Université de Poitiers 2024-2025 MASTER 2 STDV

Projet Statistique Non Paramétrique

Réalisé par :

Kodzo LIMA :
kodzo.lima@etu.univ-poitiers.fr

Estimation

Chargé de l'UE

MC. Yousri Slaoui

Table des matières

1 Introduction

Dans le cadre de la forction de Master Statistique des Données du Vivant, l'Ue Statistique non Paramétrique-Estimation a pour objectif de s'avoir estimer une fonction de régression ou une densité par différentes méthodes d'estimation non paramétrique pour différentes classes fonctionnelles. C'est dans cette optique que ce projet est donné pour faire dévélopper ces compétences notament la maîtrise de la méthode de sélection de la fenêtre, choix théorique, choix pratique et choix optimal en utilisant l'erreur quadratique moyenne et la validation croisée.

2 Partie Données simulées

Nous avons simulé deux varaibles X et Y avec $X \sim \mathcal{N}(0,1)$ et $Y \sim \frac{1}{3}\mathcal{N}(0,1)+\frac{2}{3}\mathcal{N}(4,\frac{1}{4})$. Nous arrivons à visualiser dans un premier temps ces variables avec le code suivant,l'histogramme des varaibles et la courbe qui est une approximation lissée de la densité réelle des données :

2.1 Visualisation des données simulées

```
#Partie Données simulées
# Paramètres
set.seed(123)
n <- 1000
#X suit une loi normale
X <- rnorm(n)
Np <- 100
a <- 0.2
hn <- n^(-a)
#Y est une somme de lois normales
# Paramètres des deux normales
n <- 1000
mu1 <- 0
sigma1 <- 1
mu2 <- 4
sigma2 <- 0.5
# Génération des données
composante <- sample(c(1, 2), size = n, replace = TRUE, prob = c(1/3,
   2/3))
Y <- ifelse(composante == 1,
            rnorm(n, mean = mu1, sd = sigma1),
            rnorm(n, mean = mu2, sd = sigma2))
# Visualisation de X
hist(X, breaks = 30, probability = TRUE, col = "lightblue",
     main = "Loi normale", xlab = "Valeurs de X")
lines(density(X), col = "red", lwd = 2)
# Visualisation de Y
hist(Y, breaks = 30, probability = TRUE, col = "lightblue",
     main = "Mixture de deux normales (1/3 et 2/3)", xlab = "Valeurs
        de Y")
lines(density(Y), col = "red", lwd = 2)
```

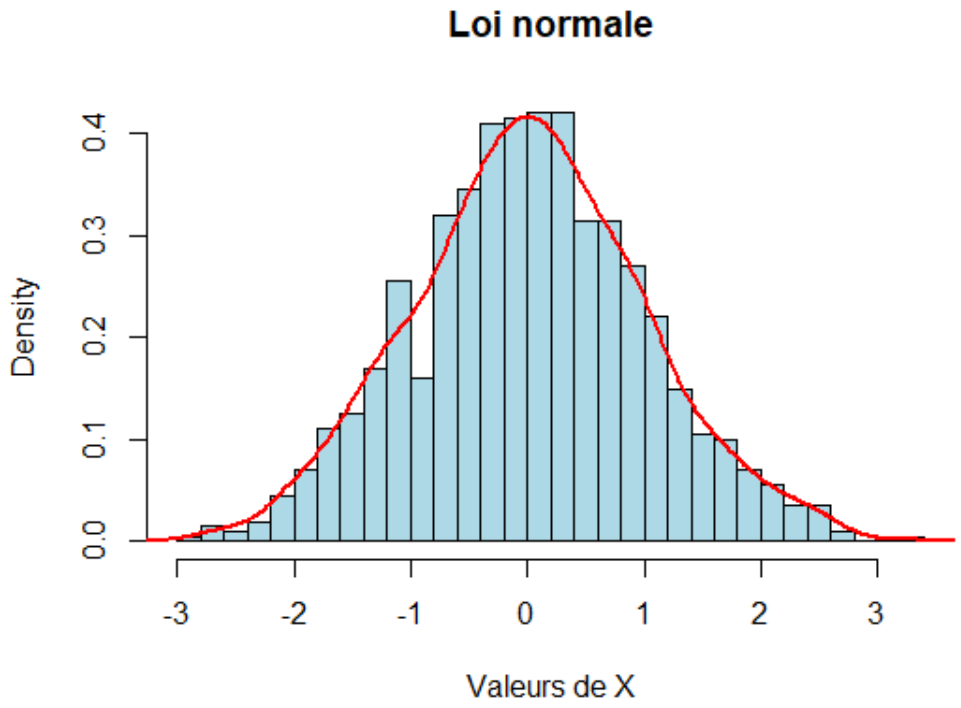

FIGURE 1 – Histogramme et courbe de Lissage de X

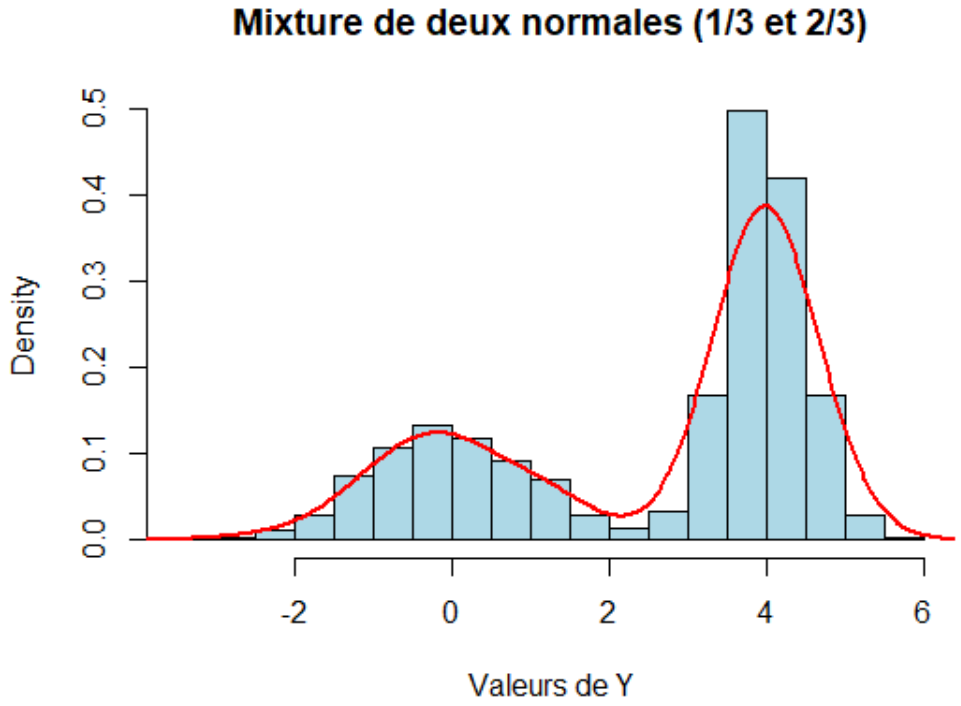

FIGURE 2 – Histogramme et courbe de Lissage de Y

2.2 Comparaison des Noyaux pour l'estimateur de la densité Parzen-Rosenblatt

L'estimateur de la densité Parzen-Rosenblatt est défini par $f_n(x) = \frac{1}{nh_n} \sum_{i=1}^{n} K\left(\frac{x-X_i}{h_n}\right)$

où $K : \mathbb{R} \to \mathbb{R}$ une fonction continue, bornée par M_k et telle que $\int_{\mathbb{R}} K(z)dz = 1$, $\int_{\mathbb{R}} zK(z)dz = 0$ et $\int_{\mathbb{R}} z^2|K(z)|dz < \infty$.

Par rapport au choix de noyaux, 5 ont été choisis à savoir Gaussien, Uniforme, Cubique, Circulaire, Quartique et Laplace.

Les noyaux sont définis mathématiquement comme suit :

- **Noyau Gaussien** :
$$K(x) = \frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2\pi}}$$

- **Noyau Uniforme** :
$$K(x) = 0.5 \cdot \mathbb{I}_{\{|x|\leq 1\}}$$
où $\mathbb{I}_{\{|x|\leq 1\}}$ est la fonction indicatrice de l'intervalle $[-1, 1]$.

- **Noyau Cubique** :
$$K(x) = 1.09375 \cdot (1 - x^2)^3 \cdot \mathbb{I}_{\{|x|\leq 1\}}$$

- **Noyau Circulaire** :
$$K(x) = \frac{\pi}{4} \cdot \cos\left(\frac{\pi}{2}x\right) \cdot \mathbb{I}_{\{|x|\leq 1\}}$$

- **Noyau Quartique** :
$$K(x) = 0.9375 \cdot (1 - x^2)^2 \cdot \mathbb{I}_{\{|x|\leq 1\}}$$

- **Noyau Laplace** :
$$K(x) = \frac{\exp(-\sqrt{2}|x|)}{\sqrt{2}}$$

Le code suivant effectue une estimation de la densité Parzen-Rosenblatt par noyau pour les deux variables simulées X et Y en utilisant les cinq (05) noyaux précédents, puis trace les densités estimées pour une comparaison.

```r
# Fonction pour estimer les densités de Parzen-Rosenblatt pour une
   variable donnée
estimation_parzen <- function(data, Np, hn, kernels) {
  # Points d'évaluation
  D <- seq(min(data), max(data), length.out = Np)

  # Stockage des résultats
  results <- list()

  # Calcul de l'estimation pour chaque noyau
  for (kernel_name in names(kernels)) {
    F_kernel <- rep(0, Np)
    for (i in 1:Np) {
      F_kernel[i] <- sum(kernels[[kernel_name]]((data - D[i]) / hn))
    }
    F_kernel <- F_kernel / (length(data) * hn)
    results[[kernel_name]] <- data.frame(D = D, Estimation = F_kernel)
  }

  return(results)
}



# Définition des différents noyaux
kernels <- list(
  Gaussien = function(x) exp(-x^2 / 2) / sqrt(2 * pi),
  Uniforme = function(x) 0.5 * (abs(x) <= 1),
  Cubique = function(x) 1.09375 * (1 - x^2)^3 * (abs(x) <= 1),
  Circulaire = function(x) (pi / 4) * cos((pi / 2) * x) * (abs(x) <=
     1),
  Quartique = function(x) 0.9375 * (1 - x^2)^2 * (abs(x) <= 1),
  Laplace = function(x) exp(-sqrt(2) * abs(x)) / sqrt(2)
)

# Estimations pour X et Y
results_X <- estimation_parzen(X, Np, hn, kernels)
results_Y <- estimation_parzen(Y, Np, hn, kernels)

# Tracé pour X
plot(seq(-4, 4, 0.1), dnorm(seq(-4, 4, 0.1)), type = "l", col = "grey"
   ,
     ylim = c(0, max(sapply(results_X, function(res) max(res$
        Estimation)))),
     xlab = "Observations", ylab = "Densités", lwd = 5,
     main = "Comparaison des noyaux pour X")
colors <- c("black", "yellow", "green", "blue", "red", "purple")
```

```r
i <- 1
for (kernel_name in names(results_X)) {
  lines(results_X[[kernel_name]]$D, results_X[[kernel_name]]$
     Estimation, type = "l", col = colors[i], lwd = 2)
  i <- i + 1
}
legend("topleft",
       legend = c("Référence", names(results_X)),
       col = c("grey", colors),
       lty = 1, lwd = 2, cex = 0.8)
# Tracé pour Y
densite_mixture <- function(y) {
  p1 * dnorm(y, mean = mu1, sd = sigma1) + p2 * dnorm(y, mean = mu2,
     sd = sigma2)
}

plot(seq(-6, 6, 0.1), densite_mixture(seq(-6, 6, 0.1)), type = "l",
   col = "grey",
     ylim = c(0, max(sapply(results_Y, function(res) max(res$
        Estimation)))),
     xlab = "Observations", ylab = "Densités", lwd = 5,
     main = "Comparaison des noyaux pour Y")

i <- 1
for (kernel_name in names(results_Y)) {
  lines(results_Y[[kernel_name]]$D, results_Y[[kernel_name]]$
     Estimation, type = "l", col = colors[i], lwd = 2)
  i <- i + 1
}
legend("topleft",
       legend = c("Référence", names(results_Y)),
       col = c("grey", colors),
       lty = 1, lwd = 2, cex = 0.8)
```

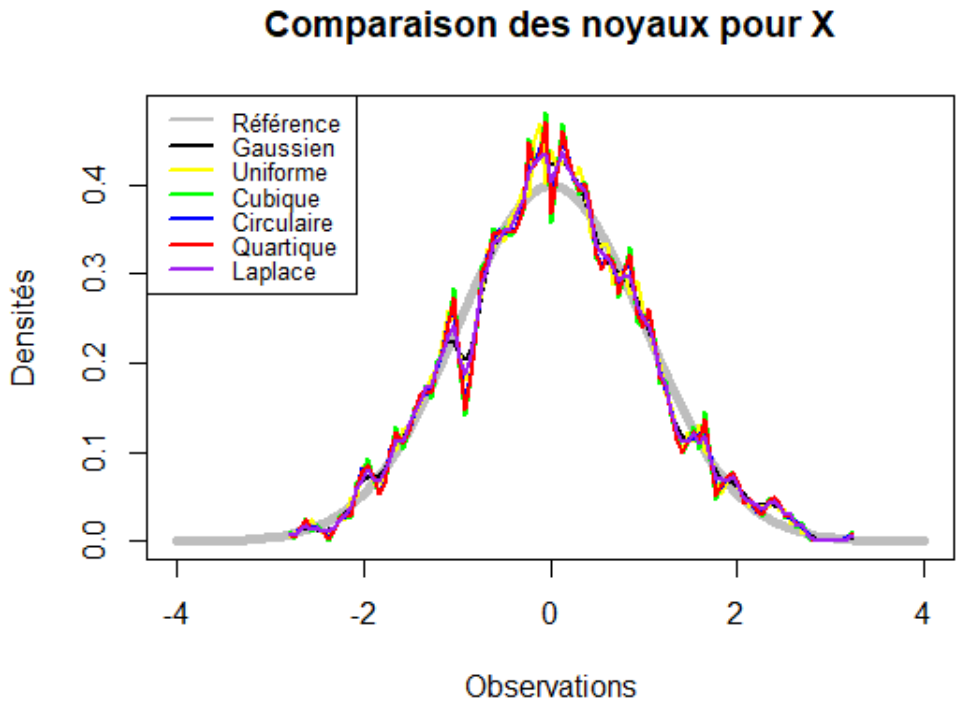

FIGURE 3 – Comparaison des Noyaux pour la varaiable X

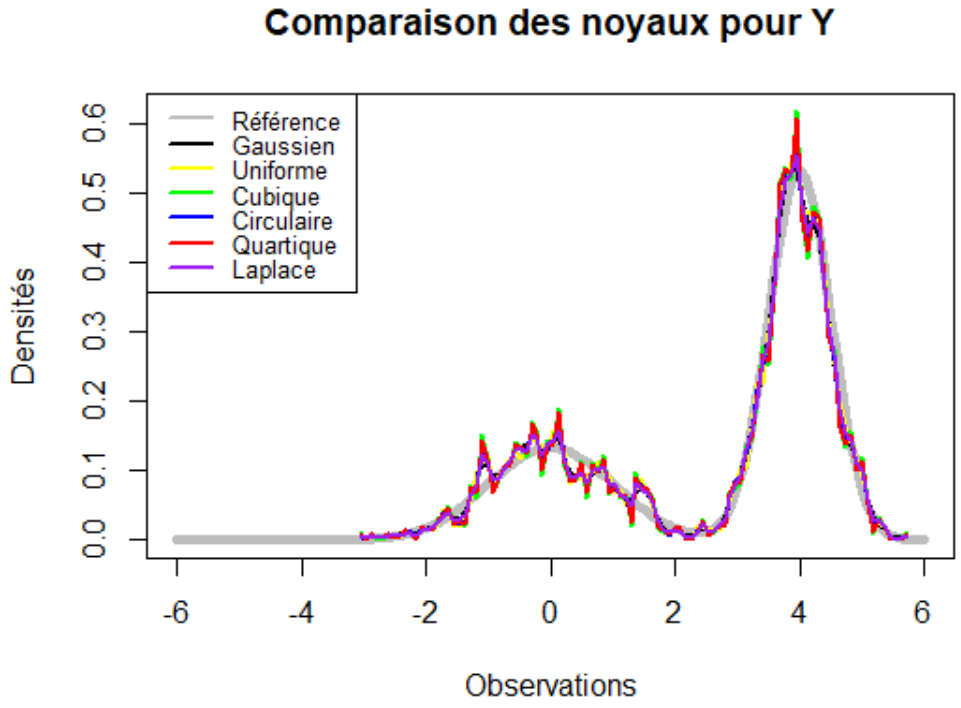

FIGURE 4 – Comparaison des Noyaux pour la varaiable Y

Interprétation des graphiques

*** Pour la variable X**

La comparaison des noyaux sur le graphique montre que le noyau Gaussien est le plus proche de la densité théorique. Les noyaux Uniforme et Circulaire, produisent des transitions plus nettes mais moins précis dans les zones de variation rapide. Les noyaux tels que Cubique et Quartique, capturent mieux les variations locales tout en maintenant un certain lissage. Enfin, le noyau Laplace se distingue par sa capacité à mieux modéliser les queues longues, en offrant un équilibre entre précision locale et lissage global. Le choix du noyau dépend donc de la forme et des particularités des données.

*** Pour la variable Y**

Le graphique montre que la distribution de Y est caractérisée par un pic prononcé et des variations locales significatives, ce qui rend les noyaux compacts (Uniforme, Circulaire, Cubique, Quartique) mieux adaptés pour capturer ces caractéristiques. Le noyau Laplace offre un bon équilibre entre lissage global et capture des variations locales, tandis que le noyau Gaussien est moins performant pour représenter les pics prononcés, mais reste utile pour des données globalement lisses. Le choix du noyau dépend ici de la priorité entre précision locale et lissage global.

2.3 Choix des différentes fenêtres

Dans cette partir nous avons choisis les fenêtres $n^{(-1/3)}$, $n^{(-1/4)}$, $n^{(-1/5)}$ et $n^{(-1/6)}$.

```
#Choix des différentes fenêtres n^(-1/3),n^(-1/4),n^(-1/5) et n^(-1/6)

# Définition des noyaux
noyaux <- list(
  Gaussien = function(x) exp(-x^2 / 2) / sqrt(2 * pi),
  Uniforme = function(x) 0.5 * (abs(x) <= 1),
  Cubique = function(x) 1.09375 * (1 - x^2)^3 * (abs(x) <= 1),
  Circulaire = function(x) (pi / 4) * cos((pi / 2) * x) * (abs(x) <=
     1),
  Quartile = function(x) 0.9375 * (1 - x^2)^2 * (abs(x) <= 1),
  Laplace = function(x) exp(-sqrt(2) * abs(x)) / sqrt(2)
)

# Points d'évaluation
Np <- 100  # Nombre de points d'évaluation
X <- runif(100, 0, 10)  # Exemple de données
D <- seq(min(X), max(X), length.out = Np)

# Paramètres
a_values <- c(1/3, 1/4, 1/5, 1/6)
n <- length(X)  # Taille des données
taille <- seq(10, n, by = 10)

# Initialisation des estimations
estimations <- list()

# Calcul des estimations pour chaque noyau
for (nom_noyau in names(noyaux)) {
```

```
  noyau <- noyaux[[nom_noyau]]

# Initialiser une matrice pour stocker les estimations
  estimation <- matrix(0, nrow = length(taille), ncol = length(a_
     values))

  for (j in seq_along(a_values)) {
    a <- a_values[j]

    for (i in seq_along(taille)) {
      n_i <- taille[i]
      hn <- n_i^(-a)
      X_i <- X[1:n_i]

      # Calcul de l'estimation pour le noyau
      estimation[i, j] <- sum(noyau((X_i - mean(D)) / hn)) / (n_i * hn
         )
    }
  }

  estimations[[nom_noyau]] <- estimation
}

# Tracé des courbes pour chaque noyau
for (nom_noyau in names(estimations)) {
  estimation <- estimations[[nom_noyau]]

# Vérifier que les valeurs sont finies
  if (!all(is.finite(estimation))) {
    warning(paste("Des valeurs non finies détectées pour le noyau",
       nom_noyau))
    next
  }

# Calcul des limites des axes
  ylim_values <- range(estimation, finite = TRUE)

# Création du graphique
  plot(
    taille, estimation[, 1], type = "l", col = "green", ylim = ylim_
       values,
    xlab = "Taille d'échantillon", ylab = "Estimation",
    main = paste("Convergence de l'estimation avec le noyau", nom_
       noyau)
  )

# Ajout des autres courbes
  lines(taille, estimation[, 2], col = "blue", lwd = 2)
  lines(taille, estimation[, 3], col = "red", lwd = 2)
  lines(taille, estimation[, 4], col = "purple", lwd = 2)
```

```
# Ligne horizontale de référence
  abline(h = 0.4, lty = 2, lwd = 2, col = "black")

# Légende
  legend(
    "topright",
    legend = c("Fenêtre avec a=1/3", "Fenêtre avec a=1/4", "Fenêtre
       avec a=1/5", "Fenêtre avec a=1/6"),
    col = c("green", "blue", "red", "purple"), lty = 1, lwd = 2
  )
}
```

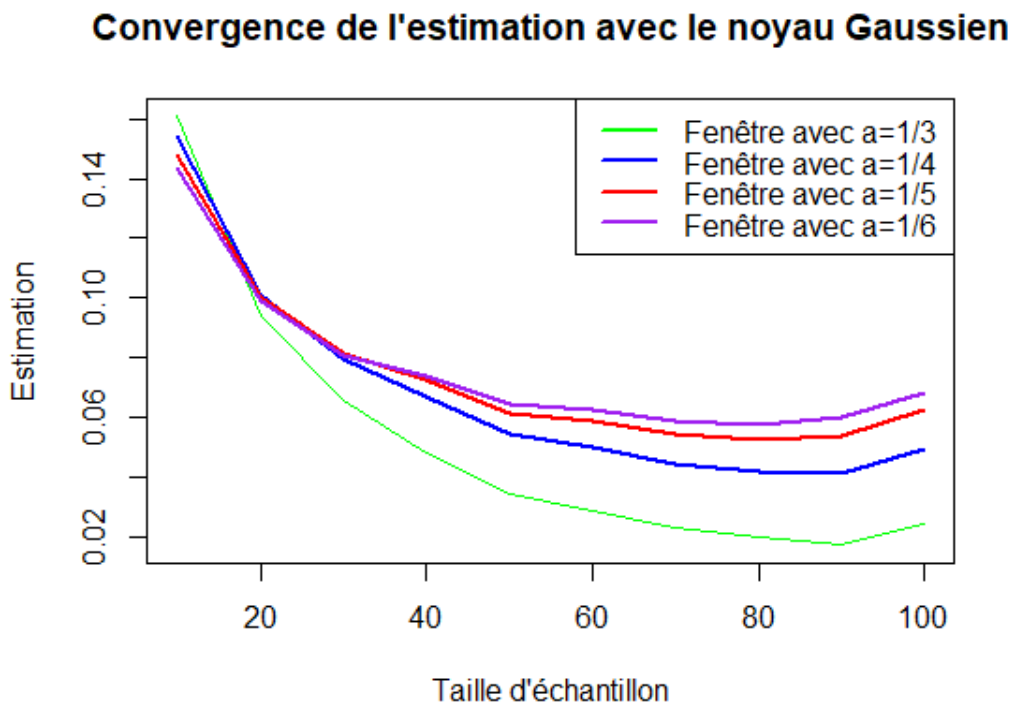

FIGURE 5 – Convergence de l'estimation avec le noyau Gaussien

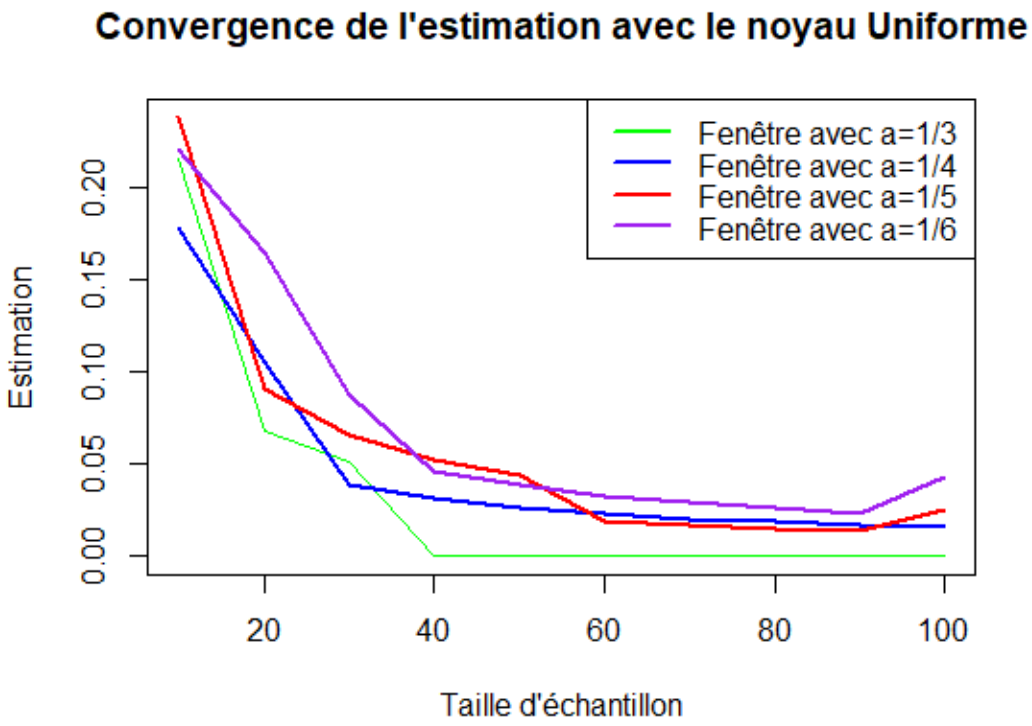

FIGURE 6 – Convergence de l'estimation avec le noyau Uniforme

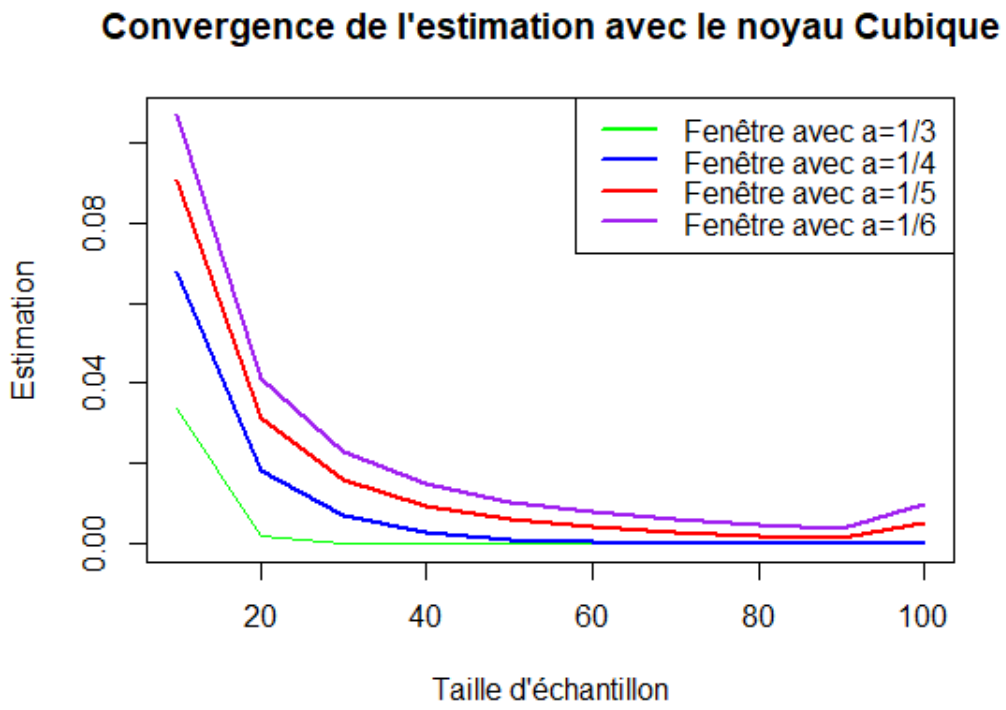

FIGURE 7 – Convergence de l'estimation avec le noyau Cubique

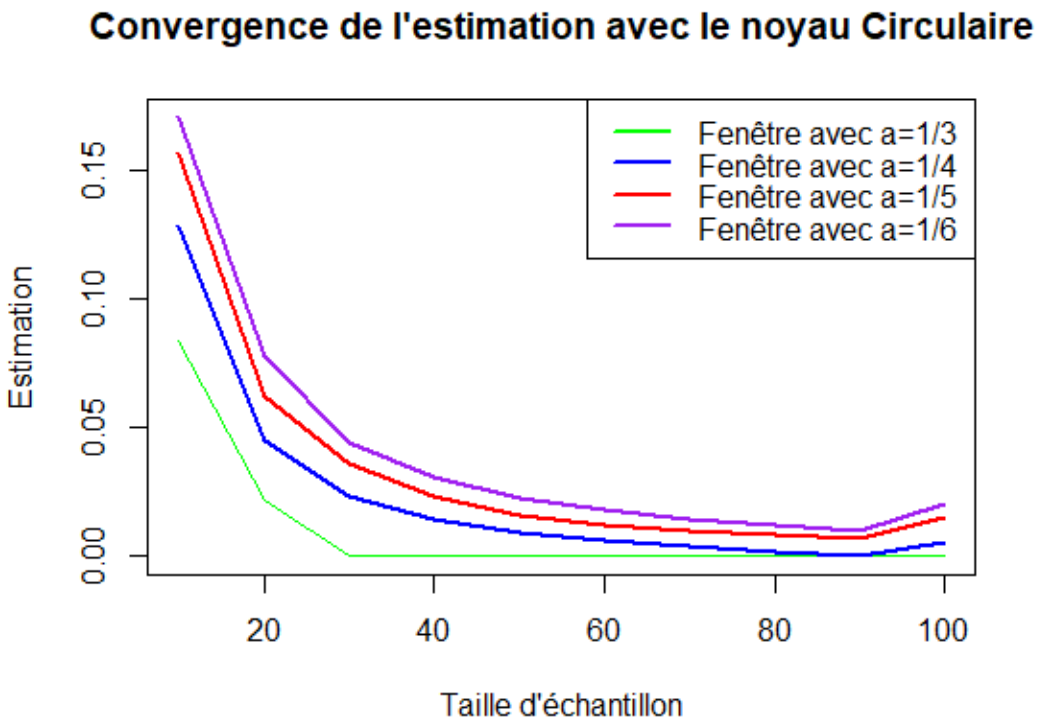

FIGURE 8 – Convergence de l'estimation avec le noyau Circulaire

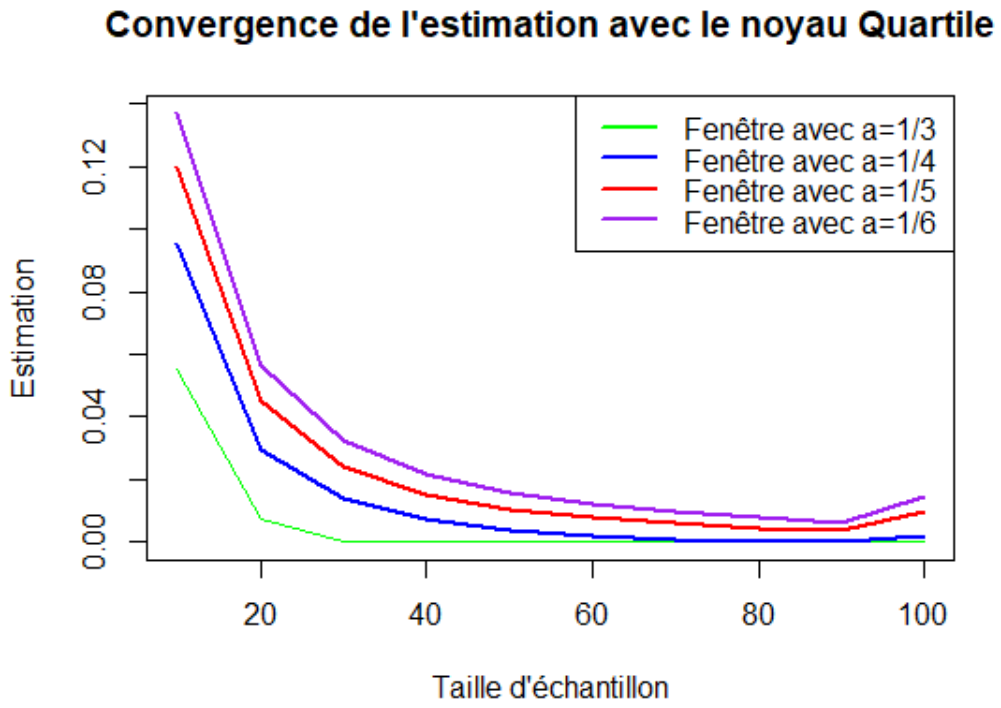

FIGURE 9 – Convergence de l'estimation avec le noyau Quartile

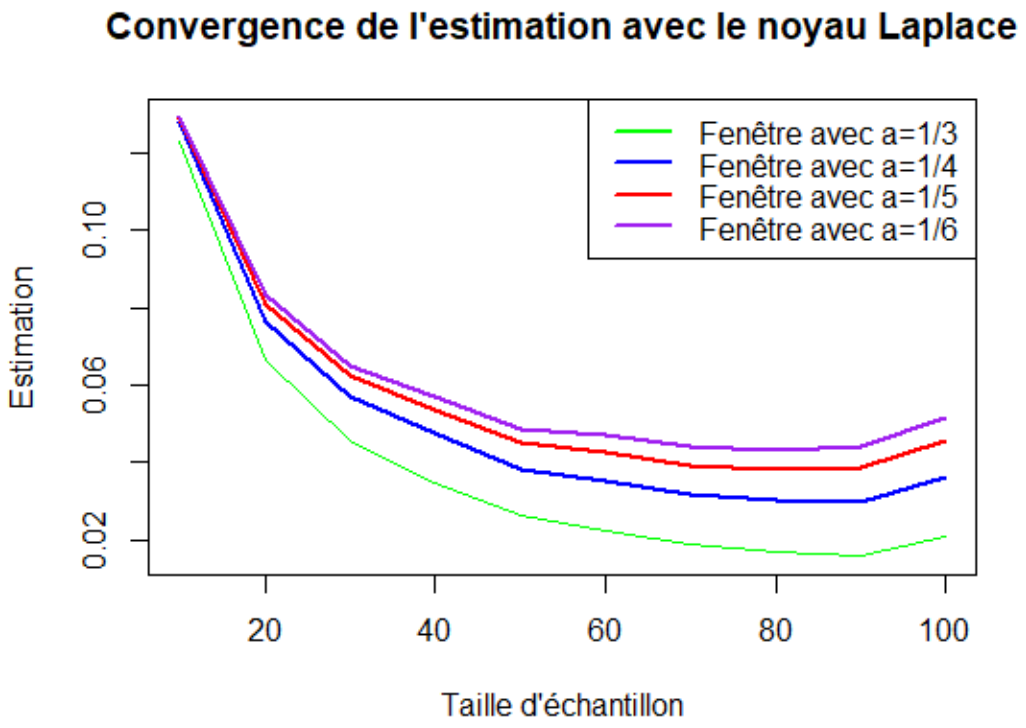

FIGURE 10 – Convergence de l'estimation avec le noyau Laplace

*** Interprétation :**
Le choix de la largeur de la fenêtre (a) dépend donc de la taille de l'échantillon. Pour des échantillons petits, il est préférable d'utiliser une fenêtre relativement large pour éviter une trop grande variance. Pour des échantillons grands, une fenêtre plus étroite permet une meilleure capture des détails de la distribution. Ces graphique illustrent ainsi le compromis biais-variance inhérent aux méthodes d'estimation par noyaux.

3 Partie Données Réelles

La donnée UScrime est un jeu de données intégré au logiciel R (dans le package MASS). Il contient des informations socio-économiques et démographiques ainsi que des statistiques sur les crimes pour 47 états des États-Unis.
Nous allons nous intésser à la vaiable **Ed** : Niveau moyen d'éducation en années d'étude.

```
#Partie Données Réelles
# Charger le jeu de données
library(MASS)
    data <- UScrime$Ed

# Définir les paramètres
n <- length(data)  # Taille de l'échantillon
Np <- 100  # Nombre de points d'évaluation
a <- 0.2  # Exposant pour la fenêtre
hn <- n^(-a)  # Taille de la fenêtre
```

```r
# Définir les noyaux
kernels <- list(
  Gaussien = function(x) exp(-x^2 / 2) / sqrt(2 * pi),
  Uniforme = function(x) 0.5 * (abs(x) <= 1),
  Cubique = function(x) 1.09375 * (1 - x^2)^3 * (abs(x) <= 1),
  Circulaire = function(x) (pi / 4) * cos((pi / 2) * x) * (abs(x) <=
     1),
  Quartique = function(x) 0.9375 * (1 - x^2)^2 * (abs(x) <= 1),
  Laplace = function(x) exp(-sqrt(2) * abs(x)) / sqrt(2)
)

# Estimation de la densité
results <- estimation_parzen(data, Np, hn, kernels)

# Tracé des résultats
plot(seq(min(data), max(data), length.out = Np), dnorm(seq(min(data),
   max(data), length.out = Np)),
     type = "l", col = "grey",
     ylim = c(0, max(sapply(results, function(res) max(res$Estimation)
        ))),
     xlab = "Observations", ylab = "Densités", lwd =  5,
     main = "Comparaison des noyaux pour les données réelles")
colors <- c("black", "yellow", "green", "blue", "red", "purple")
i <- 1
for (kernel_name in names(results)) {
  lines(results[[kernel_name]]$D, results[[kernel_name]]$Estimation,
     type = "l", col = colors[i], lwd = 2)
  i <- i + 1
}
legend("topleft",
       legend = c("Référence", names(results)),
       col = c("grey", colors),
       lty = 1, lwd = 2, cex = 0.8)
```

*** Interprétation :**

Ce graphique compare les performances de différents noyaux pour l'estimation de densité sur des données réelles. Les courbes montrent que tous les noyaux (Gaussien, Uniforme, Cubique, Circulaire, Quartique, Laplace) capturent globalement la tendance des données et sont proches de la courbe de référence, bien que des variations existent. Le noyau Gaussien, lisse et robuste, s'approche le plus de la référence, tandis que les noyaux Uniforme et Laplace capturent davantage les détails autour des pics, mais avec des transitions plus abruptes. Les noyaux comme Quartique et Circulaire offrent un compromis intermédiaire. Ainsi, le choix du noyau dépend du besoin : lisser globalement les données (Gaussien) ou capturer les variations locales (Uniforme ou Laplace).

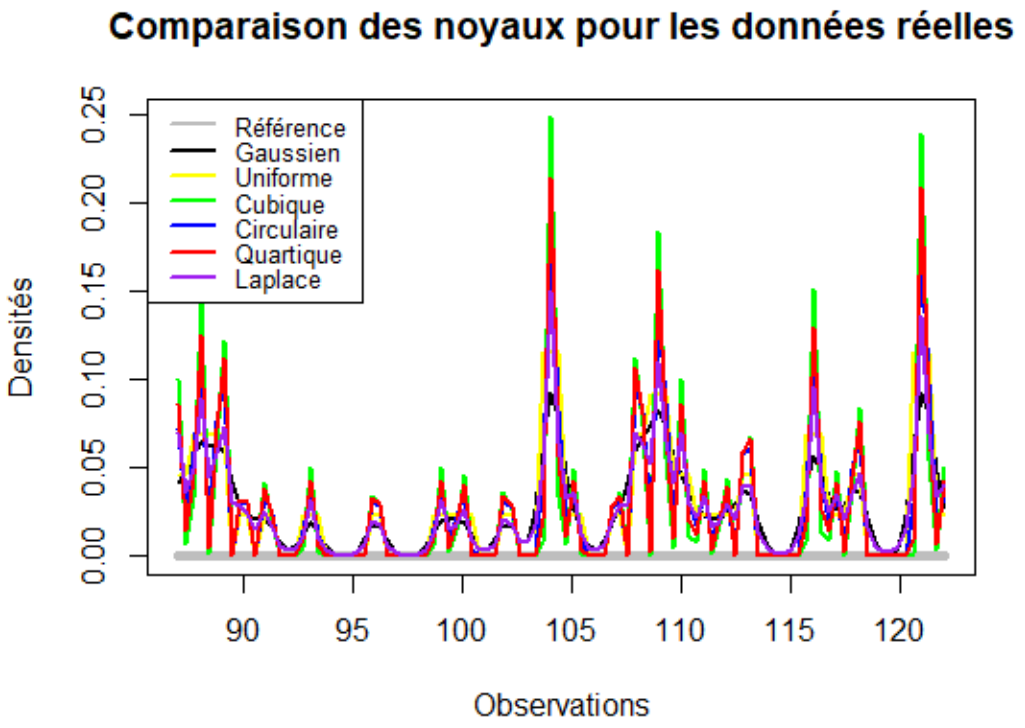

FIGURE 11 – Comparaison des noyaux pour les données réelles

4 Détermination de h_{CV} par la métode de la validation croisée

On utilise le code suivant pour déterminer les h_{CV} des varaibles X ,Y et la donnée réelle **Ed**

On définit l'estimateur à noyau basé sur l'échantillon réduit $X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n$, où l'observation X_i a été supprimée, par :

$$\hat{f}_{-i}(x) = \frac{1}{(n-1)h_n} \sum_{\substack{j=1 \\ j \neq i}}^{n} K\left(\frac{x - X_j}{h_n}\right),$$

Nous avions montré au cours que :

$$\mathbb{E}\left[\hat{f}_{-i}(X_i)\right] = \mathbb{E}\left\{\int_{\mathbb{R}} \hat{f}_n(x) f(x)\, dx\right\}$$

Et

$$CV(h_n) = \int_{\mathbb{R}} \hat{f}_n^2(x)\, dx - \frac{2}{n} \sum_{i=1}^{n} \hat{f}_{-i}(X_i)$$

est un estimateur sans biais de

$$\text{EQMI}[f_n] - \int_{\mathbb{R}} f^2(x)\, dx.$$

On a alors dans ce cas :

$$CV(h_n) = \frac{1}{n^2 h_n} \sum_{i=1}^{n} \sum_{j=1}^{n} \int K * K \left(\frac{X_j - X_i}{h_n} \right) dz - \frac{2}{n(n-1)h_n} \sum_{j=1}^{n} \sum_{\substack{i=1 \\ i \neq j}}^{n} K \left(\frac{X_i - X_j}{h_n} \right),$$

où $K * K$ représente la convolution du noyau K avec lui-même.

On considère le noyaux gaussien définit par :

$$K(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x - \mu_1)^2}{2\sigma_1^2} \right)$$

On sait que si $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$ et $Y \sim \mathcal{N}(\mu_Y, \sigma_Y^2)$,
La densité de la convolution est alors donnée par :

$$f_x * f_y(z) = \frac{1}{\sqrt{2\pi(\sigma_X^2 + \sigma_Y^2)}} \exp\left(-\frac{(z - (\mu_X + \mu_Y))^2}{2(\sigma_X^2 + \sigma_Y^2)} \right).$$

Alors on a :

$$CV(h_n) = \frac{1}{2\sqrt{\pi} n^2} \left[\frac{n}{h_n} - \frac{1}{h_n} \sum_{i<j}^{n} \left\{ 4\sqrt{2} \frac{n}{n-1} \exp\left(-\frac{(X_j - X_i)^2}{2h_n^2} \right) - 2 \exp\left(-\frac{(X_j - X_i)^2}{4h_n^2} \right) \right\} \right]$$

On peut donc avoir le h optimisé :

$$\hat{h}_{CV} = \arg\min_{h_n} CV(h_n)$$

```
#VALIDATION CROISEE
# Fonction pour calculer CV(h_n)
calculer_CV <- function(X, h_n, kernel) {
# Vérifier que le noyau est une fonction valide
  if (!is.function(kernel)) {
    stop("Le noyau doit être une fonction valide.")
  }

# Taille de l'échantillon
  n <- length(X)

# Calcul de la première somme double
  somme1 <- 0
  for (i in 1:n) {
    for (j in 1:n) {
      z <- (X[j] - X[i]) / h_n
      somme1 <- somme1 + kernel(z) * kernel(z)
    }
```

```
  }
  terme1 <- somme1 / (n^2 * h_n)

# Calcul de la deuxième somme double
  somme2 <- 0
  for (i in 1:n) {
    for (j in 1:n) {
      if (i != j) {
        z <- (X[j] - X[i]) / h_n
        somme2 <- somme2 + kernel(z)
      }
    }
  }
  terme2 <- (2 / (n * (n - 1) * h_n)) * somme2

# Calcul de CV(h_n)
  CV <- terme1 - terme2
  return(CV)
}


#Pour X
# On essaie  d en  trouver le min pour tous les noyaux
x11()
par(mfrow = c(3, 2))

results <- list()

for (nom_noyau in names(kernels)) {
  kernel <- kernels[[nom_noyau]]

  # Génération des valeurs de a
  a <- seq(0.01, 0.99, by = 0.01)

  # Calcul de CV(h_n) pour chaque a
  H <- sapply(a, function(a_val) calculer_CV(X, h_n = length(X)^(-a_
     val), kernel))
  # Sauvegarde des résultats pour ce noyau
  results[[nom_noyau]] <- list(a = a, H = H, min_a = a[which.min(H)],
     min_CV = min(H))

  # Tracer CV(h_n) pour ce noyau
  plot(a, H, type = "l", col = "red", xlab = "a", ylab = "CV",
       main = paste("Noyau :", nom_noyau))
  points(a[which.min(H)], min(H), col = "blue", pch = 19)
}

 # Résultats pour tous les noyaux
results
```

```r
#POUR Y
x11() # Ouvre une nouvelle fenêtre graphique
par(mfrow = c(3, 2)) # Divise la fenêtre graphique en une grille 3x2

results_Y <- list() # Pour stocker les résultats pour Y

for (nom_noyau in names(kernels)) {
  kernel <- kernels[[nom_noyau]] # Sélection du noyau actuel

  # Génération des valeurs de a
  a <- seq(0.01, 0.99, by = 0.01)

  # Calcul de CV(h_n) pour chaque a pour Y
  H_Y <- sapply(a, function(a_val) calculer_CV(Y, h_n = length(Y)^(-a_
     val), kernel))

  # Sauvegarde des résultats pour ce noyau
  results_Y[[nom_noyau]] <- list(a = a, H = H_Y, min_a = a[which.min(H
     _Y)], min_CV = min(H_Y))

  # Tracer CV(h_n) pour ce noyau
  plot(a, H_Y, type = "l", col = "blue", xlab = "a", ylab = "CV",
       main = paste("Noyau (Y) :", nom_noyau))
  points(a[which.min(H_Y)], min(H_Y), col = "red", pch = 19) # Point
     minimal
}

# Affichage des résultats pour Y
results_Y




#Pour les données réelles
# On essaie  d en  trouver le min pour tous les noyaux sur la donnée
   UScrime$Ed
x11()
par(mfrow = c(3, 2))

results <- list()

for (nom_noyau in names(kernels)) {
  kernel <- kernels[[nom_noyau]]

  # Génération des valeurs de a
  a <- seq(0.01, 0.99, by = 0.01)

  # Calcul de CV(h_n) pour chaque a en utilisant UScrime$Ed comme X
```

```
  H <- sapply(a, function(a_val) calculer_CV(data, h_n = length(data)
     ^(-a_val), kernel))

  # Sauvegarde des résultats pour ce noyau
  results[[nom_noyau]] <- list(a = a, H = H, min_a = a[which.min(H)],
     min_CV = min(H))

  # Tracer CV(h_n) pour ce noyau
  plot(a, H, type = "l", col = "red", xlab = "a", ylab = "CV",
       main = paste("Noyau :", nom_noyau))
  points(a[which.min(H)], min(H), col = "blue", pch = 19)
}

# Afficher les résultats pour tous les noyaux
results
```

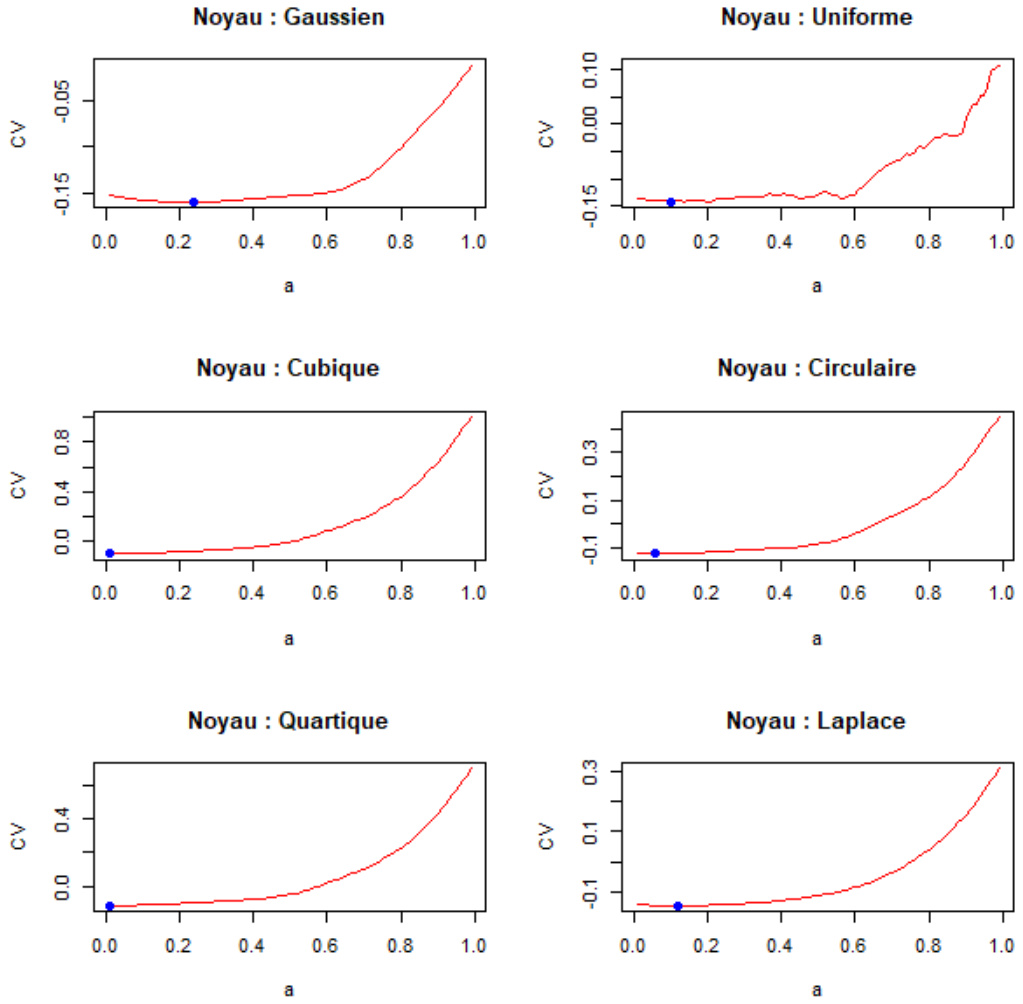

FIGURE 12 – Détermination de h_{CV} pour X

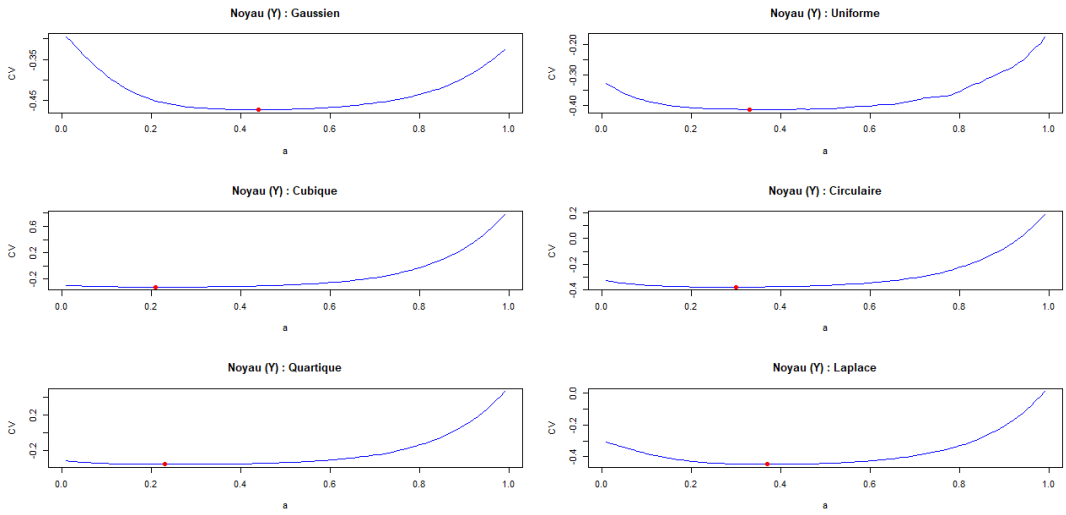

FIGURE 13 – Détermination de h_{CV} pour Y

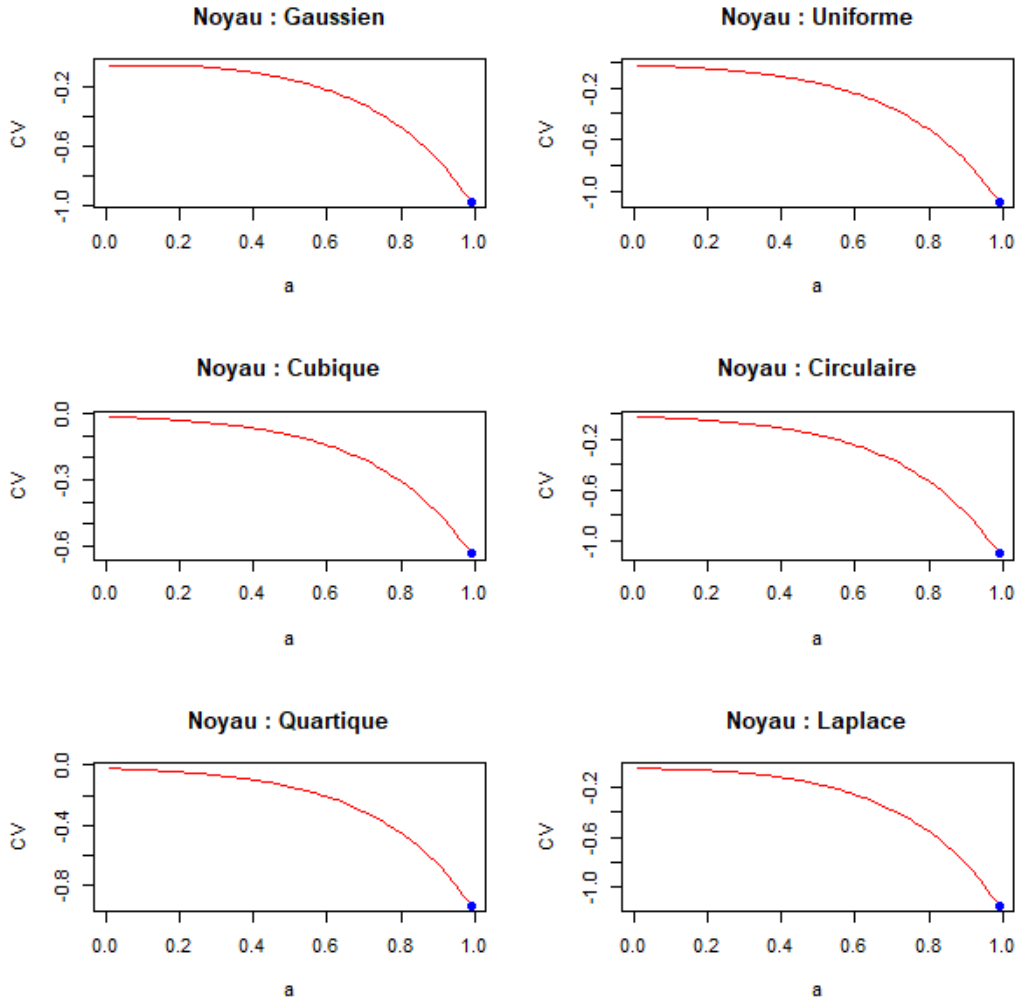

FIGURE 14 – Détermination de h_{CV} pour Ed

*** Interprétation :**
Ce graphique montre l'évolution du critère de validation croisée (CV) pour différents noyaux (Gaussien, Uniforme, Cubique, Circulaire, Quartique, Laplace) en fonction du paramètre a, représentant la largeur de la fenêtre. Pour chaque noyau, la courbe rouge illustre un compromis biais-variance : une diminution initiale du CV suivie d'une augmentation après un point optimal (marqué par le point bleu). Ce point optimal correspond à la valeur de a qui équilibre le mieux la précision de l'estimation en minimisant le CV. Les noyaux Gaussien, Cubique, et Quartique affichent des courbes plus régulières, tandis que les noyaux Uniforme et Circulaire présentent des variations plus sensibles, reflétant leur réactivité aux données.

5 Estimateur de Nadaraya-Watson

Soient (X_1, Y_1),.....,(X_n, Y_n) des varaiables aléatoires de même loi, à valeurs dans $\mathbb{R}^2$. On note g(x,y) la densité de du couple (X,Y), f(x) la densité marginale de X et $r(x) = \mathbb{E}[Y/X = x]$ la régression de Y sur X, et $m(x) = f(x)r(x)$. L'estimateur de Nadaraya-Watson est défini par :

$$r(x) = \frac{m(x)}{f(x)}$$

Avec

$$f_n(x) = \frac{1}{nh_n} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h_n}\right) Y_i$$

Et

$$f_n(x) = \frac{1}{nh_n} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h_n}\right)$$

Le code suivant implémente un estimateur de Nadaraya-Watson pour ajuster une relation non paramétrique entre une variable explicative $X \sim \mathcal{N}(5, 2)$ et $Y = sin(X) + cos(X) + \epsilon$ avec $\epsilon \sim \mathcal{N}(0, 0.3)$. Un noyau gaussien est utilisé pour attribuer des poids décroissants en fonction de la distance, et la largeur de fenêtre(h) est optimisée via validation croisée leave-one-out pour minimiser l'erreur quadratique moyenne. Une fois h optimal déterminé, l'estimateur est appliqué à une grille de valeurs pour obtenir une courbe lissée représentant la relation ajustée. Le graphique montre les données observées tracées en bleu et la courbe ajustée par l'estimateur en rouge.

```
# ESTIMATEUR DE NADARAYA-WATSON
set.seed(123)

# Générer X avec une loi normale de variance plus grande
n <- 100
X <- rnorm(n, mean = 5, sd = 2)

# Générer le bruit epsilon avec une petite variance
Y <- sin(X) + cos(X) + rnorm(n, mean = 0, sd =  0.3)

# Noyau Gaussien
kernel_gaussien <- function(x) {
  exp(-0.5 * x^2) / sqrt(2 * pi)
}

# Fonction de Nadaraya-Watson
Rn <- function(x, X, Y, h, kernel) {
  numerateur <- sum(kernel((x - X) / h) * Y)
  denominateur <- sum(kernel((x - X) / h))
  if (denominateur == 0) return(NA)
  return(numerateur / denominateur)
}

# Validation croisée pour trouver le h optimal
cv_error <- function(h, X, Y, kernel) {
  n <- length(Y)
  erreur <- 0
  for (i in 1:n) {
    # Laisser un point de  c  té (leave-one-out)
    x_val <- X[i]
```

```
    y_val <- Y[i]
    X_train <- X[-i]
    Y_train <- Y[-i]

    # Prédiction pour ce point
    y_pred <- Rn(x_val, X_train, Y_train, h, kernel)
    erreur <- erreur + (y_val - y_pred)^2
  }
  return(erreur)
}

# Tester plusieurs valeurs de h pour minimiser l'erreur
h_values <- seq(0.1, 2, by = 0.1)
errors <- sapply(h_values, function(h) cv_error(h, X, Y, kernel_
   gaussien))
h_optimal <- h_values[which.min(errors)]

# Afficher le h optimal
cat("Largeur de fenêtre optimale (h) :", h_optimal, "\n")

# Tracer le nuage de points et l'ajustement
x_grid <- seq(min(X), max(X), length.out = 100)
y_estime <- sapply(x_grid, function(x) Rn(x, X, Y, h_optimal, kernel_
   gaussien))

plot(X, Y, pch = 19, col = "blue", xlab = "X", ylab = "Y", main = "
   Estimateur de Nadaraya-Watson")
lines(x_grid, y_estime, col = "red", lwd = 2)

# Ajouter une légende avec une taille réduite
legend("topright",
       legend = c("Nuage de points", "Ajustement"),
       col = c("blue", "red"),
       pch = c(19, NA),
       lty = c(NA, 1),
       cex = 0.8)
```

La Largeur de fenêtre optimale (h) : 0.3

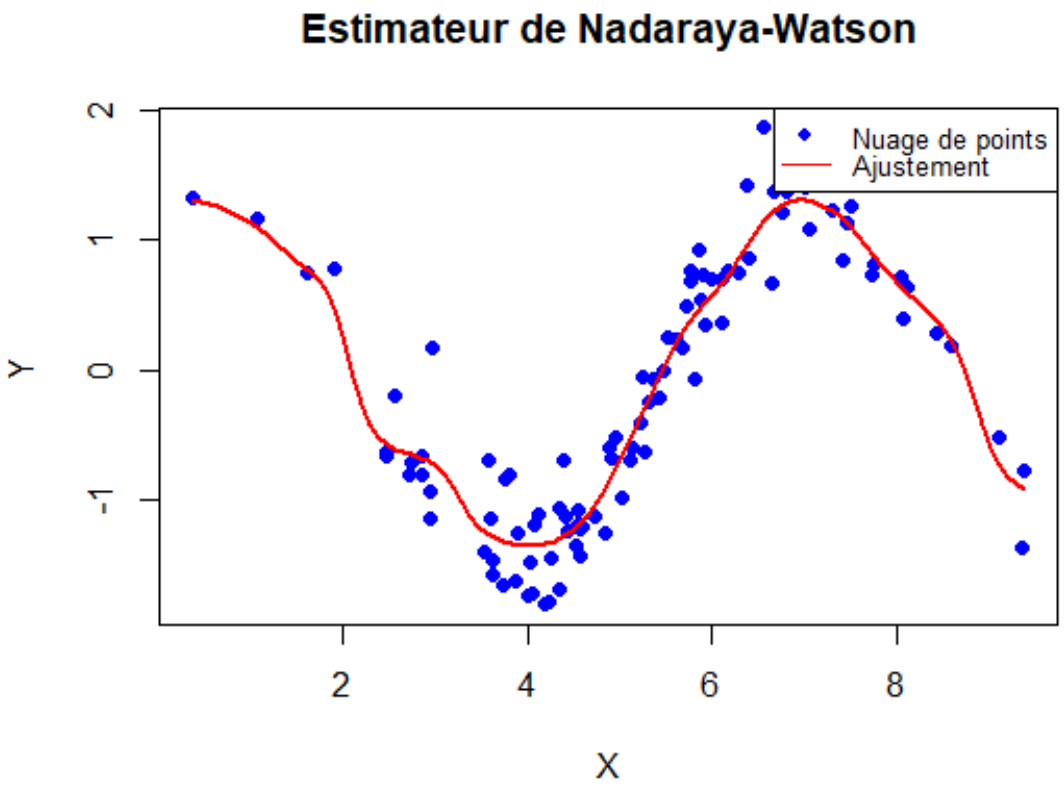

FIGURE 15 – Ajustement par l'Estimateur de Nadaraya-Watson

5.1 Comparaison des différents choix de fenêtre et différentes fonctions de régression

```
#9

# Tester plusieurs valeurs de h
h_values <- seq(0.1, 2, by = 0.1)

# Résultats pour chaque noyau
results <- list()
for (kernel_name in names(kernels)) {
  kernel <- kernels[[kernel_name]]

  # Calcul des erreurs pour chaque h
  errors <- sapply(h_values, function(h) cv_error(h, X, Y, kernel))
  errors <- ifelse(is.finite(errors), errors, NA)  # Remplacer les non
     -finis par NA
  h_optimal <- h_values[which.min(errors)]

  # Calcul de l'estimation avec h optimal
  x_grid <- seq(min(X), max(X), length.out = 100)
  y_estime <- sapply(x_grid, function(x) Rn(x, X, Y, h_optimal, kernel
     ))
```

```r
  # Stocker les résultats
  results[[kernel_name]] <- list(
    h_optimal = h_optimal,
    x_grid = x_grid,
    y_estime = y_estime,
    errors = errors
  )
}

# Tracer les résultats
par(mfrow = c(2, 3), mar = c(4, 4, 2, 1))
for (kernel_name in names(results)) {
  result <- results[[kernel_name]]

  plot(X, Y, pch = 19, col = "blue", xlab = "X", ylab = "Y",
       main = paste("Noyau", kernel_name, "\n h optimal =", round(
          result$h_optimal, 2)))
  lines(result$x_grid, result$y_estime, col = "red", lwd = 2)
}

# Affichage des erreurs de validation croisée pour chaque noyau
par(mfrow = c(1, 1))
plot(NA, xlim = range(h_values, finite = TRUE),
     ylim = range(unlist(lapply(results, function(r) r$errors)),
        finite = TRUE),
     xlab = "h", ylab = "Erreur de validation croisée",
     main = "Comparaison des noyaux")
for (kernel_name in names(results)) {
  lines(h_values, results[[kernel_name]]$errors, type = "l", lwd = 2,
        col = which(names(results) == kernel_name))
}
legend("topright", legend = names(results), col = 1:length(results),
   lwd = 2, cex = 0.8)
```

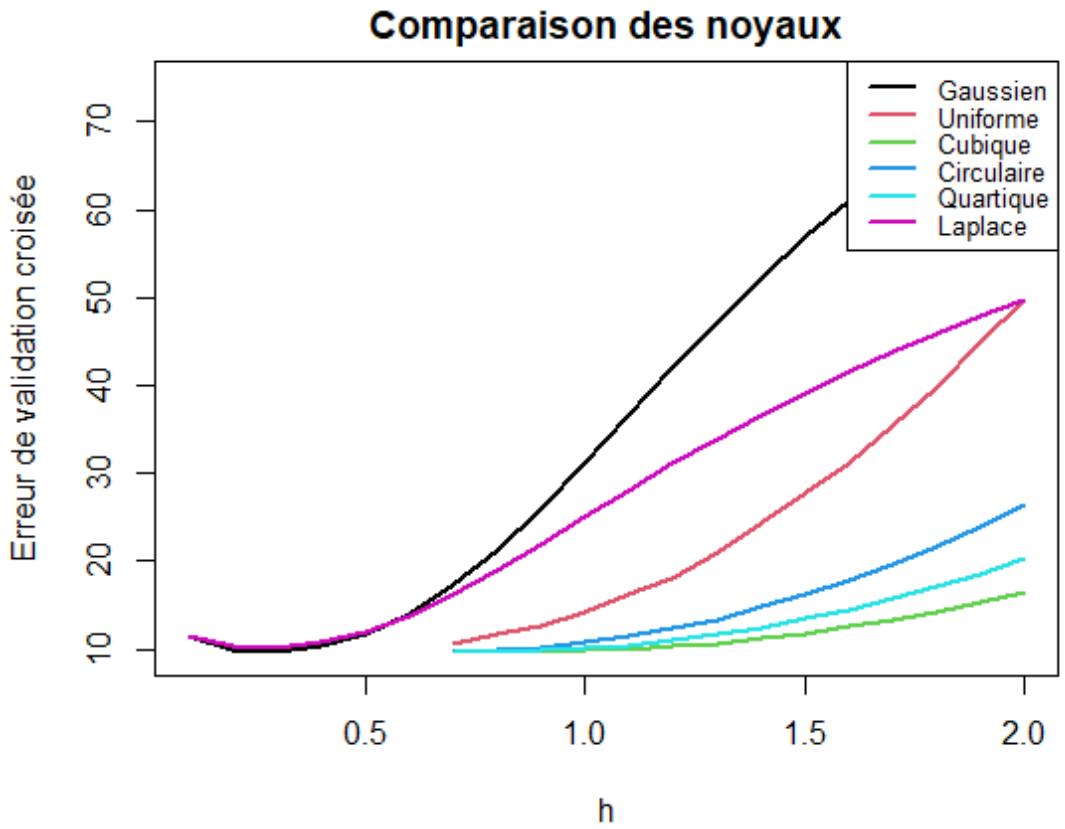

FIGURE 16 – Comparaison par différentes fenêtres

*** Interprétation :**

Ce graphique compare les performances de différents noyaux (Gaussien, Uniforme, Cubique, Circulaire, Quartique et Laplace) en fonction de la largeur de bande h, en mesurant l'erreur de validation croisée. L'erreur diminue initialement lorsque h augmente, atteint un minimum optimal, puis croît à nouveau à mesure que h devient trop grand, reflétant un compromis entre sur-ajustement (petit h) et sous-ajustement (grand h). Parmi les noyaux, le quartique (bleu foncé) et le cubique (vert) se distinguent par des erreurs minimales plus faibles, indiquant de meilleures performances globales, tandis que le Gaussien (noir) et le Laplace (rose) montrent des erreurs plus élevées pour des h plus grands. Ce graphique met en évidence que le choix de h et du noyau a un impact significatif sur la qualité de l'estimation.

5.2 Validation de l'algorithme sur les données mcycle

Les données **mcycle** font partie du package MASS dans R. Elles contiennent des mesures de données sur des accidents de moto. Ces données seront donc utilisées dans cette partie.

	times	accel
1	2.4	0.0
2	2.6	-1.3
3	3.2	-2.7
4	3.6	0.0
5	4.0	-2.7
6	6.2	-2.7
7	6.6	-2.7
8	6.8	-1.3
9	7.8	-2.7
10	8.2	-2.7
11	8.8	-1.3
12	8.8	-2.7

FIGURE 17 – Présentation de la base de donnée mcycle de "MAAS"

Le code suivant a pour objectif de comparer l'efficacité de différents noyaux pour l'estimation de la relation entre temps et accélération dans les données mcycle. Il identifie le noyau et la largeur de fenêtre optimale pour chaque cas, trace les ajustements obtenus pour chaque noyau et visualise la variation des erreurs de validation croisée en fonction de h. Cela permet d'évaluer les performances des noyaux et l'impact du choix de h sur l'estimation

```r
# Charger le package MASS pour accéder aux données mcycle
library(MASS)

# Charger les données mcycle
data(mcycle)
X <- mcycle$times
Y <- mcycle$accel

# Fonction de Nadaraya-Watson
Rn <- function(x, X, Y, h, kernel) {
  numerateur <- sum(kernel((x - X) / h) * Y)
  denominateur <- sum(kernel((x - X) / h))
  if (denominateur == 0) return(NA)
  return(numerateur / denominateur)
}

# Validation croisée pour trouver le h optimal
cv_error <- function(h, X, Y, kernel) {
  n <- length(Y)
  erreur <- 0
  for (i in 1:n) {
    # Laisser un point de c t é (leave-one-out)
    x_val <- X[i]
    y_val <- Y[i]
    X_train <- X[-i]
    Y_train <- Y[-i]

    # Prédiction pour ce point
    y_pred <- Rn(x_val, X_train, Y_train, h, kernel)
    erreur <- erreur + (y_val - y_pred)^2
  }
  return(erreur)
}

# Tester plusieurs valeurs de h
h_values <- seq(0.1, 5, by = 0.1)

# Résultats pour chaque noyau
results <- list()
for (kernel_name in names(kernels)) {
  kernel <- kernels[[kernel_name]]

  # Calcul des erreurs pour chaque h
  errors <- sapply(h_values, function(h) cv_error(h, X, Y, kernel))
  errors <- ifelse(is.finite(errors), errors, NA)  # Remplacer les non
```

```
    -finis par NA
  h_optimal <- h_values[which.min(errors)]

  # Calcul de l'estimation avec h optimal
  x_grid <- seq(min(X), max(X), length.out = 100)
  y_estime <- sapply(x_grid, function(x) Rn(x, X, Y, h_optimal, kernel
    ))

  # Stocker les résultats
  results[[kernel_name]] <- list(
    h_optimal = h_optimal,
    x_grid = x_grid,
    y_estime = y_estime,
    errors = errors
  )
}

# Tracer les ajustements pour chaque noyau
par(mfrow = c(2, 3), mar = c(4, 4, 2, 1))
for (kernel_name in names(results)) {
  result <- results[[kernel_name]]

  plot(X, Y, pch = 19, col = "blue", xlab = "Temps", ylab = "Accélé
    ration",
       main = paste("Noyau", kernel_name, "\n h optimal =", round(
         result$h_optimal, 2)))
  lines(result$x_grid, result$y_estime, col = "red", lwd = 2)
}

# Tracer les erreurs de validation croisée pour chaque noyau
par(mfrow = c(1, 1))
plot(NA, xlim = range(h_values, finite = TRUE),
     ylim = range(unlist(lapply(results, function(r) r$errors)),
       finite = TRUE),
     xlab = "h", ylab = "Erreur de validation croisée",
     main = "Comparaison des noyaux (données mcycle)")
for (kernel_name in names(results)) {
  lines(h_values, results[[kernel_name]]$errors, type = "l", lwd = 2,
        col = which(names(results) == kernel_name))
}
legend("topright", legend = names(results), col = 1:length(results),
   lwd = 2, cex = 0.8)
```

Nous obtenons le résultat suivant :

*** Interprétation :**

Les noyaux Gaussien et Quartique offrent des erreurs minimales plus faibles, indiquant qu'ils sont plus efficaces pour ces données. En revanche, les noyaux Uniforme, Circulaire et Laplace montrent des erreurs plus élevées, surtout pour des h plus grands, ce qui suggère qu'ils sont

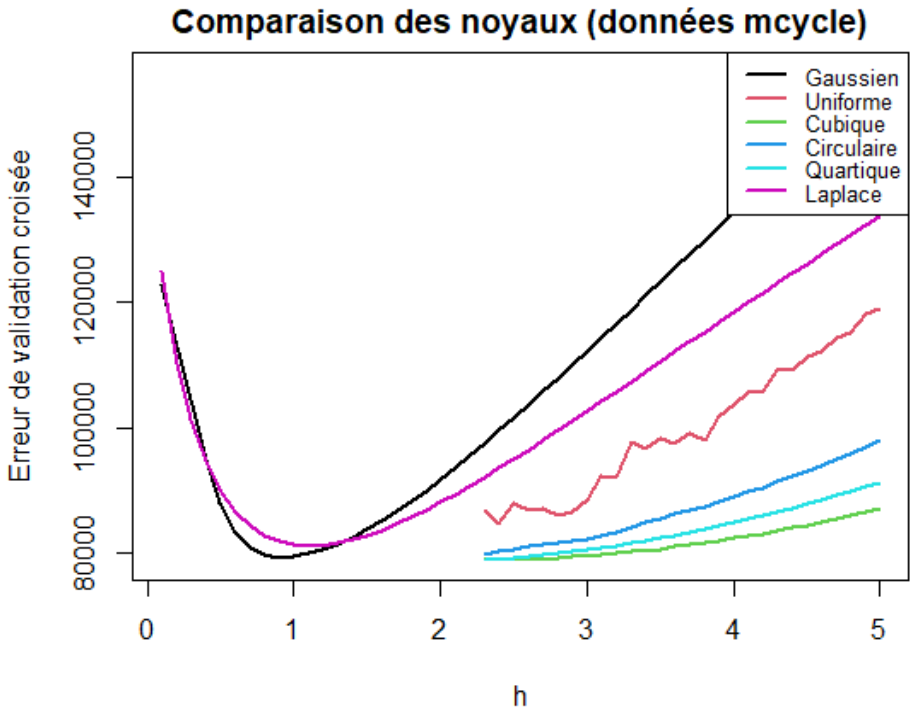

FIGURE 18 – Comparaison des noyaux pour mcycle

moins adaptés à cette application. Ce graphique illustre que le choix du noyau et de h peut significativement affecter la performance de l'estimation.

6 PROJET 1 : Validation croisée

6.1 Partie Théorie

Le score de validation croisée leave-one-out est défini par :

$$CV = \hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{r}^{(-i)}(x_i)\right)^2, \quad \text{avec } \hat{r}^{(-i)}(X_i) = \frac{\sum_{j\neq i} K\left(\frac{x_i - X_j}{h_n}\right) y_j}{\sum_{j\neq i} K\left(\frac{x_i - x_j}{h_n}\right)}.$$

où $\hat{r}^{(-i)}(x_i)$ est l'estimateur de r obtenu en enlevant les observations (x_i, y_i).
L'idée de la validation croisée leave-one-out vient du calcul suivant :

$$\mathbb{E}\left(\left(y_i - \hat{r}^{(-i)}(x_i)\right)^2\right) = \mathbb{E}\left(\left(y_i - r(x_i) + r(x_i) - \hat{r}^{(-i)}(x_i)\right)^2\right)$$

$$= \sigma^2 + \mathbb{E}\left((r(x_i) - \hat{r}^{(-i)}(x_i))^2\right)$$

car $y_i = r(x_i) + \epsilon$ avec $\epsilon \sim \mathcal{N}(0, \sigma^2)$

Donc

$$\mathbb{E}\left(\left(y_i - \hat{r}^{(-i)}(x_i)\right)^2\right) \simeq \sigma^2 + \mathbb{E}\left((r(x_i) - \hat{r}(x_i))^2\right).$$

On obtient donc $\mathbb{E}\left(\hat{R}(h)\right) \simeq \sigma^2 + R(h)$.

Le calcul de $\hat{R}(h)$ peut s'avérer long, mais dans certains cas, il n'est pas nécessaire de recalculer n fois un estimateur de la fonction de régression. Dans cette méthode, l'estimateur correspond à un algorithme de moyennes locales, c'est-à-dire est de la forme :

$$\hat{r}(x) = \sum_{j=1}^{n} y_j W_j(x),$$

avec $\sum_{j=1}^{n} W_j(x) = 1$. D'autre part :

$$\hat{r}^{(-i)}(x) = \sum_{j=1}^{n} y_j W_j^{(-i)}(x).$$

avec

$$W_j^{(-i)}(x) = \begin{cases} 0 & \text{si } j = i, \\ \frac{W_j(x)}{\sum_{k \neq i} W_k(x)} & \text{si } j \neq i. \end{cases}$$

6.2 Partie Pratique

Le code suivant implémente l'estimateur de Nadaraya-Watson avec un noyau gaussien pour ajuster une relation entre deux variables simulées (x et y) , et optimise la largeur de fenêtre h via la validation croisée leave-one-out (LOOCV).

```
#2
set.seed(123)
n <- 100
x <- rnorm(n, mean = 5, sd = 2)
y <- sin(x)  + rnorm(n, sd = 0.3)

# Nadaraya-Watson estimatoravec le noyau gaussien le noyau gaussien
kernel <- function(u) dnorm(u)

nadaraya_watson <- function(x, y, h, x_eval) {
  weights <- kernel((x_eval - x) / h)
  weights <- weights / sum(weights)
  sum(weights * y)
}

#  LOOCV pour Nadaraya-Watson
loocv <- function(x, y, h) {
  n <- length(x)
  cv_error <- 0

  for (i in 1:n) {
    # Exclure le point (x[i], y[i]) et calculer l'estimateur
    x_minus_i <- x[-i]
    y_minus_i <- y[-i]
    y_pred <- nadaraya_watson(x_minus_i, y_minus_i, h, x[i])

    # Calculer l'erreur quadratique pour le point exclu
```

```
    cv_error <- cv_error + (y[i] - y_pred)^2
  }

  return(cv_error / n)
}

# Déterminer hCV via LOOCV
h_values <- seq(0.1, 2, by = 0.05)  # Grille de valeurs pour h
cv_errors <- sapply(h_values, function(h) loocv(x, y, h))  # Calcul
   des erreurs pour chaque h

# Trouver la meilleure valeur de h
h_cv <- h_values[which.min(cv_errors)]
cat("Valeur optimale de h (hCV) :", h_cv, "\n")

# ---- Visualisation ----
plot(h_values, cv_errors, type = "l", col = "blue", lwd = 2,
     xlab = "Valeur de h", ylab = "Erreur de validation croisée (LOOCV
        )",
     main = "Optimisation de h via LOOCV")
abline(v = h_cv, col = "red", lty = 2, lwd = 2)
```

Nous obtenons le résultat que la Valeur optimale de h (hCV) est 0.3

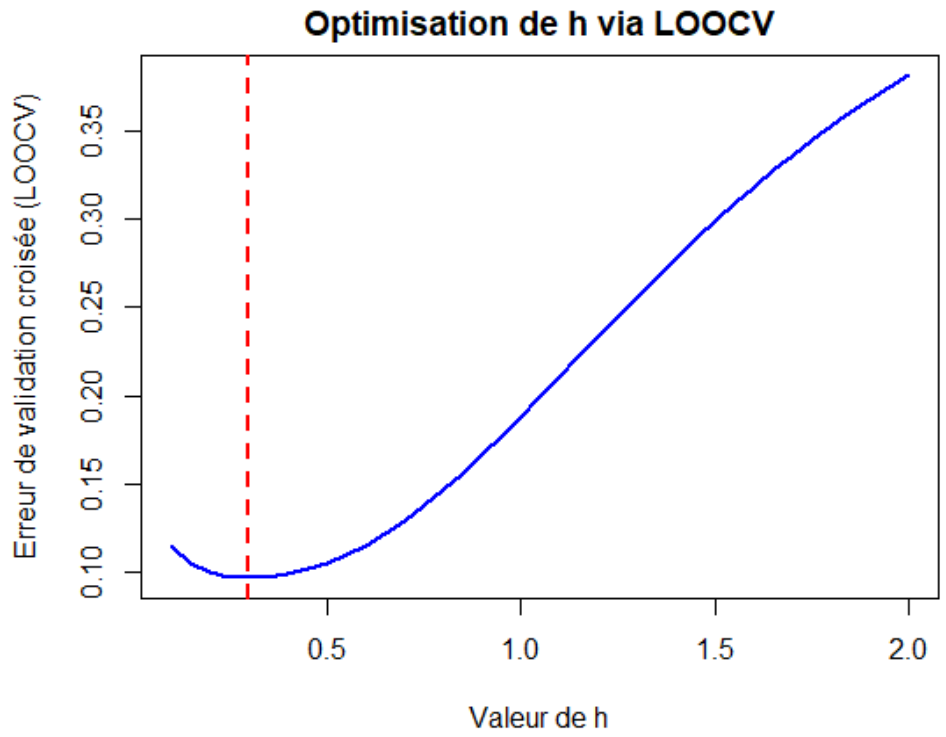

FIGURE 19 – Optimisation de la fenêtre par la validation croisée

*** Interprétation :**

Le graphique montre l'erreur de validation croisée leave-one-out (LOOCV) en fonction de la

largeur de fenêtre h dans l'estimateur de Nadaraya-Watson. La courbe bleue illustre la relation entre h et l'erreur quadratique moyenne : pour de petites valeurs de h, l'erreur est relativement élevée à cause de la variance élevée (lissage insuffisant), tandis que pour de grandes valeurs de h, l'erreur augmente également à cause d'un biais croissant (lissage excessif). La largeur de fenêtre optimale h_{CV}, indiquée par la ligne verticale rouge, est celle qui minimise l'erreur de validation croisée.

6.2.1 Données Réelles

Le jeu de données **cars** est un jeu de données intégré à R. Il contient des informations sur la relation entre la vitesse d'un véhicule et la distance de freinage.

	speed	dist
1	4	2
2	4	10
3	7	4
4	7	22
5	8	16
6	9	10
7	10	18
8	10	26
9	10	34
10	11	17
11	11	28
12	12	14

FIGURE 20 – Jeu de donnée "cars"

Le code suivant utilise les données **cars** pour illustrer l'ajustement non paramétrique entre la vitesse et la distance de freinage. La largeur de fenêtre optimale h_{CV}, déterminée par vali-

dation croisée, équilibre biais et variance pour donner une estimation lisse et précise.

```
# Validation sur des données réelles
# Charger un jeu de données réel
data(cars)
x_real <- cars$speed
y_real <- cars$dist

# Répéter la procédure pour les données réelles
cv_errors_real <- sapply(h_values, function(h) loocv(x_real, y_real, h
    ))
h_cv_real <- h_values[which.min(cv_errors_real)]
cat("Valeur optimale de h pour les données réelles :", h_cv_real, "\n"
    )

# Estimer la fonction avec le h optimal pour les données réelles
x_eval_real <- seq(min(x_real), max(x_real), length.out = 200)
y_pred_real <- sapply(x_eval_real, function(xe) nadaraya_watson(x_real
    , y_real, h_cv_real, xe))

# Visualiser les résultats sur les données réelles
plot(x_real, y_real, pch = 16, col = "gray", main = "Estimation par
    Nadaraya-Watson (Données réelles)",
      xlab = "Vitesse", ylab = "Distance de freinage")
lines(x_eval_real, y_pred_real, col = "blue", lwd = 2)
legend("topright", legend = c("Données", "Estimation"),
       col = c("gray", "blue"), lty = c(NA, 1), pch = c(16, NA))
```

Nous obtenons que la valeur optimale de h pour les données réelles est 1.65

*** Interprétation :**

Le graphique final montre comment l'estimateur de Nadaraya-Watson capture la relation entre la vitesse et la distance de freinage, en suivant globalement la tendance des données tout en atténuant les variations dues au bruit.

6.2.2 Comparaison de méthodes

Le code compare deux méthodes pour déterminer la largeur de fenêtre h dans l'estimateur de Nadaraya-Watson : h_{WISE} , calculé analytiquement à partir d'une approximation théorique, et h_{CV}, obtenu par validation croisée leave-one-out. Il utilise ces deux valeurs pour estimer une relation entre x et y et visualise les résultats.

```
n <- 100
sigma <- 0.3
kernel_variance <- 1
x_eval <- seq(min(x), max(x), length.out = 200)
f <- function(x) sin(x)  # Vraie fonction
```

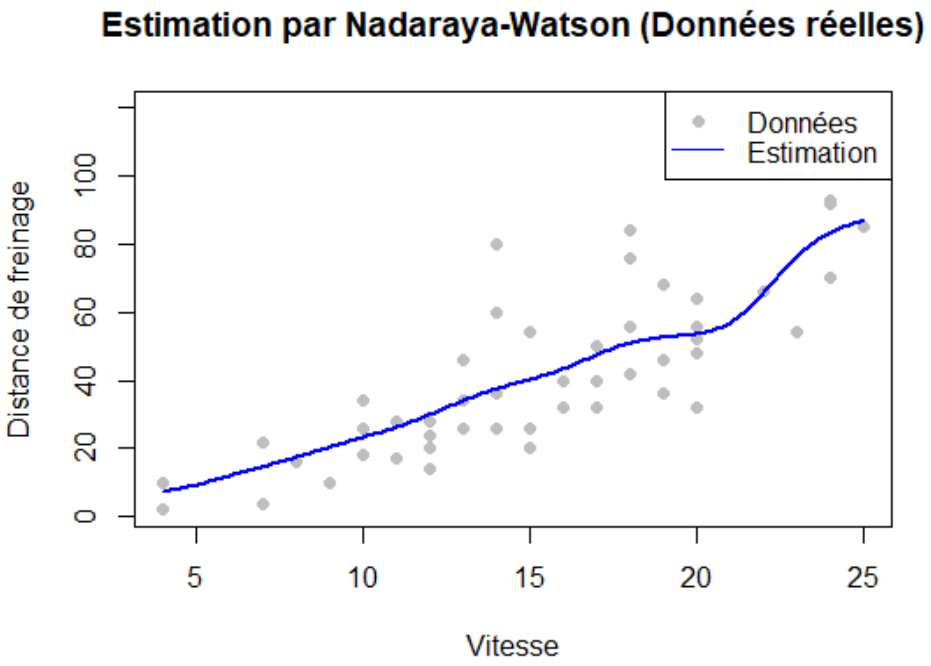

FIGURE 21 – Estimation par Nadaraya-Watson pour cars

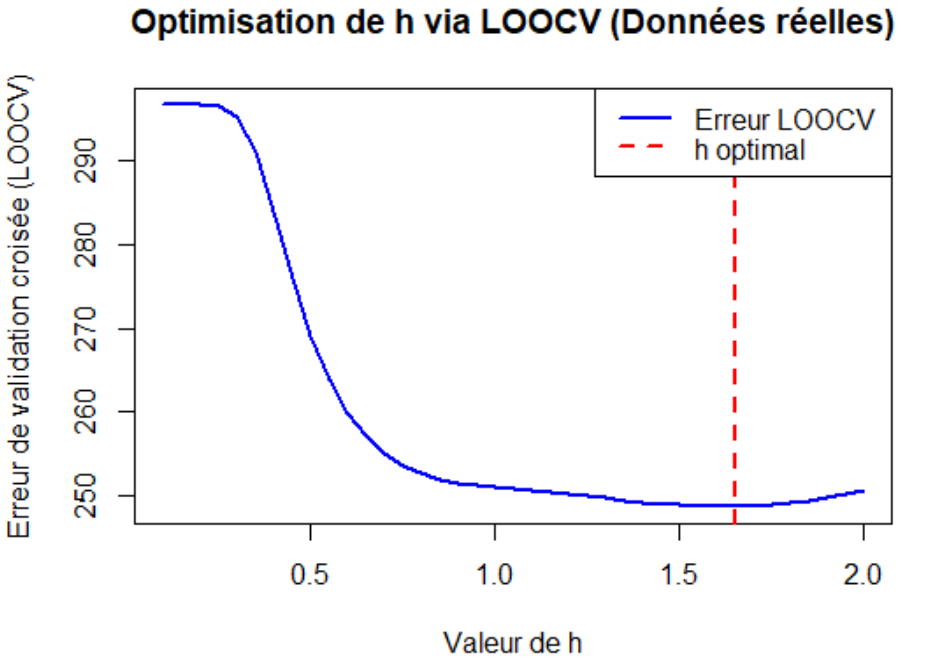

FIGURE 22 – Optimisation de la fenêtre par la validation croisée

```
#Fonction pour calculer h_MISE
h_mise <- function(n, sigma, kernel_variance) {
  # Approximation théorique pour le noyau gaussien
```

```
  C <- (4 * sqrt(pi) * kernel_variance)^(1/5)
  h <- C * sigma^(2/5) * n^(-1/5)
  return(h)
}

#Calculer h_MISE
h_mise_value <- h_mise(n = n, sigma = sigma, kernel_variance = kernel_
   variance)

# Comparer h_MISE et h_CV
cat("Valeur de h_MISE :", h_mise_value, "\n")
cat("Valeur de h_CV (par validation croisée) :", h_cv, "\n")

# Visualisation des courbes estimées
# Estimation avec h_MISE
y_pred_mise <- sapply(x_eval, function(xe) nadaraya_watson(x, y, h_
   mise_value, xe))
# Estimation avec h_CV
y_pred_cv <- sapply(x_eval, function(xe) nadaraya_watson(x, y, h_cv,
   xe))

# Tracer les courbes
plot(x, y, pch = 16, col = "gray", main = "Comparaison entre h_MISE et
    h_CV",
     xlab = "x", ylab = "y")
lines(x_eval, f(x_eval), col = "green", lwd = 2, lty = 2)
lines(x_eval, y_pred_mise, col = "red", lwd = 2, lty = 1)
lines(x_eval, y_pred_cv, col = "blue", lwd = 2, lty = 1)
legend("bottomright", legend = c("Données", "Vraie fonction", "
   Estimation (h_MISE)", "Estimation (h_CV)"),
       col = c("gray", "green", "red", "blue"), lty = c(NA, 2, 1, 1),
          pch = c(16, NA, NA, NA),cex = 0.8)
```

Nous obtenons les résultats suivants :

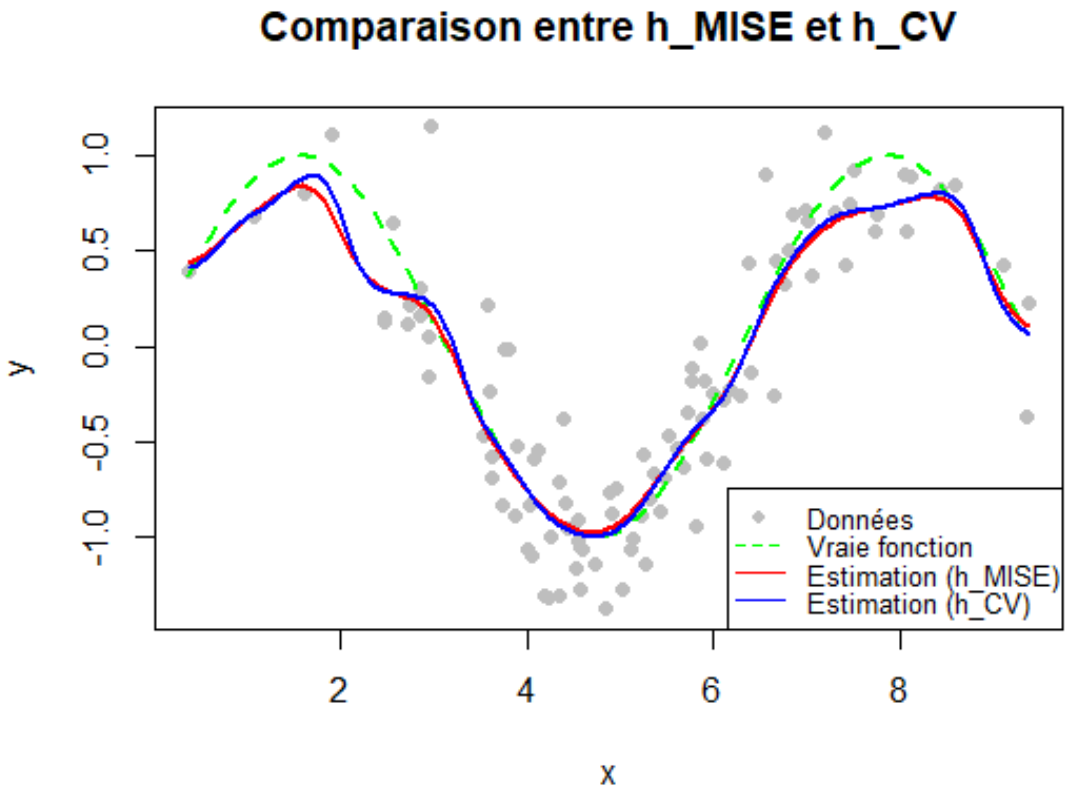

FIGURE 23 – Comparaison entre h_{WISE} et h_{CV}

*** Interprétation :**
Ce graphique compare deux estimations par l'estimateur de Nadaraya-Watson en utilisant deux largeurs de fenêtre différentes : h_{WISE} (rouge), calculé analytiquement à partir d'une approximation théorique, et h_{CV} (bleu), obtenu par validation croisée leave-one-out. Les points gris représentent les données simulées (y=sin(x)+bruit), et la courbe verte en tirets indique la vraie fonction sous-jacente. Les deux estimations suivent globalement la tendance de la vraie fonction, mais h_{CV} s'ajuste légèrement mieux aux variations des données, capturant davantage de détails. En revanche,h_{WISE} produit une estimation plus lissée, indiquant un compromis légèrement plus biaisé, mais potentiellement plus robuste au bruit. Ce graphique illustre le compromis entre lissage et ajustement précis dans le choix de la largeur de fenêtre.

6.3 Comparaison des EQM des estimateurs de Parzen-Rosenblatt et de Nadaraya-Watson

Le code simule un ensemble de données (X,Y), où Y=sin(X)+bruit, et compare les performances de deux estimateurs : Parzen-Rosenblatt et Nadaraya-Watson. Il génère des prédictions sur une grille de valeurs de x et calcule l'erreur quadratique moyenne (EQM) pour évaluer la fidélité des deux estimateurs par rapport à la vraie fonction sin(X).

```
# Simulation des données
set.seed(123)
```

```r
n <- 100
X <- X <- rnorm(n)
Y <- sin(X) + rnorm(n, sd = 0.3)
# Grille pour les prédictions
x_grid <- seq(-3, 3, length.out = 200)

# Estimateur de Parzen-Rosenblatt
parzen_rosenblatt <- function(x, X, Y, h) {
  f_hat <- sapply(x, function(x_val) {
    weights <- dnorm((x_val - X) / h) / h
    sum(weights * Y) / length(Y)
  })
  return(f_hat)
}

#Estimateur de Nadaraya-Watson
nadaraya_watson <- function(x, X, Y, h) {
  f_hat <- sapply(x, function(x_val) {
    weights <- dnorm((x_val - X) / h) / h
    sum(weights * Y) / sum(weights)
  })
  return(f_hat)
}

#Calcul des prédictions
h_parzen <- 0.7
h_nadaraya <- 0.5
y_pred_parzen <- parzen_rosenblatt(x_grid, X, Y, h_parzen)
y_pred_nadaraya <- nadaraya_watson(x_grid, X, Y, h_nadaraya)

#Calcul de l'EQM
y_true <- sin(x_grid)  # La vraie fonction
EQM_parzen <- mean((y_true - y_pred_parzen)^2)
EQM_nadaraya <- mean((y_true - y_pred_nadaraya)^2)

# Résultats
cat("EQM de Parzen-Rosenblatt :", EQM_parzen, "\n")
cat("EQM de Nadaraya-Watson :", EQM_nadaraya, "\n")

# Visualisation
plot(X, Y, pch = 16, col = "gray", main = "Comparaison des Estimateurs
    ",
     xlab = "X", ylab = "Y")
lines(x_grid, y_true, col = "black", lwd = 2, lty = 2)
lines(x_grid, y_pred_parzen, col = "blue", lwd = 3, lty = 1)
lines(x_grid, y_pred_nadaraya, col = "red", lwd = 2, lty = 1)
legend("bottomright", legend = c("Données", "Vraie fonction", "Parzen-
   Rosenblatt", "Nadaraya-Watson"),
       col = c("gray", "black", "blue", "red"), lty = c(NA, 2, 1, 1),
          pch = c(16, NA, NA, NA))
```

Les erreurs quadratiques moyennes (EQM) des deux estimateurs sont les suivantes :

EQM de Parzen-Rosenblatt : 0.4246

EQM de Nadaraya-Watson : 0.0660

Ainsi, l'estimateur de Nadaraya-Watson montre une meilleure performance avec une erreur plus faible.

Nous obtenons le graphique suivant :

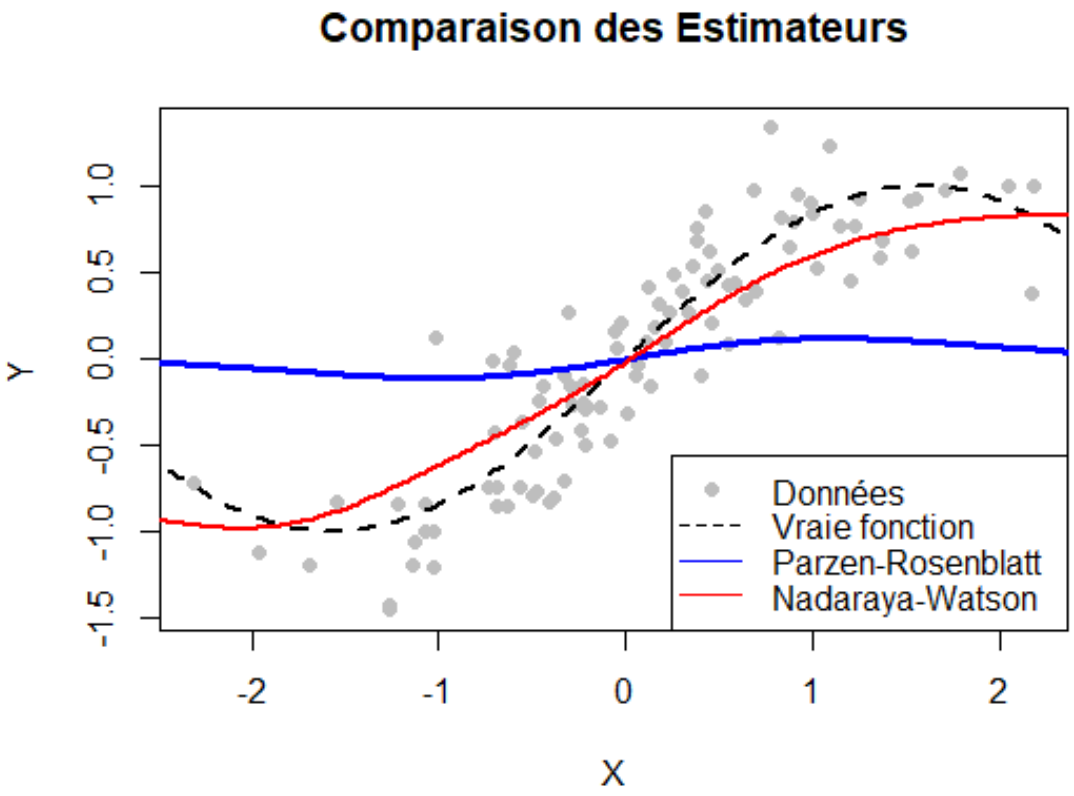

FIGURE 24 – Comparaison des Estimateurs

*** Interprétation :**
Ce graphique compare les performances des deux estimateurs : Parzen-Rosenblatt (en bleu) et Nadaraya-Watson (en rouge), en les confrontant à la vraie fonction (Y=sin(X)), représentée par la courbe en tirets noirs). Les points gris illustrent les données simulées, bruitées autour de la vraie fonction. L'estimateur de Nadaraya-Watson suit de manière plus fidèle la structure non linéaire de la vraie fonction tout en capturant les tendances locales des données. En revanche, l'estimateur de Parzen-Rosenblatt, avec une moyenne plus globale, fournit une estimation très lisse, insensible aux variations locales des données, mais moins adaptée à la régression. Cela montre que Nadaraya-Watson est plus performant pour des tâches de régression, tandis que Parzen-Rosenblatt est davantage orienté vers une estimation globale et lisse.

7 Conclusion

Ce projet m'a permis de mieux comprendre les méthodes de sélection de fenêtre, les choix théorique, pratique et optimal en utilisant l'erreur quadratique moyenne et la validation croisée. Une connaisannce théorique du calcul de ces notions ont été dévoloppée lors de la mise en place de ce projet. Que dire si ce n'est de remercier nos professeurs notement le chargé de cet Ue pour ces connaisannces qui me serviront certainement dans le domaine professionnel.