# Chapitre 4 : Méthodes avancées : GAM et boosting

Agathe Guilloux
Professeure au LaMME - Université d'Évry - Paris Saclay

# Generalized additive models (GAM)

# Les GAM : beyond linearity

On a les données $(Y_i, X_i)$ pour $i = 1, \dots, n$ où les $Y_i$ sont les labels et les $X_i$ les features. Dans les GLM, on a considéré que les $Y_i$ ont une densité de la famille exponentielle et qu'il existe une fonction $g$ connue (fixée) telle

$$g\big(\mathbb{E}(Y_i)\big) = X_i\beta^\star = \beta_0 + \sum_{j=1}^{p} X_i^j \beta_j$$

et on a estimé $\beta^\star$ par maximum de vraisemblance. Pour mémoire, les fonctions $g$ canoniques sont $g = \log(x/(1-x)$ en régression logisitique, $g = \mathrm{Id}$ en régression linéaire, etc

## Generalized additive models

Dans un modèle GAM, on considère que

$$g\big(\mathbb{E}(Y_i)\big) = \beta_0 + \sum_{j=1}^{p} f_j(X_i^j)$$

où les $f_j$ sont des fonctions "lisses" inconnues, donc à estimer.

# Régression non-paramétrique avec une covariable

On commence par le problème plus simple de l'estimation de $f_1$ quand les $X_i$ sont dans $[0,1]$ et en régression linéaire $g = \mathrm{Id}$

$$\mathbb{E}(Y_i) = \beta_0 + f_1(X_i).$$

L'idée est d'écrire la fonction $f_1$ sur une base de fonctions sur $[0,1]$ (histogrammes, ondelettes, trigonométrique, polynômes locaux, etc)

$$f_1(x) = \sum_{k \geq 1} \alpha_k h_k(x)$$

puis d'estimer les coefficients d'une approximation sur les $M$ premières fonctions de la base

$$(\hat{\alpha}_1, \ldots, \hat{\alpha}_M) = \mathrm{argmin}_{\alpha_1,\ldots,\alpha_M} \sum_{i=1}^{n} \Big(Y_i - \sum_{k=1}^{M} \alpha_k h_k(X_i)\Big)^2$$

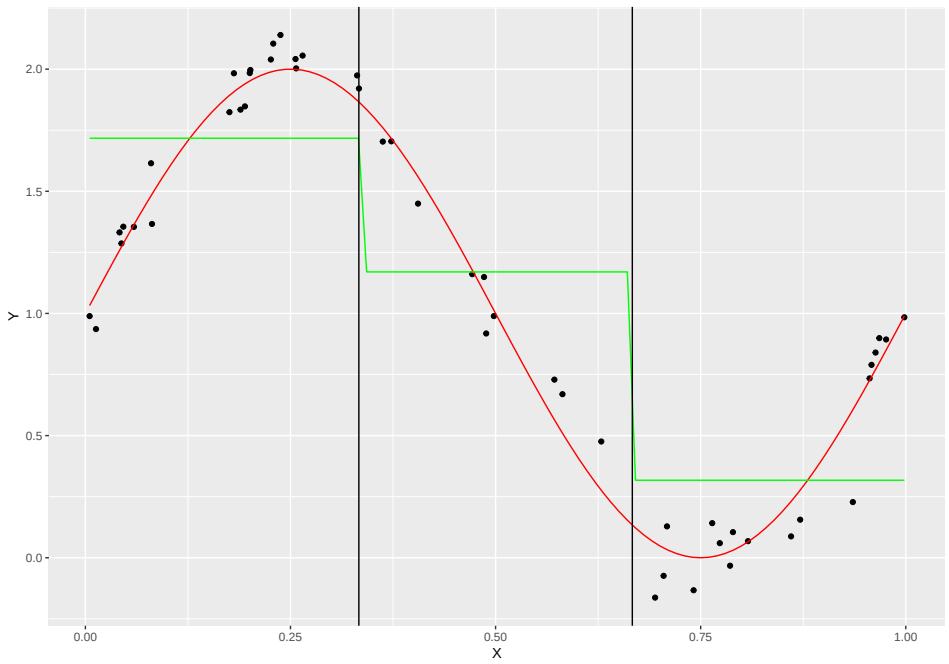
# Polynômes locaux

Nous allons considérer les polynômes locaux :

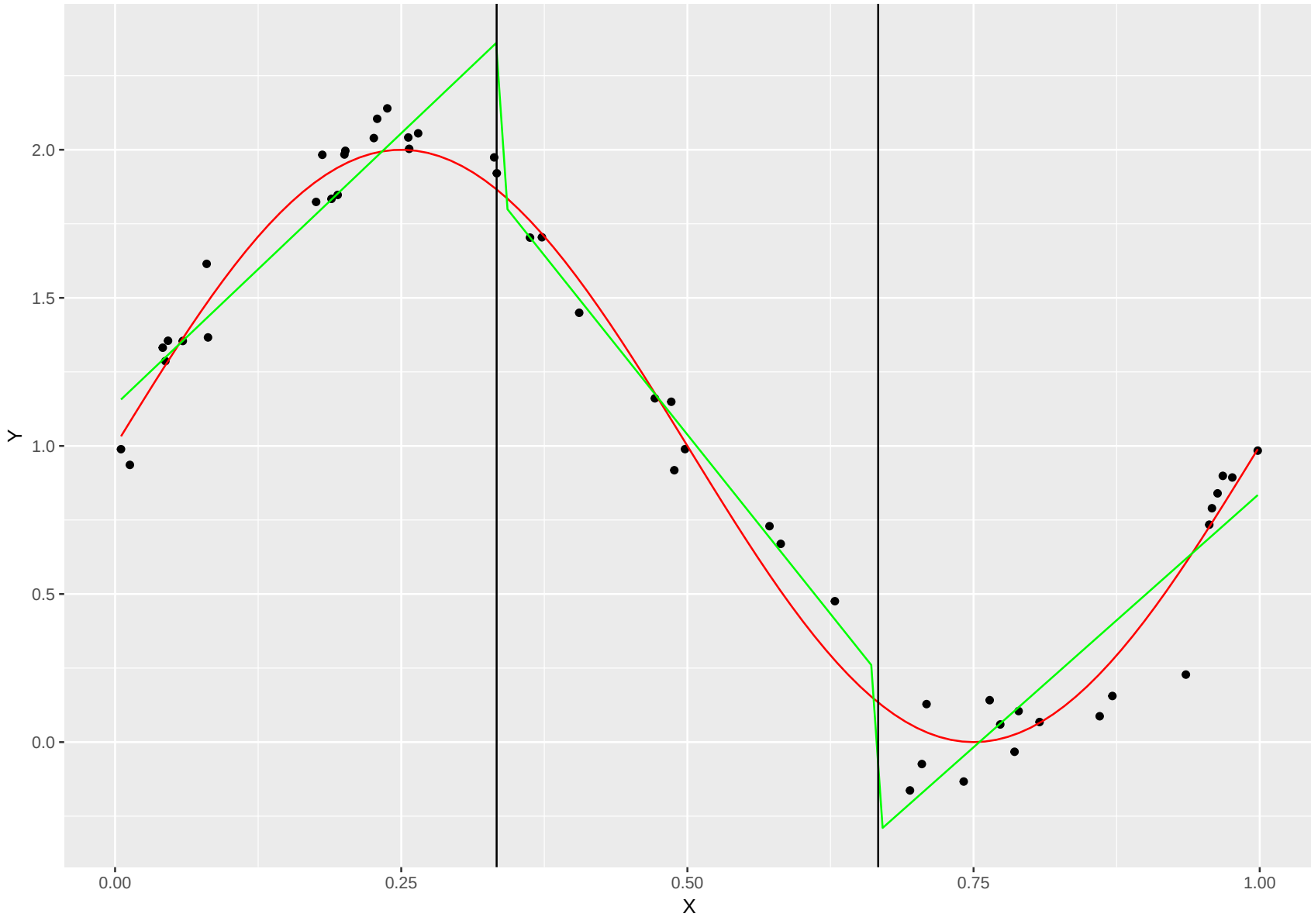
- on considère une partition de $[0, 1]$ en intervalles
- des polynômes sur chaque intervalle.

Si on divise $[0, 1]$ en 3 intervalles $[0, \xi_1[, [\xi_1, \xi_2[, [\xi_2, 1]$, on peut considérer les polynômes constants sur chaque intervalle

$$h_1(x) = \mathbf{1}(x < \xi_1),\ h_2(x) = \mathbf{1}(\xi_1 \leq x < \xi_2) \text{ et } h_3(x) = \mathbf{1}(x > \xi_2)$$



On peut également augmenter le degré des polynômes et ajouter la contrainte de continuité.



## Base de splines cubiques

Au degré 1, pour assurer la continuité, on considère la base :

$$h_1(x) = 1,\ h_2(x) = x,\ h_3(x) = (x - \xi_1)_+,\ h_4(x) = (x - \xi_2)_+.$$

Au degré 3, il faut considérer

$$h_1(x) = 1,\ h_3(x) = x^2,\ h_5(x) = (x - \xi_1)_+^3$$
$$h_2(x) = x,\ h_4(x) = x^3,\ h_6(x) = (x - \xi_2)_+^3.$$

## Spline d'ordre M

Une spline d'ordre $M$ et de noeuds $\xi_1, \ldots, \xi_K$ est une fonction polynomiale par morceaux de degré $M - 1$ et qui a des dérivées continues jusqu'à l'ordre $M - 2$.

Les splines d'ordre $M$ et de noeuds $\xi_1, \ldots, \xi_K$ sont engendrées par la base :

$$h_j(x) = x^{j-1},\ j = 1, \ldots, M$$
$$h_{M+l} = (x - \xi_l)_+^{M-1},\ l = 1 \ldots, K.$$

# Justification

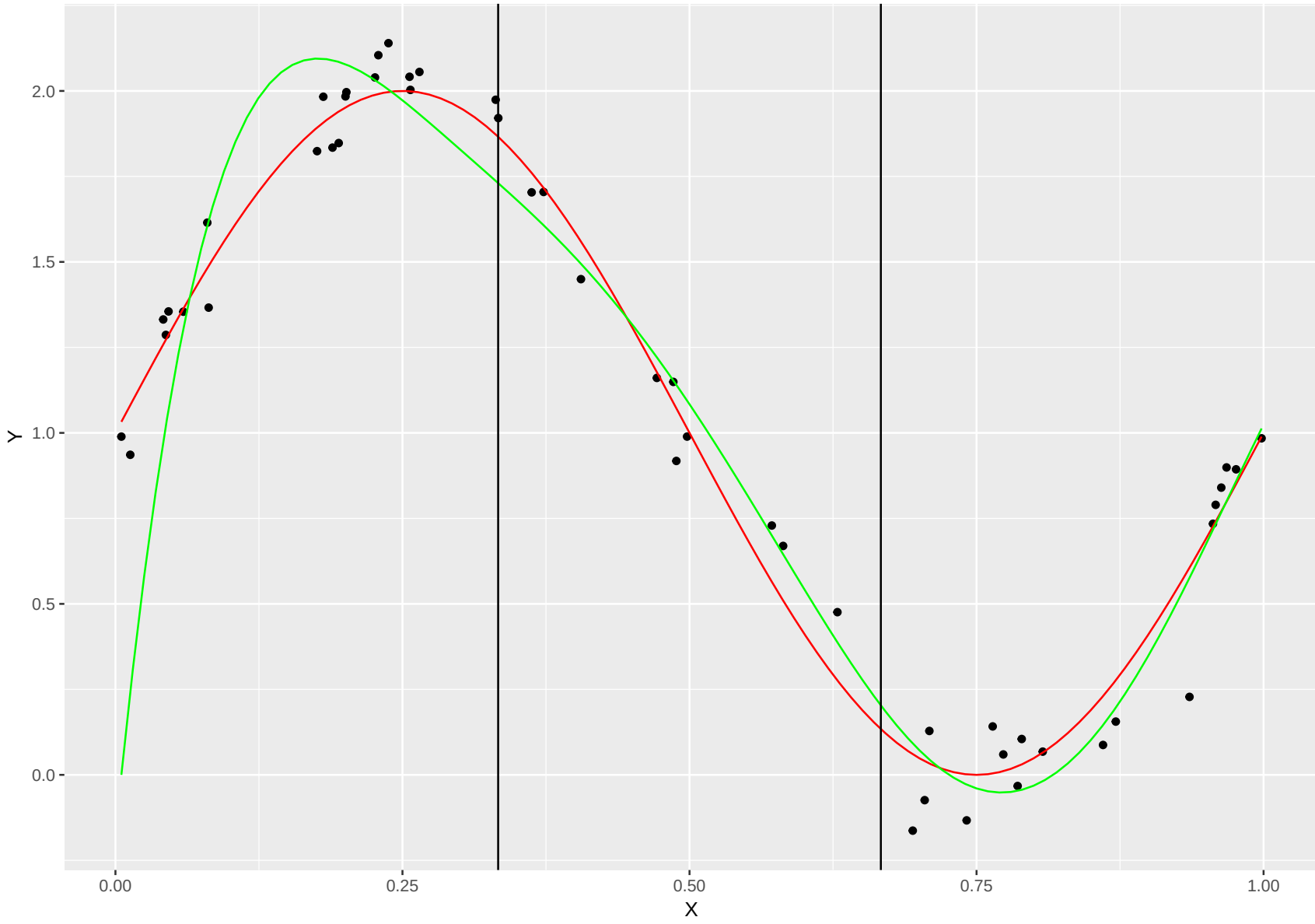
- Sur chacun des $K+1$ intervalles $[0, \xi_1[, \ldots, [\xi_K, 1[$, on définit un polynôme $P_k$ de degré $M-1$, il faut donc $(K+1)M$ coefficients.
- On ajoute les conditions aux bords des intervalles pour toutes les dérivées d'ordre $l = 0, \ldots, M-2$:

$$P_k^{(l)}(\xi_k) = P_{k-1}^{(l)}(\xi_k) \text{ pour tout } \xi_k, \ k = 1, \ldots, K$$

on obtient $K * (M-1)$ contraintes.

On est donc dans un espace vectoriel de dimension
$(K+1)M - K * (M-1) = M + K$.

Dans le cas des splines cubiques ($M = 4$) sur 2 noeuds, on a bien une dimension $12 - 6 = 6$, comme celle de la base proposée.


2.0
1.5
1.0
0.5
0.0
Y
0.00
0.25
0.50
0.75
1.00
X

# Base de splines cubiques naturels

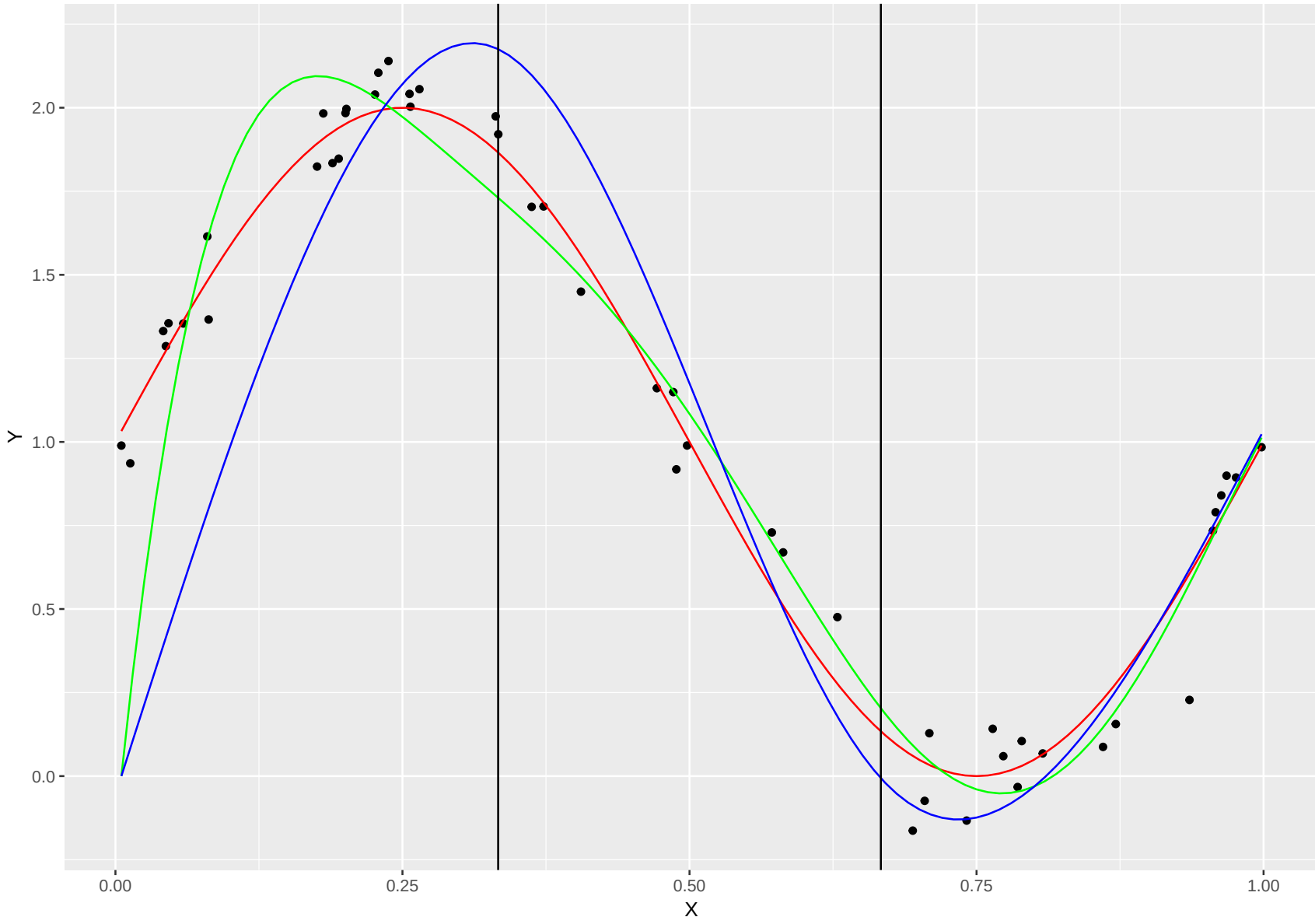
Pour améliorer les comportements aux bords, on peut forcer la fonction a avoir des dérivées d'ordre 2 et 3 nulles à gauche de $\xi_1$ et à droite de $\xi_K$. On obtient alors un espace de dimension $4(K+1) - K*3 - 4 = K$.

## Base de splines cubiques naturelles d'ordre K

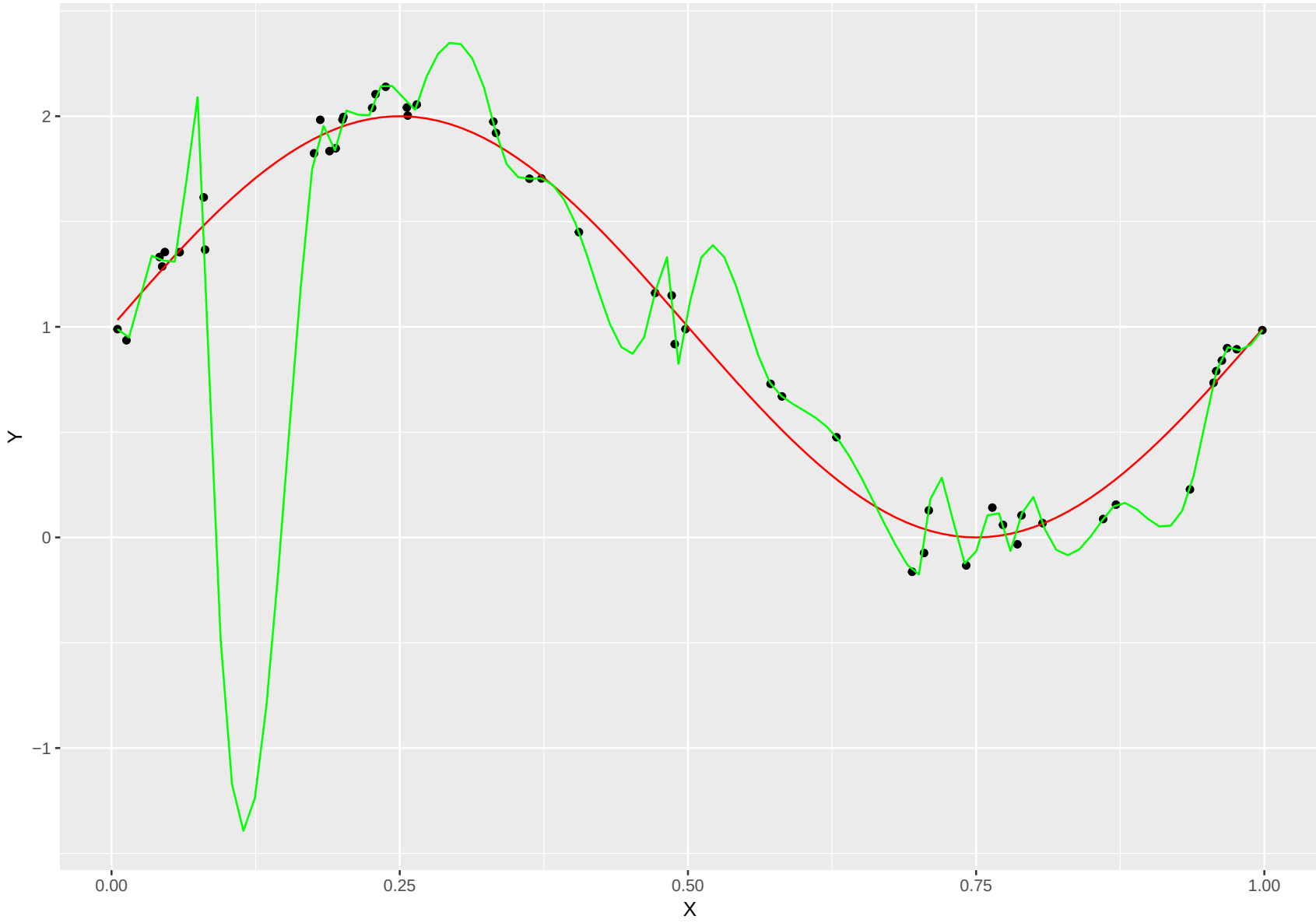
On obtient alors les splines cubiques naturelles dont une base est donnée par

$$N_1(x) = 1, N_2(x) = x, N_{k+2}(x) = d_k(x) - d_{K-1}(x) \text{ avec}$$

$$d_k(x) = \frac{(x-\xi_k)_+^3 - (x-\xi_K)_+^3}{\xi_K - \xi_k}.$$


2.0
1.5
1.0
0.5
0.0
Y
0.00
0.25
0.50
0.75
1.00
X

# Spline d'interpolation



# Propriété cruciale des splines cubiques naturelles

## Propriété

Si on considère le problème de minimisation

$$\sum_{i=1}^{n}(Y_i - f(X_i))^2 + \lambda \int (f''(x))^2 dx$$

parmi toutes les fonctions de classe $\mathcal{C}^2$ alors le minimum est atteint pour une spline cubique naturelle dont les nœuds sont au points de données $(X_1, \ldots, X_n)$.

Cette spline vit dans un espace de dimension $n$, on peut donc penser que le problème est encore sur-paramétrisé. En fait le paramètre $\lambda$ permet de rendre les coefficients devant chaque fonction de la base petits.

# Degrées de liberté effectifs et cross-validation

## Degrés de liberté effectifs

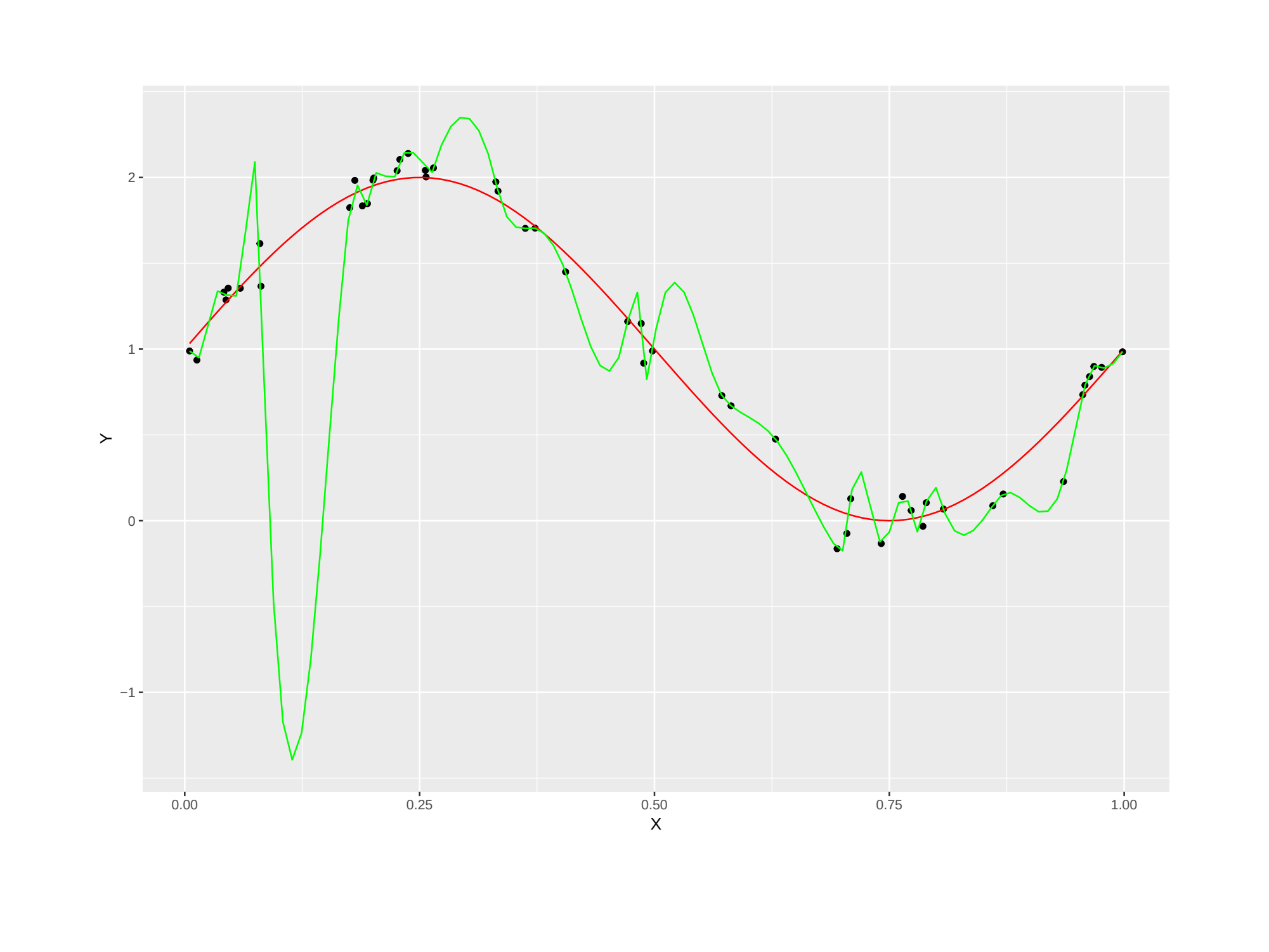
Soit $N$ la matrice de taille $n \times n$ de design associée à la base des splines cubiques naturelles de dimension $n$ alors la solution du problème précédent est donnée par

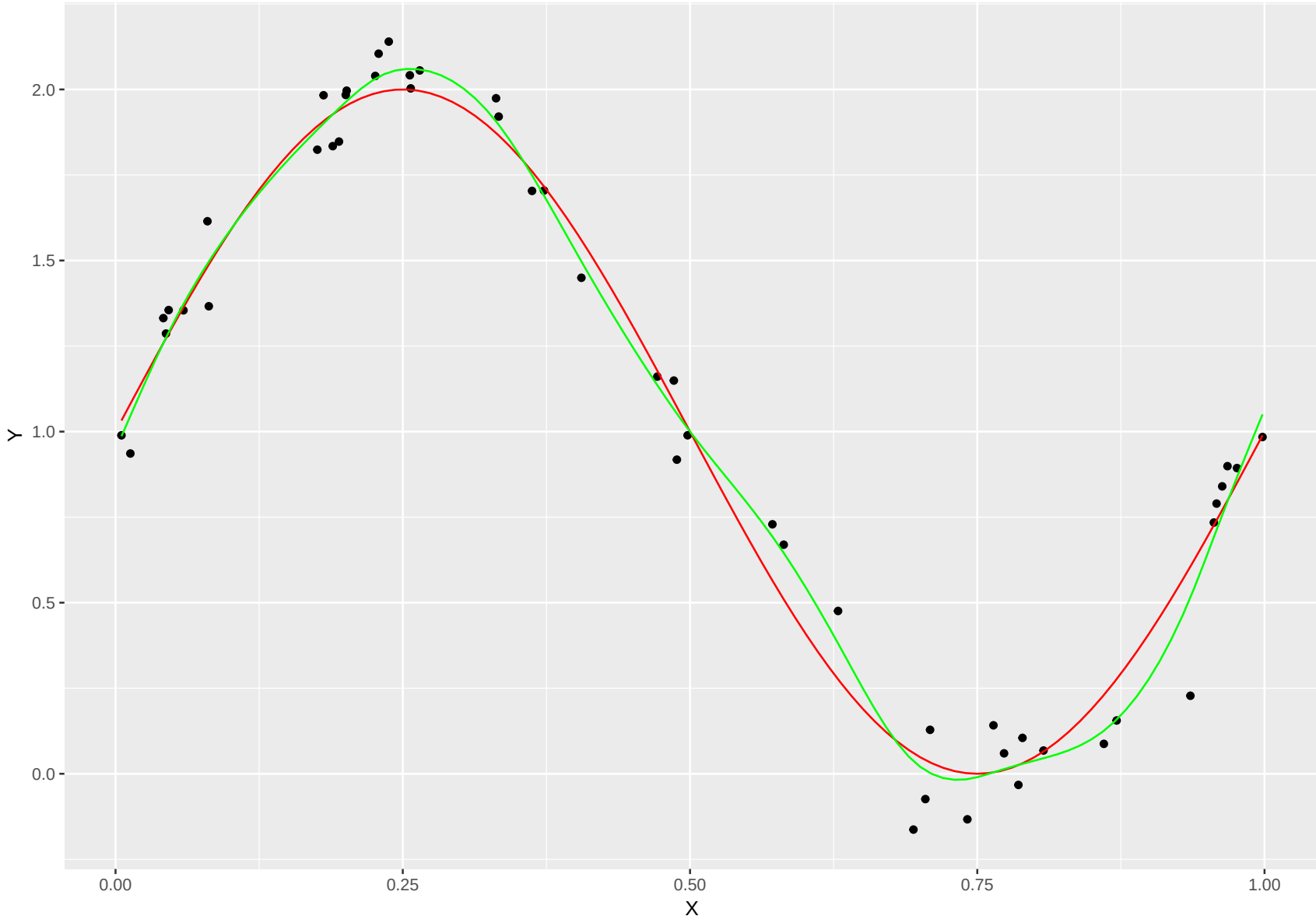
$$N(N^\top N + \lambda \Omega_n)^{-1} N^\top Y = S_\lambda Y,$$

avec $\Omega_n = \left( \int N_j^{(2)}(t) N_k^{(2)}(t) dt \right)$ Par analogie avec la régression linéaire classique, les **degrés de liberté effectifs** $\mathrm{df}_\lambda$ sont définis par

$$\mathrm{df}_\lambda = \mathrm{Trace}(S_\lambda).$$

On peut alors faire une cross-validation.


2
1
0
−1
Y
0.00
0.25
0.50
0.75
1.00
X


2.0
1.5
1.0
0.5
0.0
Y
0.00
0.25
0.50
0.75
1.00
X

# Avec plusieurs variables

On considère le problème de minimisation

$$\sum_{i=1}^{n}(Y_i - \alpha - \sum_{j=1}^{p} f_j(X_i^j))^2 + \sum_{j=1}^{p} \lambda_j \int (f_j''(x))^2 dx$$

- parmi toutes les fonctions de classe $\mathcal{C}^2$ de la forme $f(x) = \sum_{j=1}^{p} f_j(x_j)$
- alors le minimum est atteint quand chaque $f_j$ est une spline cubique naturelle dont les nœuds sont au points de données $(X_1^j, \ldots, X_n^j)$.

Attention

- pour rendre le problème identifiable, il faut enlever la fonction constante de la base des splines cubiques et considérer $\hat{\alpha} = \bar{Y}$.
- il faut faire les cross-validations pour les $p$ paramètres de régularisation $\lambda_1, \ldots, \lambda_p$
- l'algorithme d'optimisation est le "back-fitting algorithm"

# GAM pour les GLM

Pour chaque GLM de densité $d$, on estimera la fonction $f$ qui vérifie

$$g\big(\mathbb{E}(Y_i)\big) = \beta_0 + \sum_{j=1}^{p} f_j(X_i^j)$$

en considérant l'approximation $\tilde{f}_j$ de chaque $f_j$ sur une base de splines cubiques naturelles dont les nœuds sont au points de données $(X_1^j, \ldots, X_n^j)$., puis en résolvant

$$\sum_{i=1}^{n} d\big((Y_i, \alpha + \sum_{j=1}^{p} \tilde{f}_j(X_i^j))\big) + \sum_{j=1}^{p} \lambda_j \int (\tilde{f}_j''(x))^2 dx.$$

Pour aller plus loin : voir les références dans **friedman2001elements**

# Introduction

# Le boosting : les données

## Données d'apprentissage supervisé

- Features $X_i \in \mathbb{R}^d$
- Labels $Y_i \in \mathbb{R}$ (régression) $Y_i \in \{-1, 1\}$ (classification)

## Weak learners

- Considérons un ensemble de "weak learners" $\mathcal{H}$
- Chaque learner $h : \mathbb{R}^d \to \mathbb{R}$ ou $\mathbb{R}^d \to \{-1, 1\}$ est un learner très simple
- Chaque learner simple est à peine meilleur que celui appris avec les $Y_i$ (moyenne)

## Exemples de weak learners

- Pour la régression : arbres de régression simple de faible profondeur, glm à quelques variables
- Pour la classification : arbres de décision simple de faible profondeur, modèles logistiques à quelques variables.

# Principe du boosting

## Un "strong learner"

On combine additivement des weak learners

$$g^{(B)}(x) = \sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x)$$

avec $\eta^{(b)} \geq 0$ pour espérer en obtenir un meilleur. $g^{(B)}$ est un learner dans le sens où

$$\widehat{Y}_i = g^{(B)}(X_i).$$

- Chaque $b = 1, \ldots, B$ est un pas/itération de boosting
- Le boosting fait partie des **ensemble methods** puisqu'il combine des learners faibles pour en fabriquer un meilleur.

Pour aller plus loin : voir **schapire1999brief** et **friedman2001greedy**

# Principe du "gradient boosting"

On cherche donc des fonctions $h^{(b)}$ du dictionnaire $\mathcal{H}$ et des réels $\eta^{(b)}$ tels que

$$g^{(B)}(x) = \sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x)$$

minimise le risque empirique

$$\sum_{i=1}^{n} L(Y_i, g(X_i)),$$

où $L$ est la fonction de coût (de perte) que l'on se fixe suivant le problème

- en régression linéaire $L(y, u) = (1/2)(y - u)^2$
- plus généralement, on peut prendre $L$ comme la log-densité dans le modèle considéré
- on verra qu'il y a d'autres choix possible en classification.

# L'algorithme "greedy"

Si c'était possible, on pourrait définir

$$\hat{g}^{(B)} = \sum_{b=1}^{B} \hat{\eta}^{(b)} \hat{h}^{(b)}(x)$$

avec

$$(\hat{\eta}^{(1)}, \ldots, \hat{\eta}^{(B)}, \hat{h}^{(1)}, \ldots, \hat{h}^{(B)}) = \underset{\eta^{(1)}, \ldots, \eta^{(B)} \in \mathbb{R}, h^{(1)}, \ldots, h^{(B)} \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} L(Y_i, \sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x))$$

mais même à $\eta^{(1)}, \ldots, \eta^{(B)}$ fixés il faut chercher parmi $\#(\mathcal{H})^B$ solutions possibles.

# Gradient-boosting

# Algorithme "greedy-stagewise"

On procède en fait pas-à-pas en définissant une suite $\hat{g}^{(b)}$ avec pour tout $b = 1, \ldots, B$ avec

$$\hat{g}^{(b+1)} = \hat{g}^{(b)} + \hat{\eta}^{(b+1)} \hat{h}^{(b+1)} \text{ où}$$

$$(\hat{\eta}^{(b+1)}, \hat{h}^{(b+1)}) = \underset{\eta \in \mathbb{R}, h \in \mathcal{H}}{\text{argmin}} \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)} + \eta h)$$

Dans nos problèmes, cette minimisation est également un problème difficile, on va alors procéder

- par descente de gradient
- avec la contrainte que la direction du pas de descente soit une fonction de $\mathcal{H}$.

# Rappel

## Descente de gradient

Considérons le problème de la minimisation de la fonction convexe et différentiable $J : \mathbb{R}^p \to \mathbb{R}$, la suite d'itérés $\theta^{(b)}$ définis par

$$\theta^{(b+1)} = \theta^{(b)} - \gamma \nabla J(\theta^{(b)})$$

alors

$$J(\theta^{(b+1)}) \leq J(\theta^{(b)})$$

(sous des conditions sur $J$ et $\gamma$ - cf cours d'optimisation du second semestre).

# Descente non-contrainte

A l'itération $b$,

- nous avons le learner $\hat{g}^{(b)} = \hat{g}^{(b)} + \eta\, \vec{0}$,
- on cherche à faire un pas de descente de gradient (pour l'instant quelconque) pour la fonction à optimiser $u \mapsto \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)}(X_i) + \eta u(X_i))$.

Attention : $u$ est une fonction de $\mathbb{R}^p \to \mathbb{R}$. Mais on ne s'intéresse qu'à ses valeurs aux points $X_1, \ldots, X_n$, on l'identifie donc à vecteur de $\mathbb{R}^n$

$$\left(\nabla_{u_i} \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)}(X_i) + \eta u_i)\right)_{u(X_i)} = \eta \left(\nabla_{u_i} L(Y_i, \hat{g}^{(b)}(X_i) + \eta u_i)\right)_{u(X_i)}$$

## Descente non-contrainte à rester dans $\mathcal{H}$

Le pas de gradient à considérer est donc dans la direction

$$\delta^{(b+1)} = \begin{pmatrix} \left( \nabla_{u_1} \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)}(X_i) + \eta u_i) \right)_0 \\ \dots \\ \left( \nabla_{u_n} \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)}(X_i) + \eta u_i) \right)_0 \end{pmatrix}$$

$$= \begin{pmatrix} \left( \nabla_{u_1} L(Y_1, \hat{g}^{(b)}(X_1) + \eta u_1) \right)_0 \\ \dots \\ \left( \nabla_{u_n} L(Y_n, \hat{g}^{(b)}(X_n) + \eta u_n) \right)_0 \end{pmatrix}$$

# Exemple dans le modèle linéaire

Dans le modèle linéaire, on prend

- $L(y, v+u) = (1/2)(y-v-u)^2$ avec
- $\nabla_u L(y, v+u) = -(y-v-u)$.

Le pas de gradient est donc dans la direction

$$\delta_{lm}^{(b+1)} = \begin{pmatrix} \left(\nabla_{u_1} L(Y_1, \hat{g}^{(b)}(X_1) + \eta u_1)\right)_0 \\ \cdots \\ \left(\nabla_{u_n} L(Y_n, \hat{g}^{(b)}(X_n) + \eta u_n)\right)_0 \end{pmatrix} = \begin{pmatrix} -(Y_1 - \hat{g}^{(b)}(X_1)) \\ \cdots \\ -(Y_n - \hat{g}^{(b)}(X_n)) \end{pmatrix}.$$

# Retour au problème contraint

Le problème d'optimisation au départ est

$$\hat{g}^{(b+1)} = \hat{g}^{(b)} + \hat{\eta}^{(b+1)}\hat{h}^{(b+1)} \text{ où}$$

$$(\hat{\eta}^{(b+1)}, \hat{h}^{(b+1)}) = \underset{\eta\in\mathbb{R}, h\in\mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)} + \eta h)$$

il faudrait donc que $\delta^{(b+1)}$ soit dans $\mathcal{H}$, ce qui ne peut pas être assuré à cette étape.

Pour s'assurer de rester dans le dictionnaire $\mathcal{H}$, on va prendre la fonction de $\mathcal{H}$ la plus proche de $\delta^{(b+1)}$ au sens des moindres carrés (de la norme $\ell_2$ aux points d'observation):

$$\hat{h}^{(b+1)} = \hat{h} \text{ avec } (\hat{h}, \hat{\nu}) = \underset{h\in\mathcal{H}, \nu\in\mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(\nu h(X_i) - \delta^{(b+1)}(i)\right)^2.$$

# La contrainte dans le modèle linéaire

Dans le modèle linéaire, cela s'écrit

$$\hat{h}^{(b+1)} = \hat{h} \text{ avec } (\hat{h}, \hat{\nu}) = \underset{h \in \mathcal{H}, \nu \in \mathbb{R}}{\text{argmin}} \sum_{i=1}^{n} \left( \nu\{-(Y_i - \hat{g}^{(b)}(X_i))\} - \delta^{(b+1)}(i) \right)^2.$$

On essaie donc d'apprendre un modèle (faible) sur les résidus $(-(Y_i - \hat{g}^{(b)}(X_i)))$ du modèle précédent : aux points où $g^{(b)}$ n'est pas très performant (grand résidus).

# Algorithme du gradient boosting

On optimise enfin en $\eta$ pour obtenir l'algorithme suivant.

**Algorithm 1** Gradient boosting

1: Posons $\hat{g}^{(0)} = \hat{a}$ avec $\hat{a} = \text{argmin}_{a \in \mathbb{R}} \sum_{i=1}^{n} L(Y_i, a)$
2: **for** $b = 1, \ldots, B$ **do**
3: $\quad \delta^{(b+1)} \leftarrow -\Big(\nabla_u L(Y_i, \hat{g}^{(b)}(X_i) + \eta u)\Big)_0 \; i = 1, \ldots, n$
4: $\quad \hat{h}^{(b+1)} \leftarrow \hat{h}$ avec $(\hat{h}, \hat{\nu}) = \text{argmin}_{h \in \mathcal{H}, \nu \in \mathbb{R}} \sum_{i=1}^{n} \left(\nu h(X_i) - \delta^{(b+1)}(i)\right)^2$.
5: $\quad \hat{\eta}^{(b+1)} \leftarrow \text{argmin}_{\eta \in \mathbb{R}} \sum_{i=1}^{n} L(Y_i, \hat{g}^{(b)} + \eta h^{(b+1)})$
6: **end for**
7: **return** Un boosting learner $g^{(B)}(x) = \left(\sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x)\right)$

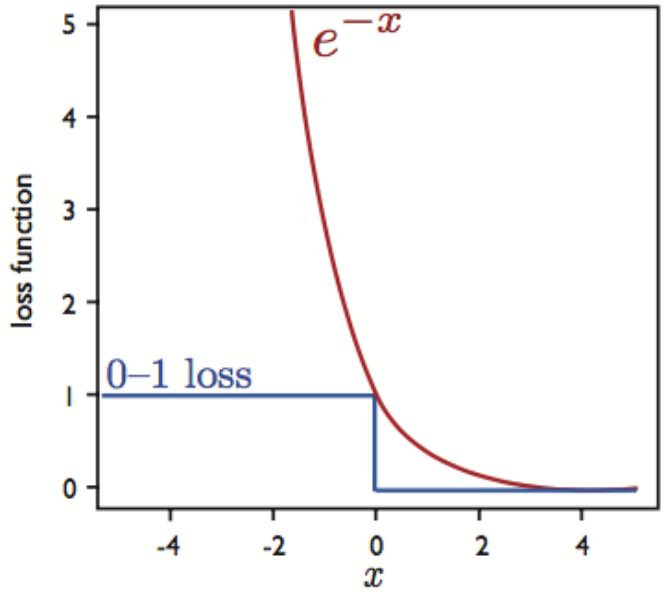
# AdaBoost

# Pour la classification

L'algorithme de boosting le plus connu en classification est AdaBoost (ADAptive BOOSTing). Il optimise la perte exponentielle

$$L(y, u) = \exp(-yu)$$

qui est une approximation de la perte 0/1 $\mathbb{1}_{y \neq u}$.




On va montrer qu'on peut aussi le voir comme un algorithme qui pondère séquentiellement les observations mal-classées

# Rappels et risque pondéré

La perte 0/1 moyenne (ou l'"accuracy") d'un classifieur $h$ est donnée par

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{Y_i \neq h(X_i)} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{Y_i h(X_i) < 0}$$

Pour pouvoir donner plus de poids à certaines observations, définissons un risque pondéré

$$\sum_{i=1}^{n} p^{(b)}(i) \mathbb{1}_{Y_i h(X_i) < 0},$$

où $\sum_{i=1}^{n} p^{(b)}(i) = 1$ et $b$ est le numéro de l'itération boosting courante.

# Initialisation

Au départ, on ne sait pas quelles observations sont difficiles à classer, on commence donc avec $p^{(0)}(i) = \frac{1}{n}$ pour $i = 1, \ldots, n$, et on choisit

$$g^{(0)} = h^{(0)} = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} p^{(0)}(i) \mathbb{1}_{Y_i h(X_i) < 0}$$

on calcule son erreur moyenne

$$\varepsilon(0) = \sum_{i=1}^{n} p^{(0)}(i) \mathbb{1}_{Y_i h^{(0)}(X_i) < 0}$$

# Comment changer les poids ?

On voudrait

- à l'itération suivante donner plus de poids aux mal-classés : $p^{(1)}(i)$ doit être grand si $Y_i h^{(0)}(X_i) < 0$
- pouvoir calculer simplement $p^{(b+1)}$ à partir des calculs de l'itération $b$, et on a
$$g^{(b+1)}(x) = g(b)(x) + \eta^{(b+1)} h^{(b+1)}(x).$$
- donc une bonne idée est de prendre
$$p^{(1)}(i) = \frac{e^{-\eta^{(0)} Y_i h^{(0)}(X_i)}}{\text{cst}},$$
car $e^{x+y} = e^x e^y$.

# A l'itération $b$

On pose alors

$$p^{(b+1)}(i) = p^{(b)}(i) \frac{e^{-\eta^{(b)} Y_i h^{(b)}(X_i)}}{Z^{(b)}},$$

de telle sorte que

$$p^{(b+1)}(i) = \frac{e^{-Y_i \sum_{a=1}^{b} \eta^{(a)} h^{(a)}(X_i)}}{n \prod_{a=1}^{b} Z^{(a)}} = \frac{e^{-Y_i g^{(b)}(X_i)}}{n \prod_{a=1}^{b} Z^{(a)}}$$

# Les poids (1)

Comme on veut $\sum_{i=1}^{n} p^{(b)}(i) = 1$, il faut poser

$$\frac{1}{n}\sum_{i=1}^{n} e^{-Y_i g^{(b)}(X_i)} = \sum_{i=1}^{n} p^{(b)}(i) \prod_{a=1}^{b} Z^{(a)} = \prod_{a=1}^{b} Z^{(a)}$$

et

$$\begin{aligned} Z^{(b)} &= \sum_{i=1}^{n} p^{(b)}(i) e^{-\eta^{(b)} Y_i h^{(b)}(X_i)} \\ &= \sum_{i:\, Y_i h^{(b)}(X_i)=1} p^{(b)}(i) e^{-\eta^{(b)}} + \sum_{i:\, Y_i h^{(b)}(X_i)=-1} p^{(b)}(i) e^{\eta^{(b)}} \end{aligned}$$

# Les poids (1)

Comme

$$\varepsilon(b) = \sum_{i=1}^{n} p^{(b)}(i) \mathbb{1}_{Y_i h^{(B)}(X_i) < 0}$$

on obtient

$$\begin{aligned} Z^{(b)} &= \sum_{i: Y_i h^{(b)}(X_i) = 1} p^{(b)}(i) e^{-\eta^{(b)}} + \sum_{i: Y_i h^{(b)}(X_i) = -1} p^{(b)}(i) e^{\eta^{(b)}} \\ &= (1 - \varepsilon^{(b)}) e^{-\eta^{(b)}} + \varepsilon^{(b)} e^{\eta^{(b)}} \end{aligned}$$

$$Z^{(b)} = (1 - \varepsilon^{(b)}) \sqrt{\frac{\varepsilon^{(b)}}{1 - \varepsilon^{(b)}}} + \varepsilon^{(b)} \sqrt{\frac{1 - \varepsilon^{(b)}}{\varepsilon^{(b)}}} = 2\sqrt{\varepsilon^{(b)}(1 - \varepsilon^{(b)})}.$$

# Les poids (2)

On met le poids aux obsservations mal-classées d'une part et bien classées d'autre, on prend donc $\eta^{(b)}$ tel que

$$(1 - \varepsilon^{(b)})e^{-\eta^{(b)}} = \varepsilon^{(b)} e^{\eta^{(b)}},$$

ou bien

$$\eta^{(b)} = \frac{1}{n} \log\left(\frac{1 - \varepsilon^{(b)}}{\varepsilon^{(b)}}\right)$$

**Algorithm 2** Adaboost

Posons $p_1(i) = 1/n$ pour $i = 1, \ldots, n$
**for** $b = 1, \ldots B$ **do**
    $h^{(b)} \in \text{argmin}_{h \in H} \sum_{i=1}^{n} p^{(b)}(i) \mathbb{1}_{Y_i h(X_i) < 0}$
    $\varepsilon^{(b)} \leftarrow \sum_{i=1}^{n} p^{(b)}(i) \mathbb{1}_{Y_i h^{(b)}(X_i) < 0}$
    $\eta^{(b)} \leftarrow \frac{1}{2} \log\left(\frac{1-\varepsilon^{(b)}}{\varepsilon^{(b)}}\right)$
    $Z^{(b)} \leftarrow 2\sqrt{\varepsilon^{(b)}(1-\varepsilon^{(b)})}$
    $p^{(b+1)}(i) \leftarrow p^{(b)}(i) \frac{e^{-\eta^{(b)} Y_i h^{(b)}(X_i)}}{Z^{(b)}}$
**end for**
**return** Un boosting classifieur $g^{(B)}(x) = \text{signe}\Big(\sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x)\Big)$

# En pratique

Il reste à fixer les hyper-paramètres

- pour le dictionnaire $\mathcal{H}$ :
  - si on choisit des arbres, il reste à fixer leurs profondeurs
  - si on choisir des glm, il reste à fixer le nombre de variables qu'ils contiennent
- et $B$ le nombre d'itérations

## Régularisation

Pour rendre la performance de l'algorithme moins dépendant du choix de $B$, on régularise en ajoutant le paramètre

$$\hat{g}^{(b+1)} = \hat{g}^{(b)} + \lambda \hat{\eta}^{(b+1)} \hat{h}^{(b+1)}$$

que l'on choisit à la fin par cross-validation.

Voir le GitHub d'XGBoost pour plus de détails sur les paramètres
https://github.com/dmlc/xgboost/blob/master/doc/parameter.md

# XGBoost

```
boosted = xgboost(data = x_train, label = y_train,
          objective = "reg:linear" , # choix de la perte
          booster = "gbtree", # choix du dictionnaire
          max.depth = 2, # longueur maximale d'un arbre faible
          eta = 1, # paramètre de régularisation
          nrounds = 5 ) # choix du nombre weak learners )
```

```
## [1]  train-rmse:0.100551
## [2]  train-rmse:0.100149
## [3]  train-rmse:0.099851
## [4]  train-rmse:0.099525
## [5]  train-rmse:0.099184
```

```
mean((predict(boosted,x_test)-y_test)^2)
```

```
## [1] 0.01117229
```

# XGBoost in caret Library

```
# 3-fold cross-validation,
# repeating twice, and
#using random search for tuning hyper-parameters
fitControl <- trainControl(method = "cv", number = 3,
                           repeats = 2, search = "random")

boosted.cv <- train(y~., data = df_train, method = "xgbTree",
                trControl = fitControl)
```

# Résultats de la cross-validation

```
boosted.cv$results[,1:5]
```

```
##         eta max_depth    gamma colsample_bytree min_child_weight
## 3 0.46922759         4 7.672972        0.3101102                3
## 2 0.13431174         9 9.793225        0.4193648               15
## 1 0.08445088        10 4.659134        0.4166562               16
```

```
boosted.cv$results[,6:10]
```

```
##   subsample nrounds      RMSE Rsquared      RMSESD
## 3 0.5587540     122 0.1007269      NaN 0.001052944
## 2 0.2993150     152 0.1007246      NaN 0.001051231
## 1 0.2614799     375 0.1007237      NaN 0.001049009
```

# MSE

```
mean((predict(lm,newdata = df_test)-y_test)^2)
```

```
## [1] 0.01079587
```

```
mean((predict(boosted,x_test)-y_test)^2)
```

```
## [1] 0.01117229
```

```
mean((predict(boosted.cv)-df_test$y)^2)
```

```
## [1] 0.01069825
```

Bibliographie (pour ce chapitre) :