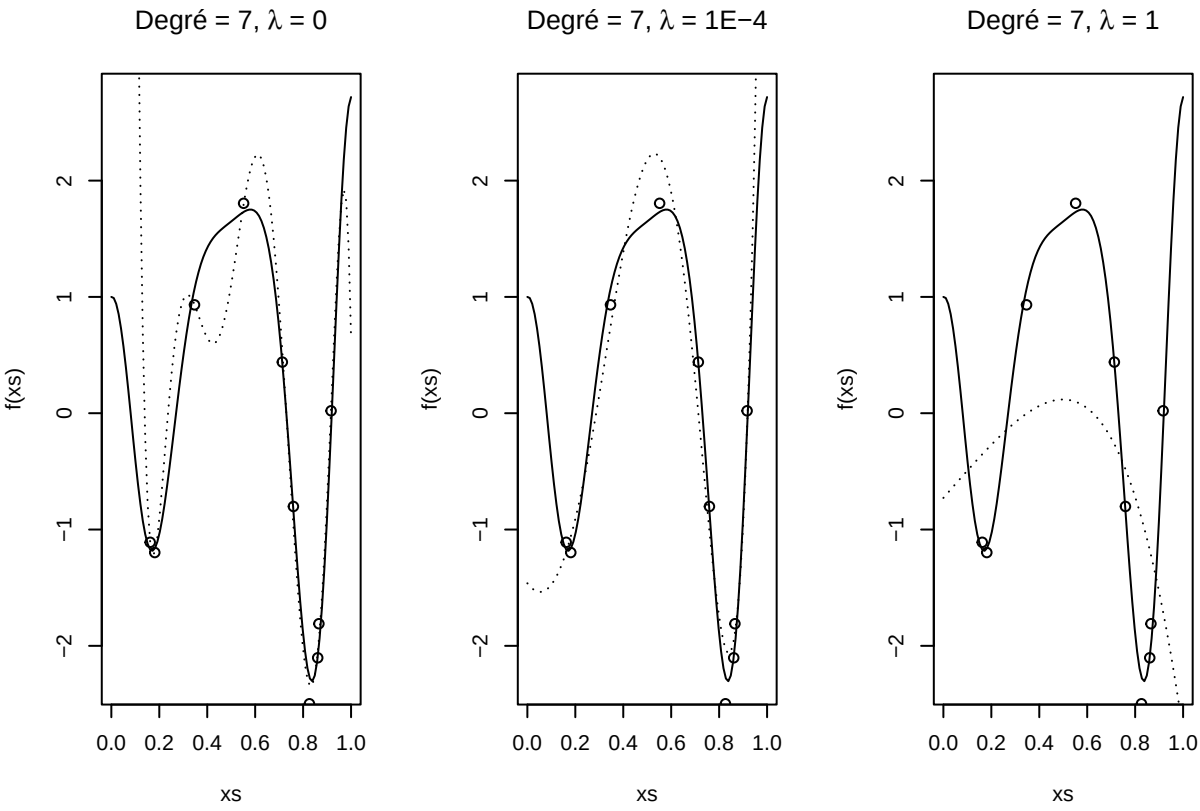
$$\begin{aligned}
&\hat{\mathbf{y}}_\mathbf{i} - \bar{\mathbf{y}} = \sum_{j=1}^{p} \hat{\beta}_j \left(\frac{x_{ij} - \bar{x_j}}{\sigma_j}\right) \\
&= \{\text{arithmétique}\} \\
&\hat{\mathbf{y}}_\mathbf{i} = \left(\bar{\mathbf{y}} - \sum_{j=1}^{p} \hat{\beta}_j \frac{\bar{x_j}}{\sigma_j}\right) + \sum_{j=1}^{p} \frac{\hat{\beta}_j}{\sigma_j} x_{ij}
\end{aligned}$$

Pour passer du modèle inféré sur les variables standardisées et la cible normalisée aux valeurs des paramètres sur les domaines originaux des variables, il faut donc ajouter une ordonnée à l'origine ("intercept") $\beta_0 = \bar{\mathbf{y}} - \sum_{j=1}^{p} \hat{\beta}_j \frac{\bar{x_j}}{\sigma_j}$ et remettre à l'échelle chaque paramètre β_j en le multipliant par le facteur $\frac{1}{\sigma_j}$.

5.4 Visulation de l'effet sur les paramètres de différents niveaux de régularisation

Sur notre exemple synthétique, nous affichons la fonction génératrice, le jeu de données et le polynôme de degré au plus sept découvert par régression ridge avec une valeur de λ égale soit à 0, soit à 10^{-4}, soit à 1.

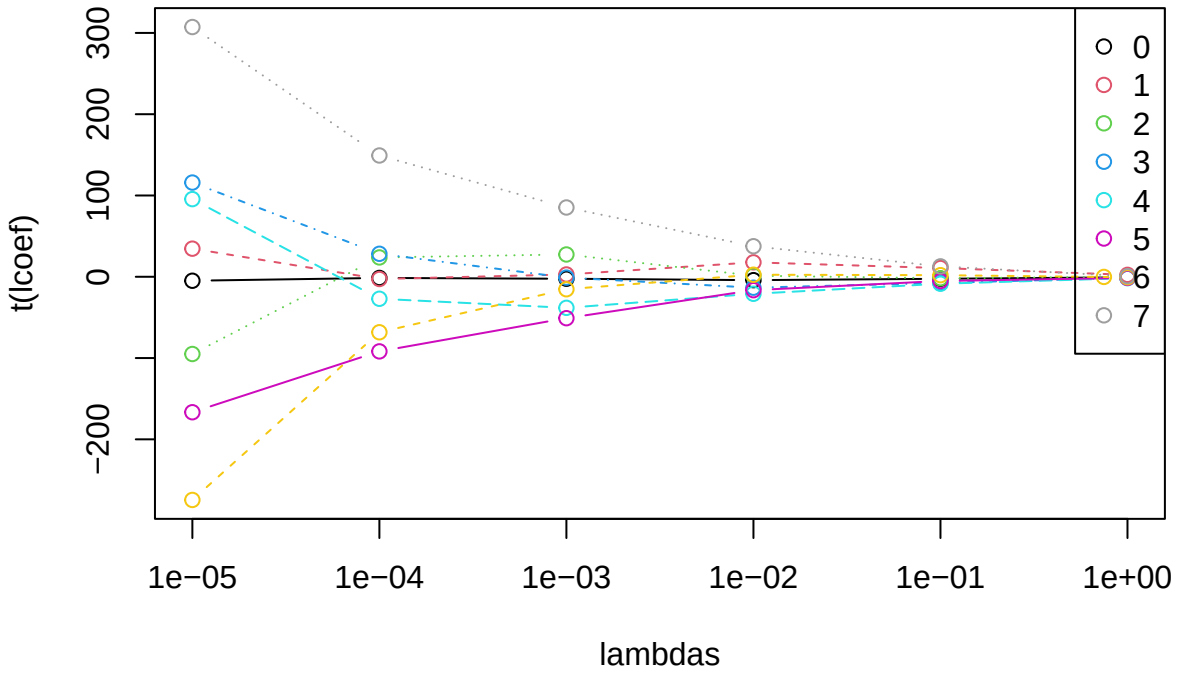
```
set.seed(1123)
# Image par f d'un échantillon uniforme sur l'intervalle [0,1], avec ajout d'un
# bruit gaussien de moyenne nulle et d'écart type 0.2
data = gendat(10,0.2)

par(mfrow=c(1,3))
coef <- ridge(0, data, 7)
plt(data,f,main=expression(paste(plain("Degré = "), 7, plain(", "), lambda,
                                 plain(" = 0"))))
pltpoly(coef)
coef <- ridge(1E-4, data, 7)
plt(data,f,main=expression(paste(plain("Degré = "), 7, plain(", "), lambda,
                                 plain(" = 1E-4"))))
pltpoly(coef)
coef <- ridge(1, data, 7)
plt(data,f,main=expression(paste(plain("Degré = "), 7, plain(", "), lambda,
                                 plain(" = 1"))))
pltpoly(coef)
```



Plus le coefficient de régularisation λ est faible, moins il contraint les paramètres (c'est-à-dire, les coefficients du polynôme) à conserver de petites valeurs et plus le polynôme découvert peut être complexe, au risque de provoquer du sur-apprentissage et donc de limiter la capacité du modèle à bien prédire les étiquettes de nouvelles observations. A l'inverse, plus le coefficient de régularisation est élevé, plus le modèle découvert sera simple, au risque de sous-apprendre en ne modélisant pas convenablement les variations propres au processus qui a généré les observations.

Pour mieux visualiser l'effet du coefficient de régularisation ridge, nous affichons les valeurs des coefficients du polynôme découvert pour différentes valeurs de λ. Plus λ augmente, plus les coefficients du polynôme diminuent et tendent vers 0.

```
lambdas <- c(1E-5, 1E-4, 1E-3, 1E-2, 1E-1, 1)
lcoef <- sapply(lambdas, ridge, data, 7)
matplot(lambdas, t(lcoef), type=c("b"), pch=1, col=1:8, log="x")
legend("topright", legend = 0:7, col=1:8, pch=1)
```



5.5 Régularisation et complexité

Essayons de mieux comprendre les raisons pour lesquelles la régularisation peut être efficace. Nous allons montrer qu'elle réduit la complexité d'un modèle prédictif. Dans ce contexte, qu'est-ce que la complexité ? C'est la sensibilité du modèle au bruit présent dans les données. Qu'est-ce que le bruit ? Il est possible de le définir après avoir fait l'hypothèse d'une famille de modèles qui auraient pu générer les données observées.

Prenons l'exemple d'un modèle linéaire : $F(\mathbf{X}) = \mathbf{W}^T\mathbf{X} + b$. Posons ensuite $\mathbf{X}^* = \mathbf{X} + \epsilon$ avec ϵ un faible bruit ($\|\epsilon\|$ est petit). $\mathbf{X}$ et $\mathbf{X}^*$ étant proches, une hypothèse de régularité demande que $F(\mathbf{X})$ et $F(\mathbf{X}^*)$ le soient aussi. La proximité de $F(\mathbf{X})$ et $F(\mathbf{X}^*)$ peut se mesurer avec $|F(\mathbf{X}) - F(\mathbf{X}^*)|$.

$$
\begin{aligned}
& |F(\mathbf{X}) - F(\mathbf{X}^*)| \\
= & \{F(\mathbf{X}) = \mathbf{W}^T\mathbf{X} + b\} \\
& |\mathbf{W}^T\mathbf{X} - \mathbf{W}^T\mathbf{X}^*| \\
= & \{\text{Algèbre linéaire}\} \\
& |\mathbf{W}^T(\mathbf{X} - \mathbf{X}^*)| \\
= & \{\mathbf{X}^* = \mathbf{X} + \epsilon\} \\
& |\mathbf{W}^T\epsilon| \\
\leq & \{\text{Inégalité de Cauchy-Schwarz}\} \\
& \|\mathbf{W}\|_2\|\epsilon\|_2
\end{aligned}
$$

Donc, en pénalisant les grandes valeurs de $\|\mathbf{W}\|_2$, on réduit la sensibilité du modèle à de petites perturbations dans les données observées. Autrement dit, par l'ajout de régularisation, les prédictions associées aux plus proches voisins de $\mathbf{X}$ tendent à être similaires à celle associée à $\mathbf{X}$.

5.6 Annexe code source

```
# 03 Tikhonov
# Résolution d'un système linéaire correspondant à la matrice de Gram pour
# un polynôme de degré fixé et avec l'ajout d'un facteur de régularisation en
# norme L2 dont l'importance est contrôlée par l'hyperparamètre alpha.
ridge <- function(alpha, data, degre) {
  A <- scale(outer(c(data$X), 1:degre, "^"))
  Y <- data$Y
  Ym <- mean(Y)
  Y <- Y - Ym
  gram <- t(A) %*% A
  diag(gram) <- diag(gram) + alpha
  coef <- solve(gram, as.vector(t(A) %*% Y))
  coef <- coef / attr(A,"scaled:scale")
  inter <- Ym - coef %*% attr(A,"scaled:center")
  coef <- c(inter, coef)
}
```

Chapitre 6

Validation croisée

6.1 Principe de la validation croisée

Comment choisir la valeur du coefficient de régularisation λ pour une régression ridge ? Notons en passant que λ est un exemple de ce que l'on appelle un hyperparamètre car sa valeur doit être fixée avant de pouvoir apprendre les paramètres du modèle (dans notre cas, les coefficients d'un modèle linéaire).

Une possibilité est de diviser le jeu de données en deux parties, l'une utilisée pour apprendre le modèle prédictif, l'autre utilisée pour valider la qualité des prédictions sur des données qui n'ont pas été vues pendant la phase d'apprentissage. On parle de jeu d'entraînement et de jeu de test. Cette méthode est appelée "validation croisée".

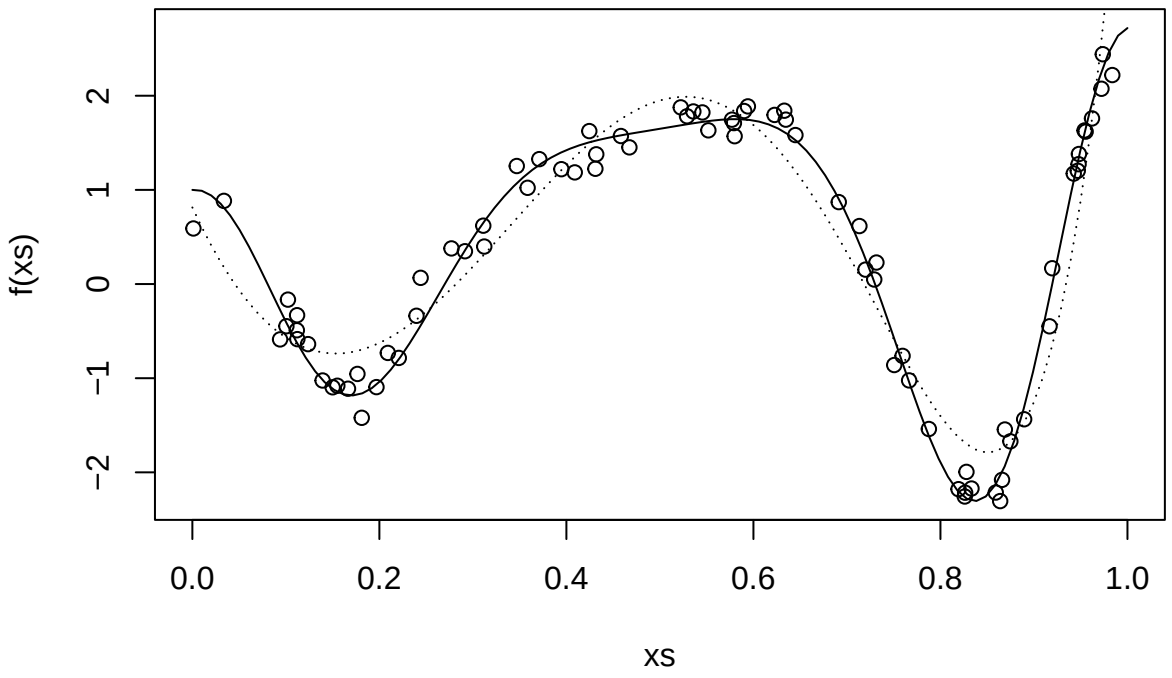
Il s'agirait donc de tester plusieurs valeurs de l'hyperparamètre λ sur le jeu d'entraînement et de conserver celle qui donne les meilleurs résultats sur un jeu de données de test qui n'a pas été utilisé pour l'entraînement. Cette approche pose problème. Le jeu de test est utilisé pour sélectionner le meilleur modèle, c'est-à-dire celui qui a le plus de chance de bien prédire pour de nouvelles données (i.e. de bien "généraliser"). Pour avoir une meilleure mesure de l'erreur, il est préférable de tester ce meilleur modèle sur des données qui n'ont jamais été utilisées pour comparer des modèles. Ainsi, nous pourrions réserver trois jeux de données : entraînement, validation (pour comparer différents modèles) et test (pour estimer l'erreur du modèle choisi).

Pour une approche souvent plus robuste, nous pouvons employer la stratégie dite de validation croisée à K plis ("K Fold Cross-Validation"). Il s'agit de diviser aléatoirement le jeu d'entraînement en K parties disjointes, appelées plis. Pour chaque valeur de l'hyperparamètre λ, nous apprenons K modèles. Notons par exemple $M[\lambda_i, j]$, le j-ème des K modèles appris pour la valeur λ_i de l'hyperparamètre λ. Le jeu d'entraînement du modèle $M[\lambda_i, j]$ est constitué de l'union de $K-1$ plis, les K plis initiaux auxquels on retire le j-ème pli qui joue le rôle de jeu de données de validation. Une estimation de l'erreur d'un modèle avec pour hyperparamètre λ_i est obtenue en faisant la moyenne des erreurs des modèles $M[\lambda_i, j]$ sur les plis de validation. Ainsi, nous découvrons un meilleur hyperparamètre λ_{best}. Nous entraînons à nouveau un modèle sur l'ensemble du jeu d'entraînement (i.e., l'union des K plis) avec un hyperparamètre λ de valeur λ_{best}. Finalement, nous testons ce dernier modèle sur le jeu de test pour estimer son erreur sur des données encore jamais utilisées.

6.2 Application de la validation croisée à la régularisation de Tikhonov

Le code source accompagnant ce chapitre comprend une fonction `splitdata` pour diviser le jeu de données en jeu d'entraînement et jeu de test. Ensuite, la fonction `kfoldridge` applique la stratégie de la validation croisée à K plis sur le jeu d'entraînement pour une liste de valeurs de l'hyperparamètre λ. Elle retourne les coefficients du meilleur modèle et les moyennes des valeurs absolues des erreurs commises sur les plis de validation.

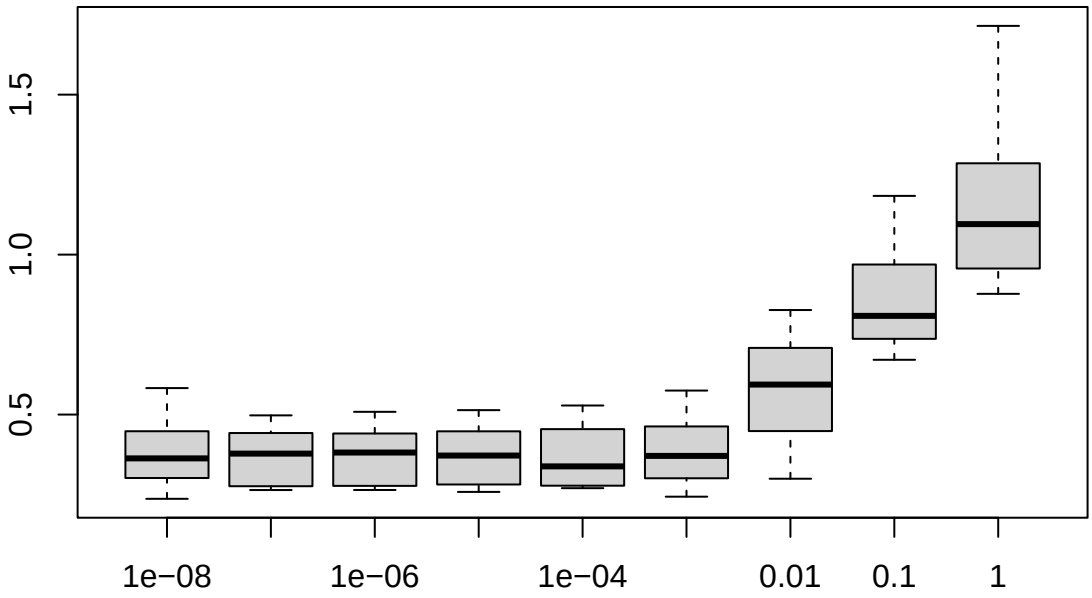
```
set.seed(1123)
N <- 100
deg1 <- 8
data = gendat(N,0.2)
splitres <- splitdata(data,0.8)
entr <- splitres$entr
test <- splitres$test
lambdas <- c(1E-8, 1E-7, 1E-6, 1E-5, 1E-4, 1E-3, 1E-2, 1E-1, 1)
reskfold <- kfoldridge(K = 10, lambdas = lambdas, data = entr, degre = deg1)
plt(entr,f)
pltpoly(reskfold$coef)
```



Avec le code ci-dessus, nous générons un nouveau jeu de données composé de 100 observations et nous calculons par validation croisée un polynôme de degré au plus égal à 8 qui modélise au mieux ces données. La valeur de λ retenue est : 10^{-4}.

Traçons un boxplot des erreurs commises sur les plis de validation pour chaque valeur de λ.

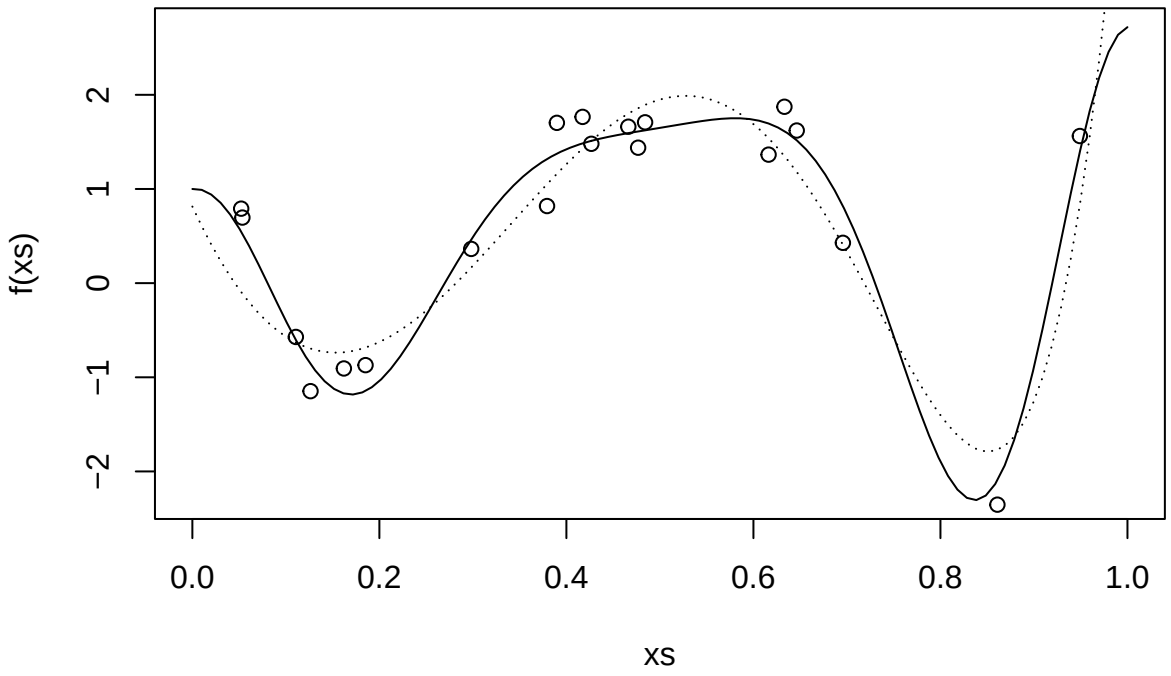
```
boxplot(reskfold$maes)
```




```
testpred <- polyeval(reskfold$coef, test$X)
testmae <- mean(abs(testpred - test$Y))
```

Ce meilleur modèle atteint une erreur absolue moyenne de 0.3557921 sur le jeu de test.

```
plt(test,f)
pltpoly(reskfold$coef)
```



6.3 Annexe code source

```
# 04 Validation croisée
# Séparer le jeu de données en un jeu d'entraînement et un jeu de test
# INPUT : jeu de données initial et proportion des données conservées pour
#         l'entraînement.
splitdata <- function(data,p) {
  n <- nrow(data$X)
  nentr <- round(p*n)
  entridx <- sample(1:n, nentr, replace=FALSE)
  list(entr  = list(X = data$X[entridx,,drop=FALSE],  Y = data$Y[entridx]),
       test = list(X = data$X[-entridx,,drop=FALSE], Y = data$Y[-entridx]))
}

# lambdas[l] est une liste de valeurs pour l'hyperparamètre lambda.
# Notons Ridge[l] un modèle avec lambda <- lambdas[l].
# Découper aléatoirement le jeu d'entraînement en K plis F[i] disjoints.
# Pour l <- [1,...,L]
#   Pour i <- [1,...,K]
#     Apprendre Ridge[l] sur l'union des plis F[j] avec j!=i
#     Calculer le score de Ridge[l] sur le pli de validation F[i]
#   Conserver les résultats du modèle sur les plis de validation.
#   Soit Moy[l] la moyenne des résultats de Ridge[l] sur les plis de validation.
# Soit l' l'indice du maximum de Moy[l]
# Apprendre Ridge[l'] sur l'ensemble du jeu de données d'entraînement.
# Retourner ce modèle.
kfoldridge <- function(K, lambdas, data, degre) {
  N <- nrow(data$X)
  folds <- rep_len(1:K, N)
  folds <- sample(folds, N)
  maes <- matrix(data = NA, nrow = K, ncol = length(lambdas))
  colnames(maes) <- lambdas
  lambda_idx <- 1
  for(lambda in lambdas) {
    for(k in 1:K) {
      fold <- folds == k
      coef <- ridge(lambda, data, degre, fold)
      pred <- polyeval(coef, data$X[fold,])
      maes[k,lambda_idx] <- mean(abs(pred - data$Y[fold]))
    }
    lambda_idx <- lambda_idx + 1
  }
  mmaes <- colMeans(maes)
  minmmaes <- min(mmaes)
  bestlambda <- lambdas[which(mmaes == minmmaes)]
  fold <- folds == K+1 # vector of FALSE
```

```
  coef <- ridge(bestlambda, data, degre, fold)
  list(coef = coef, maes = maes, lambda = bestlambda)
}

# Résolution d'un système linéaire correspondant à la matrice de Gram pour
# un polynôme de degré fixé et avec l'ajout d'un facteur de régularisation en
# norme L2 dont l'importance est contrôlée par l'hyperparamètre lambda.
# Les éléments du jeu de données indiqués par le vecteur booléen fold ne sont
# pas utilisés pour l'apprentissage du modèle. Cela permet d'implémenter une
# validation croisée à plusieurs plis.
ridge <- function(lambda, data, degre, fold) {
  xs <- c(data$X[!fold,])
  A <- outer(xs, 0:degre, "^")
  gram <- t(A) %*% A
  diag(gram) <- diag(gram) + lambda
  solve(gram, as.vector(t(A) %*% data$Y[!fold]))
}
```

Chapitre 7

Présentation de la décomposition en valeurs singulières (SVD)

7.1 Contexte

Nous présentons la décomposition en valeurs singulières, une méthode générique de compression d'un tableau rectangulaire de données. Elle possède de nombreuses applications. En particulier, elle permet de calculer en un seul entraînement les régressions ridge pour toutes les valeurs possibles de l'hyperparamètre α. Plus généralement, elle permet souvent d'adopter une perspective géométrique pour mieux comprendre les problèmes et algorithmes de l'apprentissage automatique à partir de données.

7.2 Recherche d'un sous-espace qui minimise les carrés des écarts à l'espace d'origine

Soit un tableau de données $\mathbf{X}$ composé de N observations en lignes, chacune décrite par P variables en colonnes. x_{ij} est la valeur de la variable j pour l'observation i. Les vecteurs lignes forment N points de l'espace $\mathbb{R}^P$. Les vecteurs colonnes forment P points de l'espace $\mathbb{R}^N$. Nous cherchons à projeter le nuage des points lignes sur un sous-espace $\mathcal{H} \subset \mathbb{R}^P$, tout en minimisant les déformations.

Pour commencer, nous considérons le meilleur sous-espace $\mathcal{H}$ à une dimension, c'est-à-dire une droite définie par son vecteur directeur unitaire $\mathbf{v}$ ("unitaire" signifie ici que sa norme euclidienne est égale à 1, soit $\sqrt{\mathbf{v}^T\mathbf{v}} = 1$, ou encore, $\mathbf{v}^T\mathbf{v} = 1$). Soit M_i un des N points de $\mathbb{R}^P$. A ce point correspond le vecteur $\mathbf{OM_i}$ aussi noté $\mathbf{x_i}$ car ses coordonnées se lisent sur la i-ème ligne de $\mathbf{X}$. Soit H_i la projection de M_i sur la droite $\mathcal{H}$.

La longueur OH_i est le produit scalaire de $\mathbf{x_i}$ et de $\mathbf{v}$.

$$OH_i = \mathbf{x_i}^T\mathbf{v} = \sum_{j=1}^{P} x_{ij}v_j$$

Nous obtenons un vecteur des N projections OH_i par le produit de la matrice $\mathbf{X}$ et du vecteur $\mathbf{v}$: $\mathbf{Xv}$. Choisissons pour $\mathcal{H}$ le sous-espace qui minimise la somme des carrés des écarts entre chaque point et sa projection : $\sum_i M_iH_i^2$.

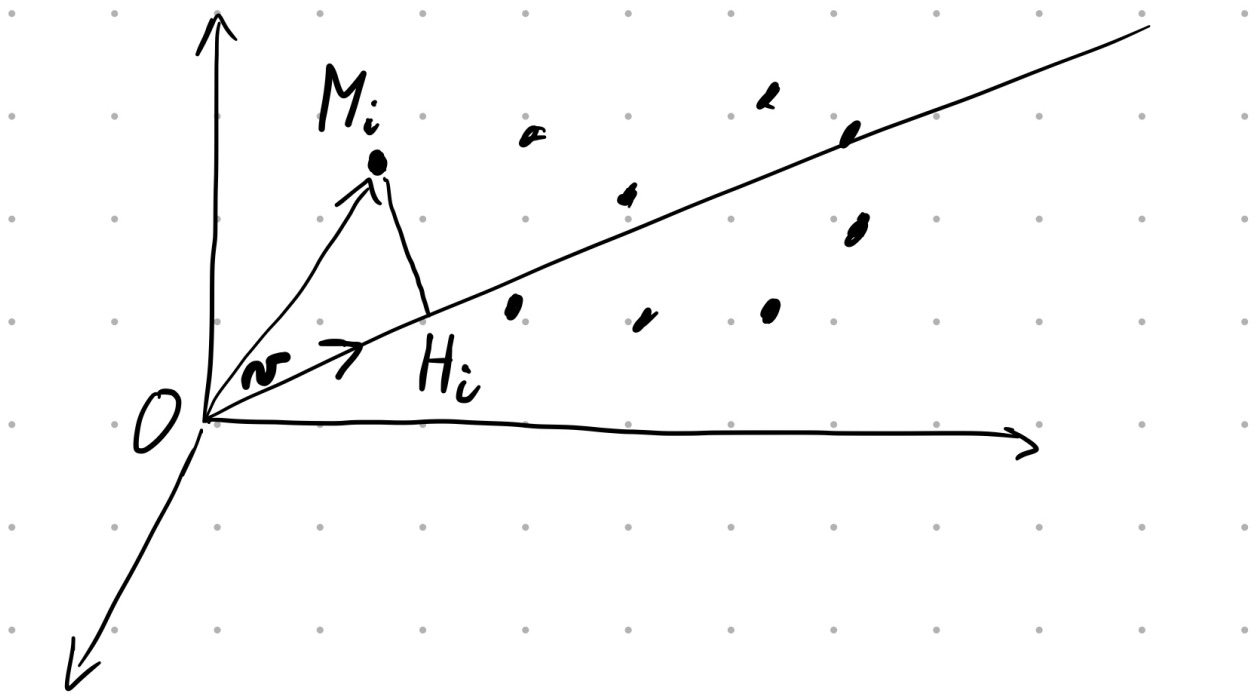


FIGURE 7.1 – svd1

Le théorème de Pythagore donne $\sum_i M_i H_i^2 = \sum_i OM_i^2 - \sum_i OH_i^2$.

Puisque $\sum_i OM_i^2$ est indépendant de $\mathbf{v}$, nous devons maximiser

$$\sum_i OH_i^2 = (\mathbf{Xv})^T(\mathbf{Xv}) = \mathbf{v}^T\mathbf{X}^T\mathbf{Xv} \tag{7.1}$$

... sous la contrainte $\mathbf{v}^T\mathbf{v} = 1$.

Nous notons $\mathbf{v_1}$ ce vecteur directeur du meilleur sous-espace de dimension 1 pour le nuage des N observations de $\mathcal{R}^P$. Le meilleur sous-espace de dimension 2 contient $\mathbf{v_1}$ et il faut le compléter avec un second vecteur unitaire $\mathbf{v_2}$, orthogonal à $\mathbf{v_1}$ et qui maximise $\mathbf{v_2}^T\mathbf{X}^T\mathbf{Xv_2}$. Etc.

Nous montrerons plus loin que $\mathbf{v_1}$ est le vecteur propre de $\mathbf{X}^T\mathbf{X}$ associé à la plus grande valeur propre λ_1, cette dernière étant justement le maximum de la forme quadratique de l'équation (7.1). De même, $\mathbf{v_2}$ est le vecteur propre de $\mathbf{X}^T\mathbf{X}$ associé à la seconde plus grande valeur propre λ_2. Etc.

Nous venons de voir que les vecteurs orthogonaux $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}$ forment une base d'un sous-espace k-dimensionnel sur lequel peuvent être projetés les N vecteurs lignes de $\mathbf{X}$ pour minimiser les carrés des écarts à l'espace d'origine qui était de dimension (au plus) $max(N, P)$.

Nous construirons de même des vecteurs orthogonaux $\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_k}$ qui forment une base d'un sous-espace k-dimensionnel sur lequel peuvent être projetés les P vecteurs colonnes de $\mathbf{X}$ pour minimiser les carrés des écarts à l'espace d'origine qui était de dimension (au plus) $max(N, P)$.

Nous trouverons de façon similaire que $\mathbf{u_1}$ est le vecteur propre de $\mathbf{XX}^T$ associé à la plus grande valeur propre μ_1. De même, $\mathbf{u_2}$ est le vecteur propre de $\mathbf{XX}^T$ associé à la seconde plus grande valeur propre μ_2. Etc.

Les vecteurs $\mathbf{u_1}, \mathbf{u_2}, \ldots$ et $\mathbf{v_1}, \mathbf{v_2}, \ldots$ sont les axes principaux de la matrice des données $\mathbf{X}$.

Nous pouvons écrire :

$$\begin{cases} \mathbf{X}^T\mathbf{X}\mathbf{v}_\alpha = \lambda_\alpha \mathbf{v}_\alpha \\ \mathbf{X}\mathbf{X}^T\mathbf{u}_\alpha = \mu_\alpha \mathbf{u}_\alpha \end{cases} \tag{7.2}$$

En prémultipliant la première des deux équations du système (7.2) par $\mathbf{X}$, nous obtenons :

$$\left(\mathbf{X}\mathbf{X}^T\right)\mathbf{X}\mathbf{v}_\alpha = \lambda_\alpha \left(\mathbf{X}\mathbf{v}_\alpha\right)$$

Cette dernière équation nous indique qu'au vecteur propre $\mathbf{v}_\alpha$ de $\mathbf{X}^T\mathbf{X}$ correspond un vecteur propre $(\mathbf{X}\mathbf{v}_\alpha)$ de $\mathbf{X}\mathbf{X}^T$ de même valeur propre λ_α. Comme μ_1 est la plus grande valeur propre de $\mathbf{X}\mathbf{X}^T$, nous avons montré que $\lambda_1 \leq \mu_1$

De même, en prémultipliant la seconde des deux équations du système (7.2) par $\mathbf{X}^T$, nous montrerions que $\mu_1 \leq \lambda_1$. Donc, $\lambda_1 = \mu_1$ et en général, $\lambda_\alpha = \mu_\alpha$.

Calculons la norme euclidienne, aussi appelée norme L2, de $\mathbf{X}\mathbf{v}_\alpha$.

$$\begin{aligned} & \text{Norme euclidienne de } \mathbf{X}\mathbf{v}_\alpha \\ = & \{ \text{ Par définition de la norme euclidienne.}\} \\ & \sqrt{\left(\mathbf{X}\mathbf{v}_\alpha\right)^T \left(\mathbf{X}\mathbf{v}_\alpha\right)} \\ = & \{\text{Propriété de la transposition}\} \\ & \sqrt{{\mathbf{v}_\alpha}^T \mathbf{X}^T \mathbf{X}\mathbf{v}_\alpha} \\ = & \{\text{Voir équation (7.2)}\} \\ & \sqrt{\lambda_\alpha {\mathbf{v}_\alpha}^T \mathbf{v}_\alpha} \\ = & \{\text{Par construction, } \mathbf{v}_\alpha\text{est de norme 1.}\} \\ & \sqrt{\lambda_\alpha} \end{aligned}$$

Nous avons montré plus haut que $(\mathbf{X}\mathbf{v}_\alpha)$ est un vecteur propre de $\mathbf{X}\mathbf{X}^T$ de valeur propre associée λ_α. Or, $\mathbf{u}_\alpha$ est le vecteur propre unitaire (i.e., de norme euclidienne égale à 1) de $\left(\mathbf{X}\mathbf{X}^T\right)$ associé à la valeur propre λ_α. C'est pourquoi, nous pouvons écrire :

$$\begin{cases} \mathbf{u}_\alpha = \frac{1}{\sqrt{\lambda_\alpha}} \mathbf{X}\mathbf{v}_\alpha \\ \mathbf{v}_\alpha = \frac{1}{\sqrt{\lambda_\alpha}} \mathbf{X}^T\mathbf{u}_\alpha \end{cases}$$

A partir de l'égalité $\mathbf{X}\mathbf{v}_\alpha = \mathbf{u}_\alpha\sqrt{\lambda_\alpha}$, nous post-multiplions par ${\mathbf{v}_\alpha}^T$ et nous sommons sur l'ensemble des P axes principaux (en supposant, sans perte de généralité, que $N > P$) :

$$\mathbf{X}\left(\sum_{\alpha=1}^{P}\mathbf{v}_\alpha{\mathbf{v}_\alpha}^T\right) = \sum_{\alpha=1}^{P}\sqrt{\lambda_\alpha}\mathbf{u}_\alpha{\mathbf{v}_\alpha}^T$$

Soit $\mathbf{V}$ la matrice formée des P vecteurs $\mathbf{v}_\alpha$ en colonnes. Comme les vecteurs $\mathbf{v}_\alpha$ sont orthogonaux deux à deux et de norme 1, $\mathbf{V}^T\mathbf{V}$ est la matrice identité $\mathbf{I_P}$, et $\sum_{\alpha=1}^{P}\mathbf{v}_\alpha{\mathbf{v}_\alpha}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I_P}$. Nous obtenons finalement :

$$\mathbf{X} = \sum_{\alpha=1}^{P}\sqrt{\lambda_\alpha}\mathbf{u}_\alpha{\mathbf{v}_\alpha}^T \tag{7.3}$$

Nous avons montré que la matrice $\mathbf{X}$ peut s'écrire comme une somme de matrices de rang 1 qui sont les produits des vecteurs $\mathbf{u}$ (appelés *vecteurs singuliers à gauche*) et $\mathbf{v}$ (appelés *vecteurs singuliers à droite*) pondérés par les *valeurs singulières* $\sqrt{\lambda_\alpha}$. L'équation (7.3) est celle de la *décomposition en valeurs singulières* de $\mathbf{X}$.

Nous pouvons écrire ce résultat sous forme matricielle en introduisant les matrices $\mathbf{U}$, $\mathbf{V}$ et $\mathbf{D}$. Sans perte de généralité, nous exprimons les résultats pour $N > P$. La matrice $\mathbf{U}$, de dimension $N \times P$, contient en colonnes les vecteurs singuliers à gauche $\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_P}$. La matrice $\mathbf{V}$, de dimension $P \times P$ contient en colonnes les vecteurs singuliers à droite $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_P}$. La matrice diagonale $\mathbf{D}$ contient sur sa diagonale les valeurs singulières $\sqrt{\lambda_1}, \sqrt{\lambda_2}, \ldots, \sqrt{\lambda_P}$. La décomposition en valeurs singulière de toute matrice $\mathbf{X}$ s'écrit alors :

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

Notons que le rang de la matrice $\mathbf{X}$, c'est-à-dire le nombre de colonnes indépendantes (ou le nombre de lignes indépendantes, c'est équivalent), est égal au nombre de valeurs singulières non nulles.

Par ailleurs, nous obtenons une reconstitution approchée de rang k de $\mathbf{X}$ en ne conservant dans l'équation (7.3) que les k plus grandes valeurs singulières $\sqrt{\lambda_1}, \sqrt{\lambda_2}, \ldots, \sqrt{\lambda_k}$ et en annulant toutes les autres.

7.3 Illustrations du SVD avec la compression d'une image

Considérons l'image d'un hortensia en fleurs.

Nous convertissons cette image, originellement au format JPEG, en un format en niveaux de gris, simple à manipuler, appelé PGM (*Portable Grey Map*). Dans ce format non compressé, l'image est représentée par une matrice dont chaque valeur, comprise entre 0 et 1, représente le niveau de gris d'un pixel. Nous opérons cette transformation grâce au programme *imagemagick* avec la commande `convert hortensia.jpg hortensia.pgm`.

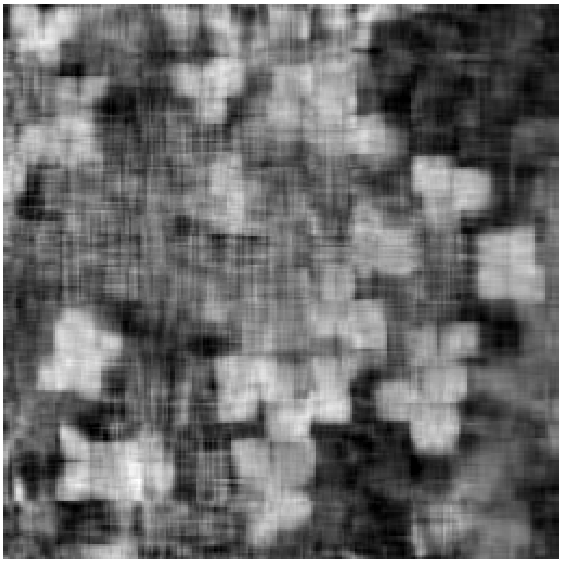
FIGURE 7.2 – hortensia

Ensuite, nous appliquons la décomposition en valeurs singulières à la matrice de l'image en niveaux de gris. Nous utilisons le résultat du SVD pour reconstituer une version compressée de l'image en ne conservant qu'un certain nombre des plus grandes valeurs singulières.

```
X <- read_grey_img("images/hortensia.pgm")

par(mfrow=c(1,2), oma=c(1,1,0,0)+0.1, mar=c(0,0,1,1)+0.1);
print_grey_img(X, main="image originale d=256", asp=1);
print_grey_img(compress_SVD(X,16), main="d=16", asp=1);
```

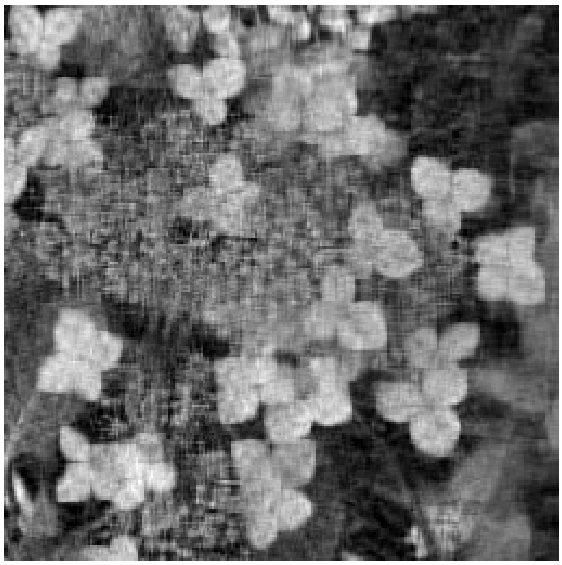
image originale d=256

d=16



```
print_grey_img(compress_SVD(X,32), main="d=32", asp=1);
print_grey_img(compress_SVD(X,64), main="d=64", asp=1);
```

d=32



d=64

```
print_grey_img(compress_SVD(X,128), main="d=128", asp=1);
print_grey_img(compress_SVD(X,256), main="d=256", asp=1);
```

d=128 d=256

7.4 Annexe code source

```
# chargement du package pixmap qui peut être installé avec la commande
# install.packages("pixmap");
library(pixmap) ;

read_grey_img <- function(file) {
  img = read.pnm("images/hortensia.pgm");
  # chaque entrée de img@grey est une intensité de gris comprise entre 0 et 1
  img@grey;
}

compress_SVD <- function(X, nd) {
  svdX <- svd(X);
  svdX$u[,1:nd] %*% diag(svdX$d[1:nd]) %*% t(svdX$v[,1:nd]);
}

print_grey_img <- function(Y,...) {
  Y[Y<0]<-0; # après reconstruction approchée à partir du résultat du svd le
  Y[Y>1]<-1; # domaine [0,1] de la matrice initiale peut ne plus être respecté
  image(t(apply(Y,2,rev)), col=grey(seq(0,1,length=256)), axes=FALSE, ...)
}
```

Chapitre 8

SVD et Analyse en Composantes Principales

```
set.seed(1123)
```

8.1 Projection sur les axes principaux

8.1.1 SVD

Soit une matrice de données $\mathbf{X} \in \mathbb{R}^{n \times p}$ dont la décomposition en valeurs singlières est :

$$\mathbf{X} = \sum_{\alpha} \sqrt{\lambda_\alpha} \mathbf{u}_\alpha {\mathbf{v}_\alpha}^T \quad \equiv \quad \mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

8.1.2 Facteurs

Nous avons montré que la projection orthogonale des lignes $\mathbf{x_i}$ de $\mathbf{X}$ sur $\mathbf{v_1}, \ldots, \mathbf{v_k}$ est la meilleure approximation k-dimensionnelle de $\mathbf{X}$ au sens de la minimisation des résidus au carré. Les axes de vecteurs directeurs $\mathbf{v}_\alpha$ sont appelés les *axes principaux*. Les coordonnées des observations sur les axes principaux sont appelées *facteurs*. Notons $\mathbf{F}$ la matrice dont les lignes sont les facteurs :

$$\mathbf{F} \triangleq \mathbf{X}\mathbf{V} = \mathbf{U}\mathbf{D}\mathbf{V}^T\mathbf{V} = \mathbf{U}\mathbf{D}$$

La covariance des facteurs est :

$$(\mathbf{U}\mathbf{D})^T (\mathbf{U}\mathbf{D}) = \mathbf{D}^2$$

Nous vérifions ainsi que, par construction, les facteurs sont orthogonaux et que λ_α est la somme des facteurs au carré sur l'axe $\mathbf{v}_\alpha$, autrement dit, la variance expliquéee par cet axe.

8.1.3 Contributions des observations

Nous pouvons mesurer la contribution $CTR_{i,\alpha}$ d'une observation $\mathbf{x_i}$ à la variance expliquée par $\mathbf{v}_\alpha$:

$$CTR_{i,\alpha} = \frac{f_{i,\alpha}^2}{\sum_i f_{i,\alpha}^2} = \frac{f_{i,\alpha}^2}{\lambda_\alpha}$$

Puisque $\sum_i CTR_{i,\alpha} = 1$, nous pouvons considérer, de façon heuristique, que les observations qui contribuent le plus à expliquer l'axe $\mathbf{v}_\alpha$ sont telles que $CTR_{i,\alpha} \geq 1/n$. Les observations dont les contributions sont les plus importantes et de signes opposés peuvent permettre d'interpréter un axe en fonction de l'opposition de ses pôles.

8.1.4 Contributions des axes

Nous pouvons similairement mesurer combien l'axe $\mathbf{v}_\alpha$ contribue à expliquer l'écart au centre de gravité d'une observation $\mathbf{x_i}$:

$$COS2_{i,\alpha} = \frac{f_{i,\alpha}^2}{\sum_\alpha f_{i,\alpha}^2} = \frac{f_{i,\alpha}^2}{d_{i,g}^2}$$

Avec $d_{i,g}^2$ le carré de la distance de l'observation $\mathbf{x_i}$ au centre de gravité g. $d_{i,g}^2 = \sum_j (x_{i,j} - g_j)^2$. Si les données sont centrées alors $d_{i,g}^2 = \sum_j x_{i,j}^2$. La somme des carrés des distances au centre de gravité pour toutes les observations est égale à la variance totale des données, ou inertie totale : $\sum_i d_{i,g}^2 = \mathcal{I} = \sum_\alpha \lambda_\alpha$.

8.1.5 Contribution des variables aux axes principaux

Les axes principaux $\mathbf{v}_\alpha$ sont des combinaisons linéaires des variables initiales. Lorsque la matrice initiale est centrée et réduite, les éléments de $(\lambda_\alpha/\sqrt{n-1})\mathbf{v}_\alpha$ représentent les corrélations entre les variables initiales et l'axe $\mathbf{v}_\alpha$. Ainsi, nous pouvons mesurer la contribution des variables initiales à l'expression de la variance expliquée par chaque axe principal (l'expression est au carré pour que la somme des contributions des variables à l'axe $\mathbf{v}_\alpha$ soit égale à 1) :

$$VARCTR_{j,\alpha} = \left(\frac{\lambda_\alpha}{\sqrt{n-1}}\mathbf{v}_\alpha\right)^2$$

8.2 Implémentation

Nous écrivons une fonction `fa` (*factor analysis*) qui :

- utilise l'algorithme `k-means` pour découvrir une centaine de clusters à partir des données standardisées,
- applique une décomposition en valeurs singulières sur les centres standardisés des clusters,
- calcule :
  - le pourcentage de variance expliquée par chaque axe principal (`prctPrcp`),
  - les facteurs (`fact`), c'est-à-dire les coordonnées des observations sur les axes principaux,
  - les contributions des observations aux axes principaux (`ctr`),
  - les contributions des axes principaux aux écarts au centre d'inertie des observations (`cos2`),
  - les contributions des variables aux axes principaux (`varctr`)

Nous écrivons une fonction `print.fa` qui affiche sur les axes principaux `d1` (par défaut 1) et `d2` (par défaut 2) les centres des clusters qui contribuent le plus à ces axes.

Nous écrivons une fonction `away.fa` qui retourne le cluster (son identifiant, sa taille et les noms des observations qui le composent) qui a le plus d'inertie (i.e., qui est le plus éloigné du centre d'inertie, c'est-à-dire l'origine du repère pour des données centrées) le long de l'axe principal `d` (par défaut 1).

```
# 05 Application du SVD à l'analyse factorielle

fa <-
function(X, nbClst=100, nstart=25)
{
  XStd <- scale(X)
  clst <- kmeans(XStd, nbClst, nstart)
  C <- scale(clst$centers)
  n <- nrow(C)
  Cs <- svd(C)
  fact <- Cs$u %*% diag(Cs$d)
  fact2 <- fact^2
  ctr <- sweep(fact2, 2, colSums(fact2), "/")
  cos2 <- sweep(fact2, 1, rowSums(fact2), "/")
  cos2 <- round(cos2*100,2)
  varctr <- ((Cs$v %*% diag(Cs$d))/sqrt(n-1))^2
  rownames(varctr) <- colnames(X)
  varctr <- round(varctr,2)
  prctPrcp <- round((Cs$d^2 / sum(Cs$d^2))*100, 2)

  r <- list(clst=clst, fact=fact, ctr=ctr, cos2=cos2, varctr=varctr,
            prctPrcp=prctPrcp)
  class(r) <- "fa"
  return(r)
}

print.fa <-
function(o,d1=NULL,d2=NULL)
{
  if(is.null(d1)) d1<-1
  if(is.null(d2)) d2<-2
  n <- dim(o$fact)[1]
  d1BestCtr <- which(o$ctr[,d1] > 1/n)
  d2BestCtr <- which(o$ctr[,d2] > 1/n)
  d1d2BestCtr <- union(d1BestCtr,d2BestCtr)
  plot(o$fact[d1d2BestCtr,d1], o$fact[d1d2BestCtr,d2], pch="",
       xlab=paste("D",d1), ylab=paste("D",d2))
  text(o$fact[d1d2BestCtr,d1], o$fact[d1d2BestCtr,d2], d1d2BestCtr, cex=0.8)
}

away <- function(x,...) UseMethod("away", x)
getCluster <- function(x,...) UseMethod("getCluster", x)

getCluster.fa <-
function(o,clstId)
{
```

```
  size <- o$clst$size[clstId]
  names <- names(o$clst$cluster[o$clst$cluster==clstId])

  r <- list(id=clstId, size=size, names=names)
  return(r)
}

away.fa <-
function(o,d=1)
{
  id <- which.max(abs(o$fact[,d]))
  return(getCluster(o,id))
}
```

8.3 Exemple

Considérons le jeu de données `abalone` introduit dans un précédent module. Nous retirons les observations pour lesquelles la variable `height` est nulle.

```
abalone.cols = c("sex", "length", "diameter", "height", "whole.wt",
                 "shucked.wt", "viscera.wt", "shell.wt", "rings")

url <- 'http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data'
abalone <- read.table(url, sep=",", row.names=NULL, col.names=abalone.cols,
                      nrows=4177)
abalone <- subset(abalone, height!=0)
```

Nous sélectionnons les variables explicatives numériques dans une matrice $\mathbf{X}$.

```
X <- abalone[,c("length", "diameter", "height", "whole.wt","shucked.wt",
                "viscera.wt", "shell.wt")]
```

Nous réalisons une première fois l'analyse en composantes principales.

```
fam <- fa(X) # fam pour 'factor analysis model'
```

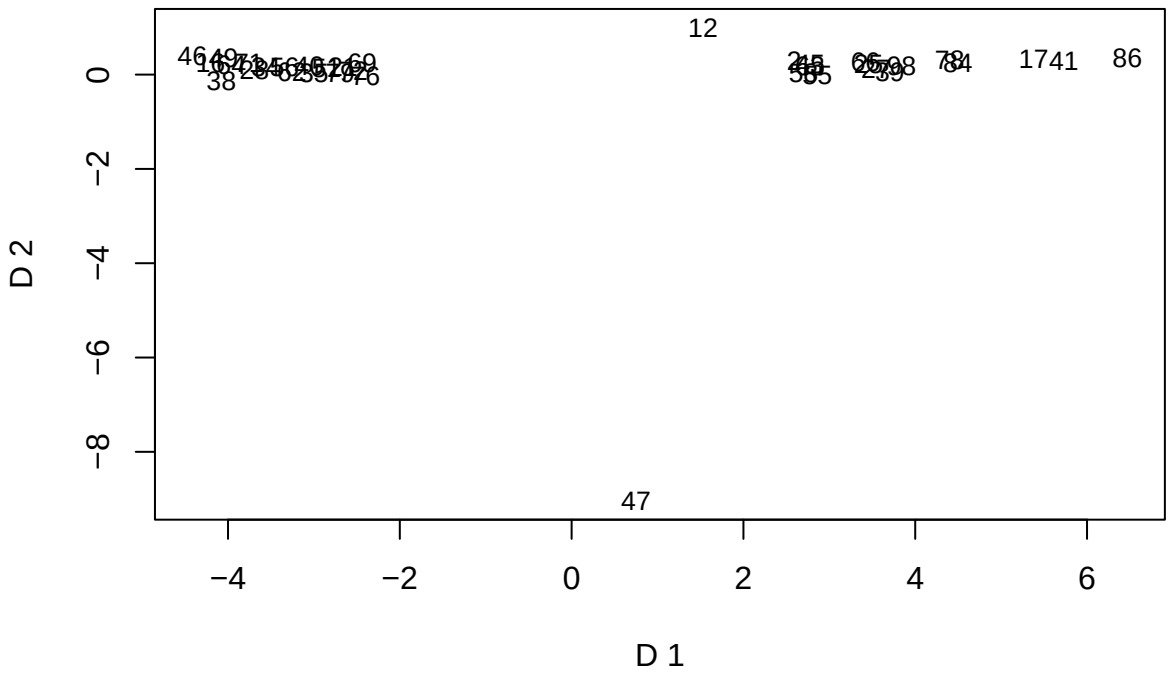
Nous affichons le pourcentage de variance expliquée par chaque axe principal.

```
fam$prctPrcp
```

```
## [1] 82.45 12.38  2.92  1.48  0.42  0.33  0.03
```

Nous affichons, sur les deux premiers axes principaux, les centres des clusters qui contribuent le plus à ces axes.

```
print(fam)
```




```
far <- away(fam,2)
```

Le cluster 47 (`far$id`) qui explique le plus la variance du deuxième axe principal semble anormalement important. Il contribue à expliquer 95.2 % (`round(fam$ctr[far$id,2]*100, 2)`) de la variance du second axe. Il est composé de seulement 1 (`far$size`) élément :

```
abalone[far$names,]
```

```
##      sex length diameter height whole.wt shucked.wt viscera.wt shell.wt rings
## 2052   F  0.455    0.355   1.13    0.594      0.332      0.116   0.1335     8
```

Nous retrouvons l'observation anormale déjà identifiée dans une précédente analyse. Nous recommençons l'analyse en le retirant :

```
abalone <- abalone[!(row.names(abalone) %in% far$names),]
X <- abalone[,c("length", "diameter", "height", "whole.wt","shucked.wt",
                "viscera.wt", "shell.wt")]
fam <- fa(X)
```

Nous affichons le pourcentage de variance expliquée par chaque axe principal.

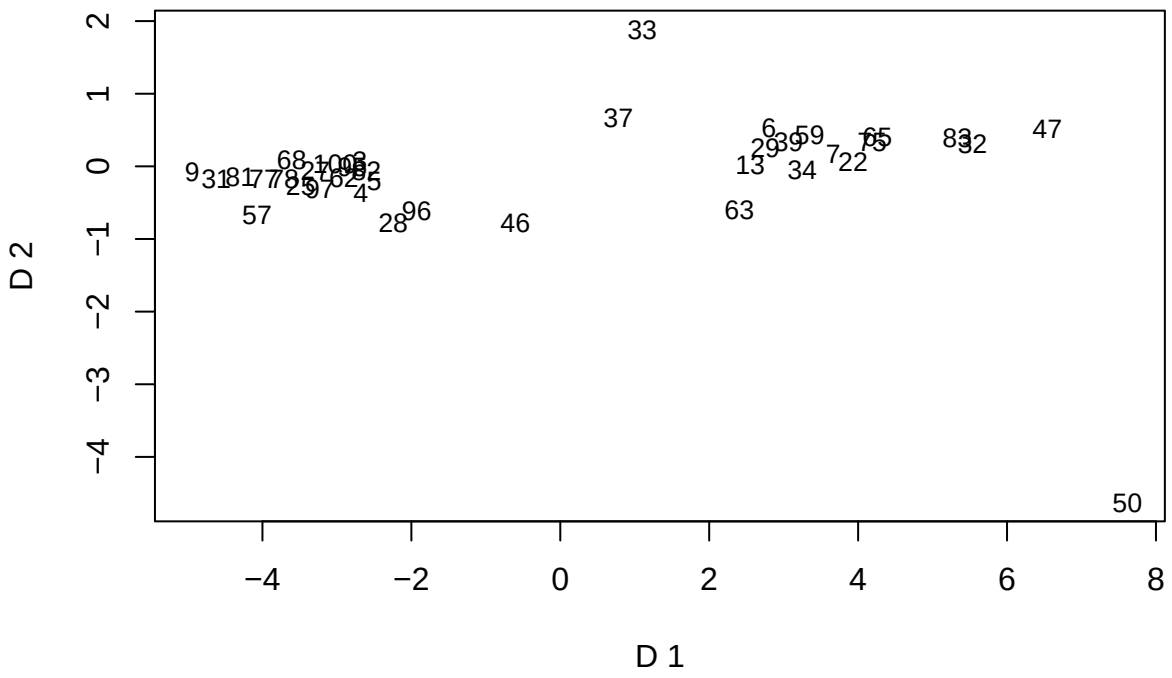
```
fam$prctPrcp
```

```
## [1] 89.64  4.85  2.95  1.70  0.44  0.38  0.05
```

Après cette correction, nous voyons qu'une part encore plus importante de la variance est expliquée par le premier axe principal. Cela peut nous indiquer que les variables explicatives sont très corrélées.

Nous calculons à nouveau les facteurs et les contributions des observations aux axes principaux. Nous affichons, sur les deux premiers axes principaux, les centres des clusters qui contribuent le plus à ces axes.

```
print(fam)
```



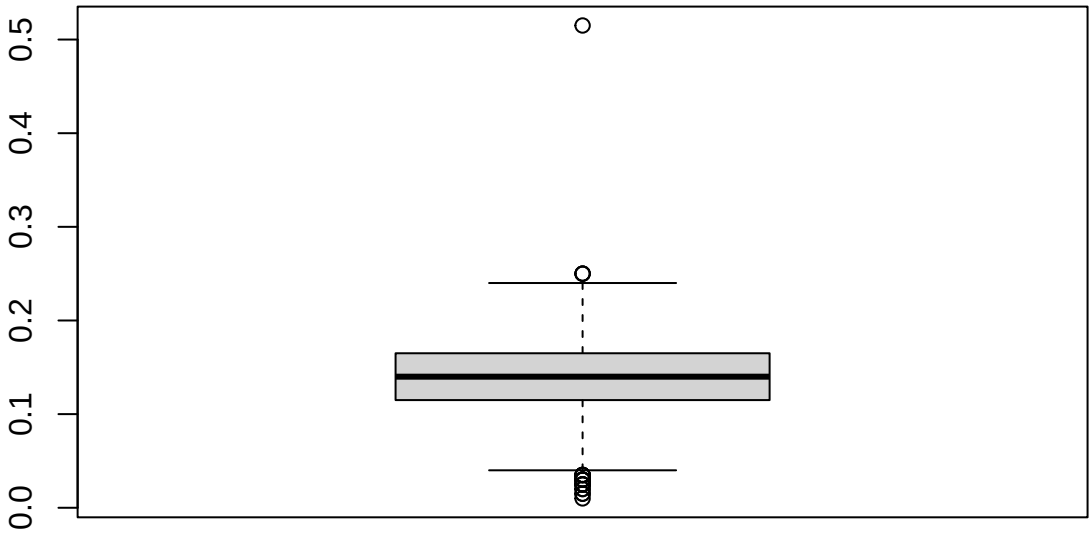

```
far <- away(fam,2)
```

Le cluster 50 est intrigant. Sa taille est : 1. C'est encore un singleton qui correspond à l'observation 1418 du jeu de données original. Nous découvrons qu'il s'agit sans doute d'une anomalie pour la variable `height` :

```
abalone[far$names,]
```

```
##      sex length diameter height whole.wt shucked.wt viscera.wt shell.wt rings
## 1418   M  0.705    0.565  0.515     2.21     1.1075     0.4865    0.512    10
```

```
boxplot(abalone$height)
```



Nous affichons les contributions des axes principaux à l'écart au centre d'inertie du cluster 50.

```
fam$cos2[far$id,]
```

```
## [1] 72.53 26.76  0.02  0.69  0.01  0.00  0.00
```

Nous vérifions que cette observation, sans doute anormale, est presque entièrement expliquée par les deux premiers axes principaux. Nous ne prenons donc pas de risque à considérer comme significatif son écart à l'origine dans le plan formé par ces deux axes.

Observons aussi les contributions des variables aux axes principaux :

```
fam$varctr
```

```
##            [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## length     0.90 0.03 0.05 0.01 0.00 0.01    0
## diameter   0.91 0.02 0.05 0.00 0.00 0.01    0
## height     0.71 0.28 0.01 0.00 0.00 0.00    0
## whole.wt   0.98 0.00 0.02 0.00 0.00 0.00    0
## shucked.wt 0.92 0.00 0.05 0.02 0.01 0.00    0
## viscera.wt 0.96 0.00 0.02 0.00 0.02 0.00    0
## shell.wt   0.91 0.00 0.00 0.09 0.00 0.00    0
```

Nous voyons à nouveau que toutes les variables sont très corrélées car elles contribuent toutes fortement à la variance expliquée par le premier axe principal.

Chapitre 9

Matrice définie non négative

Avant de présenter le cœur de la preuve de l'existence de la décomposition en valeurs singulières, nous rappelons quelques propriétés des matrices symétriques définies non négatives. Une matrice définie non négative (souvent noté PSD pour "positive semi-definite") $\mathbf{M}$ représente une métrique car elle définit un produit scalaire et donc une géométrie, c'est-à-dire qu'elle donne un sens aux notions de distance et d'angle.

Le produit scalaire défini par $\mathbf{M}$ entre deux vecteurs $\mathbf{x}$ et $\mathbf{y}$ pourra se noter $< \mathbf{x}, \mathbf{y} >_\mathbf{M} = \mathbf{x}^T\mathbf{M}\mathbf{y}$. Pour que ce produit scalaire soit valide, $\mathbf{M}$ doit respecter la contrainte $\mathbf{x}^T\mathbf{M}\mathbf{y} \geq 0$ pour tout $\mathbf{x}$ et $\mathbf{y}$.

Une fois donné un produit scalaire, les notions liées de norme et d'angle suivent~ : $\|\mathbf{x}\|_\mathbf{M} = \sqrt{< \mathbf{x}, \mathbf{x} >_\mathbf{M}}$ et $cos^2(\mathbf{x}, \mathbf{y}) =< \mathbf{x}, \mathbf{y} >_\mathbf{M} / \|\mathbf{x}\|_\mathbf{M}\|\mathbf{y}\|_\mathbf{M}$.

La géométrie euclidienne "classique" (associée à la base canonique) correspond à l'emploi de la matrice identité pour métrique, $\mathbf{M} = \mathbf{I}$.

Aussi, les valeurs propres d'une matrice PSD sont toutes positives~ :

$$
\begin{aligned}
& \mathbf{x}^T\mathbf{M}\mathbf{x} \geq 0 \\
\Rightarrow & \{\lambda \text{ est une valeur propre de } \mathbf{M}\} \\
& \mathbf{x}^T\lambda\mathbf{x} \geq 0 \\
= & \{\text{utilisation de la métrique identité}\} \\
& \lambda\|x\|^2_\mathbf{I} \geq 0 \\
= & \{\|x\|^2_\mathbf{I} \geq 0\} \\
& \lambda \geq 0
\end{aligned}
$$

Soit $\mathbf{V}$ la matrice des vecteurs propres de $\mathbf{M}$ et $\mathbf{L}$ la matrice diagonale de ses valeurs propres positives.

$$
\begin{aligned}
& \mathbf{MV} = \mathbf{VL} \\
= & \\
& \mathbf{M} = \mathbf{VLV}^{-1} \\
= & \{\mathbf{S} = \sqrt{\mathbf{L}}\} \\
& \mathbf{M} = \mathbf{VSSV}^{-1} \\
= & \{ \quad \text{Les vecteurs propres distincts sont orthogonaux deux à deux : } \mathbf{V}^{-1} = \mathbf{V}^T. \\
& \quad \mathbf{S} = \mathbf{S}^T. \text{On pose } \mathbf{T} = \mathbf{VS}.\} \\
& \mathbf{M} = \mathbf{TT}^T
\end{aligned}
$$

Ainsi, toute matrice définie non négative $\mathbf{M}$ se décompose en $\mathbf{M} = \mathbf{TT}^T$ où $\mathbf{T}$ est une transformation linéaire composée d'un changement d'échelle des axes du repère ($\mathbf{S}$) et d'une rotation ($\mathbf{V}$).

Nous remarquons par exemple que l'équation d'un cercle selon la métrique $\mathbf{M}$ correspond à celle d'une ellipse selon la métrique identité $\mathbf{I}$:

$$
\begin{aligned}
& \|\mathbf{x}\|^2_{\mathbf{M}} = c \\
= & \{\text{C'est l'équation d'un cercle, c est une constante.}\} \\
& < \mathbf{x}, \mathbf{x} >_{\mathbf{M}} = c \\
= & \{\text{Par définition du produit scalaire.}\} \\
& \mathbf{x}^T\mathbf{Mx} = c \\
= & \{\mathbf{M} = \mathbf{T}^T\mathbf{T}\} \\
& \mathbf{x}^T\mathbf{T}^T\mathbf{Tx} = c \\
= & \{\text{Par définition du produit scalaire, on retrouve le cercle déformé en une ellipse par } \mathbf{T}\} \\
& \|\mathbf{Tx}\|^2_{\mathbf{I}} = c
\end{aligned}
$$

Ainsi, une matrice symétrique définie non négative projette le cercle unité sur une ellipse dont les axes orthonormaux sont les vecteurs propres et les longueurs des axes sont les valeurs propres.

Enfin, l'angle formé entre $\mathbf{x}$ et $\mathbf{Mx}$ est inférieur ou égal à $\pi/2$ radians :

$$
\begin{aligned}
& \mathbf{x}^T\mathbf{Mx} \geq 0 \\
= & \{\text{Géométrie du produit scalaire. On note } \theta \text{ l'angle entre les deux vecteurs.}\} \\
& \|\mathbf{x}\|_{\mathbf{I}} \, \|\mathbf{Mx}\|_{\mathbf{I}} \, cos(\theta) \geq 0 \\
\Rightarrow & \\
& |\theta| \leq \pi/2
\end{aligned}
$$

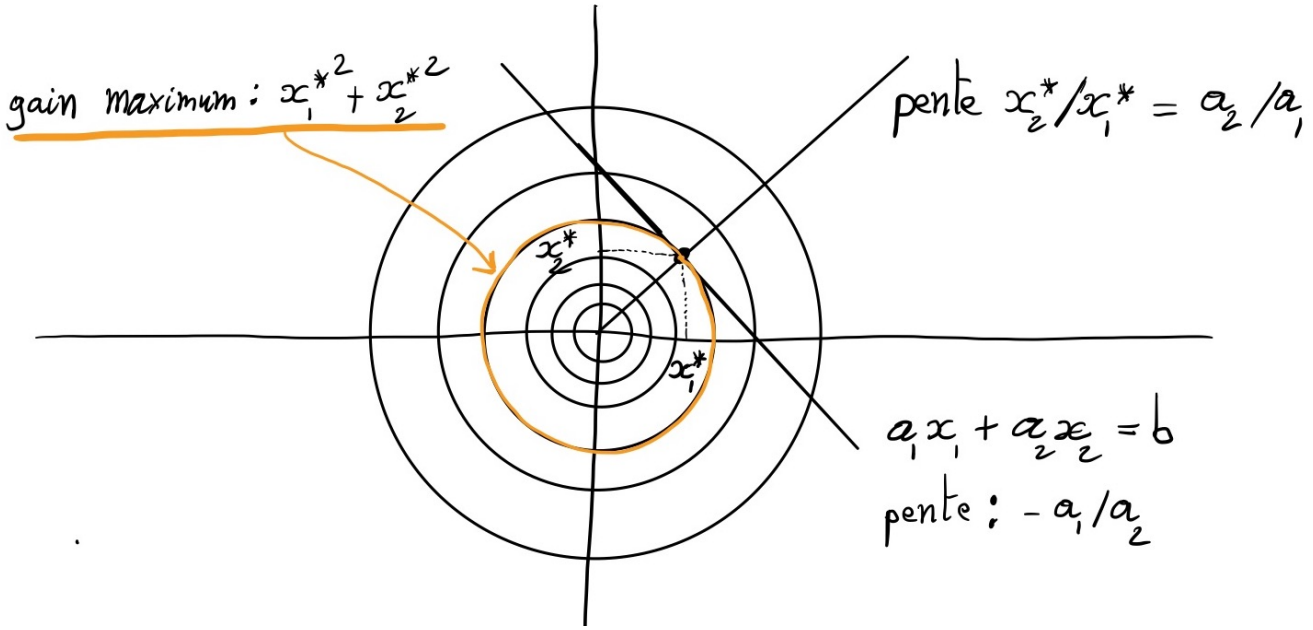
Ainsi, le vecteur résultat $\mathbf{Mx}$ reste du même côté que $\mathbf{x}$ par rapport à l'hyperplan perpendiculaire à $\mathbf{x}$ qui sépare l'espace en deux. Nous comprenons que le concept de matrice symétrique définie non négative est une généralisation pour un espace multidimensionnel du concept de nombre positif sur la droite réelle.

Chapitre 10

Optimisation sous contraintes et multiplicateurs de Lagrange

Avant de pouvoir présenter la preuve de l'existence de la décomposition en valeurs singulières, nous devons introduire une stratégie souvent très utile pour résoudre un problème d'optimisation sous contraintes : la méthode dite des multiplicateurs de Lagrange.

Prenons un exemple simple à deux dimensions. Nous cherchons à maximiser $F(\mathbf{x}) = x_1^2 + x_2^2$ sous la contrainte $K(\mathbf{x}) = b$, avec $K(\mathbf{x}) = a_1 x_1 + a_2 x_2$. C'est-à-dire que le point solution $\mathbf{x}^*$ doit être situé sur la ligne K.



Sur cet exemple, les lignes de niveaux, ou contours, de F sont des cercles centrés sur l'origine du repère cartésien. La solution $\mathbf{x}*$ appartient au contour de F tangent à la contrainte K. Il faut donc que leurs gradients soient alignés : $\nabla F = \lambda \nabla K$.

$$\nabla F = \begin{pmatrix} \partial F/\partial x_1 \\ \partial F/\partial x_2 \end{pmatrix} = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix} \quad ; \quad \nabla K = \begin{pmatrix} \partial K/\partial x_1 \\ \partial K/\partial x_2 \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

Nous avons donc un système de trois équations à trois inconnues (x_1, x_2 et λ) :

$$\begin{cases} 2x_1 = \lambda a_1 \\ 2x_2 = \lambda a_2 \\ a_1 x_1 + a_2 x_2 = b \end{cases}$$

En résolvant ce système par simples substitutions, nous trouvons la solution :

$$\begin{cases} x_1^* = \frac{a_1 b}{a_1^2 + a_2^2} \\ x_2^* = \frac{a_2 b}{a_1^2 + a_2^2} \\ \lambda^* = \frac{2b}{a_1^2 + a_2^2} \end{cases}$$

Ce même calcul s'écrit d'une façon plus systématique en rassemblant les équations $\nabla F = \lambda \nabla K$ et $K(\mathbf{x}) = b$ par l'introduction du Lagrangien :

$$\mathcal{L}(\mathbf{x}, \lambda) = F(\mathbf{x}) - \lambda(K(\mathbf{x}) - b)$$

En effet, en annulant les dérivées partielles de $\mathcal{L}$, nous retrouvons les équations $\nabla F = \lambda \nabla K$ et $K(\mathbf{x}) = b$:

$$\begin{array}{lcl} \partial\mathcal{L}/\partial x_1 = 0 & \Leftrightarrow & \partial F/\partial x_1 = \lambda \partial K/\partial x_1 \\ \partial\mathcal{L}/\partial x_2 = 0 & \Leftrightarrow & \partial F/\partial x_2 = \lambda \partial K/\partial x_2 \\ \partial\mathcal{L}/\partial \lambda = 0 & \Leftrightarrow & K(\mathbf{x}) = b \end{array}$$

Ainsi, le problème d'optimisation sous contrainte exprimé en fonction de F et K peut s'écrire comme un problème d'optimisation non contraint, en fonction de $\mathcal{L}$, dans un espace de plus grande dimension que celui d'origine puisque s'ajoute aux dimensions de $\mathbf{x}$ celle de λ.

Chapitre 11

Dérivation du SVD

Après ces préambules sur la notion de métrique associée à une matrice définie non négative et sur la méthode d'optimisation dite des multiplicateurs de Lagrange, nous reprenons la dérivation de la preuve de l'existence de la décomposition en valeurs singulières pour toute matrice.

Rappelons que nous avons noté $\mathbf{v_1}$ le vecteur directeur du meilleur sous-espace de dimension 1 pour le nuage des N observations de $\mathcal{R}^P$. Nous prouvons maintenant que $\mathbf{v_1}$ est le vecteur propre de $\mathbf{X}^T\mathbf{X}$ associé à la plus grande valeur propre λ_1.

Le résultat est sans difficulté plus général. Nous le prouvons pour toute matrice symétrique $\mathbf{A}$ (non seulement $\mathbf{X}^T\mathbf{X}$) et toute métrique de $\mathcal{R}^P$ représentée par une matrice symétrique définie non négative $\mathbf{M}$ (non seulement la matrice identité $\mathbf{I}$).

Nous rappelons que $\mathbf{v}^T\mathbf{A}\mathbf{v} = \Sigma a_{i,j}v_iv_j$. Ainsi, comme $\mathbf{A}$ et $\mathbf{M}$ sont symétriques (i.e., $a_{i,j} = a_{j,i}$ et $m_{i,j} = m_{j,i}$), les dérivées partielles des formes quadratiques ont les formes suivantes :

$$\frac{\partial \mathbf{v}^T\mathbf{A}\mathbf{v}}{\partial \mathbf{v}} = 2\mathbf{A}\mathbf{v} \quad \text{et} \quad \frac{\partial \mathbf{v}^T\mathbf{M}\mathbf{v}}{\partial \mathbf{v}} = 2\mathbf{M}\mathbf{v}$$

Pour trouver le maximum de $\mathbf{v}^T\mathbf{A}\mathbf{v}$ en intégrant la contrainte unitaire sur $\mathbf{v}$ (i.e., $\mathbf{v}^T\mathbf{M}\mathbf{v} = 1$), nous annulons les dérivées du langrangien $\mathcal{L}$:

$$\mathcal{L} = \mathbf{v}^T\mathbf{A}\mathbf{v} - \lambda(\mathbf{v}^T\mathbf{M}\mathbf{v} - 1)$$

$$\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial \mathbf{v}} = 0 \\
&= \{\text{voir calcul des dérivées ci-dessus}\} \\
&2\mathbf{A}\mathbf{v} - 2\lambda\mathbf{M}\mathbf{v} = 0 \\
&= \\
&\mathbf{A}\mathbf{v} = \lambda\mathbf{M}\mathbf{v} \\
&= \{\mathbf{v}^T\mathbf{M}\mathbf{v} = 1\} \\
&\lambda = \mathbf{v}^T\mathbf{A}\mathbf{v}
\end{aligned}$$

Nous découvrons que la valeur du multiplicateur de Lagrange λ est le maximum recherché. Par ailleurs, nous avons :

$$\begin{aligned}
&\mathbf{Av} = \lambda \mathbf{Mv} \\
=\ &\{\mathbf{M} \text{ est définie non négative et donc inversible}\} \\
&\mathbf{M}^{-1}\mathbf{Av} = \lambda \mathbf{v}
\end{aligned}$$

Donc $\mathbf{v}$ est le vecteur propre de $\mathbf{M}^{-1}\mathbf{A}$ associé à la plus grande valeur propre λ. Nous notons cette solution $\mathbf{v_1}$ et la valeur propre correspondante λ_1.

Nous cherchons ensuite $\mathbf{v_2}$, unitaire ($\mathbf{v_2}^T\mathbf{Mv_2} = 1$), orthogonal à $\mathbf{v_1}$ ($\mathbf{v_1}^T\mathbf{Mv_2} = 0$) et qui maximise $\mathbf{v_2}^T\mathbf{Av_2}$. Pour ce faire, nous annulons les dérivées du Langrangien ci-après.

$$\mathcal{L} = \mathbf{v_2}^T\mathbf{Av_2} - \lambda_2(\mathbf{v_2}^T\mathbf{Mv_2} - 1) - \mu_2\mathbf{v_2}^T\mathbf{Mv_1}$$

$$\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial \mathbf{v_2}} = 0 \\
=\ &\{\text{Par définition de } \mathcal{L} \text{ et car A et M sont symétriques.}\} \\
&2\mathbf{Av_2} - 2\lambda_2\mathbf{Mv_2} - \mu_2\mathbf{Mv_1} = 0 \\
\Rightarrow\ &\{\quad \text{Multiplier à gauche chaque membre par } \mathbf{v_1}^T \\
&\quad \mathbf{v_1}^T\mathbf{Av_2} = 0\ ;\ \mathbf{v_1}^T\mathbf{Mv_2} = 0\ ;\ \mathbf{v_1}^T\mathbf{Mv_1} = 1 \quad \} \\
&\mu_2 = 0
\end{aligned}$$

Nous avons utilisé ci-dessus la propriété $\mathbf{v_1}^T\mathbf{Av_2} = 0$ que nous prouvons ci-dessous.

$$\begin{aligned}
&\mathbf{v_1}^T\mathbf{Av_2} \\
=\ &\{\mathbf{A} \text{ est symétrique.}\} \\
&\mathbf{v_2}^T\mathbf{Av_1} \\
=\ &\{\mathbf{Av_1} = \lambda_1\mathbf{Mv_1} \text{ car } \mathbf{v_1} \text{ est vecteur propre de } \mathbf{M}^{-1}\mathbf{A}\} \\
&\lambda_1\mathbf{v_2}^T\mathbf{Mv_1} \\
=\ &\{\mathbf{v_1} \text{ et } \mathbf{v_2} \text{ sont M-orthogonaux.}\} \\
&0
\end{aligned}$$

Ainsi, nous trouvons :

$$\begin{aligned}
&\mathbf{Av_2} = \lambda_2\mathbf{Mv_2} \\
=\ &\{\mathbf{M} \text{ est définie non négative donc inversible.}\} \\
&\mathbf{M}^{-1}\mathbf{Av_2} = \lambda_2\mathbf{v_2}
\end{aligned}$$

Donc, $\mathbf{v_2}$ est le vecteur propre de $\mathbf{M}^{-1}\mathbf{A}$ qui correspond à la seconde plus grande valeur propre λ_2. Le raisonnement est similaire pour $\mathbf{v_3}$, etc.

Chapitre 12

Plus grande valeur propre et puissance itérée

Dans l'intention de comprendre plus tard comment calculer en pratique une décomposition en valeurs singulières, nous commençons par présenter la méthode dite de la puissance itérée pour calculer la plus grande valeur propre et son vecteur propre associé pour une matrice carrée.

12.1 Décomposition en valeurs propres d'une matrice carrée

Dire que $\mathbf{u}$ est un vecteur propre de la matrice $\mathbf{A}$ associé à la valeur propre λ signifie que $\mathbf{Au} = \lambda\mathbf{u}$. Remarquons que tout multiple de $\mathbf{u}$ est également un vecteur propre associé à la même valeur propre λ. Considérons l'exemple de la matrice ci-dessous.

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

Notons $\mathbf{e_1}^T = (1\ 0\ 0)$, $\mathbf{e_2}^T = (0\ 1\ 0)$ et $\mathbf{e_3}^T = (0\ 0\ 1)$. Nous avons :

$$\begin{aligned} \mathbf{Ae_3} &= -\mathbf{e_3} \\ \mathbf{A}(\mathbf{e_1} + \mathbf{e_2}) &= 3(\mathbf{e_1} + \mathbf{e_2}) \\ \mathbf{A}(\mathbf{e_1} - \mathbf{e_2}) &= -(\mathbf{e_1} - \mathbf{e_2}) \end{aligned}$$

Nous observons en particulier que la valeur propre -1 est associée à deux vecteurs propres linéairement indépendants : $\mathbf{e_3}$ et $(\mathbf{e_1} - \mathbf{e_2})$.

12.2 Plus grande valeur propre et méthode de la puissance itérée

Posons $\mathbf{A} \in \mathbb{R}^{N \times N}$ et $\mathbf{u^{(0)}} \in \mathbb{R}^N$ avec $\|\mathbf{u^{(0)}}\| = 1$. Supposons que $\mathbf{u^{(0)}}$ puisse se décomposer en une somme de vecteurs propres de $\mathbf{A}$.

$$\mathbf{u^{(0)}} = \mathbf{u_1} + \mathbf{u_2} + \cdots + \mathbf{u_r}$$
$$\text{Avec } \mathbf{A u_i} = \lambda_i \mathbf{u_i}$$

Considérons la séquence $\mathbf{u^{(0)}}, \mathbf{A u^{(0)}}, \mathbf{A^2 u^{(0)}}, \ldots$. Nous avons :

$$\mathbf{A}^m \mathbf{u^{(0)}} = \lambda_1^m \mathbf{u_1} + \lambda_2^m \mathbf{u_2} + \cdots + \lambda_r^m \mathbf{u_r} \tag{12.1}$$

Supposons par ailleurs que les valeurs propres soient ordonnées et que la première valeur propre soit strictement la plus grande en valeur absolue : $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_r|$. En divisant les deux membres de l'égalité (12.1) par λ_1^m, nous avons :

$$(\lambda_1^{-1}\mathbf{A})^m \mathbf{u^{(0)}} = \mathbf{u_1} + \sum_{i=2}^{r} \left(\frac{\lambda_i}{\lambda_1}\right)^m \mathbf{u_i}$$

Or, $\left|\frac{\lambda_i}{\lambda_1}\right| < 1$ pour $i = 2, \ldots, r$, donc nous pouvons conclure :

$$\lim_{m \to \infty} (\lambda_1^{-1}\mathbf{A})^m \mathbf{u^{(0)}} = \mathbf{u_1}$$

En résumé, si $\mathbf{u^{(0)}}$ peut s'écrire comme une somme de vecteurs propres de $\mathbf{A}$ et que la valeur propre λ_1 associée au vecteur propre $\mathbf{u_1}$ est plus grande en valeur absolue que les autres valeurs propres, alors, à un facteur multiplicatif près, $\mathbf{A}^m \mathbf{u^{(0)}}$ converge vers le vecteur propre $\mathbf{u_1}$.

Pour gérer ce facteur multiplicatif, nous considérons la séquence suivante :

$$\mathbf{w^{(k)}} = \mathbf{A u^{(k-1)}} \quad ; \quad \mathbf{u^{(k)}} = \frac{\mathbf{w^{(k)}}}{\|\mathbf{w^{(k)}}\|} \quad ; \quad \lambda^{(k)} = \mathbf{u^{(k)}}^T (\mathbf{A u^{(k)}})$$

Par induction, nous obtenons :

$$
\begin{aligned}
& \mathbf{u^{(k)}} \\
&= \{\gamma_k = \|\lambda_1^k \mathbf{u_1} + \sum_{j=2}^{r} \lambda_j^k \mathbf{u_j}\|\} \\
& \frac{1}{\gamma_k}\left(\lambda_1^k \mathbf{u_1} + \sum_{j=2}^{r} \lambda_j^k \mathbf{u_j}\right) \\
&= \\
& \frac{\lambda_1^k}{\gamma_k}\left(\mathbf{u_1} + \sum_{j=2}^{r} \left(\frac{\lambda_j}{\lambda_1}\right)^k \mathbf{u_j}\right)
\end{aligned}
$$

Par ailleurs, $\mathbf{u^{(k)}}$ étant à chaque itération normalisé, nous avons :

$$
\begin{aligned}
& \|\mathbf{u^{(k)}}\| = 1 \\
&= \\
& \frac{|\lambda_1^k|}{\gamma_k} = \frac{1}{\|\mathbf{u_1} + \sum_{j=2}^{r} \left(\frac{\lambda_j}{\lambda_1}\right)^k \mathbf{u_j}\|}
\end{aligned}
$$

D'où :

$$\lim_{k\to\infty} \frac{|\lambda_1^k|}{\gamma_k} = \frac{1}{\|\mathbf{u_1}\|}$$

Finalement :

$$\lim_{k\to\infty} \mathbf{u^{(k)}} = \frac{\mathbf{u_1}}{\|\mathbf{u_1}\|}$$

Et :

$$
\begin{aligned}
& \lim_{k\to\infty} \mathbf{u^{(k)}}^T(\mathbf{A}\mathbf{u^{(k)}}) \\
&= \{\mathbf{A}\mathbf{u_1} = \lambda_1 \mathbf{u_1}\} \\
& \lambda_1
\end{aligned}
$$

Chapitre 13

Méthode de la puissance itérée et SVD

Nous montrons comment adapter la méthode de la puissance itérée au calcul de la plus grande valeur singulière et de ses deux vecteurs singuliers associés. Nous considérons une matrice de données rectangulaire $\mathbf{A} \in \mathbb{R}^{N \times M}$. Nous notons $\mathbf{U} \in \mathbb{R}^{N \times N}$ et $\mathbf{V} \in \mathbb{R}^{M \times M}$ les deux matrices orthogonales qui contiennent les vecteurs singuliers en colonnes. Nous notons enfin $\mathbf{\Sigma} = diag(\sigma_1, \sigma_2, \dots, \sigma_r, 0, \dots) \in \mathbb{R}^{N \times M}$ la matrice qui contient sur sa diagonale principale les valeurs singulières, avec r le range de $\mathbf{A}$. Nous avons donc :

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \qquad r \text{ est le rang de } \mathbf{A}.$$
$$\mathbf{U} = [\mathbf{u_1}, \mathbf{u_2}, \dots, \mathbf{u_N}] \qquad \text{avec } \mathbf{u_i} \in \mathbb{R}^N, \text{ base orthonormale de } \mathbb{R}^N$$
$$\mathbf{V} = [\mathbf{v_1}, \mathbf{v_2}, \dots, \mathbf{v_M}] \qquad \text{avec } \mathbf{v_j} \in \mathbb{R}^M, \text{ base orthonormale de } \mathbb{R}^M$$
$$\mathbf{A} = \sum_{k=1}^{r} \sigma_k \mathbf{u_k} {\mathbf{v_k}}^T$$

Nous allons montrer que la séquence ci-dessous converge vers σ_1, $\mathbf{u}$ et $\mathbf{v}$ tels que $\mathbf{A}\mathbf{v} = \sigma_1 \mathbf{u}$.

$$\mathbf{w^{(k)}} = \mathbf{A}\mathbf{v^{(k-1)}} \qquad \alpha_k = \|\mathbf{w^{(k)}}\|^{-1} \qquad \mathbf{u^{(k)}} = \alpha_k \mathbf{w^{(k)}}$$
$$\mathbf{z^{(k)}} = \mathbf{A}^T\mathbf{u^{(k)}} \qquad \beta_k = \|\mathbf{z^{(k)}}\|^{-1} \qquad \mathbf{v^{(k)}} = \beta_k \mathbf{z^{(k)}}$$
$$err = \|\mathbf{A}\mathbf{v^{(k)}} - \beta_k \mathbf{u^{(k)}}\| \qquad \sigma_1 = \beta_k$$

Nous décomposons $\mathbf{v^{(0)}} \in \mathbb{R}^M$ dans la base orthonormée formée par les vecteurs $\mathbf{v_j}$:

$$\mathbf{v^{(0)}} = \sum_{j=1}^{M} y_j \mathbf{v_j} \quad \text{avec } y_j = {\mathbf{v_j}}^T \mathbf{v^{(0)}}$$

Partant de ce $\mathbf{v^{(0)}}$, nous calculons les deux premières itérations :

$$\mathbf{w^{(1)}} = \mathbf{A}\mathbf{v^{(0)}} = \sum_{j=1}^{r} \sigma_j y_j \mathbf{u_j} \qquad \mathbf{u^{(1)}} = \alpha_1 \sum_{j=1}^{r} \sigma_j y_j \mathbf{u_j}$$

$$\mathbf{z^{(1)}} = \mathbf{A}^T\mathbf{u^{(1)}} = \alpha_1 \sum_{j=1}^{r} \sigma_j^2 y_j \mathbf{v_j} \qquad \mathbf{v^{(1)}} = \alpha_1\beta_1 \sum_{j=1}^{r} \sigma_j^2 y_j \mathbf{v_j}$$

$$\mathbf{w^{(2)}} = \mathbf{A}\mathbf{v^{(1)}} = \alpha_1\beta_1 \sum_{j=1}^{r} \sigma_j^3 y_j \mathbf{u_j} \qquad \mathbf{u^{(2)}} = \alpha_2\alpha_1\beta_1 \sum_{j=1}^{r} \sigma_j^3 y_j \mathbf{u_j}$$

$$\mathbf{z^{(2)}} = \mathbf{A}^T\mathbf{u^{(2)}} = \alpha_2\alpha_1\beta_1 \sum_{j=1}^{r} \sigma_j^4 y_j \mathbf{v_j} \qquad \mathbf{v^{(2)}} = \alpha_2\beta_2\alpha_1\beta_1 \sum_{j=1}^{r} \sigma_j^4 y_j \mathbf{v_j}$$

Par induction, nous avons :

$$\mathbf{v^{(k)}} = \delta_{2k} \sum_{j=1}^{r} \sigma_j^{2k} y_j \mathbf{v_j} \qquad \text{Avec } \delta_{2k} = \alpha_k\beta_k\alpha_{k-1}\beta_{k-1}\ldots\alpha_1\beta_1$$

$$\mathbf{u^{(k)}} = \delta_{2k-1} \sum_{j=1}^{r} \sigma_j^{2k-1} y_j \mathbf{u_j} \qquad \text{Avec } \delta_{2k-1} = \alpha_{k+1}\alpha_k\beta_k\alpha_{k-1}\beta_{k-1}\ldots\alpha_1\beta_1$$

Par ailleurs, nous avons :

$$\begin{aligned}
& 1 \\
&= \{\text{A chaque itération, la norme de } \mathbf{u^{(k)}} \text{ est maintenue égale à 1.}\} \\
& \|\mathbf{u^{(k)}}\|^2 \\
&= \{\text{Par définition de } \mathbf{u^{(k)}} \text{ et car } \|\mathbf{u_j}\| = 1\} \\
& \delta_{2k-1}^2 \sum_{j=1}^{r} \sigma_j^{4k-2} y_j^2
\end{aligned}$$

De même :

$$\begin{aligned}
& 1 \\
&= \\
& \|\mathbf{v^{(k)}}\|^2 \\
&= \\
& \delta_{2k}^2 \sum_{j=1}^{r} \sigma_j^{4k} y_j^2
\end{aligned}$$

Donc :

$$\frac{\|\mathbf{u^{(k+1)}}\|^2}{\|\mathbf{v^{(k)}}\|^2} = 1 = \sigma_1^2 \left(\frac{\delta_{2k+1}^2}{\delta_{2k}^2}\right) \left(\frac{\sum_{j=1}^{n_1} y_j^2 \ + \ \sum_{j=n_1}^{r} \left(\frac{\sigma_j}{\sigma_1}\right)^{4k+2} y_j^2}{\sum_{j=1}^{n_1} y_j^2 \ + \ \sum_{j=n_1}^{r} \left(\frac{\sigma_j}{\sigma_1}\right)^{4k} y_j^2}\right)$$

Ci-dessus, n_1 est la multiplicité de σ_1, c'est-à-dire le nombre de vecteurs singuliers indépendants associés à cette valeur singulière. En pratique, sur de "vraies" données, cette multiplicité sera presque toujours égale à 1.

Nous avons ainsi montré que :

$$\lim_{k\to\infty} \frac{\delta_{2k+1}}{\delta_{2k}} = \sigma_1$$

Et :

$$\begin{aligned} &\lim_{k\to\infty} \|\mathbf{Av^{(k)}} - \sigma_1 \mathbf{u^{(k)}}\| \\ &= \{\mathbf{Av^{(k)}} = \frac{\delta_{2k+1}}{\delta_{2k}} \mathbf{u^{(k+1)}}\} \\ &\quad 0 \end{aligned}$$

Chapitre 14

Projecteurs orthogonaux et SVD

Nous introduisons la notion de projecteur et nous montrons son lien avec la décomposition en valeurs signulières (SVD). Ainsi, nous comprendrons mieux la géométrie du SVD et nous serons plus tard en mesure de construire un algorithme qui est une extension de la méthode de la puissance itérée pour le calcul des k premières valeurs singulières et vecteurs singuliers.

14.1 Projecteur

Un projecteur est une matrice carrée idempotente : $\mathbf{P}^2 = \mathbf{P}$. Rappelons que l'espace des colonnes de $\mathbf{P}$ formé de l'ensemble des vecteurs qui s'expriment comme une combinaison linéaire des colonnes de $\mathbf{P}$, c'est-à-dire tous les vecteurs $\mathbf{v}$ qui s'écrivent sous la forme $\mathbf{Px}$. L'espace des colonnes de $\mathbf{P}$ est également appelé l'image de $\mathbf{P}$ et se note $Im(\mathbf{P})$. Un vecteur $\mathbf{v}$ qui appartient à l'image d'un projecteur $\mathbf{P}$ est tel que $\mathbf{Pv} = \mathbf{P}$:

$$
\begin{aligned}
& \mathbf{v} \text{appartient à l'image de } \mathbf{P} \\
= & \{\text{Par définition de l'image d'une transformation linéaire.}\} \\
& \mathbf{v} = \mathbf{Px} \\
= & \{\text{Multiplication des deux membres de l'égalité par } \mathbf{P}\} \\
& \mathbf{Pv} = \mathbf{P^2x} \\
= & \{\text{Idempotence d'un projecteur.}\} \\
& \mathbf{Pv} = \mathbf{Px} \\
= & \{\mathbf{v} = \mathbf{Px}\} \\
& \mathbf{Pv} = \mathbf{v}
\end{aligned}
$$

On appelle noyau d'une transformation linéaire, l'ensemble des vecteurs que cette dernière transforme en $\mathbf{0}$, c'est-à-dire les vecteurs qui montrent une dépendance linéaire entre des colonnes de la représentation matricielle de la transformation. Dans le cas d'un projecteur $\mathbf{P}$, la projection sur l'image de $\mathbf{P}$ d'un vecteur $\mathbf{v}$ se fait selon la direction $\mathbf{Pv} - \mathbf{v}$ qui appartient au noyau de $\mathbf{P}$ noté

$Ker(\mathbf{P})$ (voir la figure 14.1) :

$$\mathbf{P}(\mathbf{Pv} - \mathbf{v}) = \mathbf{P}^2\mathbf{v} - \mathbf{Pv} = \mathbf{Pv} - \mathbf{Pv} = \mathbf{0}$$

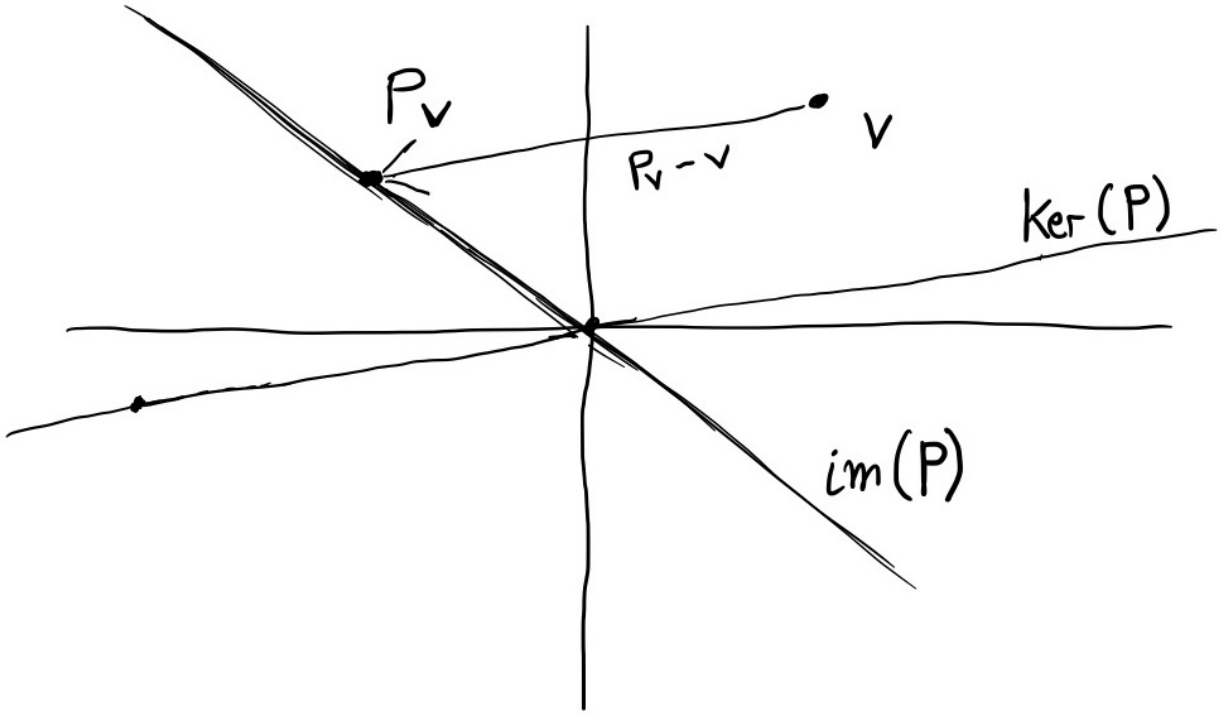


FIGURE 14.1 – Schéma d'un projecteur

Le projecteur complémentaire à $\mathbf{P}$ est $\mathbf{I} - \mathbf{P}$, il projette sur le noyau de $\mathbf{P}$. C'est-à-dire que $Im(\mathbf{I} - \mathbf{P}) = Ker(\mathbf{P})$. En effet :

$$\begin{aligned}
& \mathbf{v} \in Ker(\mathbf{P}) \\
= & \{\text{Par définition du noyau}\} \\
& \mathbf{Pv} = \mathbf{0} \\
\Rightarrow & \\
& (\mathbf{I} - \mathbf{P})\mathbf{v} = \mathbf{v} \\
= & \{\text{Par définition de l'image}\} \\
& \mathbf{v} \in Im(\mathbf{I} - \mathbf{P})
\end{aligned}$$

Donc, $Ker(\mathbf{P}) \subseteq Im(\mathbf{I} - \mathbf{P})$. Par ailleurs, un élément de l'image de $\mathbf{I} - \mathbf{P}$ peut se noter $(\mathbf{I} - \mathbf{P})\mathbf{v}$, ce qui est égal à $\mathbf{v} - \mathbf{Pv}$ et appartient donc au noyau de $\mathbf{P}$. Donc, nous avons aussi $Im(\mathbf{I} - \mathbf{P}) \subseteq Ker(\mathbf{P})$. Ainsi : $Ker(\mathbf{P}) = Im(\mathbf{I} - \mathbf{P})$ (de même : $Ker(\mathbf{I} - \mathbf{P}) = Im(\mathbf{P})$).

Donc, un projecteur $\mathbf{P}$ sépare l'espace initial en deux sous-espaces $S1 = Im(\mathbf{P})$ et $S2 = Ker(\mathbf{P})$. $\mathbf{P}$ projette **sur** $S1$ **parallèlement à** (ou **le long de**) $S2$.

14.2 Projecteur orthogonal

Un projecteur est dit orthogonal quand $S1$ et $S2$ sont orthogonaux. Montrons qu'une projection $\mathbf{P}$ est orthogonale si et seulement si $\mathbf{P} = \mathbf{P}^T$. Montrons d'abord que $\mathbf{P} = \mathbf{P}^T \Rightarrow$ orthogonalité.

$$
\begin{aligned}
& (\mathbf{Px})^T(\mathbf{I}-\mathbf{P})\mathbf{y} \quad \text{pour tout } \mathbf{x} \text{ et } \mathbf{y} \\
= & \{\text{Propriété de la transposition.}\} \\
& \mathbf{x}^T\mathbf{P}^T(\mathbf{I}-\mathbf{P})\mathbf{y} \\
= & \{\mathbf{P} = \mathbf{P}^T\} \\
& \mathbf{x}(\mathbf{P}-\mathbf{P}^2)\mathbf{y} \\
= & \{\mathbf{P} \text{ est un projecteur donc } \mathbf{P} = \mathbf{P}^2\} \\
& 0
\end{aligned}
$$

Montrons ensuite que orthogonalité $\Rightarrow \mathbf{P} = \mathbf{P}^T$. $\mathbf{P}$ projette sur $S1$ le long de $S2$ avec $S1 \perp S2$. Soit $\{\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_n}\}$ une base orthonormale de $S1$ et $\{\mathbf{q_{n+1}}, \mathbf{q_{n+2}}, \ldots, \mathbf{q_m}\}$ une base de $S2$. Soit Q la matrice orthogonale dont la jème colonne est $\mathbf{q_j}$. Nous avons : $\mathbf{PQ} = (\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_n}, 0, \ldots, 0)$. Donc, $\mathbf{Q^TPQ} = diag(1, 1, \ldots, 1, 0, \ldots, 0) = \mathbf{\Sigma}$. Ainsi, $\mathbf{P} = \mathbf{Q\Sigma Q^T} = \mathbf{P}^T$.

Etant donné un vecteur unitaire $\mathbf{q}$. Le projecteur orthogonal sur l'espace à une dimension défini par $\mathbf{q}$ est la matrice de rang 1 $\mathbf{P_q} = \mathbf{q}\mathbf{q}^T$. Le projecteur complémentaire est $\mathbf{P}_{\perp\mathbf{q}} = \mathbf{I} - \mathbf{P_q}$ de rang $m-1$ (si $\mathbf{q}$ est de dimension m).

Pour toute matrice $\mathbf{Q} \in \mathbb{R}^{M\times N}$ dont les colonnes $\mathbf{q_j}$ sont orthonormales, $\mathbf{P} = \mathbf{QQ^T} = \sum \mathbf{q_j}\mathbf{q_j}^T$ est un projecteur orthogonal sur $Im(\mathbf{Q})$.

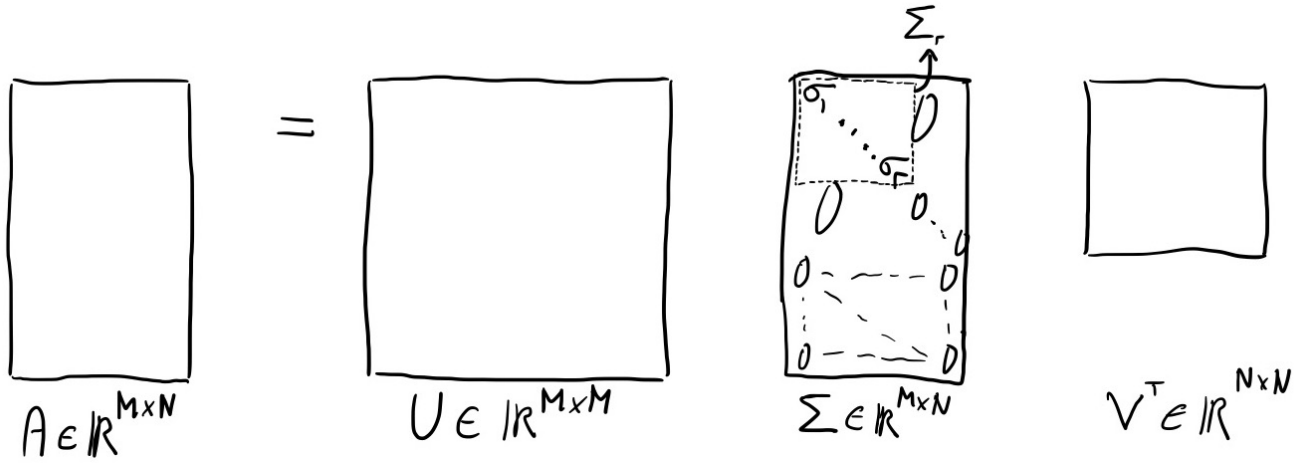
Rappelons la décomposition en valeurs singulière d'une matrice $\mathbf{A} \in \mathbb{R}^{M\times N}$ de rang r : $\mathbf{A} = \mathbf{U\Sigma V}^T$ avec $\mathbf{U} \in \mathbb{R}^{M\times M}$, $\mathbf{V} \in \mathbb{R}^{N\times N}$ des matrices aux colonnes orthonormales et $\mathbf{\Sigma} = diag(\sigma_1, \ldots, \sigma_r) \in \mathbb{R}^{M\times N}$. Nous pouvons partitionner le SVD en blocs pour obtenir la forme réduite présentée sur la figure 14.2.

Nous avons les propriétés fondamentales suivantes :

$$
\begin{aligned}
Im(\mathbf{A}) = Im(\mathbf{U_r}) \qquad & \mathbf{U_r}\mathbf{U_r}^T : \text{ est un projecteur orthogonal sur } Im(\mathbf{A}) \\
Ker(\mathbf{A}) = Im(\tilde{\mathbf{V}}_\mathbf{r}) \qquad & \tilde{\mathbf{V}}_\mathbf{r}\tilde{\mathbf{V}}_\mathbf{r}^T : \text{ est un projecteur orthogonal sur } Ker(\mathbf{A}) \\
Im(\mathbf{A}^T) = Im(\mathbf{V_r}) \qquad & \mathbf{V_r}\mathbf{V_r}^T : \text{ est un projecteur orthogonal sur } Im(\mathbf{A}^T) \\
Ker(\mathbf{A}^T) = Im(\tilde{\mathbf{U}}_\mathbf{r}) \qquad & \tilde{\mathbf{U}}_\mathbf{r}\tilde{\mathbf{U}}_\mathbf{r}^T : \text{ est un projecteur orthogonal sur } Ker(\mathbf{A}^T)
\end{aligned}
$$

Montrons par exemple que $Ker(\mathbf{A}) = Im(\tilde{\mathbf{V}}_\mathbf{r})$, c'est-à-dire que les $N-r$ dernières colonnes de $\tilde{\mathbf{V}}_\mathbf{r}$ forment une base orthonormale pour le noyau de $\mathbf{A}$, c'est-à-dire pour l'ensemble des vecteurs $\mathbf{x}$ tels que $\mathbf{Ax} = 0$.

Considérons $\mathbf{x} \in Im(\tilde{\mathbf{V}}_\mathbf{r})$, ce vecteur s'écrit sous la forme $\mathbf{x} = \sum_{i=r+1}^{N} w_i\mathbf{v_i}$ avec $w_i \in \mathbb{R}$ et $\mathbf{v_i}$ la i-ème colonne de $\tilde{\mathbf{V}}_\mathbf{r}$. Nous devons montrer que $\mathbf{Ax} = 0$.



$$A = \begin{bmatrix} U_r & \tilde{U}_r \end{bmatrix} \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_r^T \\ \tilde{V}_r^T \end{bmatrix}$$

$$U_r \in \mathbb{R}^{M \times r} \quad \tilde{U}_r \in \mathbb{R}^{M \times M-r} \quad V_r \in \mathbb{R}^{N \times r} \quad \tilde{V}_r \in \mathbb{R}^{N \times N-r}$$

FIGURE 14.2 – SVD réduit

$$
\begin{aligned}
& \mathbf{Ax} \\
= & \{\text{Par définition de } \mathbf{x}.\} \\
& \mathbf{A} \sum_{i=r+1}^{N} w_i \mathbf{v_i} \\
= & \{\text{Par linéarité.}\} \\
& \sum_{i=r+1}^{N} w_i (\mathbf{A}\mathbf{v_i})
\end{aligned}
$$

Il suffit donc de montrer que chaque $\mathbf{Av_i}$ est nul.

$$
\begin{aligned}
& \mathbf{Av_i} \\
= & \{\text{SVD de } \mathbf{A}\} \\
& (\mathbf{U\Sigma V}^T)\mathbf{v_i} \\
= & \{\text{Les colonnes de } \mathbf{V} \text{ sont orthonormales et } \mathbf{v_i} \text{ est une colonne de } \mathbf{V}. \\
& \text{Donc } \mathbf{V}^T\mathbf{v_i} = \mathbf{e_i} \text{ le vecteur avec des 0 partout sauf le 1 en position } i.\} \\
& \mathbf{U\Sigma e_i} \\
= & \{\mathbf{\Sigma e_i} = 0 \text{ car à partir de la (r+1)ème ligne, toutes les lignes de } \mathbf{\Sigma} \text{ sont nulles et } i \geq r+1.\} \\
& 0
\end{aligned}
$$

14.3 Projection sur une base quelconque

Nous cherchons à projeter un vecteur $\mathbf{v}$ sur l'espace des colonnes d'une matrice $\mathbf{A}$. Notons le résultat de cette projection $\mathbf{y} \in Im(\mathbf{A})$. $\mathbf{y} - \mathbf{v}$ est orthogonal à $Im(\mathbf{A})$ (c'est-à-dire qu'il appartient à $Ker(\mathbf{A})$). Autrement dit, en posant $\mathbf{y} = \mathbf{Ax}$, nous avons :

$$
\begin{aligned}
& \mathbf{a_j}^T(\mathbf{y} - \mathbf{v}) = 0, \forall j \\
= & \{\text{Posons } \mathbf{y} = \mathbf{Ax}.\} \\
& \mathbf{a_j}^T(\mathbf{Ax} - \mathbf{v}) = 0, \forall j \\
= & \{\text{Linéarité de la multiplication matricielle.}\} \\
& \mathbf{A}^T(\mathbf{Ax} - \mathbf{v}) = \mathbf{0} \\
= & \\
& \mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{v} \\
= & \{\text{Si } \mathbf{A}^T\mathbf{A} \text{ est inversible.}\} \\
& \mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{v}
\end{aligned}
$$

Ainsi, la projection $\mathbf{y} = \mathbf{Ax} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{v}$. Nous trouvons donc que $\mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ est un projecteur orthogonal sur l'espace des colonnes de $\mathbf{A}$. En fait, il est simple de vérifier que ce projecteur est $\mathbf{U_r}{\mathbf{U_r}}^T$:

$$
\begin{aligned}
\mathbf{A} &= \mathbf{U_r}\mathbf{\Sigma_r}{\mathbf{V_r}}^T \\
\mathbf{A}^T &= \mathbf{V_r}\mathbf{\Sigma_r}{\mathbf{U_r}}^T \\
\mathbf{A}^T\mathbf{A} &= \mathbf{V_r}{\mathbf{\Sigma_r}}^2{\mathbf{V_r}}^T \\
(\mathbf{A}^T\mathbf{A})^{-1} &= \mathbf{V_r}{\mathbf{\Sigma_r}}^{-2}{\mathbf{V_r}}^T \\
(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T &= \mathbf{V_r}{\mathbf{\Sigma_r}}^{-1}{\mathbf{U_r}}^T \\
\mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T &= \mathbf{U_r}{\mathbf{U_r}}^T
\end{aligned}
$$

Chapitre 15

Factorisation QR

15.1 Factorisation QR

Nous introduisons une décomposition matricielle qui nous permettra plus tard d'implémenter les calculs des décompositions en valeurs propres et en valeurs singulières.

Nous cherchons des vecteurs orthonormaux qui forment une base pour les espaces couverts par les colonnes successives d'une matrice $\mathbf{A} \in \mathbb{R}^{M \times N}$. Notons :

$$\begin{aligned} < \mathbf{a_1} > &: \text{l'espace (la ligne) couvert par la première colonne de } \mathbf{A} \\ < \mathbf{a_1}, \mathbf{a_2} > &: \text{l'espace (le plan) couvert par les deux premières colonne de } \mathbf{A} \\ \text{Etc.} &: \text{Etc.} \end{aligned}$$

Nous avons $< \mathbf{a_1} > \subseteq < \mathbf{a_1}, \mathbf{a_2} > \subseteq \cdots \subseteq < \mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_N} >$.

Nous cherchons des vecteurs orthogonaux $\mathbf{q_i}$ tels que :

$$< \mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_j} > = < \mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_j} > \quad \text{pour } j = 1, 2, \ldots, N$$

Ce qui peut s'écrire sous une forme matricielle :

$$(\mathbf{a_1}\ \mathbf{a_2}\ \ldots\ \mathbf{a_N}) = (\mathbf{q_1}\ \mathbf{q_2}\ \ldots\ \mathbf{q_N}) \begin{pmatrix} r_{11} & r_{12} & \ldots & r_{1N} \\ 0 & r_{22} & \ldots & r_{2N} \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & r_{NN} \end{pmatrix}$$

Sans perte de généralité, considérons la situation où $M > N$. Nous appelons alors cette expression, $\mathbf{A} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$ (avec $\hat{\mathbf{Q}} \in \mathbb{R}^{M \times N}$ et $\hat{\mathbf{R}} \in \mathbb{R}^{N \times N}$), la forme réduite de la factorisation QR.

Nous pouvons aussi considérer la forme dite complète de la factorisation QR, avec $\mathbf{Q} \in \mathbb{R}^{M \times M}$ et $\mathbf{R} \in \mathbb{R}^{M \times N}$. Dans ce cas, les $M - N$ dernières lignes de $\mathbf{R}$ sont nulles et il suffit de compléter $\hat{\mathbf{Q}}$ en lui ajoutant $M - N$ colonnes orthogonales.

15.2 Orthogonalisation de Gram-Schmidt

Nous présentons l'algorithme de Gram-Schmidt pour la décomposition QR.

Le principe est de construire $\mathbf{q_j}$ orthogonal à $\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_{j-1}}$ en retirant à la colonne $\mathbf{a_j}$ ses composantes selon les directions de $\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_{j-1}}$:

$$\mathbf{v_j} = \mathbf{a_j} - (\mathbf{q_1}^T\mathbf{a_j})\mathbf{q_1} - (\mathbf{q_2}^T\mathbf{a_j})\mathbf{q_2} - \cdots - (\mathbf{q_{j-1}}^T\mathbf{a_j})\mathbf{q_{j-1}}$$
$$\mathbf{q_j} = \mathbf{v_j}/\|\mathbf{v_j}\|$$

Nous obtenons ainsi la matrice $\hat{\mathbf{Q}}$. Pour $\hat{\mathbf{R}}$, nous avons, par construction des $\mathbf{q_j}$:

$$r_{ij} = \mathbf{q_i}^T\mathbf{a_j} \qquad i \neq j$$
$$|r_{jj}| = \|\mathbf{a_j} - \sum_{i=1}^{j-1} r_{ij}\mathbf{q_i}\| \qquad \text{par convention, nous choisissons } r_{jj} > 0$$

Nous avons ainsi dérivé l'algorithme classique de Gram-Schmidt :

Gram-Schmidt classique (GS)

Résultat: Décomposition QR de la matrice A
Entrée: $\mathbf{A} \in \mathbb{R}^{M\times N}$, avec $M \geq N$
Sortie : $\mathbf{Q} \in \mathbb{R}^{M\times N}$ une matrice orthonormale
$\mathbf{R} \in \mathbb{R}^{N\times N}$ une matrice triangulaire supérieure
telles que $\mathbf{A} = \mathbf{QR}$

```
1 pour j ← 1 à N faire
2 |   v_j ← a_j
3 |   pour i ← 1 à j − 1 faire
4 |   |   r_ij ← q_i^T a_j
5 |   |   v_j ← v_j − r_ij q_i
6 |   finpour
7 |   r_jj ← ‖v_j‖
8 |   q_j ← v_j / r_jj
9 finpour
```

15.3 Algorithme de Gram-Schmidt modifié

15.3.1 Dérivation de l'algorithme de Gram-Schmidt modifié

Nous présentons une autre manière de voir ce résultat qui nous mènera à un algorithme équivalent mais numériquement plus stable.

Nous pouvons considérer que $\mathbf{q_j}$ est obtenu en projetant $\mathbf{a_j}$ sur l'espace orthogonal à $< \mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_{j-1}} >$, cet espace étant par construction égal à $< \mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_{j-1}} >$:

$$\mathbf{q_j} = \frac{\mathbf{P_j a_j}}{\|\mathbf{P_j a_j}\|} \quad ; \quad \mathbf{P_j} = \mathbf{I} - \mathbf{Q_{j-1}} \mathbf{Q_{j-1}}^T \text{ avec } \mathbf{Q_{j-1}} = (\mathbf{q_1}\ \mathbf{q_2}\ \ldots \mathbf{q_{j-1}})$$

De façon équivalente, $\mathbf{P_j}$ peut s'exprimer comme le produit de projecteurs sur les espaces orthogonaux à $\mathbf{q_1}$, puis $\mathbf{q_2}$, etc. jusqu'à $\mathbf{q_{j-1}}$:

$$\mathbf{P_j} = \mathbf{P}_{\perp \mathbf{q_{j-1}}} \mathbf{P}_{\perp \mathbf{q_{j-2}}} \ldots \mathbf{P}_{\perp \mathbf{q_2}} \mathbf{P}_{\perp \mathbf{q_1}} \quad \text{avec } \mathbf{P}_{\perp \mathbf{q}} = \mathbf{I} - \mathbf{q}\mathbf{q}^T$$

Dans la version classique de Gram-Schmidt, $\mathbf{v_j} = \mathbf{P_j a_j}$.

Dans la version modifiée de Gram-Schmidt :

$$\mathbf{v_j} = \mathbf{P}_{\perp \mathbf{q_{j-1}}} \mathbf{P}_{\perp \mathbf{q_{j-2}}} \ldots \mathbf{P}_{\perp \mathbf{q_2}} \mathbf{P}_{\perp \mathbf{q_1}} \mathbf{a_j}$$

Ainsi, nous observons que dès que $\mathbf{q_1}$ est connu ($\mathbf{q_1} = \mathbf{v_1}/\|\mathbf{v_1}\|$, $\mathbf{v_1} = \mathbf{a_1}$), nous pouvons appliquer $\mathbf{P}_{\perp \mathbf{q_1}}$ à $\mathbf{a_2}, \mathbf{a_3}, \ldots, \mathbf{a_N}$. Puis, quand $\mathbf{q_2}$ est connu, nous pouvons appliquer $\mathbf{P}_{\perp \mathbf{q_2}}$ à $\mathbf{P}_{\perp \mathbf{q_1}} \mathbf{a_3}, \mathbf{P}_{\perp \mathbf{q_1}} \mathbf{a_4}, \ldots, \mathbf{P}_{\perp \mathbf{q_1}} \mathbf{a_N}$. Etc. Nous en déduisons l'algorithme de Gram-Schmidt modifié.

Gram-Schmidt modifié (MGS)

Résultat: Décomposition QR de la matrice A
Entrée: $\mathbf{A} \in \mathbb{R}^{M \times N}$, avec $M \geq N$
Sortie : $\mathbf{Q} \in \mathbb{R}^{M \times N}$ une matrice orthonormale
$\mathbf{R} \in \mathbb{R}^{N \times N}$ une matrice triangulaire supérieure
telles que $\mathbf{A} = \mathbf{QR}$

pour $i \leftarrow 1$ **à** N **faire**
  $\mathbf{v_i} \leftarrow \mathbf{a_i}$
finpour
pour $i \leftarrow 1$ **à** N **faire**
  $r_{ii} \leftarrow \|\mathbf{v_i}\|$
  $\mathbf{q_i} \leftarrow \mathbf{v_i}/r_{ii}$
  pour $j \leftarrow i+1$ **à** N **faire**
    $r_{ij} \leftarrow \mathbf{q_i}^T \mathbf{v_j}$
    $\mathbf{v_j} \leftarrow \mathbf{v_j} - r_{ij}\mathbf{q_i}$
  finpour
finpour

15.3.2 Test de l'algorithme

Nous comparons notre implémentation (voir la fonction `mgs` dans le code source associé à ce module) avec le résultat d'un petit exemple issu de Wikipedia.

```
mat <- matrix(c(3,2,1,2),nrow=2,ncol=2,byrow = TRUE)
qr <- mgs(mat)
all.equal(qr$Q, (1/(sqrt(10))) * matrix(c(3,1,-1,3),nrow=2,ncol=2))
```

```
## [1] TRUE
```

15.3.3 Comparaison de la stabilité numérique des deux versions de l'algorithme de Gram-Schmidt

Observons comment de petites erreurs de calculs se propagent dans le cas de l'algorithme de Gram-Schmidt (GS). Mettons qu'une petite erreur ait été commise sur le calcul de $\mathbf{q_2}$ qui s'en trouve de ce fait n'être pas parfaitement perpendiculaire à $\mathbf{q_1}$:

$$\mathbf{q_1}^T\mathbf{q_2} = \epsilon > 0$$

Quelle est l'effet de cette erreur sur la valeur de $\mathbf{v_3}$?

$$\begin{aligned}
\mathbf{v_3} &= \mathbf{a_3} - (\mathbf{q_1}^T\mathbf{a_3})\mathbf{q_1} - (\mathbf{q_2}^T\mathbf{a_3})\mathbf{q_2} \\
\mathbf{q_2}^T\mathbf{v_3} &= \mathbf{q_2}^T\mathbf{a_3} - (\mathbf{q_1}^T\mathbf{a_3})\epsilon - (\mathbf{q_2}^T\mathbf{a_3}) = -(\mathbf{q_1}^T\mathbf{a_3})\epsilon \\
\mathbf{q_1}^T\mathbf{v_3} &= \mathbf{q_1}^T\mathbf{a_3} - (\mathbf{q_1}^T\mathbf{a_3}) - (\mathbf{q_2}^T\mathbf{a_3})\epsilon = -(\mathbf{q_2}^T\mathbf{a_3})\epsilon
\end{aligned}$$

L'erreur commise sur $\mathbf{q_2}$ s'est transmise à $\mathbf{q_3}$ qui n'est plus parfaitement orthogonal à $\mathbf{q_1}$ et $\mathbf{q_2}$.

Pour l'algorithme de Gram-Schmidt modifié (MGS), nous avons :

$$\begin{aligned}
\mathbf{v_3}^{(0)} &= \mathbf{a_3} \\
\mathbf{v_3}^{(1)} &= \mathbf{v_3}^{(0)} - \mathbf{q_1}^T\mathbf{v_3}^{(0)}\mathbf{q_1} \\
\mathbf{v_3} &= \mathbf{v_3}^{(1)} - \mathbf{q_2}^T\mathbf{v_3}^{(1)}\mathbf{q_2} \\
\mathbf{q_2}^T\mathbf{v_3} &= \mathbf{q_2}^T\mathbf{v_3}^{(1)} - \mathbf{q_2}^T\mathbf{v_3}^{(1)} = 0
\end{aligned}$$

Il n'y a pas d'erreur pour l'orthogonalité entre $\mathbf{q_3}$ et $\mathbf{q_2}$. Etudions maintenant l'erreur commise sur l'orthogonalité entre $\mathbf{q_3}$ et $\mathbf{q_1}$.

$$\begin{aligned}
\mathbf{q_1}^T\mathbf{v_3} &= \mathbf{q_1}^T\mathbf{v_3}^{(1)} - \mathbf{q_2}^T\mathbf{v_3}^{(1)}\epsilon \\
\mathbf{q_2}^T\mathbf{v_3}^{(1)} &= \mathbf{q_2}^T\mathbf{v_3}^{(0)} - \mathbf{q_1}^T\mathbf{v_3}^{(0)}\epsilon \\
\mathbf{q_1}^T\mathbf{v_3}^{(1)} &= \mathbf{q_1}^T\mathbf{v_3}^{(0)} - \mathbf{q_1}^T\mathbf{v_3}^{(0)} = 0 \\
\mathbf{q_1}^T\mathbf{v_3} &= -(\mathbf{q_2}^T\mathbf{v_3}^{(0)} - \mathbf{q_1}^T\mathbf{v_3}^{(0)}\epsilon) = -\mathbf{q_2}^T\mathbf{v_3}^{(0)}\epsilon + \mathbf{q_1}^T\mathbf{v_3}^{(0)}\epsilon^2
\end{aligned}$$

Ainsi, comme pour la version classique de Gram-Schmidt, une erreur de l'ordre de grandeur de ϵ est commise sur l'orthogonalité entre $\mathbf{q_1}$ et $\mathbf{q_3}$. Par contre, il n'y a pas de propagation de l'erreur pour l'orthogonalité entre $\mathbf{q_2}$ et $\mathbf{q_3}$. C'est pourquoi l'algorithme de Gram-Schmidt modifié est numériquement plus stable que l'algorithme de Gram-Schmidt classique.

15.4 Autres algorithmes

Notons qu'une autre approche très souvent employée pour le calcul de la décomposition QR est la méthode dite de Householder. Elle assure le maintien d'une quasi parfaite orthogonalité des colonnes de $\mathbf{Q}$. Il existe également l'approche dite des rotations de Givens, plus facilement parallélisable et permettant la mise à jour de la décomposition suite à l'ajout d'une ligne à la matrice $\mathbf{X}$.

Par contre, Gram-Schmidt permet de calculer la forme réduite de la décomposition QR, alors que les algorithmes de Givens et Householder calculent la forme complète. Ainsi, Gram-Schmidt n'est pas tout à fait aussi stable mais peut parfois être plus efficace.

15.5 Décomposition QR pour résoudre un système linéaire au sens des moindres carrés

Une transformation orthogonale $\mathbf{Q}$ ne change pas la norme du vecteur transformé.

$$\|\mathbf{Qy}\|_2^2 = (\mathbf{Qy})^T (\mathbf{Qy}) = \mathbf{y}^T(\mathbf{Q}^T(\mathbf{Qy} = \mathbf{y}^T\mathbf{y} = \|\mathbf{y}\|_2^2$$

Avec $\mathbf{A} = \mathbf{QR}$ (décomposition QR en forme réduite, avec $\mathbf{Q} \in \mathbb{R}^{M\times N}$ et $\mathbf{R} \in \mathbb{R}^{N\times N}$) :

$$\|\mathbf{Ax} - \mathbf{b}\|_2 = \|\mathbf{Q}^T (\mathbf{Ax} - \mathbf{b}) \|_2 = \|\mathbf{Q}^T (\mathbf{QRx} - \mathbf{b}) \|_2 = \|\mathbf{Rx} - \mathbf{Q}^T\mathbf{b}\|_2$$

Or, si la matrice $\mathbf{Rx}$ n'est pas singulière (i.e., s'il n'y a pas de colinéarités entre ses colonnes) le système linéaire $\mathbf{Rx} = \mathbf{Q}^T\mathbf{b}$, à N équations et N inconnues, se résout simplement par substitutions car $\mathbf{R}$ est de forme diagonale supérieure.

15.6 Annexe code source

```
# 12 Factorisation QR

# Norm 2
Norm <- function(x) {
  stopifnot(is.numeric(x))
  sqrt(sum(x^2))
}

# Modified Gram-Schmidt
mgs <- function(A, tol = .Machine$double.eps^0.5) {
  stopifnot(is.numeric(A), is.matrix(A))
  M <- nrow(A); N <- ncol(A)
  if(M<N) stop("Le nombre de lignes de 'A' doit être supérieur ou égal au nombre
               de colonnes")
  Q <- matrix(0, M, N)
  R <- matrix(0, N, N)
  for (i in 1:N) Q[,i] <- A[,i]
```

```
  for (i in 1:N) {
    R[i,i] <- Norm(Q[,i])
    if (abs(R[i,i]) <= tol) stop("La matrice 'A' n'est pas de plein rang.")
    Q[,i] <- Q[,i] / R[i,i]
    j <- i+1
    while(j<=N) {
      R[i,j] <- t(Q[,i]) %*% Q[,j]
      Q[,j] <- Q[,j] - R[i,j] * Q[,i]
      j <- j+1
    }
  }
  return(list(Q = Q, R = R))
}
```

Chapitre 16

Puissance itérée par bloc et SVD

16.1 Principe et application à la décomposition en valeurs propres

Nous avons vu précédemment que la méthode de la puissance itérée permet de calculer la plus grande valeur propre et un vecteur propre associé pour une matrice carrée $\mathbf{A}$. La décomposition QR est au cœur d'une approche élégante pour étendre la méthode de la puissance itérée au calcul des S plus grandes valeurs propres et leurs vecteurs propres associés.

Une itération consiste à multiplier $\mathbf{A}$ par un bloc $\mathbf{V} \in \mathbb{R}^{N \times S}$ de S vecteurs colonnes. Si ce processus est répété, nous nous attendons à trouver le même résultat que pour la méthode de la puissance itérée : chaque colonne de $\mathbf{V}$ convergera vers le (même) plus grand vecteur propre.

Mais si, par une décomposition QR, nous forçons les colonnes de $\mathbf{V}$ à rester orthogonales, pour une matrice $\mathbf{A}$ symétrique, le processus itératif fera tendre ces colonnes vers différents vecteurs propres et la diagonale de $\mathbf{R}$ contiendra les valeurs propres correspondantes.

Puissance itérée par bloc pour les vecteurs propres

Résultat: Vecteurs propres d'une matrice symétrique par la méthode de la puissance itérée par bloc

Entrée: $\mathbf{A} \in \mathbb{R}^{N \times N}$ symétrique
$\mathbf{V} \in \mathbb{R}^{N \times S}$

Sortie : Les colonnes de $\mathbf{V}$ sont les S premiers vecteurs propres.
Les valeurs propres correspondantes sont sur la diagonale de $\mathbf{\Lambda}$.

1 **tant que** $err > \epsilon$ **faire**
2 $\quad \mathbf{B} \leftarrow \mathbf{AV}$
3 $\quad \mathbf{B} = \mathbf{QR}$
4 $\quad \mathbf{V} \leftarrow S$ premières colonnes de $\mathbf{Q}$
5 $\quad \mathbf{\Lambda} \leftarrow S$ premières valeurs diagonales de R
6 $\quad err \leftarrow \|\mathbf{AV} - \mathbf{V\Lambda}\|$
7 **fintq**

16.2 Application à la décomposition en valeurs singulières

Nous pouvons de même adapter l'algorithme de la puissance itérée au cas du SVD pour calculer les S plus grandes valeurs singulières et les vecteurs singuliers associés pour une matrice $\mathbf{A} \in \mathbb{R}^{M \times N}$.

Puissance itérée par bloc pour le SVD

Résultat: Valeurs singulières d'une matrice symétrique par la méthode de la puissance itérée par bloc

Entrée: $\mathbf{A} \in \mathbb{R}^{M \times N}$
$\mathbf{V} \in \mathbb{R}^{N \times S}$ peut être initialisée avec des 1 sur la diagonale principale et des 0 ailleurs.

Sortie : $\mathbf{U} \in \mathbb{R}^{M \times S}, \mathbf{V} \in \mathbb{R}^{N \times S}$
$\mathbf{\Sigma} = diag(\sigma_1, \sigma_2, \ldots, \sigma_S)$
telles que $\mathbf{AV} = \mathbf{U\Sigma}$

1 **tant que** $err > \epsilon$ **faire**
2 $\quad \mathbf{AV} = \mathbf{QR}$
3 $\quad \mathbf{U} \leftarrow S$ premières colonnes de $\mathbf{Q}$
4 $\quad \mathbf{A}^T\mathbf{U} = \mathbf{QR}$
5 $\quad \mathbf{V} \leftarrow S$ premières colonnes de $\mathbf{Q}$
6 $\quad \mathbf{\Sigma} \leftarrow S$ premières lignes et colonnes de R
7 $\quad err \leftarrow \|\mathbf{AV} - \mathbf{U\Sigma}\|$
8 **fintq**

Chapitre 17

Géométrie de la régression ridge et SVD

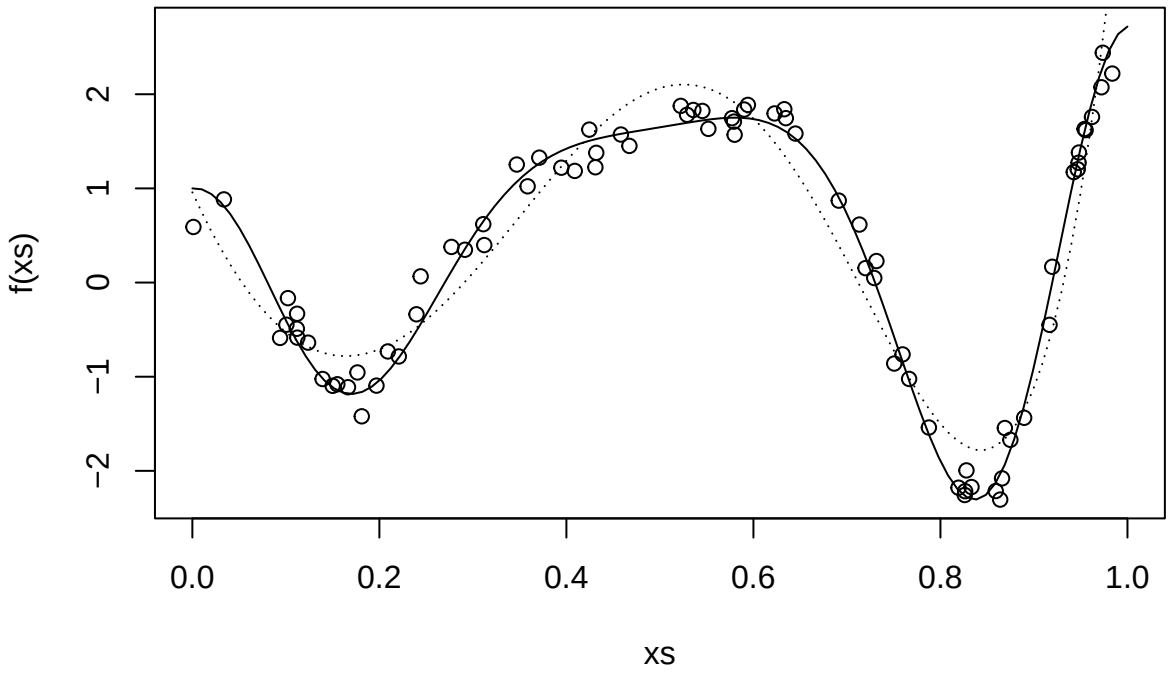
17.1 Coefficients de la régression ridge en fonction du SVD

Exprimons le calcul des coefficients d'une régression ridge en fonction de la décomposition en valeurs singulières de la matrice des données observées $\mathbf{X} \in \mathbb{R}^{M \times N}$.

$$
\begin{aligned}
& \hat{\beta}_\lambda \\
= & \{\text{Voir la dérivation de la régression ridge dans un précédent module.}\} \\
& (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \\
= & \{\text{SVD de } \mathbf{X} : \mathbf{U}\mathbf{D}\mathbf{V}^T \quad \mathbf{U} \in \mathbb{R}^{M \times M}\ ,\ \mathbf{D} \in \mathbb{R}^{M \times N}\ ,\ \mathbf{V} \in \mathbb{R}^{N \times N}\} \\
& (\mathbf{V}\mathbf{D}^T\mathbf{U}^T\mathbf{U}\mathbf{D}\mathbf{V}^T + \lambda\mathbf{I})^{-1}\mathbf{V}\mathbf{D}^T\mathbf{U}^T\mathbf{y} \\
= & \{\mathbf{U} \text{ et } \mathbf{V} \text{ sont orthogonales : } \mathbf{I} = \mathbf{V}\mathbf{V}^T\} \\
& (\mathbf{V}\mathbf{D}^T\mathbf{D}\mathbf{V}^T + \lambda\mathbf{V}\mathbf{V}^T)^{-1}\mathbf{V}\mathbf{D}^T\mathbf{U}^T\mathbf{y} \\
= & \\
& (\mathbf{V}(\mathbf{D}^T\mathbf{D} + \lambda\mathbf{I})\mathbf{V}^T)^{-1}\mathbf{V}\mathbf{D}^T\mathbf{U}^T\mathbf{y} \\
= & \{(\mathbf{X}\mathbf{Y})^{-1} = \mathbf{Y}^{-1}\mathbf{X}^{-1} \quad \mathbf{V} \text{ est orthogonale : } \mathbf{V}^{-1} = \mathbf{V}^T\} \\
& \mathbf{V}(\mathbf{D}^T\mathbf{D} + \lambda\mathbf{I})^{-1}\mathbf{V}^T\mathbf{V}\mathbf{D}^T\mathbf{U}^T\mathbf{y} \\
= & \\
& \mathbf{V}(\mathbf{D}^T\mathbf{D} + \lambda\mathbf{I})^{-1}\mathbf{D}^T\mathbf{U}^T\mathbf{y} \\
= & \{\text{Soit } d_j \text{ le jème élément sur la diagonale de } \mathbf{D},\ \mathbf{u_j} \text{ et } \mathbf{v_j} \text{ les jèmes colonnes de resp. } \mathbf{U} \text{ et } \mathbf{V}\} \\
& \sum_{d_j > 0} \mathbf{v_j} \frac{d_j}{d_j^2 + \lambda} \mathbf{u_j}^T\mathbf{y}
\end{aligned}
$$

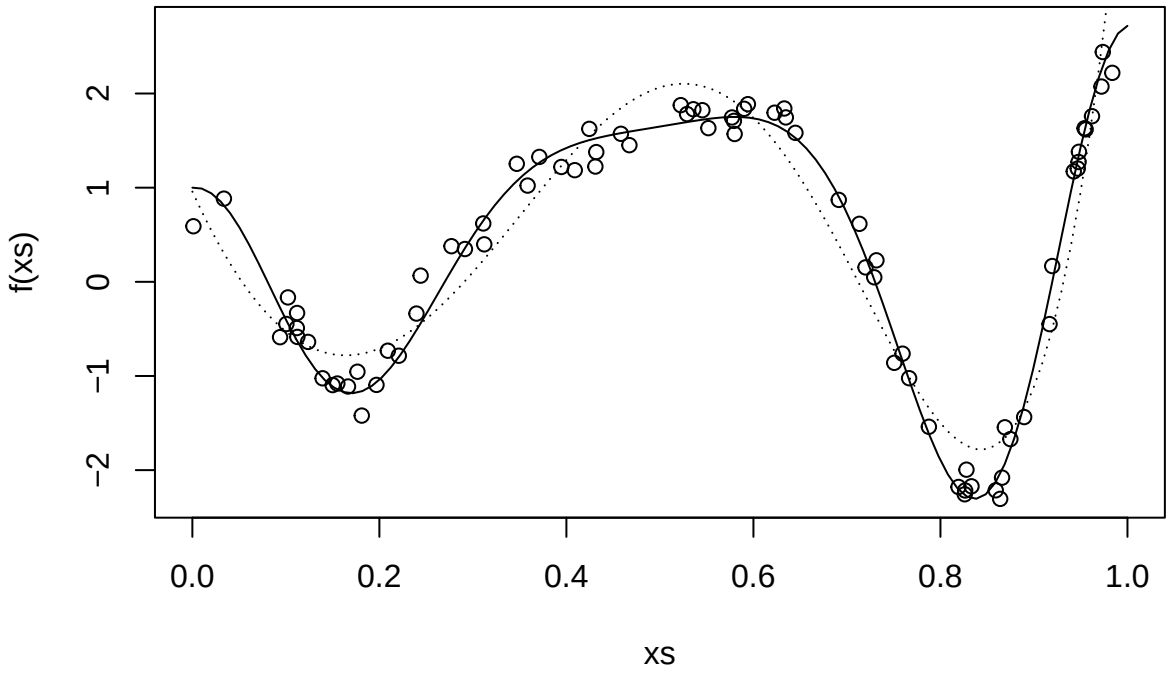
Nous vérifions sur un exemple synthétique que les coefficients inférés par résolution du système linéaire correspondant à la matrice de Gram sont identiques à ceux inférés par l'approche basée sur la décomposition en valeurs singulières. Nous commençons par l'inférence basée sur la matrice de Gram.

```
set.seed(1123)
n <- 100
deg1 <- 8
data = gendat(n,0.2)
splitres <- splitdata(data,0.8)
entr <- splitres$entr
test <- splitres$test
alpha <- 1E-5
coef.gram <- ridge.gram(alpha, entr, deg1)
plt(entr,f)
pltpoly(coef.gram)
```



Nous inférons maintenant les coefficients à partir du SVD de la matrice des données.

```
ridge <- ridge.svd(entr, deg1)
coef.svd <- ridge(alpha)
plt(entr,f)
pltpoly(coef.svd)
```



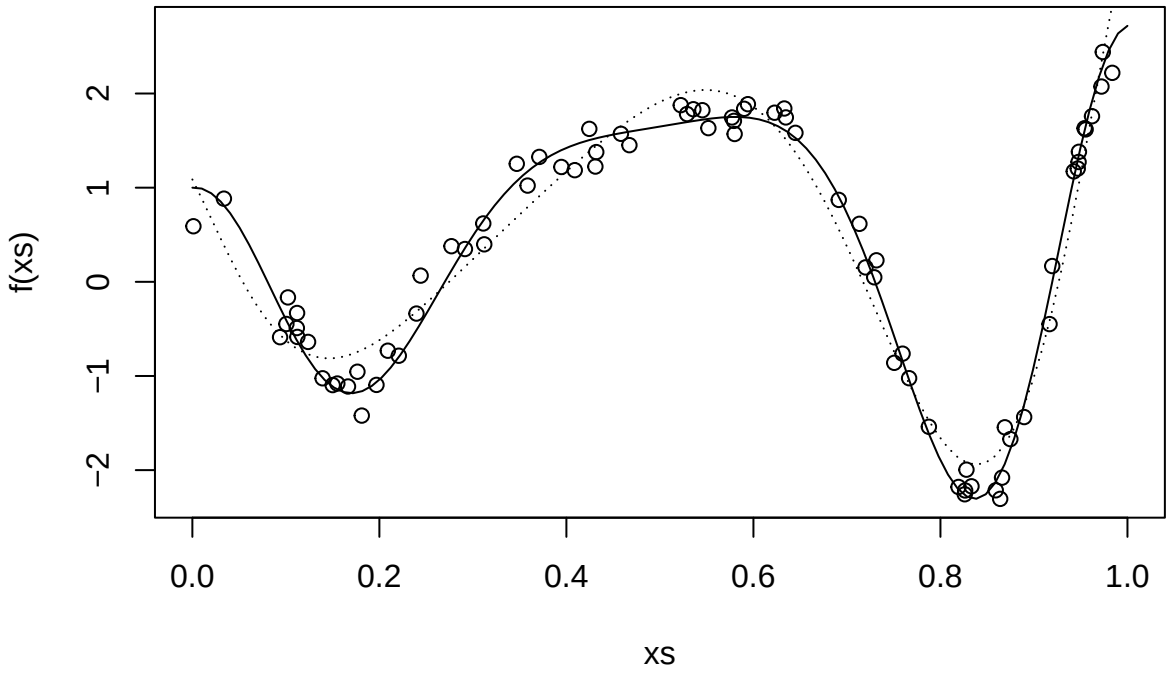
Nous vérifions que les coefficients trouvés par les deux approches sont identiques.

```
all.equal(coef.gram,coef.svd)
```

```
## [1] TRUE
```

La décomposition en valeurs singulières de $\mathbf{X}$ donne les coefficients de la régression Ridge pour toutes les valeurs possibles du coefficient de régularisation λ. Nous pouvons donc implémenter de façon efficace une validation croisée à k plis.

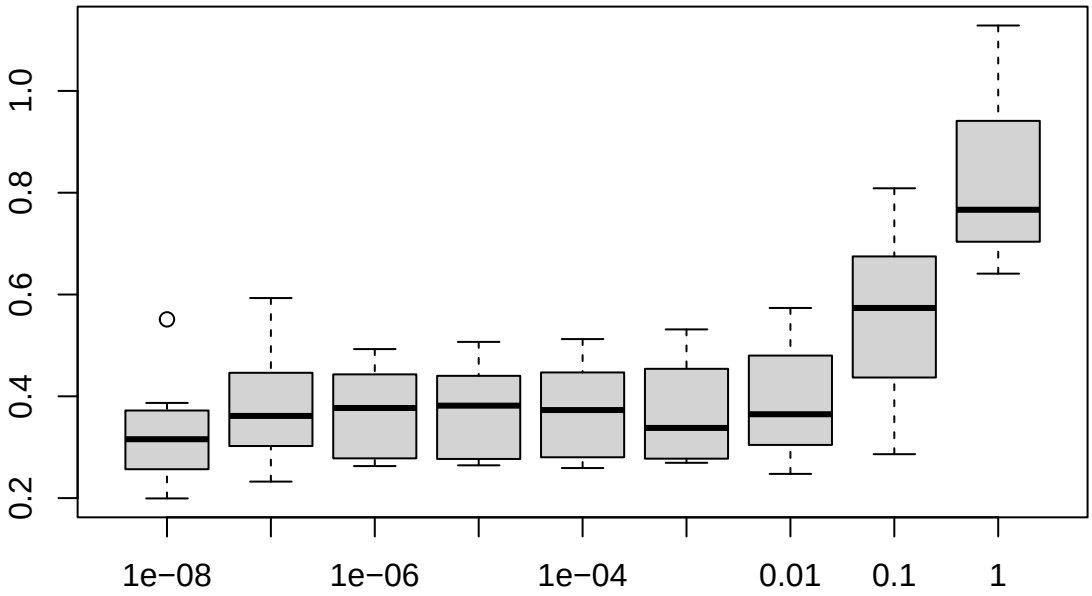
```
alphas <- c(1E-8, 1E-7, 1E-6, 1E-5, 1E-4, 1E-3, 1E-2, 1E-1, 1)
reskfold <- kfoldridge(K = 10, alphas = alphas, data = entr, degre = deg1)
plt(entr,f)
pltpoly(reskfold$coef)
```



Ci-dessus, nous avons généré un jeu de données composé de 100 observations et nous avons calculé par validation croisée un polynôme de degré au plus égal à 8 qui modélise au mieux ces données. La valeur de α retenue est : 10^{-8}.

Traçons un boxplot des erreurs commises sur les plis de validation pour chaque valeur de α.

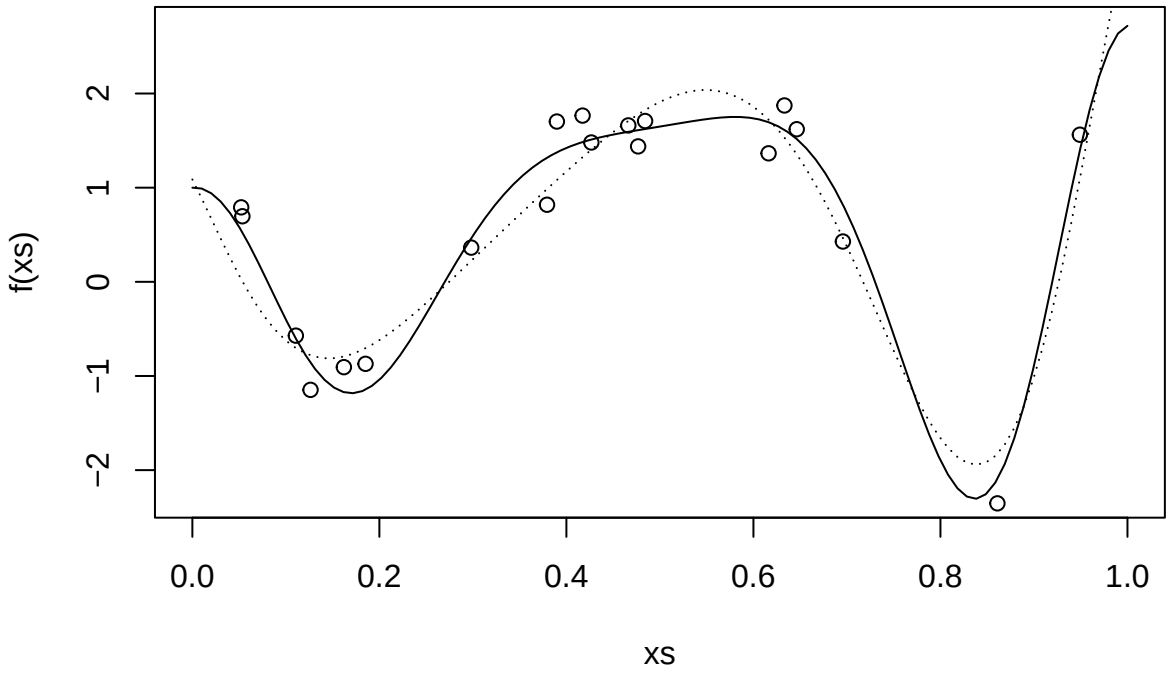
```
boxplot(reskfold$maes)
```




```
testpred <- polyeval(reskfold$coef, test$X)
testmae <- mean(abs(testpred - test$Y))
```

Ce meilleur modèle atteint une erreur absolue moyenne de 0.3089328 sur le jeu de test.

```
plt(test,f)
pltpoly(reskfold$coef)
```



17.2 Régression ridge et géométrie

Observons la relation entre les étiquettes prédites $\hat{\mathbf{y}}_\lambda$ et les étiquettes observées $\mathbf{y}$.

$$\begin{aligned}\hat{\mathbf{y}}_\lambda &= \mathbf{X}\hat{\beta}_\lambda \\ &= \mathbf{U}\mathbf{D}\mathbf{V}^T\mathbf{V}(\mathbf{D}^T\mathbf{D}+\lambda\mathbf{I})^{-1}\mathbf{D}^T\mathbf{U}^T\mathbf{y} \\ &= \sum_{d_j>0}\mathbf{u_j}\frac{d_j^2}{d_j^2+\lambda}\mathbf{u_j}^T\mathbf{y}\end{aligned}$$

Nous remarquons qu'en labsence de régularisation, $\lambda = 0$, les valeurs estimées $\hat{\mathbf{y}}$ sont les projections sur les axes principaux $\mathbf{u_j}$ – qui couvrent l'espace des colonnes de $\mathbf{X}$, i.e. $Im(\mathbf{X})$ – des valeurs observées $\mathbf{y}$.

En présence de régularisation, $\lambda > 0$, les coordonnées, sur les axes principaux, de l'estimation $\hat{\mathbf{y}}_\lambda$ sont de plus en plus contractées lorsqu'on progresse vers les axes qui expliquent de moins en moins la variabilités des données.

17.3 Annexe code source

```
# 14 Géométrie Ridge

# Séparer le jeu de données en un jeu d'entraînement et un jeu de test
# INPUT : jeu de données initial et proportion des données conservées pour
#         l'entraînement.
```

```r
splitdata <- function(data,p) {
  n <- nrow(data$X)
  nentr <- round(p*n)
  entridx <- sample(1:n, nentr, replace=FALSE)
  list(entr  = list(X = data$X[entridx,,drop=FALSE],  Y = data$Y[entridx]),
       test = list(X = data$X[-entridx,,drop=FALSE], Y = data$Y[-entridx]))
}

# Résolution d'un système linéaire correspondant à la matrice de Gram pour
# un polynôme de degré fixé et avec l'ajout d'un facteur de régularisation en
# norme L2 dont l'importance est contrôlée par l'hyperparamètre alpha.
ridge.gram <- function(alpha, data, degre) {
  A <- scale(outer(c(data$X), 1:degre, "^"))
  Y <- data$Y
  Ym <- mean(Y)
  Y <- Y - Ym
  gram <- t(A) %*% A
  diag(gram) <- diag(gram) + alpha
  coef <- solve(gram, as.vector(t(A) %*% Y))
  coef <- coef / attr(A,"scaled:scale")
  inter <- Ym - coef %*% attr(A,"scaled:center")
  coef <- c(inter, coef)
}

ridge.svd <- function(data, degre, fold = FALSE) {
  if (length(fold) == 1 && fold == FALSE) {
    X <- data$X
    Y <- data$Y
  } else {
    X <- data$X[!fold,]
    Y <- data$Y[!fold]
  }
  A <- scale(outer(c(X), 1:degre, "^"))
  Ym <- mean(Y)
  Y <- Y - Ym
  As <- svd(A)
  d <- As$d
  function(alpha) {
    coef <- c(As$v %*% ((d / (d^2 + alpha)) * (t(As$u) %*% Y)))
    coef <- coef / attr(A,"scaled:scale")
    inter <- Ym - coef %*% attr(A,"scaled:center")
    coef <- c(inter, coef)
  }
}

# alphas[l] est une liste de valeurs pour l'hyperparamètre alpha.
```

```
# Notons Ridge[l] un modèle avec alpha <- alphas[l].
# Découper aléatoirement le jeu d'entraînement en K plis F[i] disjoints.
# Pour l <- [1,...,L]
#   Pour i <- [1,...,K]
#     Apprendre Ridge[l] sur l'union des plis F[j] avec j!=i
#     Calculer le score de Ridge[l] sur le pli de validation F[i]
#   Conserver les résultats du modèle sur les plis de validation.
#   Soit Moy[l] la moyenne des résultats de Ridge[l] sur les plis de validation.
# Soit l' l'indice du maximum de Moy[l]
# Apprendre Ridge[l'] sur l'ensemble du jeu de données d'entraînement.
# Retourner ce modèle.
kfoldridge <- function(K, alphas, data, degre) {
  N <- nrow(data$X)
  folds <- rep_len(1:K, N)
  folds <- sample(folds, N)
  maes <- matrix(data = NA, nrow = K, ncol = length(alphas))
  colnames(maes) <- alphas
  for(k in 1:K) {
    fold <- folds == k
    ridge <- ridge.svd(data, degre, fold)
    alpha_idx <- 1
    X <- data$X[fold,]
    Y <- data$Y[fold]
    for(alpha in alphas) {
      coef <- ridge(alpha)
      pred <- polyeval(coef, X)
      maes[k,alpha_idx] <- mean(abs(pred - Y))
      alpha_idx <- alpha_idx + 1
    }
  }
  mmaes <- colMeans(maes)
  minmmaes <- min(mmaes)
  bestalpha <- alphas[which(mmaes == minmmaes)]
  ridge <- ridge.svd(data, degre)
  coef <- ridge(bestalpha)
  list(coef = coef, maes = maes, alpha = bestalpha)
}
```

Chapitre 18

Validation croisée un contre tous

La validation croisée un contre tous (LOOCV, Leave One Out Cross Validation) est un cas extrême de la validation croisée à k plis. Pour un jeu de données constitué de n observations, il s'agit de faire une validation croisée à n plis. Nous estimons par LOOCV l'erreur commise par un modèle de régression ridge pour une valeur fixée de l'hyper-paramètre λ.

Nous notons $\hat{\boldsymbol{\beta}}_\lambda^{(-i)}$ les coefficients de la régression ridge apprise en utilisant $n-1$ observations, après avoir retiré la paire $(\mathbf{x_i}, y_i)$.

$$LOO_\lambda = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \mathbf{x_i}^T\hat{\boldsymbol{\beta}}_\lambda^{(-i)}\right)^2$$

Nous détaillons l'expression de $\hat{\boldsymbol{\beta}}_\lambda^{(-i)}$.

$$\hat{\boldsymbol{\beta}}_\lambda^{(-i)} = \left(\mathbf{X^{(-i)}}^T\mathbf{X^{(-i)}} + \lambda\mathbf{I}\right)^{-1}\mathbf{X^{(-i)}}^T\mathbf{y^{(-i)}}$$

Nous précisons le sens de $\mathbf{X^{(-i)}}^T\mathbf{X^{(-i)}} + \lambda\mathbf{I}$.

$$\mathbf{X^{(-i)}}^T\mathbf{X^{(-i)}} + \lambda\mathbf{I} = \mathbf{X^T X} + \lambda\mathbf{I} - \mathbf{x_i x_i}^T$$

Il faut remarquer que nous opérons la soustraction de la matrice $\mathbf{x_i x_i}^T$ (un produit externe) et non du scalaire $\mathbf{x_i}^T\mathbf{x_i}$.

Comme étape intermédiaire, nous rappelons maintenant la définition de la matrice de chapeau (hat matrix) $\mathbf{H}$.

$$\hat{\mathbf{y}}_\lambda = \mathbf{X}\hat{\boldsymbol{\beta}}_\lambda = \mathbf{X}\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{H}\mathbf{y}$$

Remarque : on parle de la matrice de chapeau, ou matrice de projection car elle "ajoute un chapeau à $\mathbf{y}$". C'est-à-dire qu'elle représente la transformation linéaire, indépendante de $\mathbf{y}$ (elle ne dépend que de $\mathbf{X}$, la matrice des valeurs des variables explicatives pour chaque observation), qui projette les valeurs observées $\mathbf{y}$ sur les valeurs prédites $\hat{\mathbf{y}}$.

Nous avons déjà rencontré un projecteur orthogonal similaire à l'occasion d'un précédent module sur le concept de projecteur. C'était dans le contexte de la régression non régularisée. Dans ce cas, h_{ij} s'interprète directement comme l'influence qu'exerce y_j sur la prédiction $\hat{y}_i$. Cette influence ne dépend pas de la valeur y_j puisque $\mathbf{H}$ ne dépend que de $\mathbf{X}$. En général, l'influence de y_i sur la prédiction apparaît le plus directement lorsqu'on regarde son influence sur $\hat{y}_i$. Ainsi, les valeurs h_{ii} (la diagonale de $\mathbf{H}$) permettent d'identifier les points dont l'influence sur le modèle est prépondérante.

Nous utilisons la notation raccourcie h_i pour les éléments diagonaux de la matrice $\mathbf{H}$.

$$\mathbf{x_i}^T \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1} \mathbf{x_i} = \mathbf{H}_{ii} = h_i$$

Dans un précédent module, nous avons découvert la relation entre $\hat{\mathbf{y}}_\lambda$ et $\mathbf{y}$ en fonction de la décomposition en valeurs singulières de $\mathbf{X}$:

$$\hat{\mathbf{y}}_\lambda = \sum_{d_j>0} \mathbf{u_j} \frac{d_j^2}{d_j^2+\lambda} \mathbf{u_j}^T \mathbf{y}$$

En notant $\mathbf{S}(\lambda)$ la matrice diagonale qui compresse les dimensions originales et dont les éléments sont : $\frac{d_j^2}{d_j^2+\lambda}$, nous avons :

$$\mathbf{H}(\lambda) = \mathbf{U}\mathbf{S}(\lambda)\mathbf{U}^T$$

Nous pouvons en particulier exprimer les valeurs h_i en fonction du SVD de $\mathbf{X}$:

$$h_i = \sum_{d_j>0} \frac{d_j^2}{d_j^2+\lambda} u_{ij}$$

Observons la trace de $\mathbf{H}(\lambda)$, c'est-à-dire la somme de ses éléments diagonaux, l'expression se simplifie car, comme $\mathbf{U}$ est orthogonale, $\mathbf{u_i}^T\mathbf{u_i} = 1$ et $\mathbf{u_i}^T\mathbf{u_j} = 0$ pour $i \neq j$:

$$tr(\mathbf{H}(\lambda)) = \sum_i h_i = \sum_{d_j>0} \frac{d_j^2}{d_j^2+\lambda}$$

Pour la régression non régularisée, $tr(\mathbf{H}(\mathbf{0})) = rang(\mathbf{X})$. Quand $n > p$ et X est de plein rang (i.e., pas de colinéarités), alors $tr(\mathbf{H}(\mathbf{0})) = p$. En général, on dit que $tr(\mathbf{H}(\mathbf{0}))$ est le degré de liberté de

la régression. Par extension, on parle également de "degré de liberté" pour $tr(\mathbf{H}(\lambda))$ avec $\lambda \neq 0$. Nous observons que la régularisation Ridge diminue le degré de liberté pour donner un modèle plus "simple" (que celui d'une régression non régularisée) qui a le potentiel de mieux généraliser ses prédictions pour de nouvelles observations.

Rappelons également la formule de Morrison qui permet de calculer la mise à jour de l'inverse d'une matrice $\mathbf{A}$ après l'ajout d'une matrice de rang 1 $\mathbf{uv}^T$.

$$\left(\mathbf{A} + \mathbf{uv}^T\right)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{uv}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}$$

Nous utilisons ce résultat pour exprimer l'inversion nécessaire au calcul d'une des n régressions du calcul de l'erreur par LOOCV en fonction de l'inversion nécessaire au calcul de la régression sur toutes les données.

$$\begin{aligned}
&\left(\mathbf{X}^{(-\mathbf{i})^T}\mathbf{X}^{(-\mathbf{i})} + \lambda\mathbf{I}\right)^{-1} \\
&= \{\text{Par définition de } \mathbf{X}^{(-\mathbf{i})}\} \\
&\left(\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda\mathbf{I} - \mathbf{x_i}\mathbf{x_i}^T\right)^{-1} \\
&= \{\text{Formule de Morrison et définition de } h_i\} \\
&\left(\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda\mathbf{I}\right)^{-1} + \frac{\left(\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{x_i}\mathbf{x_i}^T\left(\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda\mathbf{I}\right)^{-1}}{1 - h_i}
\end{aligned}$$

Nous remarquons aussi que :

$$\mathbf{X}^{(-\mathbf{i})^T}\mathbf{y}^{(-\mathbf{i})} = \mathbf{X}^T\mathbf{y} - \mathbf{x_i}y_i$$

Ces résultats intermédiaires, nous permettent d'exprimer $\hat{\boldsymbol{\beta}}_\lambda^{(-i)}$ en fonction de $\hat{\boldsymbol{\beta}}_\lambda$.

$$\begin{aligned}
&\hat{\boldsymbol{\beta}}_\lambda^{(-i)} \\
&= \left(\mathbf{X}^{(-\mathbf{i})^T}\mathbf{X}^{(-\mathbf{i})} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^{(-\mathbf{i})^T}\mathbf{y}^{(-\mathbf{i})} \\
&= \left[\left(\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda\mathbf{I}\right)^{-1} + \frac{\left(\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{x_i}\mathbf{x_i}^T\left(\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda\mathbf{I}\right)^{-1}}{1 - h_i}\right]\left(\mathbf{X}^T\mathbf{y} - \mathbf{x_i}y_i\right) \\
&= \hat{\boldsymbol{\beta}}_\lambda - \left[\frac{\left(\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{x_i}}{1 - h_i}\right]\left[-\mathbf{x_i}^T\hat{\boldsymbol{\beta}}_\lambda + h_i y_i + (1 - h_i)y_i\right] \\
&= \hat{\boldsymbol{\beta}}_\lambda - \frac{\left(\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{x_i}}{1 - h_i}\left(y_i - \hat{y}_{\lambda i}\right)
\end{aligned}$$

Nous pouvons ainsi découvrir une version de l'expression $\left(y_i - \hat{y}_{\lambda i}^{(-i)}\right)$ qui ne demande pas de calculer un nouveau modèle pour chaque observation i retirée du jeu de données d'entraînement.

$$
\begin{aligned}
& y_i - \hat{y}_{\lambda i}^{(-i)} \\
=& y_i - \mathbf{x_i}^T \hat{\boldsymbol{\beta}}_\lambda^{(-i)} \\
=& y_i - \mathbf{x_i}^T \left[\hat{\boldsymbol{\beta}}_\lambda - \frac{\left(\mathbf{X^T X} + \lambda \mathbf{I}\right)^{-1} \mathbf{x_i}}{1 - h_i} \left(y_i - \hat{y}_{\lambda i}\right) \right] \\
=& \left(y_i - \hat{y}_{\lambda i}\right) + \left(y_i - \hat{y}_{\lambda i}\right) \frac{h_i}{1 - h_i} \\
=& \frac{y_i - \hat{y}_{\lambda i}}{1 - h_i}
\end{aligned}
$$

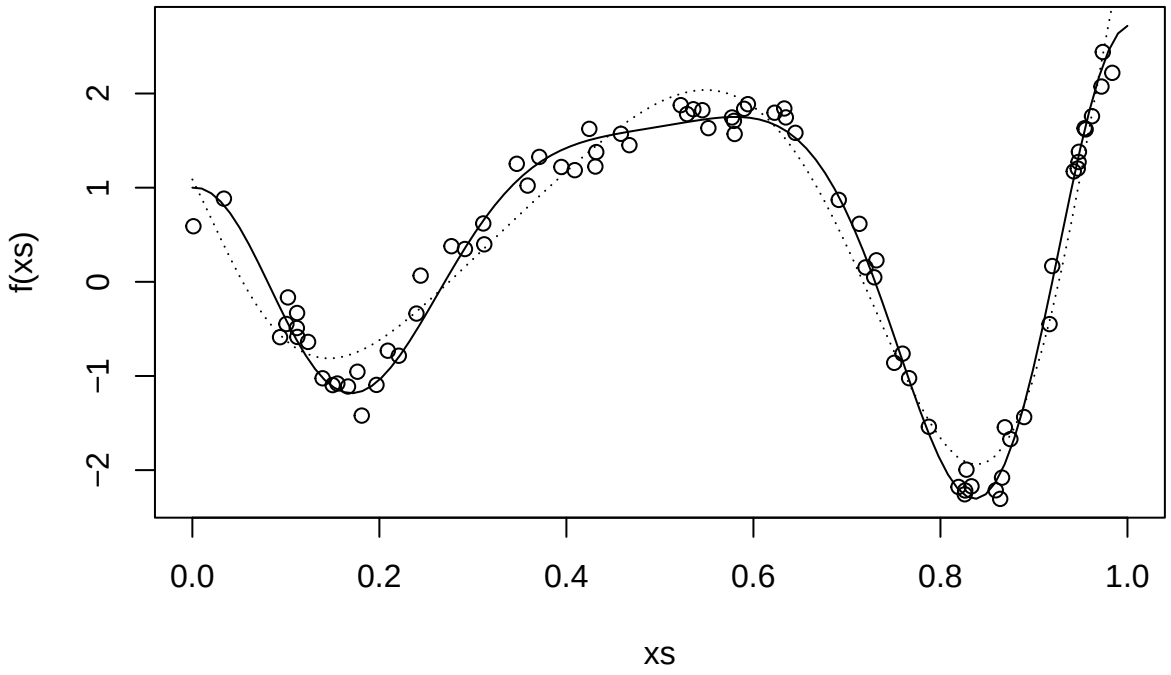
Finalement, nous avons découvert une expression de l'erreur LOOCV basée sur le SVD de $\mathbf{X}$:

$$
\begin{aligned}
LOO_\lambda &= \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_{\lambda i}}{1 - h_i} \right)^2 \\
\text{avec } h_i &= \sum_{d_j > 0} \frac{d_j^2}{d_j^2 + \lambda} u_{ij}
\end{aligned}
$$

Nous remarquons que cette mesure de l'erreur peut être instable quand au moins l'un des h_i est proche de 1. Une solution est de remplacer dans cette expression chaque h_i par la moyenne de tous les h_i, c'est-à-dire $\frac{1}{n} tr(\mathbf{H}(\lambda))$. Nous obtenons une nouvelle mesure de l'erreur appelée *validation croisée généralisée* (GCV pour "Generalized Cross Validation").

$$
\begin{aligned}
GCV_\lambda &= \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_{\lambda i}}{1 - \frac{1}{n} tr(\mathbf{H}(\lambda))} \right)^2 \\
\text{avec } tr(\mathbf{H}(\lambda)) &= \sum_{d_j > 0} \frac{d_j^2}{d_j^2 + \lambda}
\end{aligned}
$$

```
set.seed(1123)
n <- 100
deg <- 8
data = gendat(n,0.2)
splitres <- splitdata(data,0.8)
entr <- splitres$entr
test <- splitres$test
lambdas <- 10^seq(2,-8,by=-1)
entr.poly <- outer(c(entr$X), 1:deg, "^")
rm <- ridge(entr.poly, entr$Y, lambdas)
plt(entr,f)
pltpoly(rm$coef)
```

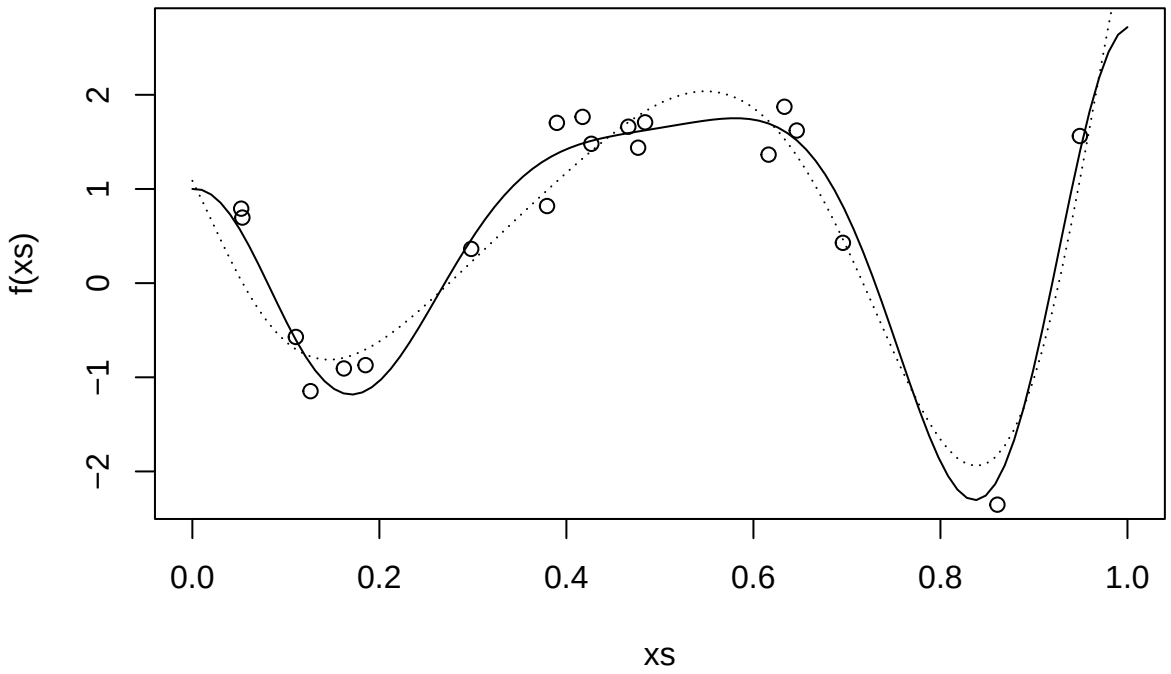


Ci-dessus, nous avons généré un jeu de données composé de 100 observations et nous avons calculé par validation croisée un contre tous un polynôme de degré au plus égal à 8 qui modélise au mieux ces données. La valeur de λ retenue est : 10^{-8} et l'erreur absolue moyenne sur le jeu d'entraînement est : 0.2547904.

```
testpred <- polyeval(rm$coef, test$X)
testmae <- mean(abs(testpred - test$Y))
```

Ce meilleur modèle atteint une erreur absolue moyenne de 0.3089328 sur le jeu de test.

```
plt(test,f)
pltpoly(rm$coef)
```



18.1 Annexe code source

```
# 15 LOOCV

ridgeSVD <-
function(X, y)
{
  n <- nrow(X)
  X.init <- X
  y.init <- y
  X <- scale(X)
  ym <- mean(y)
  y <- y - ym
  Xs <- svd(X)
  d <- Xs$d
  function(lambda) {
    coef <- c(Xs$v %*% ((d / (d^2 + lambda)) * (t(Xs$u) %*% y)))
    coef <- coef / attr(X,"scaled:scale")
    inter <- ym - coef %*% attr(X,"scaled:center")
    coef <- c(inter, coef)
    trace.H <- sum(d^2 / (d^2 + lambda))
    yh <- coef[1] + X.init%*%coef[-1]
    gcv <- sum( ((y.init - yh) / (1 - (trace.H / n))) ^ 2 ) / n
    list(coef = coef, gcv = gcv)
  }
}

ridge <-
function(X, y, lambdas=NULL)
{
  X <- as.matrix(X)
  p <- ncol(X)
  if(is.null(lambdas)) { lambdas <- 10^seq(-8,8,by=0.5) }
  errs <- double(length(lambdas))
  coefs <- matrix(data = NA, nrow = length(lambdas), ncol = p+1)
  ridge <- ridgeSVD(X, y)
  idx <- 1
  for(lambda in lambdas) {
    res <- ridge(lambda)
    coefs[idx,] <- res$coef
    errs[idx] <- res$gcv
    idx <- idx + 1
  }
  err.min <- min(errs)
  lambda.best <- lambdas[which(errs == err.min)]
  coef.best <- coefs[which(errs == err.min),]
```

```
  yh <- coef.best[1] + X%*%coef.best[-1]
  mae <- mean(abs(yh - y))
  r <- list(coef = coef.best,
            lambda = lambda.best,
            mae = mae,
            coefs=coefs,
            lambdas=lambdas)
  class(r) <- "ridge"
  return(r)
}

predict.ridge <-
function(o, newdata)
{
  newdata <- as.matrix(newdata)
  if(length(o$coef)-1!=ncol(newdata)) {
    stop("Not the same number of variables btwn fitted ridge object and new data")
  }
  yh <- o$coef[1] + newdata%*%o$coef[-1]
}
```

Chapitre 19

Biais et variance d'un estimateur

19.1 Biais et variance pour l'estimateur d'un paramètre d'un modèle paramétrique

Nous notons $\hat{\mu}$ la moyenne ($\hat{\mu} = E[\mathbf{x}]$) et $\hat{\sigma}^2$ la variance ($\hat{\sigma}^2 = E[(\mathbf{x} - E[\mathbf{x}])(\mathbf{x} - E[\mathbf{x}])]$) d'un modèle paramétrique $P(\mathbf{x}; \mu, \sigma)$ dont on fait l'hypothèse qu'il aurait généré des données observées $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$. Étant données les observations, nous nous intéressons à des estimateurs $\hat{\mu}_{est}$ et $\hat{\sigma}_{est}$ des paramètres $\hat{\mu}$ et $\hat{\sigma}$.

Un estimateur est non biaisé si sa moyenne sur l'ensemble de tous les jeux de données possibles est égale à la valeur du paramètre : $E[\hat{\mu}_{est}] = \hat{\mu}$, $E[\hat{\sigma}^2_{est}] = \hat{\sigma}^2$.

En plus d'un faible biais, un bon estimateur possède une faible variance : $Var(\hat{\mu}_{est}) = E[(\hat{\mu} - \hat{\mu}_{est})^2]$, $Var(\hat{\sigma}_{est}) = E[(\hat{\sigma} - \hat{\sigma}_{est})^2]$.

19.2 Estimateurs par maximum de vraissemblance

Etant donné un modèle paramétrique et un jeu de données supposé avoir été généré par ce modèle, l'estimateur par *maximum de vraissemblance* ("maximum likelihood") associe à un paramètre la valeur qui rend le jeu de données observé le plus probable.

$$\hat{\mu}_{ML} = \underset{\mu}{\operatorname{argmax}}\, P(\mathbf{x}; \mu, \sigma^2)$$

$\Rightarrow$ {A l'endroit d'un extremum, la dérivée première s'annule}

$$\partial P(\mathbf{x}; \mu, \sigma^2)/\partial\mu = 0$$

$$\hat{\sigma}^2_{ML} = \underset{\sigma^2}{\operatorname{argmax}}\, P(\mathbf{x}; \mu, \sigma^2)$$

$\Rightarrow$ {A l'endroit d'un extremum, la dérivée première s'annule}

$$\partial P(\mathbf{x}; \mu, \sigma^2)/\partial\sigma^2 = 0$$

19.2.1 Exemple d'une loi normale

Supposons que des échantillons scalaires $X = x_1, x_2, \ldots, x_n$ aient été générés selon une loi normale.

$$x_i \sim \mathcal{N}(\mu, \sigma^2)$$
$$P(x_i; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{1}{2\sigma^2}(x_i - \mu)^2\right)$$

$$\begin{aligned}
& P(X; \mu, \sigma) \\
= & \{\text{Hypothèse : les échantillons sont indépendants}\} \\
& \prod_i P(x_i; \mu, \sigma) \\
= & \\
& (2\pi\sigma^2)^{-n/2} exp\left[-\frac{1}{2\sigma^2}\sum_i (x_i - \mu)^2\right]
\end{aligned}$$

Supposons que la moyenne $\hat{\mu}$ du modèle soit connue. Calculons alors l'estimateur par maximum de vraissemblance de la variance.

$$\begin{aligned}
& \hat{\sigma}^2_{ML} \\
= & \{\text{Par définition d'un estimateur par maximum de vraissemblance.}\} \\
& \underset{\sigma^2}{\text{argmax}}\, P(X; \hat{\mu}, \sigma^2) \\
= & \{\text{Le logarithme est une fonction monotone qui ne change pas le lieu du maximum.}\} \\
& \underset{\sigma^2}{\text{argmax}}\, log\left[P(X; \hat{\mu}, \sigma^2)\right] \\
\Rightarrow & \{\text{A l'endroit du max, la dérivée s'annule.}\} \\
& \partial log\left[P(X; \hat{\mu}, \sigma^2)\right] / \partial\sigma^2 = 0 \\
= & \{\text{Notation : } s \triangleq \sigma^2\} \\
& \partial log\left[P(X; \hat{\mu}, s)\right] / \partial s = 0 \\
= & \{\text{Définition de } P\} \\
& -(n/2)\partial log(s)/\partial s - \partial\left[(2s)^{-1}\sum_i (x_i - \hat{\mu})^2\right]\partial s = 0 \\
= & \{\text{Calcul des dérivées.}\} \\
& -(n/2)s^{-1} + 1/2s^{-2}\sum_i (x_i - \hat{\mu})^2 = 0 \\
= & \{\text{Factorisation pour faire apparaître } s\} \\
& (n/2)s^{-2}\left[\left(1/n\sum_i (x_i - \hat{\mu})^2\right) - s\right] = 0 \\
\Rightarrow & \{\text{Pour une variance finie, le second facteur doit s'annuler.}\} \\
& \hat{s}_{ML} = \hat{\sigma}^2_{ML} = 1/n\sum_i (x_i - \hat{\mu})^2
\end{aligned}$$

Quel est le biais de cet estimateur ?

$$
\begin{aligned}
& E\left[\hat{\sigma}^2_{ML}\right] \\
= & \{\text{Voir dérivation ci-dessus.}\} \\
& E\left[n^{-1}\sum_i (x_i - \hat{\mu})^2\right] \\
= & \{\text{Linéarité de l'opérateur espérance E.}\} \\
& n^{-1}\sum_i E\left[x_i^2 - 2x_i\hat{\mu} + \hat{\mu}^2\right] \\
= & \{\text{Linéarité de l'opérateur espérance E.} \\
& \text{Chaque } x_i \text{est supposé avoir été généré par la même loi normale.}\} \\
& n^{-1}\left(nE[x_i^2] - 2n\hat{\mu}E[x_i] + n\hat{\mu}^2\right) \\
= & \{\text{Arithmétique.}\} \\
& E[x_i^2] - 2\hat{\mu}^2 + \hat{\mu}^2 \\
= & \{\hat{\sigma}^2 = E[x_i^2] - E[x_i]^2 = E[x_i^2] - \hat{\mu}^2\} \\
& \hat{\sigma}^2
\end{aligned}
$$

Cet estimateur est sans biais. On peut montrer qu'un estimateur non biaisé obtenu par l'approche du maximum de vraissemblance est également de variance minimale.

Considérons maintenant que la moyenne ne soit pas connue et calculons les estimateurs par

maximum de vraissemblance de la moyenne et de la variance.

$$
\begin{aligned}
& \hat{\mu}_{ML} \\
= & \{\text{Par définition d'un estimateur par maximum de vraissemblance.}\} \\
& \operatorname*{argmax}_{\mu} P(X; \mu, s) \\
\Rightarrow & \{\text{Le logarithme est une fonction monotone qui ne change pas le lieu du maximum.} \\
& \text{A l'endroit du max, la dérivée s'annule.}\} \\
& \partial/\partial\mu \; log \left[(2\pi s)^{-N/2} exp \left(-(2s)^{-1} \sum_i (x_i - \mu)^2 \right) \right] = 0 \\
= & \{\text{Annulation des facteurs qui ne dépendent pas de } \mu.\} \\
& \partial/\partial\mu \left(-(2s)^{-1} \sum_i (x_i - \mu)^2 \right) = 0 \\
= & \{\text{Calcul des dérivées.}\} \\
& -(2s)^{-1} \left(-2 \sum_i (x_i - \mu) \right) = 0 \\
= & \{\text{Arithmétique}\} \\
& s^{-1} \left(\sum_i x_i \; - n\mu \right) = 0 \\
\Rightarrow & \{\text{Arithmétique}\} \\
& \hat{\mu}_{ML} = \frac{1}{n} \sum_i x_i
\end{aligned}
$$

Quel est le biais de cet estimateur ?

$$E[\hat{\mu}_{ML}] = E[1/n \sum_i x_i] = 1/n \sum_i E[x_i] = 1/n \times n \times E[x_i] = \hat{\mu}$$

Il s'agit don d'un estimateur non biaisé.

$$
\begin{aligned}
& \hat{s}_{ML} \\
= & \{\text{Voir la précédente dérivation quand la moyenne était supposée connue.}\} \\
& n^{-1} \sum_i (x_i - \mu)^2 \\
= & \{\text{Utilisation de l'estimateur par maximum de vraissemblance de la moyenne.}\} \\
& n^{-1} \sum_i \left(x_i - n^{-1} \sum_i x_i \right)^2 \\
= & \{\text{Développement.}\} \\
& n^{-1} \sum_i x_i^2 \; - 2n^{-2} \sum_i \left(x_i \sum_i x_i \right) \; + \left(n^{-1} \sum_i x_i \right)^2 \\
= & \{\text{Arithmétique}\} \\
& n^{-1} \sum_i x_i^2 \; - 2 \left(n^{-1} \sum_i x_i \right)^2 \; + \left(n^{-1} \sum_i x_i \right)^2 \\
= & \{\text{Arithmétique}\} \\
& n^{-1} \sum_i x_i^2 \; - \left(n^{-1} \sum_i x_i \right)^2
\end{aligned}
$$

Quel est le biais de cet estimateur de la variance ?

$$
\begin{aligned}
& E[\hat{s}_{ML}] \\
= & \{\text{Voir dérivation ci-dessus.}\} \\
& E\left[n^{-1}\sum_i x_i^2 \; - \left(n^{-1}\sum_i x_i\right)^2\right] \\
= & \{\text{Linéarité de l'opérateur E.}\} \\
& n^{-1}\sum_i E[x_i^2] \; - n^{-2}E\left[\left(\sum_i x_i\right)^2\right] \\
= & \{\hat{s} = E[x_i^2] - E[x_i]^2\} \\
& \hat{s} + \hat{\mu}^2 - n^{-2}E\left[\sum_i x_i^2 \; + \sum_i \sum_{j\neq i} x_i x_j\right] \\
= & \{\text{Linéarité de l'opérateur E.}\} \\
& \hat{s} + \hat{\mu}^2 - n^{-2}\left(n(\hat{s} + \hat{\mu}^2) \; + \sum_i \sum_{j\neq i} E[x_i]E[x_j]\right) \\
= & \{\text{Définition de } \mu.\} \\
& \hat{s} + \hat{\mu}^2 - n^{-2}\left(n(\hat{s} + \hat{\mu}^2) \; + n(n-1)\mu^2\right) \\
= & \{\text{Arithmétique}\} \\
& \hat{s} + \hat{\mu}^2 - n^{-1}\left(\hat{s} + \hat{\mu}^2 \; + n\hat{\mu}^2 - \hat{\mu}^2\right) \\
= & \{\text{Arithmétique}\} \\
& \hat{s} - n^{-1}\hat{s} \\
= & \{\text{Arithmétique}\} \\
& \frac{n-1}{n}\hat{s}
\end{aligned}
$$

L'estimateur de la variance $\hat{s}_{ML}$ est maintenant biaisé. Nous pouvons construire un estimateur sans biais :

$$s' \; = \; \left(\frac{n-1}{n}\right)^{-1} \hat{s}_{ML} \; = \; \frac{1}{n-1}\sum_i (x_i - \mu)^2$$

s' est alors sans biais mais n'est plus de variance minimale (bien que, parmi les estimateurs non biaisés, il soit de variance minimale).

19.3 Analyse biais-variance pour la régression

Nous supposons que les données observées aient été générées par une fonction de la forme $y = f(\mathbf{x}) + \epsilon$ avec ϵ un bruit gaussien de moyenne nulle et de variance σ^2.

A partir d'un jeu de données $\{(\mathbf{x_i}, y_i)\}$, nous apprenons un modèle prédictif, par exemple un modèle linéaire $h(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x} + \beta_0$, afin de minimiser l'erreur quadratique $\sum_i (y_i - h(\mathbf{x_i}))^2$.

Pour un nouveau point $\mathbf{x}^*$ qu'elle est l'espérance de l'erreur commise sur la prédiction de $y^* = f(\mathbf{x}^*) + \epsilon$, soit $E\left[(y^* - h(\mathbf{x}^*))^2\right]$. Il s'agit de la moyenne de l'erreur sur l'ensemble infini de tous les jeux de données d'entraînement possibles.

Notons $\overline{x} = E[x]$, la valeur moyenne de x. Rappelons un résultat utile :

$$
\begin{aligned}
& E[(x - \overline{x})^2] \\
= & \{\text{Arithmétique}\} \\
& E[x^2 - 2x\overline{x} + \overline{x}^2] \\
= & \{\text{Linéarité de } E.\} \\
& E[x^2] - 2\overline{x}E[x] + \overline{x}^2 \\
= & \{\text{Par définition de } \overline{x}.\} \\
& E[x^2] - 2\overline{x}^2 + \overline{x}^2 \\
= & \{\text{Arithmétique}\} \\
& E[x^2] - \overline{x}^2
\end{aligned}
$$

Nous décomposons l'espérance de l'erreur de prédiction ("Expected Prediction Error" ou EPE) d'un modèle de régression en biais, variance et bruit.

$$
\begin{aligned}
& E\left[(h(\mathbf{x}^*) - y^*)^2\right] \\
= & \{\text{Linéarité de E}\} \\
& E[h(\mathbf{x}^*)^2] - 2E[h(\mathbf{x}^*)]E[y^*] + E[y^{*2}] \\
= & \{E[z^2] = E[(z - \overline{z})^2] + \overline{z}^2 \\
& y = f(\mathbf{x}) + \epsilon \text{ avec } \epsilon \text{ un bruit gaussien de moyenne nulle. Donc } \overline{y}^* = f(\mathbf{x}^*)\} \\
& E\left[\left(h(\mathbf{x}^*) - \overline{h(\mathbf{x}^*)}\right)^2\right] + \overline{h(\mathbf{x}^*)}^2 - 2\overline{h(\mathbf{x}^*)}f(\mathbf{x}^*) + E\left[(y^* - f(\mathbf{x}^*))^2\right] + f(\mathbf{x}^*)^2 \\
= & \{E\left[(y^* - f(\mathbf{x}^*))^2\right] = E[\epsilon^2] = \sigma^2\} \\
& E\left[\left(h(\mathbf{x}^*) - \overline{h(\mathbf{x}^*)}\right)^2\right] + \left(\overline{h(\mathbf{x}^*)} - f(\mathbf{x}^*)\right)^2 + \sigma^2 \\
= & \{\text{Introduction des définition de la variance, du biais et du bruit.}\} \\
& \text{Variance} + \text{Biais}^2 + \text{Bruit}^2
\end{aligned}
$$

— La variance mesure la variation de la prédiction $h(\mathbf{x}^*)$ d'un jeu de données d'entraînement à l'autre.
— Le biais mesure l'erreur moyenne de $h(\mathbf{x}^*)$.
— Le bruit mesure la variation de y^* par rapport à $f(\mathbf{x}^*)$.

Chapitre 20

Biais et variance des paramètres d'une régression ridge

20.1 Biais des coefficients d'un modèle de régression ridge

Nous rappelons l'expression de l'estimation des coefficients d'un modèle de régression ridge en fonction de la décomposition en valeurs singulières de la matrice des données.

SVD réduit : $\mathbf{X} = \mathbf{UDV}^T \quad \mathbf{U} \in \mathbb{R}^{n \times r}$, $\mathbf{D} \in \mathbb{R}^{r \times r}$, $\mathbf{V} \in \mathbb{R}^{r \times p}$

$$\hat{\boldsymbol{\beta}}_\lambda = \sum_{d_j > 0} \mathbf{v_j} \frac{d_j}{d_j^2 + \lambda} \mathbf{u_j}^T \mathbf{y} \tag{20.1}$$

Nous supposons que les données observées sont générées par un processus linéaire avec un bruit gaussien.

$$y_i = \mathbf{x_i}^T \boldsymbol{\beta} + \epsilon_i \quad \text{et} \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2) \tag{20.2}$$

ϵ_i représente un bruit gaussien de moyenne nulle et de variance σ^2 supposée être identique pour chaque observation (hypothèse forte, dite d'homoscédasticité, elle s'impose souvent aux modèles de régression) et sans covariances entre observations (hypothèse d'indépendance).

Nous calculons la moyenne des coefficients d'un modèle ridge sur l'ensemble des jeux de données

qui peuvent être générés par ce modèle hypothétique.

$$
\begin{aligned}
& E\left[\hat{\boldsymbol{\beta}}_\lambda\right] \\
= & \{\text{Voir (20.1).}\} \\
& E\left[\sum_{d_j>0} \frac{d_j}{d_j^2+\lambda} \mathbf{v_j} \mathbf{u_j}^T \mathbf{y}\right] \\
= & \{\text{(i) Hypothèse : seule } \mathbf{y} \text{ est une variable aléatoire.} \\
& y_i = \mathbf{x_i}^T \boldsymbol{\beta} + \epsilon_i \;\Rightarrow\; E[\mathbf{y}] = \mathbf{X}\boldsymbol{\beta} \\
& \text{Cette hypothèse n'est pas juste si la valeur de l'hyperparamètre } \lambda \text{ dépend des données.} \\
& \text{Par exemple, si la valeur de } \lambda \text{ est choisie par validation croisée.} \\
& \text{(ii) Linéarité de l'opérateur } E.\} \\
& \sum_{d_j>0} \frac{d_j}{d_j^2+\lambda} \mathbf{v_j} \mathbf{u_j}^T \mathbf{X}\boldsymbol{\beta} \\
= & \{\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T \\
& \mathbf{U} \text{ est orthogonale, } \mathbf{u_j}^T \mathbf{u_j} = 1 \text{ et } \mathbf{u_j}^T \mathbf{u_i} = 0 \text{ pour } i \neq j\} \\
& \sum_{d_j>0} \mathbf{v_j} \frac{d_j}{d_j^2+\lambda} d_j \mathbf{v_j}^T \boldsymbol{\beta}
\end{aligned}
$$

Puisque $\sum \mathbf{v_j}\mathbf{v_j}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}_p$, dans le cas de la régression non régularisée (i.e., $\lambda = 0$), $\hat{\boldsymbol{\beta}}$ est un estimateur sans biais (i.e., $E[\hat{\boldsymbol{\beta}}] = \boldsymbol{\beta}$). Mais pour la régression ridge, l'estimateur $\hat{\boldsymbol{\beta}}_\lambda$ est porteur de biais, ses composantes sont biaisées vers 0. Elles le sont d'autant plus le long des directions de faible variance du jeu de données d'entraînement (c'est-à-dire les direction avec un faible d_j).

20.2 Variance des coefficients d'un modèle de régression ridge

Avant de calculer la variance des coefficients d'un modèle de régression ridge, nous dérivons une forme utile de la variance des coefficients d'un modèle de régression linéaire non régularisé, toujours sous hypothèse de données générées selon un processus linéaire avec bruit gaussien de moyenne nulle (voir équation 20.2).

20.2.1 Variance des coefficients d'un modèle de régression linéaire

Nous utilisons la décomposition en valeurs singulières pour exprimer le modèle de l'équation 20.2 en fonction d'une orthogonalisation de la matrice des données $\mathbf{X}$.

$$
\begin{aligned}
& \mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \\
= & \{\text{SVD} : \mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T\} \\
& \mathbf{y} = \mathbf{U}\mathbf{D}\mathbf{V}^T\boldsymbol{\beta} + \boldsymbol{\epsilon} \\
= & \{\boldsymbol{\alpha} \triangleq \mathbf{D}\mathbf{V}^T\boldsymbol{\beta}\} \\
& \mathbf{y} = \mathbf{U}\boldsymbol{\alpha} + \boldsymbol{\epsilon}
\end{aligned}
$$

Nous calculons l'estimateur $\hat{\boldsymbol{\alpha}}$ du paramètre $\boldsymbol{\alpha}$ au sens des moindres carrés.

$$
\begin{aligned}
& \hat{\boldsymbol{\alpha}} \\
= & \{\text{Estimateur au sens des moindres carrés.}\} \\
& \left(\mathbf{U}^T\mathbf{U}\right)^{-1}\mathbf{U}^T\mathbf{y} \\
= & \{\mathbf{U} \text{ est orthogonale.}\} \\
& \mathbf{U}^T\mathbf{y} \\
= & \{\mathbf{y} = \mathbf{U}\boldsymbol{\alpha} + \boldsymbol{\epsilon}\} \\
& \mathbf{U}^T\left(\mathbf{U}\boldsymbol{\alpha} + \boldsymbol{\epsilon}\right) \\
= & \{\mathbf{U} \text{ est orthogonale.}\} \\
& \boldsymbol{\alpha} + \mathbf{U}^T\boldsymbol{\epsilon}
\end{aligned}
$$

Nous remarquons que $\hat{\boldsymbol{\alpha}}$ est un estimateur sans biais : $E[\hat{\boldsymbol{\alpha}}] = E[\boldsymbol{\alpha}]$. Nous nous y attendions car nous avons déjà montré plus haut que $\boldsymbol{\beta}$ est un estimateur sans biais (or $\boldsymbol{\alpha}$ est la "version de" $\boldsymbol{\beta}$ après orthogonalisation de $\mathbf{X}$).

Calculons la variance de l'estimateur $\hat{\boldsymbol{\alpha}}$.

$$
\begin{aligned}
& Var(\hat{\alpha_j}) \\
= & \{\text{Par définition de la variance.}\} \\
& E\left[(\hat{\alpha_j} - \alpha_j)^2\right] \\
= & \{\text{Nous avons montré ci-dessus : } \hat{\boldsymbol{\alpha}} - \boldsymbol{\alpha} = \mathbf{U}^T\boldsymbol{\epsilon}\} \\
& E\left[\left(\sum_l u_{lj}\epsilon_j\right)^2\right] \\
= & \{\text{Développement du produit.}\} \\
& E\left[\left(\sum_l \sum_t u_{lj}u_{tj}\epsilon_l\epsilon_t\right)^2\right] \\
= & \{\text{Le bruit Gaussien } \boldsymbol{\epsilon} \text{ est de moyenne nulle, sans covariance et de variance } \sigma^2. \\
& E\left[\epsilon_l\epsilon_t\right] = 0 \text{ pour } l \neq t \\
& E\left[\epsilon^2\right] = \sigma^2 \\
& \text{Les } u \text{ ne sont pas des variables aléatoires et, par linéarité de l'espérance, peuvent sortir de sous } E.\} \\
& \left(\sum_l u_{lj}^2\right)\sigma^2 \\
= & \{\text{Les vecteurs } \mathbf{u} \text{ sont unitaires.}\} \\
& \sigma^2
\end{aligned}
$$

Exprimons $\hat{\boldsymbol{\beta}}$ en fonction de $\hat{\boldsymbol{\alpha}}$ pour pouvoir ensuite trouver une expression de la variance $Var\left(\hat{\boldsymbol{\beta}}\right)$.

$$\begin{aligned}
&\hat{\boldsymbol{\alpha}} = \mathbf{D}\mathbf{V}^T\hat{\boldsymbol{\beta}} \\
=\ &\{\text{Hypothèse : } \mathbf{X} \text{ est de plein rang. Alors } \mathbf{V}^T \text{ est de dimension } p \times p \text{ avec } p \text{ le nombre de colonnes de } \mathbf{X}.\} \\
&\hat{\boldsymbol{\beta}} = (\mathbf{V}^T)^{-1}\mathbf{D}^{-1}\hat{\boldsymbol{\alpha}} \\
=\ &\{\mathbf{V} \text{ est orthogonale.}\} \\
&\hat{\boldsymbol{\beta}} = \mathbf{V}\mathbf{D}^{-1}\hat{\boldsymbol{\alpha}}
\end{aligned}$$

D'où :

$$\begin{aligned}
&Var\left(\hat{\boldsymbol{\beta}}\right) \\
=\ &\{\text{Voir ci-dessus.}\} \\
&Var\left(\mathbf{V}\mathbf{D}^{-1}\hat{\boldsymbol{\alpha}}\right) \\
=\ &\{\mathbf{V}\mathbf{D}^{-1} \text{ est orthogonale. Ses colonnes ont deux à deux des covariances nulles.}\} \\
&\left(\mathbf{V}\mathbf{D}^{-1}\right) Var\left(\hat{\boldsymbol{\alpha}}\right) \left(\mathbf{V}\mathbf{D}^{-1}\right)^T \\
=\ &\{Var\left(\hat{\boldsymbol{\alpha}}\right) = \sigma^2\} \\
&\sigma^2 \sum_{j=1}^{P} \frac{1}{d_j^2} \mathbf{v_j} \mathbf{v_j}^T
\end{aligned}$$

20.2.2 Relation linéaire entre $\hat{\boldsymbol{\beta}}_\lambda$ et $\hat{\boldsymbol{\beta}}$

Nous allons découvrir une relation linéaire entre $\hat{\boldsymbol{\beta}}_\lambda$ et $\hat{\boldsymbol{\beta}}$. Elle nous permettra ensuite de calculer $Var\left(\hat{\boldsymbol{\beta}}_\lambda\right)$ sous hypothèse du modèle génératif (20.2).

$$\begin{aligned}
&\hat{\boldsymbol{\beta}} \\
=\ &\{\text{Équation normale d'un modèle linéaire non régularisé.}\} \\
&\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \\
=\ &\{\text{SVD : } \mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T\} \\
&\left(\mathbf{V}\mathbf{D}\mathbf{U}^T\mathbf{U}\mathbf{D}\mathbf{V}^T\right)^{-1}\mathbf{V}\mathbf{D}\mathbf{U}^T\mathbf{y} \\
=\ &\{\mathbf{U} \text{ est orthogonale, donc } \mathbf{U}^T\mathbf{U} = \mathbf{I}\ \mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T\} \\
&\left(\mathbf{V}\mathbf{D}^2\mathbf{V}^T\right)^{-1}\mathbf{V}\mathbf{D}\mathbf{U}^T\mathbf{y} \\
=\ &\{\ (\mathbf{X}\mathbf{Y})^{-1} = \mathbf{X}^{-1}\mathbf{Y}^{-1}, \mathbf{V} \text{ est orthogonale, donc } \mathbf{V}^{-1} = \mathbf{V}^T\ \} \\
&\left(\mathbf{D}^2\mathbf{V}^T\right)^{-1}\mathbf{V}^T\mathbf{V}\mathbf{D}\mathbf{U}^T\mathbf{y} \\
=\ &\{\text{Règles déjà utilisées ci-dessus.}\} \\
&\mathbf{V}\mathbf{D}^{-1}\mathbf{U}^T\mathbf{y} \\
=\ &\{\text{Produit matricielle.}\} \\
&\sum_{d_j>0} \mathbf{v_j} \frac{1}{d_j} \mathbf{u_j}^T\mathbf{y}
\end{aligned}$$

De manière similaire, nous avions déjà calculé dans un précédent module une expression de $\hat{\boldsymbol{\beta}}_\lambda$ en fonction du SVD de $\mathbf{X}$.

$$\hat{\boldsymbol{\beta}}_\lambda = \sum_{d_j>0} \mathbf{v_j} \frac{d_j}{d_j^2+\lambda} \mathbf{u_j}^T \mathbf{y}$$

Ainsi, puisque $\frac{d_j}{d_j^2+\lambda} = \frac{d_j^2}{d_j^2+\lambda}\frac{1}{d_j}$, en notant $\mathbf{W}$ une matrice diagonale dont un j-ème élément sur la diagonale a pour valeur $\frac{d_j^2}{d_j^2+\lambda}$, nous avons découvert une relation linéaire entre $\hat{\boldsymbol{\beta}}_\lambda$ et $\hat{\boldsymbol{\beta}}$:

$$\hat{\boldsymbol{\beta}}_\lambda = \mathbf{W}\hat{\boldsymbol{\beta}}$$

20.2.3 Variance des coefficients d'un modèle de régression linéaire régularisée

Toujours sous hypothèse du modèle génératif (20.2), nous pouvons maintenant calculer une expression de la variance des coefficients d'un modèle de régression linéaire régularisée en fonction de la décomposition en valeurs singulières de la matrice des données.

$$\begin{aligned}
& Var\left(\hat{\boldsymbol{\beta}}_\lambda\right) \\
&= \{\hat{\boldsymbol{\beta}}_\lambda = \mathbf{W}\hat{\boldsymbol{\beta}}, \text{ propriétés de l'opérateur Var.}\} \\
& \mathbf{W} Var\left(\hat{\boldsymbol{\beta}}\right) \mathbf{W}^T \\
&= \{Var\left(\hat{\boldsymbol{\beta}}\right) = \sigma^2 \sum_{j=1}^{P} \frac{1}{d_j^2} \mathbf{v_j}\mathbf{v_j}^T\} \\
& \mathbf{W}\left[\sigma^2 \sum_{d_j>0} \frac{1}{d_j^2} \mathbf{v_j}\mathbf{v_j}^T\right] \mathbf{W}^T \\
&= \{\mathbf{W} \text{ est diagonale, de terme } \frac{d_j^2}{d_j^2+\lambda}\} \\
& \sigma^2 \sum_{d_j>0} \frac{d_j^2}{(d_j^2+\lambda)^2} \mathbf{v_j}\mathbf{v_j}^T
\end{aligned}$$

La variance de l'estimateur $\hat{\boldsymbol{\beta}}_\lambda$ diminue uniformément suivant tous les axes principaux quand le paramètre λ, qui contrôle le degré de régularisation, augmente.

Chapitre 21

Régression ridge à noyau

21.1 Noyau gaussien

Soit un jeu de données :

$$(\mathbf{x_i}, y_i) \quad \text{avec} \quad i = 1, \dots, n \quad ; \quad y_i \in \mathbb{R} \quad ; \quad \mathbf{x_i} \in \mathbb{R}^d$$

Nous appelons *noyau* une mesure de similarité entre points :

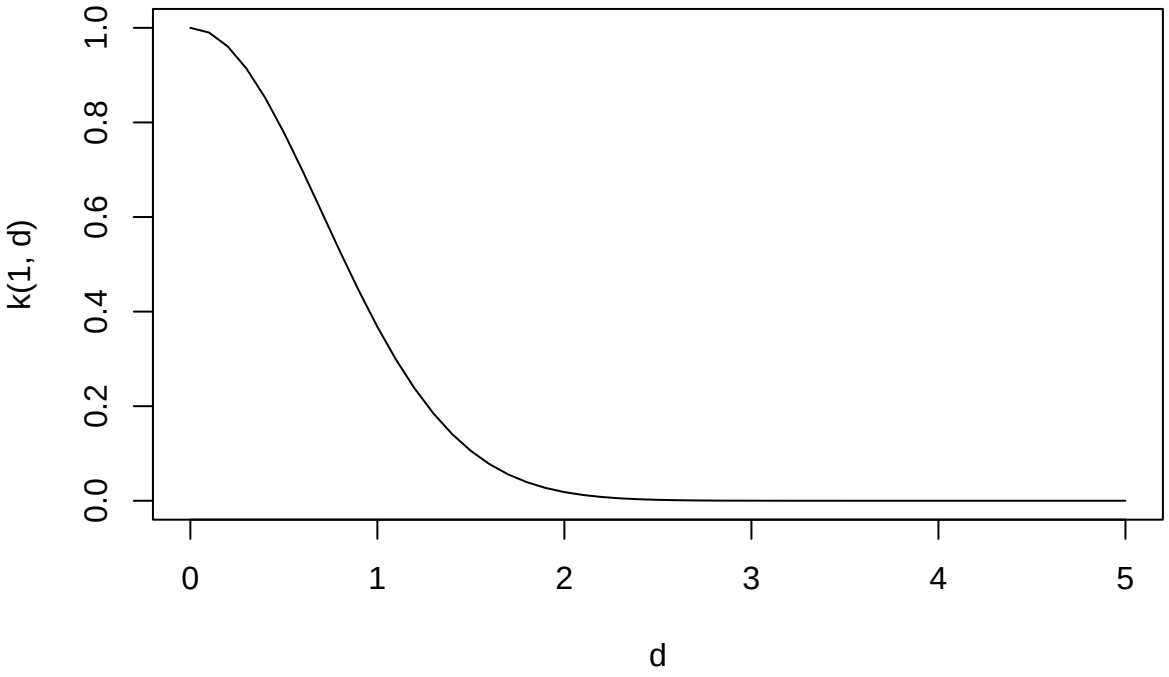
$$k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

La similarité peut, par exemple, être représentée par une fonction gaussienne :

$$k(\mathbf{x_j}, \mathbf{x_i}) = exp\left(-\frac{1}{\sigma^2}\|\mathbf{x_j} - \mathbf{x_i}\|^2\right)$$

Avec $\|\mathbf{x_j} - \mathbf{x_i}\|$ la distance euclidienne entre observations. Nous traçons ci-dessous la forme de cette décroissance exponentielle. Lorsque la distance entre $\mathbf{x_j}$ et $\mathbf{x_i}$ est nulle, la similarité est maximale et vaut 1.

```
d <- seq(from=0, to=5, by=0.1)
k <- function(sigma,d){exp(-(1/sigma^2)*d^2)}
plot(x=d, y=k(1,d), type="l")
```

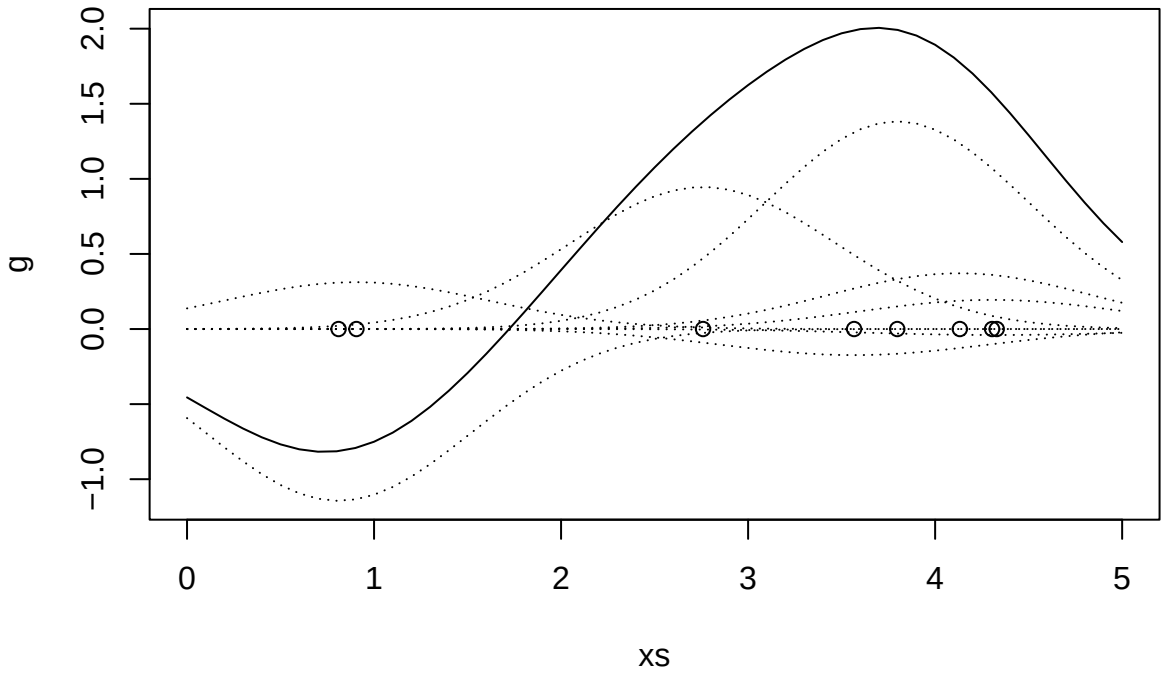


Nous proposons ensuite de représenter la relation entre les observations et la cible par une combinaison linéaire des similarités d'une nouvelle observation $\mathbf{x}$ avec chaque observation du jeu d'entraînement :

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x_i}) \tag{21.1}$$

Plus $\mathbf{x}$ est proche de $\mathbf{x_i}$, plus $\mathbf{x_i}$ pèse dans le calcul de la valeur prédite pour $\mathbf{x}$. Chaque $k(\cdot, \mathbf{x_i})$ est une fonction gaussienne et f est une superposition de fonctions gaussiennes. Nous proposons une illustration d'une telle superposition.

```
set.seed(1123)
npts <- 8
xi <- runif(npts, min=0, max=5)
ci <- rnorm(npts)
xs <- seq(from=0, to=5, by=0.1)
fi <- function(ci,xi) function(x) ci * exp(-(x-xi)^2)
call_f <- function(f,...) f(...)
m <- t(matrix(unlist(lapply(Map(fi,ci,xi), call_f, xs)), nrow=npts, byrow=TRUE))
g <- rowSums(m)
plot(xs, g, type="l", lty="solid", ylim=range(cbind(g,m)))
matplot(xs, m, type="l", lty="dotted", col=1, add=TRUE)
points(x=xi, y=rep(0,npts))
```



Cette approche est qualifiée de *non paramétrique* car elle s'adapte aux données au lieu de chercher à adpater les paramètres d'une forme fonctionnelle fixée a priori comme nous le faisions auparavant dans le cas de la régression régularisée.

21.2 Régression ridge à noyau

Nous montrons que l'approche non paramétrique introduite ci-dessus peut se présenter sous la forme d'une régression ridge après transformation des variables $\mathbf{x_i}$ par une fonction ϕ.

$$\mathbf{X} \in \mathbb{R}^{n \times d}$$
$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \quad \text{Régression normale}$$
$$\hat{\boldsymbol{\beta}}_\lambda = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_d\right)^{-1}\mathbf{X}^T\mathbf{y} \quad \text{Régression ridge}$$
$$\phi : \mathbb{R}^d \to \mathbb{R}^k \quad \text{avec } 1 \leq k \leq +\infty$$
$$\mathbf{\Phi} = [\phi(\mathbf{x_1}), \ldots, \phi(\mathbf{x_n})] \in \mathbb{R}^{n \times k}$$

Nous introduisons une variante de *l'identité de Woodbury* qui sera utile au calcul des coefficients de

la régression ridge après application de ϕ.

$$
\begin{aligned}
& \text{Identité de Woodbury} \\
= & \{\text{https://en.wikipedia.org/wiki/Woodbury_matrix_identity}\} \\
& (\mathbf{I} + \mathbf{UV})^{-1}\mathbf{U} = \mathbf{U}(\mathbf{I} + \mathbf{VU})^{-1} \\
= & \{\text{Algèbre}\} \\
& \frac{\lambda}{\lambda}(\mathbf{I} + \mathbf{UV})^{-1}\mathbf{U} = \frac{\lambda}{\lambda}\mathbf{U}(\mathbf{I} + \mathbf{VU})^{-1} \\
= & \{ (\lambda\mathbf{X})^{-1} = \lambda^{-1}\mathbf{X}^{-1}\} \\
& (\lambda\mathbf{I} + \lambda\mathbf{UV})^{-1}\lambda\mathbf{U} = \lambda\mathbf{U}(\lambda\mathbf{I} + \mathbf{V}\lambda\mathbf{U})^{-1} \\
= & \{\mathbf{W} = \lambda\mathbf{U}\} \\
& (\lambda\mathbf{I} + \mathbf{WV})^{-1}\mathbf{W} = \mathbf{W}(\lambda\mathbf{I} + \mathbf{VW})^{-1} \\
= & \{\mathbf{W} = \mathbf{\Phi}^T \quad ; \quad \mathbf{V} = \mathbf{\Phi}\} \\
& (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I_k})^{-1}\mathbf{\Phi}^T = \mathbf{\Phi}^T(\mathbf{\Phi}\mathbf{\Phi}^T + \lambda\mathbf{I_n})^{-1}
\end{aligned}
$$

Nous calculons maintenant les coefficients d'une régression ridge après application de la transformation ϕ.

$$
\begin{aligned}
& \hat{\boldsymbol{\beta}}_\lambda = \left(\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I_k}\right)^{-1}\mathbf{\Phi}^T\mathbf{y} \\
= & \{\text{Identité de Woodbury, voir ci-dessus.}\} \\
& \hat{\boldsymbol{\beta}}_\lambda = \mathbf{\Phi}^T\left(\mathbf{\Phi}\mathbf{\Phi}^T + \lambda\mathbf{I_n}\right)^{-1}\mathbf{y} \\
= & \{\alpha = \left(\mathbf{\Phi}\mathbf{\Phi}^T + \lambda\mathbf{I_n}\right)^{-1}\mathbf{y}\} \\
& \hat{\boldsymbol{\beta}}_\lambda = \sum_{i=1}^{n}\alpha_i\phi(\mathbf{x_i})
\end{aligned}
\tag{21.2}
$$

Nous calculons la prédiction $\hat{y}$ que ce modèle associe à une observation $\mathbf{x}$.

$$
\begin{aligned}
& \hat{y} = \phi(\mathbf{x})^T\hat{\boldsymbol{\beta}}_\lambda \\
= & \{\text{Voir (21.2)}\} \\
& \hat{y} = \sum_{i=1}^{n}\alpha_i\phi(\mathbf{x})^T\phi(\mathbf{x_i}) \\
= & \{k(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x})^T\phi(\mathbf{y})\} \\
& \hat{y} = \sum_{i=1}^{n}\alpha_i k(\mathbf{x},\mathbf{x_i})
\end{aligned}
$$

Nous retrouvons l'équation (21.1). Nous pouvons écrire ce résultat sous forme matricielle en introduisant la matrice noyau $\mathbf{K}$.

$$
\begin{aligned}
& \hat{y} = \sum_{i=1}^{n}\alpha_i k(\mathbf{x},\mathbf{x_i}) \\
= & \{K_{ij} = \phi(\mathbf{x_i})^T\phi(\mathbf{x_j}) \quad ; \quad \mathbf{k}(\mathbf{x}) = [k(\mathbf{x},\mathbf{x_1}),\ldots,k(\mathbf{x},\mathbf{x_n})]\} \\
& \hat{y} = \mathbf{k}(\mathbf{x})^T\left(\mathbf{K} + \lambda\mathbf{I_n}\right)^{-1}\mathbf{y}
\end{aligned}
$$

Ainsi, l'approche non paramétrique par juxtaposition des similarités d'une observation avec chaque observation du jeu d'entraînement correspond à une régression ridge après application de la transformation ϕ aux observations. Nous remarquons qu'il n'est pas nécessaire d'opérer explicitement cette transformation, seules sont nécessaires les valeurs des produits scalaires $\phi(\mathbf{x_i})^T\phi(\mathbf{x_j})$. En fait, l'application explicite de la transformation ϕ serait souvent impossible. Par exemple, dans le cas d'un noyau gaussien, ϕ projette vers un espace de dimension infinie...

21.3 Calcul des paramètres de la régression ridge à noyau

Notons $\mathbf{G} \triangleq \mathbf{K} + \lambda\mathbf{I_n}$. Nous venons de montrer :

$$\boldsymbol{\alpha}_\lambda = \mathbf{G}^{-1}\mathbf{y} \tag{21.3}$$

$$\hat{\mathbf{y}} = \mathbf{K}\mathbf{G}^{-1}\mathbf{y} \tag{21.4}$$

Nous calculons une expression de $\boldsymbol{\alpha}_\lambda$ en fonction de la décomposition en valeurs propres $\mathbf{K} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T$.

$$\begin{aligned}
&\mathbf{G}\\
=\ &\{\text{Par définition.}\}\\
&\mathbf{K} + \lambda\mathbf{I_n}\\
=\ &\{\mathbf{K} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T\}\\
&\mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T + \lambda\mathbf{I_n}\\
=\ &\{\text{Algèbre linéaire.}\}\\
&\mathbf{Q}\left(\boldsymbol{\Lambda} + \lambda\mathbf{I_n}\right)\mathbf{Q}^T
\end{aligned}$$

D'où :

$$\begin{aligned}
&\mathbf{G}^{-1}\\
=\ &\{\mathbf{G} = \mathbf{Q}\left(\boldsymbol{\Lambda} + \lambda\mathbf{I_n}\right)\mathbf{Q}^T\\
&\mathbf{Q}^{-1} = \mathbf{Q}^T \text{ car } \mathbf{Q} \text{ est orthogonale.}\}\\
&\mathbf{Q}\left(\boldsymbol{\Lambda} + \lambda\mathbf{I_n}\right)^{-1}\mathbf{Q}^T
\end{aligned}$$

Enfin :

$$\boldsymbol{\alpha}_\lambda = \mathbf{Q}\left(\boldsymbol{\Lambda} + \lambda\mathbf{I_n}\right)^{-1}\mathbf{Q}^T\mathbf{y}$$

Ainsi, après avoir effectuée la décomposition en valeurs propres de $\mathbf{K}$ en $\mathcal{O}(n^3)$, le calcul de $\boldsymbol{\alpha}_\lambda$ pour une valeur λ donnée se fait en $\mathcal{O}(n^2)$.

21.4 LOOCV pour le choix de l'hyper-paramètre λ

Pour la régression ridge, nous avons découvert une forme de la validation croisée un-contre-tous (LOOCV) qui peut être calculée à partir d'un seul calcul des paramètres sur tout le jeu d'entraînement :

$$LOO_\lambda = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y_{\lambda_i}}}{1 - h_i}\right)^2$$

Avec h_i le i-ème élément sur la diagonale de la matrice chapeau.

Le même raisonnement nous mène à découvrir une expression similaire pour la régression ridge à noyau :

$$LOO_\lambda = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - (\mathbf{K}\mathbf{G}^{-1}\mathbf{y})_i}{1-(\mathbf{K}\mathbf{G}^{-1})_{ii}}\right)^2$$

Nous simplifions l'expression $\mathbf{K}\mathbf{G}^{-1}$ en utilisant la décomposition en éléments propres de $\mathbf{K}$.

$$\begin{aligned}
&\mathbf{K}\mathbf{G}^{-1}\\
=\ &\{\text{Décomposition en éléments propres de } \mathbf{K}\}\\
&\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T\mathbf{Q}\left(\mathbf{\Lambda}+\lambda\mathbf{I_n}\right)^{-1}\mathbf{Q}^T\\
=\ &\{\mathbf{Q} \text{ est orthogonale, donc } \mathbf{Q}^T\mathbf{Q}=\mathbf{I}\}\\
&\mathbf{Q}\mathbf{\Lambda}\left(\mathbf{\Lambda}+\lambda\mathbf{I_n}\right)^{-1}\mathbf{Q}^T\\
=\ &\{\text{Astuce arithmétique pour faire apparaître une simplification.}\}\\
&\mathbf{Q}\left(\mathbf{\Lambda}+\lambda\mathbf{I_n}-\lambda\mathbf{I_n}\right)\left(\mathbf{\Lambda}+\lambda\mathbf{I_n}\right)^{-1}\mathbf{Q}^T\\
=\ &\{(\mathbf{\Lambda}+\lambda\mathbf{I_n}-\lambda\mathbf{I_n}) \text{ et } (\mathbf{\Lambda}+\lambda\mathbf{I_n})^{-1} \text{ sont des matrices diagonales.}\\
&+ \text{ et } \times \text{ sont des opérateurs commutatifs et associatifs pour les matrices diagonales.}\\
&\mathbf{G}^{-1}=\mathbf{Q}\left(\mathbf{\Lambda}+\lambda\mathbf{I_n}\right)^{-1}\mathbf{Q}^T\}\\
&\mathbf{I_n}-\lambda\mathbf{G}^{-1}
\end{aligned}$$

Nous simplifions le numérateur de LOO_λ :

$$\begin{aligned}
&y_i - \left(\mathbf{K}\mathbf{G}^{-1}\mathbf{y}\right)_i\\
=\ &\{\mathbf{K}\mathbf{G}^{-1}=\mathbf{I_n}-\lambda\mathbf{G}^{-1}\}\\
&\left(\lambda\mathbf{G}^{-1}\mathbf{y}\right)_i
\end{aligned}$$

Nous simplifions le dénominateur de LOO_λ :

$$\begin{aligned}
&1 - \left(\mathbf{K}\mathbf{G}^{-1}\right)_{ii}\\
=\ &\{\mathbf{K}\mathbf{G}^{-1}=\mathbf{I_n}-\lambda\mathbf{G}^{-1}\}\\
&\left(\lambda\mathbf{G}^{-1}\right)_{ii}
\end{aligned}$$

Nous obtenons une nouvelle expression simplifiée de LOO_λ :

$$LOO_\lambda = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\alpha_i}{(\mathbf{G}^{-1})_{ii}}\right)^2$$

Montrons comment calculer efficacement $(\mathbf{G}^{-1})_{ii}$ en calculant l'expression d'un élément de la

matrice $\mathbf{G}^{-1}$:

$$
\begin{aligned}
& \left(\mathbf{G}^{-1}\right)_{ij} \\
= & \{\text{Définition de } \mathbf{G}^{-1} \text{ en fonction de la décomposition en éléments propres de } \mathbf{K}.\} \\
& \left(\mathbf{Q}\left(\mathbf{\Lambda}+\lambda\mathbf{I_n}\right)^{-1}\mathbf{Q}^T\right)_{ij} \\
= & \{\text{Définition du produit matricielle.}\} \\
& \sum_{k=1}^{n}\frac{Q_{ik}Q_{jk}}{\Lambda_{kk}+\lambda}
\end{aligned}
$$

Nous pouvons donc calculer $\left(\mathbf{G}^{-1}\right)_{ii}$ en $\mathcal{O}(n)$ et LOO_λ en $\mathcal{O}(n^2)$.

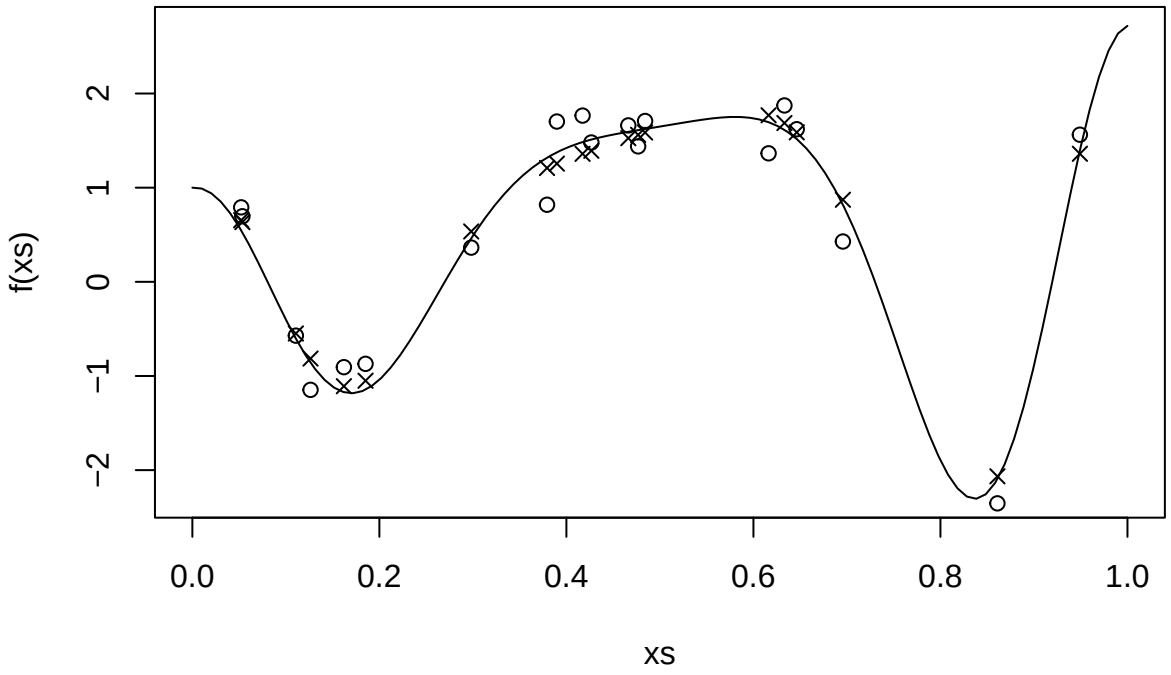
21.5 Choix de l'hyper-paramètre σ^2

Il est possible de montrer qu'une fois les données standardisées (i.e., moyenne nulle et écart-type unité), la moyenne de la distance euclidienne entre deux points est égale à deux fois la dimension de l'espace des observations : $E\left[\|\mathbf{x_j}-\mathbf{x_i}\|^2\right]=2d$. Ainsi, $\sigma^2=d$ est un choix raisonnable pour permettre au noyau gaussien de bien capturer les similarités entre points (i.e., les points les plus proches auront une similarité proche de 1, tandis que les points les plus éloignés auront une similarité proche de 0).

21.6 Exemple sur un jeu de données synthétique

Nous reprenons le jeu de données synthétique utilisé depuis le premier module pour tester l'implémentation de la régression ridge à noyau gaussien.

```
set.seed(1123)
n <- 100
data = gendat(n,0.2)
splitres <- splitdata(data,0.8)
entr <- splitres$entr
test <- splitres$test

km <- krr(entr$X,entr$Y)
yh <- predict(km,test$X)
plt(test,f)
points(test$X, yh, pch=4)
```




```
  testmae <- mean(abs(yh-test$Y))
```

Ce modèle atteint une erreur absolue moyenne de 0.2187576 sur le jeu de test.

21.7 Annexe code source

```
gausskernel <-
function(X, sigma2)
{
  return(exp(-1*as.matrix(dist(X)^2)/sigma2))
}

# For X a square matrix, efficient impl of X %*% diag(d)
multdiag <-
function(X,d)
{
  R <- matrix(NA, nrow=dim(X)[1], ncol=dim(X)[2])
  for (i in 1:dim(X)[2]) { R[,i]=X[,i]*d[i] }
  return(R)
}

krr <-
function(X, y, sigma2=NULL, lambdas=NULL)
{
  X <- as.matrix(X)
  n <- nrow(X)
  p <- ncol(X)
```

```
  if(is.null(lambdas)) { lambdas <- 10^seq(-8, 2,by=0.5) }
  if(is.null(sigma2)) { sigma2 <- p }

  X <- scale(X)
  y <- scale(y)

  K <- gausskernel(X, sigma2=sigma2)
  eig <- eigen(K, symmetric=TRUE)

  qty <- matrix(NA,n,n)
  qty <- crossprod(eig$vectors, y)

  looe <- double(length(lambdas))
  coef <- matrix(data = NA, nrow = n, ncol = length(lambdas))
  i <- 1
  for(lambda in lambdas) {
    diag <- 1/(eig$values + lambda)
    qdiag <- multdiag(eig$vectors, diag)
    coef[,i] <- qdiag %*% qty
    ginvdiag <- rowSums(multdiag(eig$vectors^2, diag))
    looe[i] <- mean((coef[,i]/ginvdiag)^2)
    i <- i+1
  }
  looe.min <- min(looe)
  lambda <- lambdas[which(looe == looe.min)]
  coef <- coef[,which(looe == looe.min)]
  yh <- K%*%coef
  yh <- yh * attr(y,"scaled:scale") + attr(y,"scaled:center")

  r <- list(K=K,
            X=X,
            y=y,
            sigma2=sigma2,
            coef=coef,
            looe=looe.min,
            lambda=lambda,
            yh=yh
           )
  class(r) <- "krr"
  return(r)
}

predict.krr <-
function(o, newdata)
{
  if(class(o) != "krr") {
```

```
    warning("Object is not of class 'krr'")
    UseMethod("predict")
    return(invisible(NULL))
  }
  newdata <- as.matrix(newdata)
  if(ncol(o$X)!=ncol(newdata)) {
    stop("Not the same number of variables btwn fitted krr object and new data")
  }
  newdata <- scale(newdata,center=attr(o$X,"scaled:center"),
                   scale=attr(o$X,"scaled:scale"))
  n <- nrow(o$X)
  nn <- nrow(newdata)
  K <- gausskernel(rbind(newdata,o$X),sigma2=o$sigma2)[1:nn,(nn+1):(nn+n)]
  K <- matrix(K,nrow=nn,byrow=FALSE)
  yh <- K%*%o$coef
  yh <- (yh * attr(o$y,"scaled:scale")) + attr(o$y,"scaled:center")
}


# Tests

test.multdiag <-
function()
{
  A <- matrix(seq(from=1, to=5*5, by=1), nrow=5)
  d <- seq(from=1, to=5, by=1)
  B <- multdiag(A,d)
  C <- A %*% diag(d)
  identical(B,C)
}
```

Chapitre 22

Approximation de Nyström

22.1 Approximation de Nyström d'un noyau

Soit un jeu de données.

$$(\mathbf{x_i}, y_i) \quad \text{avec} \quad i = 1, \ldots, n \quad ; \quad y_i \in \mathbb{R} \quad ; \quad \mathbf{x_i} \in \mathbb{R}^d$$

Soit un noyau $\mathbf{K}$ (par ex. gaussien).

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x_1}, \mathbf{x_1}) & \ldots & k(\mathbf{x_1}, \mathbf{x_n}) \\ \ldots & \ldots & \ldots \\ k(\mathbf{x_n}, \mathbf{x_1}) & \ldots & k(\mathbf{x_n}, \mathbf{x_n}) \end{bmatrix}$$

Soit une approximation de rang m de $\mathbf{K}$ (exprimée en fonction de la décomposition en valeurs propres).

$$\mathbf{K} \approx \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T \quad ; \quad \mathbf{U} \in \mathbb{R}^{n \times m} \quad ; \quad \mathbf{\Lambda} \in \mathbb{R}^{m \times m}$$

Lorsque le noyau original est utilisé, chacune des n observations du jeu de données d'apprentissage sert de centre à partir duquel mesurer la proximité d'une nouvelle observation. Le principe de l'approximation de rang m, selon l'approche dite de Nyström, est de ne conserver que m des n observations comme centres à partir desquels mesurer la proximité de nouvelles observations. Nous sélectionnons ainsi m observations (m lignes de $\mathbf{X}$). Plusieurs approches sont possibles (par ex., de façon aléatoire, par l'algorithme des k-médoïdes, etc.).

$$\mathbf{K} = \begin{bmatrix} \mathbf{K_{11}} & \mathbf{K_{21}^T} \\ \mathbf{K_{21}} & \mathbf{K_{22}} \end{bmatrix} \quad ; \quad \mathbf{K_{11}} \in \mathbb{R}^{m \times m} \quad ; \quad \mathbf{K_{21}} \in \mathbb{R}^{(n-m) \times m}$$

L'approximation de Nyström construit une approximation de rang m de $\mathbf{K}$ sans évaluer $\mathbf{K_{22}}$. Commençons par exprimer l'approximation de rang m sous forme de matrices blocs.

$$\mathbf{K} \approx \begin{bmatrix} \mathbf{U_1} \\ \mathbf{U_2} \end{bmatrix} \mathbf{\Lambda} \begin{bmatrix} \mathbf{U_1} \\ \mathbf{U_2} \end{bmatrix}^T = \begin{bmatrix} \mathbf{U_1}\mathbf{\Lambda}\mathbf{U_1}^T & \mathbf{U_1}\mathbf{\Lambda}\mathbf{U_2}^T \\ \mathbf{U_2}\mathbf{\Lambda}\mathbf{U_1}^T & \mathbf{U_2}\mathbf{\Lambda}\mathbf{U_2}^T \end{bmatrix} \quad ; \quad \mathbf{U_1} \in \mathbb{R}^{m \times m} \quad ; \quad \mathbf{U_2} \in \mathbb{R}^{(n-m) \times m}$$

Nous remarquons que $\mathbf{K_{11}} \approx \mathbf{U_1 \Lambda {U_1}}^T$ et $\mathbf{K_{21}} \approx \mathbf{U_2 \Lambda {U_1}}^T$.

Nous souhaitons éviter d'avoir à évaluer $\mathbf{U_2}$ dans l'approximation de $\mathbf{K_{22}}$. Nous cherchons donc une expression de $\mathbf{U_2}$ en fonction d'autres termes.

$$
\begin{aligned}
& \mathbf{K_{21}} \approx \mathbf{U_2 \Lambda {U_1}}^T \\
= & \{\text{Multiplication des deux membres de l'égalité par un même terme non nul.}\} \\
& \mathbf{K_{21}} \left(\mathbf{\Lambda {U_1}}^T\right)^{-1} \approx \mathbf{U_2 \Lambda {U_1}}^T \left(\mathbf{\Lambda {U_1}}^T\right)^{-1} \\
= & \{\text{Algèbre linéaire. } \mathbf{U_1} \text{ est orthogonale donc } \mathbf{{U_1}}^{-1} = \mathbf{{U_1}}^T\} \\
& \mathbf{K_{21} U_1 \Lambda^{-1}} \approx \mathbf{U_2}
\end{aligned}
$$

Nous pouvons maintenant découvrir une approximation de $\mathbf{K_{22}}$ en fonction de celles de $\mathbf{K_{21}}$ et $\mathbf{K_{11}}$.

$$
\begin{aligned}
& \mathbf{K_{22}} \\
\approx & \{\text{Définition de la décomposition en valeurs propres pour l'approximation de rang } m \text{ de } \mathbf{K}\} \\
& \mathbf{U_2 \Lambda {U_2}}^T \\
= & \{\text{Expression de } \mathbf{U_2} \text{ découverte ci-dessus.}\} \\
& \left(\mathbf{K_{21} U_1 \Lambda^{-1}}\right) \mathbf{\Lambda} \left(\mathbf{K_{21} U_1 \Lambda^{-1}}\right)^T \\
= & \{\text{Algèbre linéaire.}\} \\
& \mathbf{K_{21} U_1 \Lambda^{-1} {U_1}}^T \mathbf{{K_{21}}}^T \\
= & \{\text{Définition de } \mathbf{K_{11}} \text{ dans la décomposition en valeurs propres pour l'approximation de rang } m \text{ de } \mathbf{K}\} \\
& \mathbf{K_{21} {K_{11}}^{-1} {K_{21}}}^T \\
= & \{\text{Algèbre linéaire.}\} \\
& \left(\mathbf{K_{21} {K_{11}}^{-1/2}}\right) \left(\mathbf{K_{21} {K_{11}}^{-1/2}}\right)^T
\end{aligned}
$$

Ainsi, nous pouvons introduire une matrice $\mathbf{\Phi}$ de dimension m telle que $\mathbf{K} \approx \mathbf{\Phi \Phi}^T$.

$$
\mathbf{\Phi} = \begin{bmatrix} \mathbf{{K_{11}}^{1/2}} \\ \mathbf{K_{21} {K_{11}}^{-1/2}} \end{bmatrix}
$$

22.2 Calcul de l'approximation de Nyström

Notons $\mathbf{C}$ la sélection des m colonnes de $\mathbf{K}$.

$$
\mathbf{C} = \begin{bmatrix} \mathbf{K_{11}} \\ \mathbf{K_{21}} \end{bmatrix}
$$

Nous avons :

$$
\mathbf{C {K_{11}}^{-1} C}^T = \begin{bmatrix} \mathbf{I} \\ \mathbf{K_{21} {K_{11}}^{-1}} \end{bmatrix} \begin{bmatrix} \mathbf{{K_{11}}}^T & \mathbf{{K_{21}}}^T \end{bmatrix} = \begin{bmatrix} \mathbf{{K_{11}}}^T & \mathbf{{K_{21}}}^T \\ \mathbf{K_{21}} & \mathbf{K_{21} {K_{11}}^{-1} {K_{21}}}^T \end{bmatrix} \approx \mathbf{K}
$$

Nous pouvons donc calculer une approximation de rang m de $\mathbf{K} \in \mathbb{R}^{n \times n}$ en ne calculant que $\mathbf{C} \in \mathbb{R}^{n \times m}$ (c'est-à-dire, m colonnes de $\mathbf{K}$) et la décomposition en valeurs propres de $\mathbf{K_{11}} \in \mathbb{R}^{m \times m}$.

$$
\begin{aligned}
& \mathbf{K} \approx \mathbf{C}{\mathbf{K_{11}}}^{-1}\mathbf{C}^T \\
= & \{\mathbf{K_{11}} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^T\} \\
& \mathbf{K} \approx \mathbf{C}\mathbf{U}\mathbf{\Sigma}^{-1}\mathbf{U}^T\mathbf{C}^T \\
= & \{\mathbf{L} \triangleq \mathbf{C}\mathbf{U}\mathbf{\Sigma}^{-1/2}\} \\
& \mathbf{K} \approx \mathbf{L}\mathbf{L}^T
\end{aligned}
$$

22.3 Régression ridge à noyau et approximation de Nyström

Dans le cadre de la régression ridge à noyau (voir un précédent module), nous notons : $\mathbf{G} = \mathbf{K} + \lambda\mathbf{I_n}$. Les coefficients du modèle ridge sont alors donnés par : $\boldsymbol{\alpha}_\lambda = \mathbf{G}^{-1}\mathbf{y}$. Nous cherchons à calculer efficacement $\mathbf{G}^{-1}$ à partir d'une approximation Nyström de rang m de $\mathbf{K} \approx \mathbf{L}\mathbf{L}^T$. Pour ce faire, nous utilisons une forme de l'identité de Woodbury :

$$(\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{V})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{C}^{-1} + \mathbf{V}\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}\mathbf{A}^{-1}$$

Nous obtenons ainsi :

$$
\begin{aligned}
& \mathbf{G}^{-1} \\
= & \{\text{Définition de } \mathbf{G}\} \\
& (\lambda\mathbf{I_n} + \mathbf{K})^{-1} \\
\approx & \{\text{Approximation de Nyström}\} \\
& \left(\lambda\mathbf{I_n} + \mathbf{L}\mathbf{L}^T\right)^{-1} \\
= & \{\text{Identité de Woodbury}\} \\
& \lambda^{-1}\mathbf{I_n} - \lambda^{-1}\mathbf{L}\left(\mathbf{I_m} + \lambda^{-1}\mathbf{L}^T\mathbf{L}\right)^{-1}\lambda^{-1}\mathbf{L}^T \\
= & \{\text{Algèbre linéaire : si } \mathbf{A} \text{ et } \mathbf{B} \text{ sont des matrices carrés inversibles alors } (\mathbf{A}\mathbf{B})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}\} \\
& \lambda^{-1}\mathbf{I_n} - \lambda^{-1}\mathbf{L}\left(\lambda\mathbf{I_m} + \mathbf{L}^T\mathbf{L}\right)^{-1}\mathbf{L}^T
\end{aligned}
$$

Chapitre 23

TP - Régression ridge et dilemme biais-variance - Sujet

```
set.seed(1123)
```

L'idée de cette expérimentation provient de la référence Hastie (2020).

23.1 Générer un jeu de données synthétique

Générer un jeu de données simulé à partir d'un modèle linéaire :

$$y_i = \mathbf{x_i}^T \boldsymbol{\beta} + \epsilon_i, i = 1, \ldots, n \quad \mathbf{x_i} \in \mathcal{R}^p \quad ; \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

```
n <- 70
p <- 55
```

Utiliser $n = 70$ et $p = 55$. Les $\mathbf{x_i}$ sont indépendants et suivent, par exemple, une distribution uniforme entre 0 et 1 (voir la fonction `runif`). Faire en sorte que les colonnes de $\mathbf{X}$ soient de moyenne nulle et de variance unité (voir la fonction `scale`). Cela simplifie les calculs ultérieurs sans perte de généralité.

23.2 Calculer les coefficients d'une régression ridge

Tracer l'évolution des valeurs des coefficients d'un modèle ridge pour différentes valeurs de l'hyper-paramètre de régularisation λ. Réutiliser le code introduit au chapitre 15 sur la validation croisée un-contre-tous (voir le fichier `15_loocv.R`).

Dans cette implémentation, les étiquettes $\mathbf{y}$ sont centrées avant d'opérer la régression ridge. Dans le contexte de cette expérimentation, il peut être intéressant de proposer une version pour laquelle les étiquettes ne sont pas centrées. Quelle est la différence ? La réponse est liée à la présentation de la standardisation au chapitre 3 sur la régularisation de Tikhonov.

23.3 Espérance de l'erreur de prédiction

Pour les différentes valeurs de λ, calculer l'espérance de l'erreur de prédiction (c'est ici possible car les données sont simulées et nous connaissons le modèle qui les a générées). Cette notion a été introduite à la fin du chapitre 16. Quelle est la valeur de λ qui minimise une estimation de l'espérance de l'erreur de prédiction ? Comment se compare-t-elle à la valeur trouvée par validation croisée un-contre-tous ? Étudier les effets du nombre d'observations, du nombre de variables et de la quantité de bruit.

Chapitre 24

TP - Régression ridge et dilemme biais-variance - Correction

```
set.seed(1123)
```

L'idée de cette expérimentation provient de la référence Hastie (2020).

24.1 Générer un jeu de données synthétique

Générer un jeu de données simulé à partir d'un modèle linéaire :

$$y_i = \mathbf{x_i}^T \boldsymbol{\beta} + \epsilon_i, i = 1, \ldots, n \quad \mathbf{x_i} \in \mathcal{R}^p \quad ; \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

```
n <- 70
p <- 55
```

Utiliser $n = 70$ et $p = 55$. Les $\mathbf{x_i}$ sont indépendants et suivent, par exemple, une distribution uniforme entre 0 et 1 (voir la fonction `runif`). Faire en sorte que les colonnes de $\mathbf{X}$ soient de moyenne nulle et de variance unité (voir la fonction `scale`). Cela simplifie les calculs ultérieurs sans perte de généralité.

```
sig2 <- 6 # standard deviation for zero-mean gaussian noise
X <- matrix(runif(n*p),nrow=n,ncol=p)
X <- scale(X)
beta <- runif(p, min=-10, max=10)
y <- X%*%beta + rnorm(n, mean=0, sd=sig2)
```

24.2 Calculer les coefficients d'une régression ridge

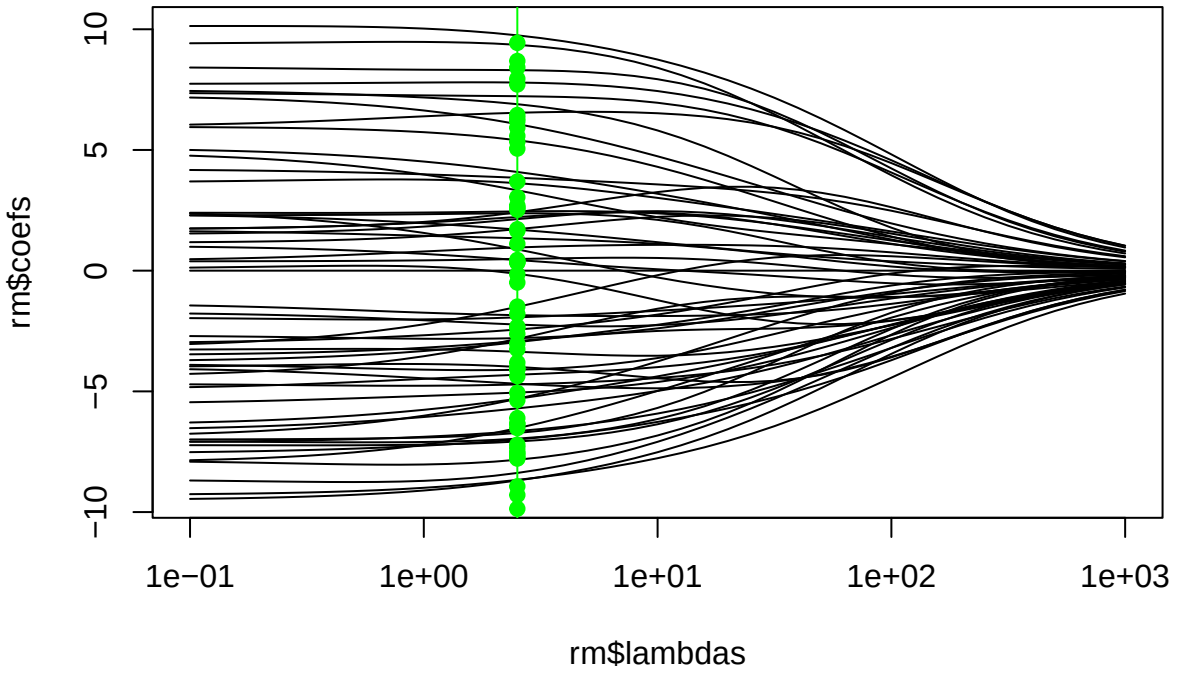
Tracer l'évolution des valeurs des coefficients d'un modèle ridge pour différentes valeurs de l'hyper-paramètre de régularisation λ. Réutiliser le code introduit au chapitre 15 sur la validation croisée un-contre-tous (voir le fichier `15_loocv.R`).

Dans cette implémentation, les étiquettes $\mathbf{y}$ sont centrées avant d'opérer la régression ridge. Dans le contexte de cette expérimentation, il peut être intéressant de proposer une version pour laquelle les étiquettes ne sont pas centrées. Quelle est la différence ? La réponse est liée à la présentation de la standardisation au chapitre 3 sur la régularisation de Tikhonov.

```
ridgeSVD <-
function(X, y)
{
  n <- nrow(X)
  X.init <- X
  y.init <- y
  X <- scale(X)
  Xs <- svd(X)
  d <- Xs$d
  function(lambda) {
    coef <- c(Xs$v %*% ((d / (d^2 + lambda)) * (t(Xs$u) %*% y)))
    coef <- coef / attr(X,"scaled:scale")
    inter <- - coef %*% attr(X,"scaled:center")
    coef <- c(inter, coef)
    trace.H <- sum(d^2 / (d^2 + lambda))
    yh <- coef[1] + X.init%*%coef[-1]
    gcv <- sum( ((y.init - yh) / (1 - (trace.H / n))) ^ 2 ) / n
    list(coef = coef, gcv = gcv)
  }
}

ridge <-
function(X, y, lambdas)
{
  p <- ncol(X)
  errs <- double(length(lambdas))
  coefs <- matrix(data = NA, nrow = length(lambdas), ncol = p+1)
  ridge <- ridgeSVD(X, y)
  idx <- 1
  for(lambda in lambdas) {
    res <- ridge(lambda)
    coefs[idx,] <- res$coef
    errs[idx] <- res$gcv
    idx <- idx + 1
  }
  err.min <- min(errs)
  lambda.best <- lambdas[which(errs == err.min)]
  coef.best <- coefs[which(errs == err.min),]
  yh <- coef.best[1] + X%*%coef.best[-1]
  mae <- mean(abs(yh - y))
  r <- list(coef = coef.best,
            lambda = lambda.best,
```

```
              mae = mae,
              coefs=coefs,
              lambdas=lambdas)
  class(r) <- "ridge"
  return(r)
}
```

```
lambdas <- 10^seq(-1,3,by=0.1)
rm <- ridge(X, y, lambdas)
matplot(rm$lambdas, rm$coefs, type=c('l'), pch=1, col='black', lty=1, log="x")
abline(v=rm$lambda, col='green')
points(x=rep(rm$lambda,length(beta)), y=beta, col='green', pch=19)
```



24.3 Espérance de l'erreur de prédiction

Pour les différentes valeurs de λ, calculer l'espérance de l'erreur de prédiction (c'est ici possible car les données sont simulées et nous connaissons le modèle qui les a générées). Cette notion a été introduite à la fin du chapitre 16. Quelle est la valeur de λ qui minimise une estimation de l'espérance de l'erreur de prédiction ? Comment se compare-t-elle à la valeur trouvée par validation croisée un-contre-tous ? Étudier les effets du nombre d'observations, du nombre de variables et de la quantité de bruit.

Soit h le modèle ridge et $\mathbf{x}^*$ une observation dont on cherche à prédire la cible. Nous savons que

l'espérance de l'erreur de prédiction est (voir chapitre 16) :

$$
\begin{aligned}
& E\left[(h(\mathbf{x}^*) - y^*)^2\right] \\
= & \{\text{Voir module “16 Biais et variance d'un estimateur”}\} \\
& E\left[\left(h(\mathbf{x}^*) - \overline{h(\mathbf{x}^*)}\right)^2\right] + \left(\overline{h(\mathbf{x}^*)} - f(\mathbf{x}^*)\right)^2 + \sigma^2 \\
= & \{\text{Introduction des définition de la variance, du biais et du bruit.}\} \\
& \text{Variance} \ + \ \text{Biais}^2 \ + \ \text{Bruit}^2
\end{aligned}
$$

Sous hypothèse d'un modèle génératif linéaire, nous avons calculé les valeurs de la variance et du biais des estimateurs d'un modèle ridge exprimés en fonction de la décomposition en valeurs singulières de $\mathbf{X}$ (notée, $\mathbf{X} = \mathbf{UDV}^T$) (voir chapitre 17).

$$
\begin{aligned}
Var\left(\hat{\boldsymbol{\beta}}_\lambda\right) &= \sigma^2 \sum_{d_j>0} \frac{d_j^2}{(d_j^2+\lambda)^2} \mathbf{v_j}\mathbf{v_j}^T \\
Bias\left(\hat{\boldsymbol{\beta}}_\lambda\right) &= E\left[\hat{\boldsymbol{\beta}}_\lambda\right] - \boldsymbol{\beta} = \left(\sum_{d_j>0} \mathbf{v_j} \frac{d_j}{d_j^2+\lambda} d_j \mathbf{v_j}^T \boldsymbol{\beta}\right) - \boldsymbol{\beta} = \sum_{d_j>0} \mathbf{v_j} \frac{\lambda}{d_j^2+\lambda} \mathbf{v_j}^T \boldsymbol{\beta}
\end{aligned}
$$

Comme $h(\mathbf{x}^*) = \hat{\boldsymbol{\beta}}_0 + \mathbf{x}^{*T}\hat{\boldsymbol{\beta}}_\lambda$, et que nous pouvons oublier $\hat{\boldsymbol{\beta}}_0$ qui doit être nul car le processus qui a généré les données n'a pas d'intercept, nous avons :

$$
E\left[(h(\mathbf{x}^*) - y^*)^2\right] = \mathbf{x}^{*T} Var\left(\hat{\boldsymbol{\beta}}_\lambda\right) \mathbf{x}^* + \left(\mathbf{x}^{*T} Bias\left(\hat{\boldsymbol{\beta}}_\lambda\right)\right)^2 + \sigma^2
$$

Pour implémenter le calcul de la variance et du biais, nous introduisons une fonction accessoire qui opère la multiplication d'une matrice diagonale (représentée par un vecteur de ses éléments diagonaux) par une matrice carrée.

```
multdiag <-
function(X,d)
{
  R <- matrix(NA, nrow=dim(X)[1], ncol=dim(X)[2])
  for (i in 1:dim(X)[2]) { R[,i]=X[,i]*d[i] }
  return(R)
}
```

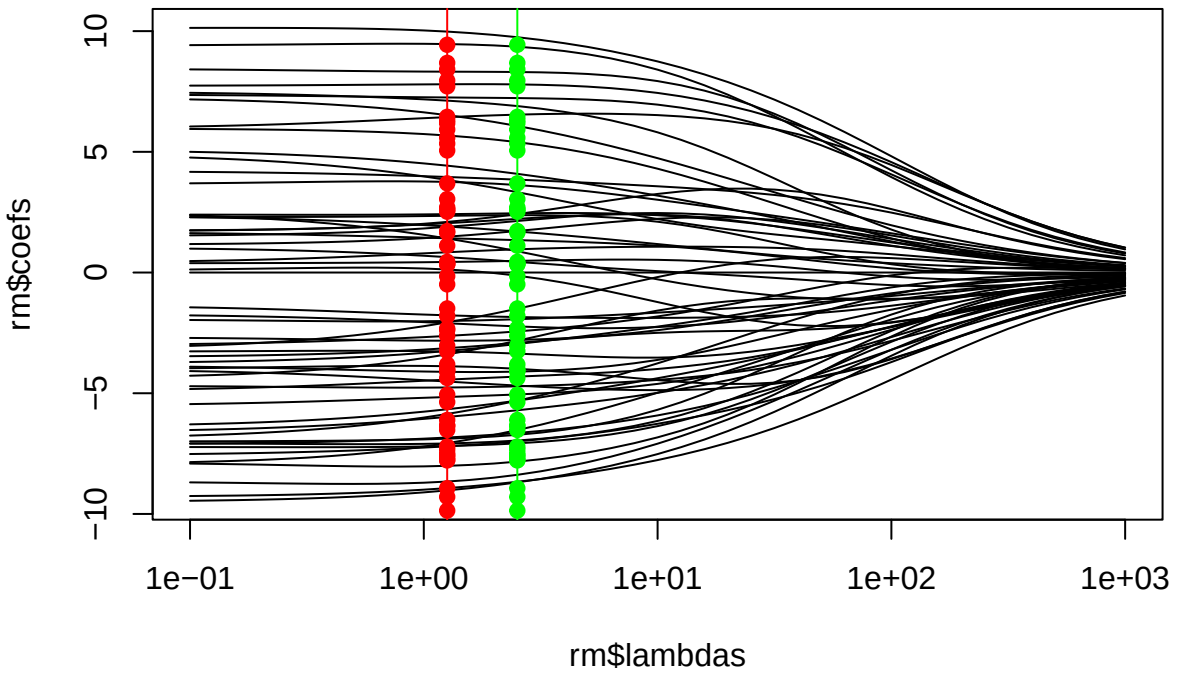
Nous introduisons les fonctions qui calculent $Var\left(\hat{\boldsymbol{\beta}}_\lambda\right)$ et $Bias\left(\hat{\boldsymbol{\beta}}_\lambda\right)$ pour une valeur de λ fixée. Nous utilisons ces fonctions pour calculer l'espérance de l'erreur pour une valeur de λ.

```
n2 <- 10000
X2 <- matrix(runif(n2*p),nrow=n2,ncol=p)
X2 <- scale(X2)
y2 <- X2%*%beta + rnorm(n2, mean=0, sd=sig2)


Xs <- svd(X2)
```

```
var <-
function(lambda)
{
  d <- (Xs$d^2)/(Xs$d^2 + lambda)^2
  var <- multdiag(Xs$v,d)
  var <- sig2^2 * tcrossprod(var,Xs$v)
}

bias <-
function(lambda)
{
  d <- lambda/(Xs$d^2+lambda)
  bias <- multdiag(Xs$v,d)
  bias <- bias %*% crossprod(Xs$v,beta)
}

epe <-
function(lambda)
{
  var <- var(lambda)
  bias <- bias(lambda)
  epe <- mean(rowSums(X2*(X2%*%var)))
  epe <- epe + mean((X2%*%bias)^2)
  epe <- epe + sig2^2
}

epes <- sapply(lambdas, epe)
```

Nous pouvons maintenant comparer la meilleure valeur théorique de l'hyper-paramètre de régularisation λ avec sa valeur estimée par validation croisée un-contre-tous.

```
epes.min <- min(epes)
lambda.th <- lambdas[which(epes == epes.min)]
coef.th <- rm$coefs[which(epes == epes.min),]
matplot(rm$lambdas, rm$coefs, type=c('l'), pch=1, col='black', lty=1, log="x")
abline(v=rm$lambda, col='green')
points(x=rep(rm$lambda,length(beta)), y=beta, col='green', pch=19)
abline(v=lambda.th, col='red')
points(x=rep(lambda.th,length(beta)), y=beta, col='red', pch=19)
```


rm$coefs
10
5
0
−5
−10
1e−01
1e+00
1e+01
1e+02
1e+03
rm$lambdas

Chapitre 25

TP - Projection aléatoire - Sujet

```
set.seed(1123)
```

L'idée de cette expérimentation provient de la la référence (Huang, Zhu, and Siew 2006) qui introduit le modèle prédictif dit *Extreme Learning Machine*.

25.1 Générer un jeu de données synthétique avec la fonction `sinc`

Utiliser la fonction `sinc` pour générer un jeu de données $\{y_i, x_i\}$.

$$y(x) = \begin{cases} sin(x)/x & x \neq 0, \\ 1 & x = 0 \end{cases}$$

Les x_i sont distribués de façon uniforme sur l'intervalle $(-10, 10)$. Un bruit (par exemple uniforme ou gaussien) est ajouté aux étiquettes y_i.

25.2 Régression ridge après projections non-linéaires aléatoires

Créer un modèle qui a la forme d'un réseau de neurones à une couche.

$$\hat{y}_i = \sum_{j=1}^{m} \beta_j g\left(\mathbf{w_j} \mathbf{x_i}^T + b_j\right) \quad i = 1, \ldots, n$$

Cependant, contrairement à un réseau de neurones, les m vecteurs $\mathbf{w_j}$ et scalaires b_j sont initialisés aléatoirement et ne sont jamais modifiés. La fonction g est une transformation non-linéaire (par ex., la fonction ReLU $g(x) = max(0, x)$, ou la fonction sigmoïde $g(x) = 1/(1 + exp(-x))$, etc.). Seuls les paramètres β_j sont inférés par régression ridge en utilisant la matrice $\mathbf{H}$ ci-dessous au lieu de la

martice $\mathbf{X}$ initiale.

$$\mathbf{H} = \begin{bmatrix} g\left(\mathbf{w_1}\mathbf{x_1}^T + b_1\right) & \dots & g\left(\mathbf{w_m}\mathbf{x_1}^T + b_m\right) \\ \vdots & \dots & \vdots \\ g\left(\mathbf{w_1}\mathbf{x_n}^T + b_1\right) & \dots & g\left(\mathbf{w_m}\mathbf{x_n}^T + b_m\right) \end{bmatrix}$$

Étudier les effets de la taille du jeu d'apprentissage, de la quantité de bruit, de l'hyperparamètre de régularisation, du nombre de transformations aléatoires, de la fonction non linéaire choisie, etc.

25.3 Jeu de données `housing`

`housing` est un jeu de données célèbre aux nombreuses vertues pédagogiques[1]. Il permet d'expérimenter sur un problème de régression réaliste, viz. prédire la valeur médiane d'une maison en fonction des caractéristiques de son quartier. Après une phase d'exploration des données, comparer, sur un jeu de test, un modèle linéaire par régression ridge et un modèle par projection aléatoire.

```
housing <- read.csv(file="data/housing.csv",header=TRUE)
str(housing)
```

```
## 'data.frame':    20640 obs. of  10 variables:
##  $ longitude         : num  -122 -122 -122 -122 -122 ...
##  $ latitude          : num  37.9 37.9 37.9 37.9 37.9 ...
##  $ housing_median_age: num  41 21 52 52 52 52 52 52 42 52 ...
##  $ total_rooms       : num  880 7099 1467 1274 1627 ...
##  $ total_bedrooms    : num  129 1106 190 235 280 ...
##  $ population        : num  322 2401 496 558 565 ...
##  $ households        : num  126 1138 177 219 259 ...
##  $ median_income     : num  8.33 8.3 7.26 5.64 3.85 ...
##  $ median_house_value: num  452600 358500 352100 341300 342200 ...
##  $ ocean_proximity   : chr  "NEAR BAY" "NEAR BAY" "NEAR BAY" "NEAR BAY" ...
```

Variable	Type	Commentaire
longitude	numeric	élevée pour un quartier à l'ouest
latitude	numeric	élevée pour un quartier au nord
housing_median_age	numeric	âge médian d'une maison du quartier
total_rooms	numeric	total des pièces pour les maisons du quartier
total_bedrooms	numeric	total des chambres pour les maisons du quartier
population	numeric	nombre de résidents du quartier
households	numeric	nombre de familles du quartier
median_income	numeric	revenu médian des familles du quartier (USD 10k)
median_house_value	numeric	valeur médiane d'une maison du quartier (USD)
ocean_proximity	factor	évaluation grossière de la distance à l'océan

1. https://www.kaggle.com/datasets/harrywang/housing

Chapitre 26

Projection aléatoire - `sinc` - Correction

```
set.seed(1123)
```

L'idée de cette expérimentation provient de la la référence (Huang, Zhu, and Siew 2006) qui introduit le modèle prédictif dit *Extreme Learning Machine*.

26.1 Générer un jeu de données synthétique avec la fonction `sinc`

Utiliser la fonction `sinc` pour générer un jeu de données $\{y_i, x_i\}$.

$$y(x) = \begin{cases} sin(x)/x & x \neq 0, \\ 1 & x = 0 \end{cases}$$

Les x_i sont distribués de façon uniforme sur l'intervalle $(-10, 10)$. Un bruit (par exemple uniforme ou gaussien) est ajouté aux étiquettes y_i.

Nous commençons par définir la fonction `sinc`.

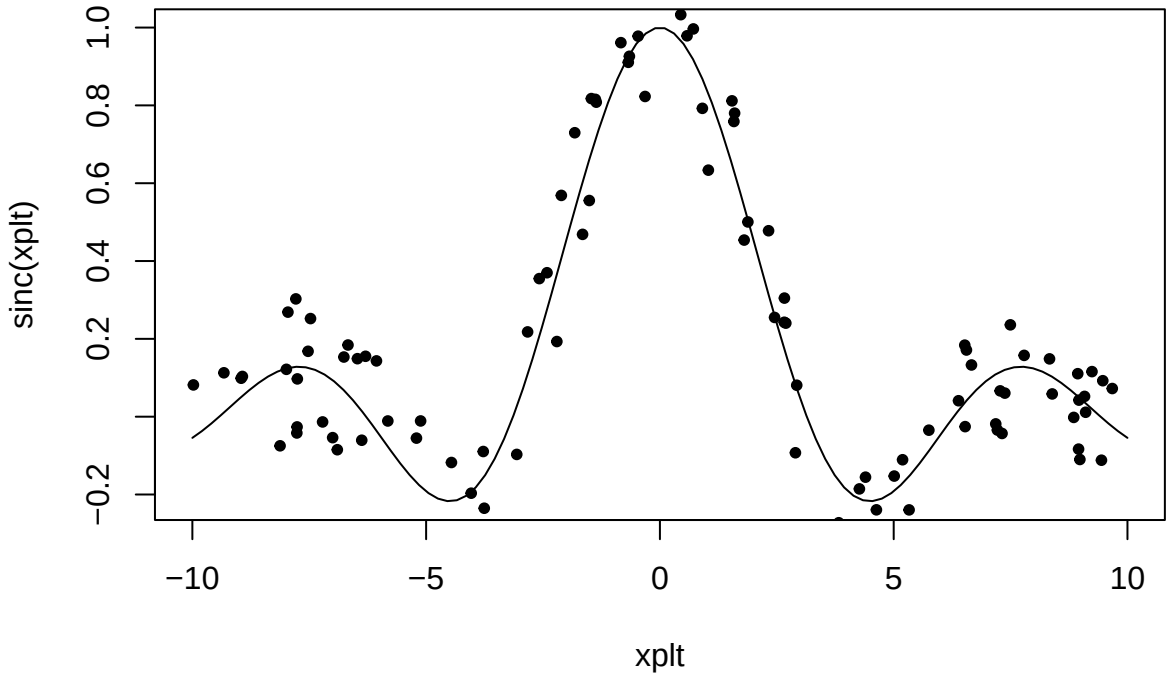
```
sinc <-
function(x)
{
  y <- sin(x) / x
  y[x==0] <- 1
  return(y)
}
```

Puis nous créons le jeu de données.

```
n <- 100
X <- runif(n, min=-10, max=10)
X <- as.matrix(X)
y <- sinc(X) + runif(n, min=-0.2, max=0.2)
```

Et nous l'affichons.

```
xplt <- seq(-10,10,length.out=100)
plot(xplt,sinc(xplt), type='l')
points(X,y, pch=20)
```



26.2 Régression ridge après projections non-linéaires aléatoires

Créer un modèle qui a la forme d'un réseau de neurones à une couche.

$$\hat{y}_i = \sum_{j=1}^{m} \beta_j g\left(\mathbf{w_j x_i}^T + b_j\right) \quad i = 1, \ldots, n$$

Cependant, contrairement à un réseau de neurones, les m vecteurs $\mathbf{w_j}$ et scalaires b_j sont initialisés aléatoirement et ne sont jamais modifiés. La fonction g est une transformation non-linéaire (par ex., la fonction ReLU $g(x) = max(0, x)$, ou la fonction sigmoïde $g(x) = 1/(1 + exp(-x))$, etc.). Seuls les paramètres β_j sont inférés par régression ridge en utilisant la matrice $\mathbf{H}$ ci-dessous au lieu de la martice $\mathbf{X}$ initiale.

$$\mathbf{H} = \begin{bmatrix} g\left(\mathbf{w_1 x_1}^T + b_1\right) & \ldots & g\left(\mathbf{w_m x_1}^T + b_m\right) \\ \vdots & \ldots & \vdots \\ g\left(\mathbf{w_1 x_n}^T + b_1\right) & \ldots & g\left(\mathbf{w_m x_n}^T + b_m\right) \end{bmatrix}$$

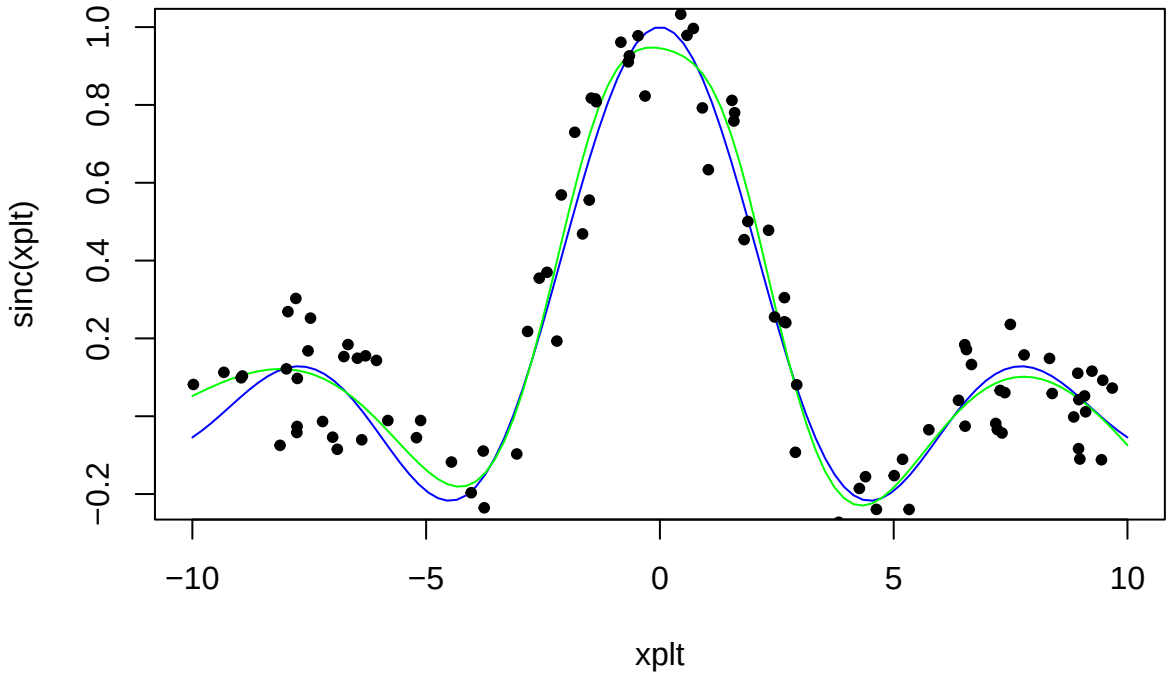
Étudier les effets de la taille du jeu d'apprentissage, de la quantité de bruit, de l'hyperparamètre de régularisation, du nombre de transformations aléatoires, de la fonction non linéaire choisie, etc.

Nous commençons par créer le modèle en étendant la régression ridge.

```
relu <- function(x){ifelse(x>=0,x,0)}
logistic <- function(x){1/(1+exp(-x))}

elm <-
function(X,y,m=2000,method=c('logistic', 'ReLU'),lambdas=NULL)
{
  method <- match.arg(method)

  X <- as.matrix(X)
  p <- ncol(X)
  W <- matrix(runif(p*m, min=-1, max=1), nrow=p, ncol=m)
  H <- X%*%W
  b <- runif(m)
  H <- sweep(H,2,b,'+')

  if(method=='logistic') {
    H <- logistic(H)
  } else if(method=='ReLU') {
    H <- relu(H)
  }

  rm <- ridge(H,y,lambdas)
  r <- list(ridge = rm,
            m = m,
            method=method,
            W = W,
            b = b)
  class(r) <- "elm"
  return(r)
}

predict.elm <-
function(o, newdata)
{
  newdata <- as.matrix(newdata)
  H <- newdata %*% o$W
  H <- sweep(H,2,o$b,'+')
  if(o$method=='logistic') {
    H <- logistic(H)
  } else if(o$method=='ReLU') {
    H <- relu(H)
  }

  yh <- predict(o$ridge,H)
}
```

Nous appliquons le modèle au jeu de données synthétique.

```
rm <- elm(X,y,m=500,method='logistic')
yh <- predict(rm,xplt)
plot(xplt,sinc(xplt),type='l',col='blue')
lines(xplt,yh,type='l',col='green')
points(X,y, pch=20)
```



Chapitre 27

TP - Projection aléatoire - `housing` PCA - Correction

```
set.seed(1123)
```

27.1 Présentation du jeu de données `housing`

`housing` est un jeu de données célèbre aux nombreuses vertues pédagogiques[1]. Il permet d'expérimenter sur un problème de régression réaliste, viz. prédire la valeur médiane d'une maison en fonction des caractéristiques de son quartier. Après une phase d'exploration des données, comparer, sur un jeu de test, un modèle linéaire par régression ridge et un modèle par projection aléatoire.

```
housing <- read.csv(file="data/housing.csv",header=TRUE)
str(housing)
```

```
## 'data.frame':    20640 obs. of  10 variables:
##  $ longitude         : num  -122 -122 -122 -122 -122 ...
##  $ latitude          : num  37.9 37.9 37.9 37.9 37.9 ...
##  $ housing_median_age: num  41 21 52 52 52 52 52 52 42 52 ...
##  $ total_rooms       : num  880 7099 1467 1274 1627 ...
##  $ total_bedrooms    : num  129 1106 190 235 280 ...
##  $ population        : num  322 2401 496 558 565 ...
##  $ households        : num  126 1138 177 219 259 ...
##  $ median_income     : num  8.33 8.3 7.26 5.64 3.85 ...
##  $ median_house_value: num  452600 358500 352100 341300 342200 ...
##  $ ocean_proximity   : chr  "NEAR BAY" "NEAR BAY" "NEAR BAY" "NEAR BAY" ...
```

Variable	Type	Commentaire
longitude	numeric	élevée pour un quartier à l'ouest
latitude	numeric	élevée pour un quartier au nord
housing_median_age	numeric	âge médian d'une maison du quartier

1. https://www.kaggle.com/datasets/harrywang/housing

Variable	Type	Commentaire
total_rooms	numeric	total des pièces pour les maisons du quartier
total_bedrooms	numeric	total des chambres pour les maisons du quartier
population	numeric	nombre de résidents du quartier
households	numeric	nombre de familles du quartier
median_income	numeric	revenu médian des familles du quartier (USD 10k)
median_house_value	numeric	valeur médiane d'une maison du quartier (USD)
ocean_proximity	factor	estimation de la distance à l'océan

27.2 Prétraitements

Nous transformons la variable `ocean_proximity`, pour l'instant représentée par une chaîne de caractères, en une variable catégorielle (ou facteur) qui sera représentée par un codage disjonctif complet (ou *one hot encoding*).

```
housing$ocean_proximity <- as.factor(housing$ocean_proximity)
summary(housing)
```

```
##    longitude         latitude     housing_median_age  total_rooms
##  Min.   :-124.3   Min.   :32.54   Min.   : 1.00      Min.   :    2
##  1st Qu.:-121.8   1st Qu.:33.93   1st Qu.:18.00      1st Qu.: 1448
##  Median :-118.5   Median :34.26   Median :29.00      Median : 2127
##  Mean   :-119.6   Mean   :35.63   Mean   :28.64      Mean   : 2636
##  3rd Qu.:-118.0   3rd Qu.:37.71   3rd Qu.:37.00      3rd Qu.: 3148
##  Max.   :-114.3   Max.   :41.95   Max.   :52.00      Max.   :39320
##
##  total_bedrooms     population      households     median_income
##  Min.   :   1.0   Min.   :    3   Min.   :   1.0   Min.   : 0.4999
##  1st Qu.: 296.0   1st Qu.:  787   1st Qu.: 280.0   1st Qu.: 2.5634
##  Median : 435.0   Median : 1166   Median : 409.0   Median : 3.5348
##  Mean   : 537.9   Mean   : 1425   Mean   : 499.5   Mean   : 3.8707
##  3rd Qu.: 647.0   3rd Qu.: 1725   3rd Qu.: 605.0   3rd Qu.: 4.7432
##  Max.   :6445.0   Max.   :35682   Max.   :6082.0   Max.   :15.0001
##  NA's   :207
##  median_house_value   ocean_proximity
##  Min.   : 14999     <1H OCEAN :9136
##  1st Qu.:119600     INLAND    :6551
##  Median :179700     ISLAND    :   5
##  Mean   :206856     NEAR BAY  :2290
##  3rd Qu.:264725     NEAR OCEAN:2658
##  Max.   :500001
##
```

Nous remplaçons les valeurs manquantes pour la variable `total_bedrooms` par la médiane.

```
housing$total_bedrooms[is.na(housing$total_bedrooms)] <- median(housing$total_bedrooms, na
sum(is.na(housing))
```

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

La catégorie `ISLAND` de la variable `ocean_proximity` est extrêmement sous représentée. Nous proposons de supprimer les observations qui lui correspondent.

```
housing <- housing[housing$ocean_proximity != "ISLAND", ]
nrow(housing)
```

```
## [1] 20635
```

27.3 Analyse exploratoire par PCA

Explorons les variables numériques par analyse en composantes principales.

Nous sélectionnons les variables explicatives numériques dans une matrice $\mathbf{X}$.

```
X <- housing[,c("longitude", "latitude", "housing_median_age", "total_rooms",
                "total_bedrooms", "population", "households", "median_income",
                "median_house_value")]
```

27.3.1 Première analyse

Nous réalisons une première fois l'analyse en composantes principales.

```
fam <- fa(X) # fam pour 'factor analysis model'
```

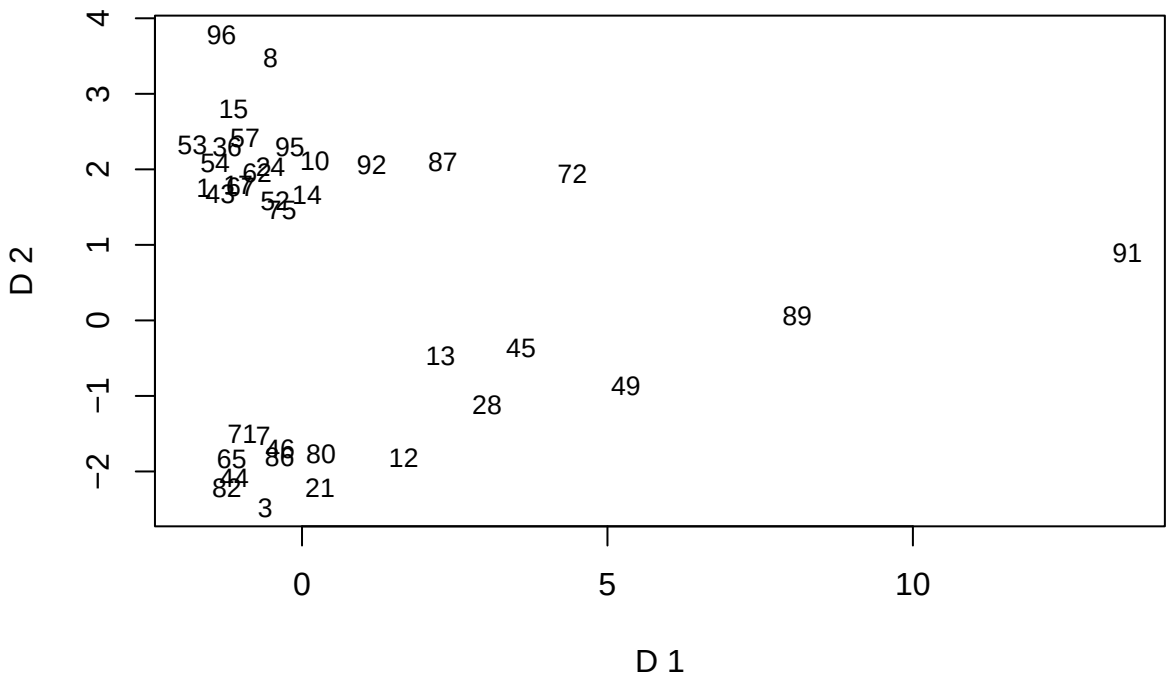
Nous affichons le pourcentage de variance expliquée par chaque axe principal.

```
fam$prctPrcp
```

```
## [1] 45.91 22.37 19.33  8.90  2.26  0.80  0.24  0.13  0.05
```

Nous affichons, sur les deux premiers axes principaux, les centres des clusters qui contribuent le plus à ces axes.

```
print(fam)
```




```
far <- away(fam)
```

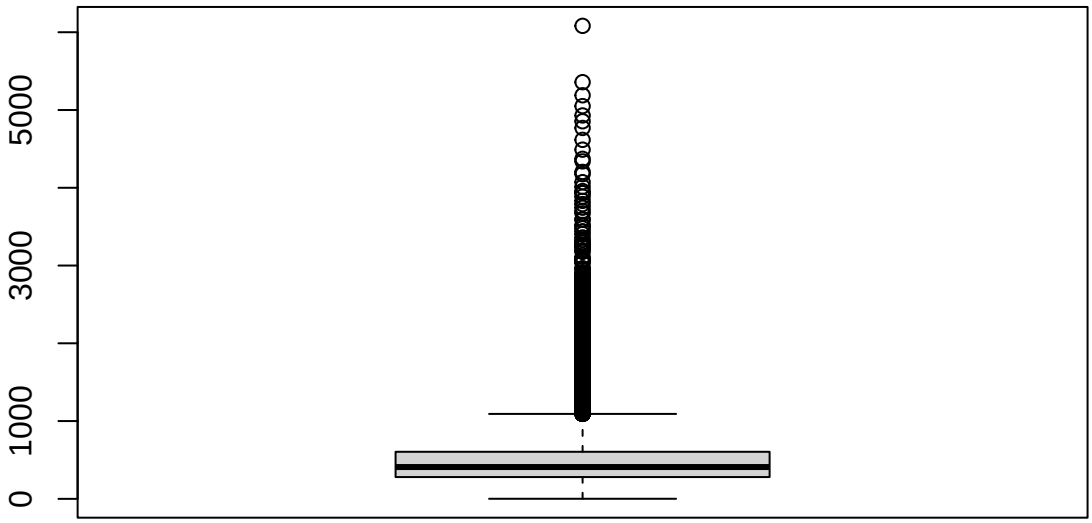
Le cluster 91 (`far$id`) contribue à expliquer 44.64 % (`round(fam$ctr[far$id,1]*100, 2)`) de la variance du premier axe. Il est composé de 5 (`far$size`) quartiers :

```
housing[far$names,]
```

```
##       longitude latitude housing_median_age total_rooms total_bedrooms
## 6058    -117.78    34.03                  8       32054           5290
## 9881    -121.79    36.64                 11       32627           6445
## 10310   -117.74    33.89                  4       37937           5471
## 13140   -121.44    38.43                  3       39320           6210
## 15361   -117.42    33.35                 14       25135           4819
##       population households median_income median_house_value ocean_proximity
## 6058       15507       5050        6.0191             253900       <1H OCEAN
## 9881       28566       6082        2.3087             118800       <1H OCEAN
## 10310      16122       5189        7.4947             366300       <1H OCEAN
## 13140      16305       5358        4.9516             153700          INLAND
## 15361      35682       4769        2.5729             134400       <1H OCEAN
```

Ces quartiers semblent remarquables par le grand nombre de familles qui les habitent.

```
boxplot(housing$households)
```



Nous remarquons que le cluster 91 est principalement expliqué par le premier axe principal.

```
fam$cos2[far$id,]
```

```
## [1] 96.42  0.42  0.07  2.03  0.24  0.75  0.04  0.01  0.00
```

Les contributions des variables aux axes principaux nous montrent que le premier axe principal correspond surtout à un facteur taille de variables très corrélées : `total_rooms`, `total_bedrooms`, `population` et `households`.

```
fam$varctr
```

```
##                     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## longitude           0.01 0.76 0.22 0.00 0.00 0.00 0.01 0.00    0
## latitude            0.01 0.87 0.10 0.01 0.00 0.00 0.01 0.00    0
## housing_median_age  0.29 0.00 0.01 0.69 0.01 0.00 0.00 0.00    0
## total_rooms         0.97 0.00 0.01 0.01 0.00 0.00 0.00 0.01    0
## total_bedrooms      0.97 0.00 0.00 0.01 0.00 0.01 0.00 0.00    0
## population          0.92 0.01 0.00 0.02 0.01 0.05 0.00 0.00    0
## households          0.97 0.00 0.00 0.02 0.00 0.00 0.01 0.00    0
## median_income       0.00 0.17 0.70 0.04 0.09 0.00 0.00 0.00    0
## median_house_value  0.00 0.20 0.70 0.01 0.09 0.00 0.00 0.00    0
```

27.3.2 Seconde analyse

Pour mieux visualiser des phénomènes intéressants qui seraient sinon masqués par ce facteur taille, nous proposons d'introduire de nouvelles variables pour décrire le nombre de pièces, le nombre de chambres et le nombre de personnes relativement au nombre de familles du quartier.

```
housing['rooms_per_household'] <- housing['total_rooms'] / housing['households']
housing['bedrooms_per_household'] <- housing['total_bedrooms'] / housing['households']
housing['population_per_household'] <- housing['population'] / housing['households']
```

Nous recommençons l'analyse avec ces nouvelles variables.

```
X <- housing[,c("longitude", "latitude", "housing_median_age", "rooms_per_household",
                "bedrooms_per_household", "population_per_household", "median_income",
                "median_house_value")]
fam <- fa(X)
```

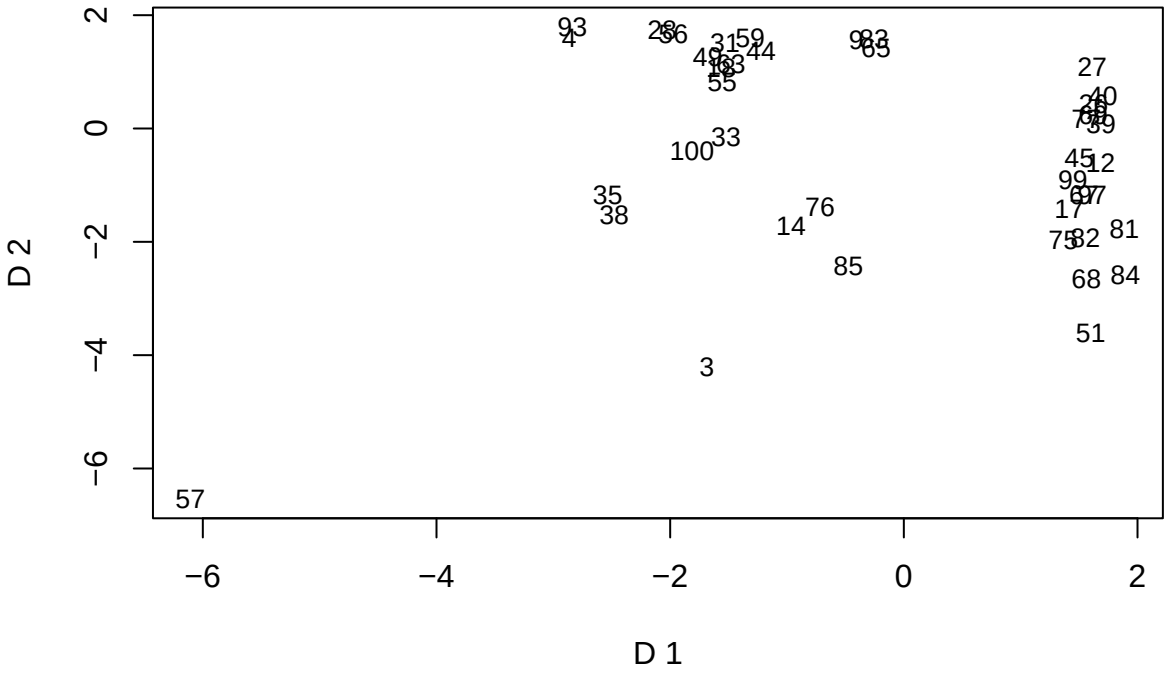
Nous affichons le pourcentage de variance expliquée par chaque axe principal.

```
fam$prctPrcp
```

```
## [1] 26.45 22.48 20.91 12.69 11.39  3.96  1.74  0.38
```

Nous affichons, sur les deux premiers axes principaux, les centres des clusters qui contribuent le plus à ces axes.

```
print(fam)
```




```
far <- away(fam)
```

Le cluster 57 contribue à expliquer 17.8 % de la variance du premier axe. Il est composé de 2 quartiers :

```
housing[far$names,]
```

```
##      longitude latitude housing_median_age total_rooms total_bedrooms
## 1915   -120.10    38.91                 33        1561            282
## 1980   -120.08    38.80                 34        1988            511
##      population households median_income median_house_value ocean_proximity
## 1915         30         11         1.875              500001          INLAND
## 1980         36         15         4.625              162500          INLAND
##      rooms_per_household bedrooms_per_household population_per_household
## 1915            141.9091               25.63636                 2.727273
## 1980            132.5333               34.06667                 2.400000
```

Cependant, la contribution du cluster 57 à la variance est surtout expliquée par l'axe 3. Ce dernier est lui-même surtout expliqué par les variables `rooms_per_household` et `bedrooms_per_household` (elles-mêmes très corrélées, ce n'est pas étonnant).

```
fam$cos2[far$id,]
```

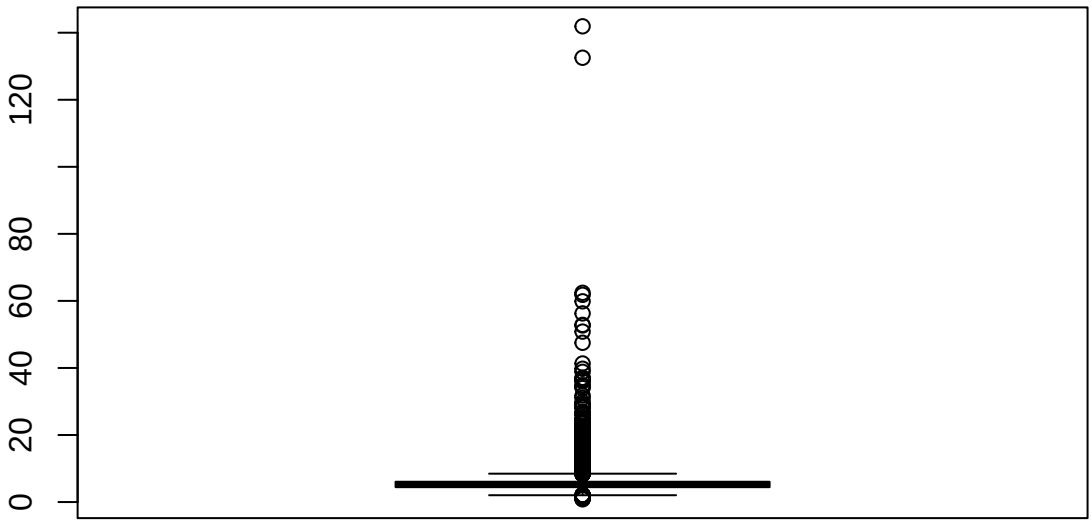
```
## [1] 28.74 33.01 35.55  0.37  0.00  2.06  0.17  0.10
```

```
fam$varctr
```

```
##                            [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## longitude                  0.66 0.02 0.23 0.07 0.01 0.00 0.00 0.01
## latitude                   0.86 0.03 0.05 0.04 0.00 0.00 0.00 0.01
## housing_median_age         0.02 0.05 0.11 0.29 0.52 0.00 0.01 0.00
## rooms_per_household        0.23 0.29 0.37 0.00 0.01 0.07 0.04 0.00
## bedrooms_per_household     0.21 0.34 0.33 0.02 0.00 0.06 0.04 0.00
## population_per_household 0.04 0.01 0.17 0.46 0.29 0.02 0.00 0.00
## median_income              0.03 0.53 0.24 0.02 0.07 0.09 0.02 0.00
## median_house_value         0.07 0.54 0.16 0.11 0.02 0.08 0.02 0.00
```

Nous vérifions que ces quartiers correspondent effectivement à des valeurs très atypiques de la variable `rooms_per_household`.

```
boxplot(housing$rooms_per_household)
```



Nous remarquons aussi que la valeurs des maisons sur l'un de ces quartiers est très élevée. En fait, c'est la plus grande valeur rencontrée sur ce jeu de données.

```
capMedianHouseValue <- 500001
nbCapMedianHouseValue <- table(housing$median_house_value)[as.character(capMedianHouseValue
```

Pour 965 quartiers, la valeur de la variable `median_house_value` est égale à 5.00001×10^5. Il semble que les maisons dont la valeur dépasse une certaine somme aient été toutes enregistrées à cette somme maximale. Ces quartiers risquent d'avoir une mauvaise influence sur notre modèle prédictif. Nous proposons donc de les retirer.

```
housing <- housing[housing$median_house_value < capMedianHouseValue, ]
nrow(housing)
```

```
## [1] 19670
```

```
X <- housing[,c("longitude", "latitude", "housing_median_age", "rooms_per_household",
                "bedrooms_per_household", "population_per_household", "median_income",
                "median_house_value")]
fam <- fa(X)
```

Nous affichons le pourcentage de variance expliquée par chaque axe principal.

```
fam$prctPrcp
```

```
## [1] 26.82 21.66 20.25 13.28 11.89  4.20  1.52  0.38
```

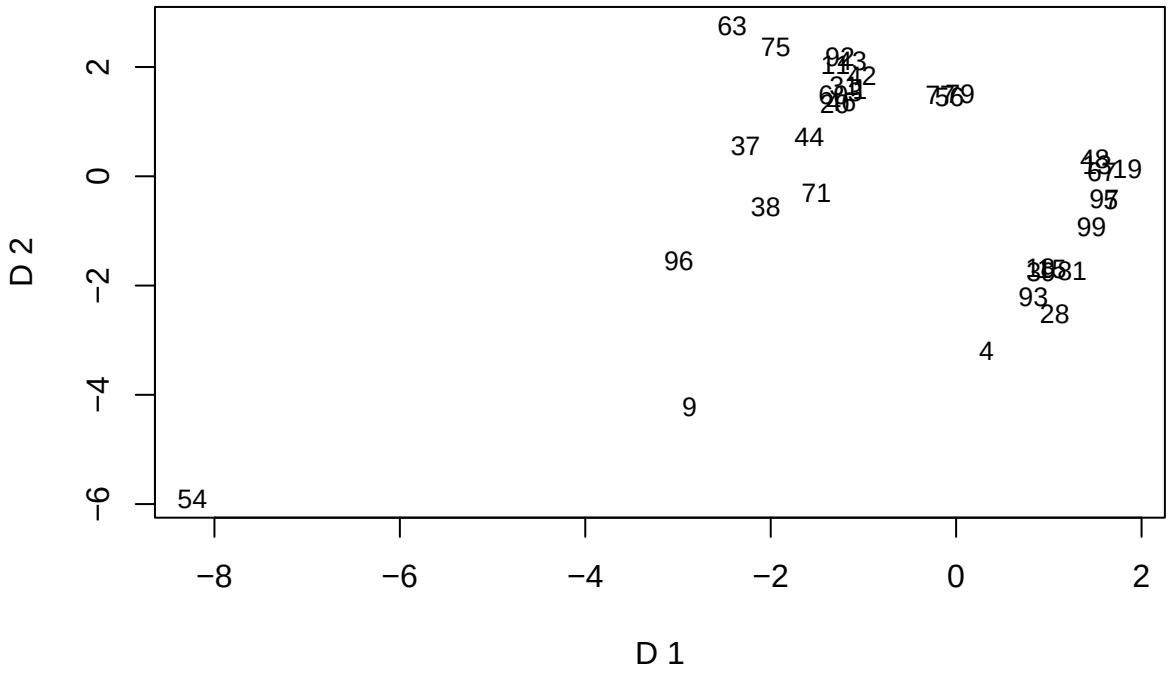
Nous vérifions les contributions des variables aux axes principaux.

```
fam$varctr
```

```
##                            [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## longitude                  0.54 0.21 0.19 0.02 0.01 0.00 0.00 0.01
## latitude                   0.72 0.19 0.04 0.04 0.00 0.00 0.00 0.01
## housing_median_age         0.02 0.01 0.03 0.66 0.27 0.01 0.01 0.00
## rooms_per_household        0.42 0.25 0.23 0.00 0.00 0.07 0.04 0.00
## bedrooms_per_household     0.40 0.32 0.18 0.02 0.00 0.03 0.05 0.00
## population_per_household   0.02 0.00 0.08 0.30 0.58 0.01 0.00 0.00
## median_income              0.01 0.46 0.36 0.02 0.02 0.11 0.02 0.00
## median_house_value         0.00 0.29 0.52 0.01 0.06 0.11 0.01 0.00
```

Nous affichons, sur les deux premiers axes principaux, les centres des clusters qui contribuent le plus à ces axes.

```
print(fam)
```




```
far <- away(fam)
```

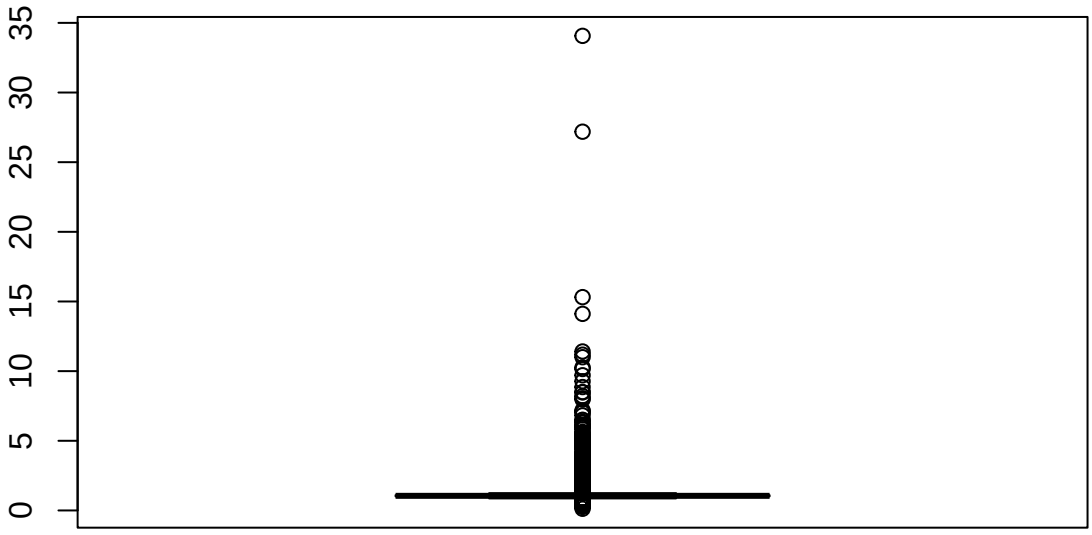
Le cluster 54 contribue à expliquer 31.95 % de la variance du premier axe. Il est composé de 1 quartiers :

```
housing[far$names,]
```

```
##      longitude latitude housing_median_age total_rooms total_bedrooms
## 1980   -120.08     38.8                34        1988            511
##      population households median_income median_house_value ocean_proximity
## 1980         36         15         4.625              162500          INLAND
##      rooms_per_household bedrooms_per_household population_per_household
## 1980            132.5333               34.06667                       2.4
```

Ce quartier, que nous avons déjà rencontré dans la précédente analyse est vraiment atypique. Il s'agit d'un petit quartier avec énormément de pièces par foyer. Nous proposons de le retirer.

```
boxplot(housing$bedrooms_per_household)
```



27.3.4 Quatrième analyse

```
housing <- housing[!(row.names(housing) %in% far$names),]
```

```
X <- housing[,c("longitude", "latitude", "housing_median_age", "rooms_per_household",
                "bedrooms_per_household", "population_per_household", "median_income",
                "median_house_value")]
fam <- fa(X)
```

Nous affichons le pourcentage de variance expliquée par chaque axe principal.

```
fam$prctPrcp
```

```
## [1] 25.15 21.37 19.37 13.58 12.00  6.23  2.00  0.30
```

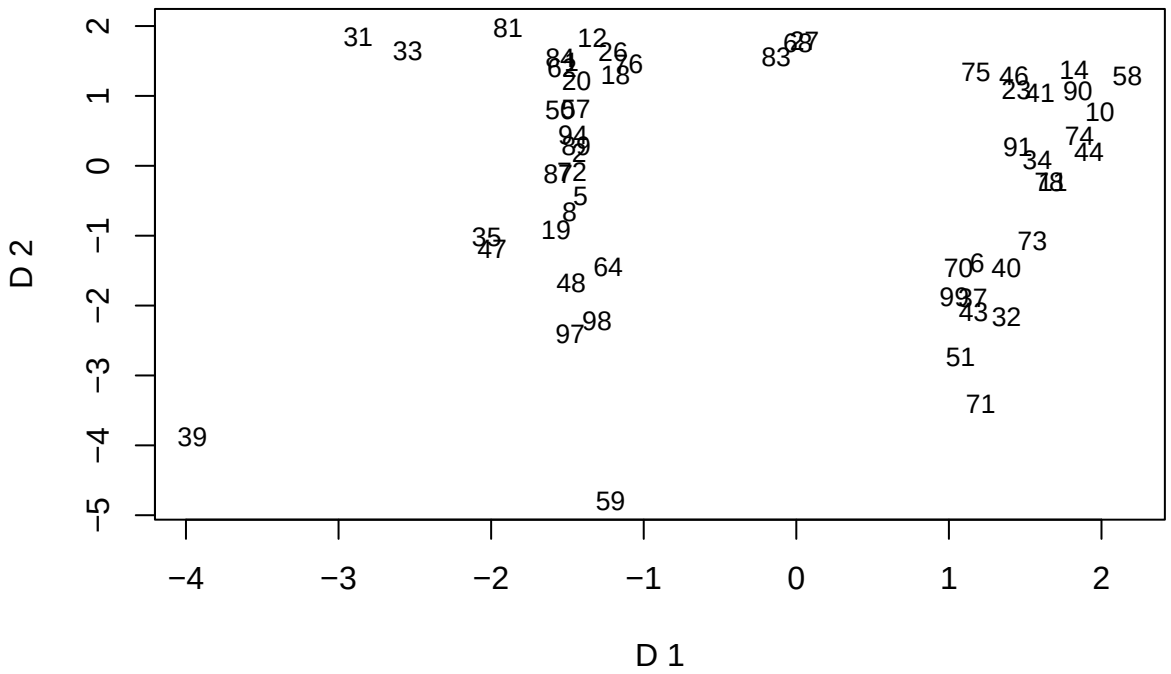
Nous vérifions les contributions des variables aux axes principaux.

```
fam$varctr
```

```
##                            [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## longitude                  0.87 0.01 0.05 0.04 0.02 0.00 0.00 0.01
## latitude                   0.93 0.03 0.00 0.02 0.00 0.00 0.00 0.01
## housing_median_age         0.02 0.01 0.14 0.50 0.29 0.03 0.01 0.00
## rooms_per_household        0.04 0.02 0.70 0.00 0.00 0.23 0.01 0.00
## bedrooms_per_household     0.03 0.12 0.51 0.14 0.05 0.13 0.02 0.00
## population_per_household   0.10 0.10 0.07 0.23 0.45 0.02 0.02 0.00
## median_income              0.01 0.84 0.00 0.02 0.03 0.04 0.06 0.00
## median_house_value         0.01 0.58 0.07 0.14 0.11 0.05 0.04 0.00
```

Nous affichons, sur les deux premiers axes principaux, les centres des clusters qui contribuent le plus à ces axes.

```
print(fam)
```



Nous proposons d'arrêter ici l'analyse exploratoire.

Chapitre 28

TP - Projection aléatoire - housing ELM - Correction

```
set.seed(1123)
```

28.1 Créations des jeux d'entraînement et de test

Nous reproduisons les prétraitements découverts pendant l'analyse exploratoire

```
housing <- read.csv(file="data/housing.csv",header=TRUE)
housing$ocean_proximity <- as.factor(housing$ocean_proximity)
housing$total_bedrooms[is.na(housing$total_bedrooms)] <-
                        median(housing$total_bedrooms, na.rm=TRUE)
housing <- housing[housing$ocean_proximity != "ISLAND", ]
housing['rooms_per_household'] <- housing['total_rooms'] / housing['households']
housing['bedrooms_per_household'] <- housing['total_bedrooms'] / housing['households']
housing['population_per_household'] <- housing['population'] / housing['households']
housing <- housing[housing$median_house_value < 500001, ]
housing <- housing[-1980,]
```

Nous rappelons une fonction pour créer un jeu d'entraînement et un jeu de test.

```
# Séparer le jeu de données en un jeu d'entraînement et un jeu de test
# INPUT : jeu de données initial et proportion des données conservées pour
#         l'entraînement.
splitdata <-
function(data,p)
{
  n <- nrow(data$X)
  nentr <- round(p*n)
  entridx <- sample(1:n, nentr, replace=FALSE)
  list(entr  = list(X = data$X[entridx,,drop=FALSE],  Y = data$Y[entridx]),
       test = list(X = data$X[-entridx,,drop=FALSE], Y = data$Y[-entridx]))
```

```
}
```

Nous créons les jeux de données.

```
data <- list(X=housing[,c("longitude", "latitude", "housing_median_age",
                          "rooms_per_household", "bedrooms_per_household",
                          "population_per_household", "households",
                          "median_income")],
             Y=housing[,c("median_house_value")])
splitres <- splitdata(data,0.8)
entr <- splitres$entr
test <- splitres$test
```

Si nous essayons d'inférer un modèle ELM avec l'appel ci-dessous qui utilise le code développé pour les données synthétiques `sinc`, nous rencontrons un bug car la projection aléatoire des données peut créer des colonnes avec seulement des valeurs négative. Lorsqu'une telle colonne est transformée, par exemple par la fonction non linéaire `ReLU`, elle deviendra pleine de zéros et rendra impossible le calcul de la décomposition en valeurs singulières. C'est pourquoi, pour les modèles qui emploient des fonctions de transformation non linéaires, les données sont souvent normalisées entre 0 et 1. Par ailleurs, s'il reste des colonnes nulles dans $\mathbf{H}$, nous les retirons (et nous mettons à jour en conséquence les vecteurs aléatoires $\mathbf{W}$ et $\mathbf{b}$).

```
rm <- elm(entr$X,entr$Y)
```

28.2 Modèle ELM

```
relu <- function(x){ifelse(x>=0,x,0)}
logistic <- function(x){1/(1+exp(-x))}

elm <-
function(X,y,m=1000,method=c('logistic', 'ReLU'),lambdas=NULL)
{
  method <- match.arg(method)

  X <- as.matrix(X)
  Xisnum <- apply(X,2,is.numeric) # les colonnes qui sont des facteurs
                                  # n'ont pas besoin d'être normalisées
  Xmax <- apply(X[,Xisnum],2,max)
  Xmin <- apply(X[,Xisnum],2,min)
  Xrange <- Xmax - Xmin
  X[,Xisnum] <- sweep(X[,Xisnum],2,Xmin,'-')
  X[,Xisnum] <- sweep(X[,Xisnum],2,Xrange,'/')

  p <- ncol(X)
  W <- matrix(runif(p*m, min=-1, max=1), nrow=p, ncol=m)
  H <- X%*%W
```

```
  b <- runif(m)
  H <- sweep(H,2,b,'+')

  if(method=='logistic') {
    H <- logistic(H)
  } else if(method=='ReLU') {
    H <- relu(H)
  }

  nullcols <- which(apply(H,2,sum)<=1)
  if(length(nullcols)>0) {
    H <- H[,-nullcols]
    W <- W[,-nullcols]
    b <- b[-nullcols]
    m <- m - length(nullcols)
  }

  rm <- ridge(H,y,lambdas)
  r <- list(ridge = rm,
            m = m,
            W = W,
            b = b,
            method=method,
            isnum = Xisnum,
            min = Xmin,
            range = Xrange)
  class(r) <- "elm"
  return(r)
}

predict.elm <-
function(o, newdata)
{
  newdata <- as.matrix(newdata)
  newdata[,o$isnum] <- sweep(newdata[,o$isnum],2,o$min,'-')
  newdata[,o$isnum] <- sweep(newdata[,o$isnum],2,o$range,'/')

  H <- newdata %*% o$W
  H <- sweep(H,2,o$b,'+')
  if(o$method=='logistic') {
    H <- logistic(H)
  } else if(o$method=='ReLU') {
    H <- relu(H)
  }
  yh <- predict(o$ridge,H)
}
```

28.3 Prédictions sur le jeu de test

Nous apprenons un modèle par projection aléatoire et un modèle ridge.

```
lambdas <- 10^seq(-3,2,length.out=10)
rm <- elm(entr$X,entr$Y,m=200,method='logistic',lambdas=lambdas)
lm <- ridge(entr$X,entr$Y)
rmYh <- predict(rm,test$X)
lmYh <- predict(lm,test$X)
rmTestMAE <- mean(abs(rmYh - test$Y))
lmTestMAE <- mean(abs(lmYh - test$Y))
```

Sur le jeu de test, le modèle par projection aléatoire commet une erreur absolue moyenne de 4.1233631×10^4 tandis que le modèle linéaire régularisé commet une erreur de 4.8478881×10^4.

Chapitre 29

TP - Régression ridge à noyau - Sujet

```
set.seed(1123)
```

L'idée de cette expérimentation provient de la la référence (Rupp 2015) qui introduit les concepts essentiels du machine learning pour un public de spécialistes en mécanique quantique.

29.1 Analyse de l'effet du paramètre σ^2 pour un noyau gaussien

Soit un noyau gaussien k.

$$k(\mathbf{x_j}, \mathbf{x_i}) = exp\left(-\frac{1}{\sigma^2}\|\mathbf{x_j} - \mathbf{x_i}\|^2\right)$$

Générer des points x_i, par exemple entre -5 et 5.

```
X <- seq(from=-5,to=5,by=0.1)
```

Générer la matrice des similarités (ou noyau) $\mathbf{K}$, résultat de l'application de la fonction k à chaque paire de points (x_i, x_j). Le faire pour σ égal à 0.5, 1 ou 2.

Afficher la courbe du noyau en 0, c'est-à-dire $k(0, x)$.

Afficher la courbe obtenue par une combinaison linéaire aléatoire des similarités entre une observation x et l'ensemble des observations x_i :

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x_i})$$

Comparer le type de courbes obtenues pour les différentes valeurs de σ.

29.2 Régression ridge à noyau sur un petite exemple synthétique

Soit un jeu de données synthétique construit à partir de la fonction $cos(x)$ appliquée aux points $\{0, \pi/8, 2\pi/8, 3\pi/8, 4\pi/8\}$

```
X <- pi/8 * seq(0,4)
y <- cos(X)
```

Apprendre un modèle par régression ridge à noyau sur ce jeu de données. Dans un premier temps :

- considérer le jeu de données non bruité
- fixer l'hyper-paramètre de régularisation λ presque à zéro (e.g., 10^{-14})
- comparer différentes valeurs de l'hyper-paramètre σ qui contrôle le rayon du noyau gaussien (e.g., $0.01, 0.5, 10^4$)
- Afficher à chaque fois : le jeu de données, la courbe théorique, la courbe prédite, les courbes gaussiennes centrées sur chaque observation du jeu d'entraînement et dont la combinaison linéaire est la courbe prédite.

Ensuite, vérifier le bon fonctionnement de l'approche proposée en cours qui fixe σ^2 au nombre de dimensions du jeu de données et qui détermine la valeur de λ par validation croisée un contre tous. Discuter les rôles complémentaires des hyperparamètres σ^2 et λ.

De nombreuses expériences complémentaires sont intéressantes à mener (comme étudier l'effet de l'ajout de bruit, etc.).

Chapitre 30

TP - Régression ridge à noyau - Correction

```
set.seed(1123)
```

L'idée de cette expérimentation provient de la la référence (Rupp 2015) qui introduit les concepts essentiels du machine learning pour un public de spécialistes en mécanique quantique.

30.1 Analyse de l'effet du paramètre σ^2 pour un noyau gaussien

Soit un noyau gaussien k.

$$k(\mathbf{x_j}, \mathbf{x_i}) = exp\left(-\frac{1}{\sigma^2}\|\mathbf{x_j} - \mathbf{x_i}\|^2\right)$$

Générer des points x_i, par exemple entre -5 et 5.

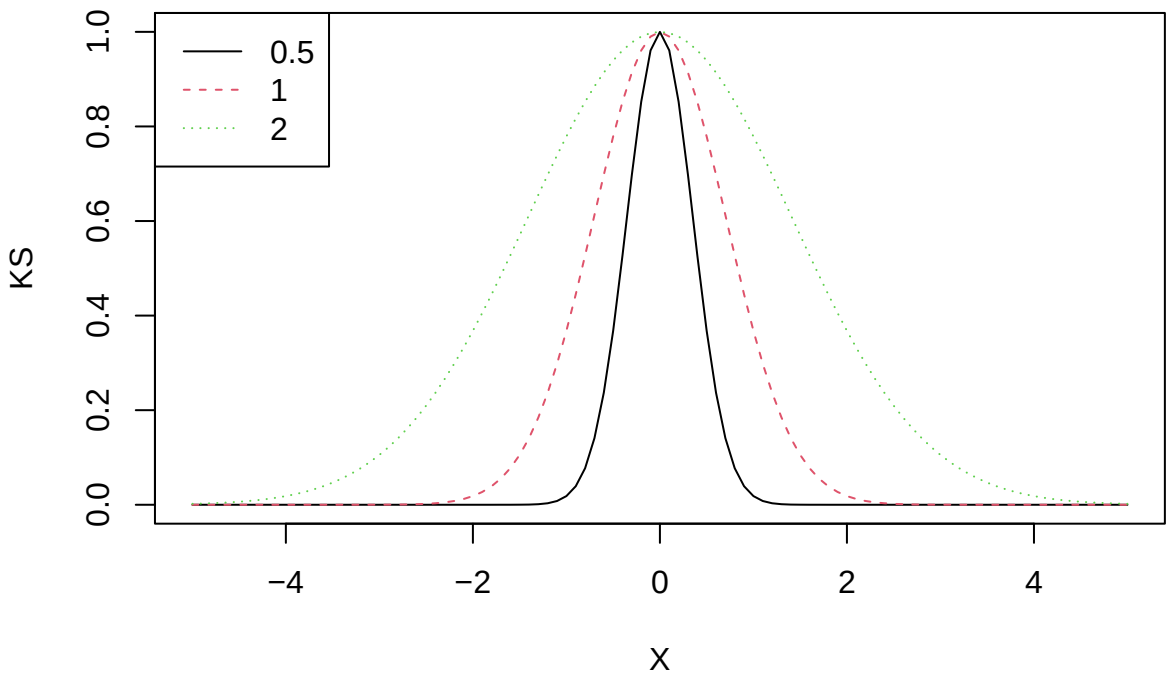
```
X <- seq(from=-5,to=5,by=0.1)
```

Générer la matrice des similarités (ou noyau) $\mathbf{K}$, résultat de l'application de la fonction k à chaque paire de points (x_i, x_j). Le faire pour σ égal à 0.5, 1 ou 2.

```
K05 <- gausskernel(X,0.5^2)
K1 <- gausskernel(X,1)
K2 <- gausskernel(X,2^2)
```

Afficher la courbe du noyau en 0, c'est-à-dire $k(0, x)$.

```
zi <- which(X==0)
KS <- cbind(K05[zi,],K1[zi,],K2[zi,])
matplot(X,KS,type='l',lty=1:3,col=1:3)
legend("topleft", legend = c('0.5','1','2'), col=1:3, lty=1:3)
```

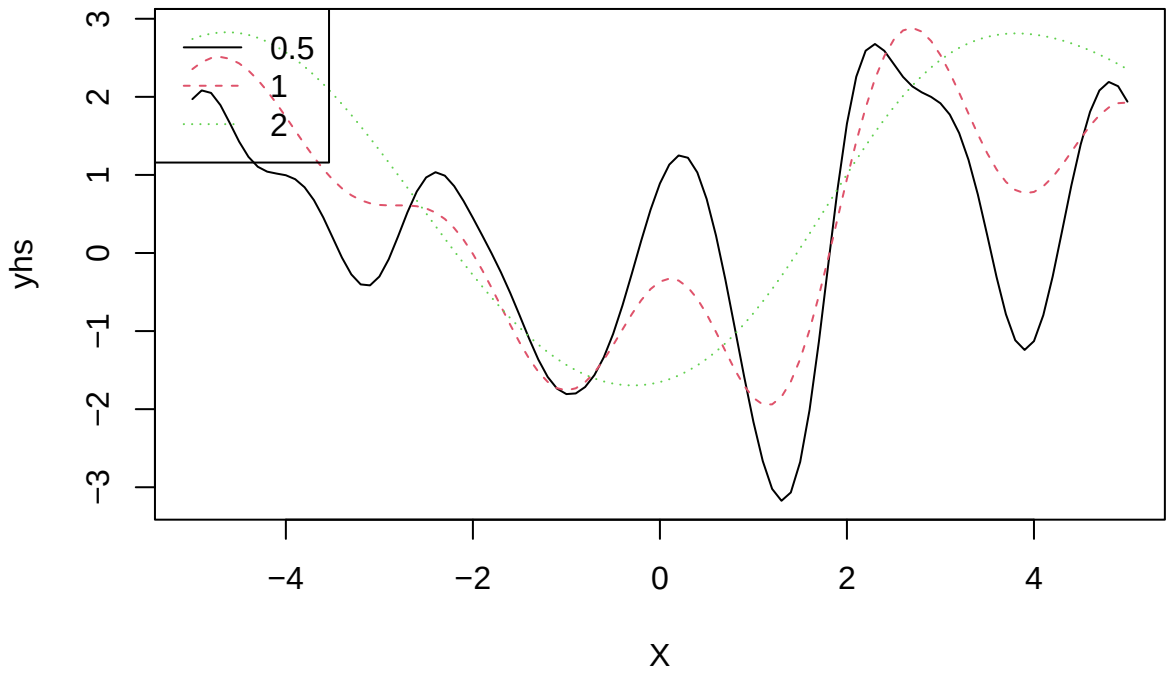


Afficher la courbe obtenue par une combinaison linéaire aléatoire des similarités entre une observation x et l'ensemble des observations x_i :

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x_i})$$

Comparer le type de courbes obtenues pour les différentes valeurs de σ.

```
alphas=runif(length(X),min=-1,max=1)
yh05 <- K05 %*% alphas
yh1 <- K1 %*% alphas
yh2 <- K2 %*% alphas
yhs <- cbind(yh05,yh1,yh2)
matplot(X,yhs,type='l',lty=1:3,col=1:3)
legend("topleft", legend = c('0.5','1','2'), col=1:3, lty=1:3)
```



30.2 Régression ridge à noyau sur un petite exemple synthétique

Soit un jeu de données synthétique construit à partir de la fonction $cos(x)$ appliquée aux points $\{0, \pi/8, 2\pi/8, 3\pi/8, 4\pi/8\}$

```
X <- pi/8 * seq(0,4)
y <- cos(X)
```

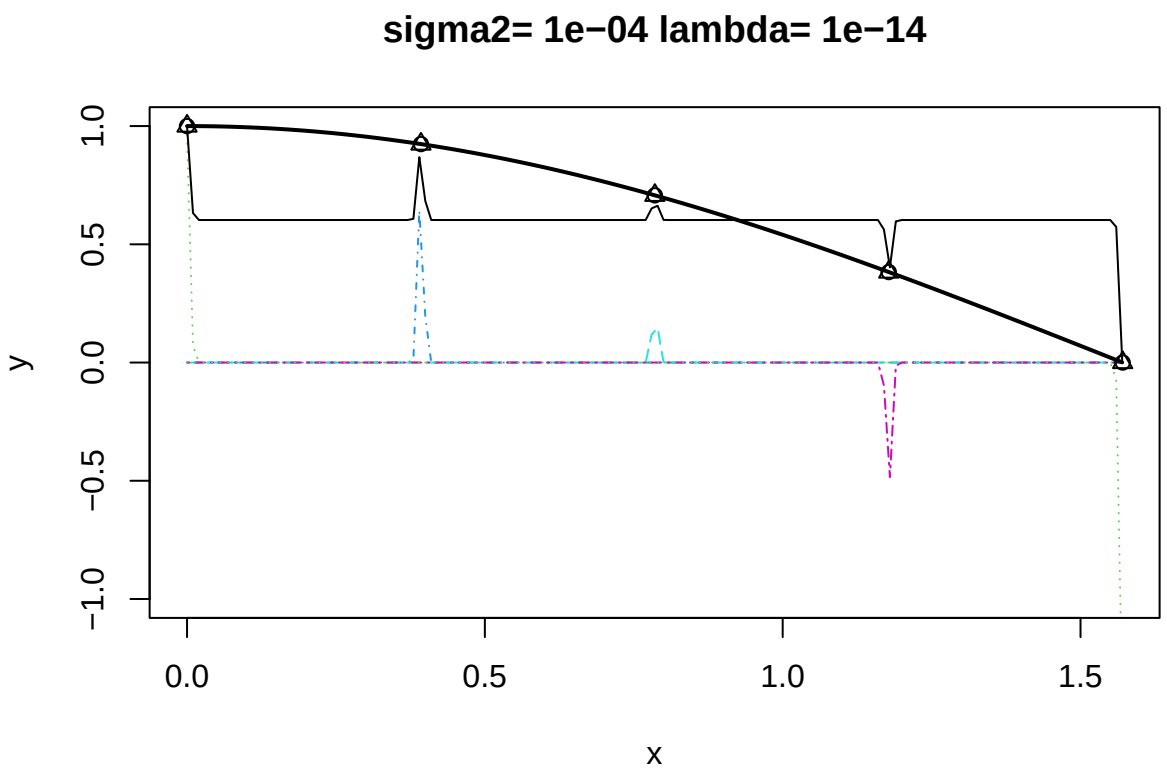
Apprendre un modèle par régression ridge à noyau sur ce jeu de données. Dans un premier temps :

- considérer le jeu de données non bruité
- fixer l'hyper-paramètre de régularisation λ presque à zéro (e.g., 10^{-14})
- comparer différentes valeurs de l'hyper-paramètre σ qui contrôle le rayon du noyau gaussien (e.g., $0.01, 0.5, 10^4$)
- Afficher à chaque fois : le jeu de données, la courbe théorique, la courbe prédite, les courbes gaussiennes centrées sur chaque observation du jeu d'entraînement et dont la combinaison linéaire est la courbe prédite.

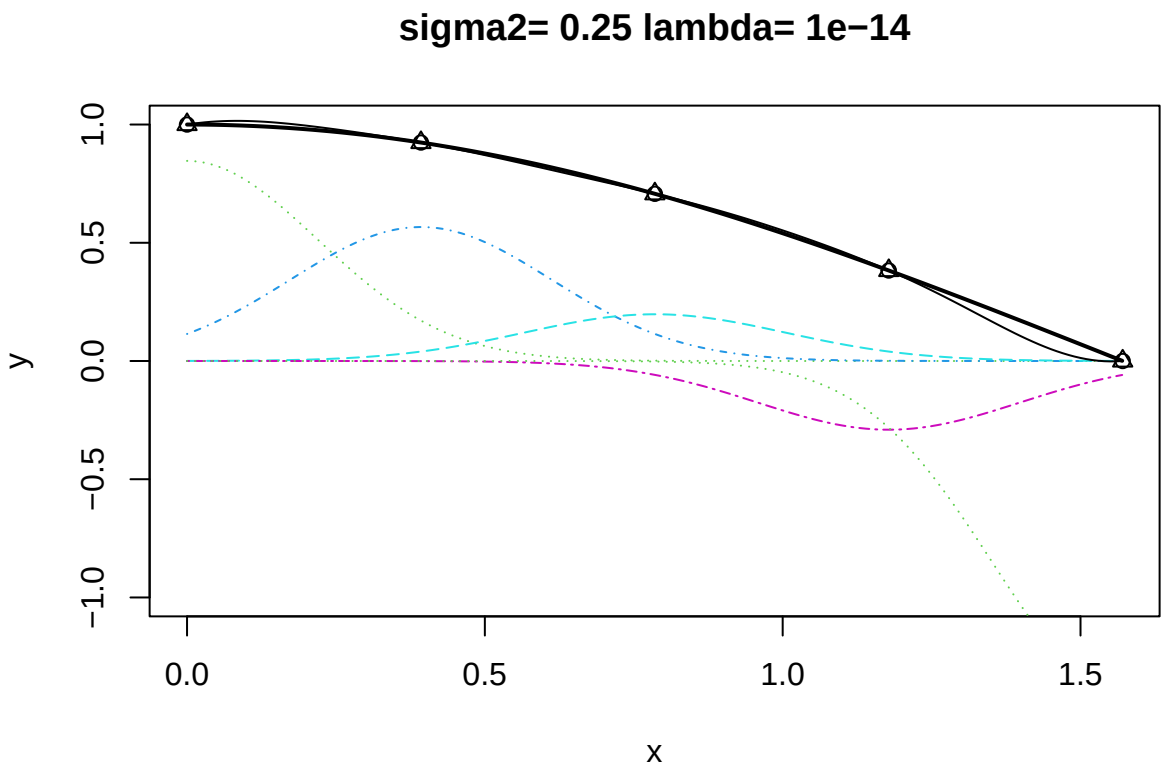
```
plotEachGaussian <-
function(X,y,km)
{
  xplt <- seq(0,pi/2,by=0.01)

  # plot the individual gaussian associated with each observation in X
  # we use the code from predict.krr
  newdata <- as.matrix(xplt)
  newdata <- scale(newdata,center=attr(km$X,"scaled:center"),
                           scale=attr(km$X,"scaled:scale"))
```

```
  n <- nrow(km$X)
  nn <- nrow(newdata)
  K <- gausskernel(rbind(newdata,km$X),sigma2=km$sigma2)[1:nn,(nn+1):(nn+n)]
  matplot(xplt,sweep(K,2,km$coef,'*'),type='l',lty=2:n+1,col=2:n+1,
          ylim=c(-1,1),
          xlab='x',ylab='y', main=paste('sigma2=',km$sigma2,'lambda=',km$lambda))

  # plot the true function and the dataset
  points(xplt,cos(xplt),type='l',lty=1,col=1,lwd=2)
  points(X,y)

  # plot the prediction for the training set
  points(x=X,y=km$yh,pch=2)

  # plot the prediction for new points
  yh <- predict(km,xplt)
  points(xplt,yh,type='l',lty=1, col=1, lwd=1)
}
```

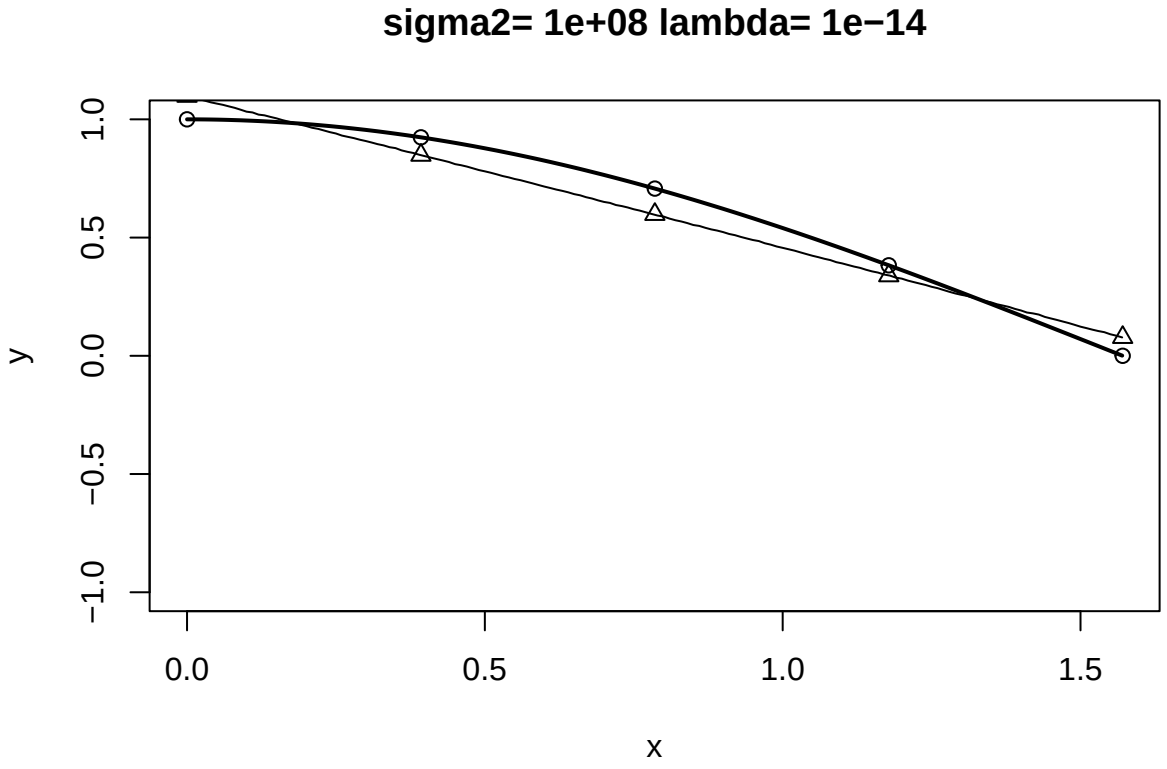
```
km <- krr(X,y,sigma2=0.01^2,lambdas=c(10^(-14)))
plotEachGaussian(X,y,km)
```




```
km <- krr(X,y,sigma2=0.5^2,lambdas=c(10^(-14)))
plotEachGaussian(X,y,km)
```

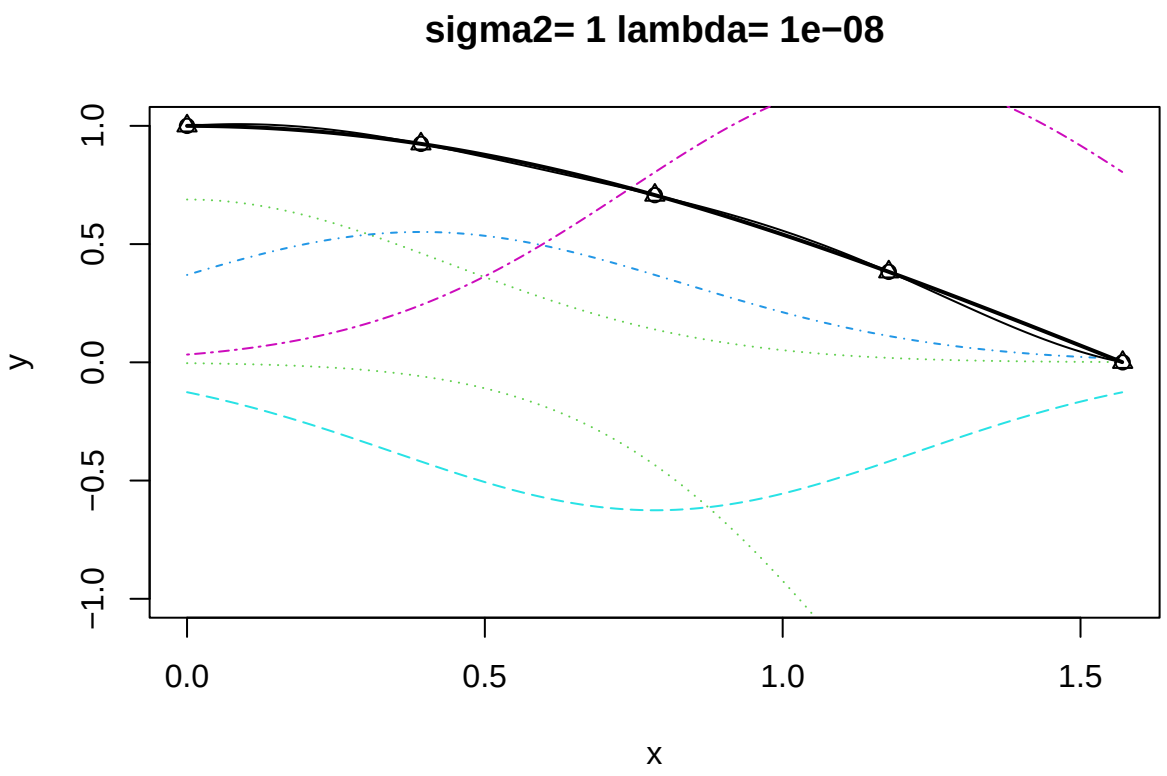



```
km <- krr(X,y,sigma2=(10^4)^2,lambdas=c(10^(-14)))
plotEachGaussian(X,y,km)
```



Ensuite, vérifier le bon fonctionnement de l'approche proposée en cours qui fixe σ^2 au nombre de dimensions du jeu de données et qui détermine la valeur de λ par validation croisée un contre tous. Discuter les rôles complémentaires des hyperparamètres σ^2 et λ.

```
km <- krr(X,y)
plotEachGaussian(X,y,km)
```



De nombreuses expériences complémentaires sont intéressantes à mener (comme étudier l'effet de l'ajout de bruit, etc.).

Chapitre 31

Références

Hastie, Trevor. 2020. “Ridge Regularization : An Essential Concept in Data Science.” *Technometrics* 62 (4) : 426–33.

Huang, Guang-Bin, Qin-Yu Zhu, and Chee-Kheong Siew. 2006. “Extreme Learning Machine : Theory and Applications.” *Neurocomputing* 70 (1-3) : 489–501.

Rupp, Matthias. 2015. “Machine Learning for Quantum Mechanics in a Nutshell.” *International Journal of Quantum Chemistry* 115 (16) : 1058–73.