Executive Master : Régression non-paramétrique

Devoir maison obligatoire (Dauphine)

Paul Hardouin

September 16th, 2019

Contents

1. Etude de la densité g des X

Pour cette première partie, on utilise la première colonne **X** des données **data1.csv**

1.1 Construction d'un estimateur non paramétrique de g(x)

==
Construire un estimateur non paramétrique $\hat{g}_{n,h}$(x) de g(x) pour une fenêtre de lissage h > 0 donnée, et représenter graphiquement $x \to \hat{g}_{n,h}$(x) pour différentes valeurs de h que vous choisirez. On discutera la raison pour laquelle ce choix est important et ce qui se produit si h est mal choisi.
==

Tout d'abord, nous chargeons les librairies nécessaires, et nous importons les fonctions vues en cours.

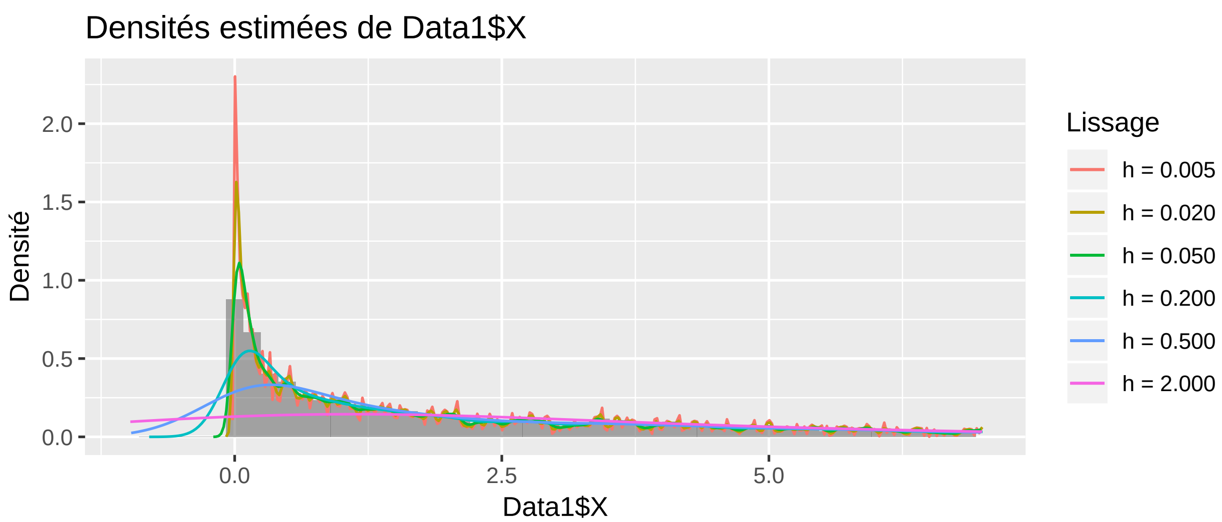
```
# Load libraries
  library(ggplot2)
  library(KernSmooth)
  library(stats)
  library(np)
  library(tidyverse)
  library(plotly)
  library(egg)
# Load functions
  source("customFunctions.R")
```

On calcule les densités pour différentes fenêtres de lissage (voir figure ci-dessous).

- Si h trop petit (0.005), l'estimation est très proche de la vraie mais trop oscillante.
- Si h trop grand (2.000), l'estimation est très lisse mais trop éloignée de la vraie valeur.
- On devine une valeur optimale entre 0.02 et 0.50.

```
# Lecture des données
  df = read.csv("Data1.csv")
# Listede fenêtres de lissage
  Lh = c(.005,.02,.05,.2,.5,2)
# Calcul pour différentes valeurs de h, avec un noyau gaussien
  res=bkde(df$X, kernel = "normal", bandwidth=Lh[1], truncate = TRUE); x1 = res$x; y1=res$y;
  res=bkde(df$X, kernel = "normal", bandwidth=Lh[2], truncate = TRUE); x2 = res$x; y2=res$y;
  res=bkde(df$X, kernel = "normal", bandwidth=Lh[3], truncate = TRUE); x3 = res$x; y3=res$y;
  res=bkde(df$X, kernel = "normal", bandwidth=Lh[4], truncate = TRUE); x4 = res$x; y4=res$y;
  res=bkde(df$X, kernel = "normal", bandwidth=Lh[5], truncate = TRUE); x5 = res$x; y5=res$y;
  res=bkde(df$X, kernel = "normal", bandwidth=Lh[6], truncate = TRUE); x6 = res$x; y6=res$y;
  df_ggp = data_frame(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6)
```

```r
# Affichage
  ggplot() + geom_histogram(data=df,aes(x=X,y = ..density..),bins = 50, alpha=.5) +
    geom_line(data=df_ggp,aes(x=x1, y=y1, colour = sprintf("h = %05.3f",Lh[1]))) +
    geom_line(data=df_ggp,aes(x=x2, y=y2, colour = sprintf("h = %05.3f",Lh[2]))) +
    geom_line(data=df_ggp,aes(x=x3, y=y3, colour = sprintf("h = %05.3f",Lh[3]))) +
    geom_line(data=df_ggp,aes(x=x4, y=y4, colour = sprintf("h = %05.3f",Lh[4]))) +
    geom_line(data=df_ggp,aes(x=x5, y=y5, colour = sprintf("h = %05.3f",Lh[5]))) +
    geom_line(data=df_ggp,aes(x=x6, y=y6, colour = sprintf("h = %05.3f",Lh[6]))) +
    labs(title="Densités estimées de Data1$X", x="Data1$X", y="Densité") +
    scale_color_discrete(name = "Lissage") + xlim(-1,7)
```



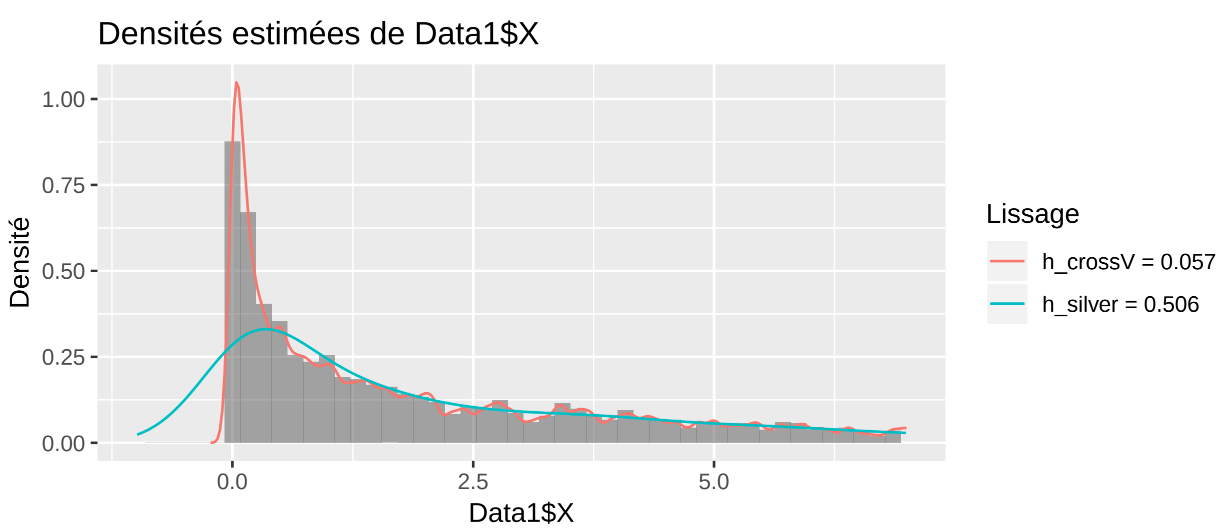
1.2 Représentation graphique

==
Représenter graphiquement $x \to \hat{g}_{n,\hat{h}_n}(\mathrm{x})$, avec $\hat{h}_n$ la fenêtre donnée par validation croisée ou par une autre méthode que l'on précisera.
==

On détermine 2 fenêtres de lissage, respectivement par validation croisée, et par la méthode de Silvermann. Il y a tout de même un facteur 10 entre les 2 valeurs de h. D'après la figure suivante :

- La fenêtre obtenue par validation croisée semble trop près des données.
- La fenêtre de Silvermann semble elle un peu trop lisse.

```
# Calcul des fenêtres de lissage
  h_crossV = bw.ucv(df$X)
  h_silver = 1.06*sqrt(var(df$X))*length(df$X)**(-.2)
# Affichage
  res=bkde(df$X, kernel = "normal", bandwidth=h_crossV, truncate = TRUE); x1 = res$x; y1=res$y;
  res=bkde(df$X, kernel = "normal", bandwidth=h_silver, truncate = TRUE); x2 = res$x; y2=res$y;
  df_ggp = data_frame(x1,y1,x2,y2)
  ggplot() + geom_histogram(data=df,aes(x=X,y = ..density..),bins = 50, alpha=.5) +
    geom_line(data=df_ggp,aes(x=x1, y=y1, colour = sprintf("h_crossV = %05.3f",h_crossV))) +
    geom_line(data=df_ggp,aes(x=x2, y=y2, colour = sprintf("h_silver = %05.3f",h_silver))) +
    labs(title="Densités estimées de Data1$X", x="Data1$X", y="Densité") +
    scale_color_discrete(name = "Lissage") + xlim(-1,7)
```

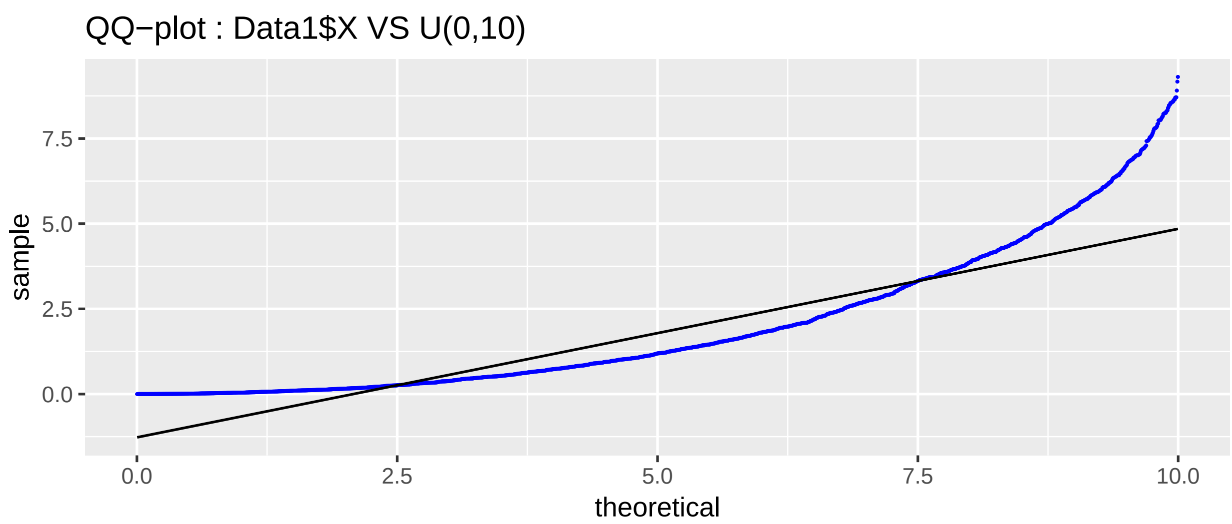


1.3 Implémenter un QQ-plot

==
Implémenter un QQ-plot pour vérifier empiriquement l'hypothèse *g(x)=1/10* pour tout x dans [0,10]. L'hypothèse selon laquelle g est uniforme semble-t-elle raisonnable?
==

Pour vérifier que la densité est uniforme , on implémente un QQ-plot des Data1.X (figure ci-dessous) sous l'hypothèse $U(0, 10)$. Vu la convexité de la courbe, l'hypothèse uniforme ne semble clairement pas raisonnable. La densité n'est pas uniforme, elle est très forte aux alentours de $x = 0$, puis elle décroit de plus en plus jusqu'à $x = 10$.

```
# QQ-plot
  df %>% ggplot(aes(sample = X)) +
    stat_qq(distribution = qunif, dparams = c(0,10), color="blue", size = 0.1) +
    stat_qq_line(distribution = qunif, dparams = c(0,10)) +
    labs(title = "QQ-plot : Data1$X VS U(0,10)")
```



1.4 Précision de l'estimation

==
Dans quelle zone de l'espace l'estimation de r sera plus précise ? Pourquoi ?
==

Plus la densité des points sera forte, plus la statistique sera forte localement, et donc plus l'estimation de $x \to r(x)$ sera précise. Donc $x \to r(x)$, sera bien estimée pour $x \to 0$, et de moins en moins bien au fur et à mesure que x augmente jusqu'à $x \to 10$.

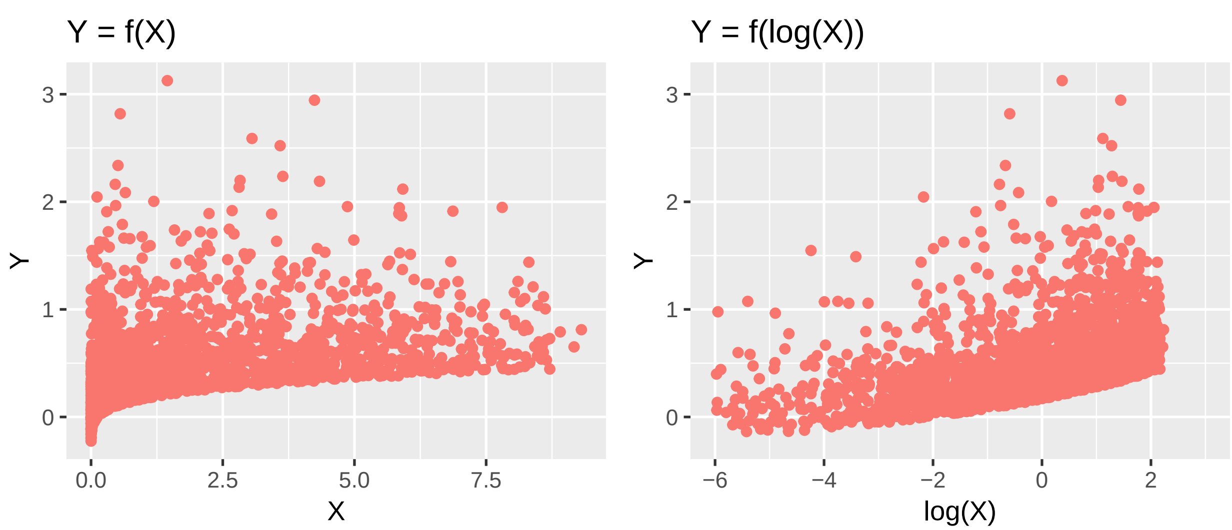
2. Reconstruction de r(x)

Pour cette partie, on utilise les données **data1.csv**

2.1 A propos de la linéarite de r

==
Est-il plausible de penser que la fonction r est linéaire ? Tracer Y1 en fonction de log(X), que remarque-t-on ?
==

```
pLin = df %>% ggplot(aes(x=X,y=Y1)) + geom_point(aes(colour="blue")) +
            labs(title="Y = f(X)", x="X", y="Y") + theme(legend.position="none")
pLog = df %>% ggplot(aes(x=log(X),y=Y1)) + geom_point(aes(colour="blue")) +
            labs(title="Y = f(log(X))", y = "Y") + theme(legend.position="none") + xlim(-6,3)
ggarrange(pLin, pLog, ncol = 2, nrow = 1)
```



On affiche les 2 nuages de points $X \to Y1$ et $log(X) \to Y1$.
Le bruit est distribué dans [0,Inf], donc on intuite ici r(x) en regardant l'enveloppe inférieure du nuage de points.

- la fonction r ne semble clairement pas linéaire par rapport à X.
- en revanche, pour $log(X) \geq -3$, on peut envisager une aprroximation linéaire par rapport à log(X).

2.2 Estimateur de r(x)

==
Construire un estimateur non-paramétrique $\hat{r}_{n,h}(\mathrm{x})$ de r(x) pour une fenêtre de lissage h > 0 bien choisie. Le représenter graphiquement.
==

On calcule les fenêtres de différentes manières, comme vu en cours. Pour l'affichage des régressions, voir la question 2.3. On fait 2 affichages en fonction de Data1.X : l'un avec $x \in [0, 10]$, l'autre zoomé avec $x \in [0, 0.3]$.

```
# Parametres
  n=length(df$X)
  std=sqrt(var(df$X))
# Application des différentes méthodes
  h_dpill=dpill(df$X,df$Y1) # adapté au bruit Gaussien
  #-----------------------------
  h_silver=1.06*std*n**(-.2) # adapté au bruit Gaussien/densité
  #-----------------------------
  h_gridCV=exp(seq(log(std/4),log(std),length=10))
  CVerr=CVbwt(h_gridCV,df$X,df$Y1)
  h_CVopt=h_gridCV[which.min(CVerr)]
  #-----------------------------
  h_CVb <- bw.cv.grid(X=df$X,Y=df$Y1) # Un peu long a calculer
  #-----------------------------
# Application des différentes h
  Y_pill  =locpoly(df$X,df$Y1,drv=0,degree=2,kernel="normal",bandwidth=h_dpill)
  Y_CVb   =locpoly(df$X,df$Y1,drv=0,degree=2,kernel="normal",bandwidth=h_CVb)
  Y_silver=locpoly(df$X,df$Y1,drv=0,degree=2,kernel="normal",bandwidth=h_silver)
  Y_CV    =locpoly(df$X,df$Y1,drv=0,degree=2,kernel="normal",bandwidth=h_CVopt)
  x = Y_pill$x  ; y = Y_pill$y  ; df1_1 = data.frame(x,y);
  x = Y_CVb$x   ; y = Y_CVb$y   ; df1_2 = data.frame(x,y);
  x = Y_silver$x; y = Y_silver$y; df1_3 = data.frame(x,y);
  x = Y_CV$x    ; y = Y_CV$y    ; df1_4 = data.frame(x,y);
```

Méthode	Valeur
pill	0.2187
silvermann	0.5057
CV_optimal	0.5454
CV_full	0.1200

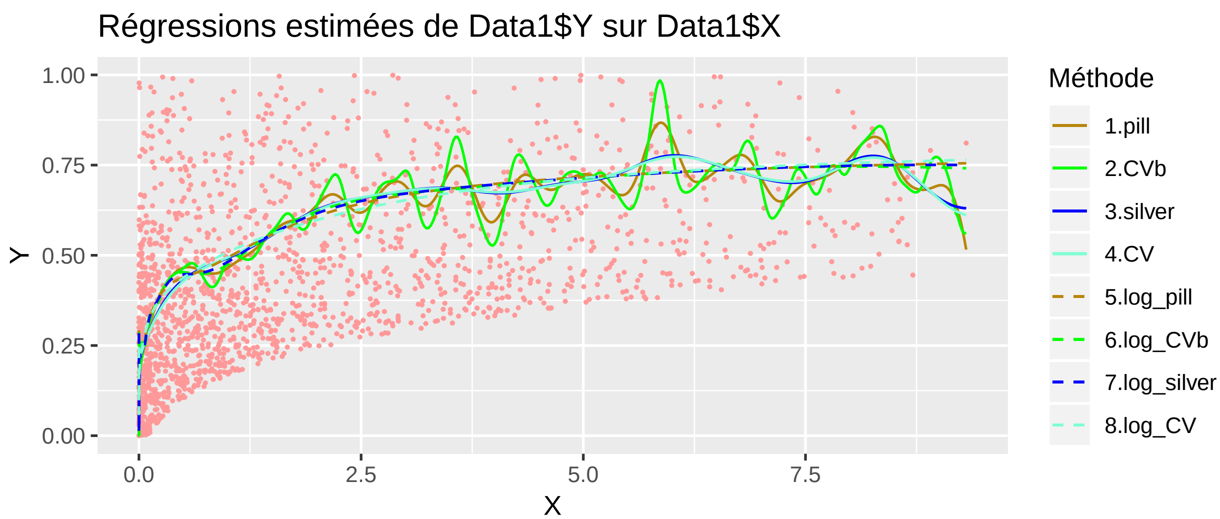
2.3 Estimation de r(log(x))

==
On se propose maintenant d'estimer r en régressant Y1 sur log(X). Construire un estimateur non-paramétrique $\tilde{r}_{n,h}(\mathrm{x})$ de $\tilde{r}(\mathrm{x})$ dans le modèle Y1 = $\tilde{r}(\log(\mathrm{X})) + \epsilon$, pour une fenêtre de lissage h > 0 bien choisie. Superposer sur le même graphe les résultats 2.2 et 2.3.
==

```
# Paramètres
  n=length(df$X)
  std=sqrt(var(log(df$X)))
# Application des différentes méthodes
  h_dpill=dpill(df$X,df$Y1) # adapté au bruit Gaussien
  #-----------------------------
  h_silver=1.06*std*n**(-.2) # adapté au bruit Gaussien/densité
  #-----------------------------
  h_gridCV=exp(seq(log(std/4),log(std),length=10))
  CVerr=CVbwt(h_gridCV,df$X,df$Y1)
  h_CVopt=h_gridCV[which.min(CVerr)]
  #-----------------------------
  h_CVb <- bw.cv.grid(X=df$X,Y=df$Y1) # Un peu long a calculer
  #-----------------------------
# Application des différentes h
  log_Y_pill  =locpoly(log(df$X),df$Y1,drv=0,degree=2,kernel="normal",bandwidth=h_dpill)
  log_Y_CVb   =locpoly(log(df$X),df$Y1,drv=0,degree=2,kernel="normal",bandwidth=h_CVb)
  log_Y_silver=locpoly(log(df$X),df$Y1,drv=0,degree=2,kernel="normal",bandwidth=h_silver)
  log_Y_CV    =locpoly(log(df$X),df$Y1,drv=0,degree=2,kernel="normal",bandwidth=h_CVopt)
  x = log_Y_pill$x  ; y = log_Y_pill$y  ; df2_1 = data.frame(x,y);
  x = log_Y_CVb$x   ; y = log_Y_CVb$y   ; df2_2 = data.frame(x,y);
  x = log_Y_silver$x; y = log_Y_silver$y; df2_3 = data.frame(x,y);
  x = log_Y_CV$x    ; y = log_Y_CV$y    ; df2_4 = data.frame(x,y);
# Paramètres pour l'affichage
  cL = c("darkgoldenrod","green","blue","aquamarine","darkgoldenrod","green","blue","aquamarine")
  nL = c("1.pill","2.CVb","3.silver","4.CV","5.log_pill","6.log_CVb","7.log_silver","8.log_CV")
  sL = c("solid","solid","solid","solid","dashed","dashed","dashed","dashed")
```

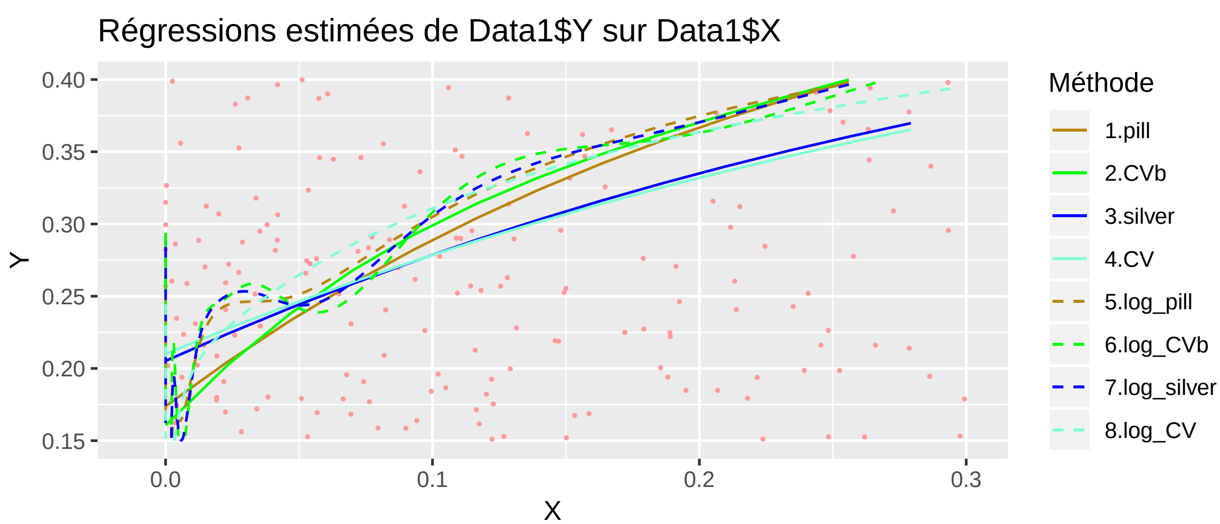
Méthode	Valeur
LOG : pill	0.6496
LOG : silvermann	0.4869
LOG : CV_optimal	1.1344
LOG : CV_full	0.4046

```
pLin = ggplot() + geom_point(data=df, aes(x=X,y=Y1), color = "#FF9999", size=.3) +
    geom_line(data=df1_1,aes(x=x, y=y, linetype = nL[1], color = nL[1])) +
    geom_line(data=df1_2,aes(x=x, y=y, linetype = nL[2], color = nL[2])) +
    geom_line(data=df1_3,aes(x=x, y=y, linetype = nL[3], color = nL[3])) +
    geom_line(data=df1_4,aes(x=x, y=y, linetype = nL[4], color = nL[4])) +
    geom_line(data=df2_1,aes(x=x, y=y, linetype = nL[5], color = nL[5])) +
    geom_line(data=df2_2,aes(x=x, y=y, linetype = nL[6], color = nL[6])) +
    geom_line(data=df2_3,aes(x=x, y=y, linetype = nL[7], color = nL[7])) +
    geom_line(data=df2_4,aes(x=x, y=y, linetype = nL[8], color = nL[8])) +
    scale_color_manual(name = "Méthode", values = cL) +
    scale_linetype_manual(name = "Méthode", values = sL) +
    labs(title="Régressions estimées de Data1$Y sur Data1$X", x="X", y="Y")
pLin + ylim(0,1)
```



On fait ensuite un petit zoom vers X proche de 0.

```
pLin + xlim(-.01,.3) + ylim(.15,.40)
```



2.4 Observation et analyse

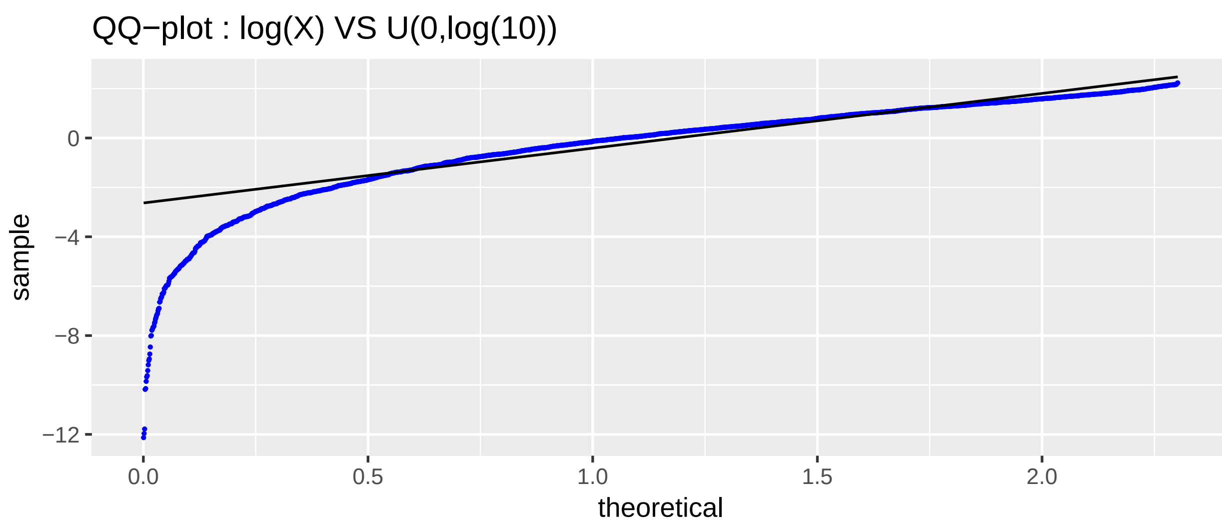
==
Que remarque-t-on ? Comment peut-on l'expliquer ?
==

Travailler sur les log(X) a uniformisé la répartition des X.

- Cela a permis de calculer une régression plus robuste pour $X \geq 0.2$ ou $log(X) \geq -1.6$
- En revanche, cela fait osciller la régression pour $X \leq 0.2$ ou $log(X) \leq -1.6$

Pour appuyer cette observation, on implémente un QQ-plot des logX sous l'hypothèse $U(0, log(10))$. La courbe et la droite sont à peu près alignées pour des valeurs de log(X) supérieure à -1.6.

```
# QQ-plot
  df$logX = log(df$X)
  ggplot(df, aes(sample = logX)) +
    stat_qq(distribution = qunif, dparams = c(0,log(10)), color="blue", size = 0.3) +
    stat_qq_line(distribution = qunif, dparams = c(0,log(10))) +
    labs(title = "QQ-plot : log(X) VS U(0,log(10))")
```



3. Etude de la densité μ des ε_i

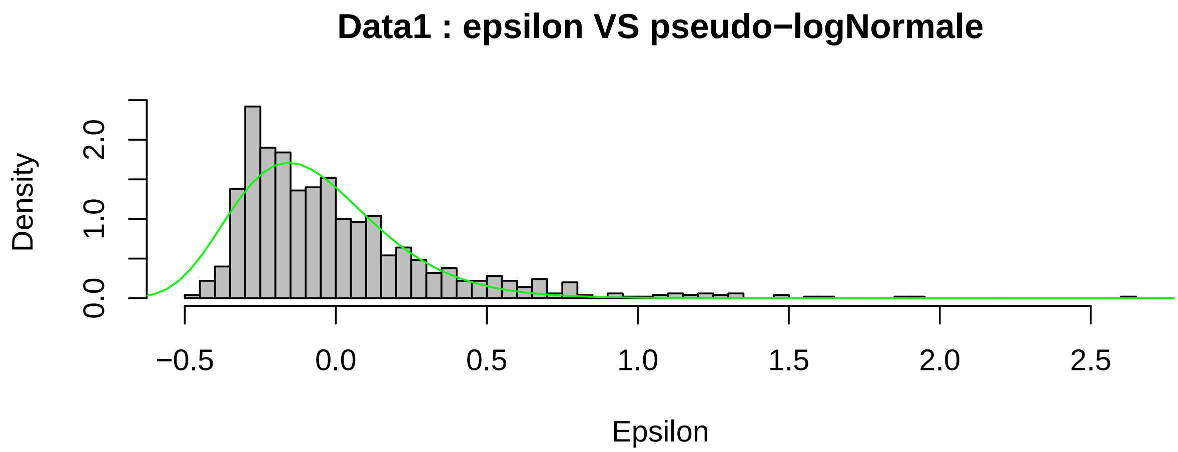
3.1 A partir du jeu de données *Data1.csv*

3.1.1 Distribution approximative des ε_i

==
On cherche à estimer $x \to \mu(x)$. Pour cela, on coupe l'échantillon en deux, selon que $i \in I_- = \{1,\ldots,1000\}$ ou que $i \in I_+ = \{1001,\ldots,2000\}$. On note $\hat{r}^-_{n,h}$(x) (pour un choix de h établi à la question 2.2) l'estimateur construit à l'aide de $(X_i, Y_i)_{1 \leq i \leq 1000}$ et on pose $\tilde{\varepsilon_i} = Y_i - \hat{r}^-_{n,h}(X_i), i \in I_+$
Quelle est la distribution approximative des $\tilde{\varepsilon_i}$?
==

On construit les résidus $\tilde{\varepsilon_i}$ comme prévu. Pour la fenêtre de lissage, on prend le critère de Silvermann, qui donnait une régression assez régulière. En dressant un histogramme (figure ci-dessous), on peut observer une asymétrie posistive. En première approche, cela laisse croire à une distribution de type log-normale : j'ai ainsi superposé à l'histogramme la densité $x \to Log - N(x + 1.1, 0, .24)$.

```
# Lecture des données et découpage en 2 parties
  df = read.csv("Data1.csv")
  df_m = df[  1:1000 ,] ; df_m = df_m[order(df_m$X),]
  df_p = df[-(1:1000),] ; df_p = df_p[order(df_p$X),]
# Calcul de Silvermann
  h = 1.06*sqrt(var(df_m$X))*length(df_m$X)**(-.2)
# Régression sur le jeu "MOINS" et application au jeu "PLUS"
  YY = ksmooth(df_m$X, df_m$Y1, kernel="normal", bandwidth=h, x.points=df_p$X )
  df_p$YY = YY$y
# Calcul des résidus
  df_p$eps = df_p$Y1 - df_p$YY
# Histogramme VS pseudo-logNormale
  hist(df_p$eps, prob=T, breaks=60, col = "grey", xlab = "Epsilon",
                       main = "Data1 : epsilon VS pseudo-logNormale")
  curve(dlnorm(x+1.1,0,.24),-1,3, add=T, col = "green")
```

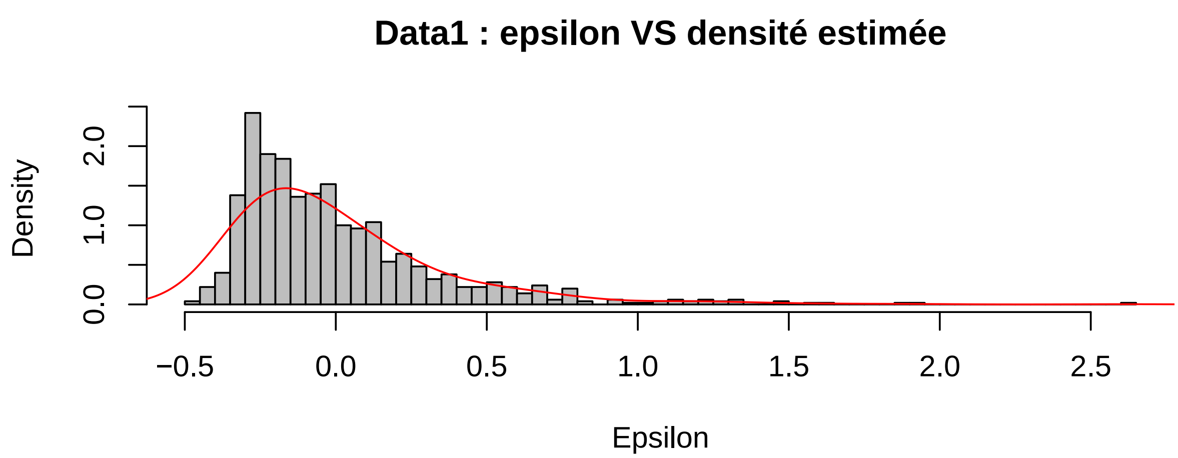


3.1.2 Estimateur de μ

===
En déduire un estimateur de $x \to \mu(x)$ et l'implémenter graphiquement.
===

On construit un estimateur de $x \to \mu(x)$, et on le superpose à l'histogramme précédent. L'estimée de $x \to \mu(x)$ ne parait pas gaussienne.

```
# Histogramme et densité
  h=5*bw.ucv(df_p$eps)
  MU = bkde(df_p$eps, kernel="normal", bandwidth=h, truncate = TRUE);
  hist(df_p$eps, prob=T, breaks=60, col = "grey", xlab = "Epsilon",
                       main = "Data1 : epsilon VS densité estimée")
  lines(MU$x, MU$y, col = "red")
```



3.1.3 Utilité du découpage

==
Quel est l'intéret d'avoir découpé le jeu de données selon I_+ et I_- ?
==

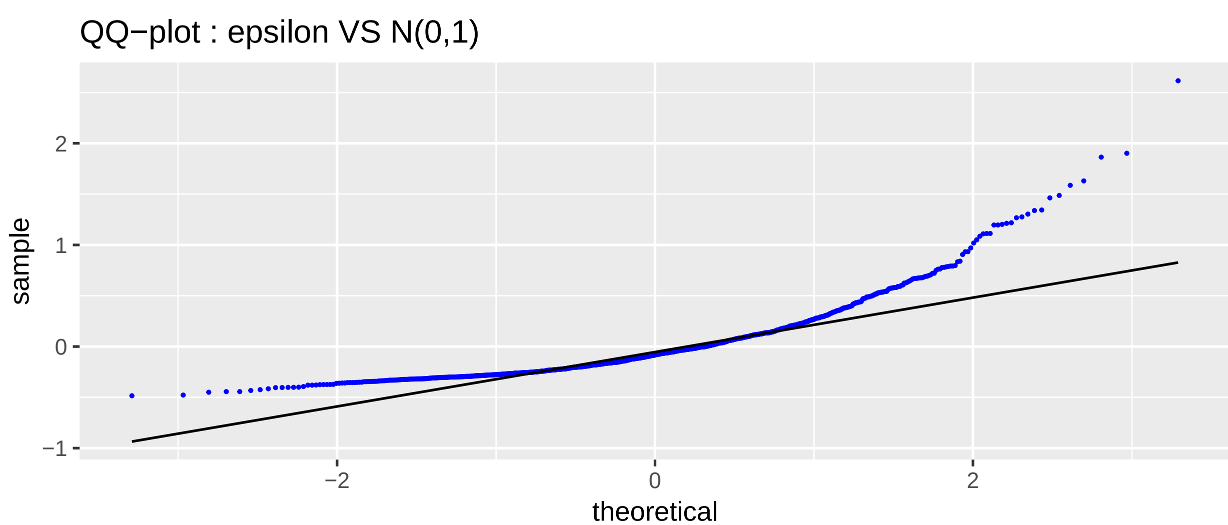
En découpant le jeu de données en 2 parties, on découple la caractérisation de la fonction de la caractérisation du bruit. En effet, la détermination de $x \to r(x)$ peut être influencée par le bruit. Comme les ε_i sont *i.i.d.*, en appliquant $x \to r(x)$ au deuxième jeu de données, on peut alors construire de vrais résidus $\tilde{\varepsilon_i}$ indépendants de l'estimation de $x \to r(x)$. Cela permet de caractériser plus rigoureusement $x \to \mu(x)$ et $x \to \sigma(x)$.

3.1.4 Test de densité gaussienne

==
La densité $x \to \mu$(x) peut-elle etre gaussienne ? Proposer un protocole pour le vérifier empiriquement et l'implémenter.
==

Comme vu sur l'histogramme précédent la densité $x \to \mu(x)$ ne parait pas gaussienne. Pour vérifier ce point, on implémente un QQ-plot des résidus (figure ci-dessous) sous l'hypothèse $N(0,1)$. Le non-alignement des points bleus sur la premiere bissectrice confirme que la densité $x \to \mu(x)$ n"est pas gaussienne.

```
# QQ-plot
  df_p %>% ggplot(aes(sample = eps)) +
            stat_qq(color="blue", size = 0.3) +
            stat_qq_line() +
            labs(title = "QQ-plot : epsilon VS N(0,1)")
```

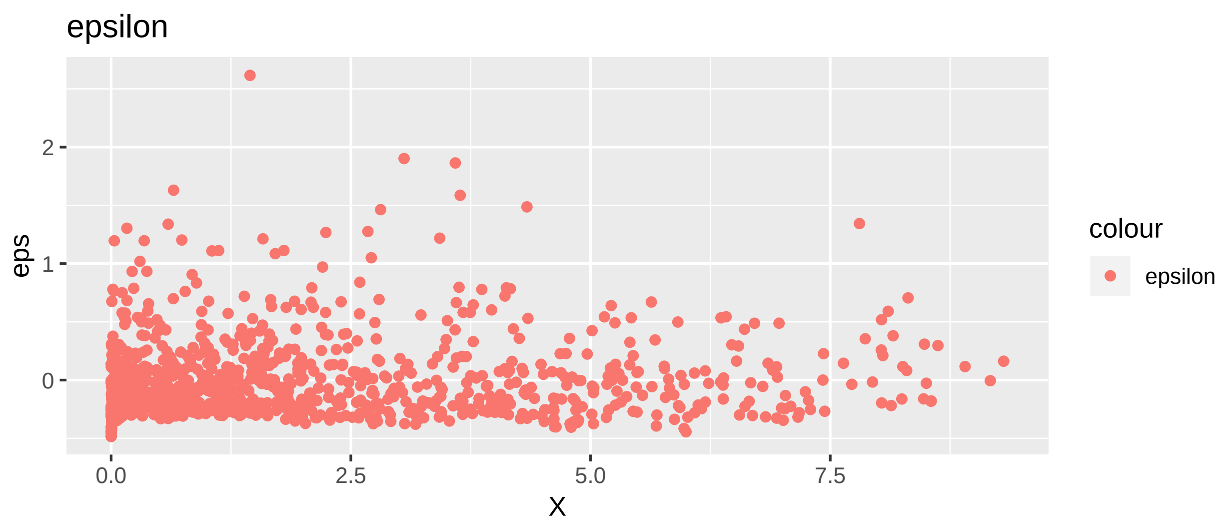


3.1.5 Test de modèle homoscédastique

==
Comment peut-on tester si le modèle est bien homoscédastique ?
==

Pour cela, on affiche les résidus ε_i. On constate alors que la dispersion des points ne semble pas dépendre de x. Cela laisse donc à penser aue $x \to \sigma(x)$ est ici une fonction constante, et donc que le modèle est bien homoscédastique.

```
# Plot epsilon
  df_p %>% ggplot() +
          geom_point(aes(x=X,y=eps,colour="epsilon")) +
          labs(title = "epsilon")
```



3.2 A partir du jeu de données *Data2.csv*

On cherche à estimer $x \to \mu$(x) et $x \to \sigma$(x). Pour cela, on coupe à nouveau l'échantillon en deux, et on considère à nouveau $\tilde{\varepsilon_i}$.

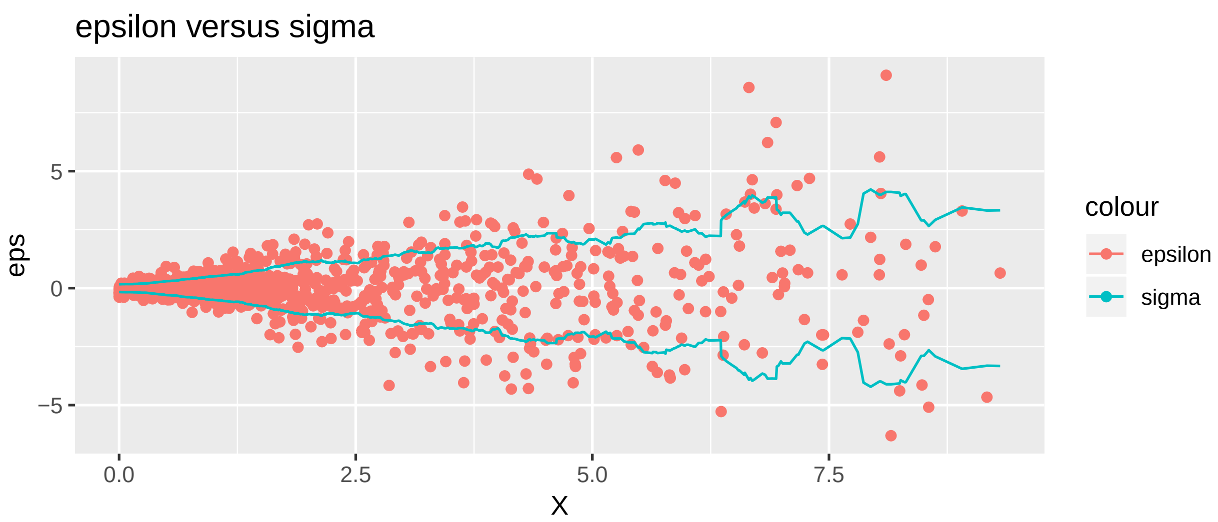
3.2.1 Test de modèle hétéroscédastique

==
Justifier qu'en régressant $\tilde{\varepsilon_i}^2$ sur X_i on obtient un estimateur de $x \to \sigma^2$(x). L'implémenter et le visualiser graphiquement. En comparant avec le jeu de données (Figure1: droite), retrouve-t-on un résultat attendu ?
==

Localement, en x0, $\tilde{\varepsilon}^2(x0) = \sigma^2(x0)\varepsilon^2(x0)$. En prenant l'espérance, on obtient $E[\tilde{\varepsilon}^2(x0)] = \sigma^2(x0)E[\varepsilon^2(x0)] = \sigma^2(x0)$ car $E[\varepsilon^2(x0)] = 1$ par définition. Cela justifie que la régression $\tilde{\varepsilon_i}^2$ sur X_i permet d'obtenir estimateur de $x \to \sigma^2$(x).

On implémente cette régression, et dans la figure ci-dessous, on affiche le nuage des $\tilde{\varepsilon_i}^2$ ainsi les courbes des estimateurs de $\sigma(X_i)$ et de $-\sigma(X_i)$.

```
# Lecture des données et découpage en 2 parties
  df = read.csv("Data2.csv")
  df_m = df[  1:1000 ,] ; df_m = df_m[order(df_m$X),]
  df_p = df[-(1:1000),] ; df_p = df_p[order(df_p$X),]
# Calcul de Silvermann
  h = 1.06*sqrt(var(df_m$X))*length(df_m$X)**(-.2)
# Régression sur le jeu "MOINS" et application au jeu "PLUS"
  YY = ksmooth(df_m$X, df_m$Y2, kernel="normal", bandwidth=h, x.points=df_p$X )
  df_p$YY = YY$y
# Calcul des résidus
  df_p$eps = df_p$Y2 - df_p$YY
# Estimation de la variance : sigmaCarre
  varEstim = ksmooth(df_p$X, (df_p$eps)**2, bandwidth=h, x.points=df_p$X)
  df_p$sigmaPlus  =  sqrt(varEstim$y)
  df_p$sigmaMinus = -sqrt(varEstim$y)
# Affichage
  df_p %>% ggplot() +
            geom_point(aes(x=X,y=eps,colour="epsilon")) +
            geom_line(aes(x=X,y=sigmaPlus,colour = "sigma")) +
            geom_line(aes(x=X,y=sigmaMinus,colour = "sigma")) +
            labs(title = "epsilon versus sigma")
```

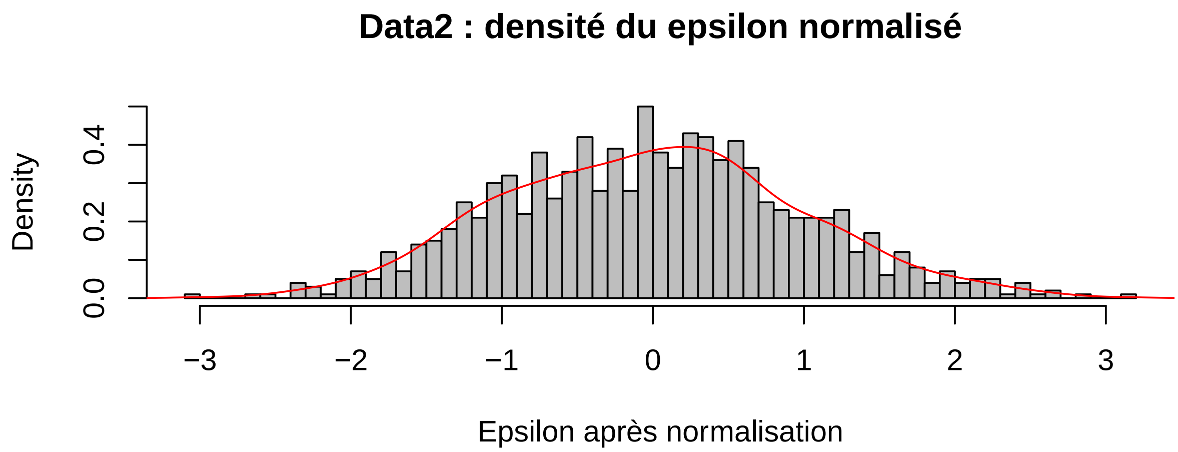


3.2.2 Test de densité gaussienne

==
La densité $x \to \mu(\mathrm{x})$ peut-elle être gaussienne ? Proposer un protocole pour le vérifier empiriquement et l'implémenter. *On pourra penser a renormaliser $\tilde{\varepsilon}_i$ par la fonction estimée a la question précédente et s'aider des questions de la section 3.1)*
==

Pour étudier $x \to \mu(x)$, on normalise les résidus $\tilde{\varepsilon}_i$ par la fonction estimée à la question précédente. On fait ensuite une estimation de la densité que l'on superpose à un histogramme des résidus normalisés. La figure ci-dessous laisse penser à une distribution gaussienne.

```
# Normalisation
  df_p$newEps = df_p$eps/df_p$sigmaPlus
# Histogramme et densité
  h=5*bw.ucv(df_p$eps)
  MU = bkde(df_p$newEps, kernel="normal", bandwidth=h, truncate = TRUE);
  hist(df_p$newEps, prob=T, breaks=60, col = "grey", xlab = "Epsilon après normalisation",
                          main = "Data2 : densité du epsilon normalisé");
  lines(MU$x, MU$y, col = "red")
```



Pour vérifier que la densité $x \to \mu(\mathrm{x})$ est gaussienne , on implémente un QQ-plot des résidus normalisés (figure ci-dessous) sous l'hypothèse $N(0,1)$. L'alignement des points bleus sur la premiere bissectrice confirme notre hypothèse.

```
# QQ-plot
  df_p %>% ggplot(aes(sample = newEps)) +
            stat_qq(color="blue", size = 0.3) +
            stat_qq_line() +
            labs(title = "QQ-plot : epsilon normalisé VS N(0,1)")
```