Projet Régression non paramétrée

SEFFANE Asmaa

2023-02-20

On dispose de données (xi, yi)1≤i≤5000 où les xi et les yi sont les réalisations de variables aléatoires réelles (Xi, Yi)1≤i≤5000 admettant la représentation

$$Yi = r(Xi) + \xi i , i = 1, \ldots 5000,$$

où:

—les ξi sont indépendantes et identiquement distribuées, E[ξ1] = 0 et E[(ξ^2)1]= σ

— les Xi sont indépendantes et identiquement distribuées de densité g, et indépendantes des ξi

Ce projet porte sur l'étude de la densité g et de la fonction de régression r.

```
library(reader)
```

```
## Loading required package: NCmisc
```

```
##
## Attaching package: 'reader'
```

```
## The following objects are masked from 'package:NCmisc':
##
##     cat.path, get.ext, rmv.ext
```

```
library(stats)
library(KernSmooth)
```

```
## KernSmooth 2.23 loaded
## Copyright M. P. Wand 1997-2009
```

```
library(np)
```

```
## Nonparametric Kernel Methods for Mixed Datatypes (version 0.60-16)
## [vignette("np_faq",package="np") provides answers to frequently asked ques
tions]
## [vignette("np",package="np") an overview]
## [vignette("entropy_np",package="np") an overview of entropy-based methods]
```

```
library(caret)
```

```
## Loading required package: ggplot2
```

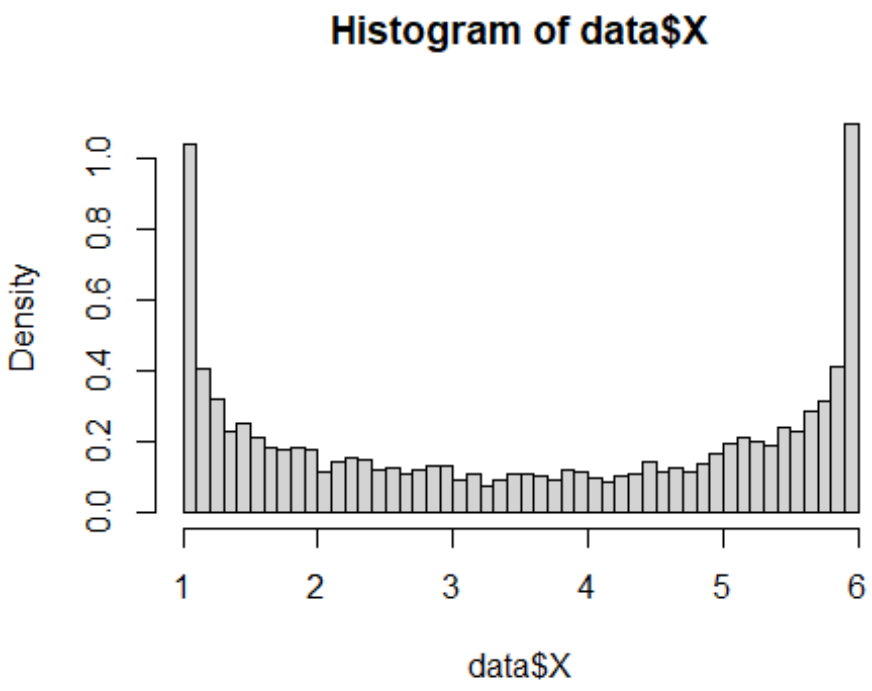
```
## Loading required package: lattice
```

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.
0.0 ──
## ✓ dplyr     1.1.0     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
```

```
## ✓ lubridate 1.9.2     ✓ tibble    3.1.8
## ✓ purrr     1.0.1     ✓ tidyr     1.3.0

## ── Conflicts ──────────────────────────────────────── tidyverse_conflict
s() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ✖ purrr::lift()   masks caret::lift()
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force
all conflicts to become errors

library(caTools)
library(ISLR)

## Warning: package 'ISLR' was built under R version 4.2.3

data <- read.table("Data.txt", header = T)
```

Je dessine les histogrammes des X et des Y



A première vue, On ne peut pas facilement dire à quelle loi de probabilité l'histogramme des X ressemble.

Etude de la densité g de X:

Je rappelle que l'estimateur de la densité s'écrit sous la forme:

$$\hat{f}_{n,h}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x_i - x}{h}\right)$$

avec K est une application dans R intégrable tel que:

$$\int K(x)dx = 1$$

Et h>0 est appelé fenêtre, c'est un paramètre dit de lissage.

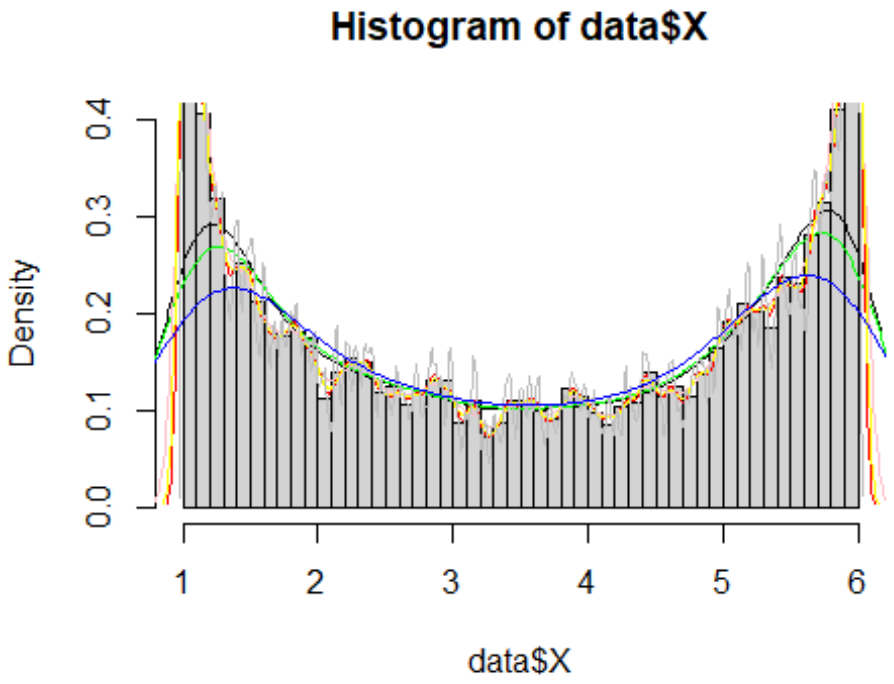
Maintenant on va définir différents estimateurs de la densité g(X) avec différentes fenêtres de lissage h.

les arguments utilisés cette fonction sont:

- kernel: le noyaux, on sait que le choix de noyaux a une influence non négligeable quand h est bien choisit, donc je vais utiliser dans la suite le noyaux Gaussien.
- bw: exprime la fenêtres de lissage, j'en ai choisi plusieurs:
    1. « bw.nrd » est celle de bendwidth.
    2. « nrd0 » est par défaut de R.
    3. « dpill » est en plugging.
    4. « bw.ucv » calculé en utilisant la méthode de cross validation.
    5. Des valeurs numériques en dur (0.05, 0.01, 0.5).

```
d1 = density(data$X, bw = bw.nrd(data$X) , kernel = "gaussian")

d = density(data$X, bw = "nrd0", kernel = "gaussian")

d2 = density(data$X, bw = dpill(data$X, data$Y), kernel = "gaussian")

d3 = density(data$X, bw = bw.ucv(x = data$X) , kernel = "gaussian")

## Warning in bw.ucv(x = data$X): minimum occurred at one end of the range

d4 = density(data$X, bw = 0.05, kernel = "gaussian")

d5 = density(data$X, bw = 0.01, kernel = "gaussian")

d6 = density(data$X, bw = 0.5, kernel = "gaussian")
```

Je dessine tous les estimateurs de g(X) sur le même graphe avec l'histogramme des X



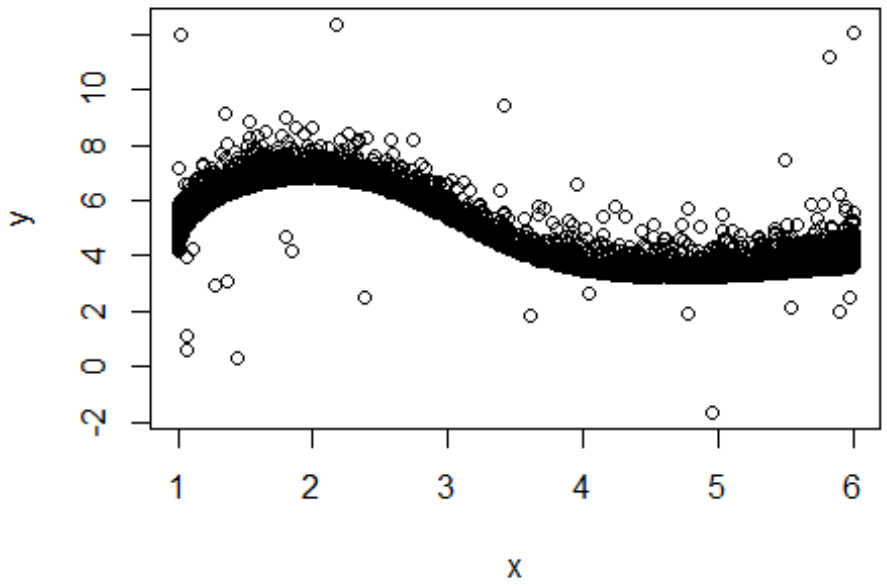
Je remarque que le choix de h a vraiment un grand effet sur la densité g(x):

- la densité d5, avec h = 0.05, n'est pas stable, on aperçoit une grande variance.
- les densité d1, avec h est de bandwith, d, h est donné par défaut de R, et d6, avec h = 0.5, sont toutes bonne au milieu mais n'atteint pas les valeurs en extrémité.
- la densité d2, avec h calculé en plugging, d3 , avec h calculé en creuse validation, et d4, avec h=0.05, sont les plus proche.

Donc on peut choisir la valeur de paramètre de lissage parmi ces trois, on peut choisir une parmi eux pour l'utilisé dans tout notre projet.

Etude de la fonction de régression :

Je commence tout d'abord par dessiner le plot des X et Y, pour avoir une idée.



il est claire du graphe que la relation entre X et Y n'est pas linéaire, et en plus on a pas une loi exacte pour le couple (X,Y). donc pour

$$Y = r(X_i) + \xi_i, i = 1, \ldots 5000$$

on cherche à estimer la fonction de régression r.

- L'estimateur de Nadaraya Watson s'écrit sous la forme:

$$\hat{r}_{n,h}(x) = \frac{\sum_{i=1}^{n} Y_i \, K\left(\frac{x_i - x}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)} 1\left(\sum K\left(\frac{x_i - x}{h}\right) \neq 0\right)$$

  avec K un noyau d'ordre 1.

Je choisis en premier l'estimateur de la régression en utilisant les polynômes locaux, car dans la suite on va voir l'estimateur de Nadaraya Watson et la méthode de validation croisée.

On a:

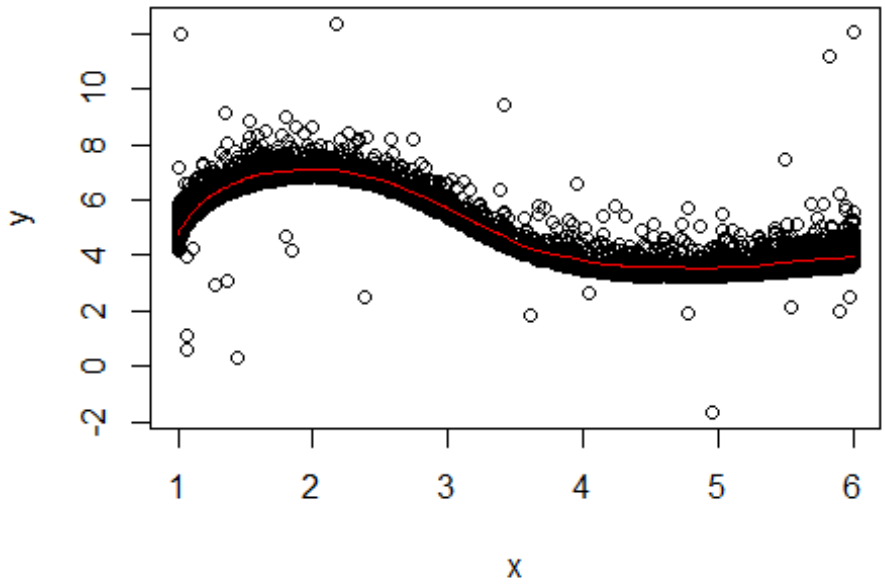
```
library(np)
library( KernSmooth)
h= dpill(data$X, data$Y)
fit <- locpoly(data$X, data$Y, bandwidth = h)
```

Avec:

- hdpill: Utilise la méthodologie de plug-in direct pour sélectionner la bande passante d'une estimation de régression de noyau gaussien linéaire local, et c'est la fenêtre de lissage qu'on utilisera dans la suite.
- locpoly: estime la fonction de régression en utilisant les polynômes locaux.

Je dessine le graphe de l'estimateur de la régression choisit :

```
plot(x, y)
lines(fit, col="red")
```



On remarque que le h choisit construit est parfait, et que l'estimateur choisit représente bien la fonction de régression .

Essayons un autre estimateur de la fonction de régression, Nadaraya Watson, la fonction ci-dessous nous permet de calculer les valeurs en utilisant cette méthode :

```
h=dpill(data$X,data$Y)

Kh=function(y){1/h*dnorm(y,0,1)}

NW=function(x0,X,Y,h){
  NW=mean(Kh(x0-X)*Y)/mean(Kh(x0-X))
  return(NW)
```

```
}
#une fonction qui nous donne les valeur de NW de x
NWX = function(x){
  y = NW(x,data$X, data$Y, h)
  return(y)
}
```

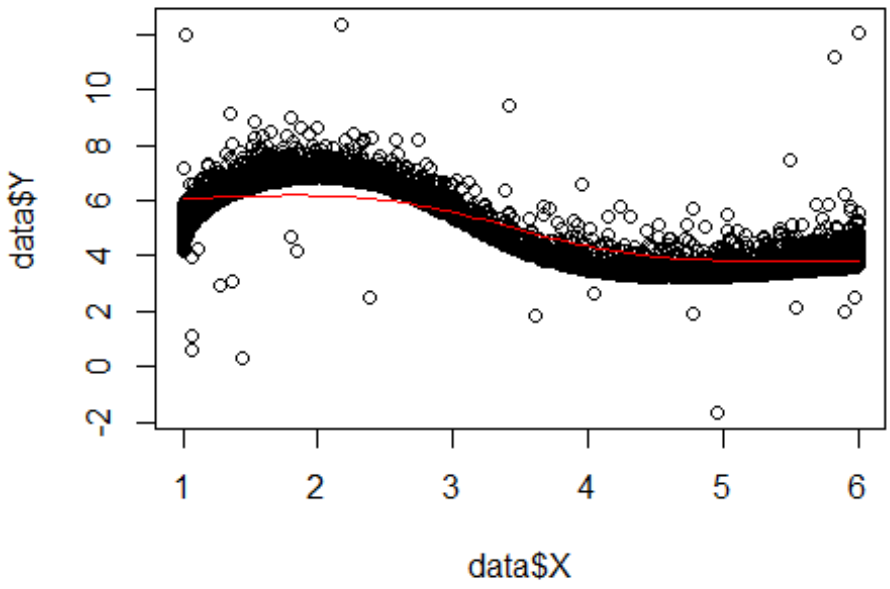
Test :

```
NW(min(data$X),data$X, data$Y, h)

## [1] 6.060012

NW(max(data$X),data$X,data$Y,h)

## [1] 3.802583
```

Notre fonction fonctionne bien et les valeurs sont correctes.

Maintenant je vais l'appliquer sur toute les données :

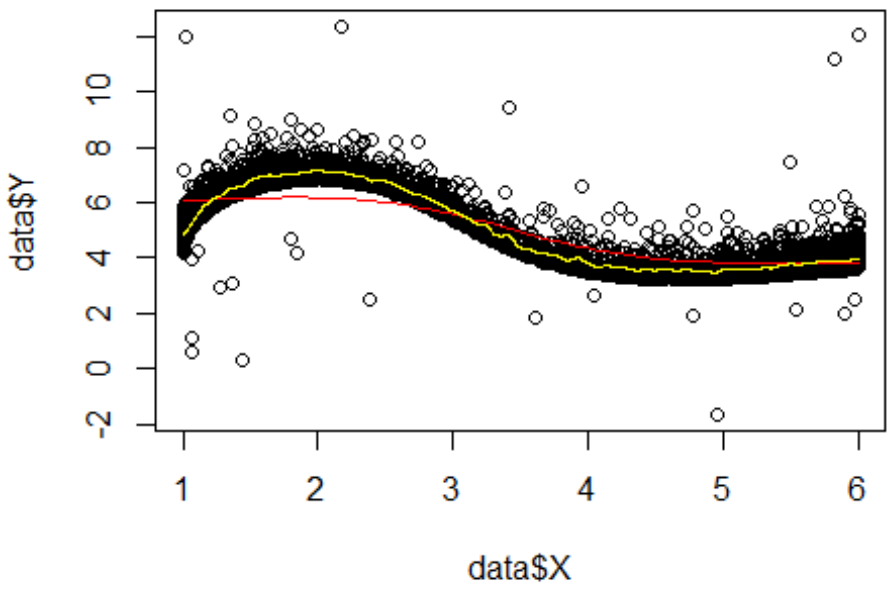
```
grid = seq(min(data$X), max(data$X), length.out = 100)
gridY = lapply(grid, NWX)

grid_x = array(grid, c(100, 1))
grid_y = array(gridY, c(100, 1))

grid_xy <- data.frame(
  grid_x = array(grid, c(100, 1)),
  grid_y = array(gridY, c(100, 1))
)


plot(data$X, data$Y)
lines(grid_xy, col = "red")
```



On voit bien ce premier estimateur n'est pas assez performant, mais il suit la tendance de la distribution des points.

Maintenant on ajoute l'estimateur de Nadaraya Watson déjà disponible dans le package KernSmooth :

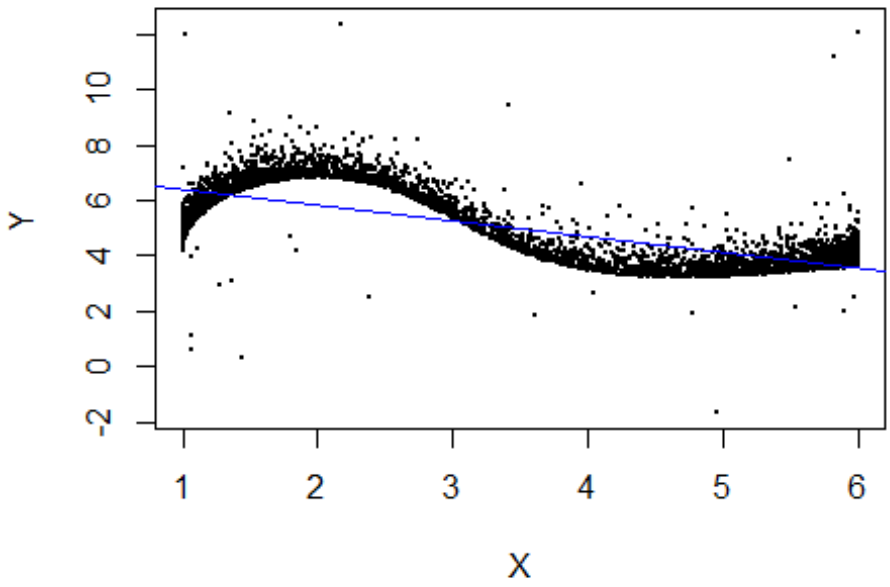
```
fit2 <- ksmooth(data$X, data$Y, "normal", bandwidth = h)
plot(data$X, data$Y)
lines(grid_xy, col = "red")
lines(fit2, col = "yellow")
```



Il est claire que ce dernier estimateur est plus performant que le premier.

Essayons le modèle linaire:

```
ML <- lm(Y ~ X, data = data)

plot(data$X,data$Y,pch=20,cex=0.01,xlab="X",ylab="Y")
abline(ML, col="blue")
```



On voit bien que la régression linéaire ne convient pas, loin derrière la régression non paramétrique.

Validation croisée:

On commence par l'estimateur de l'erreur quadratique :

$$\Re^2\left(\hat{r}(x)\right) = \frac{1}{n}\sum\left(\widehat{r_{n,h}^{(-)}}(x) - Y_i\right)^2 = \frac{1}{n}\sum(-\xi_i)^2 = \frac{1}{n}\sum(\xi_i)^2$$

La validation croisée « cross-validation » est, en apprentissage automatique, une méthode d'estimation de fiabilité d'un modèle fondée sur une technique d'échantillonnage.
Pour effectuer une cross-validation, il est nécessaire d'écarter en amont une partie des données du dataset d'entraînement.

Ces données ne seront pas utilisées pour entraîner le modèle, mais plus tard pour tester et valider le modèle.

- Les principales techniques de validation croisée sont :
  La technique du Train-Test Split:

  L'approche Train-Test Split consiste à décomposer de manière aléatoire un ensemble de données. Une partie servira à l'entraînement du modèle de Machine Learning, l'autre partie permettra de le tester pour la validation.

  En général, on réserve 70% à 80% des données du dataset pour l'entraînement. Les 20 à 30% restants seront exploités pour la Cross-Validation. Cette technique est efficace, sauf si les données sont limitées. Il peut alors manquer certaines informations sur les données qui n'ont pas été utilisées pour l'entraînement, et les résultats peuvent donc être hautement biaisés.

- La méthode K-Folds

  La technique K-Folds est simple à comprendre, et particulièrement populaire. Par rapport aux autres approches de Cross-Validation, elle résulte généralement sur un modèle moins biaisé.

  Pour cause, elle permet d'assurer que toutes les observations de l'ensemble de données original aient la chance d'apparaître dans l'ensemble d'entraînement et dans l'ensemble de test. En cas de données d'input limitées, il s'agit donc de l'une des meilleures approches

  La valeur de K ne doit être ni trop basse ni trop haute, et on choisit généralement une valeur comprise entre 5 et 10 en fonction de l'envergure du dataset. Par exemple, si K=10, le dataset sera divisé en 10 parties

  On ajuste ensuite le modèle en utilisant les folds K-1 (K moins 1). Le modèle est validé en utilisant le K-fold restant. Les scores et les erreurs doivent être notés.

Le processus est répété jusqu'à ce que chaque K-fold serve au sein de l'ensemble d'entraînement. La moyenne des scores enregistrés est la métrique de performance du modèle.

Dans notre cas, on utilisera la méthode K-Folds avec K=2 :

On coupe l''échantillon en deux, tel que:

- i ∈ J− = {1,....,2500} : échantillon 1
- i ∈ J+ ={2501, . . . , 5000} : échantillon 2

On pose l'erreur de l'estimateur :

$$\xi_i = Y_i - \hat{r}_{n,h}^{(-)}(X_i), i \in J_+$$

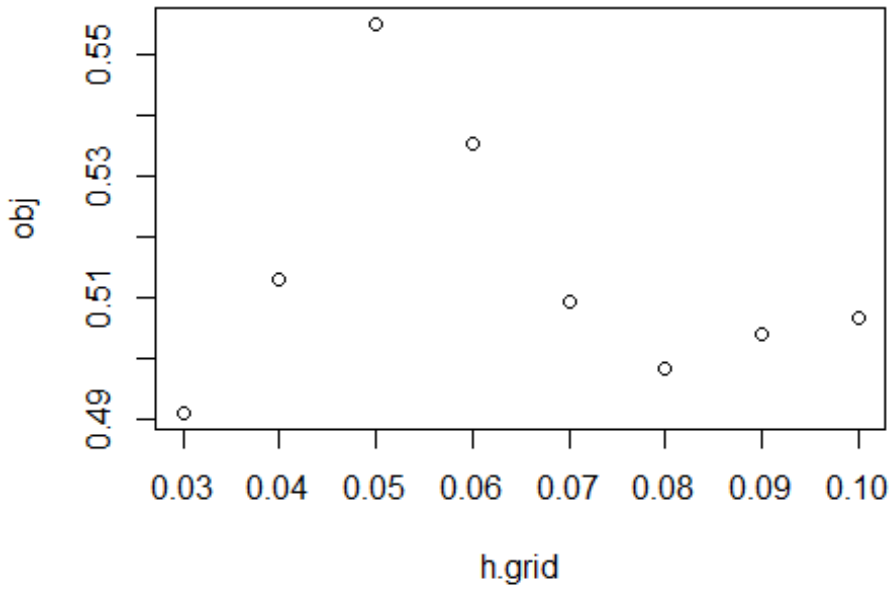
L'estimateur construit à l'aide de (Xi, Yi) i∈ J− est noté:

$$\widehat{r_{n,h}^{(-)}}$$

Ce découpage en J+ et J- permet d'appliquer la validation croisée pour obtenir un h minimal, avec le J- le dataset d'entraînement et J+ le dataset de validation.

```
cv <- function(X, Y, h, K = dnorm) {
  spl = sample.split(X, SplitRatio = 0.5)
  data_train_X = subset(X, spl == TRUE)
  data_test_X = subset(X, spl == FALSE)

  data_train_Y = subset(Y, spl == TRUE)
  data_test_Y = subset(Y, spl == FALSE)


NWX_train = function(x){
    y = NW(x,data_train_X, data_train_Y, h)
    return(y)
}
res_test_Y <- sapply(data_test_X, NWX_train)
#plot(data_test_X, data_test_Y)
#lines(res_test_Y, col = "yellow")
error_nwx <- mean((data_test_Y - res_test_Y)^2)
return(error_nwx)
}

cv.grid <- function(X, Y, h.grid = seq(0.03, 0.1, 0.01), K = dnorm, plot.cv =
TRUE)
   {
   obj <- sapply(h.grid, function(h) cv(X = X, Y = Y, h = h, K = K))
   h <- h.grid[which.min(obj)]
```

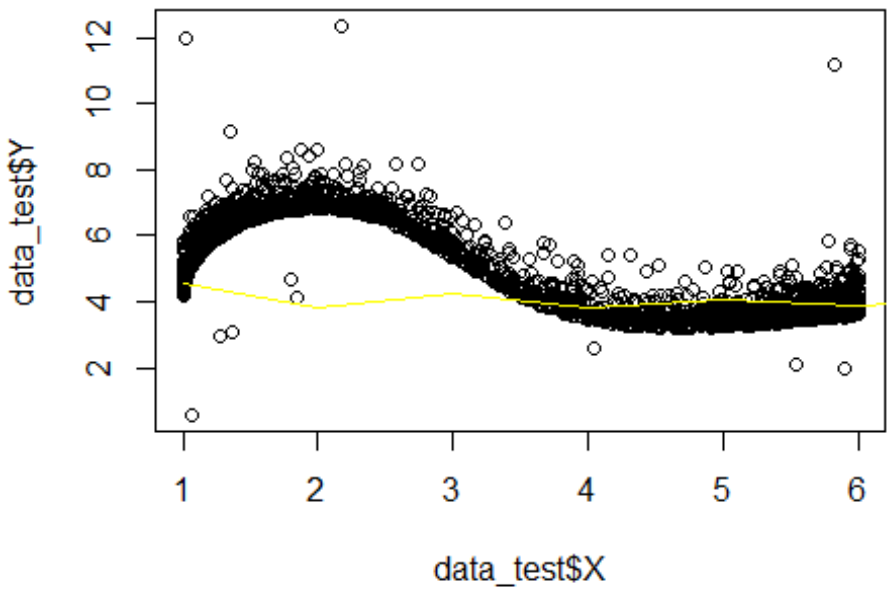
```
   if (plot.cv) { plot(h.grid, obj)}
   print(h)
}

res_test_cv.grid = cv.grid(data$X, data$Y, h.grid = seq(0.03, 0.1, 0.01), K =
dnorm, plot.cv = TRUE)
```




```
## [1] 0.03
```

La premiere fonction cv() permet d'effectuer une validation croisée. la deuxiéème fonction cv.grid() permet d'appliquer la fonction cv() sur une liste de h. dans mon exemple h = seq(0.03, 0.1, 0.01)

```
spl = sample.split(data$X, SplitRatio = 0.5)
 data_train = subset(data, spl == TRUE)
 data_test = subset(data, spl == FALSE)

 NWX_train = function(x){
   y = NW(x,data_train$X, data_train$Y, h = 0.06)
   return(y)
 }

res_test_Y <- sapply(data_test$X, NWX_train)


plot(data_test$X, data_test$Y)
lines(res_test_Y, col = "yellow")
```




```
error_nwx <- mean((data_test$Y - res_test_Y)^2)
print(error_nwx)

## [1] 0.5002144
```

Conclusion:

Grâce à la deuxième fonction on arrive à avoir la valeur optimale de h qui permet de réduire l'erreur quadratique de notre estimateur.

On constate aussi que cette valeur optimale de h n'est pas toujours la même à chaque exécution, ce qui est normale du fait que le dataset d'entrainement n'est pas toujours le même à chaque fois.

Malgré le choix optimal de h, notre estimation n'est toujours pas satisfaisante. L'estimateur ksmooth est largement meilleur (graphiquement).