UGA
UFR IM²AG Université Grenoble Alpes

LABORATOIRE JEAN KUNTZMANN
MATHÉMATIQUES APPLIQUÉES - INFORMATIQUE

Rapport de Stage du Master 1 Statistique et Sciences de Données (SSD) :
CHOIX DU PARAMETRE DE PENALISATION DANS LES MODELES LINEAIRES GENERALISES EN GRANDE DIMENSION

Kodjo Mawuena AMEKOE

Du 04 Mai 2020 au 31 Août 2020
au laboratoire Jean Kuntzmann sous la direction de :

- **Mme Sana LOUHICHI**
- **M. Didier GIRARD**
- **M. Karim BENHENNI**

Le 31 août 2020

Résumé

La sélection ou la validation de modèle en apprentissage statistique se base généralement sur la capacité de modélisation pour les raisons d'interprétation des données ou de précision de prédiction des modèles en compétition. S'agissant de la prédiction, il est important de faire une estimation des erreurs pour les données futures. Pour les régressions pénalisées comme LASSO, Ridge et Elastic-net, l'estimation de cette erreur passe souvent par l'observation de sa courbe en fonction du paramètre de lissage ou pénalisation généralement noté λ. La méthode de validation croisée (cross-validation en anglais, qui consiste à partager l'échantillon d'entrainement en plusieurs parties appelées folds) est une technique simple et intuitive permettant d'estimer l'erreur prédiction pour les nouvelles données. En grande dimension c'est-à-dire dans le cas où le nombre de variables explicatives (p) est du même ordre ou plus grand que le nombre d'observations (n), cette technique souffre d'un biais énorme quand le nombre de folds est petit. Ainsi motivé par le plus faible biais de la forme la plus extrême de validation croisée, le Leave-One-Out [1] (LO), l'article "A scalable estimate of the extra-sample prediction error via approximate leave-one-out" [RM18] propose l'Approximate Leave-One-Out (ALO) pour une large classe des estimateurs régularisés. Cette approximation réduit considérablement le temps, les ressources de calcul et avec quelques hypothèses sur les données, les résultats obtenus convergent vers ceux du LO avec une grande probabilité quand n, p deviennent grands. Dans ce rapport, nous discutons les résultats obtenus dans [RM18] et nous refaisons une grande partie de leurs simulations, ensuite nous ferons des illustrations non incluses dans [RM18] grâce aux données simulées et nous finirons par une perspective sur les travaux futurs.

1. cas particulier de la validation croisée où le nombre de fold k est le égale au nombre d'observations. C'est-à-dire que l'on apprend sur $n-1$ observations puis on valide le modèle sur la n-ième observation et l'on répète cette opération n fois

Remerciements

Je tiens à remercier l'ensemble de l'équipe du laboratoire Jean Kuntzmann pour son accueil et son aide pour le bon déroulement de mon stage.

Je remercie particulièrement Sana LOUHICHI pour avoir accepté de m'encadrer, de suivre mon évolution durant ce stage.

Je remercie également Didier GIRARD, mon co-encadrant de stage, d'avoir pris le temps de répondre à mes questions, mes inquiétudes et de m'avoir fourni la documentation nécessaire pour ce stage.

Je remercie Karim BENHENNI, co-encadrant, pour ses remarques et suggestions tout au long de ma mission.

Pour finir je remercie Adeline LECLERCQ-SAMSON et Rémy DROUILHET responsables du Master SSD ainsi que toute l'équipe pédagogique du Master SSD pour leurs enseignements m'ayant permis d'effectuer ce stage.

Table des matières

Chapitre 1

Introduction générale

1.1 Présentation du LJK

Le laboratoire Jean Kuntzmann (LJK) est un laboratoire de Mathématiques Appliquées et d'Informatique qui doit son nom à Jean Kuntzmann(1912-1992), un pionnier de l'informatique et des mathématiques appliquées à Grenoble. Il regroupe des équipes de cultures assez différentes : mathématiciens, numériciens, spécialistes de l'informatique graphique, du traitement d'images et de vision par ordinateur. Le LJK joue également un rôle d'infertace vers d'autres disciplines : nanosciences, biologie, mathématiques financières, synthèse d'images et sciences sociales.

Il est structuré en 3 départements :

— Géométrie-Image
— Algorithmes, Modèles, Analyse et Calcul
— Données et Aléatoire : Théorie et Applications (D.A.T.A.)

Le département D.A.T.A est structuré en 6 équipes et regroupe des chercheurs qui travaillent en probabilités, statistiques, mathématiques financières et traitement du signal et de l'image. Les équipes du département D.A.T.A font beaucoup d'actions en commun plus que les équipes des deux autres départements. C'est au sein de ce département plus précisement dans l'équipe IPS (Inférence Processus Stochastiques) que j'ai effectué le stage. Les membres de l'équipe IPS font des recherches sur les théories de dépendance, les systèmes de particules en interaction, les théories de processus stochastiques et l'estimation non-paramétrique.

1.2 Objectifs et contexte du stage

1.2.1 Objectifs du stage

Ma mission pour ce stage qui a duré 3 mois et demi a été de comprendre le contexte du travail de [RM18] sur l'approximation du leave-one-out, discuter les résultats, reproduire et faire d'autres simulations pour les configurations $n < p$, $n > p$, $n = p$ où n est le nombre d'observations et p le nombre de variables expliactives, trouver des perspectives pour les travaux futurs.

Outre cette mission, mon objectif personnel est de prendre connaissance des problèmes de statistique en grande dimension, comprendre l'importance des méthodes de regression pénalisée ainsi que de la validation croisée. Etre en mesure de lire et comprendre un article ou une publication à caractère statistique ou apprentissage statistique est également un objectif de ce stage. Enfin développer les compétences en programmation est une partie intégrante de l'objectif de ce stage.

1.2.2 Contexte

Considérons le jeu de données $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, où $x_i \in \mathbb{R}^p$ et $y_i \in \mathbb{R}$. Dans plusieurs applications, on considère que les observations sont indépendantes et identiquement distributées (iid) et issues de la distribution conjointe $q(y_i|x_i^\top \beta^*)p(x_i)$ avec q la fonction densité conditionelle de la variable aléatoire Y_i en y_i, β^* le vrai vecteur des coefficients de régression. $q(y_i|x_i^\top \beta^*)$ désigne la densité contidionnelle de y_i connaissant $x_i^\top \beta^*$ et $\top$ désigne la transposée d'un vecteur ou d'une matrice.

Généralement estimer β^* revient à résoudre le problème d'optimisation :

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left[\sum_{i=1}^{n} l(y_i | x_i^\top \beta) + \lambda r(\beta) \right] \tag{1.1}$$

l est souvent appelée la fonction coût ou de perte, plus précisement -$l(y_i|m)$ désigne dans ce travail le log-vraisemblance de y_i sachant m, où m est la moyenne ou une fonction monotone de la moyenne. r est la fonction de pénalité (régularisateur). Pour une nouvelle observation (x_{new}, y_{new}) issue de la distribution $q(y|x^\top \beta^*)p(x)$, indépendante de $\mathcal{D}$, on s'interesse à l'estimation de l'erreur de prédiction définie comme suit :

$$Err_{extra} = E[\phi(y_{new}, x_{new}^\top \hat{\beta}) | \mathcal{D}] \tag{1.2}$$

où ϕ est une fonction qui mesure la distance entre y_{new} et $x_{new}^\top \hat{\beta}$. Un choix standard de ϕ est $l(y|x^\top \beta)$. Comme Err_{extra} dépend de la distribution conjointe mais rarement connu de (x_i, y_i), il revient de l'estimer à partir de $\mathcal{D}$.

Il y a eu beaucoup d'apport pour les problèmes de statistique en grande dimension. Cependant le problème de l'estimation de l'erreur de prédiction n'a pas été soigneusement étudié en général et, par conséquent, les questions des techniques existantes et de leurs remèdes n'ont pas été explorées.

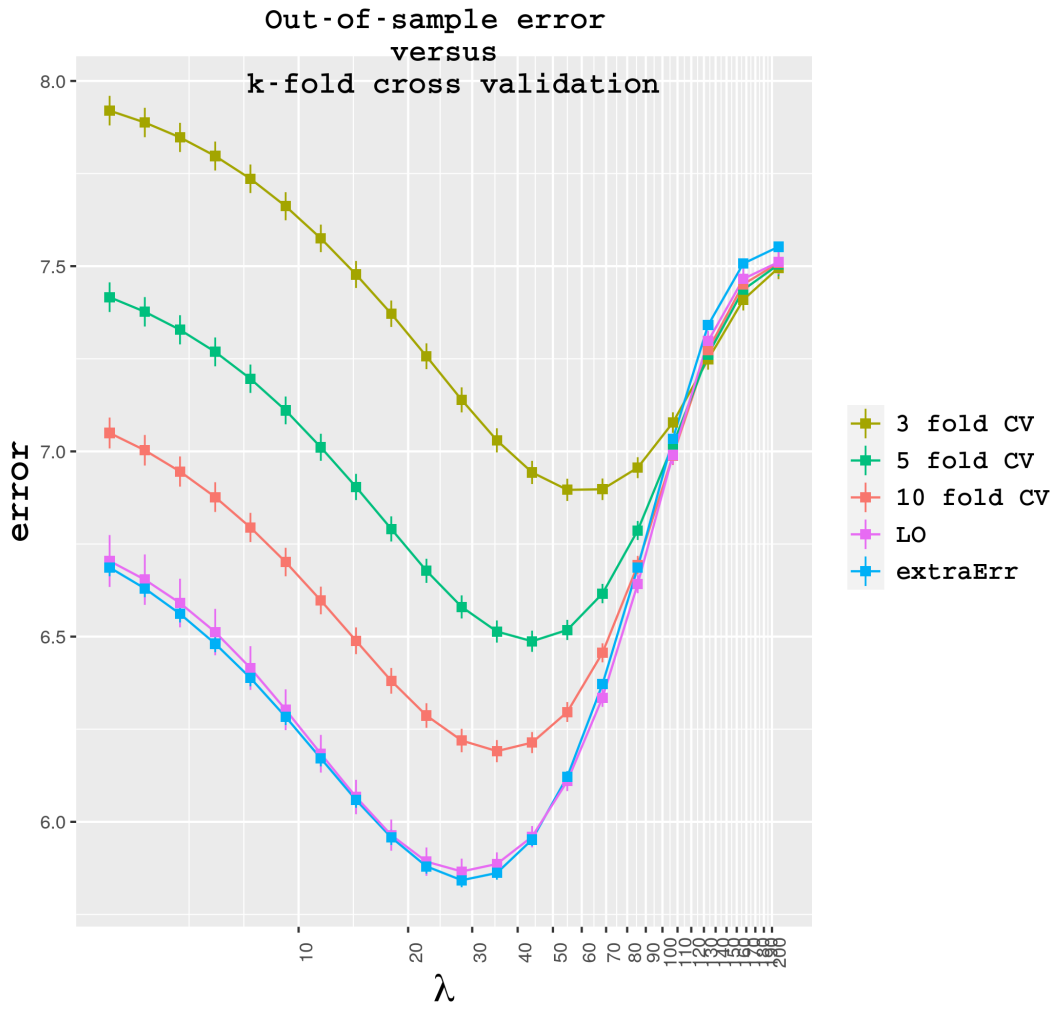


FIGURE 1.1: Comparaison de la validation croisée à 3,5,10 parties (fold) et du LO avec l'erreur de prédiction.
Cette figure illustre la capacité de la validation croisée à estimer l'erreur de prédiction en fonction de λ (20 différentes valeurs) pour une régression linéaire avec la pénalité LASSO.

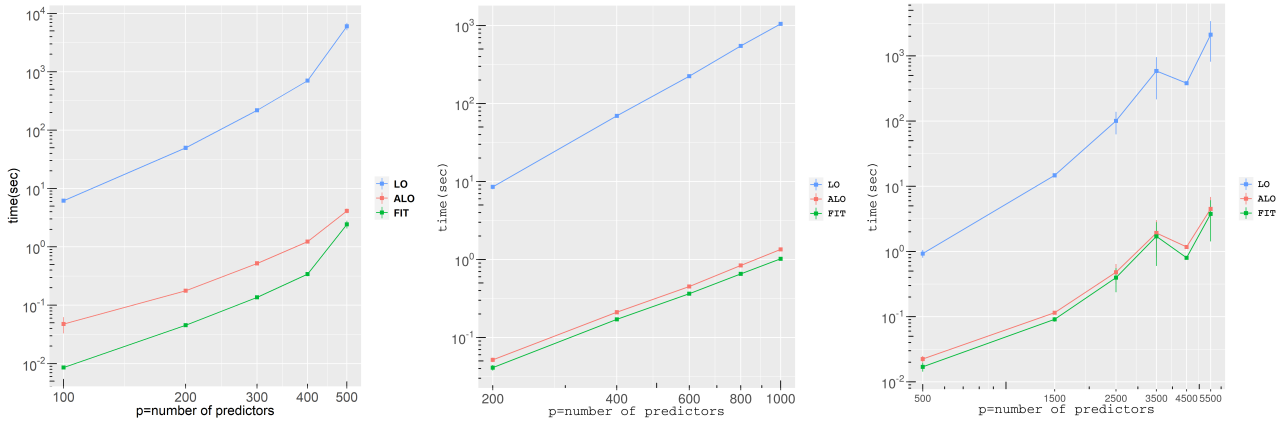
On se place dans un cadre en grande dimension où nous avons considéré notamment (comme dans l'exemple de la FIGURE 1.1) les dimensions (p, n, k)=(1000, 250, 50). La variable cible $y \sim \mathcal{N}(X\beta^\star, \sigma^2 I)$ avec $X \in \mathbb{R}^{n\times p}$. k est le nombre de coefficients non nuls de $\beta^\star$ qui sont tous fixés à $\frac{1}{3}$. Les lignes $x_i^\top$ de X sont indépendantes et suivent une loi $\mathcal{N}(0, I)$
Pour une nouvelle observation $y_{new} \sim \mathcal{N}(x_{new}\beta^\star, \sigma^2)$ avec $x_{new} \sim \mathcal{N}(0, I)$, l'erreur de prédiction est $Err_{extra} = \sigma^2 + \|\beta^\star - \hat{\beta}\|_2^2$.
En effet :

$$\begin{aligned} Err_{extra} &= E[(y_{new} - x_{new}^T\hat{\beta})^2|\mathcal{D}] \\ &= E[(x_{new}^T\beta^\star + \epsilon - x_{new}^T\hat{\beta})^2|\mathcal{D}] \\ &= E[(x_{new}^T\beta^\star - x_{new}^T\hat{\beta})^2|\mathcal{D}] + E(\epsilon^2) \\ &= E[(\beta^\star - \hat{\beta})^\top x_{new}x_{new}^T(\beta^\star - \hat{\beta})|\mathcal{D}] + \sigma^2 \\ &= (\beta^\star - \hat{\beta})^\top E[x_{new}x_{new}^T](\beta^\star - \hat{\beta}) + \sigma^2 \\ &= (\beta^\star - \hat{\beta})^\top \Sigma(\beta^\star - \hat{\beta}) + \sigma^2 \end{aligned}$$

Σ est la matrice de covariance de l'entrée x_{new}. Ici $\Sigma = I$, ce qui conduit à $Err_{extra} = \sigma^2 + \|\beta^\star - \hat{\beta}\|^2$.

Nous avous pris la variance $\sigma^2 = 2$. Pour chaque valeur de λ, les valeurs obtenues sont des moyennes faites sur 500 échantillons. Les barres d'erreur représentent une erreur standard.
On peut remarquer que la technique de validation croisée souffre d'un biais large sauf si le nombre de folds (parties) est grand. Ce biais est dû au fait qu'en grande dimension la partie des données retirée dans la phase d'apprentissage peut avoir un effet majeur sur la solution de (1.1), voir [DMM11].
Dans notre travail on se focalise sur la forme extrême (n-fold) de la validation croisée, appelée validation croisée complète ou Leave-One-out (LO) qui d'après la FIGURE 1.1 estime le mieux l'erreur Err_{extra}. En se basant sur le fait que n, p sont grands mais le rapport $\frac{n}{p}$ est fixe, on fait une approximation du LO pour les régulateurs lisses comme non lisses. Cette approximation nommée ALO est efficace sur le plan calcul puisque qu'elle nécessite une seule résolution du problème d'optimisation (1.1) . En plus d'obtenir $\hat{\beta}$, ALO nécessite une inversion de matrice et deux multiplications de matrices.

Ci-dessous est illustré le temps de calcul de $\hat{\beta}$, ALO et LO pour les différentes configurations $n > p$, $n = p$ et $n < p$.



(a) Régression linéaire avec pénalisation Elastic-net (section 4.1) pour $\frac{n}{p} = 5$

(b) Régression logistique avec pénalisation LASSO (section 4.2) pour $\frac{n}{p} = 1$

(c) Régression Poisson avec pénalisation Elastic-net pour $\frac{n}{p} = \frac{1}{10}$

FIGURE 1.2: Comparaison du temps de calcul de $\hat{\beta}$(FIT), ALO et LO

On peut voir que le temps de calcul de LO est très grand devant celui de ALO même si cette dernière en plus du calcul de $\hat{\beta}$ nécessite quelques opérations. Les temps de calculs représentés sont des moyennes faites sur 5 essais pour les graphes (a) , (c) et 10 essais pour le graphe (b). Plus d'informations sur ces simulations sont données dans le chapitre 4.

Nous soulignons que malgré la réduction majeure du temps de calcul, les résultats théoriques et empiriques présentés révèlent que dans les contextes de grande dimension ALO présente une bonne approximation de LO.

Nous commençons par citer quelques travaux remarquables dans le contexte de ce stage.

1.3 Travaux antérieurs

L'estimation de l'Erreur de prédiction Err_{extra} à partir de l'échantillon $\mathcal{D}$ a été étudiée pendant les 50 dernières années. Les méthodes les plus connues sont la validation croisée à K parties (K-fold CV) [Gei75], validation croisée généralisée (GCV) [Gea79], le bootstrap [Efr83]. Cependant en grande dimension, l'utilisation du bootstrap ou LO est très coûteuse en terme ressources de calculs et les approches moins exigeantes comme la validation croisée à 5, 10 parties souffrent d'un biais énorme. Comme approches efficaces en termes de calcul, on peut citer les extensions de la GCV aux modèles non-linéaires et classificateurs avec pénalisation Ridge [XW96], mais la performance théorique de ces méthodes en grande dimension reste inconnue sauf pour la régression linéaire simple avec la pénalisation Ridge [Li86].

Tout récemment [OK16] et [Mou18] ont étudié l'estimation de Err_{extra} en grande dimension pour la régression linéaire avec pénalisation LASSO. Cependant les résultats obtenus sont valables seulement quand la matrice X possède des entrées iid et si la distribution empirique des coefficients de régression converge vers une distribution avec un moment d'ordre deux borné.

Ainsi les résultats discutés dans ce travail concernent un large groupe de régularisateurs : LASSO [Tib96], Elastic-net [ZH05], Bridge [lEFF93], et d'autres encore. De plus les résultats présentés ici restent valables pour les matrices X sans structure iid et sans hypothèse sur la distribution des coefficients.

1.4 Notations

Pour la suite du document, les notations suivantes sont utilisées : $x_i^\top \in \mathbb{R}^{1\times p}$ désigne la ième ligne de la matrice $X \in \mathbb{R}^{n\times p}$. $y_{/i} \in \mathbb{R}^{(n-1)\times 1}$ désigne y sans son ième élément y_i et $X_{/i} \in \mathbb{R}^{(n-1)\times p}$ désigne X sans la ligne $x_i^\top$. Le vecteur $a \odot b$ désigne le produit par éléments des vecteurs a et b. $a < b$ est utilisé pour l'inégalité par éléments de deux vecteurs a et b. $|a|$ représente le vecteur obtenu en prenant la valeur absolue de chaque élément de a. Pour l'ensemble $S \subset \{1, 2, 3, ..., p\}$, X_S représente la restriction de la matrice X aux colonnes indexées par S. $x_{i,S} \in \mathbb{R}^{1\times|S|}$ l'observation $x_i^\top$ restreinte aux colonnes indexées par S. a_i ou $[a]_i$ désigne l'ième élément d'un vecteur a donné et diag$[a]$ désigne la matrice diagonale dont les éléments diagonaux sont les composantes de a. De plus nous avons :
$\dot{\phi}(y,z) = \dfrac{\partial\phi(y,z)}{\partial z}$,$\dot{l}_i(\beta) = \dfrac{\partial l(y_i|z)}{\partial z}|_{z=x_i^\top\beta}$, $\ddot{l}_i(\beta) = \dfrac{\partial^2 l(y_i|z)}{\partial^2 z}z^2|_{z=x_i^\top\beta}$,
$\dot{l}_{/i}(.) = [\dot{l}_1(.), ..., \dot{l}_{i-1}(.), \dot{l}_{i+1}(.), \dot{l}_n(.)]^\top$ et $\ddot{l}_{/i}(.) = [\ddot{l}_1(.), ..., \ddot{l}_{i-1}(.), \ddot{l}_{i+1}(.), \ddot{l}_n(.)]^\top$.

1.5 Outils

Pour le traitement et simulations, nous avons utilisé le logiciel R version 3.6.3 installé sur une machine à 4 cœurs avec système linux debian 10. Les librairies principalement utilisées sont : glmnet pour les régressions pénalisées, alocvBeta pour calculer l'approximate leave-one-out (ALO). Nous avons également utilisé la librairie ggplot2 pour la visualisation et dplyr pour quelques manipulations de données.

Pour les traitements de texte et le rapport, nous avons utilisé Latex avec l'interface TeXstudio.

Chapitre 2

Approximation de la validation croisée complète (ALO)

2.1 Fonctions de perte et pénalité lisses

La formule de l'estimation Leave-One-Out (LO) est donnée par :

$$\text{LO} = \frac{1}{n}\sum_{i=1}^{n}\phi(y_i, x_i^\top \hat{\beta}_{/i}) \tag{2.1}$$

où

$$\hat{\beta}_{/i} = \underset{\beta\in\mathbb{R}^p}{\arg\min}\left[\sum_{j\neq i} l(y_j|x_j^\top\beta) + \lambda r(\beta)\right] \tag{2.2}$$

désigne les estimations de coefficients après le retrait de la ième observation (leave-i-out). De façon classique il faut résoudre n fois ce problème d'optimisation, ce qui est très couteux surtout en grande dimension.
En supposant que $\hat{\beta}_{/i}$ est proche de $\hat{\beta}$, on utilise la méthode de Newton ce qui permet d'obtenir une approximation de $\hat{\beta}_{/i}$ comme suit :

$$\tilde{\beta}_{/i} = \hat{\beta} + \left[\sum_{j\neq i} x_j x_j^\top \ddot{l}(y_j|x_j^\top\hat{\beta}) + \lambda diag[\ddot{r}(\hat{\beta})]\right]^{-1} x_i\dot{l}(y_i|x_i^\top\hat{\beta})$$

Les étapes pour l'obtention de l'expression ci-dessus sont données dans l'Annexe A.1.1. On peut remarquer que le second membre de cette équation contient toujours l'observation retirée, ce qui entraînerait n inversions de matrices. Pour résoudre ce problème, on utilise l'identité de la matrice de Woodbury (ou la formule d'inversion de Sherman-Morrison) ce qui donne :

$$\tilde{\beta}_{/i} = \hat{\beta} + \frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\left[\frac{J^{-1}x_i\ddot{l}_i(\hat{\beta})}{1 - x_i^\top J^{-1}x_i\ddot{l}_i(\hat{\beta})}\right] \tag{2.3}$$

où

$$J = \left[\sum_{j=1}^{n} x_j x_j^\top \ddot{l}(y_j|x_j^\top\hat{\beta}) + \lambda diag[\ddot{r}(\hat{\beta})]\right].$$

On parvient finalement au ALO définie par :

$$\text{ALO} = \frac{1}{n}\sum_{i=1}^{n}\phi(y_i, x_i^\top\tilde{\beta}_{/i}) = \frac{1}{n}\sum_{i=1}^{n}\phi\left(y_i, x_i^\top\hat{\beta} + \left(\frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\right)\left(\frac{H_{ii}}{1-H_{ii}}\right)\right) \tag{2.4}$$

avec

$$H = X(\lambda\text{diag}[\ddot{r}(\hat{\beta})] + X^\top\text{diag}[\ddot{l}(\hat{\beta})]X)^{-1}X^\top\text{diag}[\ddot{l}(\hat{\beta})]. \tag{2.5}$$

Les étapes pour l'obtention du ALO sont données dans l'Annexe A.1.2.

Algorithme 1 : Estimation du risque de prédiction avec ALO pour les fonctions de perte et pénalité lisses

---

Entrée : $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$
Sortie : Err_{extra}

1. Calculer $\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left[\sum_{i=1}^{n} l(y_i|x_i^\top \beta) + \lambda r(\beta)\right]$
2. Calculer $H = X(\lambda \text{diag}[\ddot{r}(\hat{\beta})] + X^\top \text{diag}[\ddot{l}(\hat{\beta})]X)^{-1}X^\top \text{diag}[\ddot{l}(\hat{\beta})]$
3. L'estimation de Err_{extra} via ALO est donnée par : $\frac{1}{n}\sum_{i=1}^{n} \phi\left(y_i, x_i^\top \hat{\beta} + \left(\frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\right)\left(\frac{H_{ii}}{1-H_{ii}}\right)\right)$

L'algorithme montre comment obtenir l'estimation de Err_{extra} avec ALO.

2.2 Fonction de pénalité non lisse

L'approche utilisée ci-dessus nécessite des fonctions de perte et pénalité lisses [1], ce qui n'est pas généralement le cas dans plusieurs applications par exemple LASSO. Avec la régularisation LASSO, le problème (1.1) s'écrit :

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left[\sum_{i=1}^{n} l(y_i|x_i^\top \beta) + \lambda \|\beta\|_1\right] \tag{2.6}$$

La norme $\|.\|_1$ n'étant pas deux fois différentiable, il est considéré une approximation lisse introduite par [MGR07] :

$$r^\alpha(\beta) = \sum_{i=1}^{p} \frac{1}{\alpha}\left(\log(1 + e^{\alpha\beta_i}) + \log(1 + e^{-\alpha\beta_i})\right) = \sum_{i=1}^{p} r^\alpha(\beta_i)$$

vérifiant $\lim_{\alpha \to +\infty} r^\alpha(\beta) = \|\beta\|_1$.

Ainsi on considère :

$$\hat{\beta}^\alpha = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left[\sum_{i=1}^{n} l(y_i|x_i^\top \beta) + \lambda r^\alpha(\beta)\right] = h_\alpha(\beta). \tag{2.7}$$

Ce qui a permis de définir :

$$\text{ALO}^\alpha = \frac{1}{n}\sum_{i=1}^{n} \phi\left(y_i, x_i^\top \hat{\beta^\alpha} + \left(\frac{\dot{l}_i(\hat{\beta^\alpha})}{\ddot{l}_i(\hat{\beta^\alpha})}\right)\left(\frac{H_{ii}^\alpha}{1-H_{ii}^\alpha}\right)\right) \tag{2.8}$$

Pour cette approximation de la norme $\|.\|_1$, on obtient le théorème énoncé ci-dessous. Ce théorème donne principalement la limite de ALO^α quand α tend vers l'infini, quantité qui est utilisée comme approximation du LO pour le régularisateur LASSO.

1. Nous utilisons lisse pour désigner une fonction au moins deux fois différentiables

Théorème 1 *Si :*

— $\hat{\beta}$ *est le minimum global de* (1.1)

— $\hat{\beta}^\alpha$ *est le minimum global de* (2.7) *pour toute valeur de* α

— $\ddot{l}(y|x^\top\beta)$ *vue comme fonction de* β *est continue*

— *l'inégalité* $\|\hat{g}_{S^c}\| < 1$ *est vérifiée, avec* $\hat{g}$ *un sous-gradient de* $\|\beta\|_1$ *en* $\hat{\beta}$

alors

$$\lim_{\alpha\to+\infty} ALO^\alpha = \frac{1}{n}\sum_{i=1}^{n} \phi\left(y_i, x_i^\top\hat{\beta} + \left(\frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\right)\left(\frac{H_{ii}}{1-H_{ii}}\right)\right) \tag{2.9}$$

avec $H = X_S(X_S^\top diag[\ddot{l}(\hat{\beta})]X_S)^{-1}X_S^\top diag[\ddot{l}(\hat{\beta})]$ *et l'ensemble* $S = \{i, \hat{\beta}_i \neq 0\}$

Les deux premières hypothèses du Théorème 1 permettent de montrer que $\|\hat{\beta}^\alpha - \hat{\beta}\|_2 \to 0$ lorsque $\alpha \to \infty$. L'hypothèse de continuité permet d'avoir $\ddot{l}(\hat{\beta}^\alpha) \to \ddot{l}(\hat{\beta})$ quand $\alpha \to \infty$ et la dernière hypothèse est utilisée pour borner les composantes de $\hat{\beta}^\alpha$.
La preuve de ce théorème est donnée par [RM18]

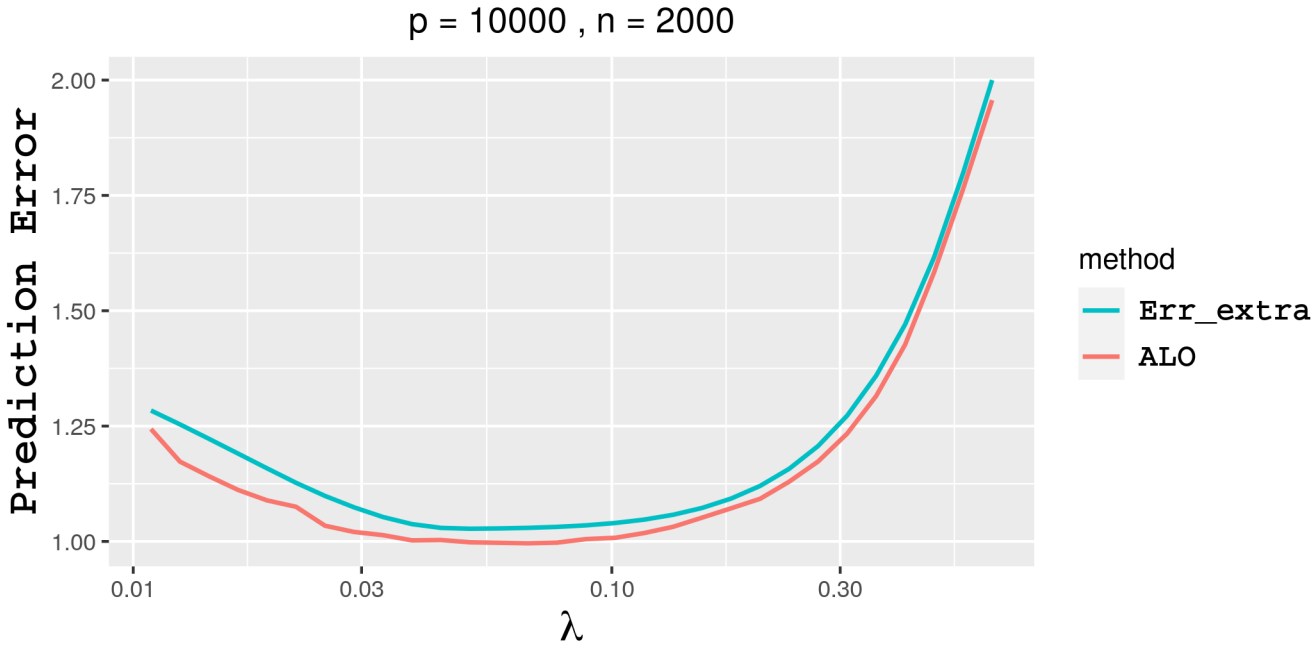


FIGURE 2.1: Comparaison de l'erreur de prédiction et ALO

La FIGURE 2.1 montre que ALO approche bien l'erreur de prédiction pour un problème de LASSO avec une regression moindre carré (linéaire).
Pour les données, on a considéré $y \sim \mathcal{N}(X\beta^\star, \sigma^2 I)$ avec $X \in \mathbb{R}^{n\times p}$ et $\sigma^2 = 1$. Les coefficients non nuls de $\beta^\star$ sont tous fixés à 1 et sont au nombre de $k = 400$. Les lignes $x_i^\top$ de X suivent une loi $\mathcal{N}(0, \Sigma)$, avec une structure de corrélation $cor(X_{ij}, X_{ij'}) = 0.3$ pour $i = 1, 2, ..., n$ et $j, j' = 1, 2, .., p$. La matrice de covariance Σ est normalisé de sorte à avoir $Var(x^\top\beta) = 1$. Les données de test, $y_{new} \sim \mathcal{N}(x_{new}\beta^\star, \sigma^2)$ avec $x_{new} \sim \mathcal{N}(0, \Sigma)$, ainsi l'erreur de prédiction est $Err_{extra} = \sigma^2 + \|\Sigma^{\frac{1}{2}}(\beta^\star - \hat{\beta})\|_2^2$.
En partant de la formule (2.9), on obtient l'expression simplifiée :

$$\text{ALO} = \lim_{\alpha\to+\infty} \text{ALO}^\alpha = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - x_i^\top\hat{\beta}}{1-H_{ii}}\right)^2 \tag{2.10}$$

avec $H = X_S(X_S^\top X_S)^{-1}X_S$.

En effet : $\lim_{\alpha\to+\infty} \text{ALO}^\alpha = \frac{1}{n}\sum_{i=1}^{n} \phi\left(y_i, x_i^\top\hat{\beta} + \left(\frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\right)\left(\frac{H_{ii}}{1-H_{ii}}\right)\right)$ et

$l_i(\beta) = \frac{1}{2}(y_i - x_i^\top \beta)^2$, $\dot{l}_i(\beta) = -(y_i - x_i^\top \beta)$ et $\ddot{l}_i(\beta) = 1$, $\phi(y_i, x_i^\top \hat{\beta}) = (y_i - x_i^\top \hat{\beta})^2$

$$\begin{aligned}
\text{ALO} &= \frac{1}{n}\sum_{i=1}^{n} \phi\left(y_i, x_i^\top \hat{\beta} + \left(\frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\right)\left(\frac{H_{ii}}{1-H_{ii}}\right)\right) \\
&= \frac{1}{n}\sum_{i=1}^{n} \left(y_i - x_i^\top \hat{\beta} - \left(\frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\right)\left(\frac{H_{ii}}{1-H_{ii}}\right)\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n} \left(y_i - x_i^\top \hat{\beta} - \left(\frac{-(y_i - x_i^\top \beta)}{1}\right)\left(\frac{H_{ii}}{1-H_{ii}}\right)\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n} \left(\frac{(y_i - x_i^\top \hat{\beta})(1-H_{ii}) + (y_i - x_i^\top \beta)H_{ii}}{1-H_{ii}}\right)^2
\end{aligned}$$

$$\text{ALO} = \frac{1}{n}\sum_{i=1}^{n} \left(\frac{y_i - x_i^\top \hat{\beta}}{1-H_{ii}}\right)^2$$

Notons que dans le Théorème 1, l'hypothèse d'unicité de $\hat{\beta}$ comme minimum global de (1.1) est une forte hypothèse. En effet en grande dimension, la condition d'unicité de la solution de LASSO n'est pas toujours vérifiée d'autant plus que la fonction à optimiser n'est pas strictement convexe. Cependant [Tib13] a discuté les cas de l'unicité de cette solution de LASSO notamment dans le cas où les variables prédictives sont continues. De même la quatrième hypothèse est également valable dans de nombreux cas avec une probabilité relative au caractère aléatoire des données [Wai09]. Même s'il arrive que cette dernière hypothèse ne soit pas vérifiée pour des problèmes spécifiques, il est établi un deuxième Théorème permettant de trouver la limite supérieure et inférieure de ALO^α à défaut de trouver la limite exacte. Ce théorème permet également de dériver l'ALO pour les régularisateurs Bridge et elastic-net.

Théorème 2 *Soient S le support (l'ensemble des coefficients non nuls) de $\hat{\beta}$ et T l'ensemble des coefficients nuls auxquels le vecteur subgradient est égal à -1 ou 1, alors*

$$x_{i,S}^T (X_S^\top \, diag[\ddot{l}(\hat{\beta})] X_S)^{-1} x_{i,S} \ddot{l}_i(\hat{\beta}) < \liminf_{\alpha\to\infty} H_{ii}^\alpha,$$

$$\limsup_{\alpha\to\infty} H_{ii}^\alpha < x_{i,S\cup T}^T (X_{S\cup T}^\top \, diag[\ddot{l}(\hat{\beta})] X_{S\cup T})^{-1} x_{i,S\cup T} \ddot{l}_i(\hat{\beta}).$$

Avec les deux inégalités du Théorème 2, on peut évaluer la précision de ALO^α avec les majorations et minorations suivantes :

$$\limsup_{\alpha\to\infty} \text{ALO}^\alpha \leq \sum_{i=1}^{n} \phi\left(y_i, x_i^\top \hat{\beta} + \left(\frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\right)\left(\frac{H_{ii}^h}{1-H_{ii}^h}\right)\right) \tag{2.11}$$

et

$$\liminf_{\alpha\to\infty} \text{ALO}^\alpha \geq \sum_{i=1}^{n} \phi\left(y_i, x_i^\top \hat{\beta} + \left(\frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\right)\left(\frac{H_{ii}^l}{1-H_{ii}^l}\right)\right) \tag{2.12}$$

avec

$$\begin{aligned}
H^l &= X_S (X_S^\top \text{diag}[\ddot{l}(\hat{\beta})] X_S)^{-1} X_S^T \text{diag}[\ddot{l}(\hat{\beta})], \\
H^h &= X_{S\cup T} (X_{S\cup T}^\top \text{diag}[\ddot{l}(\hat{\beta})] X_{S\cup T})^{-1} X_{S\cup T}^T \text{diag}[\ddot{l}(\hat{\beta})]
\end{aligned} \tag{2.13}$$

En comparant les expressions (2.11) et (2.12), on peut approcher la limite trouvée dans le Théorème 1. De plus cette approche est étendue à d'autres régularisateurs comme Bridge et Elastic-net :

— Bridge : Avec cet estimateur, le problème (1) s'écrit :

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left[\sum_{i=1}^{n} l(y_i | x_i^\top \beta) + \lambda \|\beta\|_q^q \right] \tag{2.14}$$

où $q \in \{1, 2\}$. En se basant sur l'approche utilisée pour LASSO, on considère l'approximation $r_\gamma^q(z) = \frac{1}{\gamma} \int |u|^q \psi(\frac{z-u}{\gamma}) du$ où :
— ψ est une fonction à support compact et $\psi(w) \geq 0$ pour tout w
— $\int \psi(w) dw = 1$ et $\psi(0) > 0$
— ψ est infiniment différentiable et symétrique autour de 0 dans $\mathbb{R}$.
Cette approximation de $\|\beta\|_q^q$ est infiniment différentiable et $|r_\gamma^q(z) - |z|^q| \to 0$ quand $\gamma \to 0$. On obtient ALO$^\gamma$ dont la limite est :

$$\lim_{\gamma \to 0} \text{ALO}^\gamma = \frac{1}{n} \sum_{i=1}^{n} \phi \left(y_i, x_i^\top \hat{\beta} + \left(\frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})} \right) \left(\frac{H_{ii}}{1 - H_{ii}} \right) \right) \tag{2.15}$$

où

$$H = X_S \left[X_S^\top \text{diag}[\ddot{l}(\hat{\beta})] X_S + \lambda \text{diag}[\ddot{r}_S^q(\hat{\beta})] \right]^{-1} X_S^\top \text{diag}[\ddot{l}(\hat{\beta})] \tag{2.16}$$

avec $S = \{i, \hat{\beta}_i \neq 0\}$ et $\ddot{r}^q(u) = q(q-1)|u|^{q-2}$ pour $u \neq 0$.
— Elastic-net
Pour cet estimateur , nous avons :

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left[\sum_{i=1}^{n} l(y_i | x_i^\top \beta) + \lambda_1 \|\beta\|_2^2 + \lambda_2 \|\beta\|_1 \right]. \tag{2.17}$$

En considérant l'approximation lisse de la norme $\|.\|_1$ établie pour LASSO, on obtient l'ALO pour l'elastic-net :

$$\frac{1}{n} \sum_{i=1}^{n} \phi \left(y_i, x_i^\top \hat{\beta} + \left(\frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})} \right) \left(\frac{H_{ii}}{1 - H_{ii}} \right) \right)$$

avec

$$H = X_S \left[X_S^\top \text{diag}[\ddot{l}(\hat{\beta})] X_S + 2\lambda_1 I \right]^{-1} X_S^\top \text{diag}[\ddot{l}(\hat{\beta})]. \tag{2.18}$$

L'Algorithme 2 montre l'utilisation ALO pour l'estimation de Err_{extra} pour l'elastic-net :

Algorithme 2 : Estimation du risque de prédiction avec ALO pour l'elastic-net

---

Entrée : $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$
Sortie : Err_{extra}

1. Calculer $\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left[\sum_{i=1}^{n} l(y_i | x_i^\top \beta) + \lambda_1 \|\beta\|_2^2 + \lambda_2 \|\beta\|_1 \right]$
2. Chercher $S = \{i, \hat{\beta}_i \neq 0\}$
3. Calculer $H = X_S \left[X_S^\top \text{diag}[\ddot{l}(\hat{\beta})] X_S + 2\lambda_1 I \right]^{-1} X_S^\top \text{diag}[\ddot{l}(\hat{\beta})]$
4. L'estimation de Err_{extra} via ALO est donnée par : $\sum_{i=1}^{n} \phi \left(y_i, x_i^\top \hat{\beta} + \left(\frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})} \right) \left(\frac{H_{ii}}{1 - H_{ii}} \right) \right)$.

Chapitre 3

Majoration de l'écart entre ALO et LO

Dans ce chapitre, nous discutons un résultat important, la précision de l'approximation du LO par l'ALO en grande dimension. Pour ce faire, nous énonçons le théorème montrant tout d'abord la performance du ALO à estimer les coefficients du LO.

Théorème 3 *On suppose que :*

- $\frac{n}{p} = \delta_0$*, avec* δ_0 *un nombre fini ,borné loin de zéro(0) ;*
- *les lignes de* $X \in \mathbb{R}^{n\times p}$ *sont indépendantes et suivent une loi Gaussienne de moyenne 0 , de matrice de covariance* Σ *dont la valeur propre maximale est* $\rho_{\max} = \frac{c}{n}$*, c une constante ;*
- *il existe des constantes finies* $c_1(n)$,$c_2(n)$ *et* $q_n \to 0$ *de sorte qu'avec une probabilité d'au moins* $1 - q_n$ *:*

$$\|\dot{l}(\hat{\beta})\|_\infty < c_1(n), \tag{3.1}$$

$$\frac{\|\ddot{l}_{/i}((1-t)\hat{\beta}_{/i} + t(\hat{\beta})) - \ddot{l}_{/i}(\hat{\beta})\|_2}{\|\hat{\beta}_{/i} - \hat{\beta}\|_2} < c_2(n), \tag{3.2}$$

$$\frac{\|\ddot{r}_{/i}((1-t)\hat{\beta}_{/i} + t(\hat{\beta})) - \ddot{r}_{/i}(\hat{\beta})\|_2}{\|\hat{\beta}_{/i} - \hat{\beta}\|_2} < c_2(n) \tag{3.3}$$

  pour $i = 1, 2, ..., n$ *;*

- *les problèmes d'optimisation* (1.1) *et* (2.2) *sont fortement convexes avec une probabilité d'au moins* $1 - \tilde{q}_n$*,* $\tilde{q}_n$ *une constante tendant vers 0 ;*
- n *est assez grand de sorte que* $q_n + \tilde{q}_n < 0.5$*.*

Alors avec une probabilité d'au moins $1 - 4ne^{-p} - \frac{8n}{p^3} - \frac{8n}{(n-1)^3} - q_n - \tilde{q}_n$ *nous avons :*

$$\max_{1\le i\le n} \left| x_i^\top \hat{\beta}_{/i} - x_i^\top \hat{\beta} - \begin{pmatrix} \dot{l}_i(\hat{\beta}) \\ \ddot{l}_i(\hat{\beta}) \end{pmatrix} \left(\frac{H_{ii}}{1 - H_{ii}} \right) \right| \le \frac{C_0}{\sqrt{p}}. \tag{3.4}$$

Le Théorème 3 prouve que l'erreur commise en utilisant l'ALO pour approcher les coefficients de regression estimés par le LO peut être majorée par une constante qui tend vers 0 en grande dimension ; ceci avec une probabilité relative aux nombres n et p.
Pour les estimateurs Ridge, LASSO lissé (grâce à l'approximation de Schmidt), l'Elastic-net lissé et les régressions Logistique, Robuste, Moindre carré, Poisson, on montre que $C_0 = O(PolyLog(n))$. $PolyLog(n)$ désigne une fonction polynomiale en $log(n)$.
Notons que la condition $\rho_{\max} = \frac{c}{n}$ est donnée pour s'assurer que $x_i^\top \beta = O_p(1)$ et $\beta^\top \Sigma \beta = O(1)$ en grande dimension. Pour plus d'information, se référer à [DA16]. La troisième hypothèse suppose la régularité de la dérivée seconde des fonctions pertes et régularisateurs.
L'hypothèse de forte convexité a permis d'obtenir des résultats théoriques qui excluent la pénalisation

LASSO, pour laquelle la fonction à optimiser est convexe mais pas fortement en grande dimension. Grâce au théorème ci-dessus, on peut avoir une majoration de $|ALO - LO|$ énoncé sous forme de Corollaire :

Corollaire 1 *On suppose que les hypothèses du Théorème 3 sont vérifiées. De plus on suppose que :*

$$\max_{i=1,2,..,n} \sup_{|b_i|<\frac{C_0}{\sqrt{p}}} \left|\dot{\phi}(y_i, x_i^\top \hat{\beta}_{/i} + b_i)\right| \leq c_3(n)$$

avec une probabilité r_n, alors

$$|ALO - LO| \leq \frac{c_3(n)C_0}{\sqrt{p}} \tag{3.5}$$

avec une probabilité d'au moins $1 - 4ne^{-p} - \frac{8n}{p^3} - \frac{8n}{(n-1)^3} - q_n - \tilde{q}_n - r_n$.
L'hypothèse faite pour le corollaire 1 suggère de prendre la fonction ϕ régulière.
Le corollaire 1 découle du Théorème 3 par simple utilisation du théorème des accroissements finis.
Pour avoir une majoration proprement dite du $|ALO - LO|$, il est necessaire de trouver l'expression de $c_3(n)$. Comme dans plusieurs applications, la fonction de perte utilisée dans le problème d'optimisation est la même que celle utilisée pour mesurer d'erreur de prédiction, on suppose $l(.,.) = \phi(.,.)$. Ainsi pour les différents estimateurs et méthodes de régressions cités ci-dessus, $c_3(n) = PolyLog(n)$. Finalement on obtient $|ALO - LO| \leq O_p(\frac{PolyLog(n)}{\sqrt{n}})$.
Il est important de mentionner que pour la perte carrée plus précisement pour la régression linéaire, cette majoration a été démontrée en considérant les régularisateurs de la forme $r(\beta) = \gamma\beta^2 + (1 - \gamma)r^\alpha(\beta)$ avec les restrictions $0 < \gamma < 1$ et $\alpha < \infty$. Cette majoration n'a donc pas été démontrée pour l'estimateur LASSO.

Chapitre 4

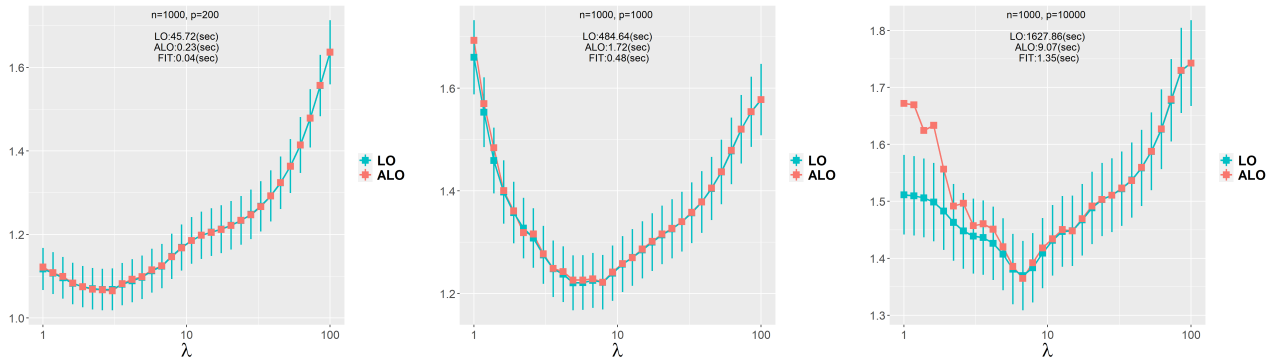
Simulations et discussions

Dans ce chapitre est illustré la précision d'ALO à approcher le LO grâce aux données simulées. Nous avons considéré $n = 1000$ et le vrai vecteur des coefficients est $\beta^\star \in \mathcal{R}^p$. Le nombre de composantes non nulles de $\beta^\star$ est $k = \frac{n}{10}$ et elles sont générées aléatoirement suivant une loi de Laplace moyenne nulle et de variance unité. Pour le nombre de variables, nous avons pris $\frac{n}{p} \in \{5, 1, \frac{1}{10}\}$ ce a permis de tester les configurations $n > p$, $n = p$ et $n < p$. Les lignes de la matrice X sont indépendantes et suivent la loi $\mathcal{N}(0, \Sigma)$. Pour simuler la matrice Σ, on a considéré deux structures de corrélation : Spiked : $corr(X_{ij}, X_{ij'}) = 0.5$ et Toeplitz : $corr(X_{ij}, X_{ij'}) = 0.9^{|j-j'|}$ avec $i = 1, 2, ..., n$ et $j, j' = 1, 2, .., p$. Σ est normalisée de sorte à avoir $Var(x^\top \beta) = 1$.

Notons que pour accélérer les programmes, nous avons simulé la matrice creuse F vérifiant $\Sigma = FF^\top$ ref scripts dans la partie Annexe. Les comparaisons ont été faites pour la regression linéaire Elastic-net, regression logistique LASSO, Regression Poisson Elastic-net.

4.1 Régression linéaire avec pénalisation Elastic-net

Pour cette régression, on prend comme fonction de perte $l(y|x^\top\beta) = \frac{1}{2}(y - x^\top\beta)^2$. La fonction de pénalité $r(\beta)$ est prise de telle sorte que $r(\beta) = \frac{\gamma}{2}\|\beta\|_2^2 + (1-\gamma)\|\beta\|_1$ avec $\gamma = 0.5$, ce qui revient donc à prendre dans l'équation (2.17) $\lambda_1 = \frac{\lambda}{4}$ et $\lambda_2 = \frac{\lambda}{2}$. De même on considère une structure spiked dans la matrice de corrélation et la variable cible $y \sim \mathcal{N}(X\beta^\star, I)$. De plus on prend $\phi(y, x^\top\beta) = (y - x^\top\beta)^2$ ce qui conduit à ALO $= \frac{1}{n}\sum_{i=1}^n (\frac{y_i - x_i^\top\hat{\beta}}{1 - H_{ii}})^2$ avec $H = X_S(X_S^\top X_S + \gamma\lambda I)^{-1}X_S$.



(a) $n > p$ (b) $n = p$ (c) $n < p$

FIGURE 4.1: Erreur quadratique moyenne du LO et ALO pour la régression linéaire elastic-net.

La FIGURE 4.1 est faite pour 30 valeurs du paramètre de pénalisation λ entre 1 et 100 et la barre verticale représente une erreur standard. Cette figure est tout à fait en accord avec les résultats

expérimentaux de la section 5.2.1 de [RM18].

4.2 Régression logistique avec pénalisation LASSO

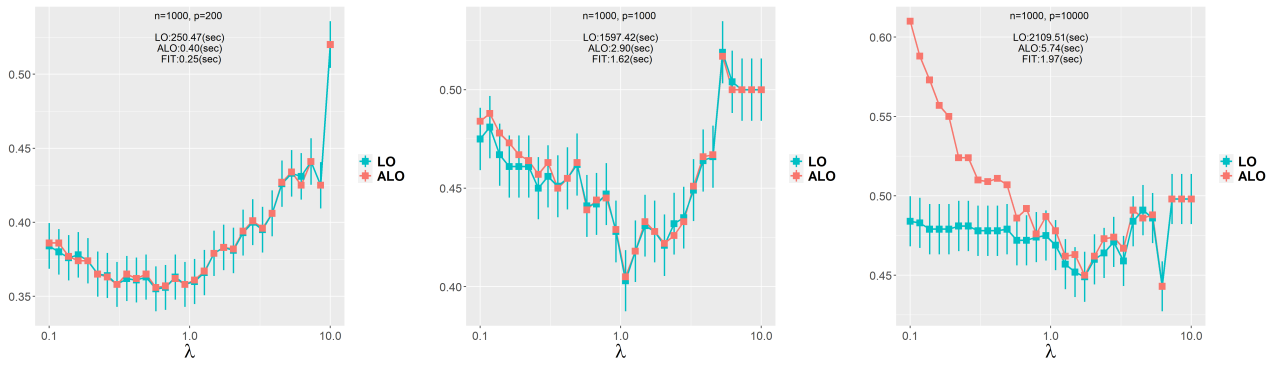
On prend comme fonction de perte $l(y|x^\top\beta) = -yx^\top\beta + log(1 + e^{x^\top\beta})$ (l'opposé de la fonction de vraisemblance ou negative log-likelihood) et la fonction de pénalité $r(\beta) = \|\beta\|_1$. La matrice de corrélation est considérée avec une structure Toeplitz et les réalisations y_i de la variable cible suivent une loi de Bernoulli de paramètre $p_i = \dfrac{e^{x_i^T\beta^\star}}{1 + e^{x_i^T\beta^\star}}$ ($y_i \sim \mathcal{B}(1, p_i)$). Le taux de mauvaise classification est pris comme mesure d'erreur et $\mathbb{1}_{\{x^\top\beta>0\}}$ pour la prédiction. $\mathbb{1}_{\{.\}}$ est la fonction indicatrice. Nous avons alors :

$$\text{ALO} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \mathbb{1}_{\{x_i^\top\hat{\beta} + \frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\frac{H_{ii}}{1-H_{ii}}>0\}}\right|$$

avec

$$H = X(\lambda\text{diag}[\ddot{r}(\hat{\beta})] + X^\top\text{diag}[\ddot{l}(\hat{\beta})]X)^{-1}X^\top\text{diag}[\ddot{l}(\hat{\beta})]$$

.



(a) $n > p$ (b) $n = p$ (c) $n < p$

FIGURE 4.2: Estimateurs LO et ALO de l'erreur de classification pour la régression logistique LASSO.

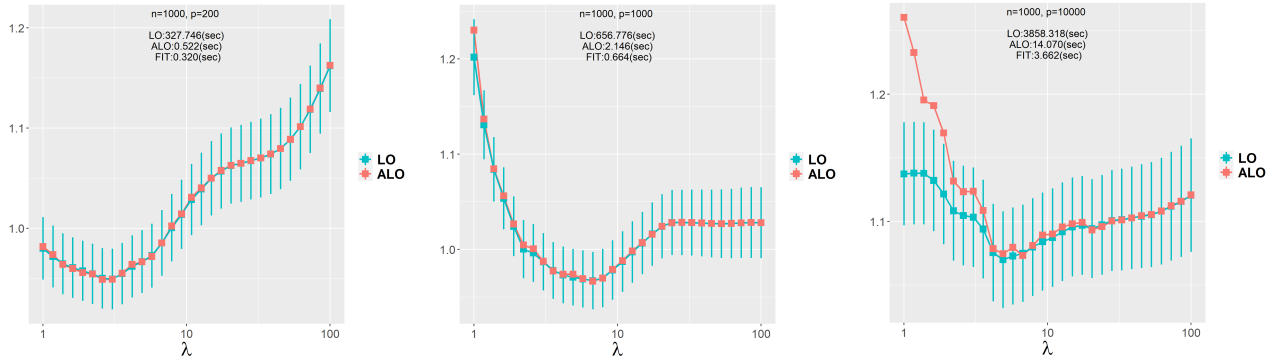
Nous considérons également pour la FIGURE 4.2, 30 valeurs de λ entre 0.1 et 10. A nouveau, nous avons pu reproduire assez bien les résultats de la section 5.2.2 de [RM18].

4.3 Régression Poisson Elastic-net

$y_i \sim \mathcal{P}(e^{x_i^T\beta^\star})$ (loi de Poisson de paramètre $\mu_i = e^{x_i^T\beta^\star}$) et la fonction de perte considérée est donc l'opposé de la fonction de vraisemblance : $l(y|x^\top\beta) = e^{yx^\top\beta} - yx^\top\beta$, $\lambda r(\beta) = \frac{\gamma}{2}\|\beta\|_2^2 + (1-\gamma)\|\beta\|_1$ avec $\gamma = 0.5$. La matrice de covariance est prise avec une structure spiked.
L'erreur absolue moyenne est utilisée comme mesure d'erreur et $e^{x^\top\beta}$ comme prédiction. Ainsi

$$\text{ALO} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - e^{\{x_i^\top\hat{\beta} + \frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\frac{H_{ii}}{1-H_{ii}}\}}\right|$$

avec $H = X_S(X_S^\top X_S + \gamma\lambda I)^{-1}X_S$, $\dot{l}_i(\hat{\beta}) = e^{x_i^\top\hat{\beta}} - y_i$ et $\ddot{l}_i(\hat{\beta}) = e^{x_i^\top\hat{\beta}}$. 30 valeurs de λ sont considérées entre 1 et 100.



(a) Régression linéaire avec pénalisation Elastic-net(section 4.1) pour $\frac{n}{p} = 5$

(b) Régression logistique avec pénalisation LASSO (section 4.2) pour $\frac{n}{p} = 1$

(c) Régression Poisson avec pénalisation Elastic-net pour $\frac{n}{p} = \frac{1}{10}$

FIGURE 4.3: Estimateurs LO et ALO de l'erreur absolue moyenne pour la régression Poisson Elastic-net.

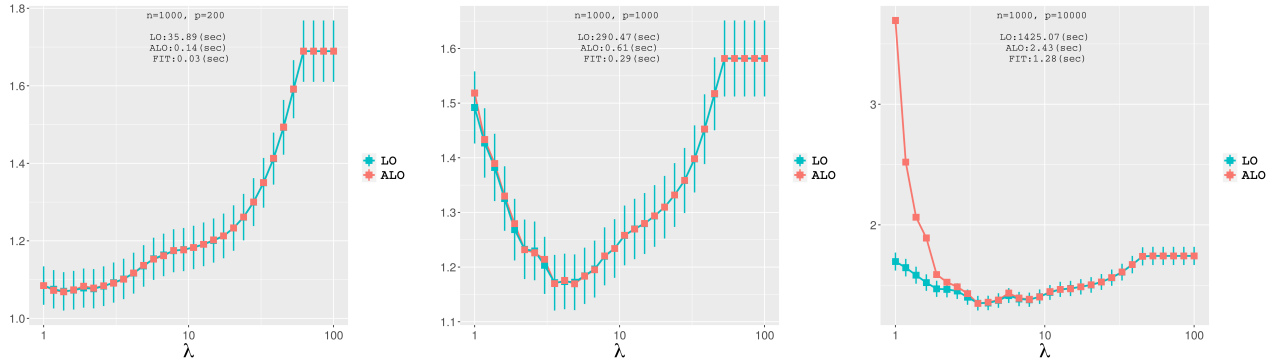
La FIGURE 4.3 est bien en accord avec la section 5.2.3 de [RM18].
Pour les trois méthodes de régressions étudiées, on voit que ALO donne une réduction du temps de calcul et une bonne approximation du LO. Cependant les simulations prouvent que ALO dégénère quand λ devient très petit, plus particulièrement dans le cas $p > n$. Ce constat n'est pas un problème d'autant plus que les valeurs de λ qui minimise l'erreur du LO et ALO ont tendance à être loins des valeurs faibles.
Il est important de mentionner dans ce travail que [RM18] ont montré grâce aux données réelles des enregistrements des neurones spatialement sensibles que ALO donne une bonne approximation de LO dans un cas où la normalité de la matrice X ne peut pas être admise. Ainsi on peut espérer que l'hypothèse de normalité dans la dérivation de ALO puisse être remplacée par des hypothèses plus faibles (les distributions sous gaussiennes).

Chapitre 5

Contributions

5.1 Impact du paramètre de pondération de l'elastic-net mis à zéro

Comme évoqué à la fin du Chapitre 3, la majoration de l'écart entre entre ALO et LO a été démontrée dans le cas de régression linéaire en considérant une pénalisation définie en pondérant "Ridge" et "Schmidt" sous la restriction $0 < \gamma$ où γ est le paramètre de pondération. Ce travail théorique de [RM18] exclut indirectement l'estimateur LASSO. Ainsi dans cette partie, nous regardons grâce aux simulations les conséquences du choix $\gamma = 0$.



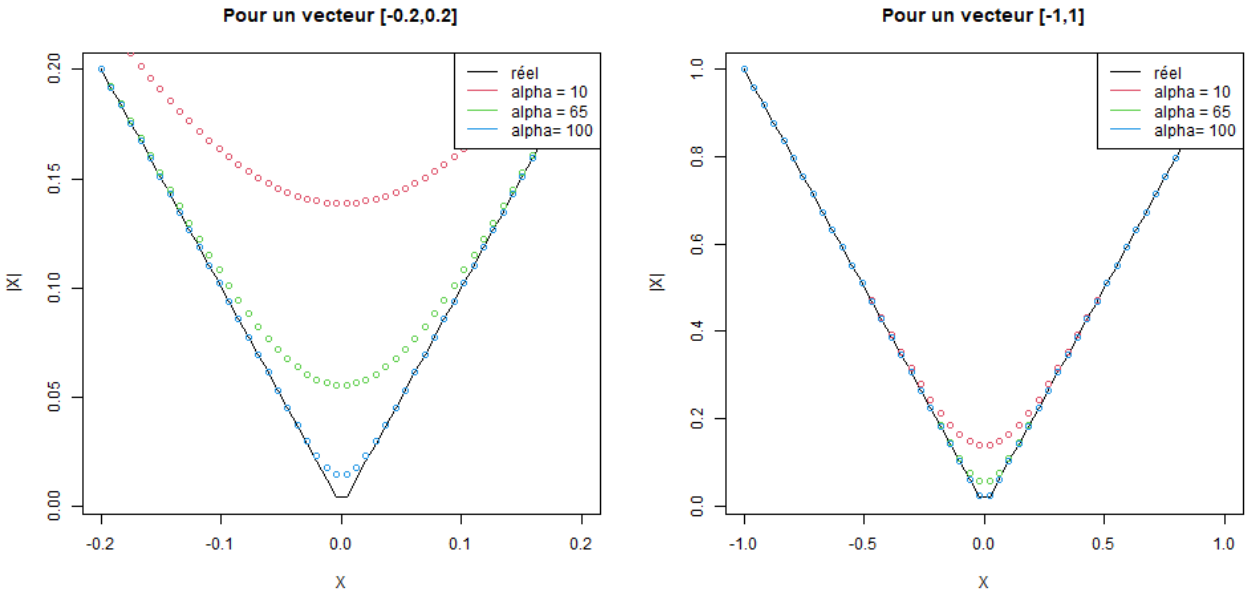
(a) $n > p$ (b) $n = p$ (c) $n < p$

FIGURE 5.1: Erreur quadratique moyenne du LO et ALO pour la régression linéaire LASSO

La FIGURE 5.1 illustre la performance de ALO pour un problème de régression linéaire avec pénalisation LASSO (correspondant à $\gamma = 0$). Les données utilisées sont similaires à celles de la section 4.1. On peut remarquer alors que ALO donne une bonne approximation du LO sauf pour les petites valeurs de λ et $p > n$. Mais on voit, graphe (c), que cette mauvaise approximation ne pertube pas le minimiseur global en λ (notons que ceci se voit aussi dans le chapitre 4 où $\gamma = 0.5$). Ce constat est encourangeant et constitue un début pour l'étude les propriétés théoriques de ALO dans un problème de régression linéaire avec pénalisation LASSO.
Rappelons que le cas de régression linéaire avec pénalisation Ridge a été étudié par [Gea79] (validation croisée généralisée) et [Li86].

5.2 Pénalisation de Schmidt

Pour dériver l'ALO pour les estimateurs non réguliers comme LASSO et Elastic-net, on a considéré une approximation de la norme $\|.\|_1$. La qualité de cette approximation dépend du choix de l'hyper-paramètre α.

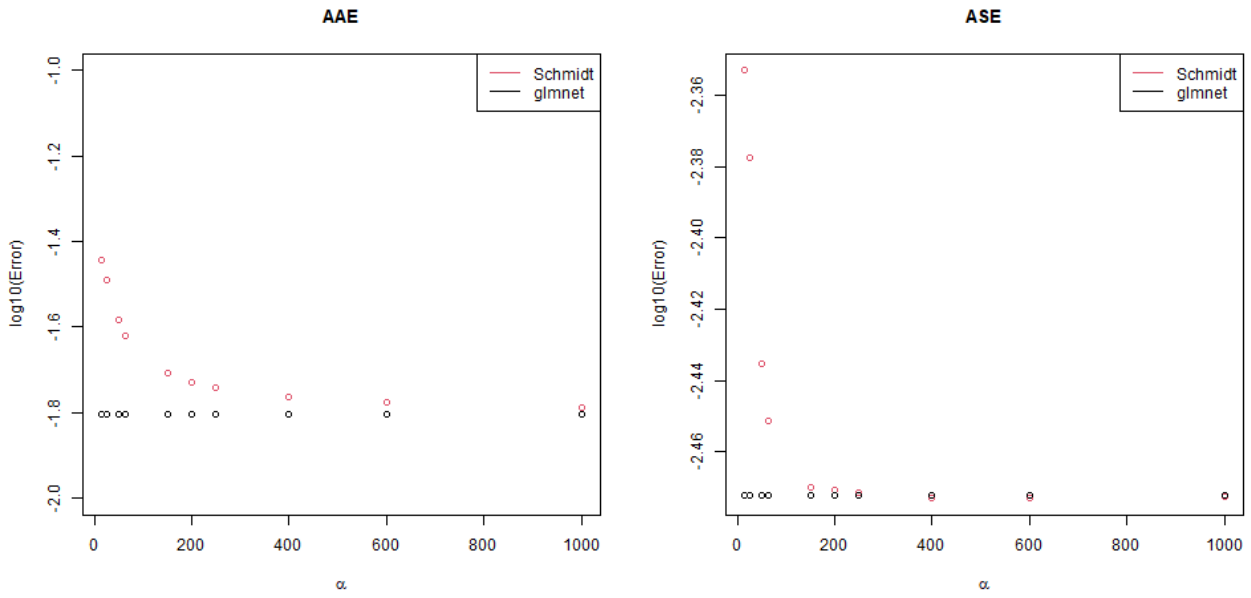


(a) Séquence de 50 points dans l'intervalle $[-0.2, 0.2]$

(b) Séquence de 50 points dans l'intervalle $[-1, 1]$

FIGURE 5.2: Approximation de Schmidt de la valeur absolue d'un réel

Sur la FIGURE 5.2, on peut remarquer que l'approximation de Schmidt est une bonne approximation de la valeur absolue surtout en augmentant α (alpha).
Cependant cette approximation devient moins bonne pour des valeurs voisines de 0. De même un choix de α très grand peut engendrer des instabilités, voir [MGR07].
On rappelle que cette approximation est deux fois différentiables et donc en plus d'avoir servi à définir ALO, elle pourrait être utilisée pour estimer plus rapidement les coefficients dans une régression pénalisée.
Nous avons donc étudié ce potentiel dans notre stage. On commence par estimer à α et λ fixés le vecteur des coefficients $\hat{\beta}^{\alpha}$ en utilisant une méthode de Newton relaxée (Damped Newton). Cette méthode est une modification du pas de la méthode classique de Newton (Newton-Raphson) enfin d'avoir une direction de descente à chaque itération ; voir plus d'explication dans l'Annexe A.2, pour la régression linéaire et logistique où les fonctions à optimiser sont maintenant strictements convexes.

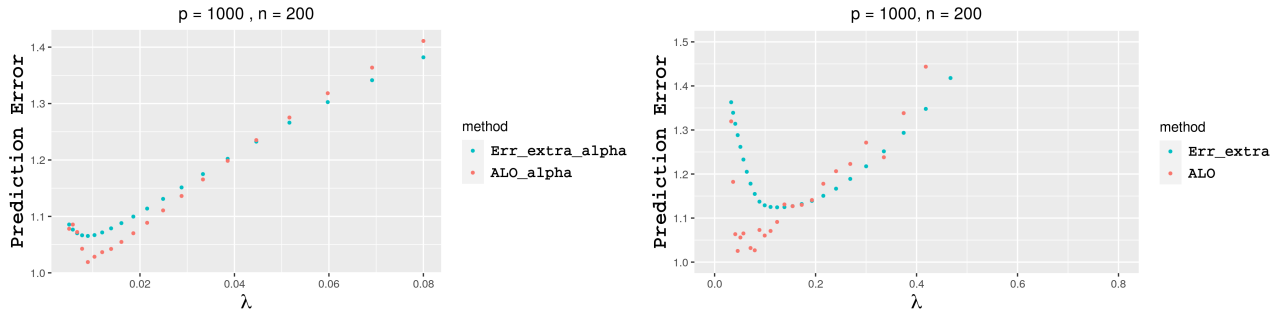


(a) Comparaison AAE Estimation de Schmidt Vs LASSO glmnet

(b) Comparaison ASE Estimation de Schmidt Vs LASSO glmnet

FIGURE 5.3: Comparaison de l'erreur d'estimateur des coefficients pour $\lambda = 30$

ASE $= \frac{1}{p}\sum_i^p(\beta_i^\star - \hat{\beta}_i)^2$ et AAE $= \frac{1}{p}\sum_i^p|\beta_i^\star - \hat{\beta}_i|$ désignent respectivement l'erreur quadratique moyenne (Average Square Error) et l'erreur absolue moyenne de l'estimation de $\beta^\star$. Ces quantités ont été calculées en utilisant les données de FIGURE 1.1, et $\lambda = 30$. Cette valeur de λ correspond à peu près l'optimale d'après la FIGURE 1.1 et donc pour l'estimateur LASSO. Pour une valeur de α voisine de 150, l'estimation de $\beta^\star$ grâce à l'approximation de Schmidt (SmoothL1) est faite avec une erreur très proche de celle du LASSO ordinaire voir même mieux en terme de ASE en augmentant α. On s'attend à obtenir de meilleurs résultats dans un problème où il y a moins de coefficients nuls.
Cependant cet estimateur de Schmidt peut réduire la valeur des coefficients sans les annuler, ce qui la rend moins utile que le LASSO dans la sélection de variables.



(a) Comparaison de l'erreur de prédiction et ALO$^\alpha$ pour $\alpha = 25$

(b) Comparaison de l'erreur de prédiction et ALO pour une régression linéaire LASSO

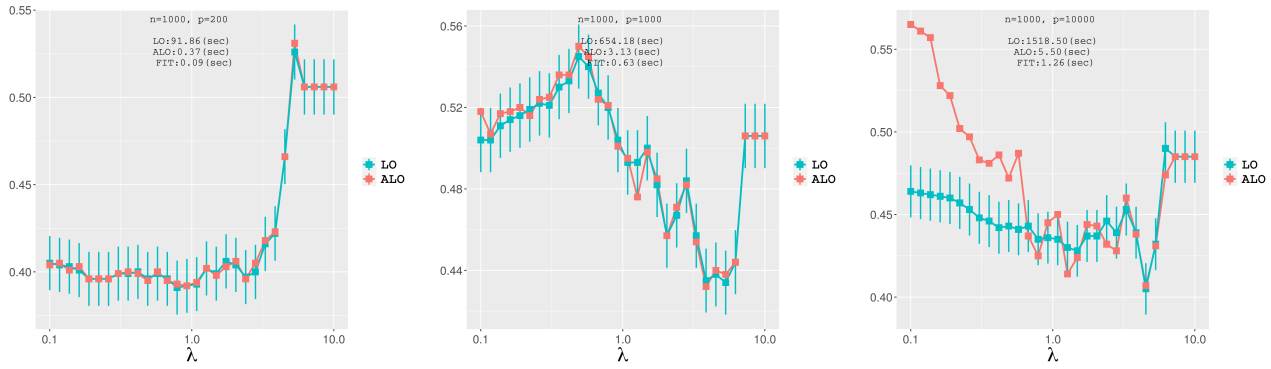
FIGURE 5.4: Estimation de l'erreur de prédiction avec la pénalisation de Schmidt et pénalisation LASSO

La FIGURE 5.4 est faite avec le même design (c'est-à-dire le même méchanisme probabiliste pour générer les données) que la FIGURE 2.1 où nous avons considéré cette fois-ci $n = 200$ et $p = 1000$. L'idée est d'examiner la performance de l'approximation de la validation croisée complète avec la pénalisation de Schmidt (ALO$^\alpha$), à estimer l'erreur de prédiction. Ainsi nous pouvons voir sur le graphe (a) que ALO$^\alpha$ est une approximation plus stable de l'erreur de prédiction que l'ALO donné par la pénalisation de LASSO, surtout pour les petites valeurs du paramètre de pénalisation λ. Rappelons que l'approximation de Schmidt converge vers le LASSO et donc ALO$^\alpha$ vers ALO pour les grandes

valeurs de α. Cependant nous avons constaté qu'une grande valeur de α ralentit la convergence de l'algorithme d'optimisation qui est ici la méthode "Damped-Newton".

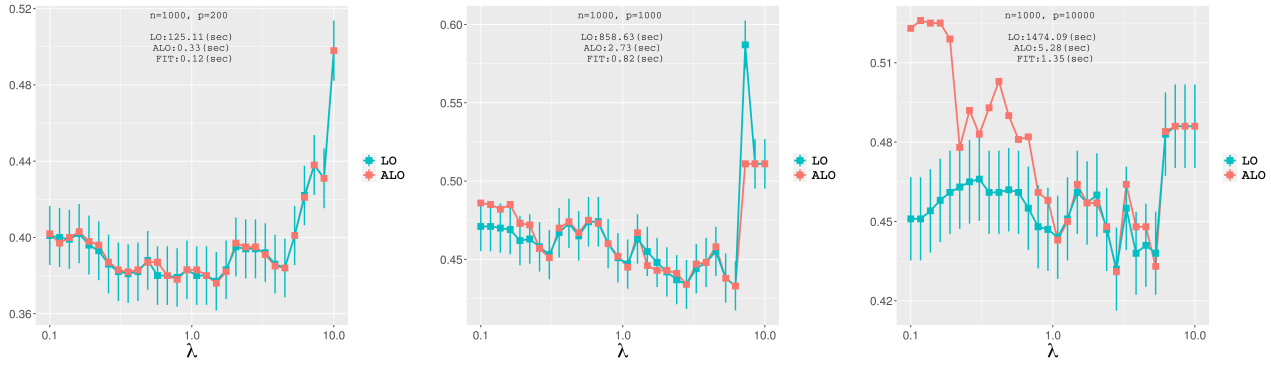
5.3 Influence de fortes et faibles corrélations

Un point fort de L'ALO est qu'il est obtenu sans faire d'hypothèse d'indépendance sur les variables explicatives. Nous nous proposons alors de tester l'influence des corrélations fortes et faibles entre les variables explicatives. Nous prenons donc la matrice X avec la structure de coorélation Toeplitz : $corr(X_{ij}, X_{ij'}) = \rho^{|j-j'|}$ avec $i = 1, 2, ..., n,\ j, j' = 1, 2, .., p$ et $\rho \in \{0.02, 0.7, 0.99\}$



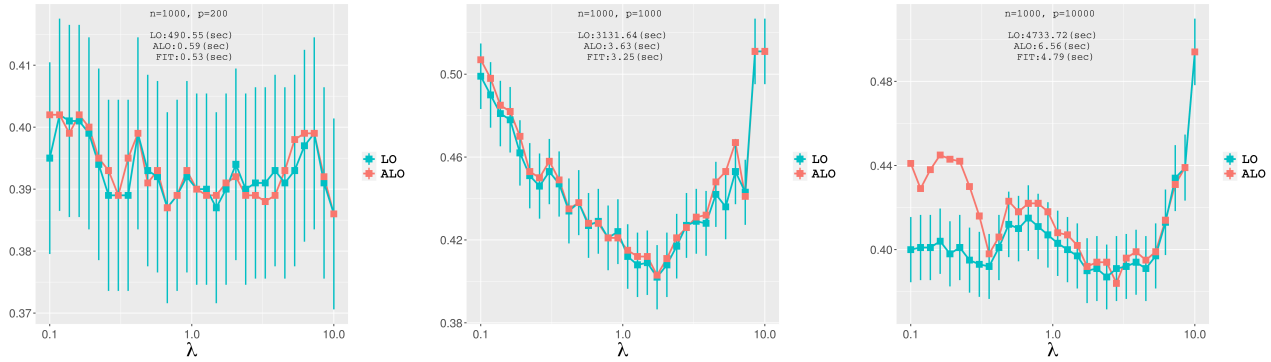
(a) $n > p$ (b) $n = p$ (c) $n < p$

FIGURE 5.5: Erreur de classification du LO et ALO pour la régression logistique LASSO pour $\rho = 0.02$



(a) $n > p$ (b) $n = p$ (c) $n < p$

FIGURE 5.6: Erreur de classification du LO et ALO pour la régression logistique LASSO pour $\rho = 0.7$



(a) $n > p$ (b) $n = p$ (c) $n < p$

FIGURE 5.7: Erreur de classification du LO et ALO pour la régression logistique LASSO pour $\rho = 0.99$

Les FIGURES 5.5, 5.6 et 5.7 sont faites avec les données de la FIGURE 4.2, avec une introduction du paramètre de corrélation ρ. On peut alors voir que ALO donne une bonne approximation du LO pour chaque valeur de ρ et donc pour une entrée X avec une faible ou forte corrélation. Pour la configuration $p > n$, comme évoqué dans le chapitre 4, l'approximation est moins bonne pour les petites valeurs du paramètre de pénalisation λ. Notons que le LO peut avoir une variance importante surtout avec une entrée X avec une forte corrélation.

Chapitre 6

Conclusion et perspectives

Les missions proposées lors ce stage ont été remplies. Les conditions requises sur les données pour l'utilisation de l'approximation de la validation croisée complète (ALO pour désigner "Approximate Leave-One-Out") ont été abordées à travers les théorèmes. Parmi ces conditions, il important de mentionner que les obervations doivent être indépendantes et identiquements distribuées (iid). Par des simulations, on a montré que ALO offre une réduction considérable du temps de calcul et une bonne approximation du LO en grande dimension quand le rapport $\frac{n}{p}$ est un nombre fixe. Nous avons également évoqué l'utilisation de la pénalisation de Schmidt qui possède des propriétés de régularités et qui peut donc être vue comme alternative au LASSO pour certains problèmes particuliers. Les objectifs personnels parmi lesquels on peut citer la visualisation avec ggplot2, l'optimation des programmes de calculs avec l'association du langage C, la parallélisation sont atteints.
Il est à noter que dans l'estimation des coefficients avec la pénalisation de Schmidt, nous avons utilisé une méthode de Netwon-Raphson, méthode simple mais qui nécessite une résolution de système linéaire et donc coûteuse en grande dimension. Ainsi pour les travaux futurs nous envisageons l'utilisation des méthodes d'optimisation accélérées comme la descente de gradient, la méthode du point intérieur.
Nous signalons que chaque expérience exige un temps considérable et une ressource de calcul importante. Ainsi compte tenu de nos ressources et du temps limité nous n'avons pas pu traiter des données réelles et nous envisageons de le faire très bientôt pour donner du poids à ce travail.
Parmi les perspectives qu'offre ce travail, on peut mentionner celle déjà détaillée dans la section "Concluding Remarks" de [RM18] : l'utilisation d'approximations par Monte-carlo est clairement possible et accélèrerait le calcul mais peut-on étendre les résultats théoriques de [RM18] à de telles approximations ?

Bibliographie

[DA16] Donoho D. and Montanari A. High dimensional robust m-estimation : Asymptotic variance via approximate message passing. *Probab Theory Relat Fields*, 166(3-4) :935–969, 2016.

[DMM11] David Donoho, Arian Maleki, and Andrea Montanari. The noise-sensitivity phase transition in compressed sensing. *Information Theory, IEEE Transactions on*, 57 :6920 – 6941, 11 2011.

[Efr83] Bradley Efron. Estimating the error rate of a prediction rule : Improvement on cross-validation. *Journal of the American Statistical Association*, 78(382) :316–331, 1983.

[Gea79] Golub and Gene H. et al. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2) :215–223, 1979.

[Gei75] Seymour Geisser. The predictive sample reuse method with applications. *Journal of the American Statistical Association*, 70(350) :320–328, 1975.

[lEFF93] lldiko E. Frank and Jerome H. Friedman. A statistical view of some chemometrics regression tools. *Technometrics*, 35(2) :109–135, 1993.

[Li86] Ker-Chau Li. Asymptotic optimality of c_l and generalized cross-validation in ridge regression with application to spline smoothing. *Ann. Statist.*, 14(3) :1101–1112, 09 1986.

[MGR07] Schmidt M., Fung G., and Rosales R. Fast optimization methods for l1 regularization : A comparative study and two new approaches. *ECML*, 4701 :286–297, 2007.

[Mou18] Arian et Baraniuk Richard G. Mousavi, Ali et Maleki. Estimation cohérente des paramètres pour lasso et passage approximatif des messages. *Ann. Statist.*, 46(1) :119–148, 02 2018.

[OK16] Tomoyuki Obuchi and Yoshiyuki Kabashima. Cross validation in LASSO and its acceleration. *Journal of Statistical Mechanics : Theory and Experiment*, 2016(5) :053304, may 2016.

[RM18] Kamiar Rahnama Rad and Arian Maleki. A scalable estimate of the extra-sample prediction error via approximate leave-one-out. *arXiv : Methodology*, 2018.

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1) :267–288, 1996.

[Tib13] Ryan J. Tibshirani. The lasso problem and uniqueness. *Electron. J. Statist.*, 7 :1456–1490, 2013.

[Wai09] M. J. Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using ℓ_1 -constrained quadratic programming (lasso). *IEEE Transactions on Information Theory*, 55(5) :2183–2202, 2009.

[XW96] Dong Xiang and Grace Wahba. A generalized approximate cross validation for smoothing splines with non-gaussian data. *Statistica Sinica*, 6(3) :675–692, 1996.

[ZH05] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 67(2) :301–320, 2005.

Annexe A

Preuves et explications

A.1 Obtention de ALO pour les fonctions régularisateurs deux fois différentiables

A.1.1 Formule de Newton

Soit $\hat{\beta}_{/i} = \underset{\beta \in \mathbb{R}^p}{\arg\min} f(\beta)$ avec $f(\beta) = \left[\sum_{j \neq i} l(y_j | x_j^\top \beta) + \lambda r(\beta)\right]$. En partant de $\hat{\beta}$, l'approximation de $\hat{\beta}_{/i}$ par la méthode de Newton est donnée par :

$$\tilde{\beta}_{/i} = \hat{\beta} - \left[H_{f(\hat{\beta})}\right]^{-1} \nabla f(\hat{\beta})$$

H_f est la matrice Hessienne et ∇f le gradient de f. Nous avons alors :

$$H_{f(\hat{\beta})} = \left[\sum_{j \neq i} x_j x_j^\top \ddot{l}(y_j | x_j^\top \hat{\beta}) + \lambda diag[\ddot{r}(\hat{\beta})]\right]$$

Comme $\hat{\beta}$ est la solution de (1.1), la condition d'optimalité s'écrit :

$$\sum_{j=1}^{n} x_j \dot{l}(y_j | x_j^\top \hat{\beta}) + \lambda \dot{r}(\hat{\beta}) = 0$$

Ce qui équivaut à $\sum_{j \neq i} x_j \dot{l}(y_j | x_j^\top \hat{\beta}) + x_i \dot{l}(y_i | x_i^\top \hat{\beta}) + \lambda \dot{r}(\hat{\beta}) = 0$. Alors on a

$$\nabla f(\hat{\beta}) = \sum_{j \neq i} x_j \dot{l}(y_j | x_j^\top \hat{\beta}) + \lambda \dot{r}(\hat{\beta}) = -x_i \dot{l}(y_i | x_i^\top \hat{\beta})$$

Il s'en suit que

$$\tilde{\beta}_{/i} = \hat{\beta} + \left[\sum_{j \neq i} x_j x_j^\top \ddot{l}(y_j | x_j^\top \beta) + \lambda diag[\ddot{r}(\beta)]\right]^{-1} x_i \dot{l}(y_i | x_i^\top \hat{\beta})$$

A.1.2 Démonstration de la formule du ALO

De la section A.1.1, nous avons :

$$\begin{aligned} H_{f(\hat{\beta})} &= \sum_{j \neq i} x_j x_j^\top \ddot{l}(y_j | x_j^\top \hat{\beta}) + \lambda diag[\ddot{r}(\hat{\beta})] \\ &= \sum_{j=1}^{n} x_j x_j^\top \ddot{l}(y_j | x_j^\top \hat{\beta}) + \lambda diag[\ddot{r}(\hat{\beta})] - x_i x_i^\top \ddot{l}(y_i | x_i^\top \hat{\beta}) \\ &= J - x_i x_i^\top \ddot{l}(y_i | x_i^\top \hat{\beta}) \end{aligned}$$

Pour calculer l'inverse de $H_{f(\hat{\beta})}$, considérons le lemme de Woodbury-Sherman-Morrison qui dit que pour $A \in \mathbb{R}^{p\times p}$ inversible , u,$v^T \in \mathbb{R}^p$, $A+uv^T$ est inversible si et seulement si $1+v^TA^{-1}u \neq 0$. On a alors dans ce cas

$$[A+uv^T]^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}$$

En prenant $J=A$,$u=x_i\ddot{l}(y_i|x_i^\top\beta)$ et $v=-x_i$, nous avons :

$$\begin{aligned}\left[H_{f(\hat{\beta})}\right]^{-1} &= \left[J - x_i\ddot{l}(y_i|x_i^\top\hat{\beta})x_i^\top\right]^{-1} \\ &= J^{-1} + \frac{J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})x_i^\top J^{-1}}{1-x_i^\top J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})}\end{aligned}$$

En remplaçant $\left[H_{f(\hat{\beta})}\right]^{-1}$ par sa valeur, nous obtenons :

$$\begin{aligned}\tilde{\beta}_{/i} &= \hat{\beta} + \dot{l}(y_i|x_i^\top\hat{\beta})\left[J^{-1}x_i + \frac{J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})x_i^\top J^{-1}}{1-x_i^\top J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})}x_i\right] \\ &= \hat{\beta} + \frac{\dot{l}(y_i|x_i^\top\hat{\beta})}{\ddot{l}(y_i|x_i^\top\hat{\beta})}\left[J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta}) + \frac{J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})x_i^\top J^{-1}}{1-x_i^\top J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})\right] \\ &= \hat{\beta} + \frac{\dot{l}(y_i|x_i^\top\hat{\beta})}{\ddot{l}(y_i|x_i^\top\hat{\beta})}\left[J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta}) + \frac{J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})x_i^\top J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})}{1-x_i^\top J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})}\right]\end{aligned}$$

Il vient :

$$\begin{aligned}x_i^\top\tilde{\beta}_{/i} &= x_i^\top\hat{\beta} + \frac{\dot{l}(y_i|x_i^\top\hat{\beta})}{\ddot{l}(y_i|x_i^\top\hat{\beta})}\left[x_i^\top J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta}) + \frac{x_i^\top J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})x_i^\top J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})}{1-x_i^\top J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})}\right] \\ &= x_i^\top\hat{\beta} + \frac{\dot{l}(y_i|x_i^\top\hat{\beta})}{\ddot{l}(y_i|x_i^\top\hat{\beta})}\left[\frac{x_i^\top J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})}{1-x_i^\top J^{-1}x_i\ddot{l}(y_i|x_i^\top\hat{\beta})}\right] \\ x_i^\top\tilde{\beta}_{/i} &= x_i^\top\hat{\beta} + \frac{\dot{l}_i(\hat{\beta})}{\ddot{l}_i(\hat{\beta})}\left[\frac{H_{ii}}{1-H_{ii}}\right]\end{aligned}$$

où on a posé $\dot{l}_i(\hat{\beta}) = \dot{l}(y_i|x_i^\top\hat{\beta})$, $\ddot{l}_i(\hat{\beta}) = \ddot{l}(y_i|x_i^\top\hat{\beta})$ et $H_{ii} = x_i^\top J^{-1}x_i\ddot{l}_i(\hat{\beta})$. On obtient alors : $H = (H_{ij}), i=1,2,...,n; j=1,2,...,n$ avec :

$$H = X(\lambda\text{diag}[\ddot{r}(\hat{\beta})] + X^\top\text{diag}[\ddot{l}(\hat{\beta})]X)^{-1}X^\top\text{diag}[\ddot{l}(\hat{\beta})]$$

A.2 Algorithme pour l'estimation $\hat{\beta}^\alpha$

On cherche à résoudre le problème d'optimition :

$$\hat{\beta} = \underset{\beta\in\mathbb{R}^p}{\arg\min}\left[f(\beta) = \sum_{i=1}^n l_i(\beta) + \lambda r^\alpha(\beta)\right]$$

— Méthode Newton-Raphson
En utilisant la méthode de Newton-Raphson, on obtient à chaque itération :

$$\hat{\beta}^{new} = \hat{\beta}^{old} - \left[\frac{\partial^2 f(\beta)}{\partial\beta\partial\beta^\top}|_{\beta=\hat{\beta}^{old}}\right]^{-1}\frac{\partial f(\beta)}{\partial\beta}|_{\beta=\hat{\beta}^{old}}$$

On a $\dfrac{\partial f(\beta)}{\partial\beta} = \sum_{i=1}^n x_i\dot{l}_i(\beta) + \lambda\dot{r}^\alpha(\beta)$ avec

$$\dot{r}^\alpha(\beta) = \left[\frac{e^{\alpha\beta_1}-e^{-\alpha\beta_1}}{e^{\alpha\beta_1}+e^{-\alpha\beta_1}+2}, ..., \frac{e^{\alpha\beta_p}-e^{-\alpha\beta_p}}{e^{\alpha\beta_p}+e^{-\alpha\beta_p}+2}\right]^\top$$

Et $\frac{\partial^2 f(\beta)}{\partial\beta\partial\beta^\top}|_{\beta=\hat{\beta}^{old}} = \sum_{i=1}^n x_i \ddot{l_i}(\beta) x_i^\top + \lambda diag[\ddot{r}^\alpha(\beta)]$ avec

$$\ddot{r}^\alpha(\beta) = \left[\frac{2\alpha}{(e^{\alpha\beta_1} + e^{-\alpha\beta_1} + 2)}, ..., \frac{2\alpha}{(e^{\alpha\beta_p} + e^{-\alpha\beta_p} + 2)}\right]^T$$

— Régression linéaire pénalisée
Pour la régression linéaire, on prend prend $l_i(\beta) = \frac{1}{2}(y_i - x_i^\top\beta)^2$, $\dot{l}_i(\beta) = -(y_i - x_i^\top\beta)$ et $\ddot{l}_i(\beta) = 1$. On obtient alors :

$$\hat{\beta}^{new} = \hat{\beta}^{old} - \left[X^\top X + \lambda diag[\ddot{r}^\alpha(\hat{\beta}^{old})]\right]^{-1}\left[-X^\top(Y - X\hat{\beta}^{old}) + \lambda\dot{r}^\alpha(\hat{\beta}^{old})\right] \tag{A.1}$$

— Régression logistique pénalisée
Pour la régression logistique, on prend le log-likelihood négatif ,c'est-à-dire $l_i(\beta) = -y_i x_i^\top\beta + \log(1 + e^{x_i^\top\beta})$, $\dot{l}_i = -y_i + \frac{e^{x_i^\top\beta}}{1 + e^{x_i^\top\beta}}$ et $\ddot{l}_i = \frac{e^{x_i^\top\beta}}{(1 + e^{x_i^\top\beta})^2}$. Ce qui permet d'avoir :

$$\hat{\beta}^{new} = \hat{\beta}^{old} - \left[X^\top W_{\ddot{l}(\hat{\beta}^{old})} X + \lambda diag[\ddot{r}^\alpha(\hat{\beta}^{old})]\right]^{-1}\left[\dot{l}(\hat{\beta}^{old}) + \lambda\dot{r}^\alpha(\hat{\beta}^{old})\right] \tag{A.2}$$

— Méthode Newton Damped
La convergence avec la méthode de Newton-Raphson n'étant pas toujours assurée, nous considérons dans cette section une méthode de Newton relaxée (Damped Newton).
Chaque itération avec la cette méthode est donnée par :

$$\hat{\beta}^{new} = \hat{\beta}^{old} - \frac{1}{2^i}\left[\frac{\partial^2 f(\beta)}{\partial\beta\partial\beta^\top}|_{\beta=\hat{\beta}^{old}}\right]^{-1}\frac{\partial f(\beta)}{\partial\beta}|_{\beta=\hat{\beta}^{old}}$$

où i est un entier défini par :

$$i = \min_j\{j \geq 0, f(\hat{\beta}^{old} - \frac{1}{2^j}\left[\frac{\partial^2 f(\beta)}{\partial\beta\partial\beta^\top}|_{\beta=\hat{\beta}^{old}}\right]^{-1}\frac{\partial f(\beta)}{\partial\beta}|_{\beta=\hat{\beta}^{old}}) < f(\hat{\beta}^{old})\}$$

La méthode de Newton est donc le cas où on fait un déplacement $i = 0$. Généralement on fixe une valeur maximale i_{max} à ne pas dépasser.
Les conditions ou critères d'arrêt possible sont :

1. $k > k_{\max}$, $k_{\max}$ le nombre maximal d'itération
2. $\frac{\|\hat{\beta}^{new} - \hat{\beta}^{old}\|_2}{\|\hat{\beta}^{new}\|_2} < \epsilon_1$, avec $\epsilon_1 > 0$
3. $\|\frac{\partial f(\beta)}{\partial\beta}|_{\beta=\hat{\beta}^{new}}\|_2 < \epsilon_2$, avec $\epsilon_2 > 0$

Annexe B

Scripts R

Certains codes présentés ici ont été faits par [RM18], parmi ces codes, il y a ceux que nous avons modifiés pour nos besoins et d'autres traduits de Matlab en R.

B.1 Code R de la FIGURE 1.1

```
#les packages

cat("\014") # pour effacer tout dans la console
library(class)
library(ggplot2)
library(dplyr)
library(glmnet)
set.seed(0) # pour la reproductibilité des résultats

# nous ferons une moyenne sur 500 echantillons

# initialisation
p               =     1000
n               =     250
k               =     50
beta.star       =     rep(0, p)
beta.star[1:k] =      sqrt(5*10/9)/sqrt(k) #qui vaut 1/3
o               =     sqrt(2)
MCMCsamples     =     500
m               =     20

for (i in 1:MCMCsamples){
# generate data

X               =     matrix(rnorm(n*p, mean = 0, sd = 1), ncol = p, nrow = n)
e               =     rnorm(n, mean = 0, sd = o)
y               =     X %*% beta.star + e




# pour la première itération, on fait le choix des 20 lambdas grâce
# à la fonction glmnet
if (i==1){
cv.lasso.3f     =     cv.glmnet(X, y, alpha = 1,  intercept = FALSE,
standardize = FALSE,
dfmax = floor(0.99 * n),
nfolds = 3
)
```

```
cv.lasso.5f     =     cv.glmnet(X, y, alpha = 1,  intercept = FALSE,
standardize = FALSE,
dfmax = floor(0.99 * n),
nfolds = 5
)
cv.lasso.10f    =     cv.glmnet(X, y, alpha = 1,  intercept = FALSE,
standardize = FALSE,
dfmax = floor(0.99 * n),
nfolds = 10
)
cv.lasso.lo     =     cv.glmnet(X, y, alpha = 1,  intercept = FALSE,
standardize = FALSE,
dfmax = floor(0.99 * n),
nfolds = n
)
lasso.fit       =     glmnet(X, y, alpha = 1,  intercept = FALSE,
standardize = FALSE,
dfmax = floor(0.99 * n)
)
lambda.min      =     max(min(lasso.fit$lambda), min(cv.lasso.3f$lambda),
min(cv.lasso.5f$lambda),
min(cv.lasso.10f$lambda),
min(cv.lasso.lo$lambda)
)
lambda.max      =     min(max(lasso.fit$lambda), max(cv.lasso.3f$lambda),
max(cv.lasso.5f$lambda),
max(cv.lasso.10f$lambda),
max(cv.lasso.lo$lambda)
)
lambda          =     rev(exp(seq(log(lambda.min),
log(lambda.max),
length.out = m)
)
)
cv.3f           =     matrix(rep(0, m*MCMCsamples), nrow = m,
ncol = MCMCsamples
)
cv.5f           =     matrix(rep(0, m*MCMCsamples), nrow = m,
ncol = MCMCsamples
)
cv.10f          =     matrix(rep(0, m*MCMCsamples), nrow = m,
ncol = MCMCsamples
)
cv.lo           =     matrix(rep(0, m*MCMCsamples), nrow = m,
ncol = MCMCsamples
)
extraErr        =     matrix(rep(0, m*MCMCsamples), nrow = m,
ncol = MCMCsamples
)
}
cv.lasso.3f     =     cv.glmnet(X, y, alpha = 1,  intercept = FALSE,
standardize = FALSE,
lambda = lambda, nfolds = 3
)
cv.lasso.5f     =     cv.glmnet(X, y, alpha = 1,  intercept = FALSE,
standardize = FALSE,
lambda = lambda, nfolds = 5
)
cv.lasso.10f    =     cv.glmnet(X, y, alpha = 1,  intercept = FALSE,
standardize = FALSE,
lambda = lambda, nfolds = 10
```

```
)
cv.lasso.lo      =      cv.glmnet(X, y, alpha = 1,  intercept = FALSE,
standardize = FALSE,
lambda = lambda, nfolds = n
)
lasso.fit        =      glmnet(X, y, alpha = 1,  intercept = FALSE,
standardize = FALSE,
lambda = lambda
)

# la fonction cv.glmnet ne fais pas forcément la validation croisée
# pour toutes les valeurs de lambda qu'on fournit.
# Cela peut empêcher de faire la comparaison entre
# les différentes technique de cv.
# Pour résoudre ce problème, on fait une prédiction pour les erreurs
# correspondates à nos lambdas avec la fonction smooth.spline

cv.3f.sspline    =      smooth.spline(cv.lasso.3f$lambda, cv.lasso.3f$cvm,
df = length(cv.lasso.3f$lambda)-1
)
cv.3f.i          =      predict(cv.3f.sspline, lambda)
cv.3f[ ,i]       =      cv.3f.i$y

cv.5f.sspline    =      smooth.spline(cv.lasso.5f$lambda, cv.lasso.5f$cvm,
df = length(cv.lasso.5f$lambda)-1
)
cv.5f.i          =      predict(cv.5f.sspline, lambda)
cv.5f[ ,i]       =      cv.5f.i$y

cv.10f.sspline   =      smooth.spline(cv.lasso.10f$lambda, cv.lasso.10f$cvm,
df = length(cv.lasso.10f$lambda)-1
)
cv.10f.i         =      predict(cv.10f.sspline, lambda)
cv.10f[ ,i]      =      cv.10f.i$y

cv.lo.sspline    =      smooth.spline(cv.lasso.lo$lambda, cv.lasso.lo$cvm,
df = length(cv.lasso.lo$lambda)-1
)
cv.lo.i          =      predict(cv.lo.sspline, lambda)
cv.lo[ ,i]       =      cv.lo.i$y

extraErr[ ,i]    =      colSums((lasso.fit$beta - beta.star)^2) + o^2
print(i)
}


# les moyennes empiriques sur les MCMC échantillons ici 500
cv.3f.mean       =      rowMeans(cv.3f)
cv.3f.se         =      sqrt(apply(cv.3f, 1, var))/sqrt(MCMCsamples)
cv.5f.mean       =      rowMeans(cv.5f)
cv.5f.se         =      sqrt(apply(cv.5f, 1, var))/sqrt(MCMCsamples)
cv.10f.mean      =      rowMeans(cv.10f)
cv.10f.se        =      sqrt(apply(cv.10f, 1, var))/sqrt(MCMCsamples)
cv.lo.mean       =      rowMeans(cv.lo)
cv.lo.se         =      sqrt(apply(cv.lo, 1, var))/sqrt(MCMCsamples)
extraErr.mean    =      rowMeans(extraErr)
extraErr.se      =      sqrt(apply(extraErr, 1, var))/sqrt(MCMCsamples)

# écriture  dans une data.frame
eror             =      data.frame(  c( rep("3␣fold␣CV", m), rep("5␣fold␣CV", m),
rep("10␣fold␣CV", m),
```

```
  rep("LO", m),   rep("extraErr", m)
  ),
  n*c(lambda, lambda, lambda, lambda, lambda),
  c(cv.3f.mean, cv.5f.mean, cv.10f.mean,
  cv.lo.mean, extraErr.mean
  ),
  c(cv.3f.se, cv.5f.se, cv.10f.se, cv.lo.se,
  extraErr.se
  )
  )
  colnames(eror) =     c("method", "lambda", "err", "se")

  # écriture dans un fichier txt , pour éviter
  #cette longue itération prochainement.
  write.table(eror, "eror.txt", sep="\t")

  # visualisation de l'erreur en fonction de lambda.
  eror           =     read.table("eror.txt")
  eror.plot      =     ggplot(eror, aes(x=lambda, y = err, color=method)) +
  geom_line(size=0.5)
  eror.plot      =     eror.plot  +
  scale_x_log10(breaks = c(seq(10,200,10)))
  eror.plot      =     eror.plot  +
  theme(legend.text =
  element_text(colour="black",
  size=12, face="bold",
  family = "Courier"
  )
  )
  eror.plot      =     eror.plot  +
  geom_pointrange(aes(ymin=err-se, ymax=err+se),
  size=0.4,  shape=15
  )
  eror.plot      =     eror.plot  +
  theme(legend.title=element_blank()
  )
  eror.plot      =     eror.plot  +
  scale_color_discrete(breaks=
  c("3 fold CV","5 fold CV","10 fold CV",
  "LO","extraErr"
  )
  )
  eror.plot      =     eror.plot  + theme(axis.title.x = element_text(size=18))
  eror.plot      =     eror.plot  + theme(axis.title.y = element_text(size=16,
  face="bold",
  family = "Courier"
  )
  )
  eror.plot      =     eror.plot  + theme(axis.text.x = element_text(angle = 90))
  eror.plot      =     eror.plot  +
  xlab( expression(paste( lambda))) + ylab("error")
  eror.plot      =     eror.plot  +
  ggtitle("Out-of-sample error \n versus
  \n k-fold cross validation"
  )
  eror.plot      =     eror.plot  + theme(plot.title =
  element_text(hjust = 0.5,
  vjust = -10,size=14,
  face="bold",
  family = "Courier"
  )
```

```
)
eror.plot
```

B.2 Code R de la FIGURE 1.2

B.2.1 Graphe (a)

```
rm(list = ls())      # pour effacer l'environment
cat("\014")
library(class)
library(ggplot2)
library(gridExtra)  #pour gérer les grid pour graphiques
library(dplyr)
library(glmnet)
library(alocvBeta)  #pour calculer ALO et le risque
library(rmutil)     #pour les régressions non linéaires
library(tictoc)     #pour mésurer le temps d'exécution des scripts
library(scales)     #gérer les échelles dans la visualisation

set.seed(0)
# generate data
p_                =     seq(100, 500, 100) # variables explicatives
delta             =     5
rho               =     0.1
alpha_elnet       =     0.5
MCMCsamples       =     5 # nombre d'échantillons
m                 =     10
error.plot        =     list()

time.lo.mean      =     rep(0, length(p_))
time.alo.mean     =     rep(0, length(p_))
time.fit.mean     =     rep(0, length(p_))

time.lo.se        =     rep(0, length(p_))
time.alo.se       =     rep(0, length(p_))
time.fit.se       =     rep(0, length(p_))


spikeCov          =     1

for (i in 1:length(p_)){
time.lo.smp     =     rep(0, MCMCsamples)
time.alo.smp    =     rep(0, MCMCsamples)
time.fit.smp    =     rep(0, MCMCsamples)
p               =     p_[i]
n               =     p * delta
k               =     rho * n
#dfmax_          =     floor(0.5* min(n,p) )
if (spikeCov){
# spike covariance
a_              =    0.5
row_            =    c(1:p, 1:p)
column_         =    c(1:p, rep(p+1, p))
elmts           =    c(rep(sqrt(1-a_), p), rep(sqrt(a_), p))
F_              =    sparseMatrix(i = row_, j = column_, x = elmts)
} else {
a_              =     0.9 # AR(1) covariance
F_              =    matrix(rep(0, p*p), nrow = p, ncol = p)
for (row_ in 1:p){
for (column_ in 1:row_){
F_[row_, column_]   =   a_^abs(row_ - column_)
```

```
}
}
F_               =     t(F_)
}
for (s in 1:MCMCsamples){
beta.star        =     rep(0, p)
iS               =     sample(1:p, k)
beta.star[iS]    =     rlaplace(k, m=0, s=1/sqrt(2))
X                =     F_ %*% matrix(rnorm( n*ncol(F_), mean = 0, sd = 1 ),
nrow = ncol(F_), ncol = n
)
X                =     t(X)
y                =     X %*% beta.star + rnorm( n, mean = 0, sd = 1 )


lambdaS          =    exp(seq(log(1/n), log(100/n), length.out = m))

ptm              =     proc.time()
lo               =     cv.glmnet(X, y, alpha = alpha_elnet,
intercept = FALSE,
standardize = FALSE, lambda = lambdaS,
nfolds = n, type.measure = "deviance"
)
ptm              =     proc.time() - ptm
time.lo.smp[s]   =     ptm["elapsed"]

ptm              =     proc.time()
fit              =     glmnet(X, y, alpha = alpha_elnet,  intercept = FALSE,
standardize = FALSE, lambda = lambdaS)
ptm              =     proc.time() - ptm
time.fit.smp[s]  =     ptm["elapsed"]

ptm              =     proc.time()
alo              =     glmnetALO(X, y, glm_obj = fit, alpha = alpha_elnet,
standardize = FALSE,
type.measure = "deviance"
)
ptm              =     proc.time() - ptm
time.alo.smp[s]  =     time.fit.smp[s] + ptm["elapsed"]
}
time.fit.mean[i]     =  mean(time.fit.smp)
time.lo.mean[i]      =  mean(time.lo.smp)
time.alo.mean[i]     =  mean(time.alo.smp)

time.fit.se[i]     =  sd(time.fit.smp)/sqrt(MCMCsamples)
time.lo.se[i]      =  sd(time.lo.smp)/sqrt(MCMCsamples)
time.alo.se[i]     =  sd(time.alo.smp)/sqrt(MCMCsamples)
cat(sprintf("n␣=␣%s|␣p␣=␣%s␣\n", n, p))
cat(sprintf("TIME:␣lo␣=␣%.2f␣+ou-␣%.2f|alo␣=␣%.2f+ou-%.2f|
fit␣=%.2f+ou-␣%.2f␣\n",
time.lo.mean[i], time.lo.se[i], time.alo.mean[i], time.alo.se[i],
time.fit.mean[i], time.fit.se[i]
)
)
cat(sprintf("--------------------------------------␣\n"))
}

time             =     data.frame( c( rep("LO", length(p_)),
rep("ALO", length(p_)),
rep("FIT", length(p_))
),
```

```
c(p_, p_, p_),
c(time.lo.mean, time.alo.mean,
time.fit.mean
),
c(time.lo.se, time.alo.se, time.fit.se)
)
colnames(time) =     c("method", "p", "time", "se")


# écriture dans un fichier
write.table(time, "time_elnet.txt", sep = "\t")

time = read.table("time_elnet.txt")

time.plot      =     ggplot( time, aes(x=p, y = time, color=method)) +
geom_line(size=0.5
)
time.plot      =     time.plot  +
theme(legend.text = element_text(colour="black",
size=12, face="bold",
family = "Courier"
)
)
time.plot      =     time.plot  + geom_pointrange(aes(ymin=time-se,
ymax=time+se
),
size=0.4,  shape=15
)
time.plot      =     time.plot  + theme(legend.title=element_blank())
time.plot      =     time.plot  + scale_color_discrete(breaks=
c("LO","ALO","FIT")
)
time.plot      =     time.plot  + theme(axis.title.x = element_text(size=16,
family = "Courier"
),
axis.text.x  = element_text(angle=0,
vjust=0.5,
size=16),
axis.text.y  = element_text(angle=0,
vjust=0.5,
size=16))
time.plot      =     time.plot  + theme(axis.title.y = element_text(size=16,
family = "Courier"
)
)
time.plot      =     time.plot  + xlab("p=number␣of␣predictors") + ylab("time(sec)")
#time.plot      =     time.plot  + ggtitle("computational complexity")
time.plot      =     time.plot  + theme(plot.title = element_text(hjust = 0.5,
vjust = -10,
size=14,
face="bold",
family = "Courier"
)
)
time.plot      =     time.plot  + annotation_logticks() +
scale_y_log10(breaks = trans_breaks("log10",
function(x) 10^x),
labels = trans_format(
"log10",
math_format(10^.x))
)
```

```
time.plot      =     time.plot  +scale_x_log10(breaks = p_)

print(time.plot)
#ggsave("time_plot_elasticnet.png",time.plot)
# pour exporter les images avec ggplot2
```

B.2.2 Graphe (b)

```
rm(list = ls())
cat("\014")
library(class)
library(ggplot2)
library(gridExtra)
library(dplyr)
library(glmnet)
library(alocvBeta)
library(rmutil)
library(tictoc)
library(scales)

set.seed(0)
# generate data
p_               =     seq(200, 1000, 200)
delta            =     1
rho              =     0.1
alpha_elnet      =     1
MCMCsamples      =     10
m                =     10

time.lo.mean     =     rep(0, length(p_))
time.alo.mean    =     rep(0, length(p_))
time.fit.mean    =     rep(0, length(p_))

time.lo.se       =     rep(0, length(p_))
time.alo.se      =     rep(0, length(p_))
time.fit.se      =     rep(0, length(p_))


spikeCov         =     0

for (i in 1:length(p_)){
time.lo.smp    =     rep(0, MCMCsamples)
time.alo.smp   =     rep(0, MCMCsamples)
time.fit.smp   =     rep(0, MCMCsamples)
p              =     p_[i]
n              =     p * delta
k              =     rho * n
#dfmax_          =     floor(0.5* min(n,p) )
if (spikeCov){
# spike covariance
a_             =    0.5
row_           =    c(1:p, 1:p)
column_        =    c(1:p, rep(p+1, p))
elmts          =    c(rep(sqrt(1-a_), p), rep(sqrt(a_), p))
F_              =    sparseMatrix(i = row_, j = column_, x = elmts)
} else {
a_               =     0.9 # AR(1) covariance
F_                =    matrix(rep(0, p*p), nrow = p, ncol = p)
for (row_ in 1:p){
for (column_ in 1:row_){
F_[row_, column_]   =   a_^abs(row_ - column_)
```

```
}
}
F_               =    t(F_)
}
for (s in 1:MCMCsamples){
beta.star        =     rep(0, p)
iS               =     sample(1:p, k)
beta.star[iS]    =     rlaplace(k, m=0, s=1/sqrt(2))
X                =     F_ %*% matrix(rnorm( n*ncol(F_), mean = 0, sd = 1 ),
nrow = ncol(F_), ncol = n
)
X                =     t(X)
py               =    exp(X %*% beta.star) / (1 + exp(X %*% beta.star))
y                =    rep(0, n)
for (ss in 1:n){
y[ss]=rbinom(1,1,py[ss])
}
lambdaS          =    exp(seq(log(0.1/n), log(10/n), length.out = m))

ptm              =     proc.time()
lo               =     cv.glmnet(X, y, family = "binomial",
alpha = alpha_elnet,  intercept = FALSE,
standardize = FALSE, lambda = lambdaS,
nfolds = n, type.measure = "deviance"
)
ptm              =     proc.time() - ptm
time.lo.smp[s]   =     ptm["elapsed"]

ptm              =     proc.time()
fit              =     glmnet(X, y, family = "binomial",
alpha = alpha_elnet,  intercept = FALSE,
standardize = FALSE, lambda = lambdaS
)
ptm              =     proc.time() - ptm
time.fit.smp[s]  =     ptm["elapsed"]

ptm              =     proc.time()
alo              =     glmnetALO(X, y, glm_obj = fit,
alpha = alpha_elnet,
standardize = FALSE,
type.measure = "deviance"
)
ptm              =     proc.time() - ptm
time.alo.smp[s]  =     time.fit.smp[s] + ptm["elapsed"]
}
time.fit.mean[i]     =  mean(time.fit.smp)
time.lo.mean[i]      =  mean(time.lo.smp)
time.alo.mean[i]     =  mean(time.alo.smp)

time.fit.se[i]     =  sd(time.fit.smp)/sqrt(MCMCsamples)
time.lo.se[i]      =  sd(time.lo.smp)/sqrt(MCMCsamples)
time.alo.se[i]     =  sd(time.alo.smp)/sqrt(MCMCsamples)
cat(sprintf("n␣=␣%s|␣p␣=␣%s␣\n", n, p))
cat(sprintf("TIME:␣lo␣=␣%.2f␣+ou-␣%.2f|␣alo␣=␣%.2f+ou-␣%.2f|
fit␣=%.2f+ou-␣%.2f␣\n", time.lo.mean[i], time.lo.se[i],
time.alo.mean[i], time.alo.se[i], time.fit.mean[i],
time.fit.se[i]))
cat(sprintf("--------------------------------------␣\n"))
}
```

```
time            =     data.frame( c( rep("LO", length(p_)),
rep("ALO", length(p_)),
rep("FIT", length(p_))
),
c(p_, p_, p_),
c(time.lo.mean,
time.alo.mean,
time.fit.mean
),
c(time.lo.se,
time.alo.se,
time.fit.se)
)
colnames(time) =     c("method", "p", "time", "se")

# écriture des résultats dans un fichier .txt
write.table(time, "time_logistic.txt", sep = "\t")

# lecture et visualisation
time = read.table("time_logistic.txt")
time.plot       =     ggplot(time, aes(x=p, y = time, color=method)) +
geom_line(size=0.5)
time.plot       =     time.plot  +
theme( legend.text =
element_text(colour="black",
size=12, face="bold",
family = "Courier"
)
)
time.plot       =     time.plot  +
geom_pointrange(aes(ymin=time-se, ymax=time+se),
size=0.4,  shape=15
)
time.plot       =     time.plot  + theme(legend.title=element_blank())
time.plot       =     time.plot  +
scale_color_discrete(breaks=c("LO","ALO","FIT"))
time.plot       =     time.plot  +
theme(axis.title.x = element_text(size=16,
family = "Courier"
),
axis.text.x  = element_text(angle=0,
vjust=0.5,
size=16
),
axis.text.y  = element_text(angle=0,
vjust=0.5,
size=16
)
)
time.plot       =     time.plot  +
theme(axis.title.y = element_text(size=16,
family = "Courier"
)
)
time.plot       =     time.plot  + xlab("p=number␣of␣predictors") +
ylab("time(sec)")
#time.plot      =     time.plot  + ggtitle("computational complexity")
time.plot       =     time.plot  +
theme(plot.title =
element_text(hjust = 0.5,
vjust = -10,
```

```
size=14,
face="bold",
family = "Courier"
)
)
time.plot      =     time.plot  + annotation_logticks() +
scale_y_log10(breaks =
trans_breaks("log10", function(x) 10^x),
labels = trans_format("log10", math_format(10^.x)
)
)
time.plot      =     time.plot  +scale_x_log10(breaks = p_)

print(time.plot)

# ggsave("Figure_2_logiqtic.png",time.plot)  # exportation de l'image en format .png
```

B.2.3 Graphe (c)

```
rm(list = ls())
cat("\014")
library(class)
library(ggplot2)
library(gridExtra)
library(dplyr)
library(glmnet)
library(alocvBeta)
library(rmutil)
library(tictoc)
library(scales)
set.seed(1)
# generate data
p_                =     seq(500, 5500, 1000)
#p_                =     seq(150, 200, 10)
delta             =     0.1 # rapport entre n et p
rho               =     0.1 # rapport entre k et p
alpha_elnet       =     0.5
MCMCsamples       =     5
m                 =     10
error.plot        =     list()

time.lo.mean      =     rep(0, length(p_))
time.alo.mean     =     rep(0, length(p_))
time.fit.mean     =     rep(0, length(p_))

time.lo.se        =     rep(0, length(p_))
time.alo.se       =     rep(0, length(p_))
time.fit.se       =     rep(0, length(p_))


spikeCov          =     1

for (i in 1:length(p_)){
time.lo.smp     =     rep(0, MCMCsamples)
time.alo.smp    =     rep(0, MCMCsamples)
time.fit.smp    =     rep(0, MCMCsamples)
p               =     p_[i]
n               =     p * delta
k               =     rho * n
#dfmax_          =     floor(0.5* min(n,p) )
if (spikeCov){
```

```
# spike covariance
a_              =    0.5
row_            =    c(1:p, 1:p)
column_         =    c(1:p, rep(p+1, p))
elmts           =    c(rep(sqrt(1-a_), p), rep(sqrt(a_), p))
F_               =    sparseMatrix(i = row_, j = column_, x = elmts)
} else {
a_                =     0.9 # AR(1) covariance
F_                =    matrix(rep(0, p*p), nrow = p, ncol = p)
for (row_ in 1:p){
for (column_ in 1:row_){
F_[row_, column_]   =   a_^abs(row_ - column_)
}
}
F_               =    t(F_)
}

# pour s'assurer que var(x^T*beta.star) = 1
F_               =    F_  / sqrt(sum(F_[1, ]^2) * k)
for (s in 1:MCMCsamples){
beta.star       =     rep(0, p)
iS              =     sample(1:p, k)
beta.star[iS]   =     rlaplace(k, m=0, s=1/sqrt(2))
X               =     F_ %*% matrix(rnorm( n*ncol(F_), mean = 0, sd = 1 ),
nrow = ncol(F_), ncol = n
)
X               =     t(X)
ez              =    exp(X %*% beta.star)

y               =    rep(0, n)
for (obs in 1:n){
y[obs]=rpois(1,ez[obs])
}
lambdaS         =    exp(seq(log(1/n), log(100/n), length.out = m))

ptm              =     proc.time()
lo              =     cv.glmnet(X, y, family = "poisson",
alpha = alpha_elnet,  intercept = FALSE,
standardize = FALSE, lambda = lambdaS,
nfolds = n, type.measure="mae"
)
ptm              =     proc.time() - ptm
time.lo.smp[s]   =     ptm["elapsed"]

ptm              =     proc.time()
fit              =     glmnet(X, y, family = "poisson",
alpha = alpha_elnet,  intercept = FALSE,
standardize = FALSE, lambda = lambdaS
)
ptm              =     proc.time() - ptm
time.fit.smp[s] =     ptm["elapsed"]

ptm              =     proc.time()
alo              =     glmnetALO(X, y, glm_obj = fit,
alpha = alpha_elnet,
standardize = FALSE,
type.measure = "mae"
)
ptm              =     proc.time() - ptm
time.alo.smp[s] =     time.fit.smp[s] + ptm["elapsed"]
}
```

```
time.fit.mean[i]     =  mean(time.fit.smp)
time.lo.mean[i]      =  mean(time.lo.smp)
time.alo.mean[i]     =  mean(time.alo.smp)

time.fit.se[i]     =  sd(time.fit.smp)/sqrt(MCMCsamples)
time.lo.se[i]      =  sd(time.lo.smp)/sqrt(MCMCsamples)
time.alo.se[i]     =  sd(time.alo.smp)/sqrt(MCMCsamples)

cat(sprintf("n␣=␣%s|␣p␣=␣%s␣\n", n, p))
cat(sprintf("TIME:␣lo␣=␣%.2f␣+ou-␣%.2f|␣alo␣=␣%.2f+ou-␣%.2f|
fit␣=%.2f+ou-%.2f␣\n", time.lo.mean[i], time.lo.se[i],
time.alo.mean[i], time.alo.se[i], time.fit.mean[i],
time.fit.se[i]
)
)
cat(sprintf("-------------------------------------␣\n"))
}

time            =     data.frame( c(rep("LO", length(p_)),
rep("ALO", length(p_)),
rep("FIT", length(p_))
),
c(p_, p_, p_),
c(time.lo.mean, time.alo.mean, time.fit.mean),
c(time.lo.se, time.alo.se, time.fit.se)
)

colnames(time) =     c("method", "p", "time", "se")

write.table(time, "time_poisson.txt",sep = "\t")

time.plot      =     ggplot(time, aes(x=p, y = time, color=method)) +
geom_line(size=0.5)
time.plot      =     time.plot  +
theme(legend.text =
element_text(colour="black",
size=12, face="bold",
family = "Courier"
)
)
time.plot      =     time.plot  +
geom_pointrange(aes(ymin=time-se, ymax=time+se),
size=0.4,  shape=15
)
time.plot      =     time.plot  + theme(legend.title=element_blank())
time.plot      =     time.plot  + scale_color_discrete(breaks=
c("LO","ALO","FIT")
)
time.plot      =     time.plot  + theme(axis.title.x =
element_text(size=16, family = "Courier"
),
axis.text.x  = element_text(angle=0,
vjust=0.5,
size=12
),
axis.text.y  = element_text(angle=0,
vjust=0.5,
size=16
)
)
time.plot      =     time.plot  + theme(axis.title.y =
```

```
element_text(size=16,
family = "Courier"
)
)
time.plot      =     time.plot  + xlab("p=number␣of␣predictors") +
ylab("time(sec)")
#time.plot      =     time.plot  + ggtitle("computational complexity")
time.plot      =     time.plot  + theme(plot.title =
element_text(hjust = 0.5,
vjust = -10,
size=14,
face="bold",
family = "Courier"
)
)
time.plot      =     time.plot  + annotation_logticks() +
scale_y_log10(breaks = trans_breaks("log10",
function(x) 10^x
),
labels = trans_format("log10",
math_format(10^.x)
)
)
time.plot      =     time.plot  +scale_x_log10(breaks = p_)
print(time.plot)

#ggsave("time_plot_poisson.png",time.plot)
```

B.3 Code R de la FIGURE 2.1

```
rm(list = ls())
library(ggplot2)
library(class)
library(dplyr)
library(gridExtra)
library(glmnet)
#avec glmnet on peut construire les matrices  creuses (sparse)

# initialisation 10000
rho = 0.2 # 1/5
delta  = 0.2 # rapport entre p et n
p = 10000
n = p*delta
k = n*rho
m = 30 # on prend 30 valeurs de lambda

# on génère les données suivant différentes structure.
# de corrélation.Ici nous utilisons le spike :

design_distribution = "spike"

if(design_distribution == "spike"){
a_             =    0.3  # corrélation

# matrice de corrélation
C              =    diag( rep(1-a_,p)) + matrix(a_,p,p )
row_           =    c(1:p, 1:p)
column_        =    c(1:p, rep(p+1, p))
elmts          =    c(rep(sqrt(1-a_), p), rep(sqrt(a_), p))

F_             =    sparseMatrix(i = row_, j = column_, x = elmts)
```

```
# on a alors C =    F_%*%t(F_)
}
if(design_distribution == "iid"){
C             =    diag(rep(1,p))
F_            =    diag(rep(1,p))

}
set.seed(0)
X             =     F_ %*% matrix(rnorm( n*ncol(F_), mean = 0, sd = 1 ),
nrow = ncol(F_), ncol = n
) # matrice (p, n)
X             =     t(X)
#snr           =      1
o             =     1 #écart type sigma
beta          =      rep(0,p)
beta[1:k]     = 1    # les coefficients non nuls
#sont au nombre de k et sont fixés à 1

# normalisation des données utilisées
scaler        = ( t(beta)%*%C )%*% beta /o^2
scaler        = as.vector(scaler) # pour avoir un réel
#et non une matrice (1,1)

X             = X / sqrt(scaler)
F_            = F_ / sqrt(scaler)
C             = C / (scaler)

Xb            = X%*%beta
y             = Xb + rnorm(n, mean = 0 , sd = o)
# choix des valeurs possibles de lambdas
# le nombre maximale de coef non nuls est n*0.65
fit = glmnet(X , y , dfmax = floor(n*0.65))

lambda = fit$lambda
lambda_min = min(lambda)
lambda_max = max(lambda)


lambda  = seq(log10(lambda_min),log10(lambda_max), length.out  = m)
lambda  = 10^(lambda)

# entrainement du modèle maintenant :
fit  = glmnet(X , y , lambda = lambda)
lambda = fit$lambda
a0 = fit$a0
B  = fit$beta
df = fit$df
cat(sprintf("␣lambda␣min␣=␣%0.3f␣|␣␣lambda␣max␣=␣%0.3f␣\n",
min(lambda), max(lambda))
)
Err_extra  = rep(0, m)
ALO = rep(0,m)

for(i in 1:m){
if( df[i]>0){
S = which(B[,i] != 0)
XS = as.matrix(X[,S])
beta_hatS = B[S,i]
y_hat = XS%*%beta_hatS + a0[i]
dh = diag( XS%*%(solve(crossprod(XS)) )%*% t(XS) )
```

```
ALO[i] = mean( ((y-y_hat)/(1-dh))^2 )

beta_hat = B[,i]
Err_extra[i]  = o^2  +
t ( (beta- beta_hat) )%*%( C%*% (beta- beta_hat) )
r  = y - y_hat
}

cat(sprintf("␣itération␣=␣%d␣|␣␣df␣=␣%d␣|␣n␣=␣%d|␣lambda␣=␣%0.3f\n",
i,df[i] , n, lambda[i])
)
}

data_fig3 = data.frame( method = c( rep("Err_extra", m), rep("ALO", m)  ),
lambda = c( lambda, lambda),
Error = c(Err_extra, ALO)
)

#write.table(data_fig3, "data_fig3.txt", sep="\t")
#data_fig3          =     read.table("data_fig3.txt")
fig3_plot = ggplot( data = data_fig3, aes(x = lambda,
y = Error ,
color = method
)
) +
geom_line(size = 0.8)

fig3_plot = fig3_plot + scale_x_log10( )

fig3_plot = fig3_plot +
theme(legend.text = element_text(colour="black",
size=12,
face="bold",
family = "Courier"
)
)

# l'ordre d'apparition des catégories dans la légende
fig3_plot = fig3_plot +
scale_color_discrete( breaks = c("Err_extra", "ALO") )


fig3_plot = fig3_plot +
theme( axis.title.x = element_text( size = 16) )

fig3_plot = fig3_plot +
theme( axis.title.y = element_text( size = 16 ,
face = "bold" ,
family = "Courier"
)
) # face peut etre italic, bold

fig3_plot = fig3_plot + xlab(expression(lambda)) +
ylab("Prediction␣Error")

fig3_plot = fig3_plot + ggtitle( "p␣=␣10000␣,␣n␣=␣2000")

fig3_plot = fig3_plot + theme(plot.title =  element_text(hjust = 0.5))
```

```
fig3_plot
#ggsave("fig3.pdf",fig3_plot)
```

B.4 Code R de la FIGURE 4.1

```
rm(list = ls())
cat("\014")
library(class)
library(ggplot2)
library(dplyr)
library(glmnet)
library(alocvBeta)
library(rmutil)
library(tictoc)
library(latex2exp)

# initialisation
p_values          =     c(200, 1000, 10000)
n                 =     1000
k                 =     100
alpha_elnet       =     0.5
o                 =     1
m                 =     30 #number of lambdas


set.seed(1)

eror              =     list() # sauvegarde des résultats sous forme d'une liste
time.lo           = rep(0 , length(p_values))
time.alo           = rep(0 , length(p_values))
time.fit          = rep(0 , length(p_values))


for (ind_p in 1:length(p_values) ){
p                 = p_values[ind_p]
# le nombre maximal de coef à non nul
dfmax_            =     floor(min(p,n) * 0.7)
spikeCov          =     1 # Spiked(=1)

if (spikeCov){
# spike covariance
a_              =    0.5 # correlation coeeficient
row_            =    c(1:p, 1:p)
column_         =    c(1:p, rep(p+1, p))
elmts           =    c(rep(sqrt(1-a_), p),
rep(sqrt(a_), p)
)
F_               =    sparseMatrix(i = row_,
j = column_,
x = elmts
)
} else {
a_                =     0.9 # Toeplitz AR(1) covariance
F_                 =    matrix(rep(0, p*p),
nrow = p,
ncol = p
)
for (row_ in 1:p){
for (column_ in 1:row_){
F_[row_, column_]   =   a_^abs(row_ - column_)
}
```

```
}
F_              =     t(F_)
}
# to make sure the var(x^T * beta.star) = 1
F_              =     F_ / sqrt(sum(F_[1, ]^2) * k)
#C    =     F_ %*% t(F_)
beta.star       =     rep(0, p)
iS              =     sample(1:p, k)
beta.star[iS]   =     rlaplace(k, m=0, s=1/sqrt(2))
X               =     F_ %*% matrix(rnorm( n*ncol(F_),
mean = 0,
sd = 1
),
nrow = ncol(F_), ncol = n
)
X               =     t(X)
e               =     rnorm(n, mean = 0, sd = o)
y               =     X %*% beta.star + e
lambdaS         =     exp(seq(log(1/n), log(100/n), length.out = m))

ptm             =      proc.time()
lo              =      cv.glmnet(X, y, alpha = alpha_elnet,
intercept = FALSE,
standardize = FALSE,
lambda = lambdaS,
nfolds = n
)
ptm             =      proc.time() - ptm
time.lo[ind_p]          =      ptm["elapsed"]

ptm             =      proc.time()
fit             =      glmnet(X, y,
alpha = alpha_elnet,
intercept = FALSE,
standardize = FALSE,
lambda = lambdaS
)
ptm             =      proc.time() - ptm
time.fit[ind_p]         =      ptm["elapsed"]

ptm             =      proc.time()
alo             =      glmnetALO(X, y, glm_obj = fit,
alpha = alpha_elnet,
standardize = FALSE,
type.measure = "mse")
ptm             =      proc.time() - ptm
time.alo[ind_p]         =      time.fit[ind_p] + ptm["elapsed"]

cat(sprintf("n = %s | p = %s \n", n, p))
cat(sprintf("TIME: lo = %.2f | alo = %.2f | fit = %.2f \n",
time.lo[ind_p],
time.alo[ind_p],
time.fit[ind_p]
)
)
cat(sprintf(" df_max/p = %.3f \n", max(fit$df/p)))
cat(sprintf("------------------------------------ \n"))

eror[[ind_p]]           =      data.frame( c(rep("LO", length(lo$lambda)),
rep("ALO", length(alo$lambda))
),
```

```
n*c(lo$lambda, alo$lambda) ,
c(lo$cvm, alo$alom),
c(lo$cvsd,
rep(0, length(alo$lambda))
)
)
colnames(eror[[ind_p]]) =     c("method", "lambda", "err", "se")


}

time_fig4_article    = data.frame(time.fit,time.alo,time.lo)

# sauvegarde des données :écriture dans un fichier txt
write.table(eror[[1]],"eror1_fig4_arcticle.txt", sep="\t" )
write.table(eror[[2]],"eror2_fig4_arcticle.txt", sep="\t" )
write.table(eror[[3]],"eror3_fig4_arcticle.txt", sep="\t" )
write.table(time_fig4_article,
"time_fig4_article.txt",
sep = "\t"
)

# Pour la figre (a)
p = p_values[1]
eror1_fig4_arcticle = read.table("eror1_fig4_arcticle.txt")
time_fig4_article   = read.table("time_fig4_article.txt")
time.alo            = time_fig4_article$time.alo[1]
time.lo            = time_fig4_article$time.lo[1]
time.fit            = time_fig4_article$time.fit[1]

eror.plot      =     ggplot(eror1_fig4_arcticle,
aes(x=lambda, y = err, color=method)) +
geom_line(size=1)
eror.plot      =     eror.plot  + scale_x_log10()
eror.plot      =     eror.plot  +
theme(legend.text = element_text(colour="black",
size=16,
face="bold",
family = "Courier"
)
)
eror.plot      =     eror.plot  +
geom_pointrange(aes(ymin=err-se, ymax=err+se),
size=0.8,
shape=15
)
eror.plot      =     eror.plot  + theme(legend.title=element_blank())
eror.plot      =     eror.plot  +
scale_color_discrete(breaks=c("LO", "ALO"))
eror.plot      =     eror.plot  +
theme( axis.title.x = element_text(size=24),
axis.text.x  = element_text(angle=0,
vjust=0.5,
size=14
),
axis.text.y  = element_text(angle=0,
vjust=0.5,
size=14
)
)
```

```
eror.plot      =     eror.plot  + xlab( expression(paste( lambda))) +
ylab("")
eror.plot      =     eror.plot  +
theme(plot.title = element_text(hjust = 0.5,
vjust = -32,
size=12,
family = "Courier"
)
)
eror.plot      =     eror.plot  +
ggtitle((sprintf("n=%s,
p=%s
\n\n␣LO:%0.2f(sec)
\n␣ALO:%0.2f(sec)
\n␣FIT:%.2f(sec)",
n,p,
time.lo,
time.alo,
time.fit
)
)
)


print(eror.plot)


# Pour la figure (b), il suffit de prendre le deuxième élément
# dans le code qui fait la figure (a)

# pour la figure (c), il faut prendre maintenant
# le troisième élément.
```

B.5 Code R de la FIGURE 4.2

```
rm(list = ls()) #delete objects
cat("\014") # pour nettoyer la console
library(class)
library(ggplot2)
library(dplyr)
library(glmnet)
library(alocvBeta)
library(rmutil)
library(tictoc)
library(latex2exp)
p_values          =     c(200, 1000, 10000)
n                 =     1000
k                 =     100
alpha_elnet       =     1
m                 =     30

# sauvegarde des résultats sous forme d'une liste

set.seed(0)
eror              =     list()

# les temps de calcul
time.lo           = rep(0 , length(p_values))
time.alo           = rep(0 , length(p_values))
time.fit          = rep(0 , length(p_values))
```

```
for (ind_p in 1:length(p_values)){
p                 = p_values[ind_p]
# ici nous considérons le toeplitz covar
spikeCov          =     0

if (spikeCov){
# spike covariance
a_              =    0.5
row_            =    c(1:p, 1:p)
column_         =    c(1:p, rep(p+1, p))
elmts           =    c(rep(sqrt(1-a_), p), rep(sqrt(a_), p))
F_               =    sparseMatrix(i = row_, j = column_, x = elmts)
} else {
a_                =     0.9
F_                 =    matrix(rep(0, p*p), nrow = p, ncol = p)
for (row_ in 1:p){
for (column_ in 1:row_){
F_[row_, column_]   =   a_^abs(row_ - column_)
}
}
F_               =    t(F_)
}

# to make sure the var(x^T * beta.star) = 1
F_                =    F_ / sqrt(sum(F_[1, ]^2) * k)
# C   =    F_ %*% t(F_)
beta.star       =    rep(0, p)
iS              =    sample(1:p, k)
beta.star[iS]   =    rlaplace(k, m=0, s=1/sqrt(2))
X               =    F_ %*% matrix(rnorm( n*ncol(F_), mean = 0, sd = 1 ),
nrow = ncol(F_),
ncol = n
)
X               =    t(X)
py              =    exp(X %*% beta.star) / (1 + exp(X %*% beta.star))
y               =    rep(0, n)
for (obs in 1:n){
y[obs]=rbinom(1,1,py[obs])
# B(1,p) = bernoulli
}

lambdaS         =    exp(seq(log(0.1/n), log(10/n), length.out = m))


ptm             =     proc.time()
lo              =     cv.glmnet(X, y, family = "binomial",
alpha = alpha_elnet,
intercept = FALSE,
standardize = FALSE,
lambda = lambdaS,
nfolds = n,
type.measure="class"
)
ptm             =     proc.time() - ptm
time.lo[ind_p]         =     ptm["elapsed"]

ptm             =     proc.time()
fit             =     glmnet(X, y, family = "binomial",
alpha = alpha_elnet,
intercept = FALSE,
standardize = FALSE,
```

```
lambda = lambdaS)
ptm             =     proc.time() - ptm
time.fit[ind_p]       =     ptm["elapsed"]

# essayons les calculs manuels pour comparer
# avec les résulatats du package alocvBeta
ptm             =     proc.time()
alo_raw         =     rep(0, length(fit$lambda))
for (i in 1:length(fit$lambda)) {
z_alo  = rep(0,n)
if (fit$df[i] > 0) {
S         =     which(fit$beta[ ,i] != 0)
XS        =     as.matrix(X[ ,S])
beta.hat  =     fit$beta[, i]
ez        =     exp(XS %*% beta.hat[S])
ld        =     ez/(1+ez) - y
ldd       =     ez/(1+ez)^2
diag_ldd  =     sparseMatrix(i = 1:n ,
j = 1:n,
x = as.numeric(ldd)
)
J         =     t(XS) %*% diag_ldd %*% XS +
fit$lambda[i] * n * (1-alpha_elnet) *
sparseMatrix(i = 1:fit$df[i] ,
j = 1:fit$df[i],
x = rep(1,fit$df[i])
)
H         =     XS %*% solve(J, t(XS)) %*% diag_ldd
z_alo     =     XS %*% beta.hat[S] + (ld/ldd) *
diag(H) / (1-diag(H))
}
alo_raw[i] =  mean(as.numeric(exp(z_alo)/(1+exp(z_alo))>0.5) != y)
}
ptm             =     proc.time()    -     ptm
time.alo_raw    =     time.fit[ind_p]      +     ptm["elapsed"]

# maintenant calculons avec le package
# il est à noter que pour calculer ALO avec le package
# alocBeta, il faut utiliser un objet glm
# provenant du package glmnet.
ptm             =     proc.time()
alo             =     glmnetALO(X, y, glm_obj = fit,
alpha = alpha_elnet,
standardize = FALSE,
type.measure = "class"
)
ptm             =     proc.time() - ptm
time.alo[ind_p]       =     time.fit[ind_p] + ptm["elapsed"]

cat(sprintf("n = %s | p = %s \n", n, p))
cat(sprintf("TIME: lo = %.2f | alo = %.2f | fit = %.2f \n",
time.lo[ind_p], time.alo[ind_p], time.fit[ind_p]
)
)
cat(sprintf(" df_max/p = %.3f \n", max(fit$df/p)))

cat(sprintf("-------------------------------------\n"))

eror[[ind_p]]           =     data.frame(c(rep("LO", length(lo$lambda)),
rep("ALO", length(alo$lambda))
),
```

```
n*c(lo$lambda, alo$lambda) ,
c(lo$cvm, alo$alom),
c(lo$cvsd, rep(0, length(alo$lambda))))
colnames(eror[[ind_p]]) =     c("method", "lambda", "err", "se")

}


time_fig5_article    = data.frame(time.fit,time.alo,time.lo)


# sauvegarde des données :écriture dans un fichier txt
write.table(eror[[1]],"eror1_fig5_arcticle.txt", sep="\t" )
write.table(eror[[2]],"eror2_fig5_arcticle.txt", sep="\t" )
write.table(eror[[3]],"eror3_fig5_arcticle.txt", sep="\t" )
write.table(time_fig5_article,"time_fig5_article.txt",sep = "\t")

# Pour visualiser la figure (c)
p = p_values[3]
eror3_fig5_arcticle = read.table("eror3_fig5_arcticle.txt")
time_fig5_article   = read.table("time_fig5_article.txt")
time.alo            = time_fig5_article$time.alo[3]
time.lo            = time_fig5_article$time.lo[3]
time.fit           = time_fig5_article$time.fit[3]



eror.plot      =     ggplot(eror3_fig5_arcticle ,
aes(x=lambda, y = err, color=method)) +
geom_line(size=1)
eror.plot      =     eror.plot  + scale_x_log10()
eror.plot      =     eror.plot  +
theme(legend.text = element_text(colour="black",
size=16,
face="bold",
family = "Courier"
)
)
eror.plot      =     eror.plot  +
geom_pointrange(aes(ymin=err-se, ymax=err+se),
size=0.8,
shape=15
)
eror.plot      =     eror.plot  + theme(legend.title=element_blank())
eror.plot      =     eror.plot  + scale_color_discrete(breaks=c("LO", "ALO"))
eror.plot      =     eror.plot  + theme(axis.title.x = element_text(size=24),
axis.text.x  = element_text(angle=0,
vjust=0.5,
size=14
),
axis.text.y  = element_text(angle=0,
vjust=0.5,
size=14
)
)
eror.plot      =     eror.plot  + xlab( expression(paste( lambda))) + ylab("")
eror.plot      =     eror.plot  + theme(plot.title = element_text(hjust = 0.5,
vjust = -32,
size=12,
family = "Courier"
)
```

```
)
eror.plot      =     eror.plot  +
ggtitle((sprintf("n=%s,␣p=%s
\n\n␣LO:%0.2f(sec)
\n␣ALO:%0.2f(sec)
\n␣FIT:%.2f(sec)",
n,p,
time.lo,
time.alo,
time.fit
)
)
)


print(eror.plot)

# Pour visualiser les graphes (a) et (b)
# il suffit de remplacer le troisième  élément par le premier
# ou le deuxième
```

B.6 Code R de la FIGURE 4.3

```
rm(list = ls())#delete objects
cat("\014")
library(class)
library(ggplot2)
library(dplyr)
library(glmnet)
library(alocvBeta)
library(rmutil)
library(tictoc)
library(latex2exp)
p_values          =     c(200, 1000, 10000)
n                 =     1000
k                 =     100
alpha_elnet       =     0.5
m                 =     30

set.seed(1)
# sauvegarde des résultats sous forme d'une liste
eror              =     list()
# les temps de calcul
time.lo           = rep(0 , length(p_values))
time.alo           = rep(0 , length(p_values))
time.fit          = rep(0 , length(p_values))
for (ind_p in 1:length(p_values)){
p                 = p_values[ind_p]
spikeCov        =     1

if (spikeCov){
# spike covariance
a_              =    0.5
row_            =    c(1:p, 1:p)
column_         =    c(1:p, rep(p+1, p))
elmts           =    c(rep(sqrt(1-a_), p), rep(sqrt(a_), p))
F_               =    sparseMatrix(i = row_, j = column_, x = elmts)
} else {
a_                =     0.9 # AR(1) covariance
F_                 =    matrix(rep(0, p*p), nrow = p, ncol = p)
for (row_ in 1:p){
```

```
for (column_ in 1:row_){
F_[row_, column_]   =   a_^abs(row_ - column_)
}
}
F_               =    t(F_)
}

# to make sure the var(x^T * beta.star) = 1
F_               =    F_ / sqrt(sum(F_[1, ]^2) * k)
# C   =    F_ %*% t(F_)
beta.star        =    rep(0, p)
iS               =    sample(1:p, k)
beta.star[iS]    =    rlaplace(k, m=0, s=1/sqrt(2))
X                =    F_ %*% matrix(rnorm( n*ncol(F_), mean = 0, sd = 1 ),
nrow = ncol(F_),
ncol = n
)
X                =    t(X)
ez               =    exp(X %*% beta.star)
y                =    rep(0, n)
for (obs in 1:n){
y[obs]=rpois(1,ez[obs])
}

lambdaS          =    exp(seq(log(1/n), log(100/n), length.out = m))

ptm              =     proc.time()
lo               =     cv.glmnet(X, y, family = "poisson",
alpha = alpha_elnet,
intercept = FALSE,
standardize = FALSE,
lambda = lambdaS, nfolds = n,
type.measure="mae"
)
ptm              =     proc.time() - ptm
time.lo[ind_p]           =     ptm["elapsed"]

ptm              =     proc.time()
fit              =     glmnet(X, y, family = "poisson",
alpha = alpha_elnet,
intercept = FALSE,
standardize = FALSE,
lambda = lambdaS
)
ptm              =     proc.time() - ptm
time.fit[ind_p]          =     ptm["elapsed"]

# calcul manuel  de ALO
ptm              =     proc.time()
alo_raw          =     rep(0, length(fit$lambda))
for (i in 1:length(fit$lambda)) {
z_alo  = rep(0,n)
if (fit$df[i] > 0) {
S         =    which(fit$beta[ ,i] != 0)
XS        =    as.matrix(X[ ,S])
beta.hat  =    fit$beta[, i]
ez        =    exp(XS %*% beta.hat[S])
ld        =    ez - y
ldd       =    ez
diag_ldd  =    sparseMatrix(i = 1:n ,
j = 1:n,
```

```
x = as.numeric(ldd)
)
J         =     t(XS) %*% diag_ldd %*% XS +
fit$lambda[i] * n * (1-alpha_elnet) *
sparseMatrix(i = 1:fit$df[i] ,
j = 1:fit$df[i],
x = rep(1,fit$df[i])
)
H         =     XS %*% solve(J, t(XS)) %*% diag_ldd
z_alo     =     XS %*% beta.hat[S] + (ld/ldd) *
diag(H) / (1-diag(H))
}
alo_raw[i] =     mean(abs(y-exp(z_alo)))
}
ptm             =     proc.time()    -     ptm
time.alo_raw    =     time.fit[ind_p]        +     ptm["elapsed"]



ptm             =     proc.time()
alo             =     glmnetALO(X, y, glm_obj = fit,
alpha = alpha_elnet,
standardize = FALSE,
type.measure = "mae"
)
ptm             =     proc.time() - ptm
time.alo[ind_p]        =     time.fit[ind_p]  + ptm["elapsed"]

cat(sprintf("n = %s | p = %s \n", n, p))
cat(sprintf("TIME: lo = %.2f | alo = %.2f | fit = %.2f \n",
time.lo[ind_p],
time.alo[ind_p],
time.fit[ind_p]
)
)
cat(sprintf("---------------------------------------- \n"))

eror[[ind_p]]           =     data.frame(c(rep("LO",
length(lo$lambda)
),
rep("ALO",
length(alo$lambda))
),
n*c(lo$lambda, alo$lambda) ,
c(lo$cvm, alo$alom),
c(lo$cvsd,
rep(0, length(alo$lambda))
)
)
colnames(eror[[ind_p]]) =     c("method", "lambda", "err", "se")

}


time_fig6_article    = data.frame(time.fit,time.alo,time.lo)

# sauvegarde des données :écriture dans un fichier txt
write.table(eror[[1]],"eror1_fig6_arcticle.txt", sep="\t" )
write.table(eror[[2]],"eror2_fig6_arcticle.txt", sep="\t" )
write.table(eror[[3]],"eror3_fig6_arcticle.txt", sep="\t" )
write.table(time_fig6_article,"time_fig6_article.txt",sep = "\t")
```

```
# Pour visualiser la figure (a)
p = p_values[1]
eror1_fig6_arcticle = read.table("eror1_fig6_arcticle.txt")
time_fig6_article   = read.table("time_fig6_article.txt")
time.alo            = time_fig6_article$time.alo[1]
time.lo             = time_fig6_article$time.lo[1]
time.fit            = time_fig6_article$time.fit[1]

eror.plot      =     ggplot(eror1_fig6_arcticle  ,
aes(x=lambda, y = err, color=method)) +
geom_line(size=1)
eror.plot      =     eror.plot  + scale_x_log10()
eror.plot      =     eror.plot  + theme(legend.text =
element_text(colour="black",
size=16,
face="bold",
family = "Courier"
)
)
eror.plot      =     eror.plot  +
geom_pointrange(aes(ymin=err-se, ymax=err+se),
size=0.8,
shape=15
)
eror.plot      =     eror.plot  + theme(legend.title=element_blank())
eror.plot      =     eror.plot  + scale_color_discrete(breaks=c("LO", "ALO"))
eror.plot      =     eror.plot  + theme(axis.title.x = element_text(size=24),
axis.text.x  = element_text(angle=0,
vjust=0.5,
size=14
),
axis.text.y  = element_text(angle=0,
vjust=0.5,
size=14
)
)
eror.plot      =     eror.plot  + xlab( expression(paste( lambda))) +
ylab("")
eror.plot      =     eror.plot  + theme(plot.title = element_text(hjust = 0.5,
vjust = -32,
size=12,
family = "Courier"
)
)
eror.plot      =     eror.plot  + ggtitle((sprintf("n=%s,␣p=%s
\n\n␣LO:%0.3f(sec)
\n␣ALO:%0.3f(sec)
\n␣FIT:%.3f(sec)",n,p,
time.lo,
time.alo,
time.fit)
)
)


print(eror.plot)

# les graphes (b) et (c) s'obtiennent en prenant
# le deuxième et troisième élément au lieu du premier.
```

B.7 Code R de la FIGURE 5.1

Le code est exactement le même que celui de la FIGURE 4.1. Seulement il faut faire la modification suivante dans la partie initialisation :

```
alpha_elnet      = 1
# cela correspond à gamma = 0 et donc
# l'estimateur LASSO
```

B.8 Code R approximation de Schmidt : FIGURE 5.2

```
# Fonction approximation de schmidt
function_schmidtNew = function(alpha, beta){
expOfalphaXbeta  = exp(alpha*beta)
(1/alpha)*(log(1+expOfalphaXbeta)+log(1+(1/expOfalphaXbeta)))
}

# visualisation avec la fonction plot de R

# graphe (a)
#png("approx_sch_1.png")
# png pour exporter la figure en format .png
x1 = seq(-0.2,0.2, length.out = 50)
plot(x1,abs(x1),col= 1,type = 'l',
main = "Pour un vecteur [-0.2,0.2]",
ylab = expression("|X|"),
xlab = expression("X")
)
points(x1,function_schmidtNew(10,x1),col = 2)
points(x1,function_schmidtNew(25,x1),col = 3)
points(x1,function_schmidtNew(100,x1),col = 4)
legend("topright",
c("réel", "alpha = 10","alpha = 65","alpha= 100"),
lty=1:1,
col=c(1, 2,3,4)
)

# graphe(b)


#png("approx_sch_2.png")
x2 = seq(-1,1, length.out = 50)
plot(x2,abs(x2),col= 1,type = 'l',
main = "Pour un vecteur [-1,1]",
ylab = expression("|X|"),
xlab = expression("X")
)
points(x2,function_schmidtNew(10,x2),col = 2)
points(x2,function_schmidtNew(25,x2),col = 3)
points(x2,function_schmidtNew(100,x2),col = 4)
legend("topright",
c("réel","alpha = 10","alpha = 65","alpha= 100"),
lty=1:1,
col=c(1, 2,3,4)
)
```

B.9 Code R approximation de la FIGURE 5.3

```
# le code de la fonction function_schmidtNew est donné
# en haut, ici on l'appelle juste
```

```
# on modifie la composante de gradient
# pour éviter les instabilités numériques
# dans le cas où le produit de alpha et beta est
# plus grand que 10 en valeur absolue, on prend
# juste son signe.
functionComponentOfGradient=function(alphabetai){
if (abs(alphabetai)<10) {
expalphabetai  = exp(alphabetai)
(expalphabetai-1/expalphabetai)/(expalphabetai+1/expalphabetai+2)
}
else sign(alphabetai)}

# la dérivée de l'approximation de Schmidt,
# étant donnée un vecteur.
function_schmidt_primeNEW= function(alpha,beta){
alphabeta = alpha *beta
sapply(alphabeta,functionComponentOfGradient)
}

# dérivée seconde de l'approximation de schmidt
# on fixe par défaut la valeur minimale de
# cette dérivée seconde à lowerBound pour éviter
# les instabilités
function_schmidt_secondeNEWnew = function(alpha,beta,lowerBound){
expOfalphaXbeta  = exp(alpha*beta)
tempValue = ( 2*alpha )/( (expOfalphaXbeta+(1/expOfalphaXbeta)+2) )
pmax( tempValue,rep( lowerBound,length(tempValue) ) )
}

# critère de moindre carré ordinaire
F_ols = function(X,y,beta,lambda,alpha){
0.5*( sum( (y-X%*%beta)^2) ) + lambda*sum(function_schmidtNew(alpha , beta))
}


# cette fonction permet d'estimer les coefficients de regréssion
# linéaire avec la pénalisation de Schmidt.
# ce programme ne donne pas l'intercept
# verbose est booléen qui permet d'afficher les messages
# lors des itérations
estimate_schmidt_Damped_ols_2NEWNEWNEWnew = function(X,
y,
alpha,
lambda,
nbOfIterationsMax=100,
tolerance=10^(-4),
betaStart=matrix(0,ncol(X),1),
lowerBound=0.0001,
verbose=FALSE){
# Nous initialisons tous les coefficients à zéro
beta_vector = betaStart

# à la sortie les coefficients estimés auront les noms des variables s'il y a en
rownames(beta_vector) = colnames(X)
tXX = crossprod(X)
tXy = crossprod(X,y)
# gradient
grad = ( tXX%*%beta_vector - tXy ) +
lambda*function_schmidt_primeNEW(alpha,beta_vector)

# Pour compter le nombre d'itération qui sera effectué
```

```
k = 0

# nous cherchons une précision de 10^(-4) tant que c'est pas attend, on continue.

# Pour voir la valeur de i pris à chaque itération k pour la recherche
#de direction, voir l'explication de la méthode Damped-Newton
I = c()

repeat{

# on fait une copie de l'ancienne estimation ou de l'estimation de démarrage
beta_vector_old = beta_vector

# on recalcule le gradient de nouveau
grad  = ( tXX%*%beta_vector - tXy )+
lambda*function_schmidt_primeNEW(alpha,beta_vector)


# la matrice hessienne précédée du signe -
correctionOftXX  = lambda*
function_schmidt_secondeNEWnew(alpha,beta_vector,lowerBound=lowerBound)
hessian = tXX + diag(as.vector(correctionOftXX))
if(verbose==TRUE){
 cat("\n min of diag of correction of t(X)%*%X=", min(correctionOftXX),"\n")
}

# la direction
#if(verbose==TRUE) cat("\n Hessian=", hessian,"\n")

# factorisation de Cholesky pour la matrice hessian
# qui doit être symétrique et définie positive
Uprov = chol(hessian)
#if(verbose==TRUE) cat("\n Uprov=", Uprov,"\n")

search_direction =  - backsolve(Uprov, backsolve(Uprov, grad, transpose = TRUE))

# partie Damped

i = 0
imax = 4
# while(  norm( grad+(1/2^i)*search_direction,"2" ) >= norm(grad,"2")){
#   i = i+1
#
# }
if(verbose == TRUE){
cat("\n i=",i,
" new Newton-val de F=", F_ols(X,y,beta_vector+search_direction,lambda,alpha) ,"\n"
)
}
# on va augmenter i jusqu'à ce que la valeur prise par la
#fonction à minimiser diminue par rapport à la précédente.
# on fixe imax à 4 pour l'instant :
F_olsOFbetaOLD = F_ols(X,y,beta_vector_old,lambda,alpha)
i = 0
repeat{
F_olsOfbetaCandidate = F_ols(X,y,beta_vector+(1/2^i)*search_direction,lambda,alpha)
if(F_olsOfbetaCandidate < F_olsOFbetaOLD)  break
if (i==imax)  break
i= i+1
}
# on retient la valeur i qui a été utilisée pour réduire la valeur
```

```
#de la fonction objectif.
I = c(I,i)
# le nouveau vecteur beta obtenu:
beta_vector = beta_vector_old + (1/2^i)*search_direction

k = k+1

# condition d'arrêt
# si on atteind la précision tolerance , s'arrête
norm_relative = norm(beta_vector - beta_vector_old, "2")/norm(beta_vector,"2")
if(norm(grad,"2") < tolerance & norm_relative < tolerance){break}

# si on dépasse le nombre maximal  d'itération, on s'arrête
if(k > nbOfIterationsMax){
cat("\n L'algorithme n'a pas convergé avec ",nbOfIterationsMax, " itération(s)\n")
break
}
}
# on retoure le résultat sous forme d'une list
return(list(beta_vector=beta_vector,k=k,I=I))
}

# Les données de la première FIGURE
set.seed(0)
p               =     1000
n               =     250
k               =     50
beta.star       =     rep(0, p)
beta.star[1:k]  =     sqrt(5*10/9)/sqrt(k)
o               =     sqrt(2)
X_fig1               =     matrix(rnorm(n*p, mean = 0, sd = 1), ncol = p, nrow = n)
e               =     rnorm(n, mean = 0, sd = o)
y_fig1              =     X_fig1 %*% beta.star + e

# on estime donc les coefficients:
# pour le glmnet
fit_fig1_glmnet = glmnet(X_fig1,y_fig1,intercept = FALSE,lambda = 30/n,standardize = F

# pour notre fonction
fits_fig1_glmnet = list()
beta_vectors = list()

# on considère donc plusieurs valeurs de alpha
# hyperamètre de la pénalisation de Schmidt
alphas = c(15,25,50,65,150,200,250,400,600,1000)

# MAE Mean Absolute Error
# Sur le figure nous l'appelons
# plutôt AAE: Average Absolute Error
MAE_Damped_glmnet = rep(0,length(alphas))
MSE_Damped_glmnet = rep(0,length(alphas))
MAE_Damped_star = rep(0,length(alphas))
MSE_Damped_star = rep(0,length(alphas))

for(i in 1:length(alphas)){
print(i)
# on essaye de construire d'estimer les coefficients
# si cela ne marque pas toutes les sorties sont
# fixées à zéro
fits_fig1_glmnet[[i]] = try(estimate_schmidt_Damped_ols_2NEWNEWNEWnew(X_fig1,y_fig1,
alpha = alphas[i],
```

```
lambda = 30,
nbOfIterationsMax=200,
tolerance=10^(-4),
lowerBound=0.001,
verbose = FALSE),
silent = TRUE
)

if(!inherits(fits_fig1_glmnet[[i]], "try-error")){
beta_vectors[[i]] = fits_fig1_glmnet[[i]]$beta_vector

# Damped vs glmnet

MAE_Damped_glmnet[i] = mean( abs(beta_vectors[[i]] - coef(fit_fig1_glmnet)[2:1001]) )
MSE_Damped_glmnet[i] = mean( (beta_vectors[[i]]- coef(fit_fig1_glmnet)[2:1001]
)^2 )

# Damped vs beta_star
MAE_Damped_star[i] = mean( abs(beta_vectors[[i]] - beta.star) )
MSE_Damped_star[i] = mean( (beta_vectors[[i]]- beta.star  )^2 )


}
else{
MAE_Damped_glmnet[i] = 0
MSE_Damped_glmnet[i] = 0


MAE_Damped_star[i] = 0
MSE_Damped_star[i] = 0


}



}

# ces deux quantités sont uniques car dépendent pas de alpha
MAE_glmnet_star =  mean( abs(coef(fit_fig1_glmnet)[2:1001]-beta.star) )
MSE_glmnet_star =  mean( (coef(fit_fig1_glmnet)[2:1001]-beta.star )^2 )


result_alpha = data.frame(alphas,MAE_Damped_glmnet,MSE_Damped_glmnet,
MAE_Damped_star,MSE_Damped_star,
MAE_glmnet_star,MSE_glmnet_star
)
```

B.9.1 Graphe (a)

```
#png("Schmidt_vs_glmenetlasso_AAE.png")
plot(alphas,log10(result_alpha$MAE_Damped_star),
col = 2,
main = "AAE",
xlab = expression(alpha),
ylim = c(-2,-1),
ylab = "log10(Error)"
)

points(alphas,
log10(result_alpha$MAE_glmnet_star),
```

```
col = 1
)
legend("topright", c("Schmidt","glmnet"), lty=1:1, col=c(2, 1))
```

B.9.2 Graphe (b)

```
#png("Schmidt_vs_glmenetlasso_ASE.png")
plot(alphas,
log10(result_alpha$MSE_Damped_star),
col = 2,
main = "ASE",
xlab = expression(alpha),
ylab = "log10(Error)"
)
points(alphas,log10(result_alpha$MSE_glmnet_star),col = 1)
legend("topright", c( "Schmidt","glmnet"), lty=1:1, col=c(2, 1))
```

B.10 Code R de la FIGURE 5.4

```
library(ggplot2)
library(glmnet)
library(parallel) # pour la parallélisation

# initialisation 1000

rho = 0.2 # 1/5
delta  = 0.2
p = 1000
(n = p*delta)
k = n*rho
m = 20 # on avait pris 30 valeurs de lambda pour la FIG 3

design_distribution = "spike"

if(design_distribution == "spike"){
a_              =    0.3

C               =    diag( rep(1-a_,p)) + matrix(a_,p,p )

row_            =    c(1:p, 1:p)
column_         =    c(1:p, rep(p+1, p))
elmts           =    c(rep(sqrt(1-a_), p), rep(sqrt(a_), p))

F_              =    sparseMatrix(i = row_, j = column_, x = elmts)

# on a alors C =    F_%*%t(F_)
}
if(design_distribution == "iid"){
C            =    diag(rep(1,p))
F_           =    diag(rep(1,p))

}

set.seed(0)
X               =     F_ %*% matrix(rnorm( n*ncol(F_), mean = 0, sd = 1 ),
nrow = ncol(F_),
ncol = n
) # matrice (p, n)
X               =     t(X)
#  snr             =       1
o               =     1
```

```
beta            =       rep(0,p)
beta[1:k]       = 1

# normalisation des données utilisées
scaler          = ( t(beta)%*%C )%*% beta /o^2
scaler          = as.vector(scaler)

X               = X / sqrt(scaler)
F_              = F_ / sqrt(scaler)
C               = C / (scaler)

Xb              = X%*%beta
y               = Xb + rnorm(n, mean = 0 , sd = o)
```

B.10.1 Graphe (a)

```
# Cette fonction permet de calculer ALO en utilisant la fonction
# estimate_schmidt_Damped_ols_2NEWNEWNEWnew qui permet
# d'estimer les coefficients de regresion
# nous avons utilisé beta(le vrai vecteur des coefficients)
# qui n'estpas nécessaire pour calculer ALO,
# mais juste pour calculer l'Err_extra
ALO_alpha_olsNEWnew = function(X,
y,
alpha,
lambda,
nbOfIterationsMax=100,
tolerance = 10^(-4),
betaStart =matrix(0,ncol(X),1),
lowerBound = 0.01,
verbose=FALSE
){


beta_alpha = estimate_schmidt_Damped_ols_2NEWNEWNEWnew(X = X,
y = y,
alpha = alpha,
lambda = lambda,
nbOfIterationsMax = nbOfIterationsMax,
tolerance = tolerance,
betaStart = betaStart,
lowerBound = lowerBound ,
verbose = verbose
)$beta_vector
# t(X)%*%X     crossprod(X)
hessian =  lambda*diag( as.vector(function_schmidt_secondeNEWnew(alpha,
beta_alpha,
lowerBound = lowerBound
)
)
) +  crossprod(X)
# H_alpha = X%*%(solve(hessian)%*%t(X)) remplacé par:
H_alpha = X%*%(solve(hessian,t(X)) )


ALO_val = mean( ( y - (X%*%beta_alpha) -
( X%*%beta_alpha - y)*( diag(H_alpha)/(1-diag(H_alpha)) ) )^2 )

errExtrUnweigthed_val = mean( (beta_alpha - beta)^2 )

# on retourne les résultats sous forme d'une list
```

```
return(list(ALO_val = ALO_val,
beta_vector = beta_alpha,
errExtrUnweigthed_val = errExtrUnweigthed_val
)
)
}

# on utilise la fonction ci-dessus pour  calculer Err_extra en tenant compte
# de la corrélation entre les variables explicatives
# cette fonction prend juste en entrée lambda dans l'intention d'utiliser la
# parallélisation et donc diminuer les temps de calcul :
# on prend alpha=25 comme on peut le voir sur la figure
computeALO_alpha_errExtrUnweight_errExtrWeight = function(nTimeslambda){

alo_alpha  = try(ALO_alpha_olsNEWnew(as.matrix(X),
as.matrix(y),
alpha = 25,
lambda = nTimeslambda,
nbOfIterationsMax= 200,
lowerBound  = 0.01
) , silent = TRUE
)
if(!inherits(alo_alpha , "try-error")){
alo_result = alo_alpha$ALO_val
}
else{
alo_result = 0
}
errExtrWeightedByC = o^2 + sum ( ( t(F_)%*%( beta - alo_alpha$beta_vector )
)^2)

# on retourne cette fois-ci un vecteur et non  une liste
c(alo_result, alo_alpha$errExtrUnweigthed_val, errExtrWeightedByC)
}


# fonction ci-dessous retourne pour chaque élément d'une liste,
# la nième composante, le résultat est un vecteur
fun1 = function(lst, n){
sapply(lst, '[', n)
}


list3_Oflambdas = 10^seq(log10(0.0030),log10(0.080), length.out  = m)
#list3_Oflambdas = seq(0.025,0.1,length.out  = m)

# fait une parallélisation avec 4 coeur
# il est donc nécessaire d'avoir un ordinateur d'au moins
# 4 coeur pour tester ce programme
system.time({
results3 =  mclapply(list3_Oflambdas,
function(lambda){computeALO_alpha_errExtrUnweight_errExtrWeight(n*lambda)},
mc.cores = 4
)
})

# extrait ALO_alpha qui est premier élément de chaque composante
ALO_alpha_list3 = fun1(results3, 1)

# Erreur_extra_alpha est le troisième
erreurExtra_sample3 = fun1(results3, 3)
```

```
data_fig3cn = data.frame(method = c( rep("Err_extra_alpha", m), rep("ALO_alpha", m)
),
lambda = c( n*lambda, n*lambda),
Error = c(erreurExtra_sample3, ALO_alpha_list3)
)


#write.table(data_fig3cn, "data_fig3cn.txt", sep="\t")


#data_fig3cn          =     read.table("data_fig3cn.txt")
fig3_plot = ggplot( data = data_fig3cn, aes(x = lambda/n, y = Error , color = method))
geom_point(size = 0.8)

#fig3_plot = fig3_plot + scale_x_log10( )

fig3_plot = fig3_plot +
theme(legend.text = element_text(colour="black",
size=12,
face="bold",
family = "Courier"))


fig3_plot = fig3_plot +
scale_color_discrete( breaks = c("Err_extra_alpha", "ALO_alpha") )

fig3_plot = fig3_plot + theme( axis.title.x = element_text( size = 16) )

fig3_plot = fig3_plot +
theme( axis.title.y = element_text( size = 16 ,
face = "bold" ,
family = "Courier"
)
)

fig3_plot = fig3_plot + xlab(expression(lambda)) + ylab("Prediction␣Error")

fig3_plot = fig3_plot + ggtitle( "p␣=␣1000␣,␣n␣=␣200")

fig3_plot = fig3_plot + theme(plot.title =  element_text(hjust = 0.5))


fig3_plot

#ggsave("fig3cn.pdf",fig3_plot,width = 7 ,height = 3
```

B.10.2 Graphe (b)

```
# choix des valeurs possibles de lambdas
# nous avonns pris cette fois 30 valeurs de lambda
m    = 30
fit = glmnet(X , y , dfmax = floor(n*0.65))

lambda = fit$lambda
lambda_min = min(lambda)
lambda_max = max(lambda)


lambda  = seq(log10(lambda_min),log10(lambda_max), length.out  = m)
lambda  = 10^(lambda)
```

```
# entrainement du modèle maintenant :


fit  = glmnet(X , y , lambda = lambda)
lambda = fit$lambda
a0 = fit$a0
B  = fit$beta
df = fit$df
cat(sprintf("␣lambda␣min␣=␣%0.3f␣|␣␣lambda␣max␣=␣%0.3f␣\n",
min(lambda), max(lambda)
)
)

Err_extra  = rep(0, m)
ALO = rep(0,m)

for(i in 1:m){
if( df[i]>0){
S = which(B[,i] != 0)
XS = as.matrix(X[,S])
beta_hatS = B[S,i]
y_hat = XS%*%beta_hatS + a0[i]
dh = diag( XS%*%(solve(crossprod(XS)) )%*% t(XS) )

ALO[i] = mean( ((y-y_hat)/(1-dh))^2 )

beta_hat = B[,i]
Err_extra[i]  = o^2  + t ( (beta- beta_hat) )%*%( C%*% (beta- beta_hat) )
r  = y - y_hat
}

cat(sprintf("␣itération␣=␣%d␣|␣␣df␣=␣%d␣|␣n␣=␣%d|␣lambda␣=␣%0.3f\n",
i,df[i] , n, lambda[i]
)
)
}

# écriture du résultat:
data_fig3_1000 = data.frame(method = c( rep("Err_extra", m), rep("ALO", m)
),
lambda = c( lambda, lambda),
Error = c(Err_extra, ALO)
)

#write.table(data_fig3_1000, "data_fig3_1000.txt", sep="\t")

data_fig3_1000          =     read.table("data_fig3_1000.txt")
fig3_plot = ggplot( data = data_fig3_1000, aes(x = lambda, y = Error , color = method)
geom_point(size = 0.8)

fig3_plot = fig3_plot + theme(legend.text = element_text(colour="black",
size=12,
face="bold",
family = "Courier"
)
)


fig3_plot = fig3_plot + scale_color_discrete( breaks = c("Err_extra", "ALO") )
```

```
fig3_plot = fig3_plot + theme( axis.title.x = element_text( size = 16) )

fig3_plot = fig3_plot + xlim(c(0,0.8))
fig3_plot = fig3_plot + ylim(c(1,1.5))

fig3_plot = fig3_plot + theme( axis.title.y = element_text( size = 16 ,
face = "bold" ,
family = "Courier"
)
)

fig3_plot = fig3_plot + xlab(expression(lambda)) + ylab("Prediction Error")

fig3_plot = fig3_plot + ggtitle( "p = 1000, n = 200")

fig3_plot = fig3_plot + theme(plot.title =  element_text(hjust = 0.5))


fig3_plot
ggsave("fig3_1000.pdf",fig3_plot,width = 7 ,height = 3.5)
```

B.11 Code R de la FIGURE 5.5

Le code est le même que celui de la FIGURE 4.2, il suffit de faire le petit changement :

```
 a = 0.02
 # avant c'était a = 0.9
```

B.12 Code R de la FIGURE 5.6

Ici aussi il suffit de faire le petit changement :

```
a = 0.7
# avant c'était a = 0.9
```

B.13 Code R de la FIGURE 5.6

Ici également il suffit de faire le petit changement :

```
a = 0.99
# avant c'était a = 0.9
```