Master 1 Mathématiques Appliquées et Statistique **2021-2022**
parcours MSS et CMI

TP2 - Statistique Non Paramétrique

Estimation non paramétrique d'une densité

Le rapport est à rendre avant le mercredi prochain (13 Avril).

Exercise 1 **Estimateur histogramme**

(a) Construire l'estimateur histogramme usuel pour une densité supportée sur [0,2], en complétant le code ci-dessous.

```
> # construire l'estimateur histogramme comme une fonction de N, de l'échantillon et de x
> # ou N est le nombre de classes / intervalles / bins
> # et x est un point dans [0,2]
> estimateur_hist <- function(x, N, ech){
+   n <- length(ech)
+   # découper [0, 2] en N classes de même taille (usuel), la fenêtre h ainsi = 2/N
+   breakpoints <- seq(0, 2, length.out = N + 1)
+   h = 2/N # la fenêtre
+   # calculer la valeur de l'éstimateur histogramme en point x: <<f_chap_x>>
+   for (i in 1:N){
+     # réperer la bonne classe <<i>> ou se trouve x
+     if(x >= breakpoints[i] & x < breakpoints[i+1]){
+       # calculer la fréquence du nb de obs qui tombent dans la classe <<i>>
+
+
+       f_chap_x = sum(ech >= breakpoints[i] & ech < breakpoints[i+1])/n/h
+
+
+       break
+     }
+   }
+   return(f_chap_x)
+ }
```

(b) Utiliser le code ci-dessous pour simuler un échantillon i.i.d de taille $n = 200$ de la loi de mélange

des deux gaussiens tronquées sur le même intervalle compact $[0, 2]$:

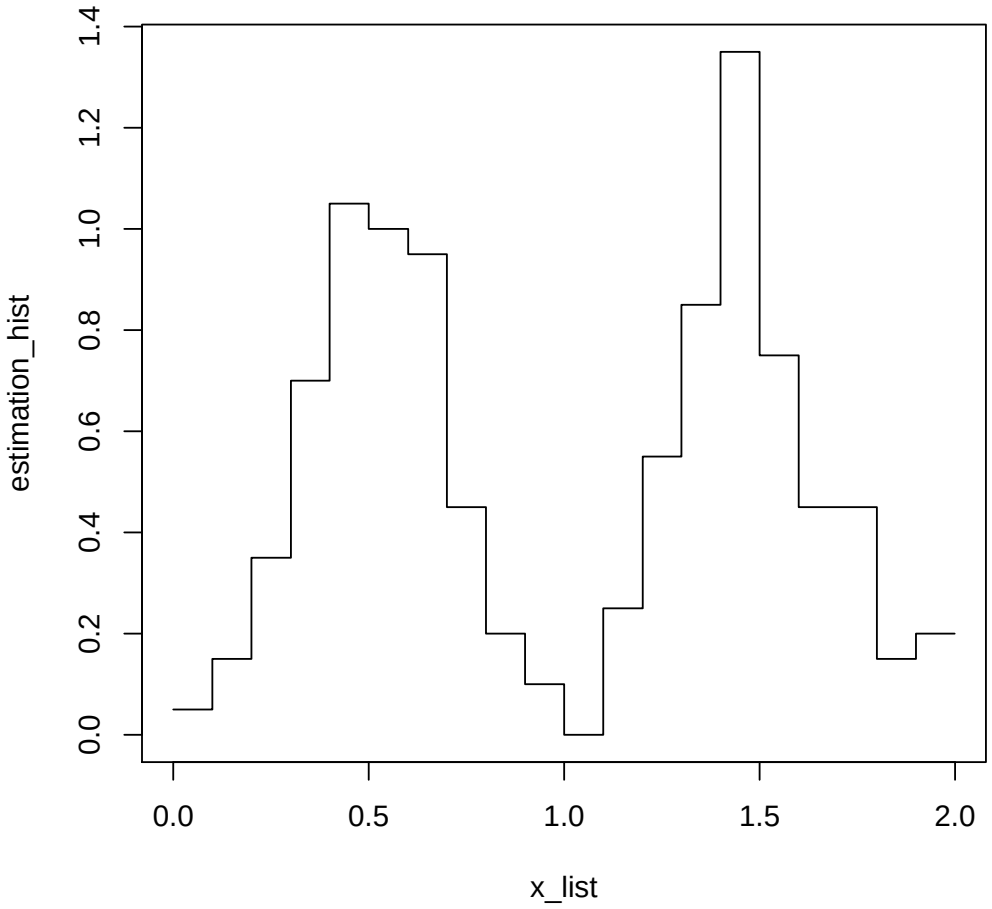
$$\frac{1}{2}\mathcal{N}(0.5, 0.04, 0, 2) + \frac{0.5}{2}\mathcal{N}(1.5, 0.04, 0, 2),$$

et appliquer l'estimateur crée avec $N = 10$ puis 200. On donne également le code pour tracer les histogrammes (densités estimées).

https ://en.wikipedia.org/wiki/Truncated_normal_distribution

https ://fr.wikipedia.org/wiki/Loi_de_mélange

```
> #install.packages(truncnorm)
> library(truncnorm)
> set.seed(1)
> ech_mgt <- c(rtruncnorm(100,a = 0,b = 2,mean=0.5,sd = 0.2),
+              rtruncnorm(100,a = 0,b = 2,mean=1.5,sd = 0.2))
> x_list <- seq(0,1.999,length.out = 2000)
> estimation_hist <- sapply(x_list, function(x_){
+                                                   return(estimateur_hist(x_,20,ech_mgt))
+                                                 }
+                           )
> plot(x_list,estimation_hist, type="s")
```



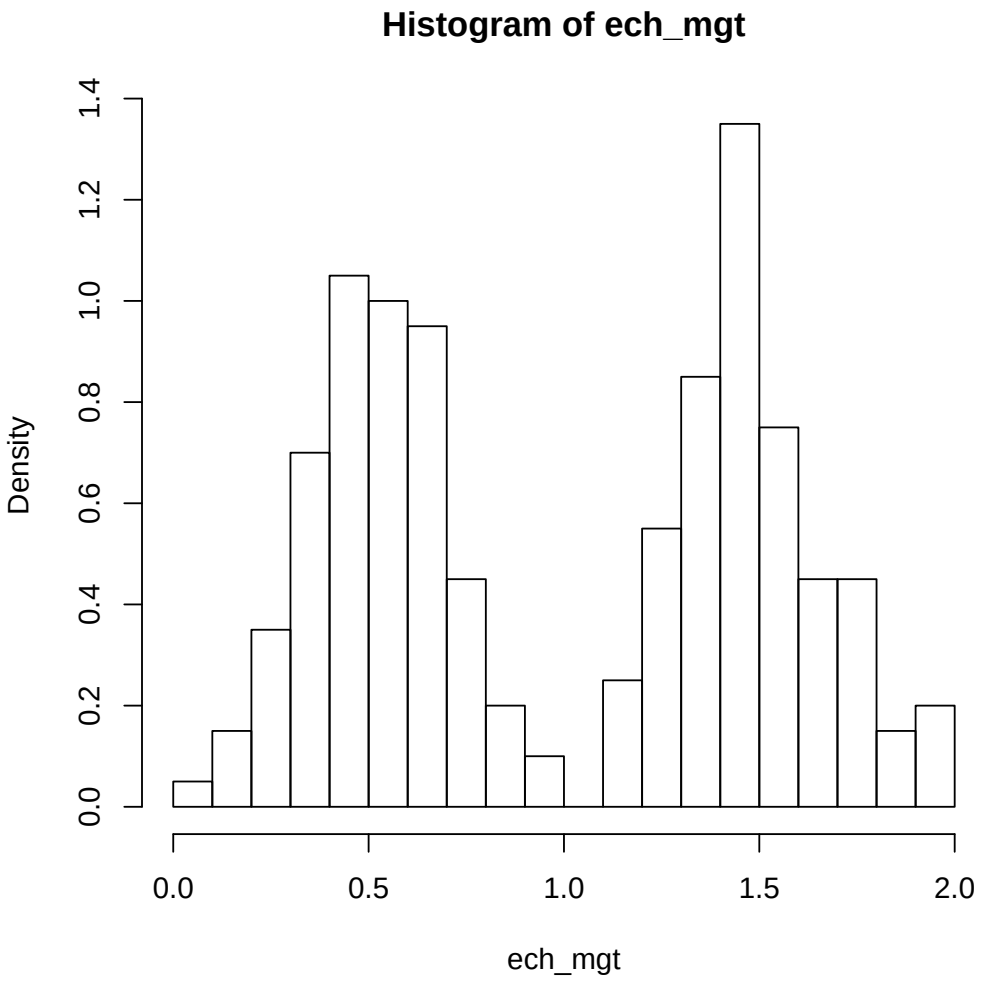
Une grille à 200 classes est trop fine pour obtenir une bonne éstimée.

(c) Retrouver ces résultats avec la fonction de R `hist`().

```
> hist(ech_mgt, breaks =, freq =)
```

(à cacher)

```
> hist(ech_mgt, breaks = 20, freq =F)
> #ou
> hist(ech_mgt, breaks = seq(0, 2, length.out = 20 + 1), freq =F)
> hist(ech_mgt, breaks = 200, freq =F)
```



`freq` précise la fréquence absolue (`freq` = T) ou la fréquence relative (`freq` = F).

Exercise 2 **Estimateur à noyau**

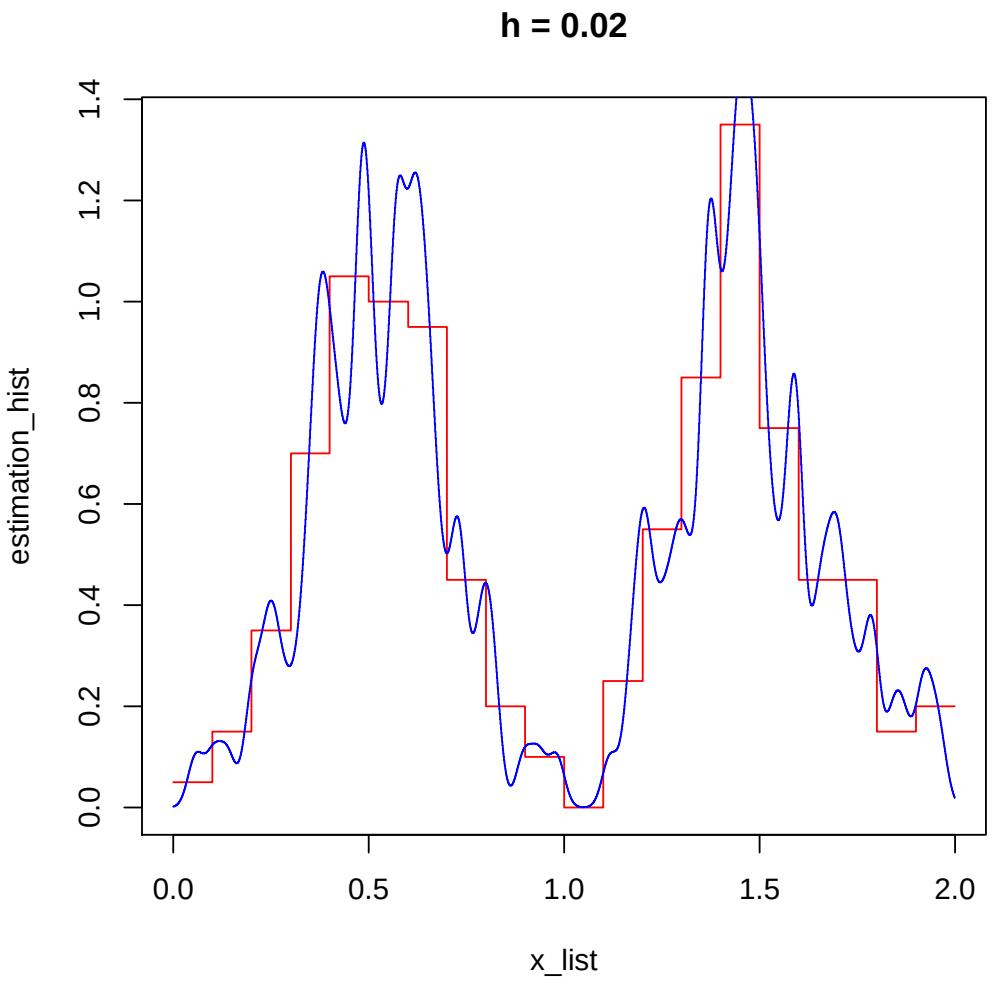
(a) Construire un estimateur à noyau avec le noyau Gaussien en complètant le code ci-dessous.

```
> #construire l'estimateur à noyau Gassien comme une fonction de h, de l'échantillon et d
> #NB: la fenêtre h n'est pas le même hyper-paramètre qu'avant, qui au lieu contrôle les
> estimateur_noyau_Gassien <- function(x, h, ech){
+   n <- length(ech)
+   # calculer la valeur de l'éstimateur à noyau en point x: <<f_chap_x>>
+   int_K <- (ech - x)/h
+
+
+   f_chap_x <- sum(dnorm(int_K))/n/h
+   #ou
+   f_chap_x <- sum(exp(-int_K^2/2))/sqrt(2*pi)/n/h
+
+
```

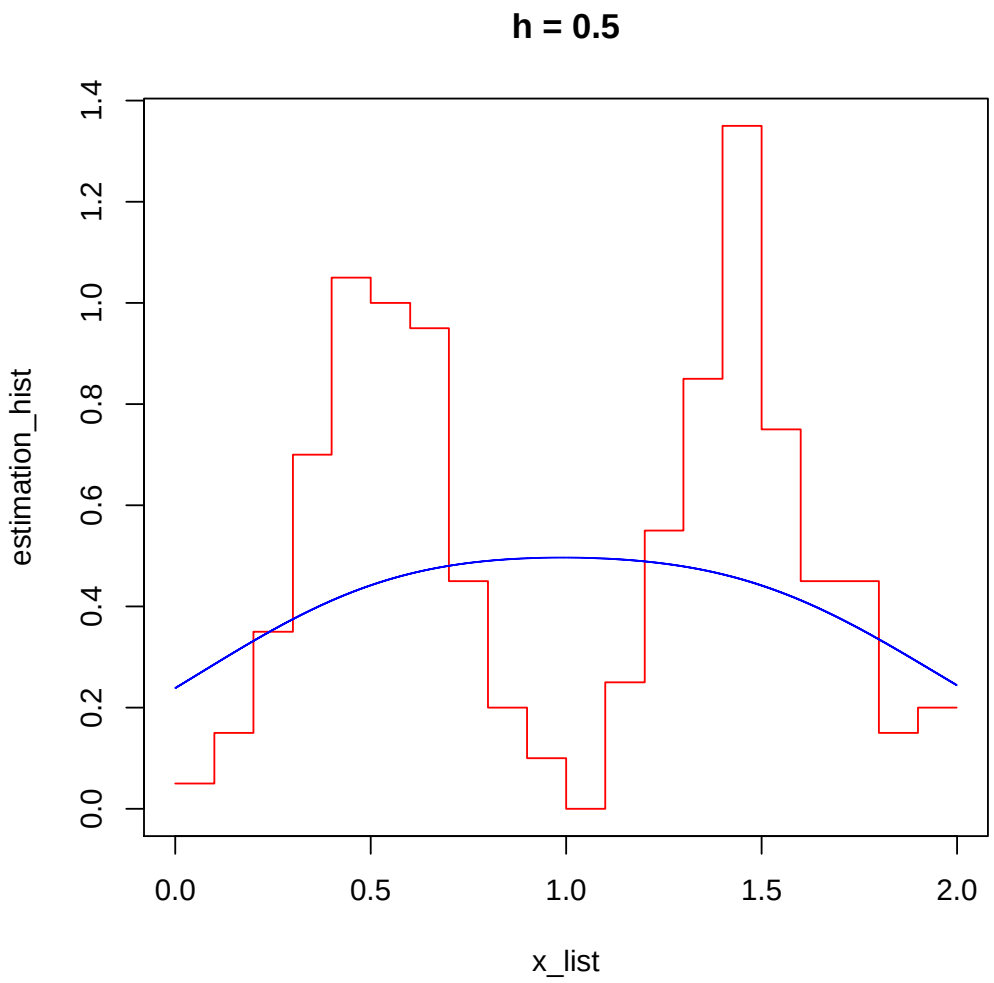
```
+   return(f_chap_x)
+ }
```

(b) Appliquer l'estimateur à noyau Gaussien à l'échantillon de l'exercise, avec h = 0.02, 0.05, 0.09, 0.2, 0.5, et tracer les densités éstimées (superposer l'histogramme à $N = 20$). Que pensez vous de l'impact de la valeur de h sur l'estimateur ? Quelle est selons vous la bonne valeur de h ?

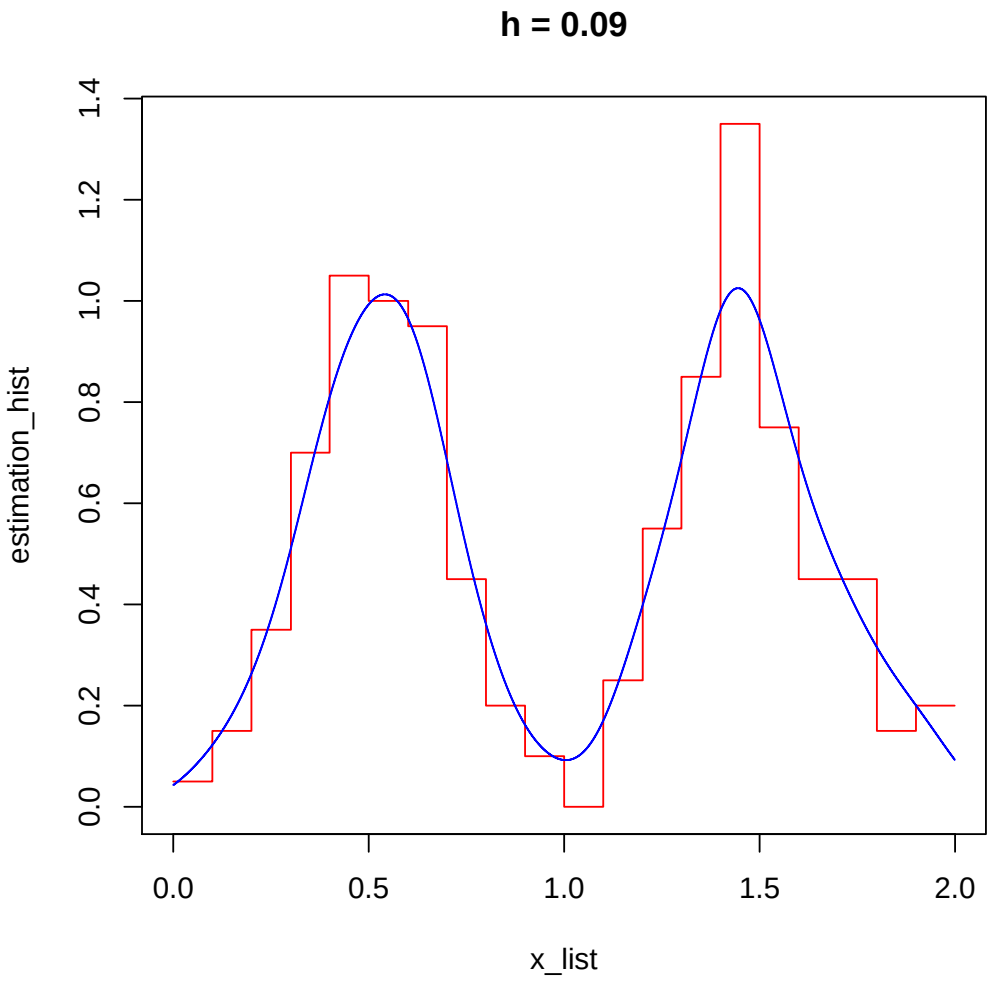
```
> x_list <- seq(0,1.999,length.out = 2000)
> estimation_noyau_Gassien <-
+   sapply(x_list, function(x_){return(estimateur_noyau_Gassien(x_,h=0.02,ech_mgt))})
> plot(x_list,estimation_hist, type="s",col="red", main = "h = 0.02")
> lines(x_list,estimation_noyau_Gassien, type="s",col="blue")
```




```
> estimation_noyau_Gassien <-
+   sapply(x_list, function(x_){return(estimateur_noyau_Gassien(x_,h=0.5,ech_mgt))})
> plot(x_list,estimation_hist, type="s",col="red", main = "h = 0.5")
> lines(x_list,estimation_noyau_Gassien, type="s",col="blue")
```




```
> estimation_noyau_Gassien <-
+   sapply(x_list, function(x_){return(estimateur_noyau_Gassien(x_,h=0.09,ech_mgt))})
> plot(x_list,estimation_hist, type="s",col="red", main = "h = 0.09")
> lines(x_list,estimation_noyau_Gassien, type="s",col="blue")
```



Plus grand h, plus de poids sur les obs. plus loins de x, plus lissé l'éstimateur sera par le noyau.

Surtout quand h est trop petit, l'éstimateur $\hat{f}(x)$ est dominé par quelques obs. dans un tout petit voisinage de x, la regularié global dans les données n'est pas considèré par $\hat{f}$, donc $\hat{f}$ est très volatile, ce qu'on apelle **sous-lissage**, e.g. $h = 0.02$.

Par contre, quand h est trop grand, $\hat{f}(x)$ considère que un grand voisinage de x, donc la proprété locale est effacée, ce qu'on apelle **sur-lissage**, e.g. $h = 0.5$.

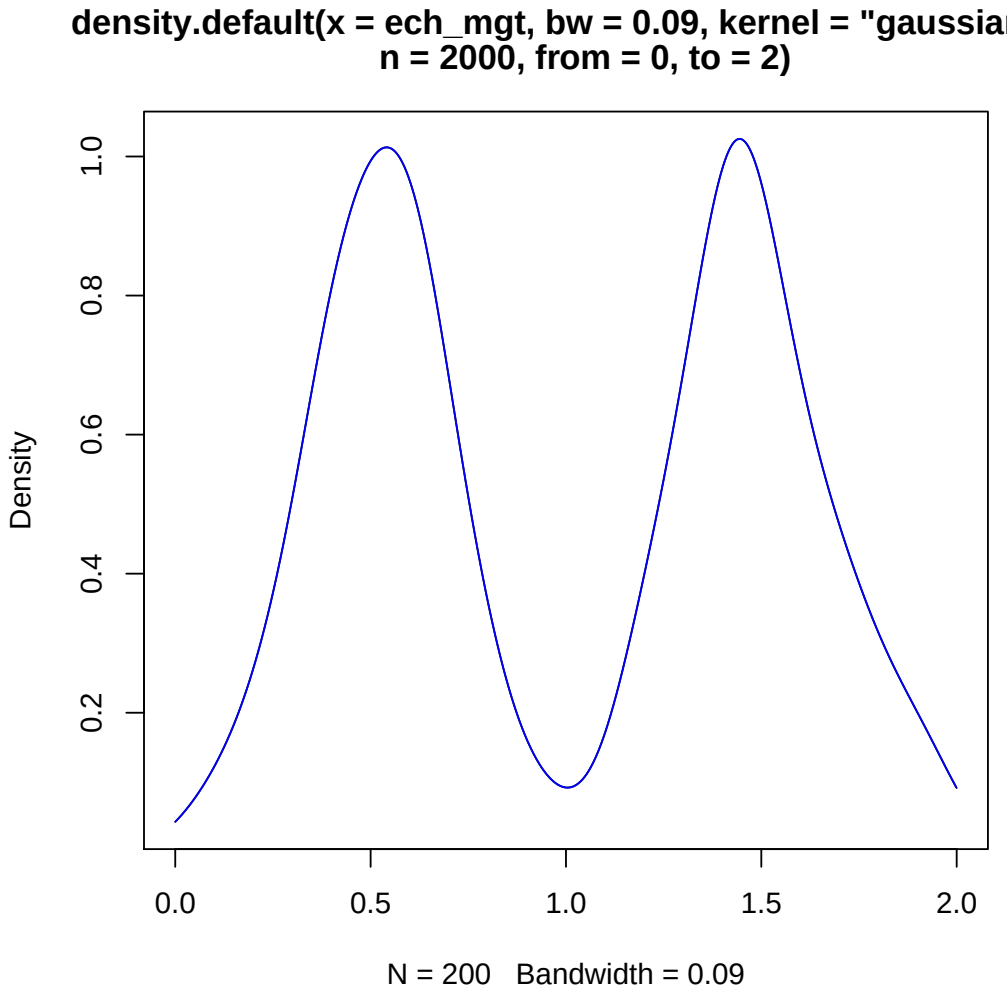
h optimal est 0.09 parmi les 5.

(c) Comparer avec les densités éstimées par la fonction de R `density()`.

```
> install.packages("kdensity")
> estimation_density <- density(ech_mgt, kernel = , bw =, n = 2000, from = 0, to = 2)
> plot(estimation_density,col="black")
```

(à cacher)

```
> estimation_density <- density(ech_mgt, kernel = "gaussian", bw = 0.09, n = 2000, from =
> plot(estimation_density,col="black")
> lines(x_list, estimation_noyau_Gassien,col="blue")
```



Exercise 3 **Sélection de la fenêtre h : la validation croisée (Leave One Out)**

On va maintenant chercher une approche qui permet de choisir le hypermaramètre h en autonomie, et puis la programmer.

Le théorie dans le Section 4.3.6 nous implique une telle approche, qui propose le minimiseur du critètre EQMI empirique, $\hat{J}_n(h; X_1, ..., X_n)$, comme la valeur optimale de h. On reprend les notations du cours (Voir Chap. 3 page 11). Notez qu'étant donnée un échantillon $X_1, ..., X_n$, les estimateurs U_n et V_n sont des fonctions de h, qu'il faudrait donc rigoureusement écrire $U_n(h; X_1, ..., X_n)$ et $V_n(h; X_1, ..., X_n)$.

Dans la suite, on va créer la fonction $\hat{J}_n(h; X_1, ..., X_n)$ en 2 étapes. La question (a), (b) vous guide respectivement de créer $U_n(h; X_1, ..., X_n)$ et $V_n(h; X_1, ..., X_n)$. Finalement, dans la question c), on minimise la $\hat{J}_n(h; X_1, ..., X_n)$ créé pour l'échantillon précedant.

(a) Créer une fonction de R qui prend la fenêtre h et l'échantillon $X_1, ..., X_n$ comme les rentrées et sort $U_n(h; X_1, ..., X_n)$, en complètant le code ci-dessous.

NB : On pourra utiliser les formules suivantes :

$$\hat{f}_{h,-i}(X_i) = \frac{n}{n-1}\left[\hat{f}_h(X_i) - \frac{K(0)}{nh}\right] \tag{0.1}$$

$$\hat{U}_n(h; X_1, ..., X_n) = \frac{1}{n-1} \sum_{i=1}^{n} \hat{f}_h(X_i) - \frac{K(0)}{(n-1)h} \tag{0.2}$$

```
> U_n <- function(h, ech){
+   n <- length(ech)
+   # 1ère terme
+
+   sum_ <- sum(sapply(ech,
+                      function(x_){
+                        return(estimateur_noyau_Gassien(x_,h,ech))
+                        }
+                      )
+               ) #one-liner: dans R privilégier les opérations vectorielles, matricielle
+                 #au lieu de faire des boucles avec le code lourd.
+
+   sum_ <- sum_/(n-1)
+   # 2ème terme
+
+   K0 <- dnorm(0)
+
+   K0 <- K0/(n-1)/h
+   return(sum_ -  K0)
+ }
```

(b) Le code ci-dessous crée une fonction permet de calculer $V_n(h; X_1, ..., X_n)$.

NB : En pratique, il faut utiliser la methode numerique pour approacher le calcul comme l'integral. Dans R, la fonction qui calcule l'integral d'une fonction univariable est `integrate`.

```
> V_n <- function(h, ech){
+   n <- length(ech)
+   # Fonction wrapper:
+   # convertir la rentrée et la sortie de la estimateur_noyau_Gassien(x_,h,ech) en vecte
+   # fixer les 2eme 3eme arguments
+   fsq <- function(x){
+     return(sapply(x, function(x_) return(estimateur_noyau_Gassien(x_,h,ech)^2)))
+   }
+   integral_fsq <- integrate(fsq, lower=0, upper =2)$value
+   return(integral_fsq)
+ }
```

(c) Construire $\hat{J}_n(h; X_1, ..., X_n)$, et tracer la courbe en fonction de h, avec l'échantillon donnée celui de les exercises précedants, pour repérer son minimum.

```
> J_n <- function(h, ech){
+   return(V_n(h, ech) - 2*U_n(h, ech))
```

```
+ }
> h_list <- seq(0.01, 2, 0.01)
> val_J_n <- sapply(h_list, function(h_) J_n(h_, ech_mgt))
> plot(h_list, val_J_n, type = "s")
> # h optimal
> h_list[which.min(val_J_n)]
[1] 0.09
```