BERGUIGA Oussama

Master Statistique & Big Data

DM Régression Non Paramétrique

#### 

#### Importation des données

getwd()

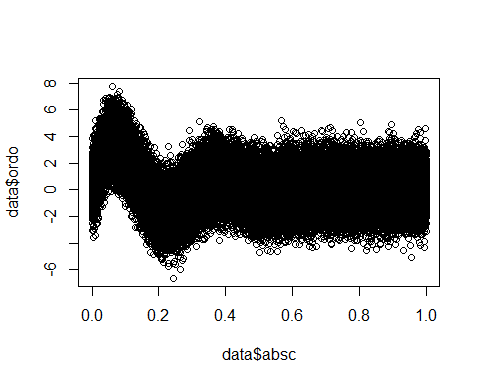
## [1] "C:/Users/oussa/Downloads/Data Science/Master Statistique Big Data Dauphine/Module 2/Regression non-paramétrique"

setwd("C:/Users/oussa/Downloads/Data Science/Master Statistique Big Data Dauphine/Module 2/Regression non-paramétrique")  
list.files()

data=read.table('DataReg')  
summary(data)

## absc ordo   
## Min. :0.0000046 Min. :-6.6500   
## 1st Qu.:0.2498369 1st Qu.:-0.8347   
## Median :0.5014559 Median : 0.1267   
## Mean :0.5009858 Mean : 0.2378   
## 3rd Qu.:0.7526427 3rd Qu.: 1.1674   
## Max. :0.9999953 Max. : 7.7522

plot(data$absc,data$ordo)



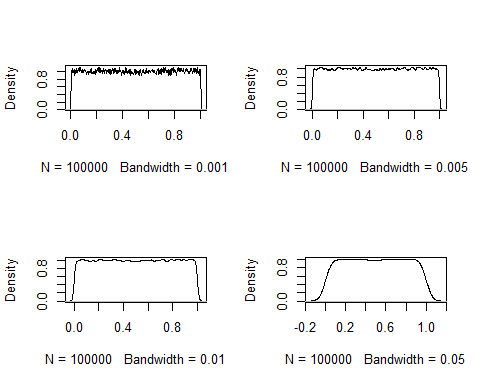
#### I) Exploration des propriete de g(x)

*Estimateur non paramétrique de g*

D'après le cours, un estimateur de g est donné par

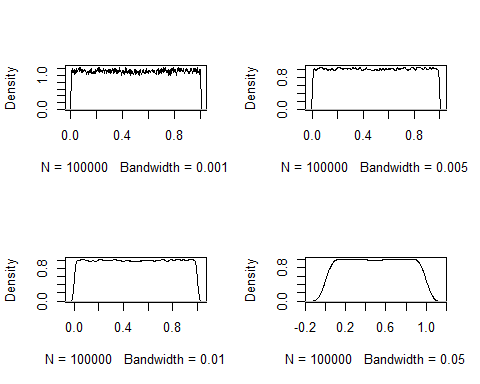
et et K un noyau ie une fonction de dans telle que . Le noyau est d'ordre k si pour tout l entier entre 1 et k:

par(mfrow=c(2,2))  
v=c(0.001,0.005,0.01,0.05)  
for (i in 1:length(v))  
   
{plot(density(data$absc,bw=v[i],kernel="gaussian"),main="")}

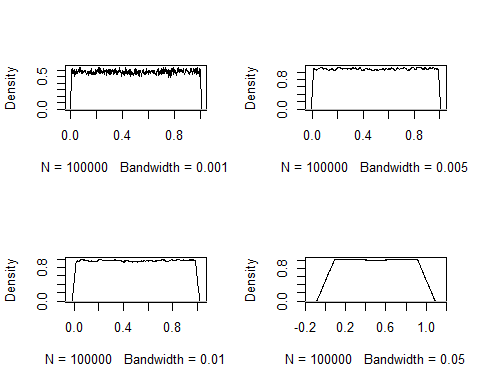


*Choix de différents noyaux et discussion de g=1, notamment aux bords de [0,1]*

par(mfrow=c(2,2))  
v=c(0.001,0.005,0.01,0.05)  
for (i in 1:length(v))  
   
{plot(density(data$absc,bw=v[i],kernel="triangular"),main="")}



par(mfrow=c(2,2))  
v=c(0.001,0.005,0.01,0.05)  
for (i in 1:length(v))  
   
{plot(density(data$absc,bw=v[i],kernel="rectangular"),main="")}



A première vue il est plausible de penser que g est constante (égale à 1) sur [0,1], ce qui signifierait que la variable X suit une loi uniforme sur [0,1]. Cependant, si on choisit un noyau "rectangular", on peut penser que la densité de g est une fonction affine sur les bords de [0,1].

*Choix de la fenêtre optimale (noyau gaussien)*

La fenêtre optimale est obtenue en minimisant un estimateur obtenu par validation croisée :

avec

#on peut utiliser la fonction bw.ucv ou dpik pour trouver la fenêtre minimale  
  
library(KernSmooth)

## Warning: package 'KernSmooth' was built under R version 3.3.3

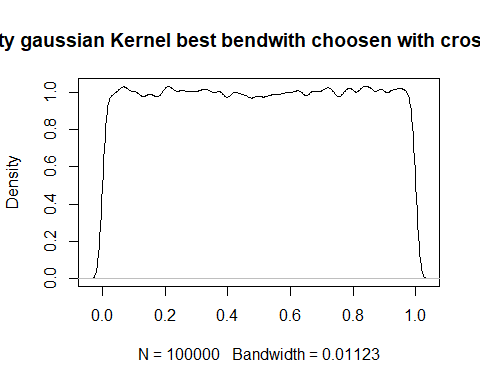
## KernSmooth 2.23 loaded  
## Copyright M. P. Wand 1997-2009

dpik(data$absc)

## [1] 0.01123104

#library(stats)  
#bw.ucv(data$absc)  
  
#Remarque : la fonction bw.ucv du package "stats" donne un tout autre résultat

plot(density(data$absc,bw=dpik(data$absc),kernel="gaussian"),main="g density gaussian Kernel best bendwith choosen with cross validation")



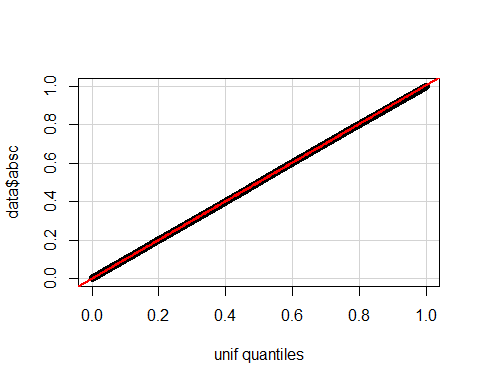
Avec un noyau gaussien et une fenêtre optimale, on peut penser que g est la densité de la loi uniforme sur [0,1]

*Test de l'hypothèse g=1,ie loi uniforme*

library(car)

## Warning: package 'car' was built under R version 3.3.3

qqPlot(data$absc,distribution="unif")



D'après le QQplot, g colle parfaitement à la densité de la loi uniforme sur [0,1]

*Question facultative*

est le moment d'ordre k des .D'après la loi des grands nombres, un estimateur de est :

# Construction d'un intervalle de confiance

Posons : g est la densité de la loi uniforme D'après la loi des grands nombres, si l'hypothese de loi uniforme est juste,l'estimateur devrait converger vers

D'après le cours, converge en loi vers une loi normale d’ espérance nulle et de variance( Ici = donc comme sous X suit la loi uniforme,on utilise la formule de l'esperance d'une transformée de variable pour trouver :

# Intervalle de confiance pour une loi gaussienne :

Soient l'espérance et la variance, appartient à l'intervalle au risque

#on calcul l'estimateur sur les 10^5 valeurs de X  
  
n=1e5  
  
c\_k=function(k){  
return (1/n\*sum((data$absc)^k))  
  
}  
  
c\_k(4)

## [1] 0.2015518

#Application  
  
  
intervalle\_confiance=function(n,k){  
v=c(0,0)  
  
  
#quantile 99% d'une gaussienne  
q\_99=qnorm(0.995)  
  
sigma\_k=sqrt(k^2/(((2\*k+1)\*(k+1)^2)))  
  
mu\_k=1/(k+1)  
  
v[1]=mu\_k-q\_99\*sigma\_k/sqrt(n)  
v[2]=mu\_k+q\_99\*sigma\_k/sqrt(n)  
 return(v)  
}  
  
  
  
  
  
  
#on genere un vecteur v de p entiers uniformement tires sur [1,m], et on teste pour tout element k apparenant à v si c\_k,n appartient à l'intervalle de confiance   
test=function(m,p){  
t=0  
   
v=round(runif(p,1,m))  
   
for (j in v)  
   
 {  
  
   
if (c\_k(j)>=(intervalle\_confiance(n,j)[1]) & c\_k(j)<=(intervalle\_confiance(n,j)[2]))  
   
 {t=t+1}  
  
 }  
return(t/p)  
   
 }  
  
test(1e10,1e4)

## [1] 1

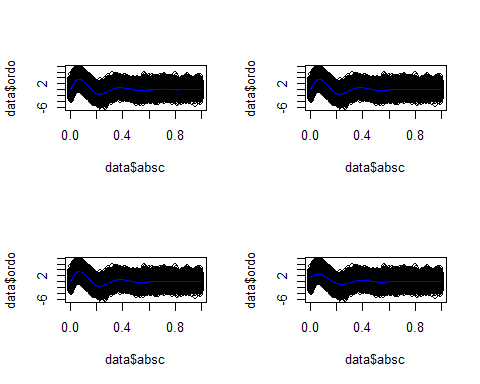
Conclusion : on a de fortes présomptions de penser que g est la densité de la loi uniforme sur [0,1]

#### II) Recondstruction de r(x)

*Construction de l’estimateur de Nadariya-Watson*

Estimateur de Nadariya-Watson est donné par :

library(KernSmooth)  
par(mfrow=c(2,2))  
v=c(0.001,0.005,0.01,0.05)  
for (i in 1:length(v))  
   
   
{plot(data$absc,data$ordo)  
 lines(locpoly(data$absc,data$ord,bandwidth=v[i]),main="",type='l',col='blue')}



*Inconvénients sur h*

h trop petit : sous-lissage, estimateur est très irrégulier reproduit simplement les observations

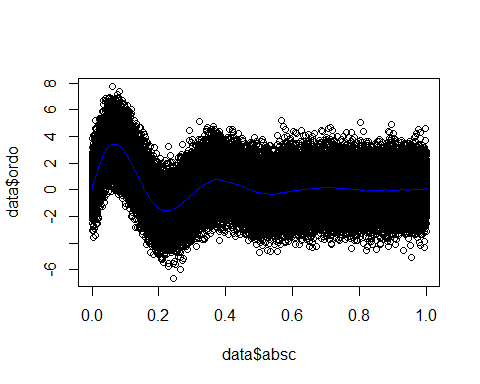
h trop grand : sur-lissage, on se rapproche de l'espérance des Yi.

Cependant, on aura tendance à préférer un h petit (puisque dans la théorie h doit tendre vers 0)

h\_opt=dpill(data$absc,data$ordo)  
print(h\_opt)

## [1] 0.008072254

plot(data$absc,data$ordo)  
lines(locpoly(data$absc,data$ord,bandwidth=h\_opt),main="",type='l',col='blue')

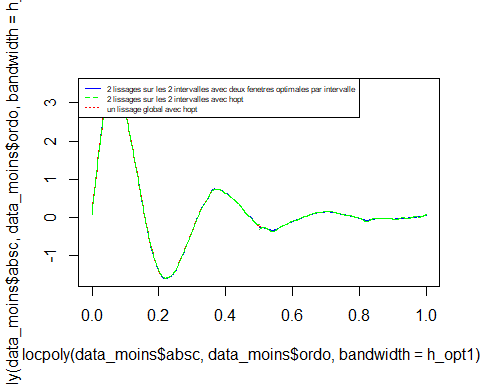


*Decoupage du jeu de données*

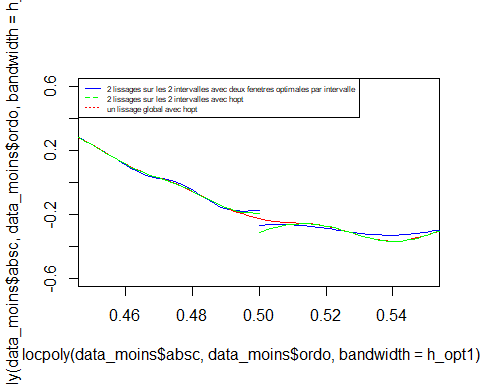
data$test=data$absc>=(1/2)  
  
data\_plus=data[data$test==TRUE,1:2]  
  
data\_moins=data[data$test==FALSE,1:2]  
  
   
h\_opt1=dpill(data\_moins$absc,data\_moins$ordo)  
h\_opt2=dpill(data\_plus$absc,data\_plus$ordo)  
  
print(c(h\_opt1,h\_opt2))

## [1] 0.006171008 0.014692359

#en bleu : 2 lissages sur les 2 intervalles avec deux fenetres optimales par intervalle  
  
#en vert 2 lissages sur les 2 intervalles avec hopt   
  
#en rouge un lissage global avec hopt  
  
plot(locpoly(data\_moins$absc,data\_moins$ordo,bandwidth = h\_opt1),col="blue",type='l',xlim=c(0,1))  
lines(locpoly(data\_plus$absc,data\_plus$ordo,bandwidth = h\_opt2),col="blue",type='l')  
lines(locpoly(data$absc,data$ordo,bandwidth=h\_opt),main="",col="red",type='l')  
  
lines(locpoly(data\_moins$absc,data\_moins$ordo,bandwidth = h\_opt),col="green",type='l',xlim=c(0,1))  
lines(locpoly(data\_plus$absc,data\_plus$ordo,bandwidth = h\_opt),col="green",type='l')  
legend("topleft", c("2 lissages sur les 2 intervalles avec deux fenetres optimales par intervalle","2 lissages sur les 2 intervalles avec hopt","un lissage global avec hopt"), col = c("blue","green","red"), lty = 1:3,cex=0.50)



plot(locpoly(data\_moins$absc,data\_moins$ordo,bandwidth = h\_opt1),col="blue",type='l',xlim=c(0.45,0.55),ylim=c(-0.6,0.6))  
lines(locpoly(data\_plus$absc,data\_plus$ordo,bandwidth = h\_opt2),col="blue",type='l')  
lines(locpoly(data$absc,data$ordo,bandwidth=h\_opt),main="",col="red",type='l')  
  
lines(locpoly(data\_moins$absc,data\_moins$ordo,bandwidth = h\_opt),col="green",type='l')  
lines(locpoly(data\_plus$absc,data\_plus$ordo,bandwidth = h\_opt),col="green",type='l')  
  
legend("topleft", c("2 lissages sur les 2 intervalles avec deux fenetres optimales par intervalle","2 lissages sur les 2 intervalles avec hopt","un lissage global avec hopt"), col = c("blue","green","red"), lty = 1:3,cex=0.50)



On peut tirer 2 enseignements de ces graphiques :

* le choix de la fenêtre optimale est lié à la variation de la fonction sur un intervalle donné : on remarque que les 2 fenêtres sont très différentes (du simple au double). En effet la variation des (et donc de ) est bien plus importante sur l'intervalle [0,0.5] que sur l'intervalle [0.5,1]
* la régression au bord d'un intervalle peut être faussée par manque d'information : cf ici la régression au point Xi=0.5.Le lissage au point 0.5 est plus juste dans le cas du lissage de r sur [0,1] car on dispose de 2 fois plus d'informations sur le comportement des au voisinage de 0.5 que dans le cas de 2 lissages séparés [0,0.5] et [0.5,1]

#### III) Etude de la loi des

**REMARQUE : dans cette partie je considère les Xi NON DETERMINISTES donc d'après la relation Y=r(X)+,et par independance de x et (donc r(X) )et : Var(Y)=Var (r(X))+**

*Implementation*

sigma\_hat=function(y,n)  
  
{  
   
 return (1/(2\*(n-1))\*sum(diff(y)[1:(n-1)]^2))  
}  
  
n=1e5  
sigma\_hat(data$ordo,n)

## [1] 2.589785

*Justification*

var(data$ordo)

## [1] 2.575987

abs(sigma\_hat(data$ordo,n)-var(data$ordo))/var(data$ordo)#0.5% d'erreur relative

## [1] 0.005356477

On pose , alors d'après la loi des grands nombres, l'estimateur de Rice converge vers l'espérance de Zi. Or = =

= car et sont independants or :

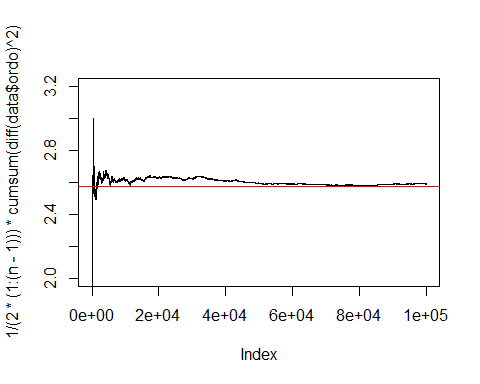
= et =

donc = =

L'estimateur de Rice est donc un estimateur sans biais de la variance des . Remarque : lien entre la variance de et celle de : =+ par indépendance de et

*Précision de l'estimateur*

plot(1/(2\*(1:(n-1)))\*cumsum(diff(data$ordo)^2),xlim=c(1,1e5),ylim=c(2,3.2),type='l')  
  
abline(h=var(data$ordo),col="red")



Le biais de l'estimateur est nul. Concernant la variance : d'après le TCL on peut construire un intervalle de confiance pour = de la forme (au risque ) :

avec s=

min(data$ordo)

## [1] -6.650033

max(data$ordo)

## [1] 7.752179

or == or comme Yi appartient à [-10,10] (cf ci-dessus), inférieur à 20 donc inferieur

à donc s inférieur à 10.

#application  
  
  
q\_99=qnorm(0.995)  
majorant\_s=10  
  
  
intervalle=c(sigma\_hat(data$ordo,n)-q\_99\*majorant\_s/sqrt(n),sigma\_hat(data$ordo,n)+q\_99\*majorant\_s/sqrt(n))  
print(min(intervalle))

## [1] 2.50833

print(max(intervalle))

## [1] 2.67124

print(var(data$ordo))

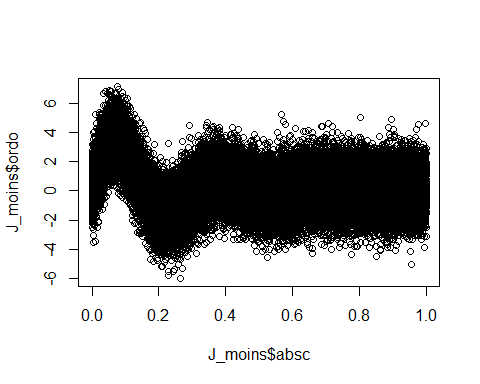
## [1] 2.575987

print(var(data$ordo)>=min(intervalle) & var(data$ordo)<=max(intervalle))

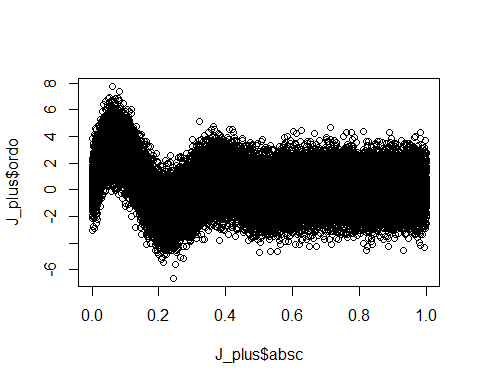
## [1] TRUE

*Distribution de*

#par(mfrow=c(2,1))  
  
J\_moins=data[1:5e4,]  
  
J\_plus=data[(5e4+1):1e5,]  
  
  
plot(J\_moins$absc,J\_moins$ordo)



plot(J\_plus$absc,J\_plus$ordo)

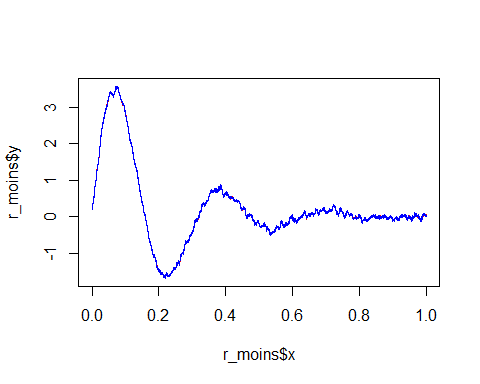


?ksmooth

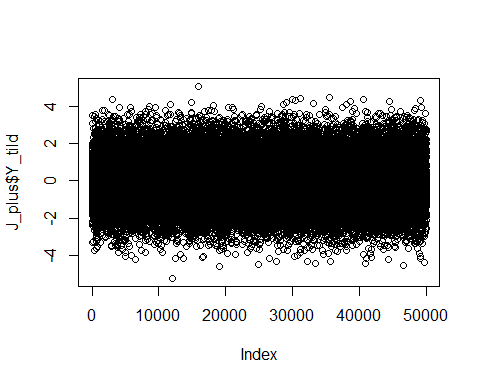
## starting httpd help server ...

## done

#on utilise la fonction ksmooth qui permet de faire des predictions  
r\_moins=ksmooth(x=J\_moins$absc,y=J\_moins$ordo,bandwidth=h\_opt)  
predict\_r\_moins=ksmooth(x=J\_moins$absc,y=J\_moins$ordo,bandwidth=h\_opt,n.points=length(J\_plus$absc)  
 ,x.points=J\_plus$absc)  
  
  
plot(r\_moins,col="red",type="l",xlim=c(0,1))  
lines(predict\_r\_moins,col="blue",type="l")



#les deux distributions semblent être extremement proches  
#r\_moins$y[1:100]-predict\_r\_moins$y[1:100]  
  
  
#normalement la distribution de Y\_tild devrait être celle de Xsi\_i\_plus  
#Attention dans l'operation a ne pas oublier de trier les valeurs par X croissant dans J\_plus car ksmooth retourne des valeurs de (x,y) avec x croissant or les Xi sont aléatoires et uniformes donc desordonnes !  
  
J\_plus$Y\_tild=J\_plus[order(J\_plus$absc),'ordo']-predict\_r\_moins$y  
  
#str(predict\_r\_moins$y)  
plot(J\_plus$Y\_tild)



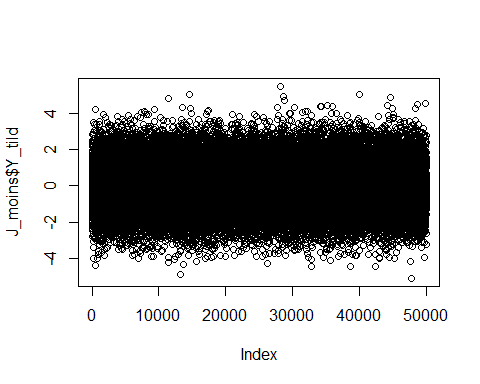
mean(J\_plus$Y\_tild)

## [1] -0.003873694

var(J\_plus$Y\_tild)

## [1] 1.43351

#generons egalement les Xsi\_moins  
r\_plus=ksmooth(x=J\_plus$absc,y=J\_plus$ordo,bandwidth=h\_opt)  
predict\_r\_plus=ksmooth(x=J\_plus$absc,y=J\_plus$ordo,bandwidth=h\_opt,n.points=length(J\_moins$absc),x.points=J\_moins$absc)  
  
J\_moins$Y\_tild=J\_moins[order(J\_moins$absc),"ordo"]-predict\_r\_plus$y  
plot(J\_moins$Y\_tild)



mean(J\_moins$Y\_tild)

## [1] 0.003518048

var(J\_moins$Y\_tild)

## [1] 1.4492

mean(c(J\_moins$Y\_tild,J\_plus$Y\_tild))

## [1] -0.0001778229

var(c(J\_moins$Y\_tild,J\_plus$Y\_tild))

## [1] 1.441354

Les distributions de devrait approximativement être celle de

*Estimation de*

pour reconstituer , on utilise le lissage par noyau pour une densite comme pour la densite g(x)

#ajoutons les Xsi\_plus et Xsi\_moins pour avoir plus de points  
Xsi\_i=c(J\_moins$Y\_tild,J\_plus$Y\_tild)  
mean(Xsi\_i)#environ 0

## [1] -0.0001778229

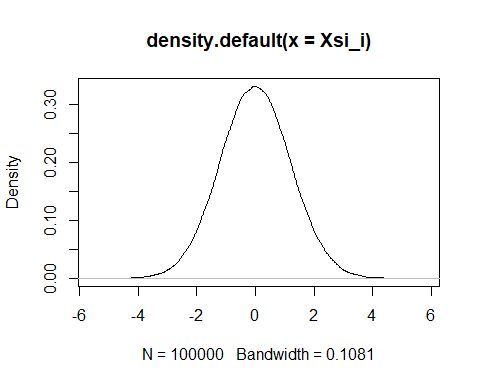
var(Xsi\_i)#1.44

## [1] 1.441354

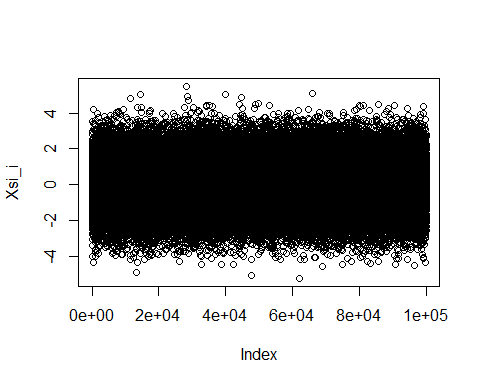
#on peut utiliser l'estimateur de Rice pour estimer sigma² :  
sigma\_hat(J\_moins$Y\_tild,length(J\_moins$Y\_tild))

## [1] 1.436067

plot(density(Xsi\_i),type='l')



plot(Xsi\_i)



A première vue la densité ressemble à une gaussienne centrée en 0.

Remarque : vaudrait donc 1.44. Or si on considère les Xi non déterministes, d'après la relation Var(Y)=Var (r(X))+ donc

Var(r(X))=Var(Y) -

Var\_r\_X=var(data$ordo)-var(Xsi\_i)  
Var\_r\_X#1.13

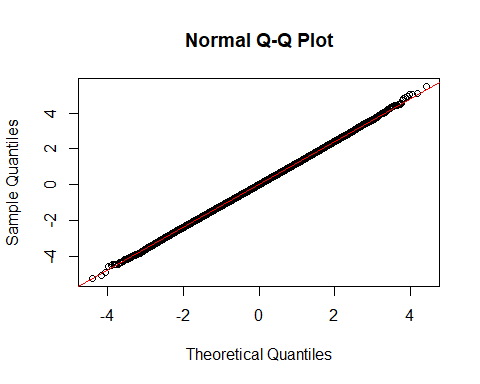
## [1] 1.134633

*question sur la gaussianité*

# qqplot

# test de gaussianité

#la densite dessinee a la question precedente   
  
qqnorm(Xsi\_i)  
qqline(Xsi\_i,col=2)



#Test de normalite pour les Xsi\_i : on peut utiliser un test de Shapiro-Wulk   
#Shapiro-Wulk test :   
#Ho= la série statistique suit une distribution normale   
#H1=la serie statistique ne suit pas une distribution gaussienne   
  
?shapiro.test  
shapiro.test(sample(Xsi\_i,5000))

##   
## Shapiro-Wilk normality test  
##   
## data: sample(Xsi\_i, 5000)  
## W = 0.99973, p-value = 0.7856

#p-value >> 5%, alors le test de Shapiro montre que les Xsi sont gaussiens  
  
#Remarque : test de Kolmogorov Smirnov :   
#Ho : la distribution est Gaussienne  
#ks.test(Xsi\_i,"pnorm")  
  
#p-value << 5 % on rejette Ho  
  
#Il semblerait donc qu'un test rejette la gaussianité distribution ...bizzare...