UNIVERSITÉ
DE MONTPELLIER

Compte-rendu de projet

Quelques méthodes de régression non linéaire

Rédigé par :
CORDOVAL Chloë, GOUJILI Nouhaila et LLINARES Laurent
- Master 1 MIND

2020/2021

Table des matières

1 Introduction **3**
1.1 Présentation des données simulées 3
1.2 Origine des données réelles 3
1.2.1 Description des variables d'étude 3

2 Problématique, modèle général et notations **4**

3 Validation croisée d'un modèle **4**

4 Régression linéaire sur indicatrices **5**
4.1 Questions préliminaires 5
4.2 Régression pénalisée sur indicatrices 6

5 Régression noyaux **16**
5.1 Estimateur de Nadaraya et Watson 16
5.2 Régression polynomiale à noyau sur une variable 19
5.3 Extension à plusieurs variables 23
5.4 Application sur des données réelles 25
5.4.1 Analyses et résultats 25

6 Régression spline **28**
6.1 L'ajustement traditionnel des splines de régression 28
6.2 Splines pénalisées 30

7 Conclusion **32**

8 Annexe **34**

1 Introduction

Nous considérons une réponse Y dont il s'agit de modéliser l'espérance sachant p variables explicatives $x^1, ..., x^p$.

Nous rappelons la formulation du modèle linéaire ci-dessous, où la variable expliquée s'écrit comme une fonction linéaire des variables explicatives :

$$\forall i : y_i = \beta_0 + \beta_1 x_{i,1} + ... + \beta_K x_{i,K} + \epsilon_i$$

Où y_i et les $x_{i,j}$ sont fixes et ϵ_i représente l'erreur.

Nous nous proposons de dépasser le modèle linéaire de Y en fonction des x^j.

Nous nous bornerons à des techniques pouvant se ramener à un modèle linéaire utilisant des fonctions non linéaires des x^j.

Le modèle ciblé est :

$$\boxed{\forall i : y_i = \sum_{j=1}^{p} f_j(x_i^j) + \epsilon_i}$$

avec $\epsilon_i \hookrightarrow N(0, \sigma^2)$, les ϵ_i étant indépendantes.

Dans ce modèle, f_j est une fonction non-linéaire de x^i. Il s'agira alors d'exprimer cette fonction dans une base spécifique.

Nous allons essayer plusieurs méthodes et ensuite comparer les résultats obtenus, d'abord sur des données simulées puis sur des données réelles.

1.1 Présentation des données simulées

Deux jeux de données nous ont été fournis par notre encadrant. Les deux contiennent une réponse avec pour le premier une seule variable explicative et pour le second deux variables explicatives.

1.2 Origine des données réelles

L'échantillon utilisé pour ce travail provient du site Web de l'Organisation des Nations Unies pour l'alimentation et l'agriculture des Nations Unies.

1.2.1 Description des variables d'étude

Les variables prises en considération dans notre étude concernent la prévalence en pourcentage de l'obésité dans chaque pays, la consommation alimentaire de viande (kg) et l'apport alimentaire des céréales - à l'exclusion de la bière en (kg).

2 Problématique, modèle général et notations

Nous allons être confrontés à la problématique suivante :

Comment se servir de la régression linéaire pour modéliser une relation non linéaire, en améliorant la capacité de prédiction de la variable à expliquée ?

La régression non linéaire permet de modéliser la relation entre des prédicteurs et une réponse lorsque des termes quadratiques ou cubiques ne sont pas adéquats.

Nous allons nous intéresser au modèle général suivant comme présenté précédemment :

$$\boxed{\forall i : y_i = \sum_{j=1}^{p} f_j(x_i^j) + \epsilon_i}$$

.

Dans ce rapport, f_j est une fonction non linéaire des x^j(variables explicatives).

y est la variable à expliquer.

ϵ représente le bruit.

3 Validation croisée d'un modèle

La validation croisée est une **méthode d'évaluation** de la qualité prédictive.

Elle permet de tirer plusieurs ensembles de calibration et de validation d'une même base de données et ainsi obtenir une estimation plus robuste, avec biais et variance, de la performance de prédiction du modèle.

Soit E l'échantillon de toutes les données et $E_{(-i)}$ l'échantillon privé de la i-ème observation.

Soit Z un ensemble de variables explicatives pour un certain type de régression .

Nous programmerons une fonction nommée "LOOSE" (Leave One Out Squared Error) calculant :

a) Une méthode d'estimation quelconque de y sur Z sur l'échantillon $E_{(-i)}$, régression dont elle tire l'estimateur $\hat{\beta}_{(-i)}$
b) La prédiction $\tilde{y}_i$de la réponse y_i
c) L'erreur quadratique de prédiction $\tilde{e}_i = (\tilde{y}_i - y_i)^2$
d) L'erreur quadratique de prédiction $\tilde{E} = \frac{1}{n}\sum_{i=1}^{n} \tilde{e}_i$

Dans la suite, nous calculerons une prédiction de la réponse pour une observation de variables explicatives que nous appliquerons au jeu de données simulées présenté précédemment.

4 Régression linéaire sur indicatrices

Nous découpons le domaine de chaque variable explicative x^j en tranches de fréquences égales, donc selon des quantiles $q^j_{k\alpha}$ de fréquences $k\alpha$, $k = 1$ à $K = \frac{1}{\alpha} - 1$.

4.1 Questions préliminaires

Nous noterons z^{jk}, l'indicatrice de l'intervalle $[q^j_{k\alpha}, q^j_{(k+1)\alpha}]$ du domaine de la variable x_j.

Par conséquent , $\boxed{z^{jk} = 1_{q^j_{k\alpha}; q^j_{(k+1)\alpha}}}$.

Nous définissons le produit scalaire suivant : $\langle f, g\rangle = \sum_i f_i g_i$ où les f_i sont les valeurs de la fonction étagée sur chaque intervalle.

Nous considérons dorénavant $k < c$.

Nous avons donc pour j :

$$\langle z^{jk}, z^{jc}\rangle = \langle 1_{[q^j_{k\alpha}, q^j_{(k+1)\alpha}]}\rangle, \langle 1_{[q^j_{c\alpha}, q^j_{(c+1)\alpha}]}\rangle$$

Or comme les intervalles sont disjoints :

$$\langle z^{jk}, z^{lk}\rangle = 1 * 0 + 0 * 1 = 0$$

Nous en déduisons donc que les indicatrices $\{z^{jk}, k = 1, ..., K\}$ forment une **base orthogonale** de fonctions étagées de la variable x_j constantes sur les intervalles inter-quantiles.

Au final plus nous découpons finement le domaine de chaque variable, mieux on pourra approcher une fonction quelconque de chaque x_j. En revanche, nous augmenterons le risque de sur-ajustement.

Montrons que dans le cas d'une unique variable explicative, le coefficient de l'indicatrice I_k est égal à la moyenne des y_i pour les individus dont la valeur de x appartient, lors d'une méthode des moindres carrés dans une régression linéaire sur ces indicatrices :

Nous avons : $\beta = (Z^T Z)^{-1} Z^T y$ avec $Z = (1_{I_1}(x_i).....1_{I_K}(x_i))_{i=1,..,n}$

D'où : $Z^T Z = \begin{bmatrix} \sum_{i=1}^n 1_{I_1}(x_i) & ... & ... & & & 0 & & \\ ... & ... & ... & & ... & ... & ... & \\ ... & & ... & & \sum_{i=1}^n 1_{I_k}(x_i) & ... & ... & \\ .... & & 0 & ... & & & & \\ ... & & ... & & ... & ... & \sum_{i=1}^n 1_{I_K}(x_i) & \end{bmatrix}$

$$\text{Et : } Z^T y = \begin{bmatrix} \sum_{i=1}^n y_i 1_{I_1}(x_i) \\ ... \\ ... \\ \sum_{i=1}^n y_i 1_{I_k}(x_i) \\ ... \\ ... \\ \sum_{i=1}^n y_i 1_{I_K}(x_i) \end{bmatrix} \quad \text{Ainsi : } \boxed{\beta = \left(\frac{\sum_{i=1}^n y_i 1_{I_k}(x_i)}{\sum_{i=1}^n 1_{I_k}(x_i)} \right)_{k=1,...K}}$$

Notre modèle est : $y_i = f(x_i) + \epsilon$ où f est une fonction non-linéaire de x_i et ϵ une erreur.

$$\text{Nous approchons } y_i \text{ par } y_i \approx \sum_{k=1}^K \beta_k 1_{I_k}(x_i) = (1_1(x_i).....1_K(x_i)) \begin{bmatrix} \beta_1 \\ ... \\ ... \\ \beta_k \\ ... \\ ... \\ \beta_K \end{bmatrix}$$

Donc pour un modèle avec plusieurs variables explicatives, nous aurons :

$$y_i \approx \sum_{i=1}^n f_p(x_i^p) = \sum_{p=1}^P \sum_{k=1}^K \beta_k^p 1_{I_k^p}(x_i^p) = (1_{1^1}(x_i^1).....1_{k^2}(x_i^2)....1_{K^p}(x_i^p)) \begin{bmatrix} \beta_1^1 \\ ... \\ \beta_K^1 \\ ... \\ \beta_1^p \\ ... \\ \beta_K^p \end{bmatrix}$$

Dans la suite, nous fabriquerons une fonction Zind de découpage d'une variable selon un nombre k de modalités passé en argument, et fournissant la matrice des indicatrices correspondantes. La matrice Z définie ci-dessus sera donc la juxtaposition de ces indicatrices dans notre application.

4.2 Régression pénalisée sur indicatrices

Dans la suite, Nous voudrions faire une **pénalisation sur la régression**, afin de limiter le sur-ajustement et de stabiliser les coefficients.

Pour cela on choisira une pénalité *ridge* de coefficient λ.

La régression ridge consiste à ajouter au critère d'ajustement une pénalité sur une norme euclidienne au carré du vecteur des coefficients. Cette norme sera à choisir parmi les métriques qui correspondent.

Nous résoudrons le programme des moindres carrés pénalisés :

Soit λ le coefficient de pénalité qui permet de contrôler l'impact de la pénalité. Il est à fixer.

$$\boxed{P_{\lambda,M} : \min_{\beta}(||y - Z\beta||_W^2 + \lambda||\beta||_M^2)}$$

Avec : $W = \frac{1}{n}Id_n$ et $S(\beta) = ||y - Z\beta||_W^2 + \lambda||\beta||_M^2$

Par conséquent, nous aurons :

$$S(\beta) = \frac{1}{n}y^T y - \frac{2}{n}\beta^T Z^T y + \frac{1}{n}\beta^T Z^T Z\beta + \lambda\beta^T M\beta$$

Nous cherchons $\hat{\beta}$ qui vérifiera $\nabla S(\hat{\beta}) = 0$:

Or, nous avons :

$\nabla S(\hat{\beta}) = \frac{-2}{n}Z^T y + \frac{1}{n}Z^T Z\hat{\beta} + 2\lambda M\hat{\beta} = 0$

$\iff \frac{1}{n}Z^T Z\hat{\beta} + \lambda M\hat{\beta} = \frac{1}{n}Z^T y$

$\iff (\frac{1}{n}Z^T Z + \lambda M)\hat{\beta} = \frac{1}{n}Z^T y \iff \hat{\beta} = (\frac{1}{n}Z^T Z + \lambda M)^{-1}\frac{1}{n}Z^T y$

Nous trouvons donc

$$\boxed{\hat{\beta} = \frac{1}{n}(\frac{1}{n}Z^T Z + \lambda M)^{-1}Z^T y}$$

Nous nous servirons de la fonction **Ridge** définie en annexe pour résoudre ce type de programme.

Pénalisation des différences entre coefficients successifs :

Nous cherchons la métrique D pour pénaliser les différences entres les coefficients des tranches voisines.

Nous posons :

$$A_1 = \begin{bmatrix} 1 & -1 & 0 & & & & ... & & \\ ... & 1 & -1 & & & ... & & & \\ & & ... & & ... & & & & \\ & & & ... & & ... & & & \\ & ... & & 1 & -1 & & & 0 & ... \\ & & ... & & ... & & & & \\ & & & ... & & ... & & 1 & -1 \end{bmatrix}$$

D'où : $$\begin{bmatrix} 1 & -1 & 0 & & & & ... & & \\ ... & 1 & -1 & & & ... & & & \\ & & ... & & ... & & & & \\ & & & ... & & ... & & & \\ & ... & & 1 & -1 & & & 0 & ... \\ & & ... & & ... & & & & \\ & & & ... & & ... & & 1 & -1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ ... \\ ... \\ ... \\ ... \\ ... \\ \beta k \end{bmatrix} = \begin{bmatrix} \beta_1 - \beta_2 \\ \beta_2 - \beta_3 \\ ... \\ ... \\ ... \\ ... \\ \beta_{k-1} - \beta k \end{bmatrix}$$

Nous trouvons donc :

$$\boxed{\beta^T A_1^T A_1 \beta = \sum_{i=1}^{p} (\beta_{i-1} - \beta_i)^2}$$

Par la suite, nous poserons D=$A_1^T A_1$.

Nous prendrons D pour métrique M dans la pénalité `ridge` qui permet de pénaliser les écarts successifs de coefficients. Ce qui permet pour la fonction en escalier de ne pas avoir de sauts brusques. Plus le coefficient λ de la pénalité sera élevé plus la fonction sera resserrée en une fonction constante.

Pour pénaliser les écarts des différences secondes, nous utilisons la formule suivante :

$$\sum_{i=1}^{K-2} ((\beta_{i+2} - \beta_{i+1}) - (\beta_{i+1} - \beta_i))^2 = \sum_{i=1}^{K-2} (\beta_{i+2} - 2\beta i + 1 + \beta_i)^2 = \beta^T A_2^T A_2 \beta$$

Avec

$$A_2 = \begin{bmatrix} 1 & -2 & 1 & & & & ... & & \\ ... & 1 & -2 & 1 & & ... & & & \\ & & ... & & ... & & & & \\ & & & ... & & ... & & & \\ & ... & & 1 & -2 & 1 & & 0 & ... \\ & & ... & & ... & & & & \\ & & & ... & & ... & 1 & -2 & 1 \end{bmatrix}$$

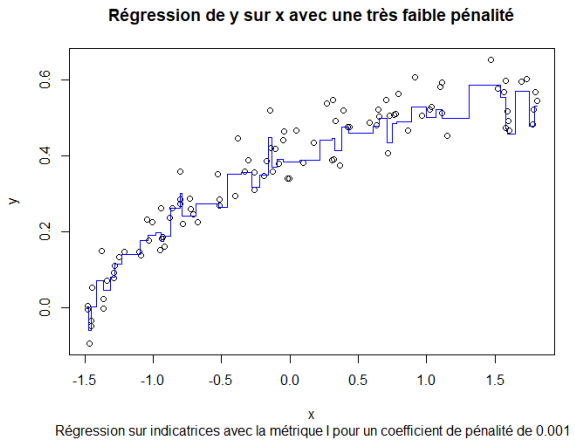
Par la suite, nous poserons $E = A_2^T A_2$. Cette matrice a pour particularité, en tant que métrique, de lisser les coefficients de manière à ce que leur progression soit régulière.

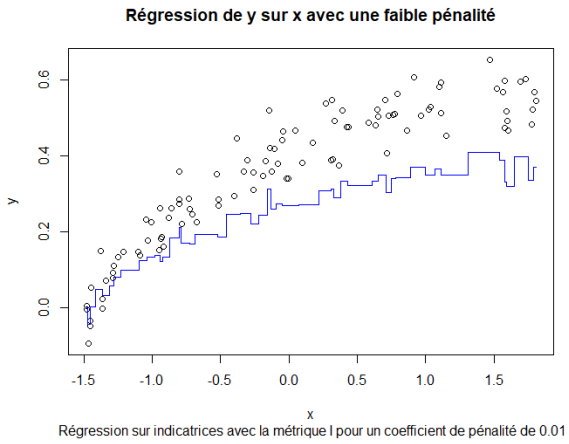
Ainsi avec un paramètre de pénalité λ très grand, nous tomberons dans le cas du sous-apprentissage. Nos coefficients seront tellement pénalisés que nous risquerons d'avoir une fonction d'hypothèse trop simple pour notre problème.

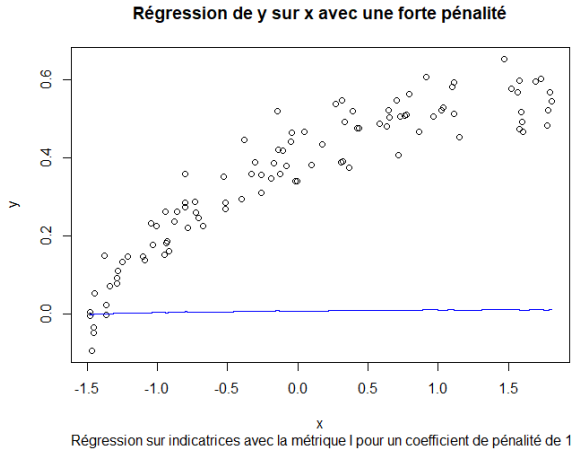
Dans la suite, nous allons présenter les graphiques de la régression sur indicatrices avec les trois métriques qui sont définies précédemment, pour les mêmes valeurs de λ.

Le paramètre *lambda* prend les valeurs suivantes : 0.001, 0.01 et 1.

Pour la métrique I :



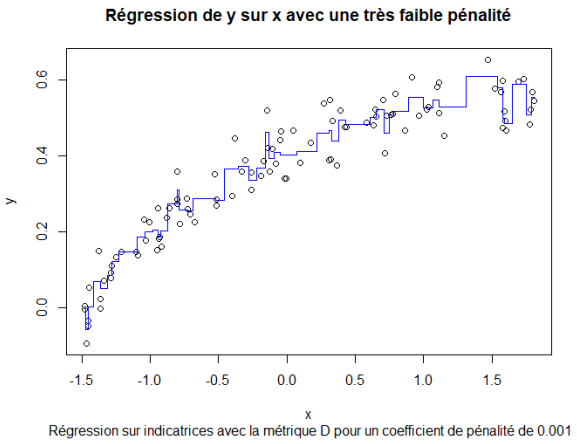




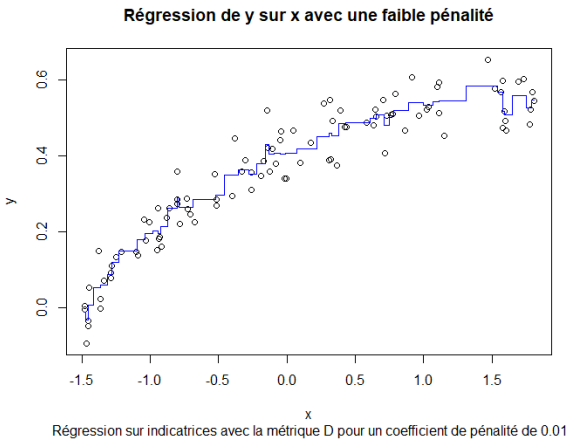
Nous remarquons que la régression sur indicatrices de métrique I et de pénalité $0,01$ et 1 écrase la courbe vers 0.

Cependant, on tombe sur le cas de sur-apprentissage pour la régression de métrique I et de pénalité 0.001.

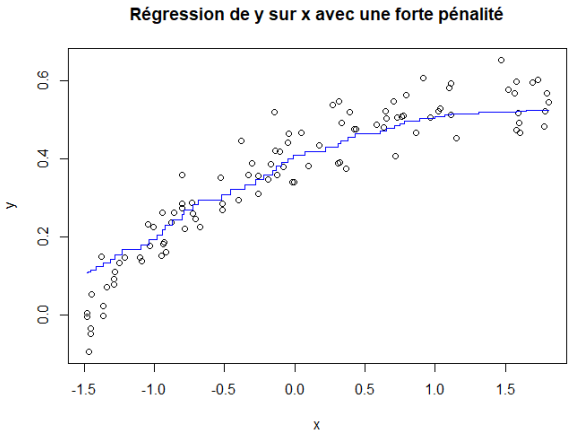
Pour la métrique D :



Régression sur indicatrices avec la métrique D pour un coefficient de pénalité de 0.001



Régression sur indicatrices avec la métrique D pour un coefficient de pénalité de 0.01

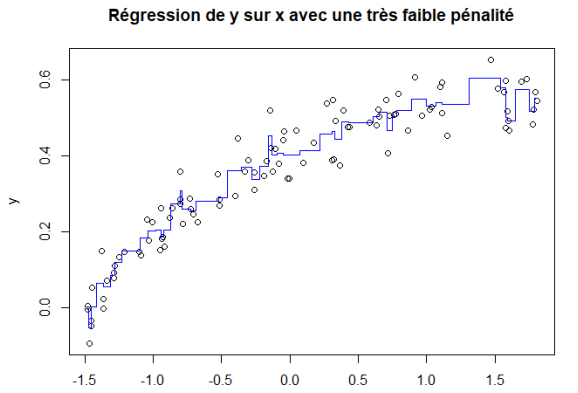


Régression sur indicatrices avec la métrique D pour un coefficient de pénalité de 1

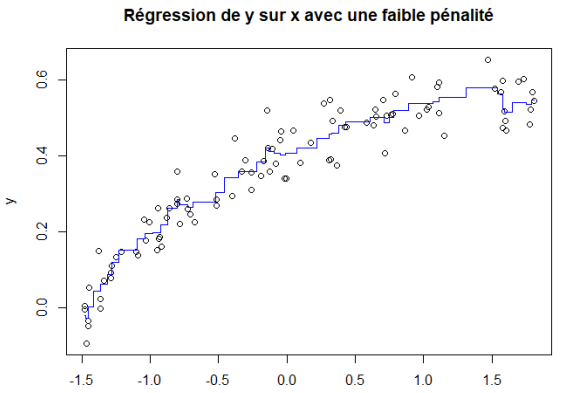
De même la régression sur indicatrices avec la métrique D définie précédemment pour un coefficient de pénalité de 1 affine la courbe en l'écrasant vers le centre.

De plus, pour un λ trop faible (*ici* 0.001), la pénalité a peu d'effets et nous tombons sur du "sur-apprentissage" ; en particulier l'erreur de validation augmente alors que l'erreur d'apprentissage continue à diminuer.

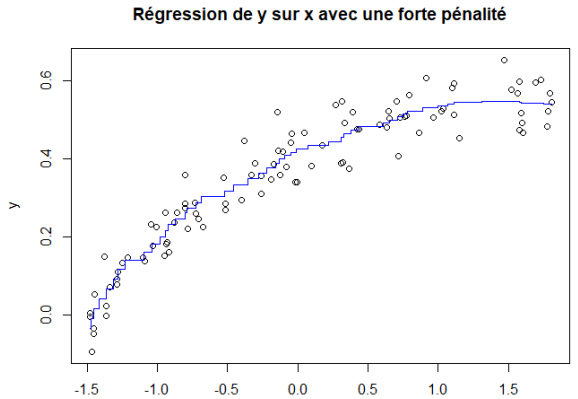
Pour la métrique E :



Régression sur indicatrices avec la métrique E pour un coefficient de pénalité de 0.001



Régression sur indicatrices avec la métrique E pour un coefficient de pénalité de 0.01



Régression sur indicatrices avec la métrique E pour un coefficient de pénalité de 1

Nous avons créer la fonction LOOLA qui grâce au `leave one out` permet de trouver λ optimum pour la métrique I respectivement la métrique D=A^TA, ainsi que la métrique E.

Nous présenterons une grille un peu grossière qui donne l'estimation de l'erreur de prédiction en `leave one out` pour différentes valeurs de λ et pour $\alpha = \frac{1}{50}$.

Nous augmentons la précision "en chiffres après la virgule" ce qui nous permet d'explorer du plus large au plus précis les différentes valeurs de λ et on sélectionne à chaque fois l'erreur minimale. La valeur de λ pour lequel l'erreur estimée est minimale est en rouge.

Pour l'identité I :

Précision	0.1	0.01	0.001	0.0001
λ	0	0	0	0.002
ϵ = erreur	0.00616	0.00616	0.00616	0.00611

Pour la métrique D=A^TA :

Précision	0.1	0.01	0.001
λ	1	0.1	0.97
ϵ = erreur	0.004483	0.003102166	0.00310205

Pour la métrique E : la matrice des écarts

Précision	10	0.1
λ	10	11.7
ϵ = erreur	0.002922605	0.002922241

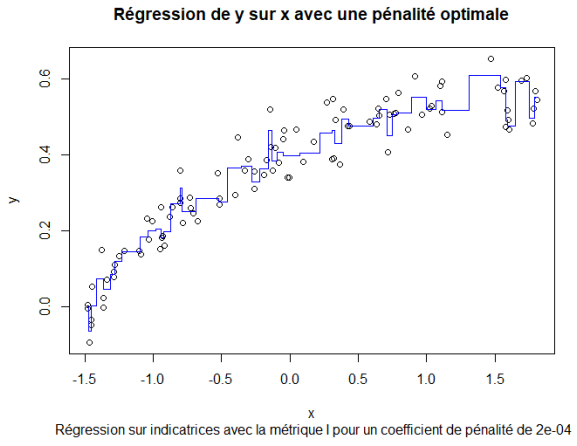
La métrique de I semble beaucoup moins performante et très sensible à la pénalité, la raison étant qu'elle ne régularise pas les sauts de la fonction en escalier. Les métriques D et E font deux fois moins d'erreurs et n'ont pas la même sensibilité à la pénalité. Ceci s'explique par leurs manières de régulariser en restant fidèles à l'allure du nuage de points.

Nous obtenons pour la métrique I, la valeur de λ ayant la plus petite erreur quadratique de 0.006120794 est 0.0002.

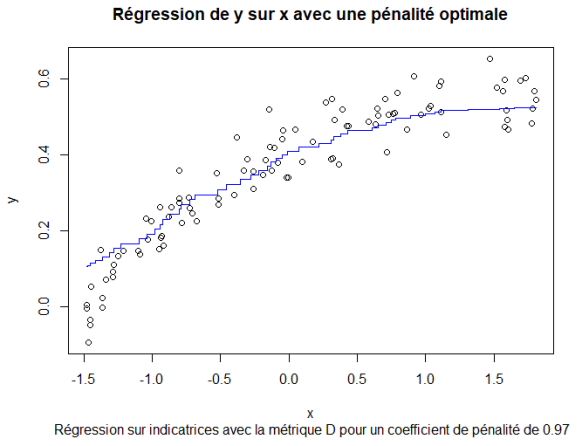
Pour la métrique D, nous trouvons une valeur de λ qui est égale à 0.97 ayant une erreur quadratique de 0.0031020.

Finalement pour la métrique E, la valeur de λ qui est égale à 11.7 minimise l'erreur qui vaut 0.002922241.

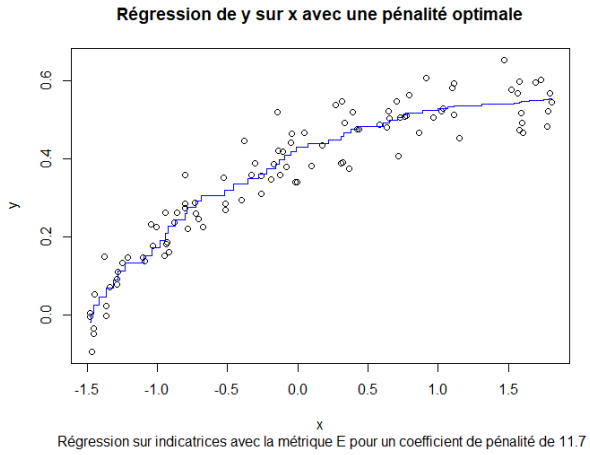
Dans la suite, nous présenterons les graphiques de ces valeurs de λ trouvées qui sont les optimums pour chaque métrique :



Régression sur indicatrices avec la métrique I pour un coefficient de pénalité de 2e-04

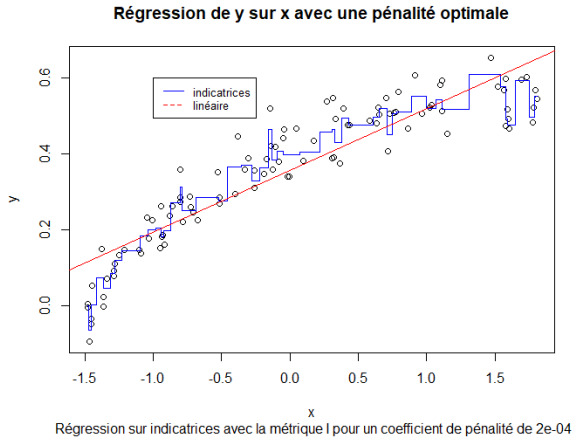


Régression sur indicatrices avec la métrique D pour un coefficient de pénalité de 0.97

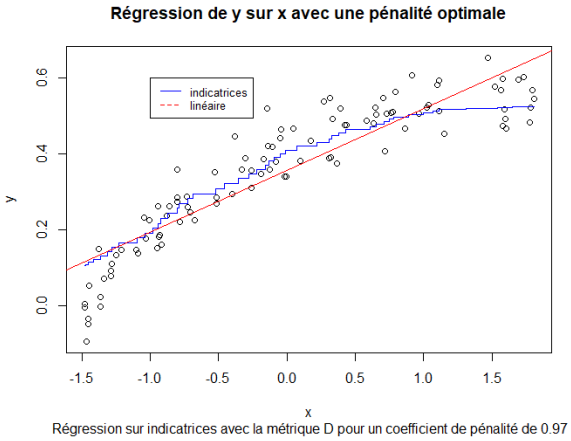


Régression sur indicatrices avec la métrique E pour un coefficient de pénalité de 11.7

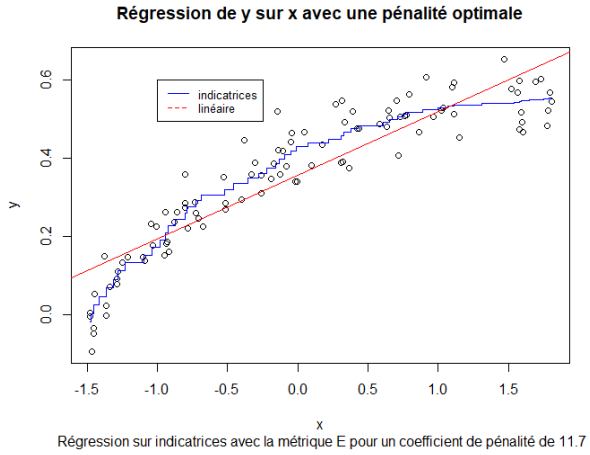
Les graphiques suivants représentent une comparaison de la régression de y sur x avec une pénalité optimale pour les trois métriques I, D et E, avec la "vraie" régression linéaire simple où l'erreur quadratique moyenne par `leave one out` est de **0.00627** :



Régression sur indicatrices avec la métrique I pour un coefficient de pénalité de 2e-04



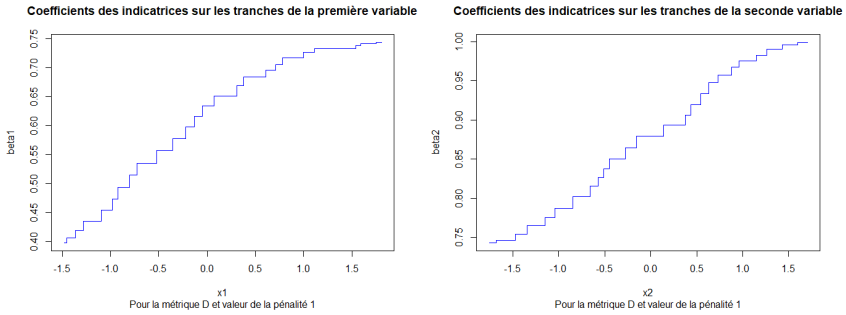
Régression sur indicatrices avec la métrique D pour un coefficient de pénalité de 0.97



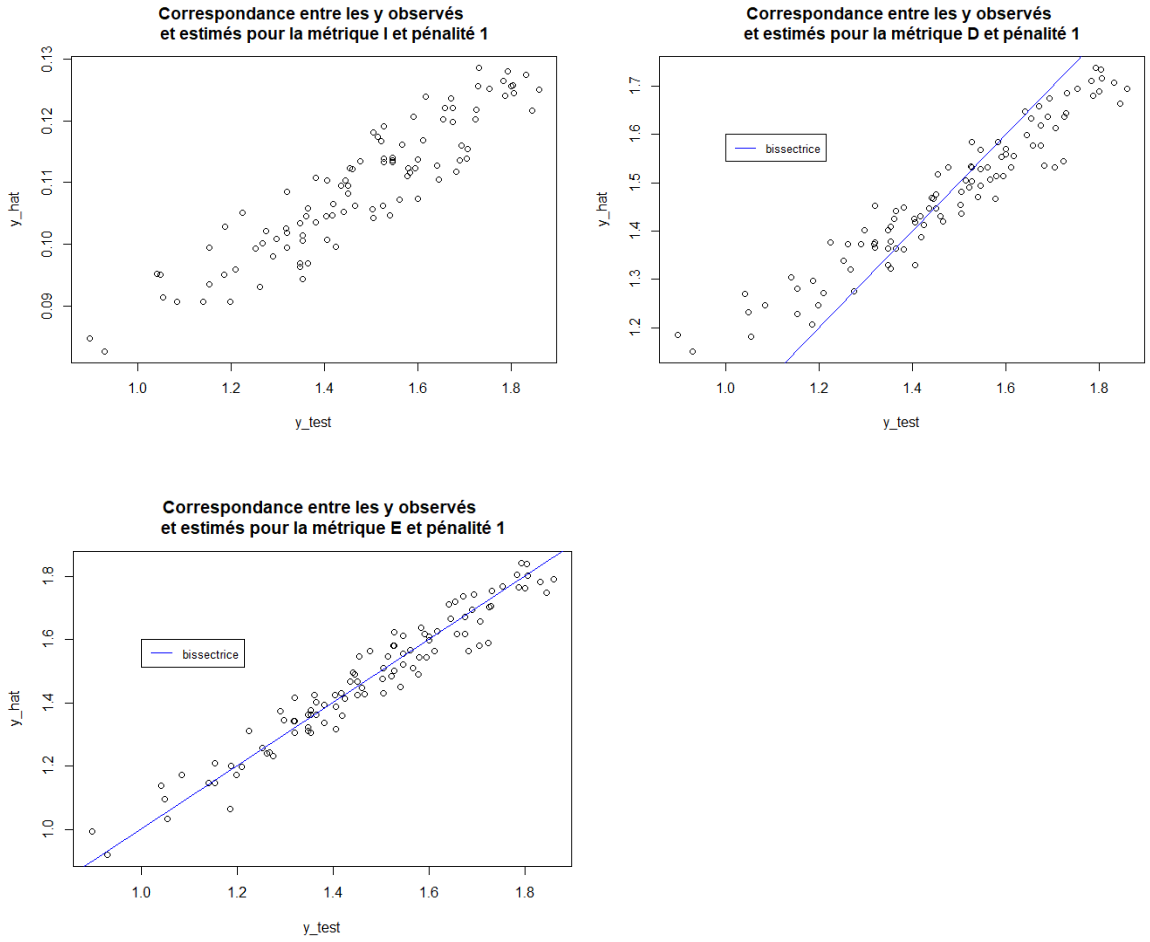
Régression sur indicatrices avec la métrique E pour un coefficient de pénalité de 11.7

Nous reprenons nos données simulées (avec deux variables explicatives) puis nous regardons la décomposition lorsqu'il y a plusieurs variables explicatives.

Pour cela nous présenterons cette décomposition sur la première variable ensuite sur la deuxième, pour la métrique D et valeur de pénalité égale à 1.



Afin d'observer l'efficacité du modèle dans le cas de deux variables explicatives , nous regarderons la différence entre les y_i et $\hat{y}_i$ où les y_i sont les variables dépendantes de nos données simulées pour les trois métriques I, D et E, et pour $\lambda = 1$.



Nous remarquons qu'il y a un biais très fort pour la métrique I où l'on ne voit pas la bissectrice. Cela se comprend puisqu'on a vu précédemment que la fonction en escalier s'écrasait, vers 0 quand nous augmentions la pénalité, sans tenir compte du nuage de point, elle s'en éloigne.

5 Régression noyaux

La régression sur indicatrices vue précédemment consiste à considérer comme voisines d'une observation, et avec le même poids, toutes celles qui sont dans la même tranche de x, cependant que toutes celles qui n'y sont pas reçoivent un poids nul. Ainsi que cette régression ne prend pas en compte le comportement global mais local des observations.

Nous regarderons dans cette partie la régression noyaux qui consiste à retrouver la continuité et qui permet d'utiliser toutes les observations avec un poids décroissant.

5.1 Estimateur de Nadaraya et Watson

Soit le noyau gaussien de "largeur" h,

$$K_h(t) = \frac{1}{h\sqrt{2\pi}} e^{-\frac{t^2}{2h^2}}$$

Nadaraya et Watson proposent d'estimer non-paramétriquement E(y|x) en chaque point x par :

$$\hat{y_h}(x) = \frac{\sum_{i=1}^{n} y_i K_h(x_i - x)}{\sum_{i=1}^{n} K_h(x_i - x)}$$

Nous pouvons représenter l'estimateur de Nadaraya-Watson comme une somme pondérée des y_i

$$\hat{y_h}(x) = \sum_{i=1}^{n} w_h(x_i - x) y_i$$

où $w_h(x_i - x) = \frac{K_h(x_i - x)}{\sum_{i=1}^{n} K_h(x_i - x)}$ est la fonction poids telle que $\sum_{i=1}^{n} w_h(x_i - x) = 1$

En effet l'espérance conditionnelle est définie par

$$E(Y|X = x) = \int y f(y|x)\, dy = \int y \frac{f(x,y)}{f(x)}\, dy.$$

En utilisant l'estimation de la densité du noyau pour la distribution conjointe f(x,y) et f(x) avec un noyau K, nous avons

$$\hat{f}(x,y) = \frac{1}{n} \sum_{i=1}^{n} K_h(x_i - x) K_h(y_i - y)$$

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x_i - x)$$

D'où

$$\hat{E}(Y|X=x) = \frac{\sum_{i=1}^{n} K_h(x_i - x) \int K_h(y_i - y)\ dy}{\sum_{i=1}^{n} K_h(x_i - x)} = \frac{\sum_{i=1}^{n} K_h(x_i - x) y_i}{\sum_{i=1}^{n} K_h(x_i - x)}$$

qui est l'estimateur Nadaraya-Watson.

Intuitivement, nous pouvons le voir comme la moyenne des y_i pondérée par la proximité des x_i à un x donné.

Par ailleurs, montrons que cet estimateur est la solution du programme :

$$\hat{y_h}(x) = argmin_{\beta_0} \sum_{i=1}^{n} (y_i - \beta_0)^2 K_h(x_i - x)$$

Nous aurons $\frac{d}{d\beta_0} \sum_{i=1}^{n} (y_i - \beta_0)^2 K_h(x_i - x) = 0 \iff \sum_{i=1}^{n} -2(y_i - \beta_0) K_h(x_i - x) = 0$

$$\iff \sum_{i=1}^{n} (y_i - \beta_0) K_h(x_i - x) = 0 \iff \sum_{i=1}^{n} y_i K_h(x_i - x) = \beta_0 \sum_{i=1}^{n} K_h(x_i - x)$$

Donc :

$$\boxed{\hat{\beta_0} = \frac{\sum_{i=1}^{n} y_i K_h(x_i - x)}{\sum_{i=1}^{n} K_h(x_i - x)}}$$

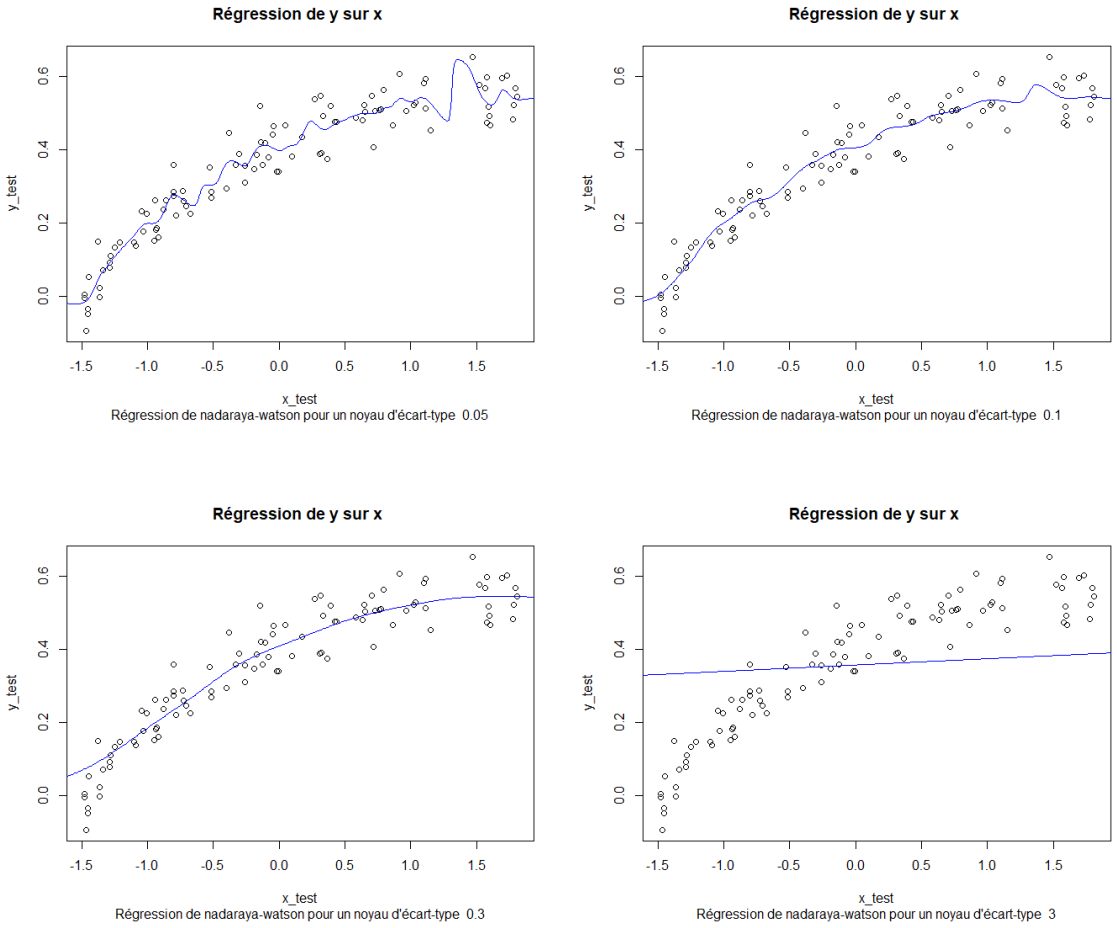
Ce programme s'interprète comme l'approximation de y en x par une constante locale β

Dans la suite, nous reprendrons les mêmes données simulées et nous analyserons les comportements de chacune avec l'estimateur de Nadaraya-Watson pour différentes valeurs du paramètre de réglage h.

Nous utiliserons les fonctions `kernelgaussian` et `nadarayawatson` pour respectivement le noyau et l'estimation des y.

Les graphiques suivants représentent l'ajustement des données pour plusieurs valeurs de h. Ainsi nous remarquons que pour la valeur **h= 0.05** nous tombons dans le `sur-ajustement des données`.

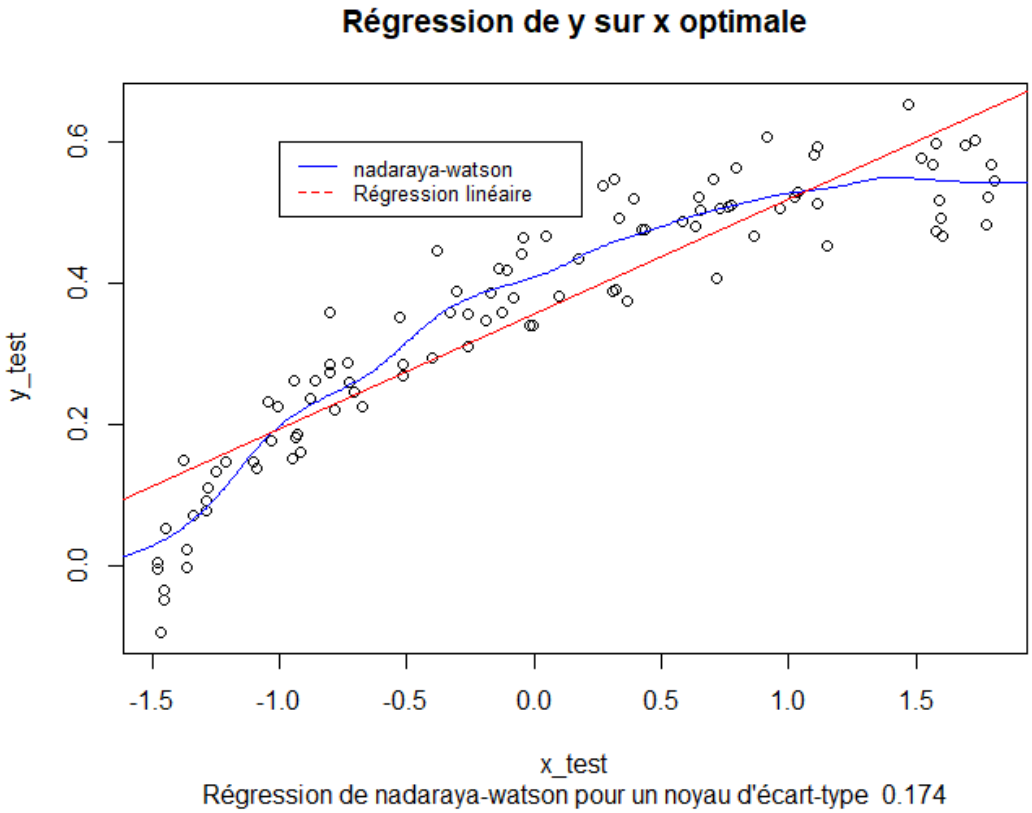
Pour **h= 3** cet ajustement devient une droite.



Afin de trouver un bon paramètre de réglage h optimal, nous utiliserons la méthode de `Leave One Out`. Ensuite, nous regarderons les erreurs moyennes pour plusieurs valeurs de h possibles.

La valeur de h minimisant l'erreur prédiction en LOO est 0.174 pour une erreur de 0.0031.

Le graphique ci-dessous montre l'ajustement des données par l'estimation de Nadaraya-Watson pour h= 0.174 en le comparant avec la"vraie" régression linéaire où l'erreur est de **0.00627**.



5.2 Régression polynomiale à noyau sur une variable

Nous généralisons l'estimateur de Nadaraya-Watson en cherchant, au delà de l'ajustement par une constante locale, l'ajustement par un polynôme local de degré fixé. Nous cherchons ainsi à résoudre le programme :

$$Q_{h,d} = min_\beta \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1(x - x_i) + ... + \beta_d(x - x_i^d)))^2 K_h(x - x_i)$$

Nous pouvons réécrire le programme sous la forme suivante :

$$Q_{h,d} = min_\beta \sum_{i=1}^{n} (y_i - \sum_{j=0}^{d} \beta_j (x - x_i)^j))^2 K_h(x - x_i)$$

Nous savons comment obtenir une estimation $\hat{\beta}$ qui minimise ce programme puisqu'il s'agit précisément du problème des moindres carrés ordinaires.

La touche finale est de pondérer les contributions de chaque couple (x_i, y_i) à l'estimation du polynôme en fonction de la proximité de x_i à x.

Nous pouvons y parvenir précisément par le noyau

$$\hat{\beta}_h = argmin_\beta \sum_{i=1}^{n} (y_i - \sum_{j=0}^{d} \beta_j (x - x_i^j))^2 K_h(x - x_i)$$

Résoudre le programme est facile une fois que la notation appropriée est introduite.

À cette fin, nous noterons que :

$$X = \begin{bmatrix} 1 & x - x_1 & ... & ... & ... & (x - x_1)^d \\ ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... \\ 1 & x - x_n & ... & ... & ... & (x - x_n)^d \end{bmatrix}$$

et $W = diag(K_h(x - x_1), ..., K_h(x - x_n))$; $Y = \begin{bmatrix} y_1 \\ y_2 \\ ... \\ ... \\ ... \\ ... \\ ... \\ y_n \end{bmatrix}$

Ensuite nous pouvons ré-exprimer $\hat{\beta}_h$ définit ci-dessus dans le programme des moindres carrés ordinaires dont la solution exacte est :

$$\hat{\beta}_h = argmin_\beta (Y - X\beta)^{'} W (Y - X\beta) = (X^{'} W X)^{-1} X^{'} W Y$$

L'estimation de $E(y|x)$ est donc

$$\hat{y_h}(x) = \hat{\beta}_{h,0} = e_1^{'} (X^{'} W X)^{-1} X^{'} W Y = \sum_{i=1}^{n} w_i^d(x) Y_i$$

où $w_i^d(x) = e_1^{'} (X^{'} W X)^{-1} X^{'} W e_i$

Avec e_i est le i-ème vecteur canonique.

Tout comme le Nadaraya-Watson était, l'estimateur polynomiale local qui est une combinaison linéaire pondérée des réponses.

Deux cas méritent une attention particulière pour l'expression du prédicteur $\hat{y_h}(x)$.

Lorsque **d=0**, le prédicteur est l'estimateur constant local ou l'estimateur de Nadaraya-Watson. Dans cette situation, l'estimateur a des poids explicites, comme nous l'avions vu auparavant :

$$w_i^0(x) = \frac{K_h(x_i - x)}{\sum_{i=1}^n K_h(x_i - x)}$$

Lorsque **d=1**, le prédicteur est l'estimateur linéaire locale, qui a des poids égaux à :

$$w_i^1(x) = \frac{1}{n} \frac{\hat{s_2}(x;h) - \hat{s_1}(x;h)(x - x_i)}{\hat{s_2}(x;h)\hat{s_0}(x;h) - (\hat{s_1}(x;h))^2} K_h(x - x_i)$$

où $\hat{s_j}(x;h) = \frac{1}{n}\sum_{i=1}^n (x - x_i)^j K_h(x - x_i)$

En effet pour $d = 1$:

$$\hat{\beta}_h = \begin{bmatrix} \sum_{i=1}^n K_h(x_i - x) & \sum_{i=1}^n (x_i - x) K_h(x_i - x) \\ \sum_{i=1}^n (x_i - x) K_h(x_i - x) & \sum_{i=1}^n (x_i - x)^2 K_h(x_i - x) \end{bmatrix}^{-1} * \begin{bmatrix} \sum_{i=1}^n y_i K_h(x_i - x) \\ \sum_{i=1}^n y_i (x_i - x) K_h(x_i - x) \end{bmatrix}$$

Dans la suite, nous regardons la construction de l'estimateur polynomial local $\hat{y_h}$, donné par `regpolynoy`, pour plusieurs valeurs du paramètre de réglage h et de degré du polynôme.

Nous supposons que le noyau est fixé.

Par la suite, nous nous intéresserons seulement au choix de la fenêtre h.

Ensuite, nous allons effectuer une validation croisée sur les données simulées, afin de déterminer le paramètre h qui minimise l'erreur quadratique moyenne.

Nous fixons le degré du polynôme p=2 pour les deux premiers graphiques :

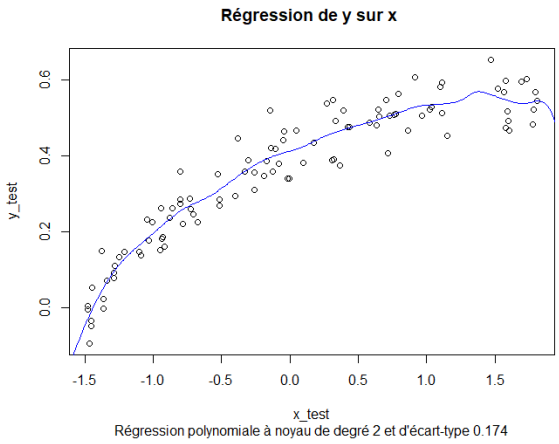


FIGURE 1 – pour **h=0.174**

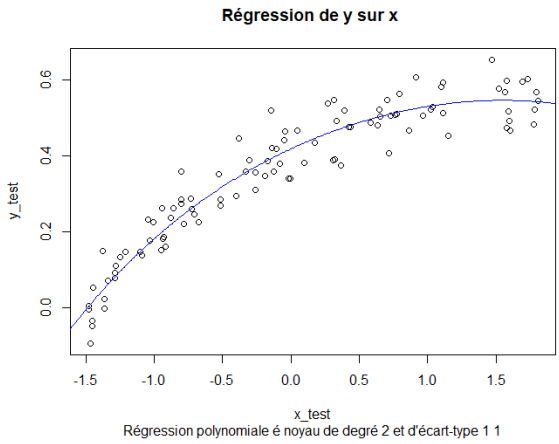
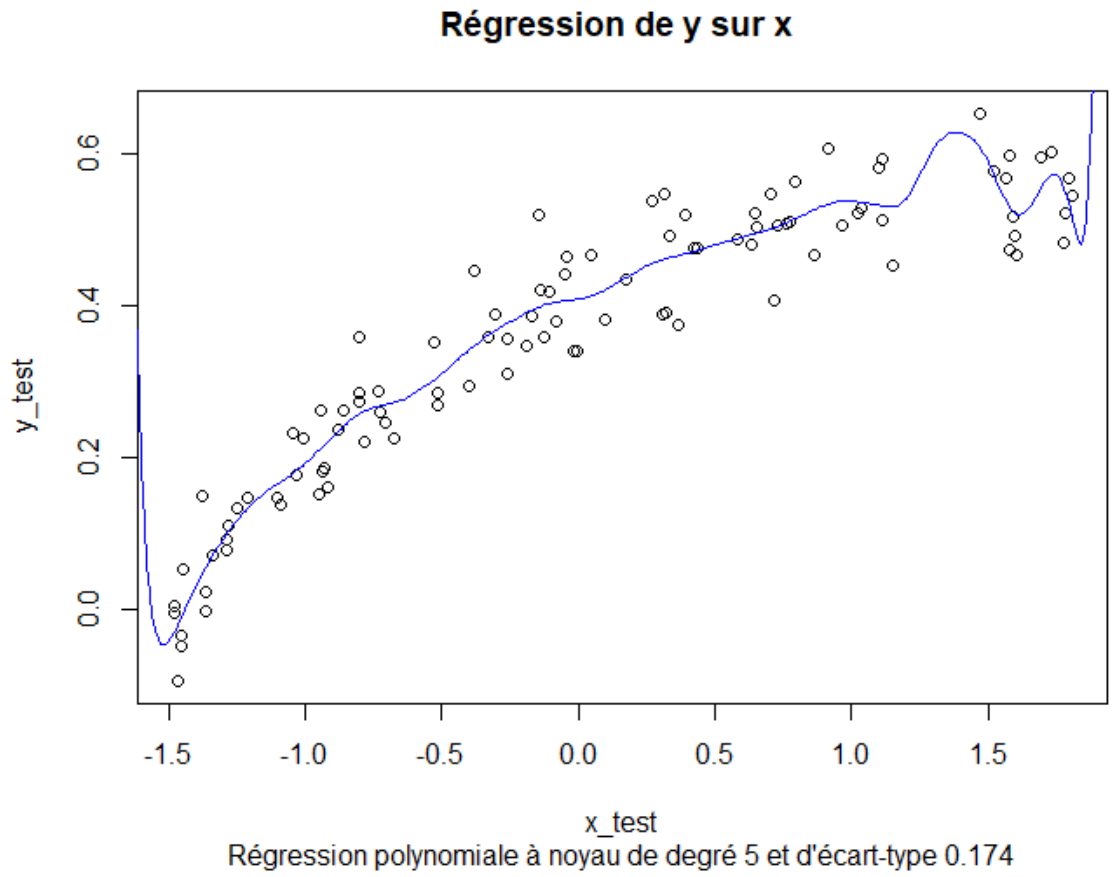


FIGURE 2 – pour **h=1**

Pour **p=5** et **h=0.174**, nous obtenons le graphique suivant :

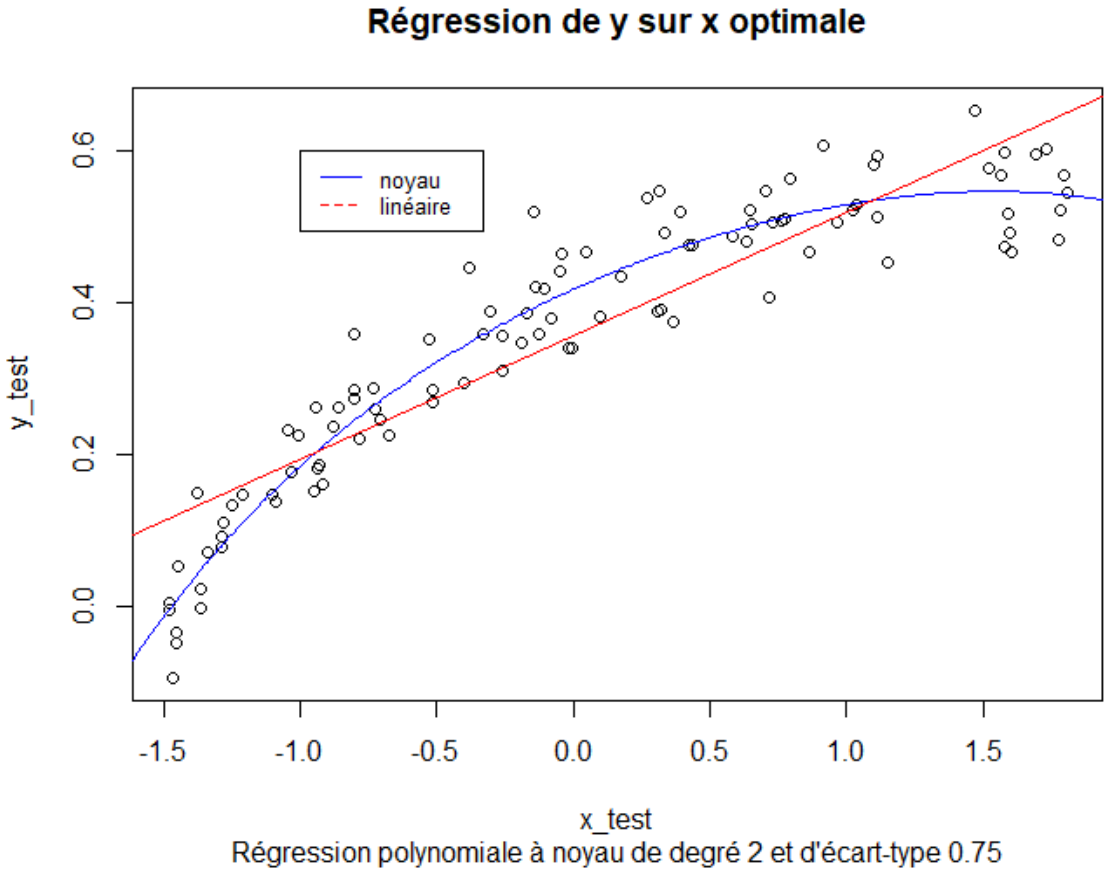


Les valeurs aux extrémités de l'axe des abscisses partent vers le haut et la ligne bleu a quelques petites bosses. On reconnaît une approximation locale par un polynôme dans cet exemple. Il conviendrait donc de prendre un plus petit degré de polynôme pour lisser la courbe car sinon ce polynôme a tendance à prendre de grandes valeurs trop vite.

Un choix "optimal" de h est obtenu par minimisation de l'erreur moyenne.

Pour le degré du polynôme qui vaut 2, le paramètre h ayant la plus petite erreur moyenne de **0.0027** est **h=0.75** contre une erreur de **0.00627** pour une régression linéaire.

Le graphique suivant représente la régression polynomiale à noyau de degré 2 pour $h = 0.75$ avec une erreur de validation croisée égale à 0.0027 :



5.3 Extension à plusieurs variables

Nous considérons l'extension suivante du programme $Q_{h,d}$ à plusieurs variables explicatives x^j :

$$R_{h,d} = min_\beta \sum_{i=1}^{n} (y_i - (\beta_0 + \sum_{j=1}^{p} [\beta_1^j (x^j - x_i^j) + ... + \beta_d^j (x^j - x_i^j)^d]))^2 K_h(x - x_i)$$

Il s'agit d'un modèle dit `"additif"`.

Son but est d'additionner des fonctions non-linéaires des variables explicatives.

Nous écrirons le programme sous forme de minimisation de la somme des carrés pondérées des résidus d'un modèle linéaire en fonction des variables explicatives.

Nous pouvons réécrire le programme sous la forme suivante :

$$R_{h,d} = min_\beta \sum_{i=1}^{n} (y_i - (\beta_0 + \sum_{j=1}^{p} \sum_{l=1}^{d} \beta_l^j (x^j - x_i^j)^l))^2 K_h(x - x_i)$$

Ce programme s'agit bien du problème des moindres carrés ordinaires.

Notons :

$$X = \begin{bmatrix} 1 & x^1 - x_1^1 & ... & ... & ... & (x^p - x_1^p)^d \\ ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... \\ 1 & x^1 - x_n^1 & ... & ... & ... & (x^p - x_n^p)^d \end{bmatrix}$$

et $W = diag(K_h(x - x_1), ..., K_h(x - x_n))$; $Y = \begin{bmatrix} y_1 \\ y_2 \\ ... \\ ... \\ ... \\ ... \\ ... \\ y_n \end{bmatrix}$

Nous pouvons donc exprimer $\hat{\beta}_h$ dans un problème des moindres carrés ordinaires :

$$\hat{\beta}_h = argmin_\beta (Y - X\beta)^{'} W (Y - X\beta) = (X^{'} W X)^{-1} X^{'} W Y$$

D'où,

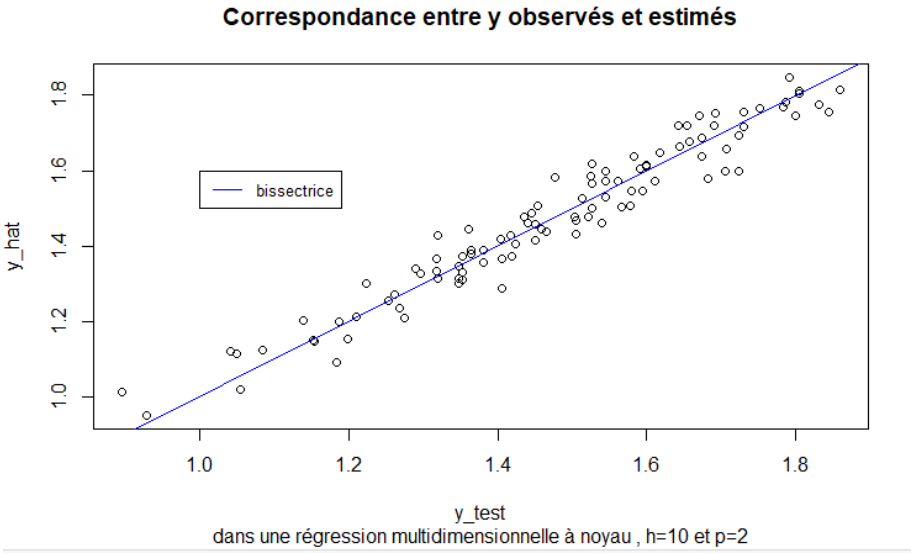
$$\hat{\beta}_h = \begin{bmatrix} \beta_0^1 & ... & ... & ... & ... & \beta_0^p \\ ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... \\ \beta_n^1 & ... & ... & ... & ... & \beta_n^p \end{bmatrix}$$

La fonction `regpolymulti` permet de récuperer cet estimateur.

Nous reprendrons dans la suite le jeu de données simulées, et regarderons la construction de l'estimateur polynomial $\hat{y_h}$ pour quelques valeurs du paramètre de réglage h.

Nous fixerons h à 10 par `Leave One Out` pour une erreur de 0.0028905.

Nous fixons le degré du polynôme à p=2 et nous obtenons le graphique suivant :



5.4 Application sur des données réelles

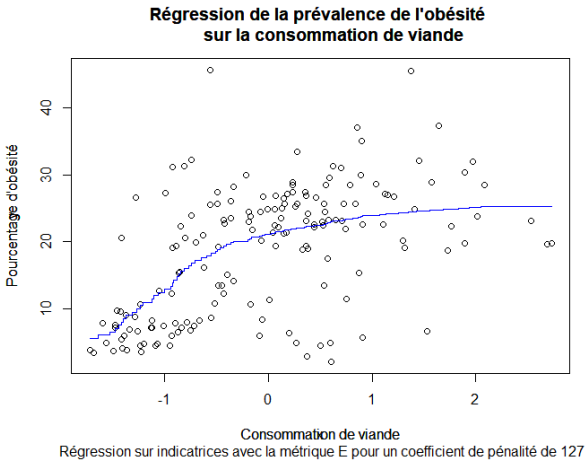
5.4.1 Analyses et résultats

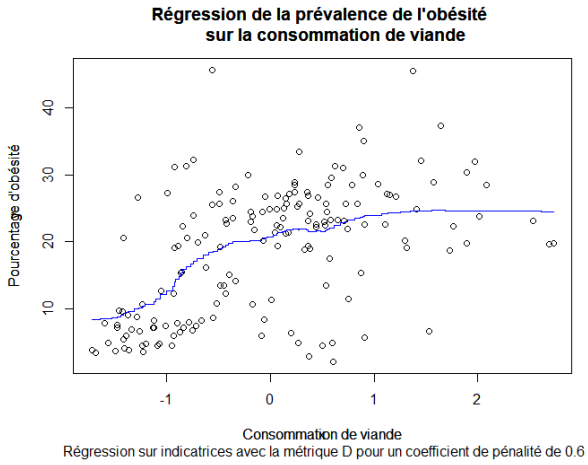
Nous appliquerons notre régression sur indicatrices ainsi que la régression polynomiale à noyau sur ces variables afin d'expliquer la prévalence de l'obésité dans la population par les deux variables explicatives :

Consommation alimentaire de viande

Apport alimentaire des céréales - à l'exclusion de la bière.

Nous ferons une régression sur la consommation de la viande seulement :

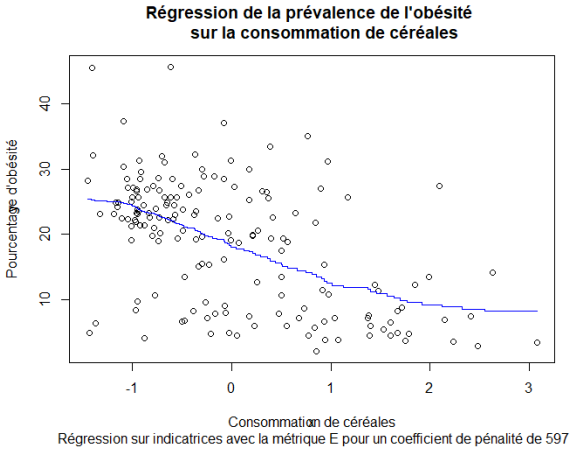


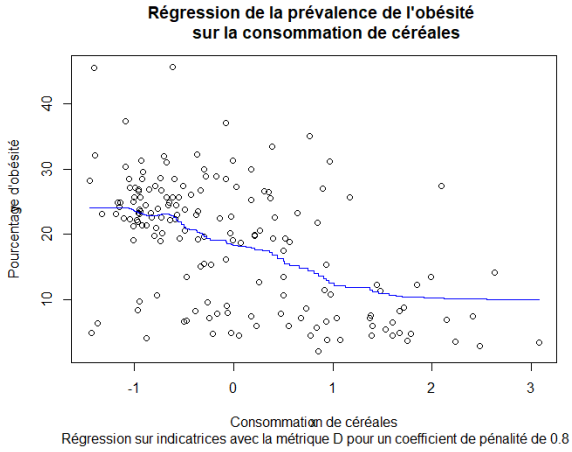


Pour la métrique E avec une valeur de λ qui est égale à 127, l'erreur de la validation croisée vaut 62.41674489.

Et pour la métrique D avec un $\lambda = 0.6$, l'erreur est égale à 63.55188252.

Dans la suite, nous ferons une régression sur l'apport alimentaire des céréales :



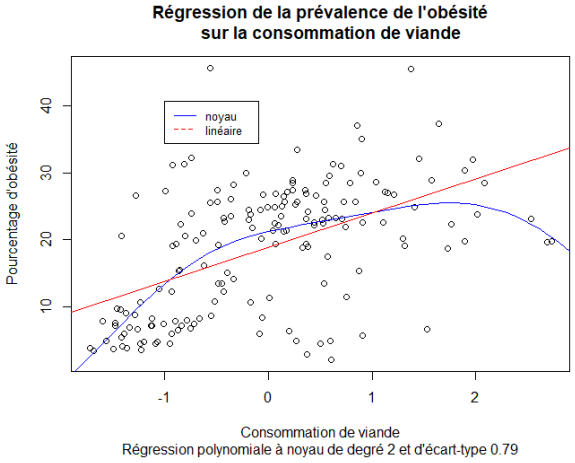


Pour la métrique E avec une valeur de λ qui est égale à 597, l'erreur de la validation croisée vaut 67.20206989. Pour la métrique D avec un $\lambda = 0.8$, l'erreur est égale à 67.887.

Nous procéderons maintenant à une régression à noyaux sur ce jeux de données réelles.

Afin d'obtenir le bon paramètre h pour cette régression, nous procédons par une `validation croisée`. Nous ferons une régression sur la consommation de la viande seulement.

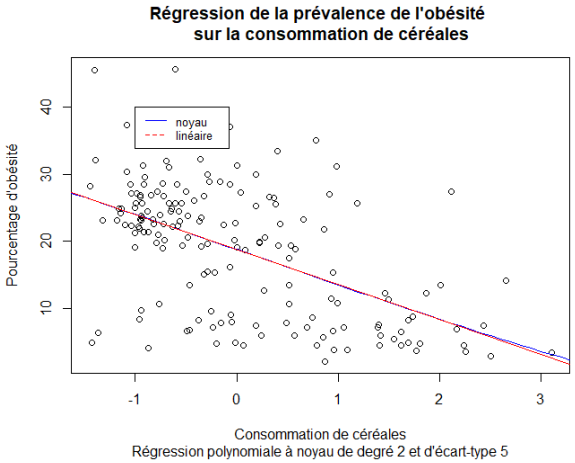
Nous obtenons **h=0.79** ayant la plus petite erreur moyenne qui est égale à 61.92282. Contre une erreur de 68.80 pour une régression linéaire.



Nous remarquons que la courbe croît avec la consommation alimentaire de la viande.

Nous regarderons maintenant l'estimation de la prévalence de l'obésité en fonction de l'apport alimentaire des céréales pour un h optimal (**h=5**) pour une erreur de 67.00844.

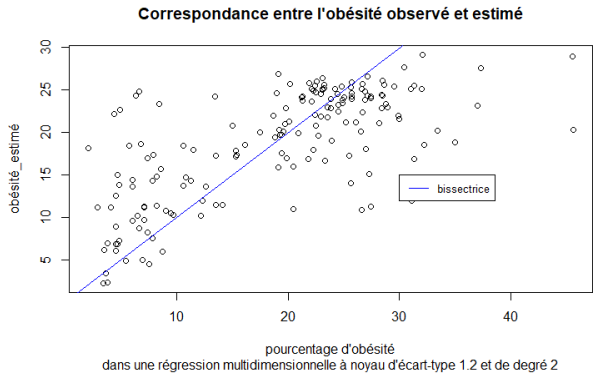
Nous en concluons que la relation est mieux modélisée par une régression linéaire pour une erreur de 66.1.



Dans le cas où nous souhaiterions expliquer l'obésité par ces deux paramètres :

Nous obtenons pour une régression linéaire l'erreur quadratique moyenne de 57.20668. Et avec notre nouveau type de régression une erreur de 52.57091.

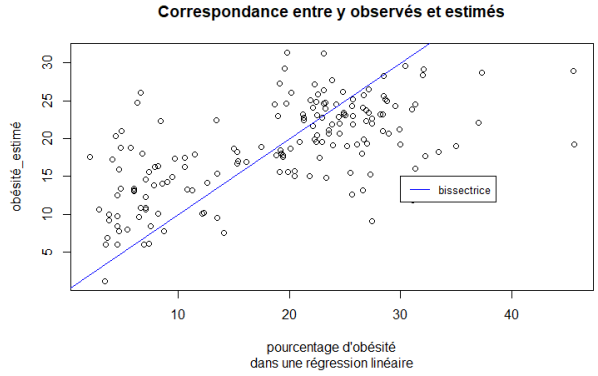
Dans la suite, nous regarderons la correspondance entre les y observés et estimés dans une régression multidimensionnelle à noyau pour le même jeu de données réelles.



Pour $h = 1.2$, l'erreur est égale à 52.57091.

On trouve une erreur de `Leave One Out` qui vaut 57.20668 dans le cas d'une régression linéaire.

Ainsi que pour la correspondance entre les y observés et estimés dans une régression linéaire, nous obtenons le graphique suivant :



6 Régression spline

6.1 L'ajustement traditionnel des splines de régression

Dans la section précédente, la fonction moyenne inconnue était supposée être localement bien approximée par un polynôme ; ce qui a conduit à une régression polynomiale locale.

Une autre approche consiste à représenter l'ajustement comme un polynôme par morceaux, avec les pièces se connectant à des points appelés **nœuds**.

Une fois les nœuds sélectionnés, un estimateur peut être calculé globalement de manière similaire à celle d'une fonction moyenne paramétriquement spécifiée.

Une fonction moyenne ajustée représentée par une courbe continue par morceaux ne fournit que rarement un ajustement. Généralement la fonction et au moins sa première dérivée sont contraints d'être continues partout, avec seulement la deuxième ou les dérivées supérieures autorisées à être discontinues aux nœuds.

Pour des raisons historiques, ces polynômes contraints par morceaux sont appelés **splines** ; ce qui conduit au nom de régression spline ou de lissage spline pour ce type de régression non paramétrique.

Considérons le type simple suivant de spline polynomiale de degré p :

$$\boxed{\beta_0 + \beta_1 x + ... + \beta_p + \sum_{k=1}^{K} \beta_{p+k}(x - K_k)_+^p}$$

Clairement, le modèle ci-dessus a des dérivées continues jusqu'au degré (p - 1), mais la p-dérivée peut être discontinue aux nœuds.

Le modèle est construit comme une **combinaison linéaire** de fonctions de base 1, x,...,x^p, $(x - K_1)_+^p, ..., (x - K_k)_+^p$.

Dans ce qui suit, nous écrirons $\psi_j(x)$, j=1,...,J. Pour un ensemble de fonctions de base (génériques) utilisées pour ajuster les splines de régression et remplacer le modèle par

$\beta_1\psi_1(x) + ... + \beta_J\psi_J$.

Pour les nœuds fixes, une spline de régression est linéaire dans les paramètres inconnus $\beta = (\beta_1, ...\beta_J)^T$ et peut être ajusté de manière paramétrique en utilisant les techniques des moindres carrés.

L'estimateur spline de régression est obtenu en résolvant

$$\boxed{\hat{\beta} = armin_\beta \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{J} \beta_j \psi_j(x_i) \right)^2}$$

Les écarts par rapport à la forme paramétrique se produisent qu'au niveau des nœuds. Par conséquent , la quantité de lissage est déterminée par le degré de base, l'emplacement et le nombre de nœuds.

En pratique, le degré est fixe (avec p = 1, 2 ou 3 comme choix courants) et les emplacements des nœuds sont généralement choisis pour être également espacés sur la plage des données ou placés à des quantiles de données régulièrement espacés.

Par conséquent, le nombre de nœuds K est le seul paramètre de lissage de l'estimateur de régression spline.

L'ajustement traditionnel des splines de régression est généralement effectué à l'aide d'un petit nombre de nœuds.

Par construction, les splines de lissage utilisent un grand nombre de nœuds (typiquement N nœuds), mais la régularité de la fonction est contrôlée par un terme de pénalité et le paramètre de lissage λ.

6.2 Splines pénalisées

L'estimateur spline pénalisé représente un compromis entre ces deux approches.

Il utilise un nombre modéré de nœuds et met une pénalité sur les coefficients des fonctions de base.

Plus précisément, un type simple d'estimateur spline pénalisé pour m(·) est obtenu en résolvant

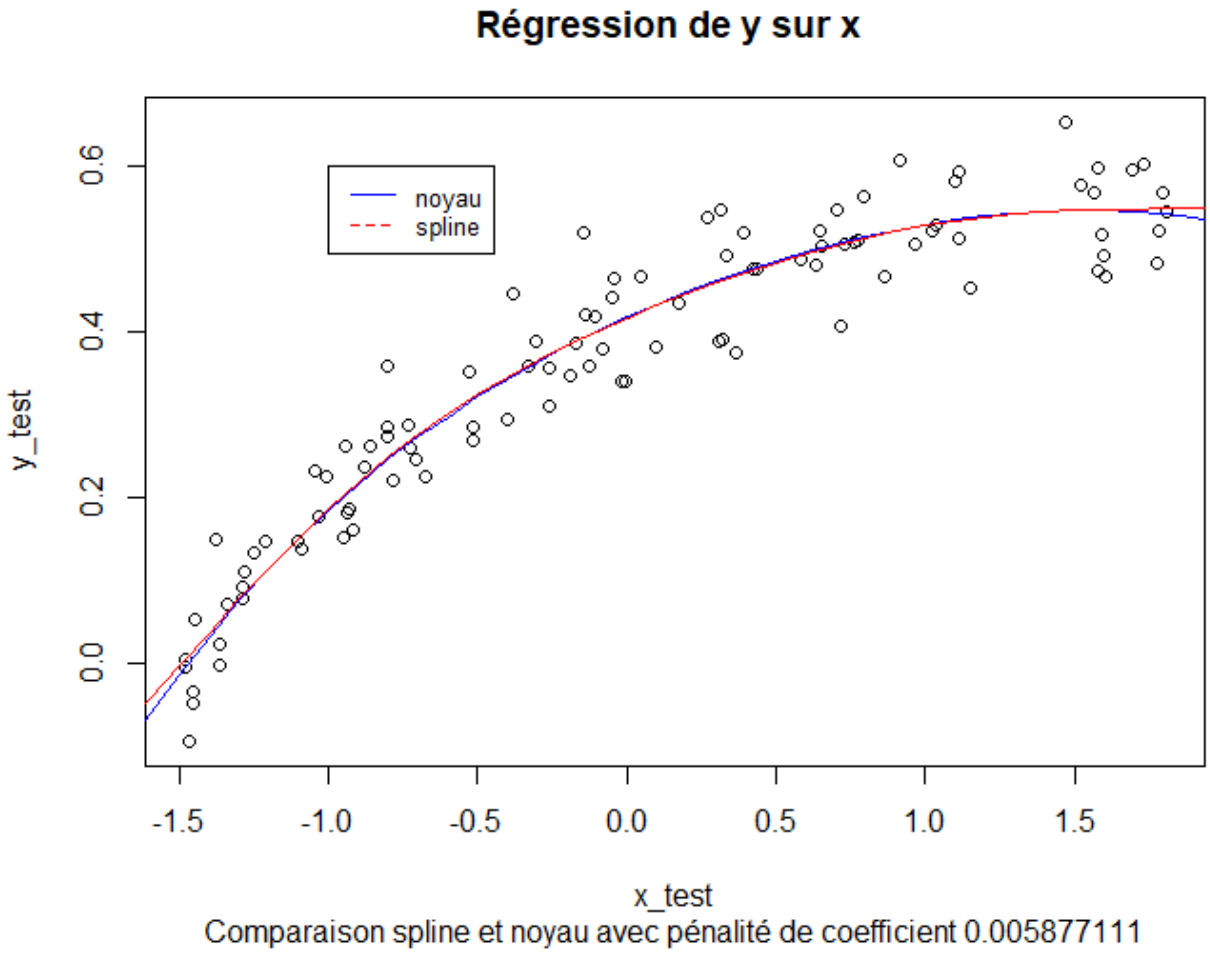
$$\hat{\beta} = argmin_{\beta} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{J} \beta_j \psi_j(x_i) \right)^2 + \lambda \sum_{j=1}^{J} \beta_j^2.$$

Les splines pénalisées combinent l'avantage d'une méthode d'ajustement paramétrique, comme pour les splines de régression, avec l'ajustement flexible du degré de lissage comme dans les splines de lissage.

La fonction de base et la forme exacte de la pénalisation des coefficients peuvent être modifiées pour s'adapter à une large gamme de paramètres de régression.

Nous reprenons les données simulées puis nous regardons l'ajustement des données par les fonctions splines.

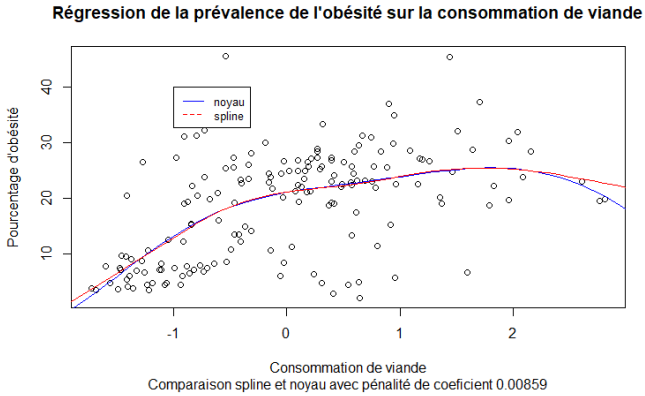
On trouve pour λ= 0.005877111, une erreur de 0.0027799500 pour le paramètre **h=0.75**.



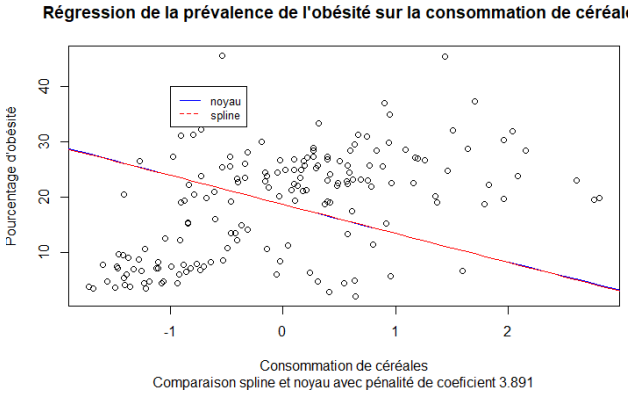
Nous regarderons par la suite l'ajustement des données par les fonctions splines en modélisant la contribution du pourcentage de l'obésité (jeu de données réelles), pour le paramètre de lissage $\lambda = 0.00859$.

Nous trouvons une erreur quadratique moyenne qui est égale à 61.9229, pour un h qui vaut 0.79.

Nous avons superposé en rouge le spline sur la régression à noyau.



Et pour le pourcentage de l'apport alimentaire des céréales, on choisit lambda=3.891786 pour une erreur quadratique égale à 67.00844, pour $h = 5$.



7 Conclusion

Nous présenterons des tableaux qui désignent les erreurs quadratiques moyennes de prédiction pour nos données simulées pour une variable ainsi que pour plusieurs variables explicatives, afin de conclure sur la méthode la plus performante :

Pour une variable explicative :

Modèles à une variable explicative	Erreur de prédiction quadratique moyenne
Régression linéaire	0.00627
Régression sur indicatrices sans pénalité	0.006165907
Régression sur indicatrices avec la métrique I	0.006120794
Régression sur indicatrices avec la métrique D	0.00310205
Régression sur indicatrices avec la métrique E	0.002922241
Régression Nadarya-Watson	0.003100559
Régression polynomiale à noyau	0.002731755
Régression spline	0.00277995

Pour plusieurs variables explicatives :

Modèles à deux variables explicatives	Erreur de prédiction quadratique moyenne
Régression linéaire	0.00645482
Régression polynomiale à noyau	0.002899354

La plus petite erreur quadratique moyenne de prédiction est celle de la régression polynomiale à noyau qui est aux alentours de 0.00278.

Les tableaux suivants affichent les erreurs quadratiques moyennes de prédiction pour nos données réelles en expliquant dans un premier temps la prévalence de l'obésité par la consommation de la viande, puis par celle des céréales et enfin par la consommation des deux variables (céréales et viandes).

Les erreurs quadratiques moyennes de prédiction pour l'estimation de la prévalence de l'obésité sur la consommation de la viande :

Modèles à une variable explicative	Erreur de prédiction quadratique moyenne
Régression linéaire	68.82583
Régression sur indicatrices sans pénalité	126.3639759
Régression sur indicatrices avec la métrique D	63.55188252
Régression sur indicatrices avec la métrique E	62.41674489
Régression polynomiale à noyau	61.9229
Régression spline	62.13156

Sur la consommation des céréales :

Modèles à une variable explicative	Erreur de prédiction quadratique moyenne
Régression linéaire	66.47949
Régression sur indicatrices sans pénalité	117.72421687
Régression sur indicatrices avec la métrique D	67.887
Régression sur indicatrices avec la métrique E	67.20206989
Régression polynomiale à noyau	67.0084
Régression spline	66.142971

Sur la consommation de céréales et de viandes :

Modèles à deux variables explicatives	Erreur de prédiction quadratique moyenne
Régression linéaire	57.20668
Régression polynomiale à noyau	52.57091

A partir des valeurs des erreurs quadratiques moyennes de prédiction présentées ci-dessus, nous pouvons en déduire que la **régression polynomiale à noyau** est la méthode la plus performante.

La raison derrière le meilleur modèle, statistiquement prédictif pour nos données simulées et réelles, est que les coefficients de régression polynomiale à noyau sont estimés avec le paramètre h de lissage.

Par ce fait, nous ajustons un polynôme, de degré 0 ou 1, localement à chaque point de requête en utilisant la méthode de sélection de validation croisée des moindres carrés et fonction noyau (gaussienne) qui aident à minimiser l'erreur quadratique moyenne du modèle individuel.

8 Annexe

```
scalebiaised<-function(x){
  ce<-x-mean(x)
  re<-sqrt(var(x)*(length(x)-1)/length(x))
  return(ce/re)
}
Predict<-function(y,Z,lambda,M,x){
  beta_hat<-solve(1/length(y)*t(Z)%*%Z+lambda*M)%*%t(Z)%*%cbind(y)*1/length(y)
  pred=rbind(x)%*%cbind(beta_hat)
  return(list("beta_hat"=beta_hat,"predi"=pred))
}
loose<-function(Z,y,lambda,M,afhist){
  erreurquadr=c()
  beta_isuc=list()
  ytilde_isuc=list()
  for (i in 1:length(y)){
    y_i=y[-i]
    Z_i=Z[-i,]
    pred=Predict(y_i,Z_i,lambda,M,Z[i,])
    beta_isuc=append(beta_isuc,list(pred$beta_hat))
    ytilde_isuc=append(ytilde_isuc,list(pred$predi))
    erreurquadr_i=(y[i]-pred$predi)^2
    erreurquadr[i]=erreurquadr_i
  }
  erreurquadrmoy=mean(erreurquadr)
  if (!missing(afhist)){
    hist(erreurquadr,main=NULL,ylab=NULL,xlab=NULL)
  }
  return(list("beta_isuc"=beta_isuc,"ytilde_isuc"=ytilde_isuc,
  "erreurquadr"=erreurquadr,"erreurquadrmoy"=erreurquadrmoy))
}
set.seed(20210127)
n <- 100
x_test1 <- scalebiaised(runif(n, min = -2, max = 2))
x_test2 <- scalebiaised(runif(n, min = -2, max = 2))
beta_0 <- -2
beta_1 <- 3
beta_2<-1.5
epsilon_test <- rnorm(n, mean = 0, sd = 1)
Z_test<-cbind(rep(1,n),x_test1,x_test2)
y_test <- beta_0 + beta_1 * x_test1 +beta_2*x_test2+epsilon_test
b<-loose(Z_test,y_test,0,diag(1,3),afhist="la")
b
```

```
title(main="Erreurs quadratiques de pr?diction ",
sub="pour une r?gression lin?aire sur donn?es simul?es",
```

```
      ylab="Fr?quence",xlab="erreurs quadratiques")
```

```
Zind<-function(x,N){
  inter=c()
  if (!(length(x)%%N==0))
  {
    print("mauvaise valeur")
    return()
  }
  Tr=length(x)/N
  xs=x
  for (i in  1:N){
    for (j in 1:Tr){
      ind=which.min(xs)
      xs[ind]=max(xs)+1
      inter[ind]=i
    }
  }
  d<-sort(x)
  tuple=list()
  tuple=append(tuple,list(rbind(d[1],(d[Tr]+d[Tr+1])/2)))
  for (i in 1:(N-2)){
    tuple=append(tuple,list(rbind((d[i*Tr]+d[i*Tr+1])/2,(d[(i+1)*Tr]+d[(i+1)*Tr+1])/2)))
  }
  tuple=append(tuple,list(rbind((d[Tr*(N-1)]+d[Tr*(N-1)+1])/2,d[length(x)])))
  return(list("matrice"=model.matrix(~as.factor(inter)-1),"codage"=cbind(x,inter),
  "intervalle"=tuple))
}
RidgeM<-function(y,Z,lambda,M){
  beta_hat<-solve((1/length(y)*t(Z)%*%Z+lambda*M))%*%t(Z)%*%cbind(y)*1/length(y)
  y_hat<-Z%*%cbind(beta_hat)
  return(list("beta_hat"=beta_hat,"y_hat"=y_hat))
}
div<-function(n){
  d=c()
  for (i in 2:(n-1)){
    if (n%%i==00){
      d=c(d,i)
    }}
  return(d)
}
ind <- function(x,a,b)  ifelse(x >= a & x <= b, 1,0)
estimbetadim<-function(X,y,N,lambda,M,x){
  Z<-c()
  for (i in 1:dim(X)[2]){
    Z_i<-Zind(cbind(X)[,i],N)$matrice
```

```
    Z<-cbind(Z,Z_i)
  }
  Z<-as.matrix(Z)
  return(list("beta_hat"=RidgeM(y,Z,lambda,M)$beta_hat,
  "y_hat"=RidgeM(y,Z,lambda,M)$y_hat))  }

affichereg<-function(x,y,lambda,M,N){
  Z<-Zind(x,N)$matrice    Ma=metriqueIetD(N)
  if(M=="I"){Ma=Ma$I}
  if(M=="D"){Ma=Ma$D}
  if(M=="E"){Ma=Ma$E}
  beta<-RidgeM(y,Z,lambda,Ma)$beta_hat
  plot(x,y,sub=paste(" R?gression sur indicatrices avec la m?trique",M,
  "pour un coefficient de p?nalit? de",as.character(lambda)))
  alpha=1/N
  ii<-Zind(x,N)$intervalle
  lia=rep(0,N*2)
  for (j in 1:N){
    lia[2*j-1]<-ii[[j]][1]
    lia[2*j]<-ii[[j]][2]
  }
  lio=c()
  for (j in 1:N){
    lio<-c(lio,rep(beta[j],2))
  }
  lines(lia,lio,col='blue')
}
metriqueIetD<-function(N,d=1){
  N1=N*d
  if (missing(d)){
    N1=N
  }
  M1=diag(1,N1)
  M2demi<-diag(1,N1)
  M2demi[row(M2demi)-col(M2demi)==-1]<--1
  M2demi=rbind(M2demi[-N1,])
  M2<-t(M2demi)%*%M2demi
  M3demi<-diag(1,N1)
  M3demi[row(M3demi)-col(M3demi)==-1]<--2
  M3demi[row(M3demi)-col(M3demi)==-2]<-1
  M3demi<-M3demi[-N*d,]
  M3demi<-M3demi[-(N*d-1),]
  M3<-t(M3demi)%*%M3demi
  return(list("I"=M1,"D"=M2,"E"=M3))
}
par(mfrow=c(1,1))
```

```
par(mar=c(5.1, 4.1, 4.1, 2.1))
library(readODS)
dataset <- data.matrix(read_ods("Datasim1.ods"))
x_test<-scalebiaised(dataset[,2])
y_test<-dataset[,1]
plot(x_test, y_test)
```

```
N=50
affichereg(x_test,y_test,0.001,"I",N)
title(main="R?gression de y sur x avec une tr?s faible p?nalit?")
```

```
affichereg(x_test,y_test,0.01,"I",N)
title(main="R?gression de y sur x avec une faible p?nalit?")
```

```
affichereg(x_test,y_test,1,"I",N)
title(main="R?gression de y sur x avec une forte p?nalit?")
```

```
affichereg(x_test,y_test,0.001,"D",N)
title(main="R?gression de y sur x avec une tr?s faible p?nalit?")
```

```
affichereg(x_test,y_test,0.01,"D",N)
title(main="R?gression de y sur x avec une faible p?nalit?")
```

```
affichereg(x_test,y_test,1,"D",N)
title(main="R?gression de y sur x avec une forte p?nalit?")
```

```
affichereg(x_test,y_test,0.001,"E",N)
title(main="R?gression de y sur x avec une tr?s faible p?nalit?")
```

```
affichereg(x_test,y_test,0.01,"E",N)
title(main="R?gression de y sur x avec une faible p?nalit?")
```

```
affichereg(x_test,y_test,1,"E",N)
title(main="R?gression de y sur x avec une forte p?nalit?")
```

```
loola<-function(x,y,lambda,choix,alt,afhist){
  alpha=1/div(length(x))
  if (!missing(alt)){
    alpha=alt
  }
  errormoy<-list()
  for(alp in alpha){      for(lam in lambda){
      if (choix=="I"){
     error<-loose(Zind(x,1/alp)$matrice,y,lam,metriqueIetD(1/alp)$I,afhist)
$erreurquadrmoy
      }
      if(choix=="D"){
        error<-loose(Zind(x,1/alp)$matrice,y,lam,metriqueIetD(1/alp)$D,afhist)
$erreurquadrmoy
```

```
      }
      if (choix=="E"){
        error<-loose(Zind(x,1/alp)$matrice,y,lam,metriqueIetD(1/alp)$E,afhist)
$erreurquadrmoy
      }
      errormoy<-append(errormoy,list(c(error,lam,alp)))}}
  return(errormoy)}
alt=1/50
lambda=seq(0,1,0.1)
loola(x_test,y_test,lambda,"I",alt)
```

```
print("lambda=(0),  erreur = 0.00616")
```

```
affichereg(x_test,y_test,0,"I",N)
title(main="R?gression de y sur x sans p?nalit?")
```

```
lambda=seq(0,0.1,0.01)
loola(x_test,y_test,lambda,"I",alt)
```

```
print("lambda=(0) erreur 0.00616")
```

```
lambda=seq(0,0.001,0.0001)
loola(x_test,y_test,lambda,"I",alt)
```

```
print("lambda=(0.0002) erreur 0.006120794")
```

```
affichereg(x_test,y_test,0.0002,"I",N)
title(main="R?gression de y sur x avec une p?nalit? optimale")
loofit<-function(x,y){
  error=c()
  if (!(is.null(dim(x)[2]))){
    for( i in 1:dim(x)[1]){
      fit=lm(y[-i]~x[-i,])
      error[i]=(fit$coefficients[1]+rbind(fit$coefficients[-1])%*%cbind(x[i,])-y[i])^2
    }
    erreurquadrmoy=mean(error)
    return(erreurquadrmoy)}
  for(i in 1:length(x)){
    fit=lm(y[-i]~x[-i])
    error[i]=(fit$coefficients[1]+fit$coefficients[2]*x[i]-y[i])^2
  }
  erreurquadrmoy=mean(error)
  return(erreurquadrmoy)
}
fit=lm(y_test~x_test)
loofit(x_test,y_test)
```

```
abline(fit$coefficients[1],fit$coefficients[2],col="red")
legend(-1,0.6,legend=c("indicatrices","lin?aire"),col=c("blue","red"),lty=1:2,cex=0.8)
```

```
loola(x_test,y_test,0.0002,"I",alt,T)
```

```
title(main="Erreurs quadratiques de pr?diction",
sub="Pour une r?gression sur indicatrices avec la m?trique I et de coeff de p?nalit? 0.0002"
,ylab="Fr?quence",xlab="erreurs quadratiques")
```

```
lambda=seq(0,5,1)
loola(x_test,y_test,lambda,"D",alt)
```

```
print("lambda=1, erreur =0.004483")
```

```
lambda=seq(0.08,0.12,0.01)
loola(x_test,y_test,lambda,"D",alt)
```

```
print("lambda=(0.1), erreur =0.003102166")
```

```
lambda=seq(0.095,0.105,0.001)
loola(x_test,y_test,lambda,"D",alt)
```

```
print("lambda=(0.097), erreur =0.00310205")
```

```
affichereg(x_test,y_test,0.97,"D",N)
title(main="R?gression de y sur x avec une p?nalit? optimale")
abline(fit$coefficients[1],fit$coefficients[2],col="red")
legend(-1,0.6,legend=c("indicatrices","lin?aire"),col=c("blue","red"),lty=1:2,cex=0.8)
```

```
loola(x_test,y_test,0.97,"D",alt,T)
```

```
title(main="Erreurs quadratiques de pr?diction",sub=
"Pour une r?gression sur indicatrices avec la m?trique D et de coefficient de p?nalit? 0.97"
,ylab="Fr?quence",xlab="erreurs quadratiques")
```

```
lambda=seq(0,100,10)
loola(x_test,y_test,lambda,"E",alt)
```

```
print("lambda=(10),aplha=1/50 erreur =0.002922605")
```

```
affichereg(x_test,y_test,10,"E",N)
```

```
lambda=seq(9,12,0.1)
lambda=11.7
loola(x_test,y_test,lambda,"E",alt)
```

```
print("lambda=(11.7),aplha=1/50 erreur =0.002922241")
```

```
affichereg(x_test,y_test,11.7,"E",N)
title(main="R?gression de y sur x avec une p?nalit? optimale")
```

```
abline(fit$coefficients[1],fit$coefficients[2],col="red")
legend(-1,0.6,legend=c("indicatrices","lin?aire"),col=c("blue","red"),lty=1:2,cex=0.8)
```

```
loola(x_test,y_test,11.7,"E",alt,T)
```

```
title(main="Erreurs quadratiques de pr?diction",
sub="Pour une r?gression sur indicatrices avec la m?trique E et de coefficient de p?nalit? 11
,ylab="Fr?quence",xlab="erreurs quadratiques")
```

```
library(readODS)
dataset <- data.matrix(read_ods("Datasim1.ods",sheet=2))
print(dataset)
```

```
x_test1<-scalebiaised(dataset[,2])
x_test2<-scalebiaised(dataset[,3])
y_test<-dataset[,1]
N=25
M=metriqueIetD(N,2)
beta<-estimbetadim(cbind(x_test1,x_test2),y_test,N,1,M$D)$beta_hat
alpha=1/N
ii<-Zind(x_test1,N)$intervalle
lia=rep(0,N*2)
for (j in 1:N){
  lia[2*j-1]<-ii[[j]][1]
  lia[2*j]<-ii[[j]][2]
}
lio=c()
for (j in 1:N){
  lio<-c(lio,rep(beta[j],2))
}
plot(lia,lio,col='blue',type='l',xlab="x1",ylab="beta1",
main=
"Coefficients des indicatrices sur les tranches de la premi?re variable",sub="Pour la m?triqu
```

```
ii<-Zind(x_test2,N)$intervalle
lia=rep(0,N*2)
for (j in 1:N){
  lia[2*j-1]<-ii[[j]][1]
  lia[2*j]<-ii[[j]][2]
}
lio=c()
for (j in 1:N){
  lio<-c(lio,rep(beta[j+N],2))
}
plot(lia,lio,col='blue',type='l',xlab="x2",ylab="beta2",main="Coefficients des indicatrices s
```

```
plot(y_test,estimbetadim(cbind(x_test1,x_test2),y_test,N,1,M$D)$y_hat,ylab="y_hat",main="Corr
     et estim?s pour la m?trique D et p?nalit? de coefficient 1")
```

```
abline(0,1,col="blue")
legend(1,1.6,legend="bissectrice",col="blue",lty=1:2,cex=0.8)
```

```
plot(y_test,estimbetadim(cbind(x_test1,x_test2),y_test,N,1,M$I)$y_hat,ylab="y_hat",main="Corr
     et estim?s pour la m?trique I et p?nalit? de coefficient 1")
abline(0,1,col="blue")
legend(1,1.6,legend="bissectrice",col="blue",lty=1:2,cex=0.8)
```

```
plot(y_test,estimbetadim(cbind(x_test1,x_test2),y_test,N,1,M$E)$y_hat,ylab="y_hat",main="Corr
     et estim?s pour la m?trique E et p?nalit? de coefficient 1")
abline(0,1,col="blue")
legend(1,1.6,legend="bissectrice",col="blue",lty=1:2,cex=0.8)
```

```
loolm<-function(x,y,lambda,alp){
  li<-list()
  li<-append(li,list("I"=loola(x,y,lambda,"I",alp)))
  li<-append(li,list("D"=loola(x,y,lambda,"D",alp)))
  li<-append(li,list("E"=loola(x,y,lambda,"E",alp)))

  return(li)
}
dataset <- data.matrix(read_ods("Datasim1.ods"))
x_test<-scalebiaised(dataset[,2])
y_test<-dataset[,1]

loolm(x_test,y_test,1,0.02)
```

```
dataset <- as.matrix(read.csv("obesity.csv",sep=',',header=T,row.names=1))
dim(dataset)
```

```
dataset
```

```
N=83
obesity=as.double(dataset[,24])
obesity=obesity[-c(110,53,148)]
meat=scalebiaised(as.double((dataset[,9])))
meat=meat[-c(110,53,148)]
meat=meat[-c(1)]
obesity=obesity[-c(1)]
affichereg(meat,obesity,127,"E",N)
title(main="R?gression de la pr?valence de l'ob?sit? sur la consommation de viande",xlab="Con
```

```
h=seq(100,200,1)
h=127
loola(meat,obesity,h,"E",1/83)
```

```
print("erreur=62.41674489 lambda= (127)")
```

```
loola(meat,obesity,127,"E",1/83,T)

title(main="Erreurs quadratiques de pr?diction",sub="Pour une r?gression sur indicatrices ave

affichereg(meat,obesity,0.6,"D",N)
title(main="R?gression de la pr?valence de l'ob?sit? sur la consommation de viande",xlab="Con

h=seq(0,1,0.1)
loola(meat,obesity,h,"D",1/83)

print('erreur=63.55188252 lambda=(0.6)')

loola(meat,obesity,0.6,"D",1/83,T)

title(main="Erreurs quadratiques de pr?diction",sub="Pour une r?gression sur indicatrices ave

cereal=as.double((dataset[,5]))
cereal=scalebiaised(cereal[-c(110,53,148)])
cereal=cereal[-c(1)]
affichereg(cereal,obesity,597,"E",N)
title(main="R?gression de la pr?valence de l'ob?sit? sur la consommation de c?r?ales",xlab="C

h=seq(580,600,1)
loola(cereal,obesity,h,"E",1/83)

print("erreur=67.20206989 lambda= 597")

loola(cereal,obesity,597,"E",1/83,T)

title(main="Erreurs quadratiques de pr?diction",sub="Pour une r?gression sur indicatrices ave

affichereg(cereal,obesity,0.8,"D",N)
title(main="R?gression de la pr?valence de l'ob?sit? sur la consommation de c?r?ales",xlab="C

h=seq(0,1,0.1)
loola(cereal,obesity,h,"D",1/83)

print("erreur=67.887 lambda= 0.8")

loola(cereal,obesity,0.8,"D",1/83,T)

title(main="Erreurs quadratiques de pr?diction",sub="Pour une r?gression sur indicatrices ave

loofit(cereal,obesity)

loofit(cereal,obesity)

loofit(meat,obesity)

kernelgaussian<-function(x,h,p){
  if (missing(p)){
```

```
    return(1/(h*sqrt(2*pi))*exp(-(x**2)/(2*h**2)))
  }
  return(1/((h*sqrt(2*pi))**p)*exp(-(t(x)%*%diag(rep(1/h**2,length(x)))%*%x)/2))

}
kernelgaussian(cbind(c(1,2,0)),2,T)
```

```
kernelgaussian(1,3)
```

```
nadarayawatson<-function(z,x,y,h){
  num<-0
  den<-0
  for (i in 1:length(x)){
    num<-num + y[i]* kernelgaussian(x[i]-z,h)
    den<-den +kernelgaussian(x[i]-z,h)
  }
  return(num/den)
}
library(readODS)
dataset <- data.matrix(read_ods("Datasim1.ods"))
x_test<-scalebiaised(dataset[,2])
y_test<-dataset[,1]
titre="R?gression de nadaraya-watson pour un noyau d'?cart-type "
h=0.05
plot(x_test,y_test,sub=paste(titre,h),main="R?gression de y sur x ")
z=seq(-2,2,0.01)
y_hat<-nadarayawatson(z,x_test,y_test,h)
a<-data.frame(z,y_hat)
a<-a[order(z),]
```

```
h=0.1
plot(x_test,y_test,sub=paste(titre,h),main="R?gression de y sur x ")
z=seq(-2,2,0.01)
y_hat<-nadarayawatson(z,x_test,y_test,h)
a<-data.frame(z,y_hat)
a<-a[order(z),]
lines(z,a$y_hat,col="blue")
```

```
h=0.3
plot(x_test,y_test,sub=paste(titre,h),main="R?gression de y sur x ")
z=seq(-2,2,0.01)
y_hat<-nadarayawatson(z,x_test,y_test,h)
a<-data.frame(z,y_hat)
a<-a[order(z),]
lines(z,a$y_hat,col="blue")
```

```
h=3
plot(x_test,y_test,sub=paste(titre,h),main="R?gression de y sur x ")
z=seq(-2,2,0.01)
y_hat<-nadarayawatson(z,x_test,y_test,h)
a<-data.frame(z,y_hat)
a<-a[order(z),]
lines(z,a$y_hat,col="blue")
```

```
loonw<-function(x,y,h,afhist){
  erreurquadr=c()
  pred_suc=list()
  ytilde_isuc=list()
  for (i in 1:length(y)){
    y_i=y[-i]
    x_i=x[-i]
    pred=nadarayawatson(x[i],x_i,y_i,h)
    pred_suc=append(pred_suc,list(pred))
    erreurquadr_i=(y[i]-pred)^2
    erreurquadr[i]=erreurquadr_i
  }
  erreurquadrmoy=mean(erreurquadr)
  if (!missing(afhist)){
    hist(erreurquadr,main=NULL,ylab=NULL,xlab=NULL)
  }
  return(list("pred_suc"=pred_suc,"erreurquadr"=erreurquadr,"erreurquadrmoy"=erreurquadrmoy))
}

loonw(x_test,y_test,0.1)
```

```
loonwch<-function(x,y,hach,afhist){
  errormoy<-list()
  for(h in hach){
    error<-loonw(x,y,h,afhist)$erreurquadrmoy
    errormoy<-append(errormoy,list(c(error,h)))}

  return(errormoy)}
h=seq(0.1,5,0.1)
loonwch(x_test,y_test,h)
```

```
print("h=(0.2) erreur= 0.003114466")
```

```
h=seq(0.1,0.3,0.01)
loonwch(x_test,y_test,h)
```

```
print("h=(0.17) erreur= 0.003100911")
```

```
h=seq(0.1,5,0.1)
loonwch(x_test,y_test,h)
```

```
print("h=0.174 erreur= 0.003100559")
```

```
h=0.174
plot(x_test,y_test,sub=paste(titre,h),main="R?gression de y sur x optimale")
z=seq(-2,2,0.01)
y_hat<-nadarayawatson(z,x_test,y_test,h)
a<-data.frame(z,y_hat)
a<-a[order(z),]
lines(z,a$y_hat,col="blue")
fit=lm(y_test~x_test)
loofit(x_test,y_test)
```

```
abline(fit$coefficients[1],fit$coefficients[2],col="red")
legend(-1,0.6,legend=c("nadaraya-watson","r?gression lin?aire"),col=c("blue","red"),lty=1:2,c
```

```
loonwch(x_test,y_test,0.174,T)
```

```
title(main="Erreurs quadratiques de pr?diction ",sub="pour une r?gression ? noyau d'?cart-typ
      ylab="Fr?quence",xlab="erreurs quadratiques")
```

```
matecpuis<-function(X,x,p) {
  z=c()
  for (j in 0:p){
    z=cbind(z,(X-x)**j)
  }

  return(z)
}

x_test<-scalebiaised(dataset[,2])
x<-2
matecpuis(x_test,x,3)
```

```
regpolynoy<-function(z,x,y,h,p){
  y_hat=c()
  for(i in 1:length(z)){
    X=matecpuis(x,z[i],p)
    beta<-solve(t(X)%*%diag
(c(kernelgaussian(x-z[i],h)))%*%X)%*%t(X)%*%diag(kernelgaussian(x-z[i],h))%*%y
    y_hat[i]=beta[1]
  }
  return(y_hat)

}

h=0.174
p=2
```

```
titre=paste("R?gression polynomiale ? noyau de degr?",p,"et d'?cart-type",h)
plot(x_test,y_test,sub=paste(titre),main="R?gression de y sur x ")
y_hat<-regpolynoy(z,x_test,y_test,0.174,2)
lines(z,y_hat,col="blue")
```

```
h=0.174
p=5
titre=paste("R?gression polynomiale ? noyau de degr?",p,"et d'?cart-type",h)
plot(x_test,y_test,sub=paste(titre,h),main="R?gression de y sur x ")
y_hat<-regpolynoy(z,x_test,y_test,0.174,5)
lines(z,y_hat,col="blue")
```

```
h=1
p=2
titre=paste("R?gression polynomiale ? noyau de degr?",p,"et d'?cart-type",h)
plot(x_test,y_test,sub=paste(titre,h),main="R?gression de y sur x ")
y_hat<-regpolynoy(z,x_test,y_test,1,2)
lines(z,y_hat,col="blue")
```

```
loonwp<-function(x,y,h,afhist){
  erreurquadr=c()
  pred_suc=list()
  ytilde_isuc=list()
  for (i in 1:length(y)){
    y_i=y[-i]
    x_i=x[-i]
    pred=regpolynoy(x[i],x_i,y_i,h,2)
    pred_suc=append(pred_suc,list(pred))
    erreurquadr_i=(y[i]-pred)^2
    erreurquadr[i]=erreurquadr_i
  }
  erreurquadrmoy=mean(erreurquadr)
  if (!missing(afhist)){
    hist(erreurquadr,main=NULL,ylab=NULL,xlab=NULL)
  }
  return(list("pred_suc"=pred_suc,"erreurquadr"=erreurquadr,"erreurquadrmoy"=erreurquadrmoy))
}

loonwpch<-function(x,y,hach,afhist){
  errormoy<-list()
  for(h in hach){
    error<-loonwp(x,y,h,afhist)$erreurquadrmoy
    errormoy<-append(errormoy,list(c(error,h)))}

  return(errormoy)}

h=seq(0.7,1,0.01)
```

```
loonwpch(x_test,y_test,h)
```

```
print("h=0.75, erreur= 0.002731755")
```

```
h=0.75
p=2
titre=paste("R?gression polynomiale ? noyau de degr?",p,"et d'?cart-type",h)
plot(x_test,y_test,sub=paste(titre),main="R?gression de y sur x optimale")
y_hat<-regpolynoy(z,x_test,y_test,h,p)
lines(z,y_hat,col="blue")
abline(fit$coefficients[1],fit$coefficients[2],col="red")
legend(-1,0.6,legend=c("noyau","lin?aire"),col=c("blue","red"),lty=1:2,cex=0.8)
```

```
loonwpch(x_test,y_test,0.75,T)
```

```
title(main="Erreurs quadratiques de pr?diction ",sub="pour une r?gression polynomiale ? noyau
      ylab="Fr?quence",xlab="erreurs quadratiques")
```

```
dataset <- as.matrix(read.csv("obesity.csv",sep=',',header=T,row.names=1))
dim(dataset)
```

```
dataset
```

```
obesity=as.double(dataset[,24])
obesity=obesity[-c(110,53,148)]
meat=scalebiaised(as.double((dataset[,9])))
meat=meat[-c(110,53,148)]
z=seq(-2,3,0.1)

h=0.79
p=2
titre=paste("R?gression polynomiale ? noyau de degr?",p,"et d'?cart-type",h)
plot(meat,obesity,main="R?gression de la pr?valence de l'ob?sit? sur la consommation de viand
y_hat<-regpolynoy(z,meat,obesity,0.79,2)
lines(z,y_hat,col="blue")
fit=lm(obesity~meat)

loofit(meat,obesity)
```

```
abline(fit$coefficients[1],fit$coefficients[2],col="red")
legend(-1,40.6,legend=c("noyau","linÃ©aire"),col=c("blue","red"),lty=1:2,cex=0.8)
```

```
h=seq(0.1,3,0.01)
h=0.79
loonwpch(meat,obesity,h)
```

```
print("h=0.79 erreur= 61.9229")
```

```
loonwpch(meat,obesity,0.79,T)
```

```
title(main="Erreurs quadratiques de pr?diction ",sub="pour une r?gression polynomiale ? noyau
      ylab="Fr?quence",xlab="erreurs quadratiques")
```

```
cereal=scalebiaised(as.double((dataset[,5])))
dataset
```

Fonctionnement de la fonction Predict :*Annexe*

- prend en argument la variable expliquée, la matrice des variables explicatives, une pénalité, une métrique et un nouvel individu avec toutes ses caractéristiques .
- résout le programme des moindres carrés pénalisé de la régression linéaire
- renvoie l'estimation de β et la valeur prédite pour le nouvel individu

Fonctionnement de la fonction Loose :

- prends en argument la variable expliquée,la matrice des variables explicatives, une pénalité, une métrique et une option pour afficher l'histogramme .
- Pour chaque individu, on l'enlève puis on prédit sa valeur y avec la fonction Predict à l'aide de l'échantillon privé de l'individu
- renvoie les estimation de β et y pour les individus manquant de chaque échantillon ;
- calcule l'écart quadratique entre la prédiction et la valeur exacte pour chaque individu
- renvoie l'erreur quadratique moyenne

Fonctionnement de la fonction Zind :

- prend en argument une variable x et un entier N .
- Il faut que le nombre de tranches N divise la taille de x
- on range dans l'ordre croissant chaque x_i
- on met aux mêmes indices le numéro de la tranche auxquels les valeurs de x appartiennent
- on associe à chaque tranche un intervalle contenant tous les éléments de la tranche
- on renvoie le codage, la matrice des indicatrices vue comme un codage disjonctif complet des tranches et les intervalles

Fonctionnement de la fonction Ridge :

- prend en argument une variable expliquée y, la matrice des variables explicatives Z,une pénalité λ et une métrique M
- utilise la formule pour faire la régression pénalisée
- renvoie l'estimation de β et de y

Fonctionnement de la fonction estimbetadim :

- prend en argument une variable expliquée y, la matrice des variables explicatives X,une pénalité λ et une métrique M, un entier N
- prend en compte chaque variable explicative et la transforme en matrice d'indicatrices pour les concaténer
- renvoie l'estimation de β et de y après avoir utilisée la fonction Ridge

Fonctionnement de la fonction metriqueIetD :

- prend en argument un entier N et une dimension d
- multiplie N par d dans le cas où on a d variables explicatives
- renvoie les métrique I, D, E précédemment définies

Fonctionnement de la fonction affichereg :

- prend en argument les variables x explicative et une variable y expliquée, une pénalité λ , une métrique, un nombre de tranches N

— trouve l'estimation des β
— trace la fonction en escalier en N tranches par-dessus le nuage de point

Fonctionnement de la fonction Loola :

— prend en argument la variables x explicative et une variable y expliquée, un ensemble de pénalités , un choix de métrique et un argument optionnel pour le nombre de tranche
— fournit les erreurs moyennes pour les lambdas et alphas associés pour une régression pénalisée

Fonctionnement de la fonction loolm :

— prend en argument les variables x explicative et une variable y expliquée, un nombre de tranches $\frac{1}{\alpha}$ et une pénalité λ

— utilise la fonction loola pour chaque métrique pour récupérer les erreurs

— il faut fixer α et λ pour comparer seulement les métriques

Fonctionnement de la fonction loofit et loosplin :

— loofit fait le leave one out pour un modèle linéaire.
— loosplin fait le leave one out pour une régression spline de smooth.spline.

Fonctionnement de la fonction kernelgaussian :

— applique vectoriellement (à chaque élément du vecteur) la formule du noyau gaussien pour un h donné
— ou la valeur de la densité normale multidimensionnelle pour un vecteur

Fonctionnement de la fonction nadaraywatson :

— calcule en chaque point de z la moyenne pondérée de Nadaraya-Watson avec les données x et y et une largeur h

Fonctionnement des fonctions loonw et loonwch :

— loonw fait le leave one out avec x et y comme précédemment mais avec comme fonction nadaraya-watson pour un h fixé.
— loonwch permet de récupérer les erreurs moyennes pour plusieurs h

Fonctionnement des fonctions regpolynoy :

— regpolynoy résout le programme pour chaque élément de z dont on veut connaître la réponse et renvoie les estimations de ces réponses grâce aux données x et y pour largeur de noyau h, un degré p .

Fonctionnement des fonctions loonwp et loonwpch :

— loonwp fait le leave one out avec x et y comme précédemment mais avec comme fonction regpolynoy pour un h fixé et le degré 2 fixé.
— loonwpch permet de récupérer les erreurs moyennes pour plusieurs h

Fonctionnement des fonctions regpolymulti :

— regpolymulti résout le programme pour une matrice dont les lignes représentent un vecteur de variables explicatives dont on veut connaître la réponse et renvoie les estimations de celles-ci pour une largeur de noyau h, un degré p et les données X et y.

Fonctionnement des fonctions loonwpmulti et loonwpmultich :

— loonwpmulti fait le leave one out avec X et y comme précédemment mais avec comme fonction regpolymulti pour un h fixé et le degré 2 fixé.
— loonwpmultich permet de récupérer les erreurs moyennes pour plusieurs h

Fonctionnement des fonctions loonwpmulti et loonwpmultich :

— exactement la même chose que pour la première version sauf qu'on fixe à 2 le degré du polynôme et que c'est multidimensionnel.

Fonctionnement de la fonction loosplin :

— fait le leave one out pour la fonction smooth.spline de R