# Rapport Régression Non-Paramétrique

Olivier Berthier

2024-05-01

L'objet de ce rapport consiste à étudier un jeu de données par des méthodes non-paramétriques.

```
library(FNN)
library(KernSmooth)
library(readr)
d <- read_table("C:/Users/olivi/OneDrive - Université Paris-Dauphine/Documents ODD gram/Forma
tion/Cours/Estimation non paramétrique/Data.txt",
    col_types = cols(nb = col_skip()))
```

# Exploration des données

```
x <- d$X# Pour toute la suite du rapport nous appellerons x les données de la colonne "X" du
dataframe à étudier
y <- d$Y# Pour toute la suite du rapport nous appellerons y les données de la colonne "Y" du
dataframe à étudier
summary (d)
```

```
##        X               Y
##  Min.   :1.000   Min.   :-1.656
##  1st Qu.:1.622   1st Qu.: 3.695
##  Median :3.573   Median : 4.484
##  Mean   :3.538   Mean   : 4.926
##  3rd Qu.:5.424   3rd Qu.: 6.241
##  Max.   :6.000   Max.   :12.348
```

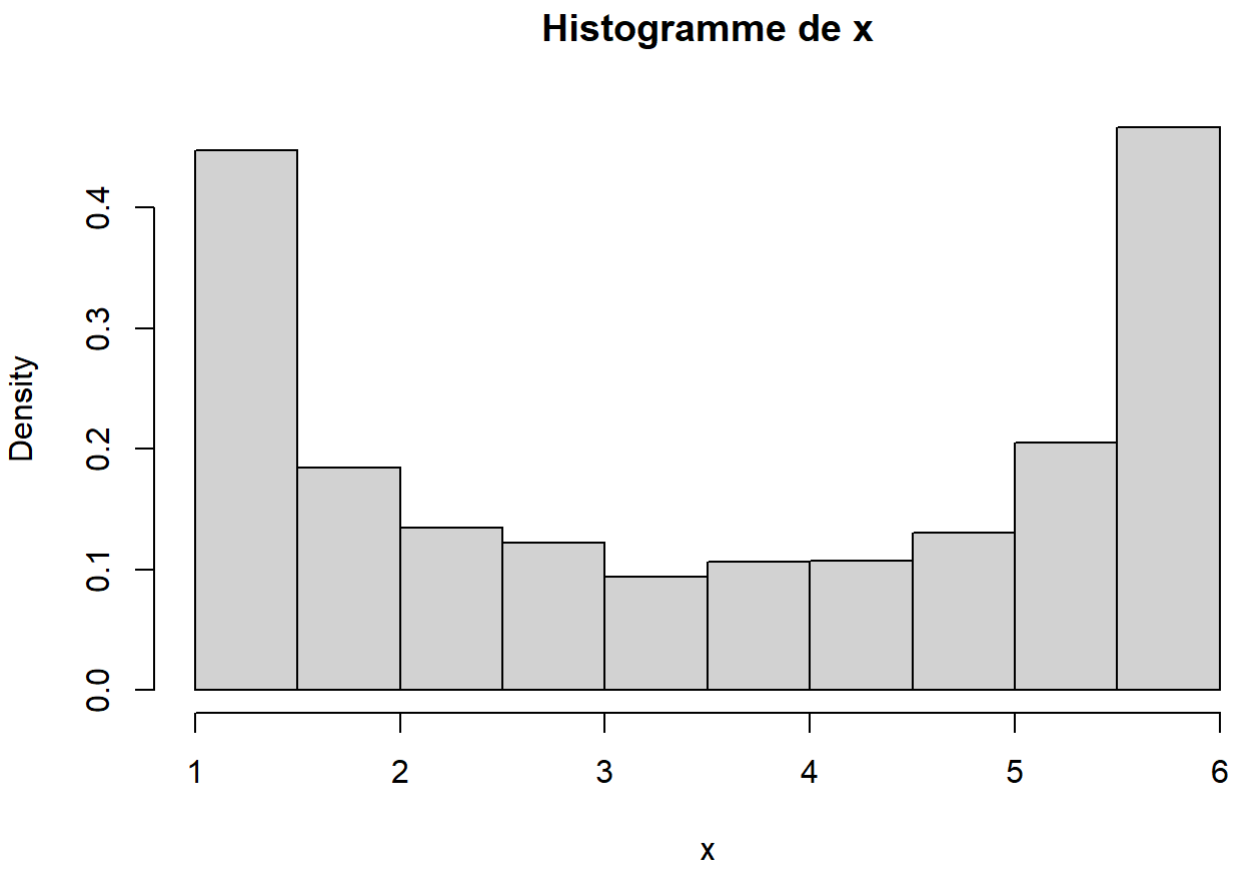
```
length (x)
```

```
## [1] 5000
```

Le dataframe à étudier comporte 5000 observations. La variable explicative x prend des valeurs entre 1 et 6. La variable à expliquer y prend des valeurs entre -1.656 et 12.348.

# Éstimation de la densité de X

```
hist(x, freq=F, main="Histogramme de x")
```




L'axe des y est représenté en densité de probabilité plutôt qu'en fréquence brute (freq= False). Par défaut, la fonction "hist()" calcule le nombre optimal de classes selon la méthode de Sturges où :

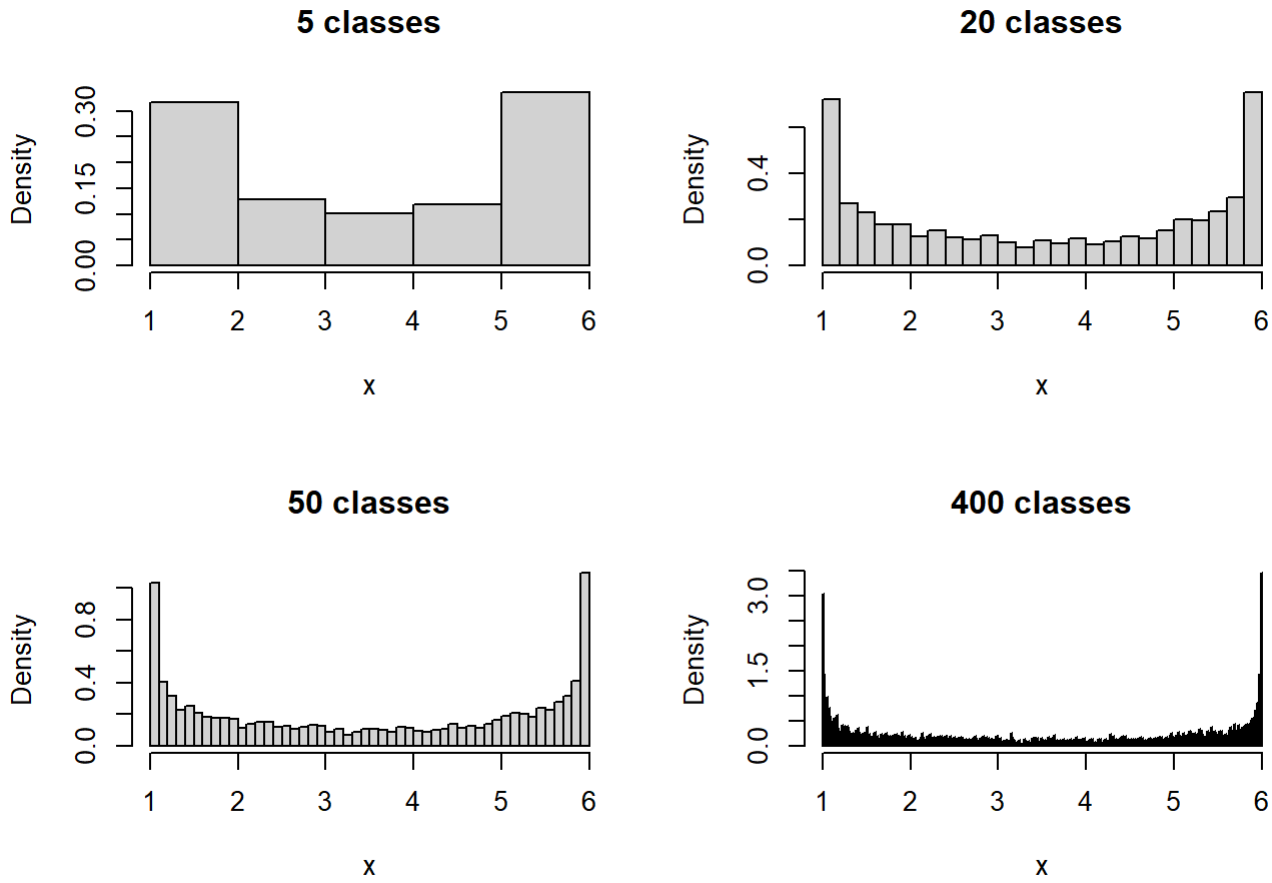
$$k = \lceil \log_2(n) + 1 \rceil$$

où k est le nombre de classes et n le nombre d'échantillons. Le calcul donne ici 10 classes.

On voit que la densité suit grossièrement une forme de U avec le plus grand nombre de valeurs rassemblées autour des extrêmes (ici 1 et 6) et peu de valeurs au milieu.

Essayons d'autres nombres de classes pour l'histogramme de x:

```
par(mfrow=c(2,2))
hist(x, breaks = 5, freq=F, main = "5 classes")
hist(x, breaks = 20, freq=F, main = "20 classes")
hist(x, breaks = 50, freq=F, main = "50 classes")
hist(x, breaks = 400, freq=F, main = "400 classes")
```




Nous constatons que plus le nombre de classes est grand plus le nombre de variations augmente (au risque de ne pas être significatives) : risque de variance trop grande. A l'inverse, moins il y a de classes, plus la représentation tend à être grossière et moins discriminante, on risque alors de perdre de l'information significative : risque du biais trop grand.

Dans notre cas, une valeur du nombre de classess choisie entre 10 et 20 semble proposer un bon compromis.

Notons que l'on peut faire le même raisonnement en manipulant la largeur de la classe h au lieu du nombre de classes k, selon la formule :

$$h = \frac{b - a}{k}$$

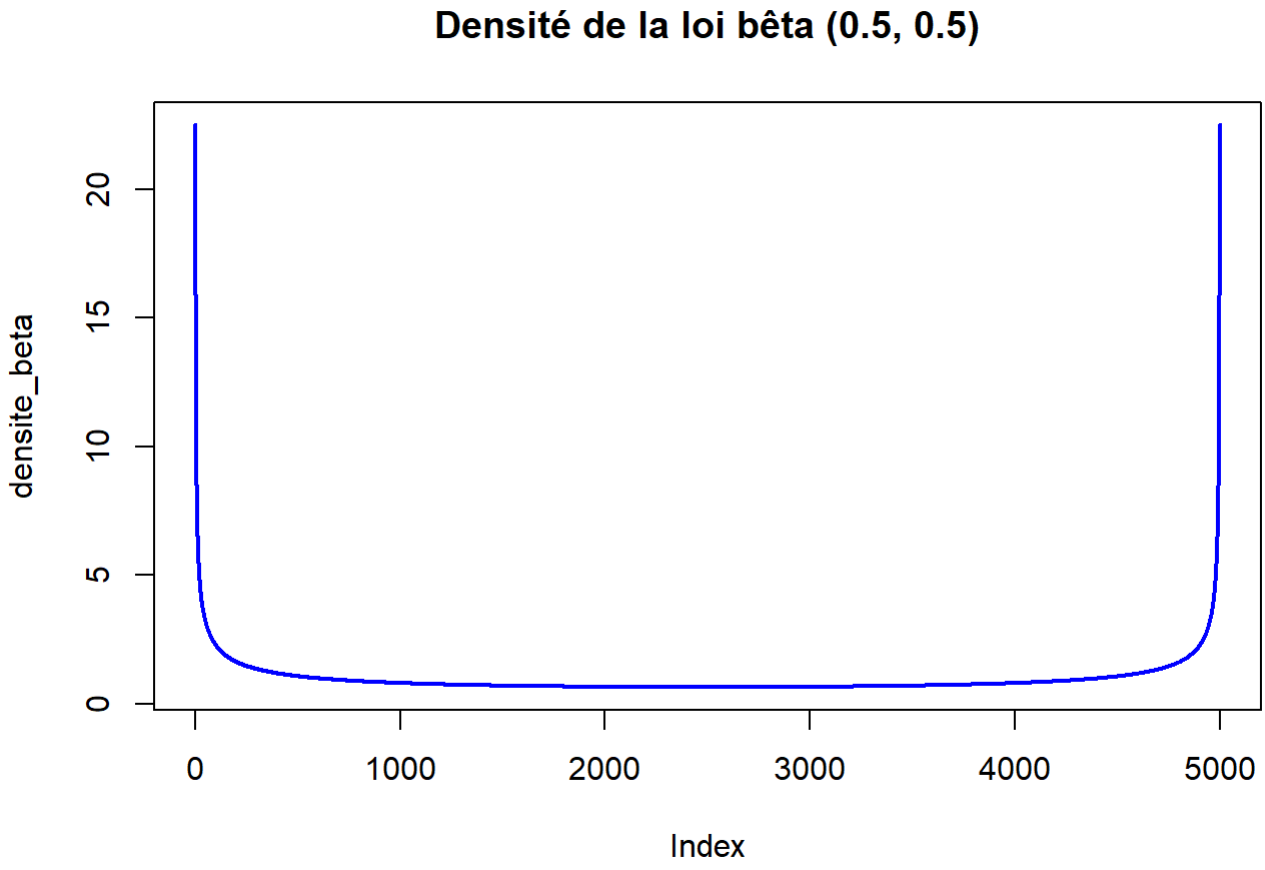
où a est la plus petite valeur des données, b la plus grande valeur des données, k le nombre de classes, et h la largeur de classe.

Le nombre optimal de classes dépend donc aussi du nombre d'observations, i.e. : pour un échantillon de la même population mais de taille différente de celui proposé ici, le h optimal serait différent.

Avant d'entamer une estimation de la densité par des méthodes non-paramétriques, voyons si l'on peut approcher x par une loi connue. Graphiquement, la densité de x peut faire penser à une loi bêta. Après quelques recherches documentaires sur la loi bêta, nous trouvons intéressants les paramètres alpha = 0.5 et bêta = 0.5.

Observons:

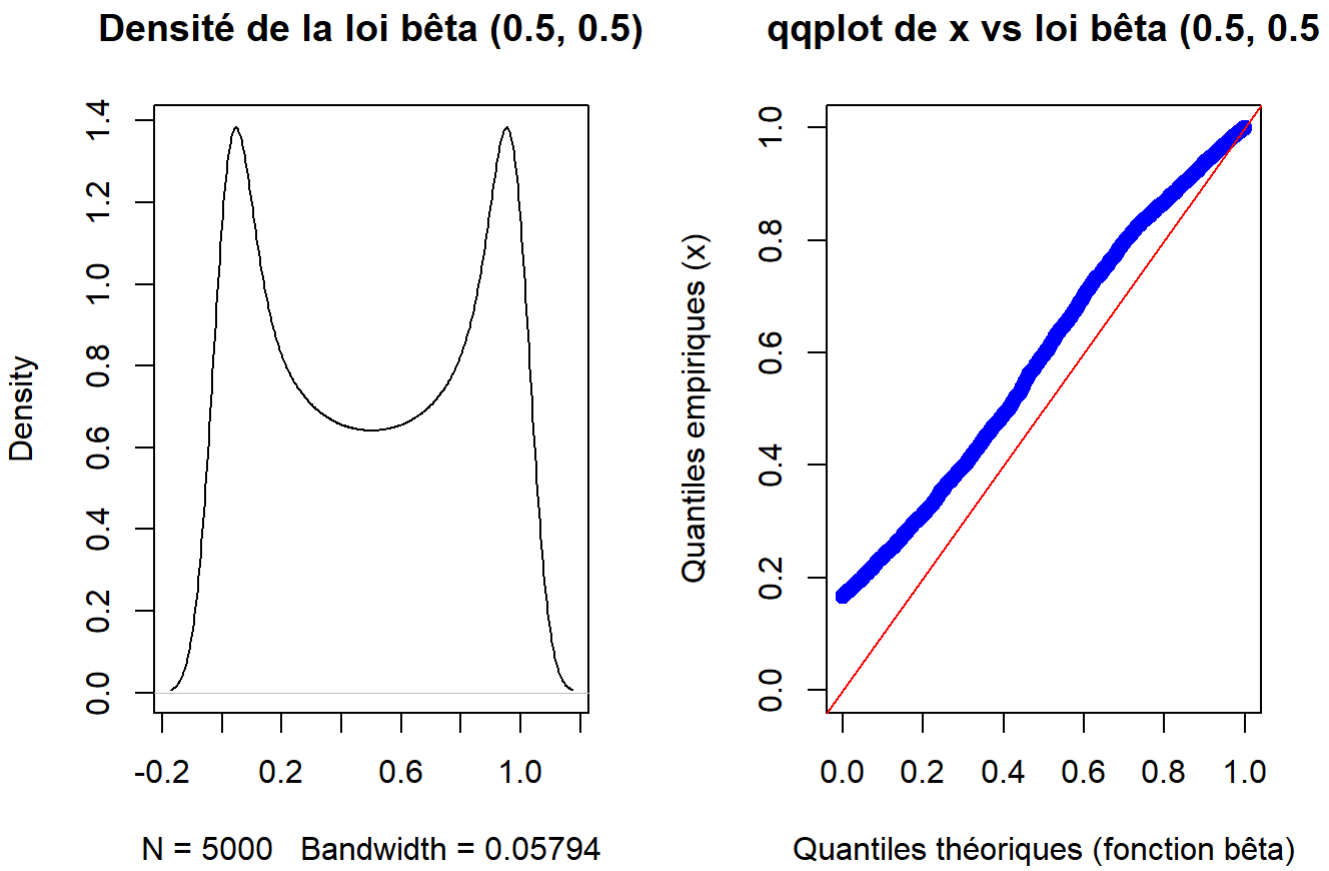
```
alpha <- 0.5
beta <- 0.5
grid <- seq(0 ,1, length.out = 5000)
densite_beta <- dbeta(grid, alpha, beta)         # Calcul de la densité de la fonction bêta
plot (densite_beta, type="l", col="blue", lwd =2, main = "Densité de la loi bêta (0.5, 0.5)")
```




En effet, la densité de la loi bêta (0.5, 0.5) ressemble suffisamment à la densité de x pour que l'on investigue plus avant.

Comparons les deux distributions:

```
par(mfrow=c(1,2))
q_beta <- qbeta (ppoints(5000), 0.5,0.5) # Calcule les quantiles théoriques de la distributio
n bêta
plot (density(q_beta),main = "Densité de la loi bêta (0.5, 0.5)") # Graphique de la densité d
e la loi bêta (0.5, 0.5)
x_norm = x / 6                          # Normalise x pour prendre des valeurs dans [0,1]
qqplot(q_beta, x_norm, col = "blue",    # QQ plot x vs loi bêta (0.5, 0.5)
       xlab = "Quantiles théoriques (fonction bêta)",
       ylab = "Quantiles empiriques (x)",
       main = "qqplot de x vs loi bêta (0.5, 0.5",
       ylim = c(0, max(x/6)))  # Définit les limites de l'axe y
abline(0, 1, col = "red") # Ajoute une ligne de référence pour montrer une relation linéaire
entre les quantiles
```




```
ks.test(x/6, q_beta)  # Test de Kolmogorov-Smirnov x vs loi bêta (0.5, 0.5)
```

```
##
##  Asymptotic two-sample Kolmogorov-Smirnov test
##
## data:  x/6 and q_beta
## D = 0.2678, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

Le qqplot montre bien une “ressemblance” entre les deux lois mais non exploitable car insuffisante. Le test de Kolmogorov-Smirnov avec une p-value très faible le confirme (notons cependant que la valeur de la distance D reste assez faible).

Nous pouvons essayer d’affiner notre démarche en calculant par la méthode des moments les meilleurs paramètres possibles alpha et bêta pour faire “coller” le plus possible la loi bêta à x.

Les paramètres de la loi bêta en fonction de la moyenne empirique et de la variance empirique de x suivent les équations:

$\alpha = \bar{x}\left(\frac{\bar{x}(1-\bar{x})}{v} - 1\right)$ et $\beta = (1-\bar{x})\left(\frac{\bar{x}(1-\bar{x})}{v} - 1\right)$

```
mean_x = mean(x_norm)      # Moyenne empirique de x
var_x = var(x_norm)        # Variance empirique de x
alpha = mean_x * (mean_x * (1-mean_x)/ var_x - 1)    # Calcul de alpha
beta = (1 - mean_x) * ( mean_x * (1-mean_x)/ var_x - 1) # calcul de bêta
cat("alpha =", alpha, "\n")
```

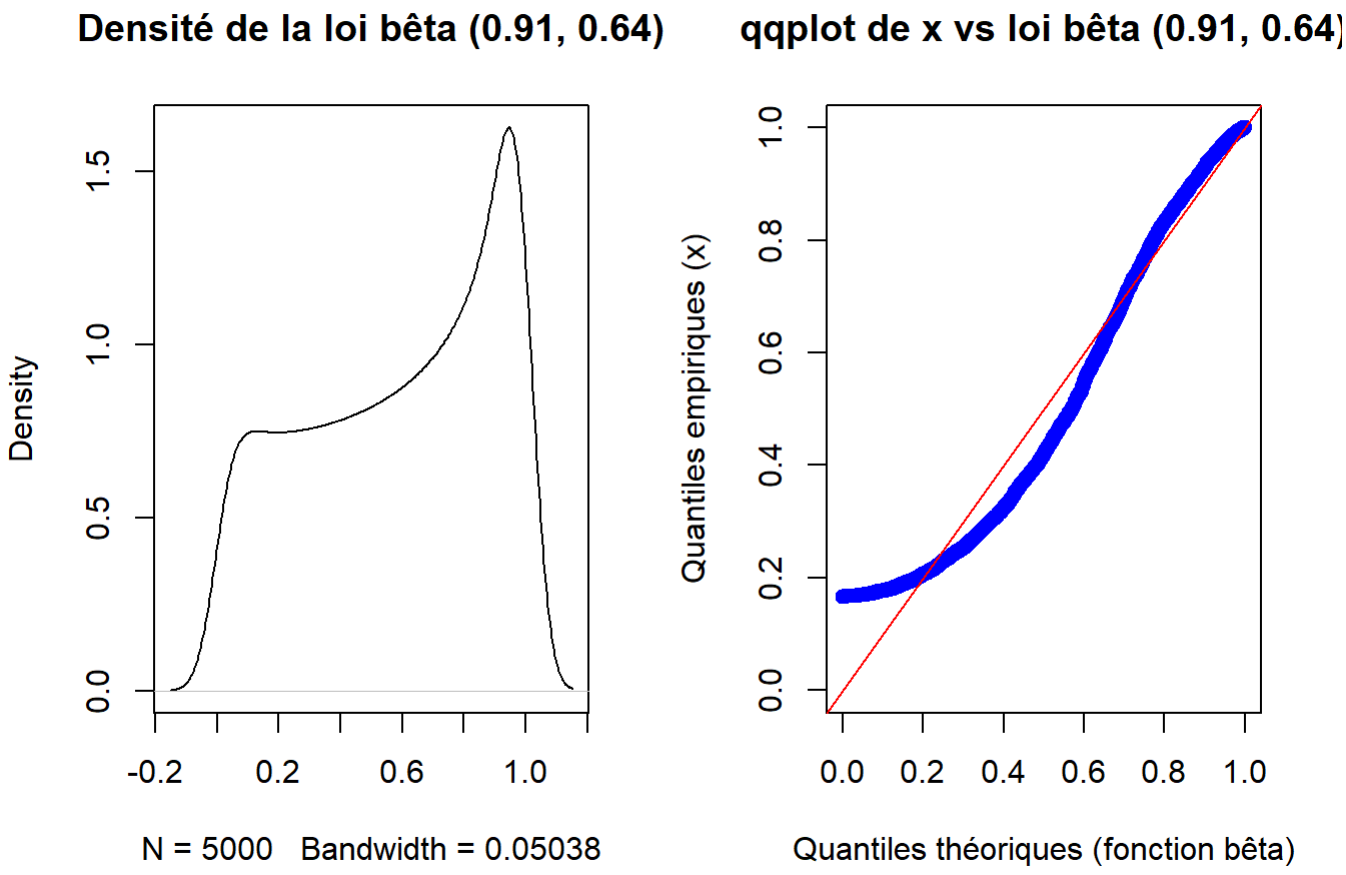
```
## alpha = 0.9199037
```

```
cat("beta =", beta, "\n")
```

```
## beta = 0.6403525
```

Comparons les deux distributions avec ces nouveaux paramètres:

```
par(mfrow=c(1,2))
q_beta <- qbeta (ppoints(5000), 0.9199037,0.6403525) # Calcule les quantiles théoriques de la distribution bêta
plot (density(q_beta),main = "Densité de la loi bêta (0.91, 0.64)") # Graphique de la densité de la loi bêta (0.5, 0.5)
x_norm = x / 6                          # Normalise x pour prendre des valeurs dans [0,1]
qqplot(q_beta, x_norm, col = "blue",    # QQ plot x vs loi bêta (0.5, 0.5)
       xlab = "Quantiles théoriques (fonction bêta)",
       ylab = "Quantiles empiriques (x)",
       main = "qqplot de x vs loi bêta (0.91, 0.64)",
       ylim = c(0, max(x/6)))  # Définit les limites de l'axe y
abline(0, 1, col = "red") # Ajoute une ligne de référence pour montrer une relation linéaire entre les quantiles
```




```
ks.test(x/6, q_beta)  # Test de Kolmogorov-Smirnov x vs loi bêta (0.9199037,0.6403525)
```

```
##
##  Asymptotic two-sample Kolmogorov-Smirnov test
##
## data:  x/6 and q_beta
## D = 0.1298, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

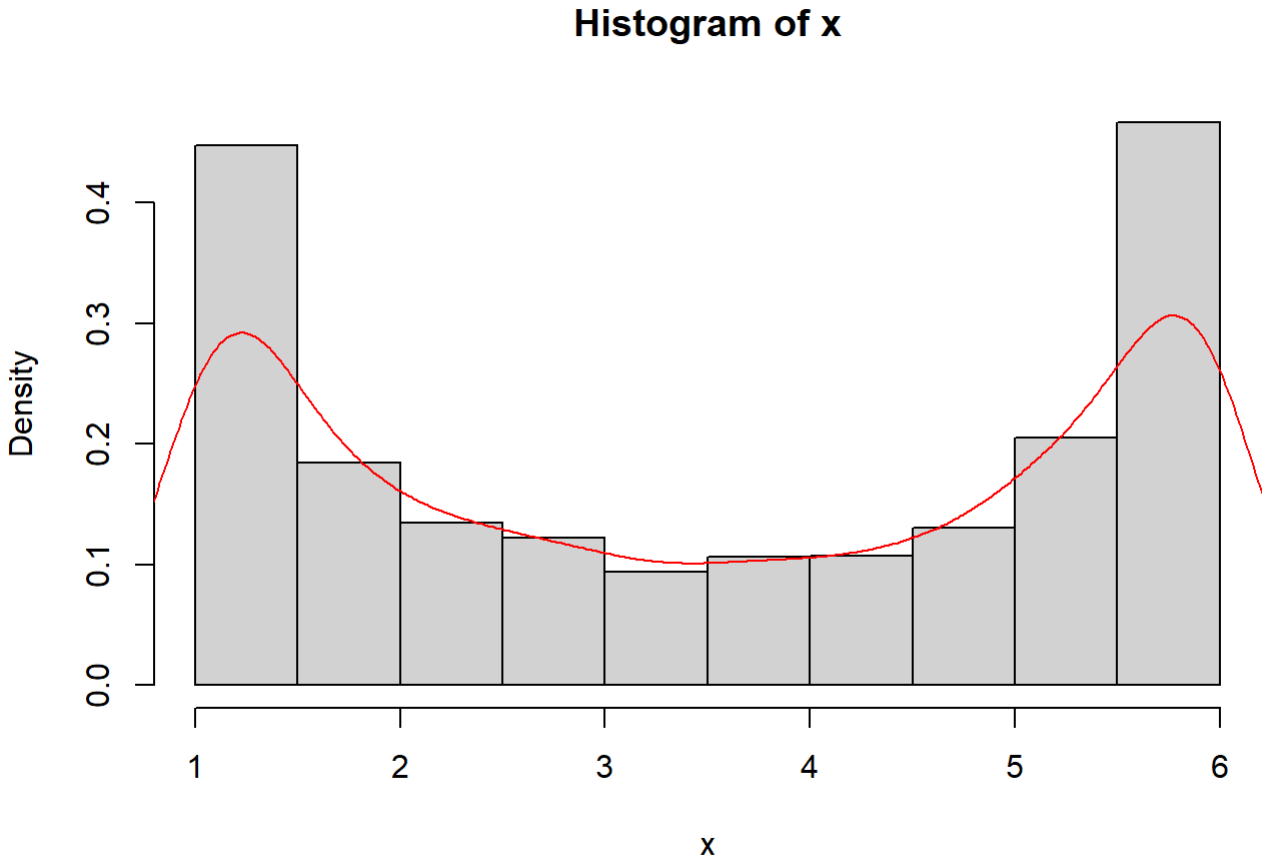
On s'approche un peu mieux. D est maintenant égal à 0.1298 contre 0.2678 avec la loi bêta (0.5, 0.5), mais le test rejette encore notre hypothèse de similarité entre x et la loi bêta. D'ailleurs le graphique de la loi béta avec ces nouveaux paramètres montrait bien les limites de cette méthode (la forme générale s'éloignait de notre densité à estimer). Nous concluons que la variable x ne suit pas de loi connue pour la suite de l'étude. Étudions donc la densité par des méthodes non-paramétriques.

# Estimation de la densité de x par des méthodes non-paramétriques

## Estimateurs à noyau

En première approche, nous utilisons la fonction "density()" pour afficher la densité de x:

```
hist (x, freq=F)    # Superposition de la densité de x par méthode des noyaux et de son histo
gramme
lines (density(x, kernel = "gaussian"), col= "red")
```



Cette représentation confirme au passage que l'histogramme avec 10 classes donne une assez bonne représentation de la densité. Nous savons que le choix du noyau n'est pas d'une importance capitale. Notons cependant que le noyau Epanechnikov obtient la meilleure valeur d'optimisation.

La valeur optimale de h est obtenue par dérivation du IMSE (Integrated Mean Squared Error) ou MISE:

$$h_{\text{opt}} = \Big(\frac{R(K)}{n\mu_2(K)^2 R(f''(x))}\Big)^{1/5}$$

l'erreur correspondante est:

$$\text{IMSE}_{\text{opt}} = \frac{5}{4}\ \frac{1}{n^{4/5}}\ \Big(R(K)^2\mu_2(K)\Big)^{2/5}\ R(f''(x))^{1/5}$$

où $R(K) = \int_{-\infty}^{\infty} K(x)^2\,dx$ (appelé "roughness du noyau") et $R(f''(x)) = \int f''(x)^2\,dx$.

On doit donc choisir un noyau qui minimise $\big(R(K)^2\mu_2(K)\big)$ (le reste de l'équation ne dépend pas du choix du noyau).

Le calcul donne:

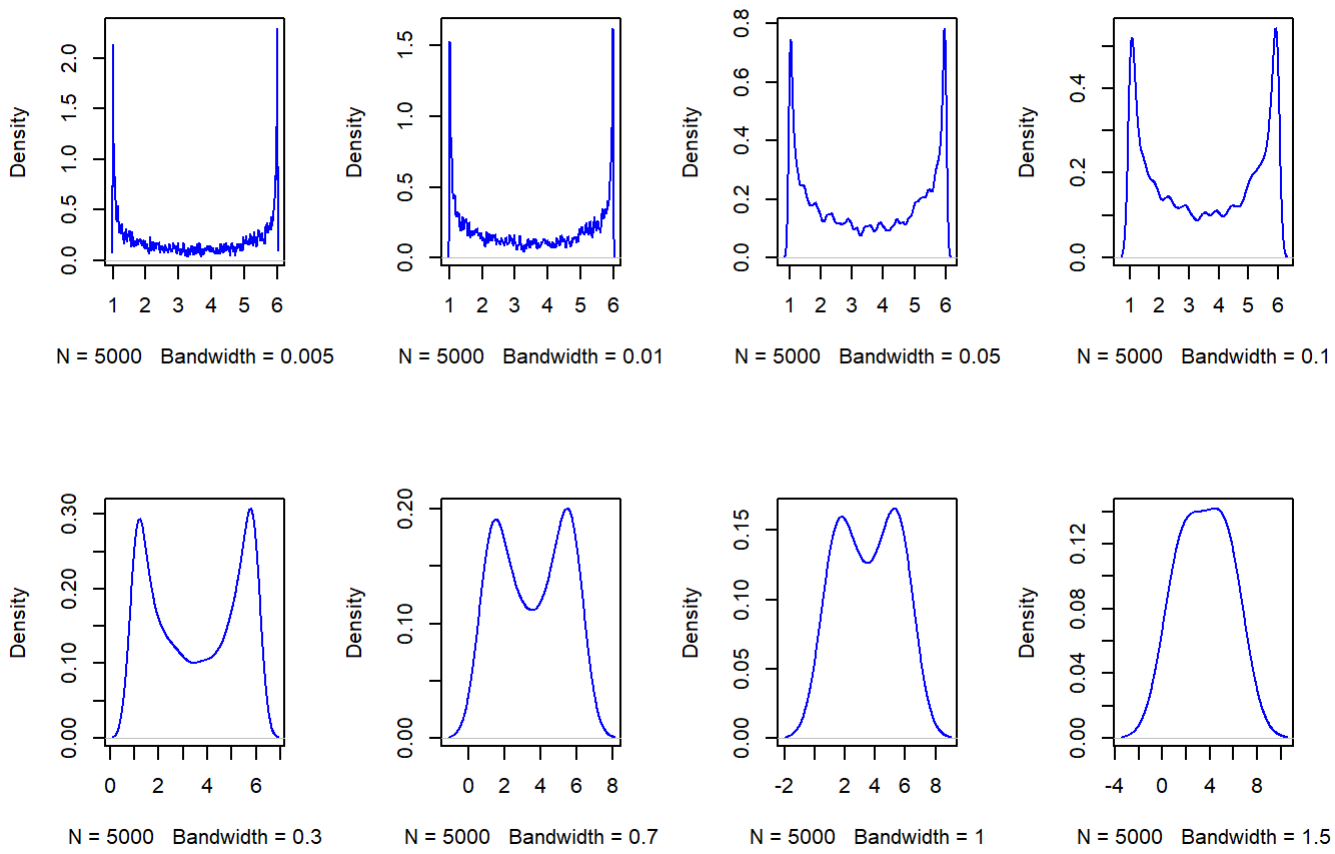
```
##          noyau  Valeur
## 1      Uniform 0.08333
## 2 Triangulaire 0.07407
## 3 Epanechnikov 0.07200
## 4     Gaussien 0.07958
```

Les différences étant très faibles, nous retenons le noyau gaussien pour le reste de notre étude.

# Choix de la fenêtre optimale

Avant de discuter des méthodes d'optimisation de h, essayons naïvement différentes valeurs:

```
par(mfrow=c(2,4))
plot (density(x, bw = 0.005, kernel = "gaussian"), col= "blue",main = "")
plot (density(x, bw = 0.01, kernel = "gaussian"), col= "blue",main = "")
plot (density(x, bw = 0.05, kernel = "gaussian"), col= "blue",main = "")
plot (density(x, bw = 0.1, kernel = "gaussian"), col= "blue",main = "")
plot (density(x, bw = 0.3, kernel = "gaussian"), col= "blue",main = "")
plot (density(x, bw = 0.7, kernel = "gaussian"), col= "blue",main = "")
plot (density(x, bw = 1, kernel = "gaussian"), col= "blue",main = "")
plot (density(x, bw = 1.5, kernel = "gaussian"), col= "blue",main = "")
```




Ces 8 graphiques montrent bien comment évolue la représentation de la densité en fonction de la largeur de la fenêtre h. De manière similaire au cas de l'histogramme on doit arbitrer entre une variance trop importante (surajustement, h trop petit) et un biais trop important (sous-ajustement, h trop grand).

Il existe de nombreuses manières de calculer h. Les méthodes de type "Rule of Thumb" sont faciles et économiques en terme calculatoire: citons par exemple celle proposée par David W. Scott (**"nrd"** dans R): $h \approx 1.06 \cdot \hat{\sigma} n^{-1/5}$ où $\hat{\sigma}$ est l'écart-type de l'échantillon. Cela donne pour nos valeurs de x: $h \approx 0.35$.

Limite: fonctionne bien si la distribution s'approche d'une distribution normale.

Une autre "Rule of thumb" célèbre est celle proposée par Silverman (**"nrd0"** dans R): $h = 0.9 \cdot \min(\hat{\sigma}, IQR/1.35) n^{-1/5}$ où IQR (interquartile range) estime la différence entre les quantiles 0.75 et 0.25 de l'échantillon.

Limite: quoique plus robuste que celle proposée par Scott, elle obtient également ses meilleurs résultats quand la distribution ne s'éloigne pas trop d'une distribution normale.

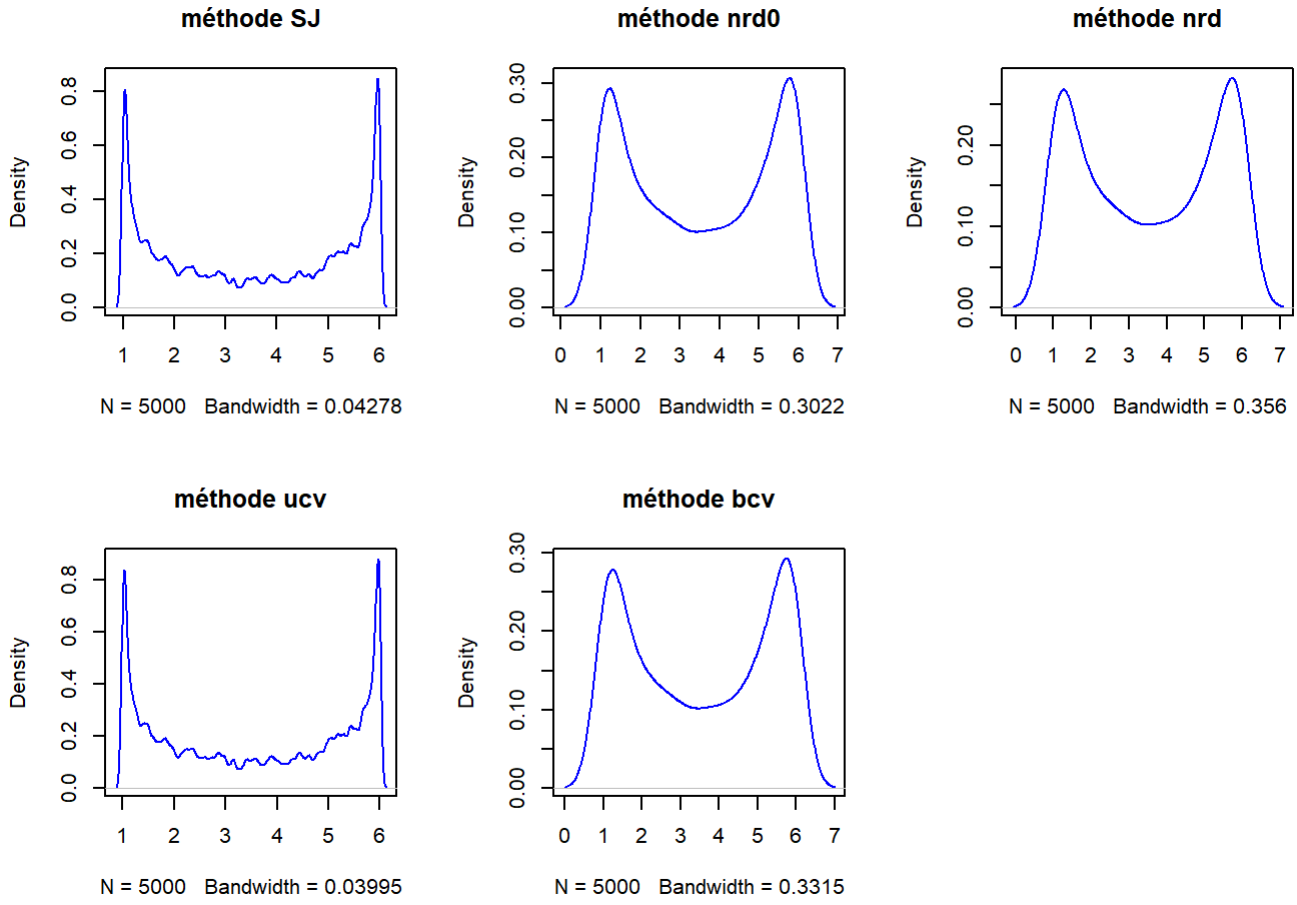
Ces méthodes sont donc à éviter pour des distributions trop compliquées.

R propose plusieurs méthodes d'estimation de la fenêtre:

```
data.frame(name = c("SJ", "nrd0", "nrd", "ucv", "bcv"),
  bw = c(bw.SJ(x), bw.nrd0(x), bw.nrd(x), bw.ucv(x), bw.bcv(x)))
```

```
##   name         bw
## 1   SJ 0.04277608
## 2 nrd0 0.30223369
## 3  nrd 0.35596412
## 4  ucv 0.03994789
## 5  bcv 0.33151273
```

```
par(mfrow=c(2,3))
plot (density(x, bw = "SJ", kernel = "gaussian"), col= "blue", main="méthode SJ")
plot (density(x, bw = "nrd0", kernel = "gaussian"), col= "blue", main="méthode nrd0")
plot (density(x, bw = "nrd", kernel = "gaussian"), col= "blue", main="méthode nrd")
plot (density(x, bw = "ucv", kernel = "gaussian"), col= "blue", main="méthode ucv")
plot (density(x, bw = "bcv", kernel = "gaussian"), col= "blue", main="méthode bcv")
```




La méthode de Sheather & Jones (SJ) est une évolution de la méthode de Park et Marron (toutes deux de type “plug-in”).

La méthode “Biased Cross-Validation” (**bcv** dans R) proposée par Scott and Terrell (1987), cherche à minimiser l’AMISE (Asymptotic Mean Integrated Squared Error). La méthode “Unbiased Cross-Validation” (**ucv** dans R) utilise la procédure de validation croisée des moindres carrés.

La littérature est très vaste quant aux différentes méthodes d’optimisation de la largeur de la fenêtre h. Dans sa thèse présentée en 2011 (**Bandwidth Selection in Nonparametric Kernel Estimation**) Anja Schindler compte plus de 30 méthodes différentes dont beaucoup sont des adaptations de méthodes existantes pour répondre à des situations particulières.

En général, dans la littérature sont distinguées deux grandes familles de méthodes: “cross-validation” (CV) et “plug-in” (PI). Il faut prendre cette nomenclature avec précaution, les méthodes dites “plug-in” pouvant intégrer une étape de calcul utilisant la “cross-validation”. Cependant, pour les distinguer, nous pouvons noter que la famille CV s’attache à minimiser l’ISE (Integrated Squared Error), tandis que la famille PI cherche à mminimiser le MISE (en substituant les paramètres inconnus par ceux d’une loi connue, d’où le terme “plug-in”).

Les méthodes CV sont antérieures (dès 1982 avec Rudemo et 1984 avec Bowman) aux méthodes PI (en 1986 avec Silvermann). Beaucoup d’évolutions ont eu lieu dans chacune de ces deux familles.

Pour résumer leurs différences, les méthodes PI sont réputées avoir une meilleure vitesse de convergence mais fonctionnent mal avec de petits échantillons alors que les méthodes CV tendent au surajustement sur les grands échantillons (en privilégiant des petits h).

# Méthode de Cross-Validation pour déterminer le h optimal

Nous utilisons la méthode “Leave One Out” ou LOO (cas particulier de “cross validation”) pour déterminer la valeur optimale de h.

Note: La méthode de validation croisée consiste à diviser les données en ensembles d’entraînement et de validation. Elle permet d’utiliser efficacement toutes les données disponibles pour l’entraînement et l’évaluation du modèle. La méthode “Leave One Out” est en quelque sorte la version “maximale” de validation croisée. Elle consiste à diviser les données en ensembles d’entraînement et de validation de manière à ce **qu’un seul** échantillon soit utilisé comme ensemble de validation à chaque fois (par itération). C’est donc la méthode de CV la plus efficace mais aussi forcément la plus coûteuse en calculs. Nous nous permettons ici d’y avoir recours, la taille de l’échantillon proposé étant raisonnable.

Nous cherchons à minimiser:

$$\mathrm{IMSE}\big(\hat{f}_h\big) = \int_{-\infty}^{\infty} \mathrm{MSE}\big(\hat{f}_h(x)\big)\, dx$$

En développant l’expression nous pouvons montrer que h doit minimiser:

$$\mathrm{CV}(h) = \int \hat{f}_h(x)^2 dx - \frac{2}{n}\sum_{i=1}^{n} \hat{f}_h^{(i)}(x_i) = A - B$$

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

Pour la partie A:

pour calculer l’intégrale de $\hat{f}_h^2$, on évalue $\hat{f}_h$ sur une grille de x valeurs $(\tilde{x}_1, \ldots, \tilde{x}_m)$.

Nous utilisons : $\int \hat{f}_h(x)^2\, dx \approx \sum_{j=1}^{m} \hat{f}_h(\tilde{x}_j)^2\, \Delta\tilde{x}$

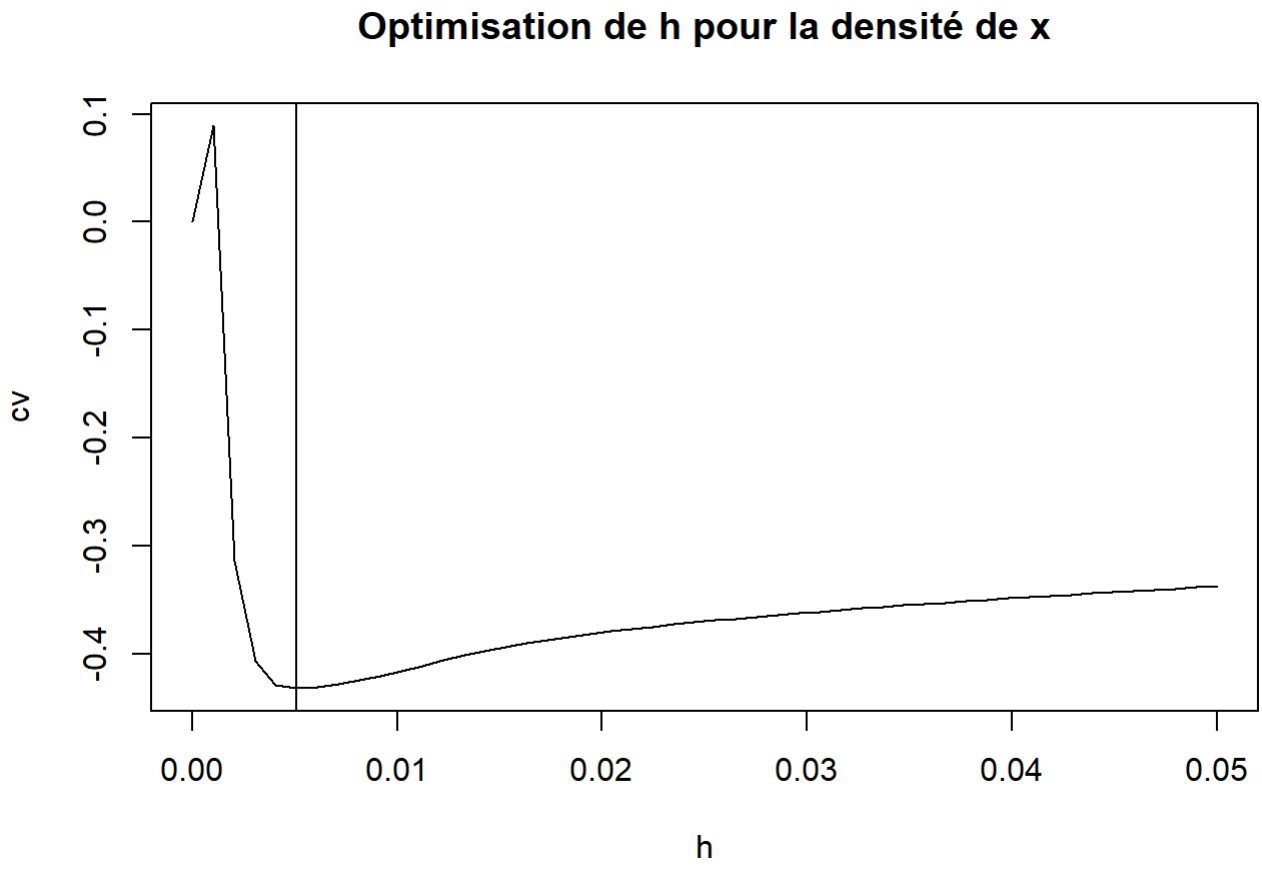
Nous calculons toutes les différences $\tilde{x}_j - x_i$ en utilisant la fonction “outer()”, la sortie est sous forme d’une matrice (n x m). Pour calculer le noyau gaussien, nous utilisons la fonction “dnorm()” avec h comme écart-type. La moyenne de chaque colonne par la fonction “colMeans()” permet de calculer $\hat{f}_h$. Les $\hat{f}_h(\tilde{x}_j)$ sont multipliés par la distance entre 2 points de la grille et nous sommons.

Pour la partie B:

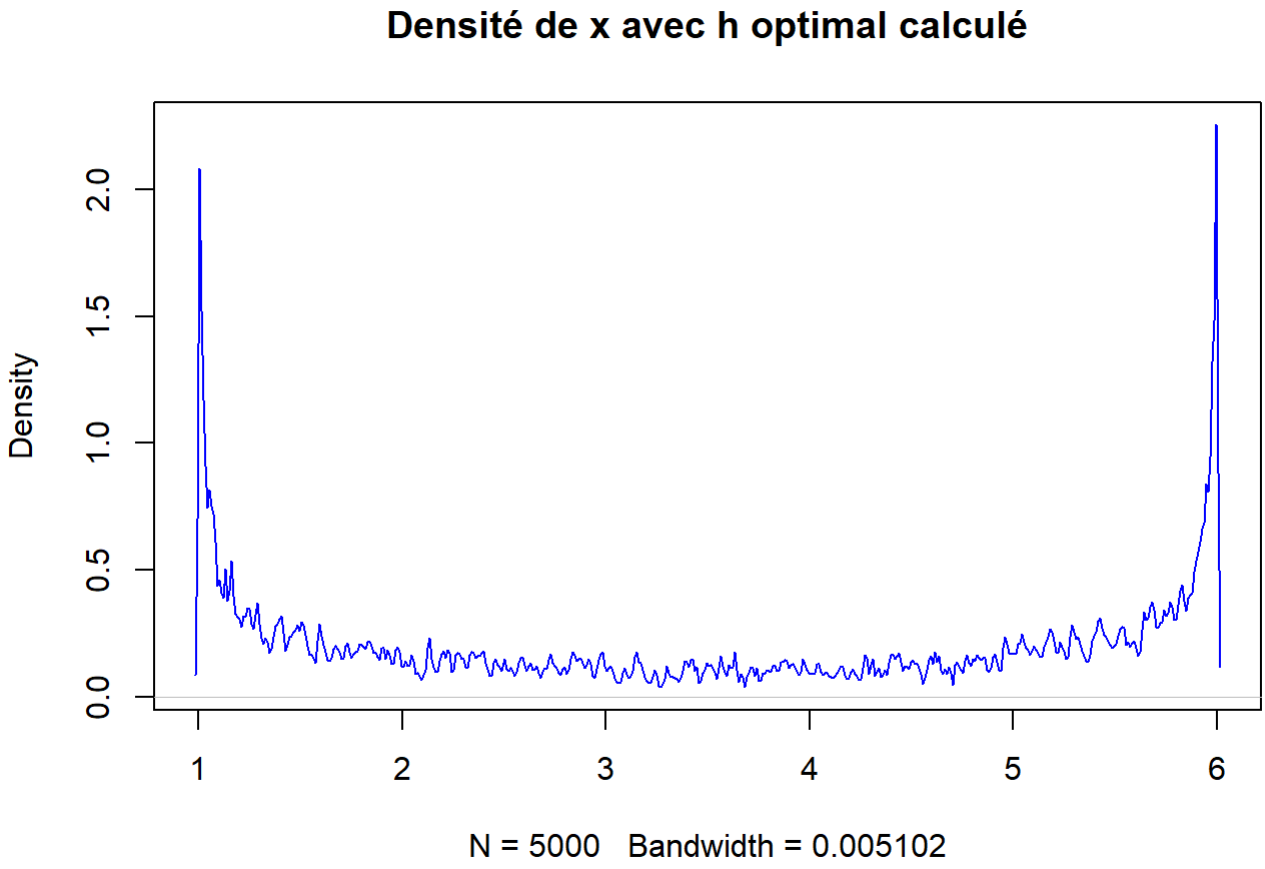
$\sum_{j=1}^{n} \hat{f}_h^{(j)}(x_j) = \sum_{j=1}^{n} \frac{1}{n-1}\sum_{i\neq j} K_h(x_j - x_i) = \frac{1}{n-1}\sum_{\substack{i,j=1 \\ i\neq j}}^{n} K_h(x_j - x_i)$

Nous utilisons la fonction “outer()” qui permet de paralléliser le calcul et implémentons la condition $i \neq j$ en mettant les éléments correspondant à $i = j$ à zéro dans la matrice K.

```r
h_LOO <- function(h) {
## Partie A
  x.min <- 0
  x.max <- 6
  n <- length(x)
  x.tilde <- seq(x.min, x.max, length.out = 200)     # grille d'évaluation de 200 valeurs
  step <- x.tilde [2] - x.tilde [1]          # largeur du rectangle sur l'axe des x
  K <- dnorm(outer(x, x.tilde, "-"), sd = h)     # Crée la matrice ("dnorm()" car noyau gauss
ien)
  f.hat <- colMeans(K)              # Moyenne des colonnes de la matrice K
  A <- sum(f.hat^2 * step)   # Intégration
## Partie B
  K <- dnorm(outer(x, x, "-"), sd = h)         # Crée la matrice ("dnorm" car noyau gaussien)
  diag(K) <- 0                      # Diagonale de la matrice K à 0 pour LLO
  f.hat <- colSums (K) / (n-1)
  B <- 2 * sum(f.hat) / n

  return(A - B)
}
h <- seq(0,0.05, length.out = 50)   # Grille de h
cv <- numeric(length(h))
for (i in seq_along(h)) {
cv[i] <- h_LOO(h[i])
}
plot(h, cv, type = "l", main="Optimisation de h pour la densité de x")
best.h <- h[which.min(cv)]
abline(v = best.h)
```



```
plot (density(x, bw = best.h, kernel = "gaussian"), col= "blue", main="Densité de x avec h optimal calculé")
```

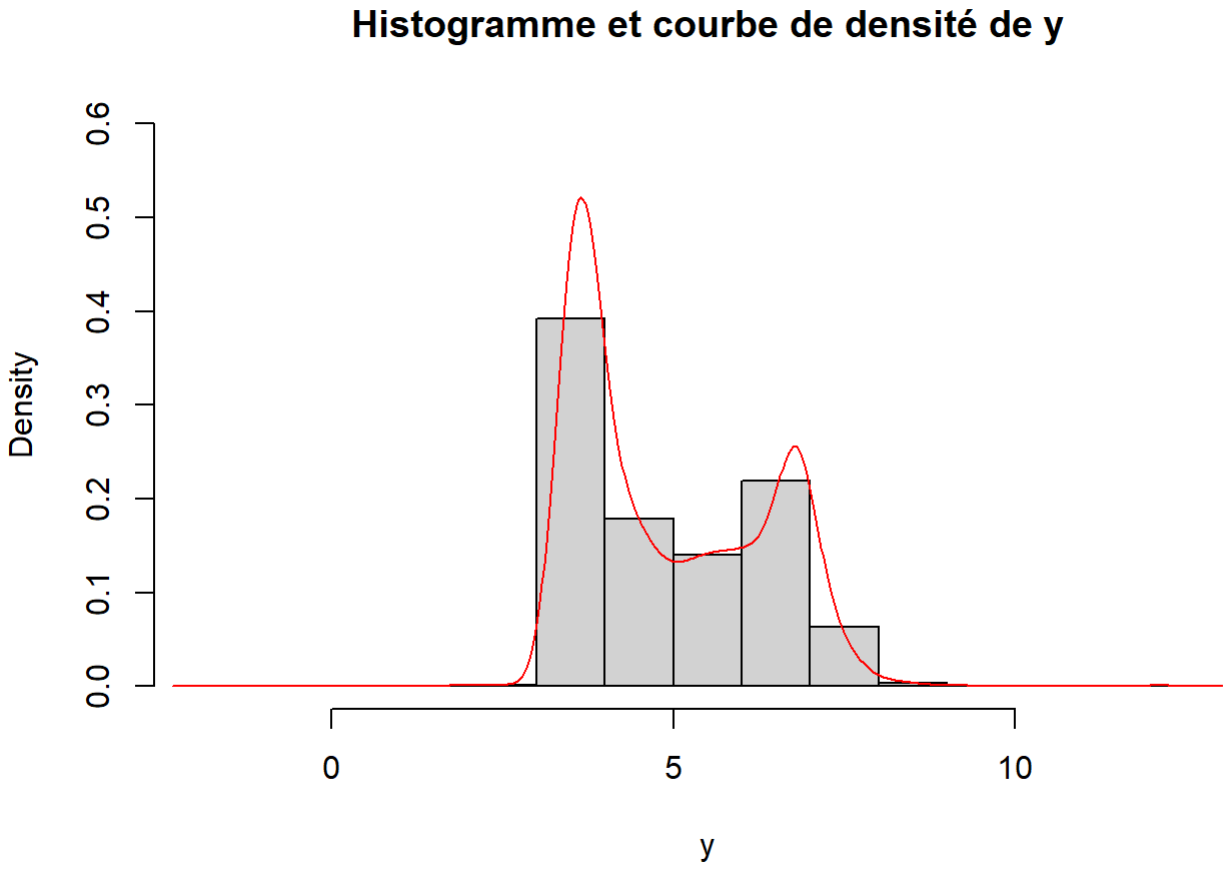



La valeur calculée par notre méthode est plus petite que celles calculées par les fonctions de R. Elle est plus “exhaustive” et cherche à ne pas rater des valeurs de x, mais tend vers le surajustement.

# Regression non-paramétrique

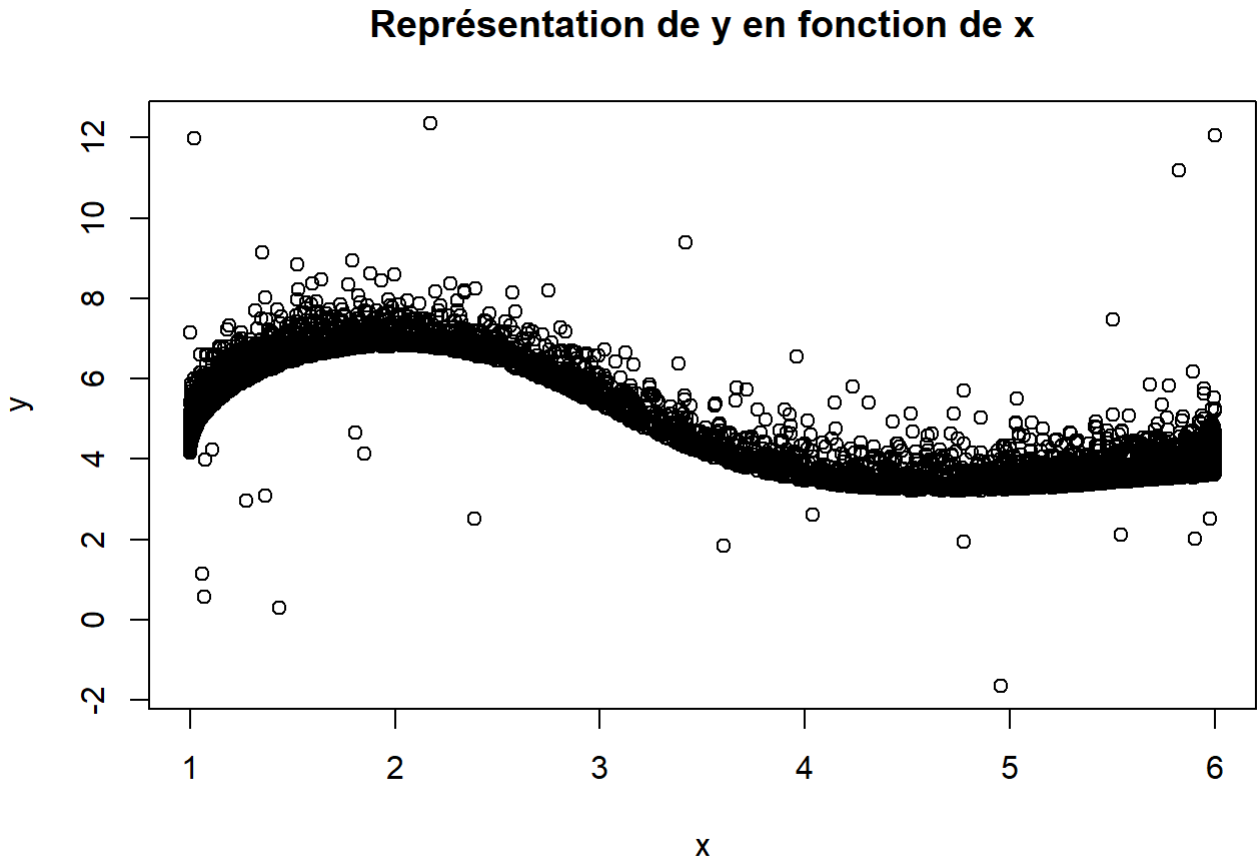
Avant de proposer une régression, intéressons-nous à la variable à expliquer y.

```
hist (y, freq=F, main = "Histogramme et courbe de densité de y", ylim= c(0,0.6)  )
lines (density(y, kernel = "gaussian"), col= "red")
```




Où nous voyons que y peut éventuellement ressembler à un mélange de deux lois gaussiennes.

```
plot (x, y,main = "Représentation de y en fonction de x")
```



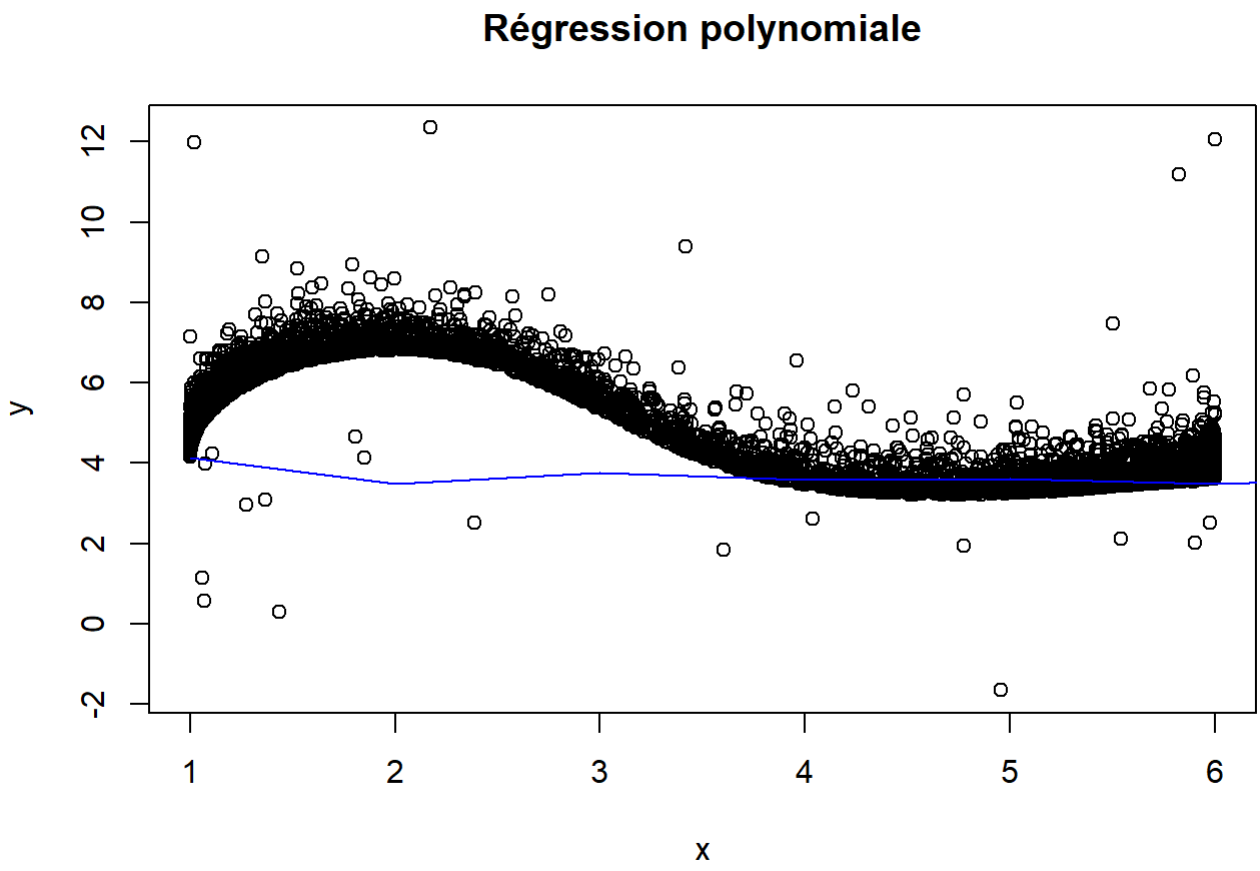

Relation non linéaire entre les deux variables, même si pour des valeurs de x entre 4 et 6, la relation s'approche de la linéarité.

# Régression polynomiale

Idée: La relation entre x et y n'étant pas linéaire, nous pouvons penser à une régression polynomiale, le modèle s'écrivant ainsi (pour un polynôme de degré p):

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_p x^p + \varepsilon$$

```
plot (x, y, main = "Régression polynomiale")
m <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4))  # Régression polynomiale
lines(fitted(m), col = "blue")
```




La régression polynomiale de donne pas de résultats probants. (Sans le représenter ici, nous avons essayé d'autres degrés de polynômes et le résultat n'est pas meilleur.)

Essayons autre chose.

# Estimateur de Nadaraya-Watson

L'estimateur de Nadaraya-Watson est donné par :

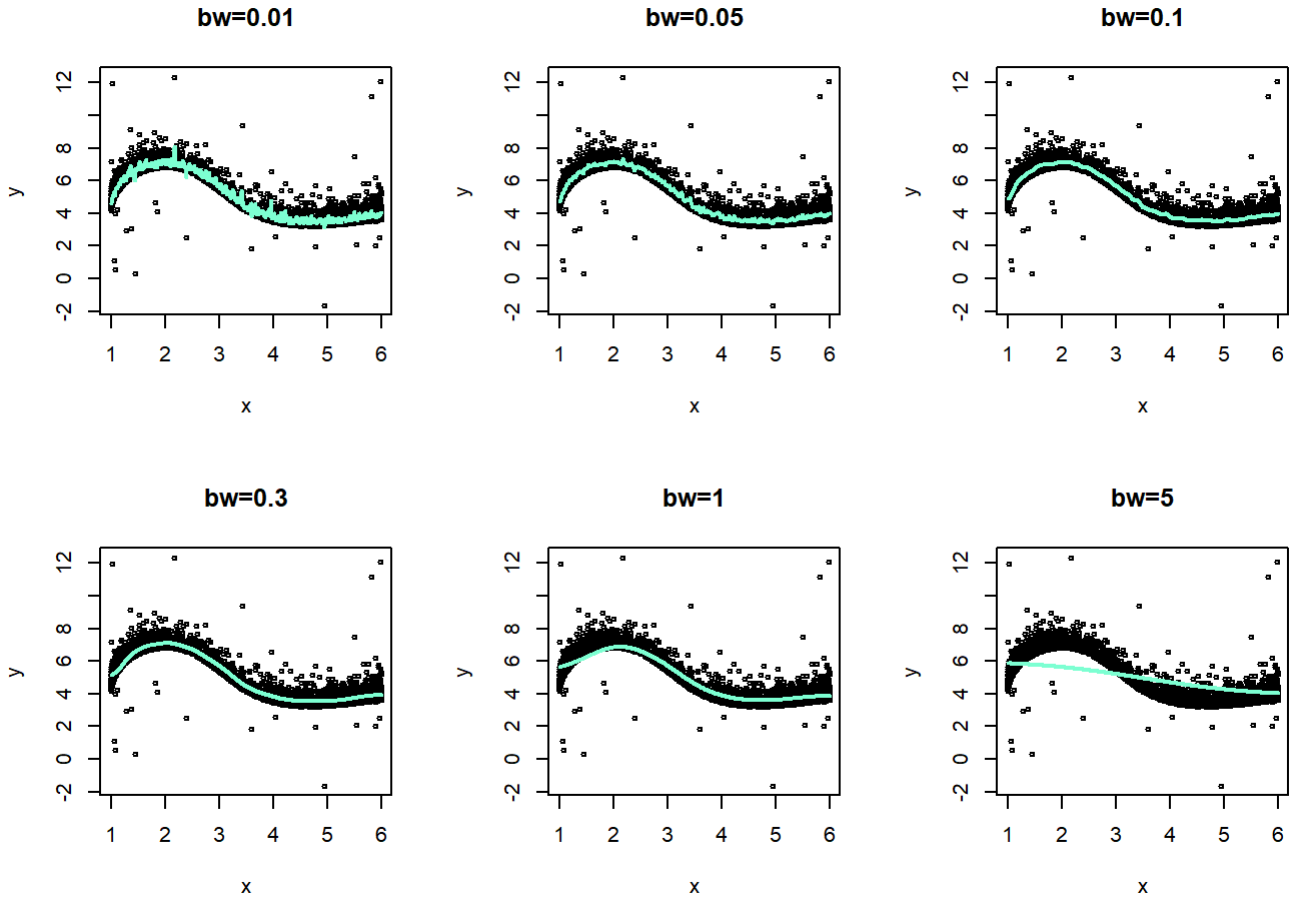
$$\hat{m}_h(x) = \frac{\sum_{i=1}^{n} K_h(x - x_i) y_i}{\sum_{j=1}^{n} K_h(x - x_j)}$$

où $K_h$ est le noyau "dimensionné" par la largeur de la fenêtre h.

De manière similaire avec l'étude vue plus haut sur la densité de x, le problème se réduit au choix de la largeur de la fenêtre h.

Essayons plusieurs valeurs de h:

```
par(mfrow=c(2,3))
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="bw=0.01")
lines(ksmooth(x, y, kernel = "normal", bandwidth = 0.01, n.points = 1000), col ="aquamarine",
lwd = 2)
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="bw=0.05")
lines(ksmooth(x, y, kernel = "normal", bandwidth = 0.03, n.points = 1000), col ="aquamarine",
lwd = 2)
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="bw=0.1")
lines(ksmooth(x, y, kernel = "normal", bandwidth = 0.1, n.points = 1000), col ="aquamarine",
lwd = 2)
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="bw=0.3")
lines(ksmooth(x, y, kernel = "normal", bandwidth = 0.3, n.points = 1000), col ="aquamarine",
lwd = 2)
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="bw=1")
lines(ksmooth(x, y, kernel = "normal", bandwidth = 1, n.points = 1000), col ="aquamarine", lw
d = 2)
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="bw=5")
lines(ksmooth(x, y, kernel = "normal", bandwidth = 5, n.points = 1000), col ="aquamarine", lw
d = 2)
```




Toujours ce compromis biais/variance !

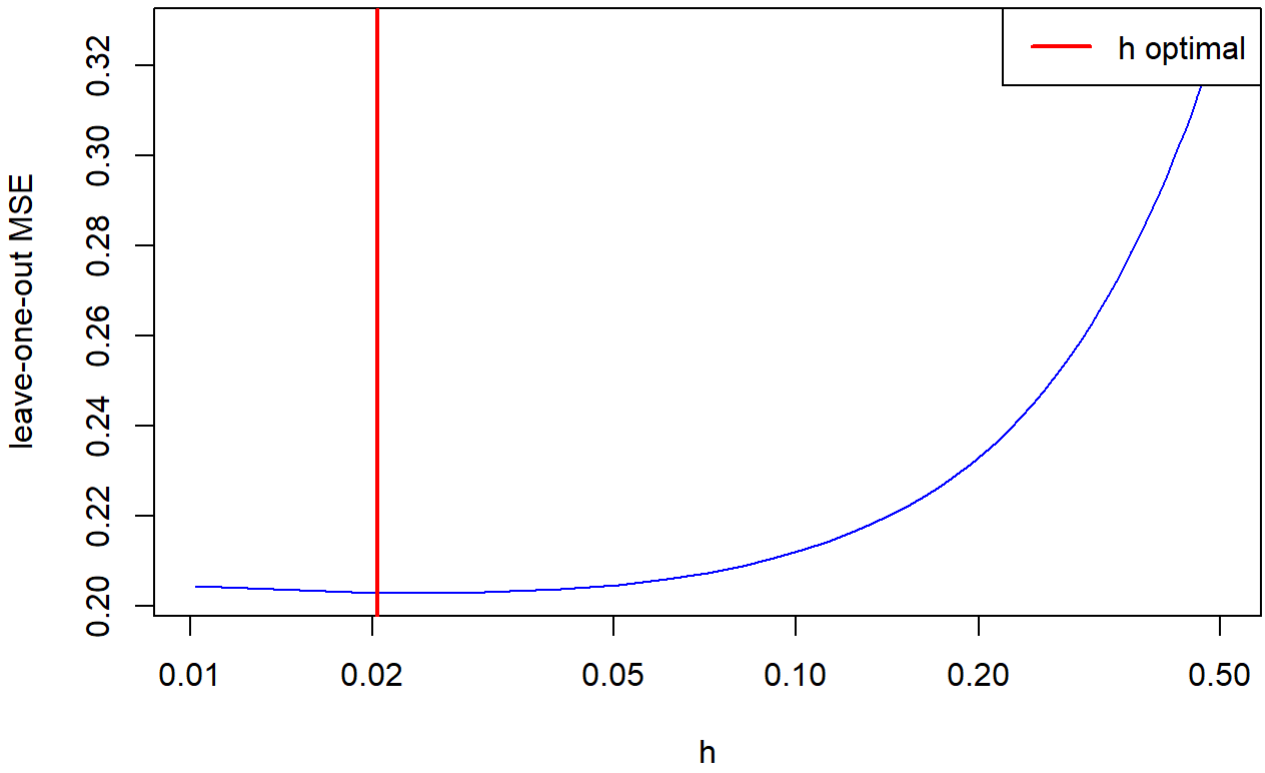
Apparemment, un bon compromis pour les valeurs de h devrait se trouver entre 0.01 et 0.1. Utilisons à nouveau la méthode “Leave One Out” pour déterminer la valeur optimale de h.

Nous cherchons à minimiser $r_{\mathrm{LOO}}(h) = \sum_{i=1}^{n} \left(y_i - \hat{m}_h^{(i)}(x_i)\right)^2$

où $m^{(i)}$ est l’estimateur à noyau calculé sans la valeur d’indice i selon:

$$\hat{m}_h^{(i)}(x_i) = \frac{\sum_{j=1, j\neq i}^{n} K_h(x_i - x_j) y_j}{\sum_{j=1, j\neq i}^{n} K_h(x_i - x_j)}$$

```
NW <- function(h) {
  K <- dnorm(outer(x, x, "-"), sd = h)  # Créer la matrice pour le noyau gaussien

  diag(K) <- 0    # Diagonale de la matrice K à 0 pour leave-One-Out
  m.hat <- colSums(K*y) / colSums(K)
  mean((m.hat - y)^2)
}

h <- seq(0,0.5, length.out = 50)     # Grille d'analyse de h de 0 à 0.5 sur 50 valeurs
mse <- numeric(length(h))
for (i in seq_along(h)) {
  mse[i] <- NW(h[i])
}

best_h <- h[which.min(mse)]

plot(h, mse, log="x", type = "l", ylab = "leave-one-out MSE", col = "blue")
abline(v = best_h, col = "red",lwd = 2)
legend("topright", legend = "h optimal", col = "red",lwd = 2)
```




```
cat("h optimal =", best_h)
```

```
## h optimal = 0.02040816
```

Où nous voyons que le h optimal calculé selon cette méthode est d'environ 0.02 . Nous pouvons remarquer aussi que les valeurs du MSE varient peu pour des valeurs de h allant de 0.01 à 0.05. Ceci correspond à nos observations faites plus haut.

# Estimateurs par polynomes locaux

Cette méthode est une généralisation de l'estimateur de Nadaraya-Watson (qui correspond au cas où l'ordre p est égal à zéro). L'idée est de faire une régression polynomiale en attribuant des poids aux observations en fonction de la distance aux x d'évaluation.

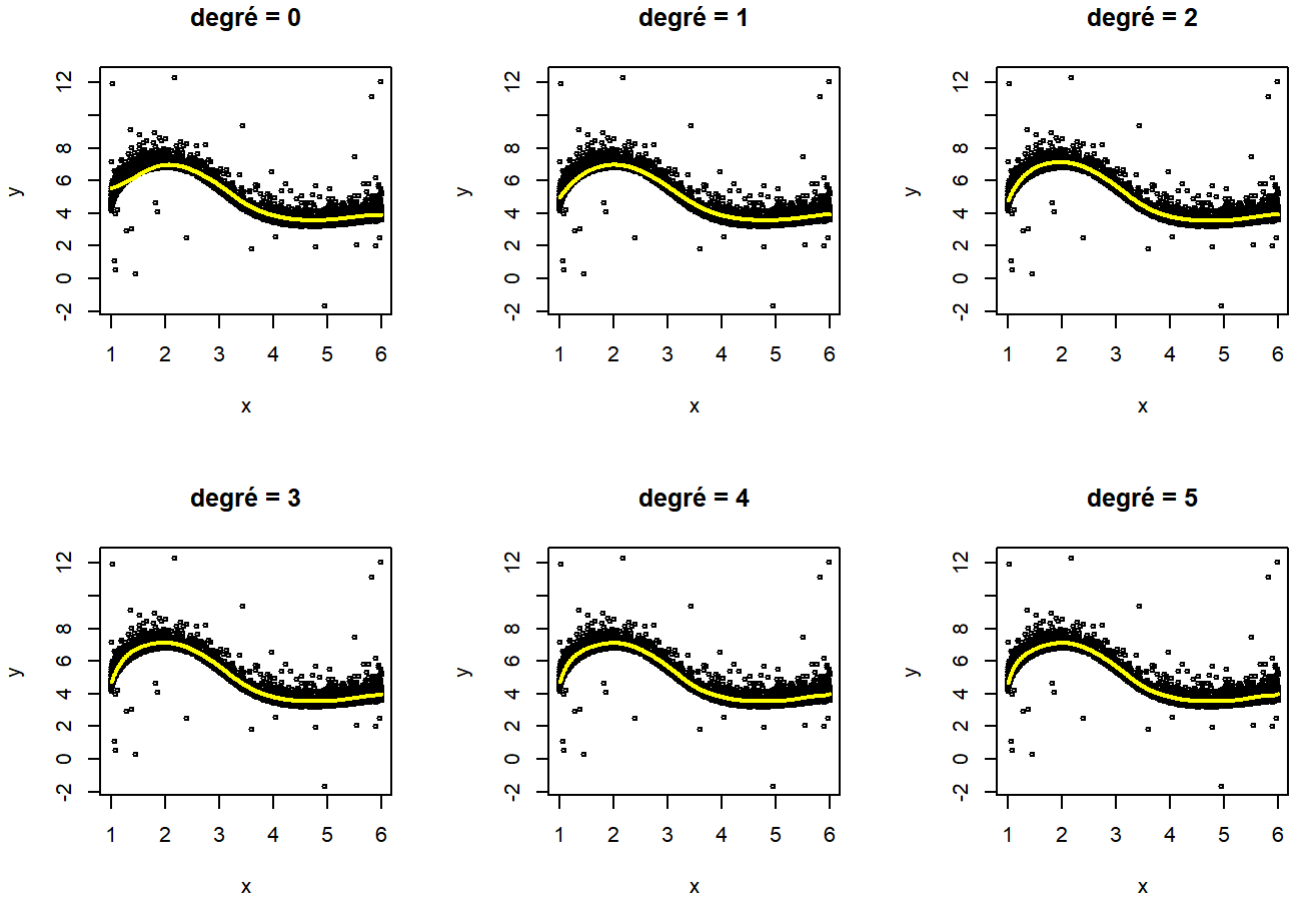
Les coefficients de la régression sont estimés par : $\hat{\beta} = (X^\top W X)^{-1} X^\top W y$

où **W** est la matrice diagonale des poids.

Le modèle s'écrit pour les valeurs proches de $\tilde{x}$ :
$\hat{m}_h(x; \tilde{x}) = \hat{\beta}_0 + \hat{\beta}_1(x - \tilde{x}) + \hat{\beta}_2(x - \tilde{x})^2 + \cdots + \hat{\beta}_p(x - \tilde{x})^p$

Nous pouvons utiliser la fonction "locpoly" du package KernSmooth:

```
par(mfrow=c(2,3))
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="degré = 0")
lc_0 <- locpoly(x, y, degree = 0, bandwidth = 0.3)
lines(lc_0, col = "yellow",lwd = 2)
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="degré = 1")
lc_1 <- locpoly(x, y, degree = 1, bandwidth = 0.3)
lines(lc_1, col = "yellow",lwd = 2)
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="degré = 2")
lc_2 <- locpoly(x, y, degree = 2, bandwidth = 0.3)
lines(lc_2, col = "yellow",lwd = 2)
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="degré = 3")
lc_3 <- locpoly(x, y, degree = 3, bandwidth = 0.3)
lines(lc_3, col = "yellow",lwd = 2)
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="degré = 4")
lc_4 <- locpoly(x, y, degree = 4, bandwidth = 0.3)
lines(lc_4, col = "yellow",lwd = 2)
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="degré = 5")
lc_5 <- locpoly(x, y, degree = 5, bandwidth = 0.3)
lines(lc_5, col = "yellow",lwd = 2)
```




Le résultat est probant ! On observe qu'à partir de l'ordre 2, il n'y a plus vraiment d'évolution de la qualité de notre régression. Ceci sans doute parce que la régression ressemble à un polynôme simple. En général, pour un h donné, choisir un p plus grand diminue le biais mais augmente la variance.

Notons qu'au degré zéro, la courbe ressemble bien à celle obtenue par l'estimateur de Nadaraya-Watson (à comparer à celle obtenue plus haut).

Se pose ici la même question que précédemment: quelle valeur de h choisir ?

Le modèle sous forme matricielle s'écrit: $\hat{m}_h(x) = e_0^\top (X^\top W X)^{-1} X^\top W y$

où :

$$X = \begin{pmatrix} 1 & (x_1 - \tilde{x}) & (x_1 - \tilde{x})^2 & \cdots & (x_1 - \tilde{x})^p \\ 1 & (x_2 - \tilde{x}) & (x_2 - \tilde{x})^2 & \cdots & (x_2 - \tilde{x})^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (x_n - \tilde{x}) & (x_n - \tilde{x})^2 & \cdots & (x_n - \tilde{x})^p \end{pmatrix}$$

$e_0 = (1, 0, \ldots, 0)$ et **W** représente la matrice diagonale des poids (noyau gaussien ici).

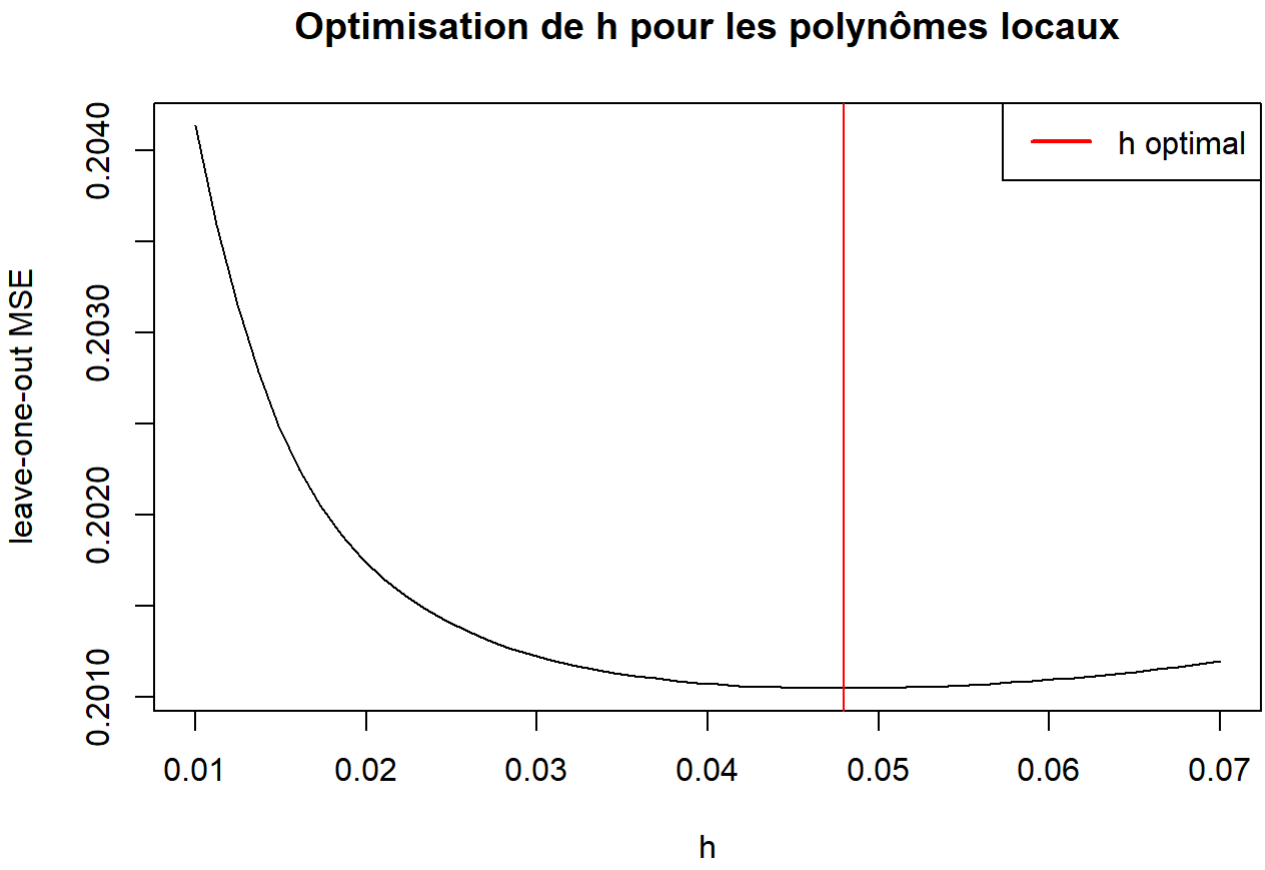
Nous calculons :

$T_1 = \sum_{j=1}^n K_h(x - x_j) y_j$, $T_2 = \sum_{j=1}^n K_h(x - x_j) x_j (x_j - x)$, $T_3 = \sum_{j=1}^n K_h(x - x_j) x_j y_j$, $T_4 = \sum_{j=1}^n K_h(x - x_j)(x_j - x)$

et

$B_1 = \sum_{j=1}^n K_h(x - x_j)$, $B_2 = \sum_{j=1}^n K_h(x - x_j) x_j^2$, $B_3 = \sum_{j=1}^n K_h(x - x_j) x_j$

```
PL <- function(h) {
    dx <- outer(x, x, "-")
    K <- dnorm(dx, sd = h)
    diag(K) <- 0
    T1 <- colSums(y*K)
    T2 <- colSums(x*dx*K)
    T3 <- colSums(x*y*K)
    T4 <- colSums(dx*K)
    B1 <- colSums(K)
    B2 <- colSums(x^2*K)
    B3 <- colSums(x*K)
    m.hat <- (T1*T2 - T3*T4) / (B1*B2 - B3^2)
    mean((m.hat - y)^2)
}
h <- seq(0.01, 0.07, length.out = 50)
mse.PL <- numeric(length(h))
for (i in seq_along(h)) {
    mse.PL[i] <- PL(h[i])
}
plot(h, mse.PL, type = "l", ylab = "leave-one-out MSE", main="Optimisation de h pour les poly
nômes locaux")
best.h.PL <- h[which.min(mse.PL)]
abline(v = best.h.PL, col="red")
legend("topright", legend = "h optimal", col = "red",lwd = 2)
```




```
cat("h optimal =", best.h.PL)
```

```
## h optimal = 0.04795918
```

La valeur de h est plus grande que celle trouvée avec l'estimateur de Nadaraya-Watson. On peut expliquer cette différence ainsi: la méthode par polynômes locaux étant plus discriminante par nature, elle a moins "besoin" d'une fenêtre étroite pour estimer la régression.

# K-Nearest Neighbour (K-NN)

(ou méthode des k plus proches voisins)

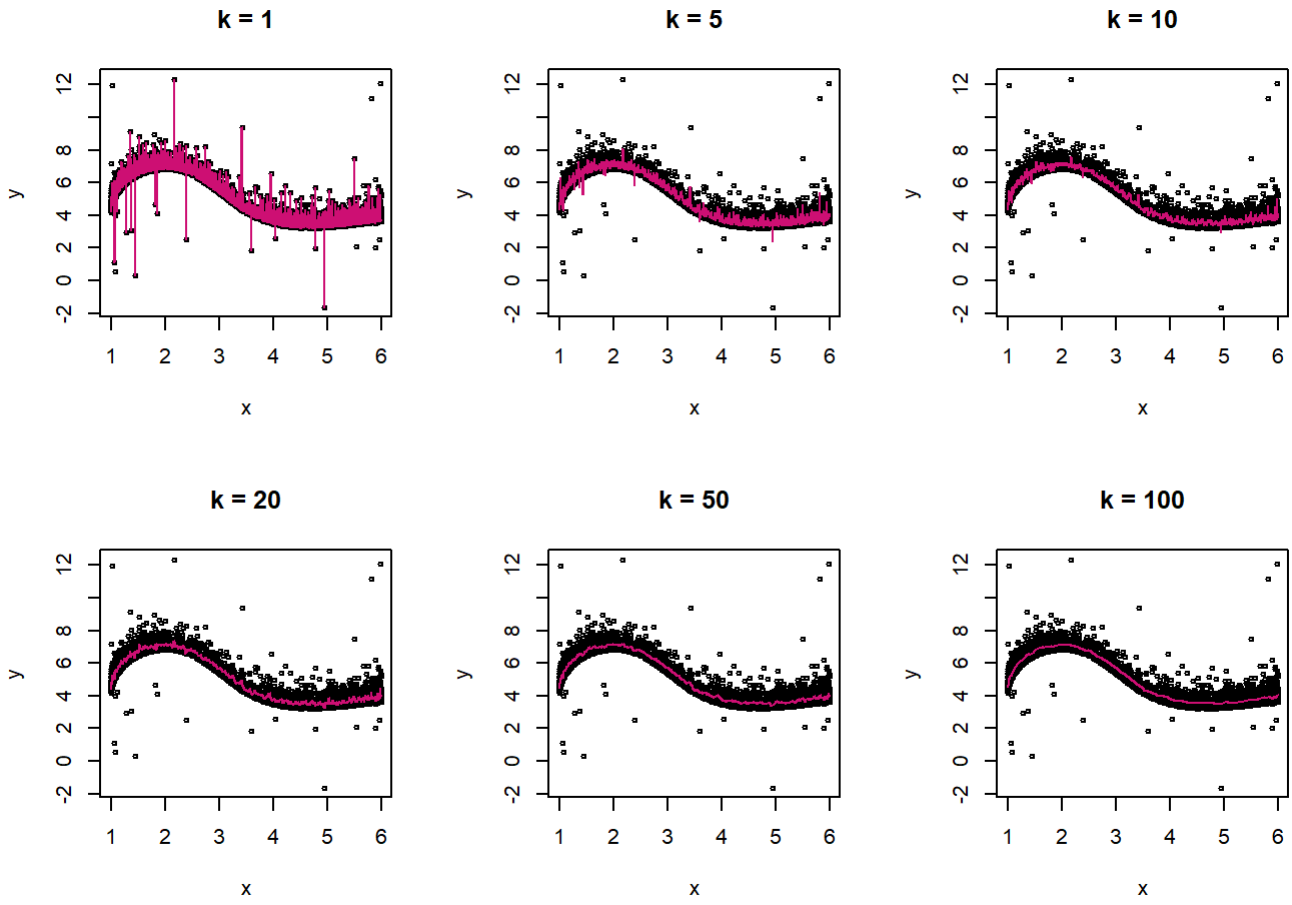
L'idée : Au lieu de fixer la taille de la fenêtre h, nous fixons le nombre k des observations les plus proches de x.

Ainsi $\hat{m}_h(x) =$ "la moyenne des valeurs y pour les k plus proches voisins".

Plus mathématiquement, cela donne: $\hat{m}_k(x) = \sum_{i=1}^{n} w_i(x) y_i$

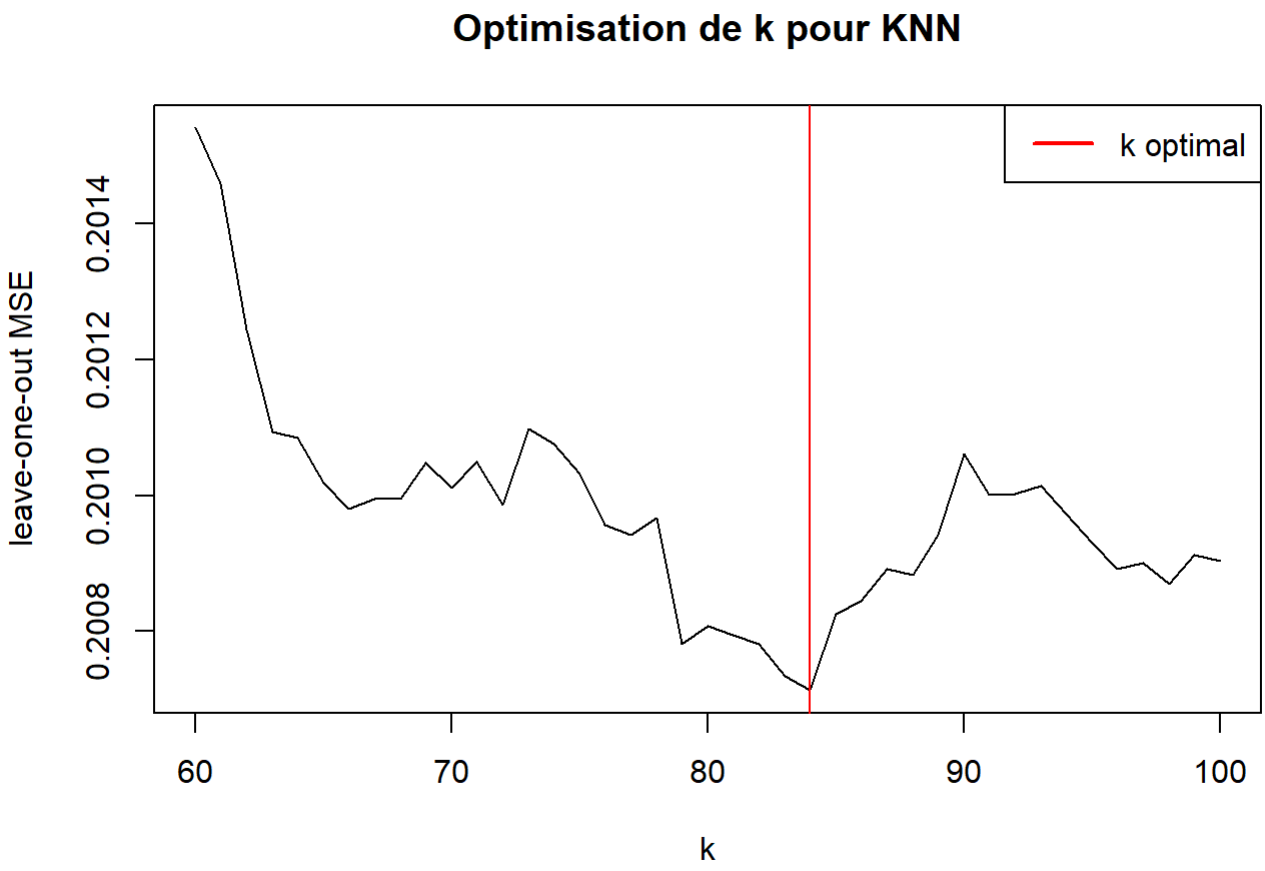
avec le poids $w_i(x) = \begin{cases} \frac{1}{k}, \text{si } i \in J_k(x) \\ 0, \text{sinon.} \end{cases}$

```
mat_X <- as.matrix(d[,1]) # la fonction "knn.reg()" nécessite une matrice en entrée
mat_Y <- as.matrix(d[,2])

par(mfrow=c(2,3))
X_test = seq(1, 6, length.out = 5000)

knn_1 <- knn.reg(train = mat_X, test= data.frame(X_test), y = mat_Y, k = 1)
knn_5 <- knn.reg(train = mat_X, test= data.frame(X_test), y = mat_Y, k = 5)
knn_10 <- knn.reg(train = mat_X, test= data.frame(X_test), y = mat_Y, k = 10)
knn_20 <- knn.reg(train = mat_X, test= data.frame(X_test), y = mat_Y, k = 20)
knn_50 <- knn.reg(train = mat_X, test= data.frame(X_test), y = mat_Y, k = 50)
knn_100 <- knn.reg(train = mat_X, test= data.frame(X_test), y = mat_Y, k = 100)

plot(x, y, cex = .5, xlab = "x", ylab = "y", main="k = 1")
lines(X_test, knn_1$pred, col = "deeppink3")
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="k = 5")
lines(X_test, knn_5$pred, col = "deeppink3")
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="k = 10")
lines(X_test, knn_10$pred, col = "deeppink3")
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="k = 20")
lines(X_test, knn_20$pred, col = "deeppink3")
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="k = 50")
lines(X_test, knn_50$pred, col = "deeppink3")
plot(x, y, cex = .5, xlab = "x", ylab = "y", main="k = 100")
lines(X_test, knn_100$pred, col = "deeppink3")
```




De même que pour l'optimisation de la fenêtre h, nous proposons un code pour trouver la valeur de k qui minimise le MSE (par LOO). Notons que cette méthode devient rapidement inutilisable si la taille de l'échantillon est trop grande, l'algorithme étant très coûteux en calculs.

```
k <- 60:100
knn <- numeric(length(k))
for (j in seq_along(k)) {
    y.pred <- numeric(length(x))
    for (i in seq_along(x)) {
        m <- knn.reg(data.frame(x = x[-i]),
                     y = y[-i],
                     test = data.frame(x = x[i]),
                     k = k[j])
        y.pred[i] <- m$pred
    }
    knn[j] <- mean((y - y.pred)^2)
}
plot(k, knn, type = "l", ylab = "leave-one-out MSE", main= "Optimisation de k pour KNN")
best_k <- k[which.min(knn)]
abline(v = best_k, col="red")
legend("topright", legend = "k optimal", best_k, col = "red",lwd = 2)
```

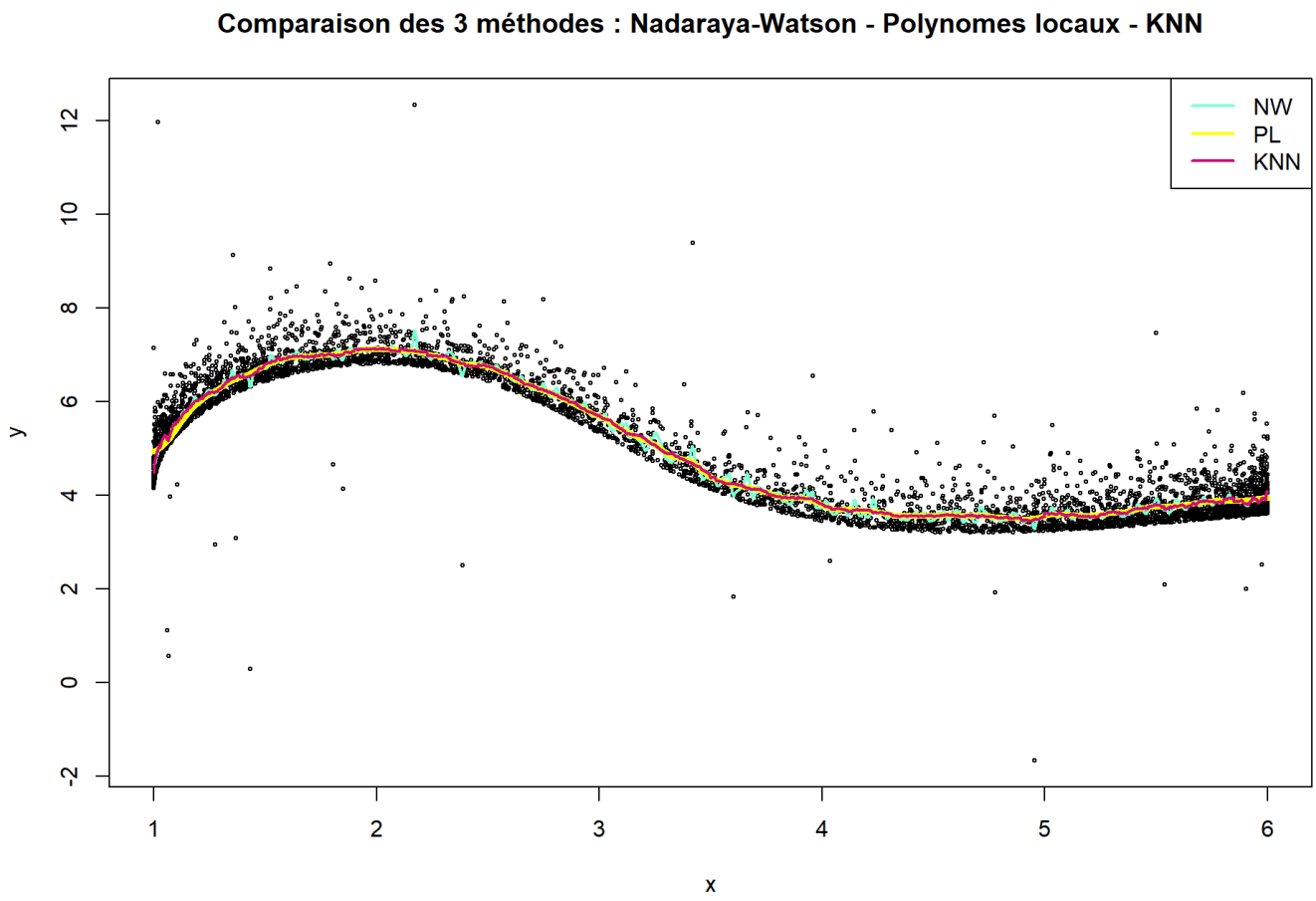



```
cat("k optimal =", best_k)
```

```
## k optimal = 84
```

Rassemblons au sein d'un même graphique les trois régressions avec les valeurs de h et k optimales:

```
plot(x, y, cex = .4, xlab = "x", ylab = "y", main="Comparaison des 3 méthodes : Nadaraya-Wats
on - Polynomes locaux - KNN")

lines(ksmooth(x, y, kernel = "normal", bandwidth = 0.02040816, n.points = 1000), col ="aquama
rine", lwd = 2) # Régression NW avec h optimal

lc <- locpoly(x, y, degree = 0, bandwidth = 0.04795918)
lines(lc, col = "yellow",lwd = 4)         # Régression polynôme locaux avec h optimal

knn <- knn.reg(train = x, test= data.frame(X_test), y = mat_Y, k = 84)
lines(X_test, knn$pred, col = "deeppink3", lwd = 2) # Régression KNN avec k optimal

legend("topright", legend = c("NW", "PL", "KNN"), col = c("aquamarine", "yellow","deeppink
3"),lwd = 2)
```




Nous pouvons avancer que pour notre étude, les deux méthodes KNN et polynômes locaux donnent de meilleurs résultats que la régression par estimateur de Nadaraya-Watson. Le compromis recherché entre sous et surajustement semble plus intéressant (NW tend ici au surajustement). Gardons à l'esprit que cette remarque ne s'applique qu'aux résultats obtenus avec nos choix (comme les séquences d'évaluation de h et k, les grilles de x, la méthode d'optimisation retenue, etc.) et ne saurait être généralisable.

# Conclusion

Nous avons utilisé trois méthodes principales pour notre régression: l'estimateur de Nadaraya-Watson, les polynômes locaux (généralisation de NW) et la méthode des K plus proches voisins.

La méthode des polynômes locaux est plus performante que l'estimateur de Nadaraya-Watson pour estimer les zones de rapides changements de valeurs (pics ou zones à forte courbure), NW étant très biaisé dans ces zones. Elle est également meilleure aux bornes de l'échantillon.

La méthode KNN est particulièrement adaptée si les données sont rassemblées par paquets ("cluster" dans la littérature) espacés entre eux.

D'autres approches existent. Par exemple, l'analyse de Fourier obtient de bons résultats en utilisant un noyau sinus. Ou encore les fonctions orthogonales, dont un cas particulier dit "spatially adaptive" (ou "locally adaptive") se base sur les ondelettes ("wavelets" dans la littérature). Cette dernière méthode est particulièrement adaptée quand la fonction à estimer est non-homogène dans l'espace (par exemple assez linéaire en certains endroits mais avec des sauts abrupts), là où les méthodes que nous avons appliquées seraient inopérantes car "rateraient" les sauts avec un h trop grand, et deviendraient très instables avec un h petit qui prendrait les sauts en compte.
Le principe de la méthode adaptative peut aussi s'appliquer à la régression par noyau où la largeur de la fenêtre peut être différente pour chaque point (méthode difficile à mettre en oeuvre).

Citons également la régression “Spline” qui consiste à ajuster des polynômes locaux ou des morceaux de polynômes entre des points adjacents formant des groupes appelés nœuds où les polynômes sont connectés de manière continue pour former une courbe lisse. Cette approche permet de capturer des structures de données complexes tout en évitant le surajustement.

Comme dans beaucoup d’autres domaines de la statistique, il existe une version bayésienne de l’approche non-paramétrique, où l’on spécifie une distribution à priori pour la fonction de régression. Ensuite, à partir des données observées et de la distribution à priori, une distribution postérieure pour la fonction de régression est calculée à l’aide du théorème de Bayes. Les avantages de cette méthode sont la possibilité de quantifier l’incertitude, la flexibilité dans la spécification des modèles et la gestion naturelle du surajustement. Cependant, elles peuvent nécessiter des calculs computationnels très coûteux, voire rédhibitoires.