TP4 - Simulation, échantillonnage, bootstrap, validation croisée avec R

Thomas Laurent

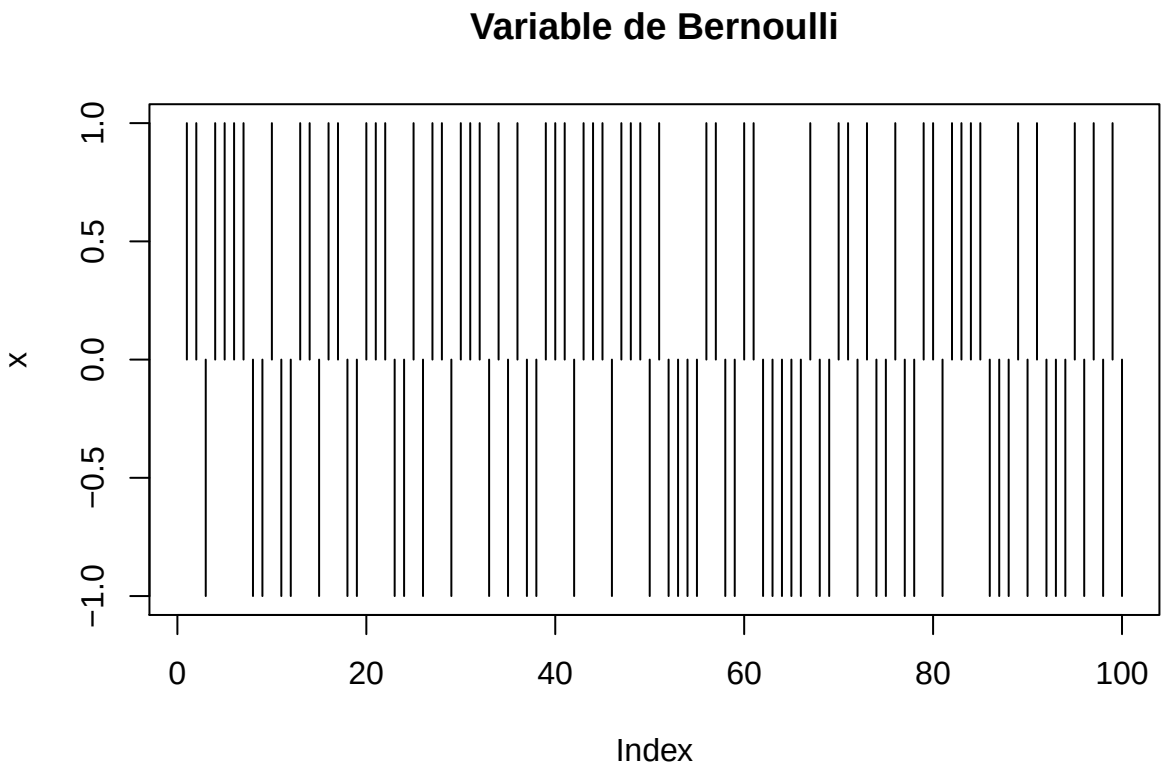
21/02/2018

Simulation de lois

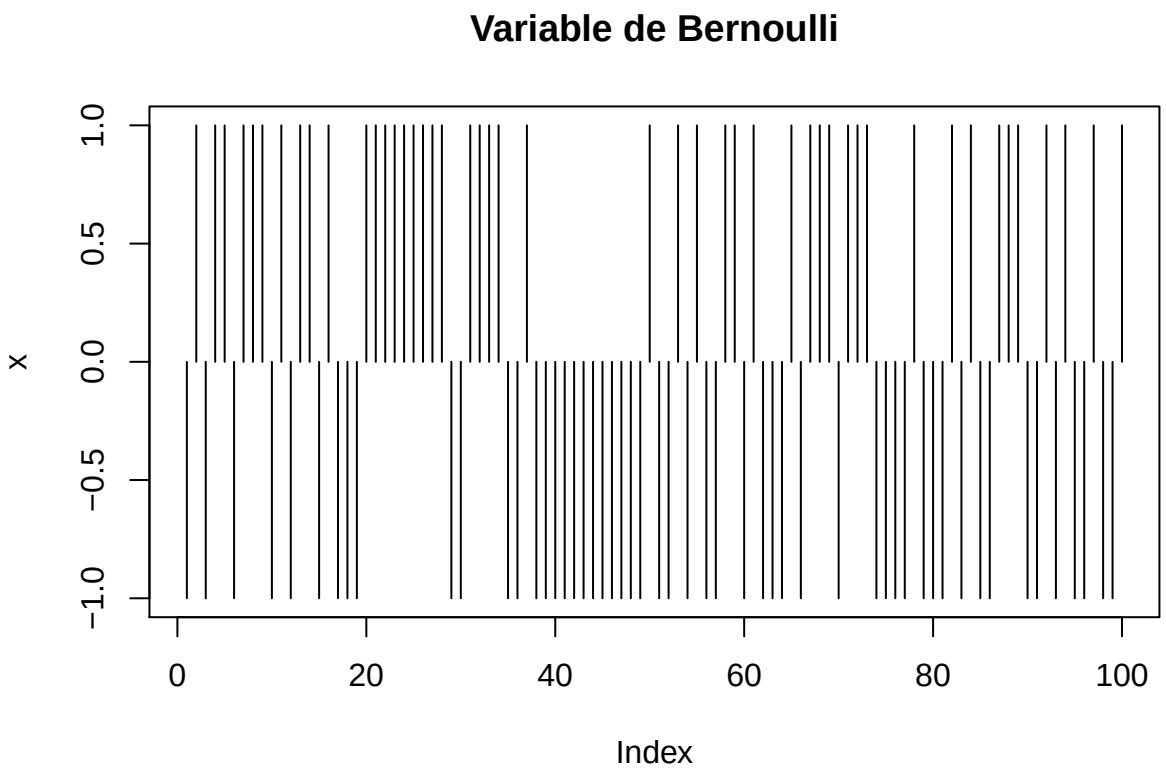
Bernouilli

On réalise 100 tirages.

```
n=100
x=sample(c(-1,1), n, replace=T)
plot(x, type='h', main="Variable de Bernoulli")
```

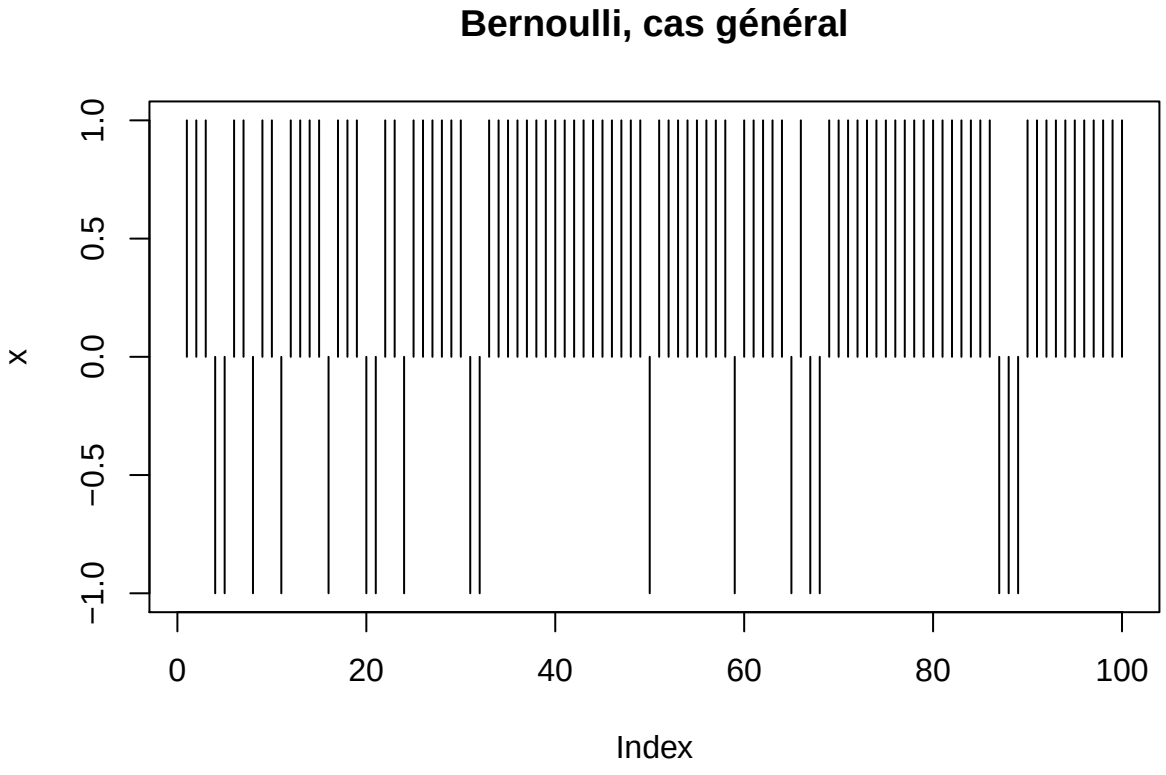



```
set.seed(123)
n=100
x=sample(c(-1,1), n, replace=T)
plot(x, type='h', main="Variable de Bernoulli")
```



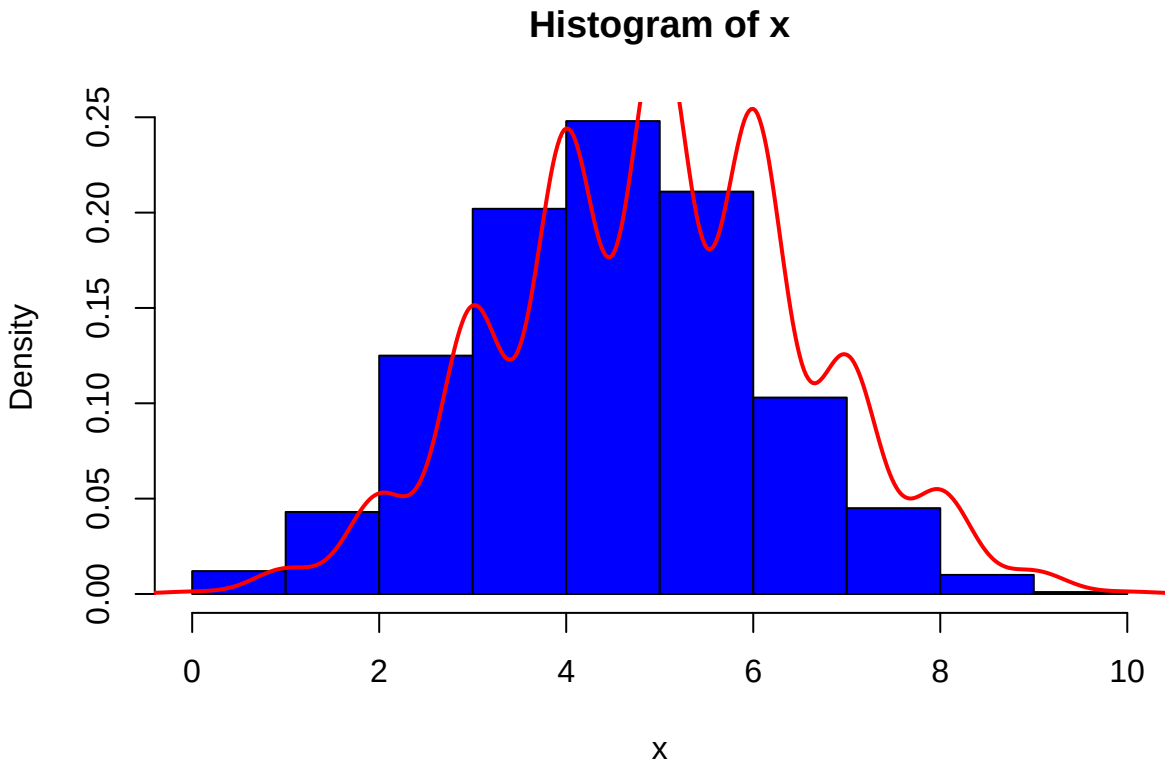
On fixe la semence pour pouvoir reproduire la séquence ultérieurement.

```
set.seed(123)
n=100
x=sample(c(-1,1), n, replace=T, prob=c(.2,.8))
plot(x, type='h',main="Bernoulli, cas général")
```



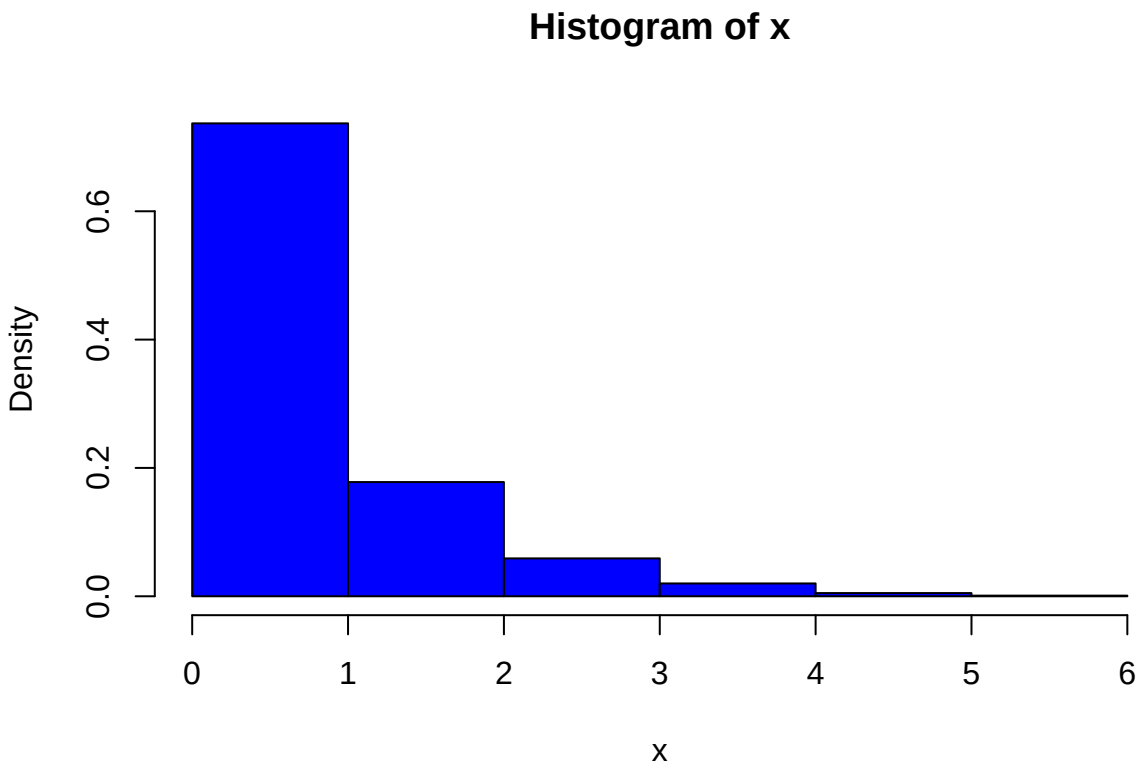
Binomiale

```
#Tirage d'un echantillon de taille 1000 suivant une loi binomiale
x=rbinom(1000,10,0.5)
#Tracage de l'histogramme
hist(x,xlim=c(min(x),max(x)), probability=T, col='blue',nclass=max(x)-min(x))
#Estimation non parametrique
lines(density(x), col='red', lwd=2)
```

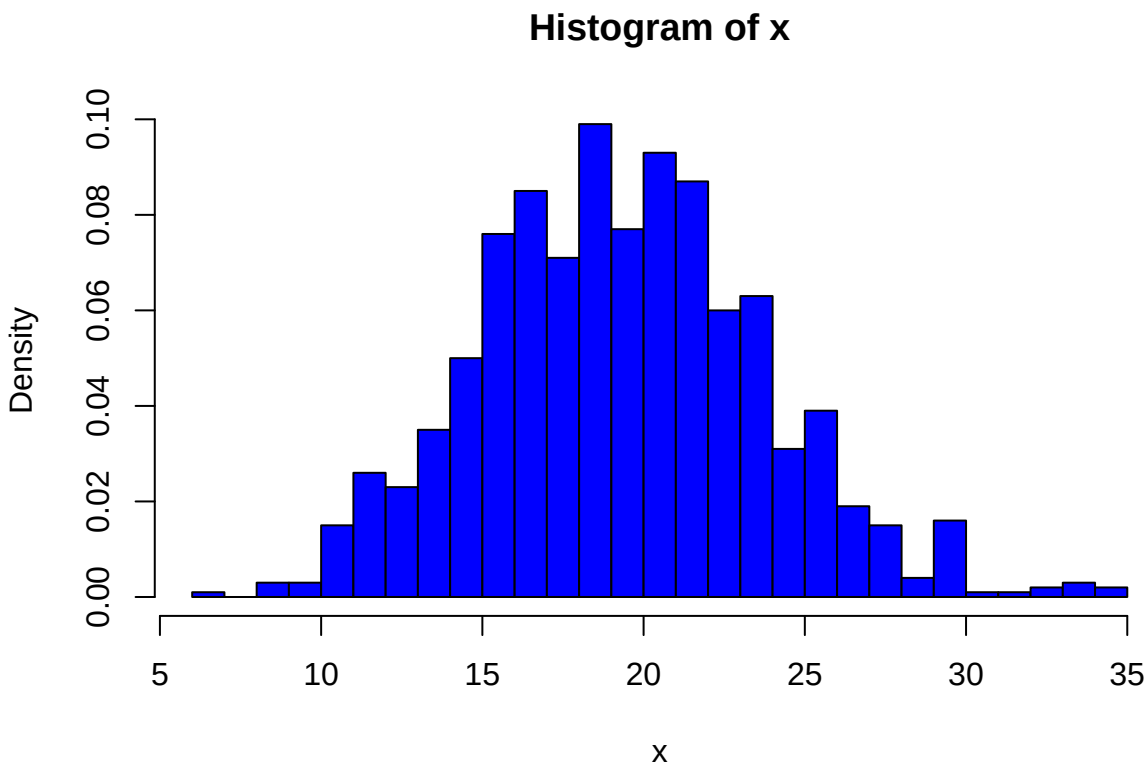


Loi de Poisson

```
x=rpois(1000,1)
hist(x,xlim=c(min(x),max(x)), probability=T, col='blue',nclass=max(x)-min(x))
```




```r
x=rpois(1000,20)
hist(x,xlim=c(min(x),max(x)), probability=T, col='blue',nclass=max(x)-min(x))
```

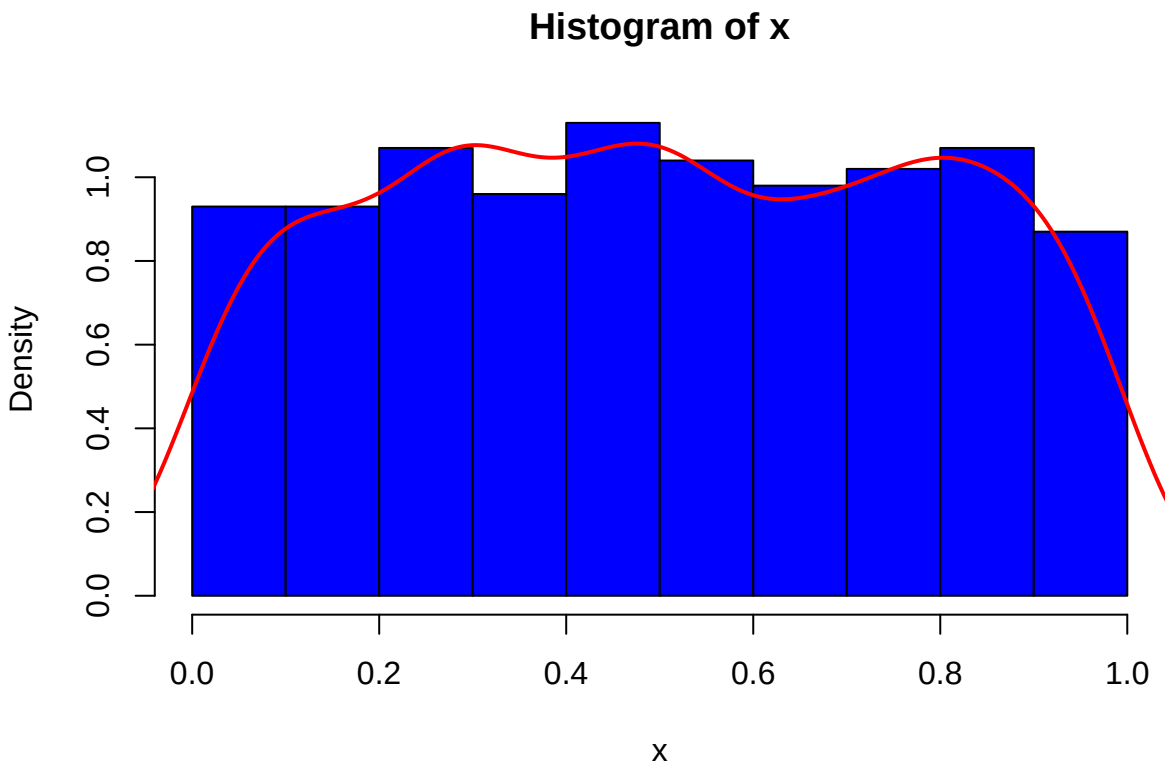


Loi uniforme, de Cauchy

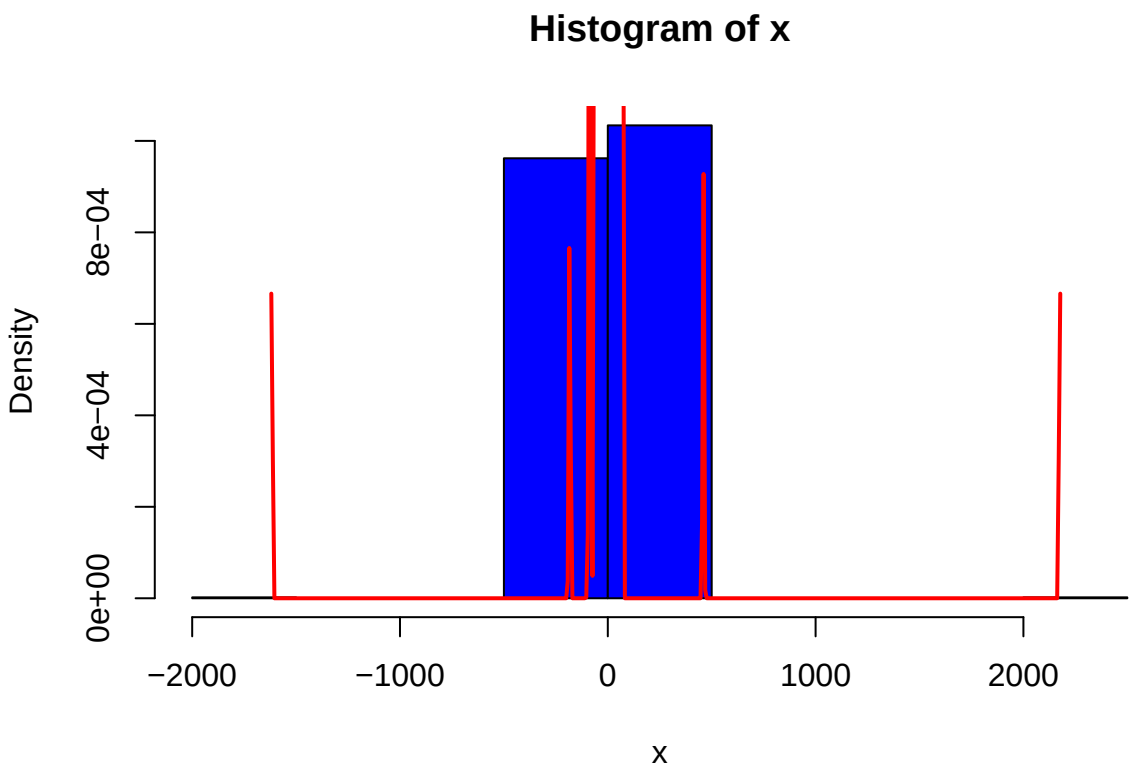
```
#Loi uniforme
round(runif(10), digits=3)
```

```
##  [1] 0.950 0.948 0.645 0.315 0.532 0.675 0.857 0.557 0.094 0.711
```

```
x=runif(1000)
hist(x, probability=T, col='blue')
lines(density(x), col='red', lwd=2)
```

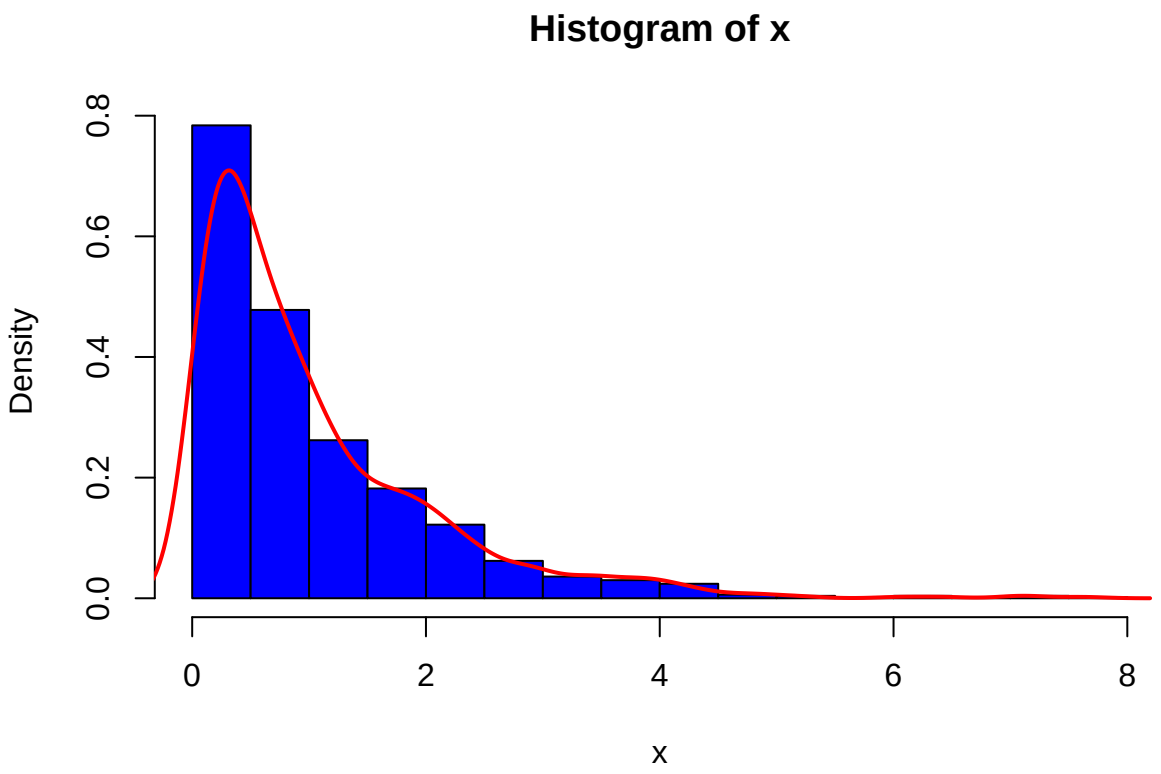



```
#Loi de cauchy
x=rcauchy(1000)
hist(x, probability=T, col='blue')
lines(density(x), col='red', lwd=2)
```

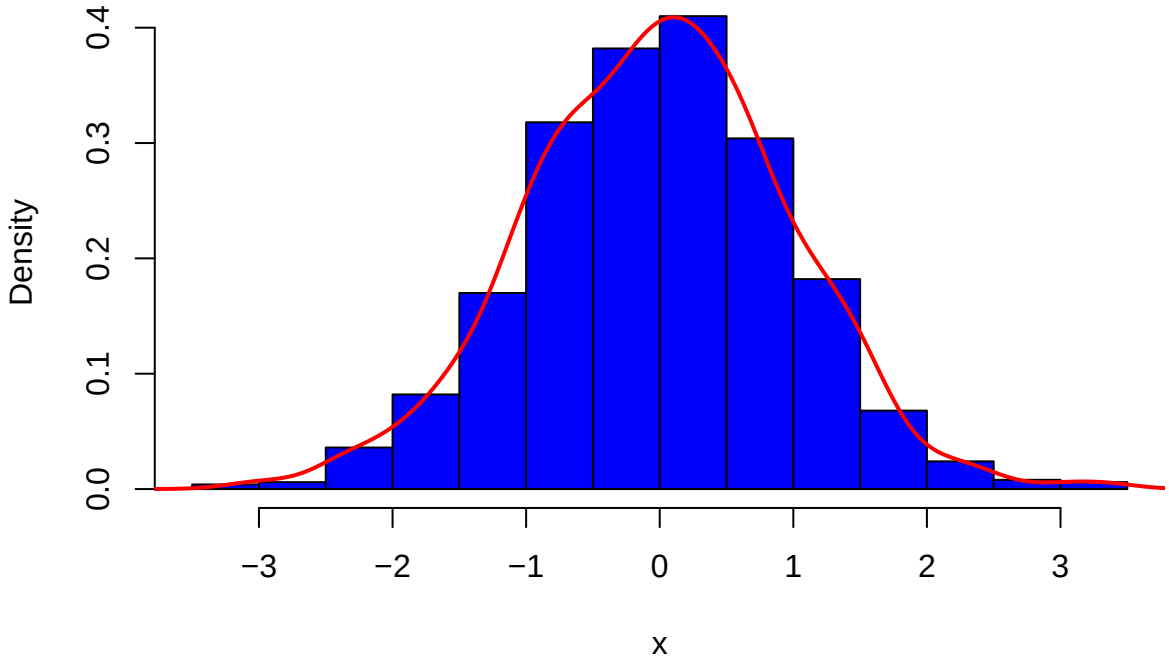


Autres lois

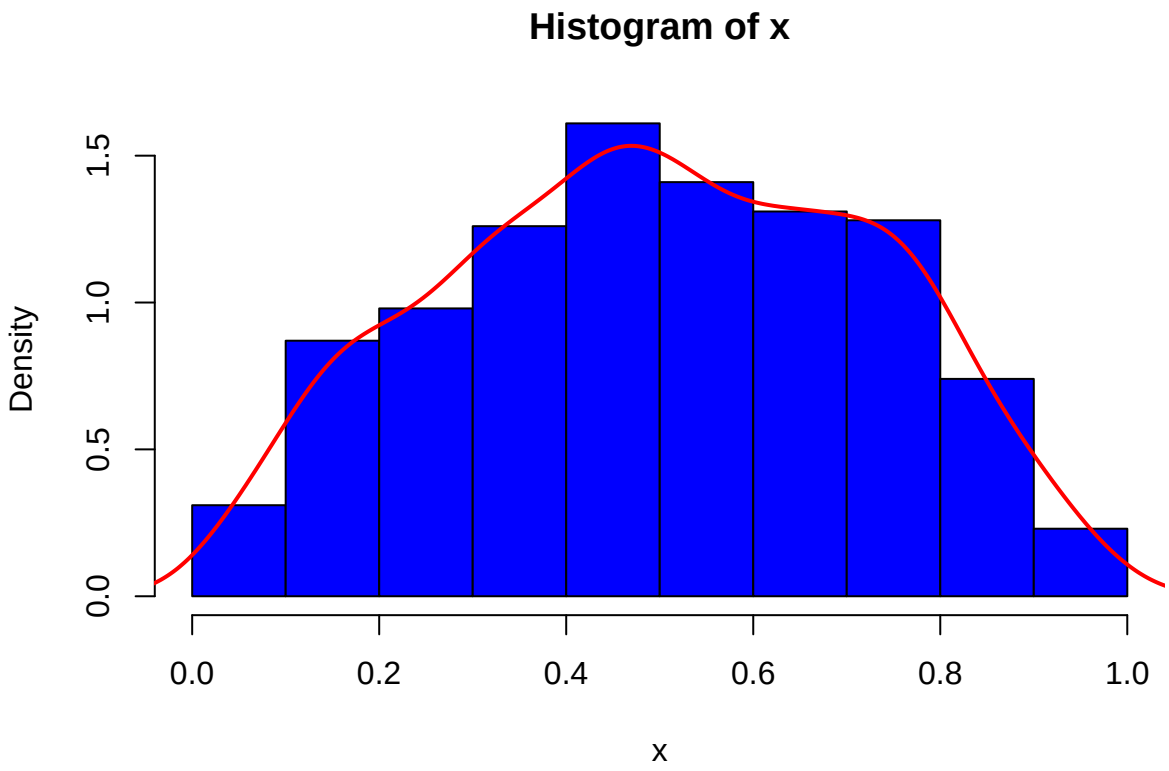
```
#Loi exponentielle
x=rexp(1000)
hist(x, probability=T, col='blue')
lines(density(x), col='red', lwd=2)
```




```r
#Loi normale
x=rnorm(1000)
hist(x, probability=T, col='blue')
lines(density(x), col='red', lwd=2)
```

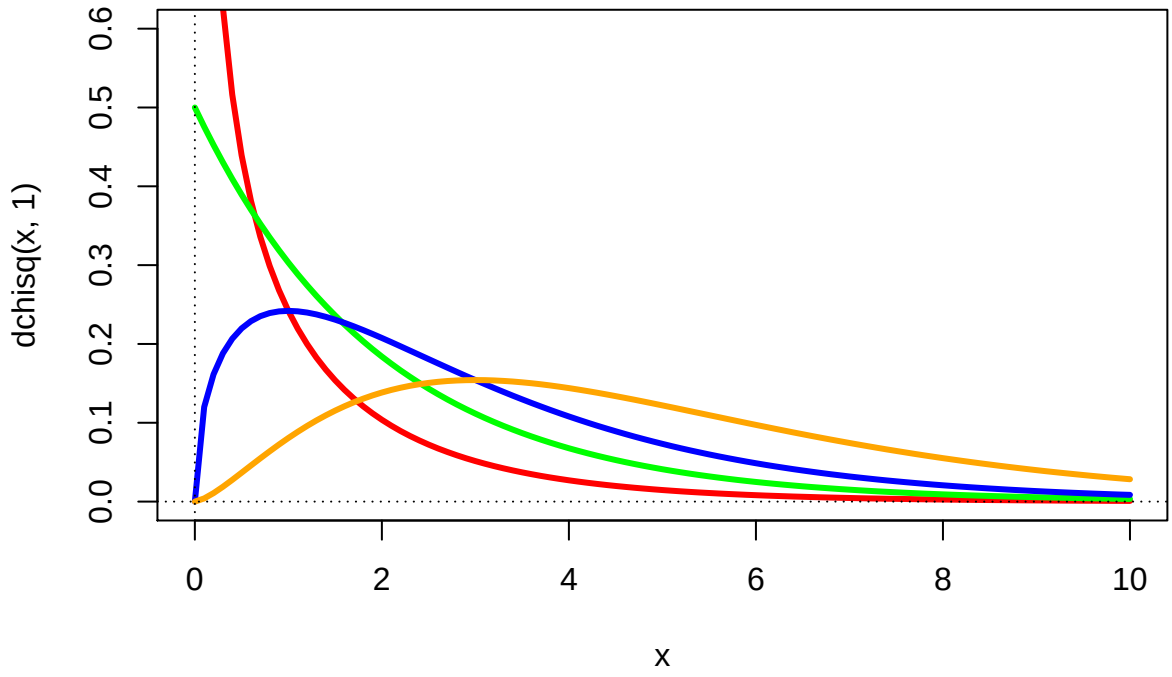


```r
#Loi beta(2,2)
x=rbeta(1000,shape1=2,shape2=2)
hist(x, probability=T, col='blue')
lines(density(x), col='red', lwd=2)
```



Loi théoriques

```r
curve(dchisq(x,1), xlim=c(0,10), ylim=c(0,.6), col='red', lwd=3)
curve(dchisq(x,2), add=T, col='green', lwd=3)
curve(dchisq(x,3), add=T, col='blue', lwd=3)
curve(dchisq(x,5), add=T, col='orange', lwd=3)
abline(h=0,lty=3)
abline(v=0,lty=3)
```



Loi des grands nombres

```
#Loi uniforme
mean(runif(10))
```

```
## [1] 0.3908858
```

```
mean(runif(1000))
```

```
## [1] 0.4997179
```

```
#Loi de cauchy
mean(rcauchy(10))
```

```
## [1] 2.249079
```

```
mean(rcauchy(1000))
```

```
## [1] -0.1200063
```

```
mean(rcauchy(100000))
```

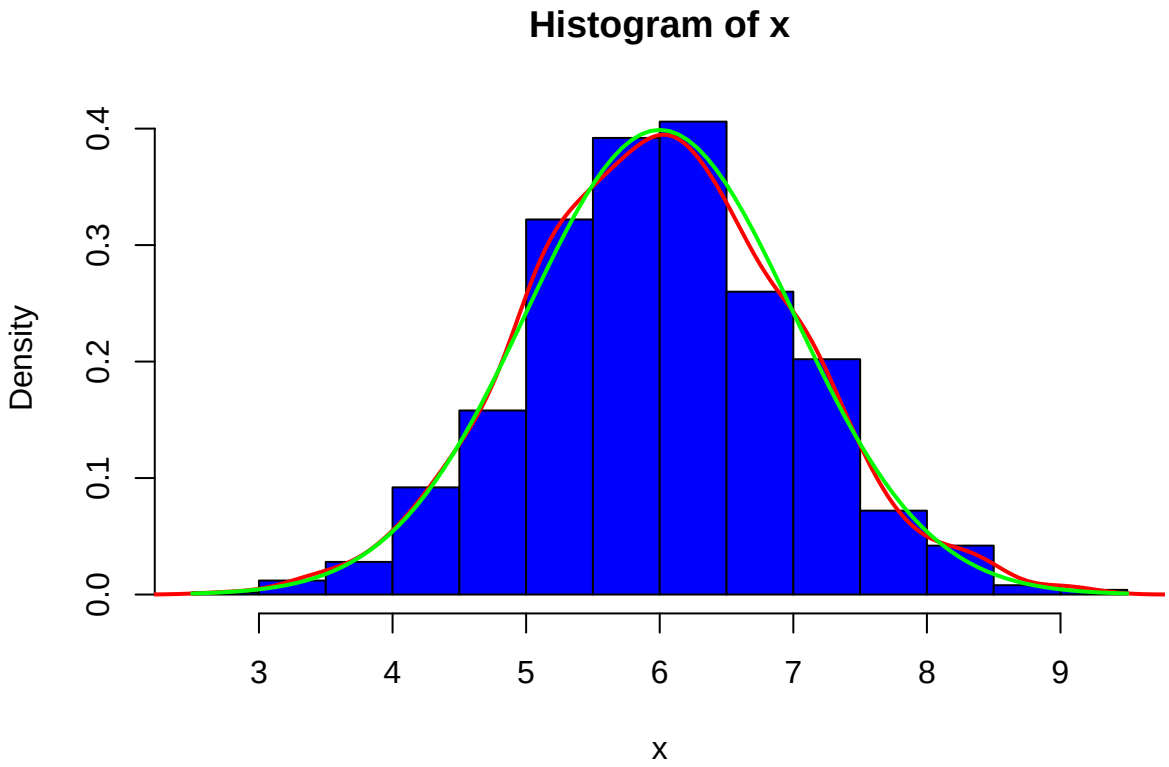
```
## [1] -0.2351628
```

Pour la loi uniforme, la convergence est rapide puisque l'estimation varie peu pour n=1000. En revanche, les valeurs varient très fortement pour l'échantillon selon une loi de cauchy même avec une taille d'échantillon importante.

Limite centrale et loi gaussienne

```
#Echantillon de la somme de 12 variables aleatoires uniformes sur [0,1]
x=rep(0,1000)
for (i in 1 :1000) x[i]=sum(runif(12))
```

```r
hist(x, col='blue', probability=T)
lines(density(x), col='red', lwd=2)
curve(dnorm(x,mean=6,sd=1), add=T, col='green', lwd=2)
```



Plus la variance σ^2 de la variable aléatoire est petite, plus la vitesse de convergence sera grande.

Echantillonage

Tirage aléatoire simple

```r
#Selection des echantillon d'apprentissage et de validation
set.seed(123)
npop=1000
testi<-sample(1 :npop,200)
appri<-setdiff(1 :npop,testi)
```

Avec remise ou bootstrap

```r
#Tirage sans remise
sample(1 :20,20)
```

```
##  [1]  5 19 11  9  7 14  6  4  3  2 20 12 18 10  1 13  8 16 17 15
```

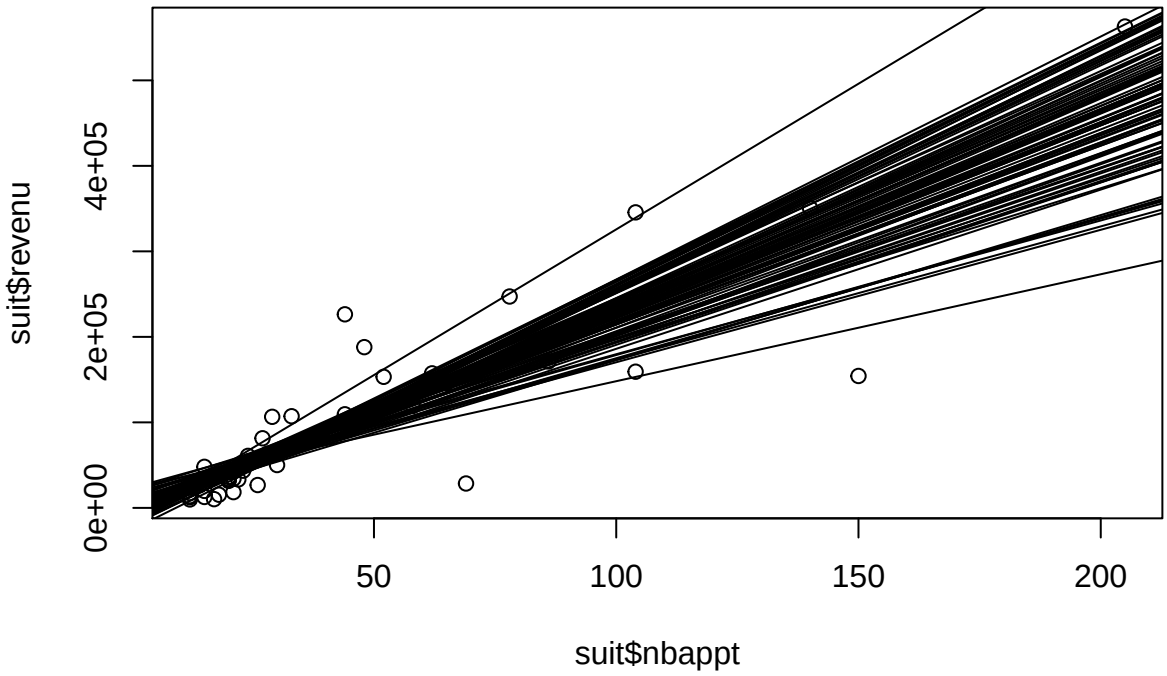
```r
#Bootstrap
sample(1 :20, 20, replace=TRUE)
```

```
##  [1]  6 20 15 14  2  8 10 12 14 19 13  9 11  2  6  8  4 17  4 17
```

On vérifie bien que l'échantillon bootstrap est avec remise car on obtient des valeurs identiques de la séquence.

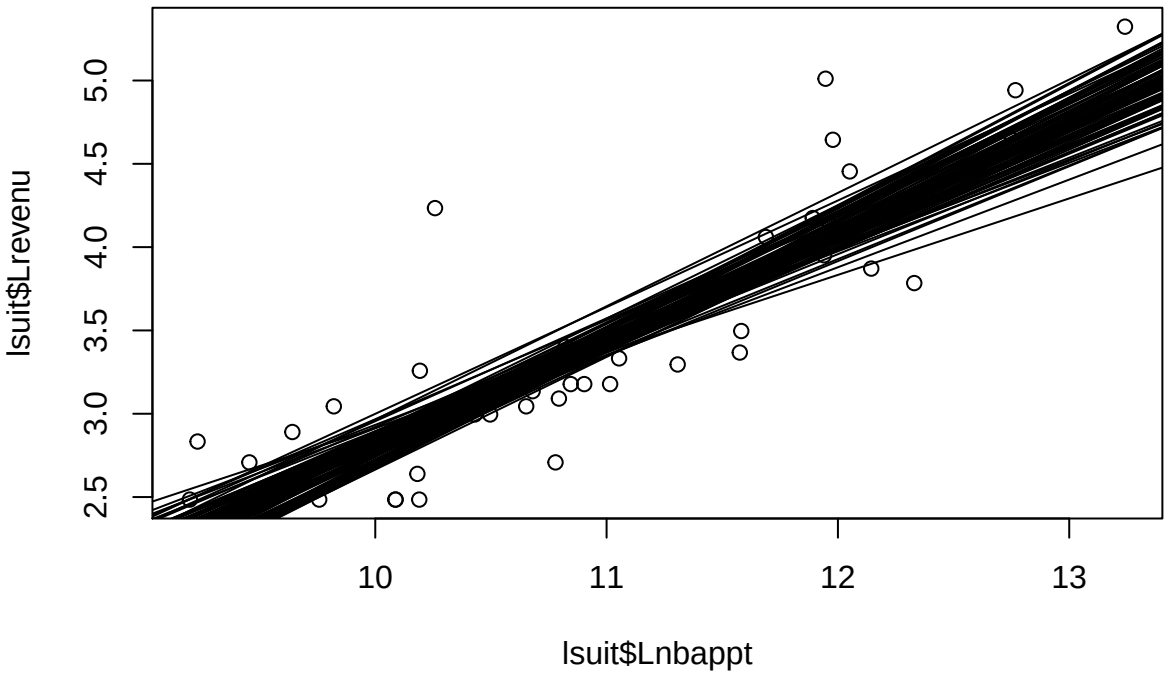
Régression simple

```
suit=read.table("/Users/thomaslaurent/Documents/Cours-M2/Data-Mining2/Etheme4/suitincom.dat.txt")
names(suit)=c("revenu","nbappt")
#Nuage de points
plot(suit$nbappt,suit$revenu)
#Iteeration du tirage de l'echantillon bootstrap et des estimations
for (i in 1 :100) {
  suit.b=suit[sample(47,47,replace=TRUE),]
  reg=lm(revenu~nbappt,data=suit.b)
  abline(reg)}
```



On estime ensuite sur le modèle linéaire log-log.

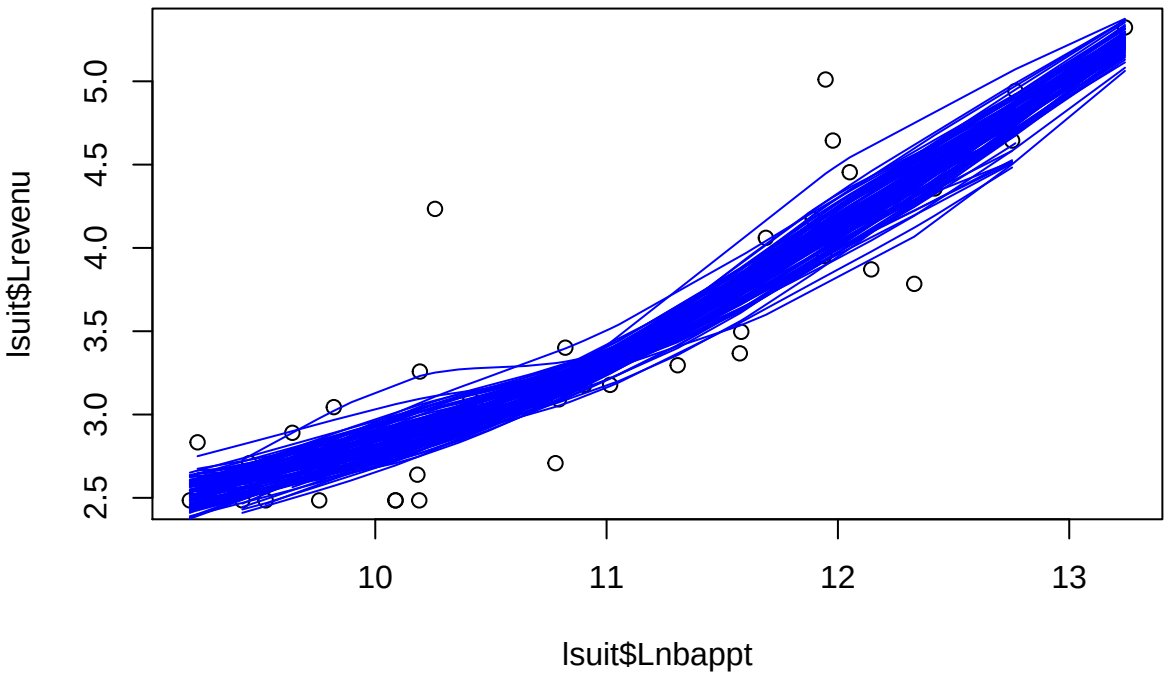
```
#On reitere le procede pour le modele log-log
lsuit=data.frame(log(suit$nbappt),log(suit$revenu))
names(lsuit)=c("Lrevenu","Lnbappt")
plot(lsuit$Lnbappt,lsuit$Lrevenu)
for (i in 1 :100) {
  suit.b=lsuit[sample(47,47,replace=TRUE),]
  reg=lm(Lrevenu~Lnbappt,data=suit.b)
  abline(reg)}
```



La région de confiance est plus étroite que pour le modèle précédent.

On réalise ensuite une régression non-paramétrique.

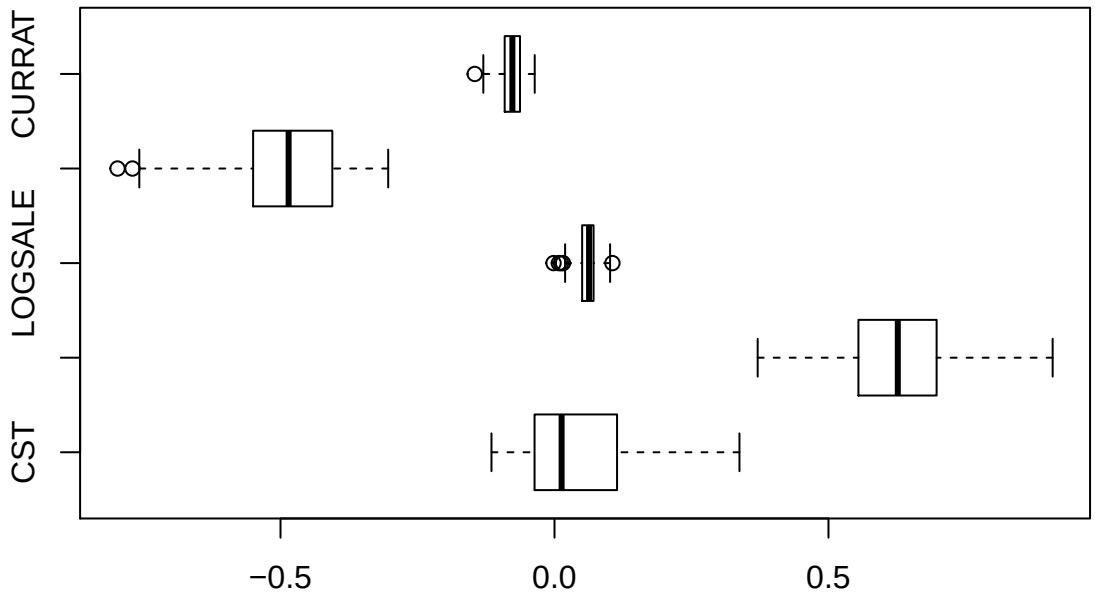
```
plot(lsuit$Lnbappt,lsuit$Lrevenu) >for (i in 1 :100) {
  suit.b=lsuit[sample(47,47,replace=TRUE),]
  lsuit.spl=smooth.spline(suit.b$Lnbappt,suit.b$Lrevenu,df=4)
  lines(lsuit.spl, col = "blue")}
```



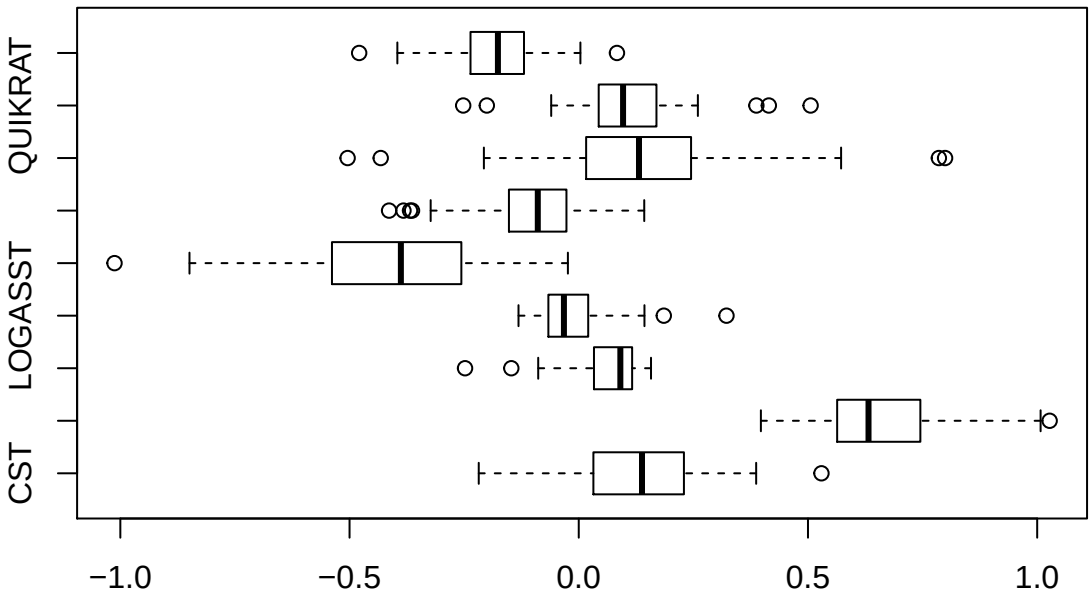

```
## logical(0)
```

Régression linéaire multiple

```
#Estimation du modele lineaire avec 5 variables
ukcomp1=read.table("/Users/thomaslaurent/Documents/Cours-M2/Data-Mining2/Etheme4/ukcomp1.datr.txt",head
stock=data.frame(matrix(0,100,5))
names(stock)=c("CST","CFTDT","LOGSALE","NFATAST","CURRAT")
for (i in 1 :100) {
Ib=sample(40,40,replace=TRUE)
stock[i,]=coef(lm(RETCAP~WCFTDT+LOGSALE+NFATAST+CURRAT,data=ukcomp1[Ib,]))}
boxplot(stock,horizontal=TRUE)
```




```
#Estimation du modele lineaire avec 9 variables
stock=data.frame(matrix(0,100,9))
names(stock)=c("CST","WCFTDT","LOGSALE","LOGASST","NFATAST","FATTOT","INVTAST","QUIKRAT","CURRAT")
for (i in 1 :100) {
Ib=sample(40,40,replace=TRUE)
stock[i,]=coef(lm(RETCAP~WCFTDT+LOGSALE+LOGASST+NFATAST+FATTOT+INVTAST+QUIKRAT+CURRAT,data=ukcomp1[Ib,]
boxplot(stock,horizontal=TRUE)
```



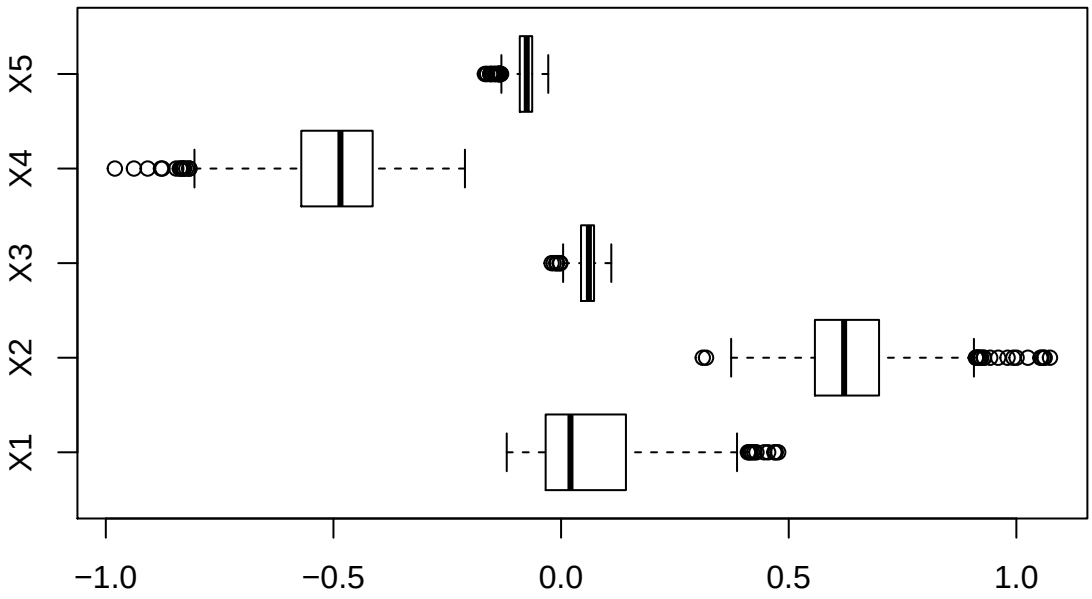
On observe que les dispersions sont plus importantes pour le modèle optimal au sens du R2 ajusté.

Librairie spécifique

```
library(boot)
#Fonction pour effectuer un bootstrap sur le modele
uk1.fun=function(d,i) coef(lm(RETCAP~WCFTDT+LOGSALE+NFATAST+CURRAT,data=d[i,]))
uk1.b=boot(ukcomp1,uk1.fun,R=999)
uk1.b
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = ukcomp1, statistic = uk1.fun, R = 999)
##
##
## Bootstrap Statistics :
##        original       bias    std. error
## t1*  0.02420409  0.033896177  0.12001319
## t2*  0.61188470  0.019231852  0.11255428
## t3*  0.06096181 -0.003138740  0.02186049
## t4* -0.47444843 -0.025034192  0.11888083
## t5* -0.06894891 -0.009742837  0.02254792
```

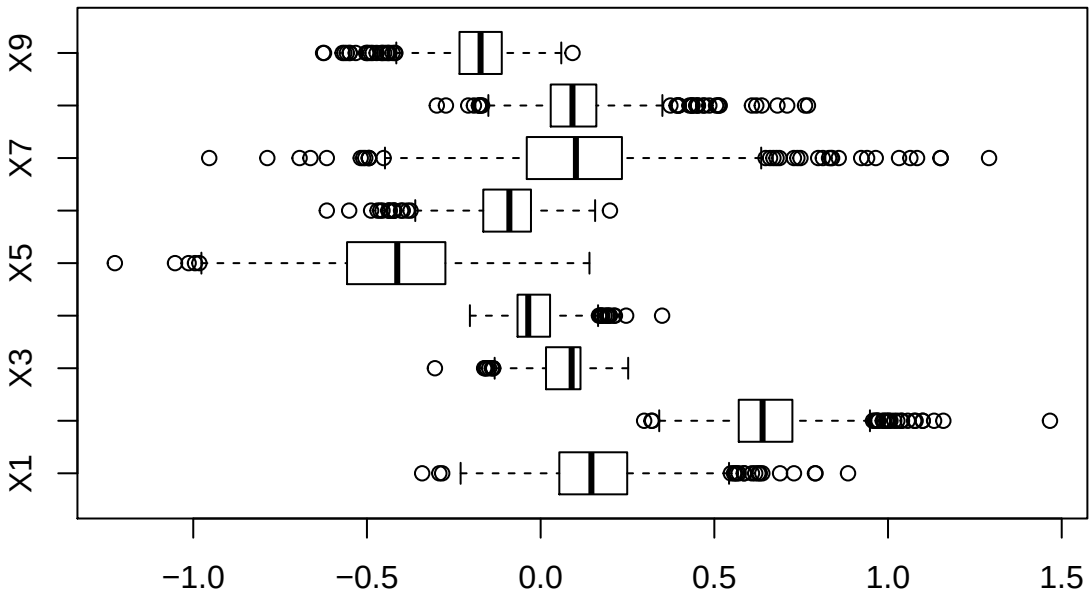
```
boxplot(data.frame(uk1.b$t),horizontal=TRUE)
```




```
#Autre modele
uk2.fun=function(d,i) coef(lm(RETCAP~WCFTDT+LOGSALE+LOGASST+NFATAST+FATTOT+INVTAST+QUIKRAT+CURRAT,data=
uk2.b=boot(ukcomp1,uk2.fun,R=999)
uk2.b
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = ukcomp1, statistic = uk2.fun, R = 999)
##
##
## Bootstrap Statistics :
##        original       bias    std. error
## t1*  0.09530273  0.063658245  0.15784069
## t2*  0.62561281  0.030150061  0.12886968
## t3*  0.09792035 -0.032971402  0.06901051
## t4* -0.04615771  0.029197878  0.06855796
## t5* -0.35556352 -0.065608016  0.20118111
## t6* -0.10596331  0.003592968  0.10671292
## t7*  0.18727272 -0.079000588  0.24136239
## t8*  0.13284141 -0.031983631  0.11836219
## t9* -0.19344432  0.014716305  0.09928170
```

```
boxplot(data.frame(uk2.b$t),horizontal=TRUE)
```



Validation croisée

Application du modèle linéaire

```
#Estimation du modele
uk1.glm = glm(RETCAP~WCFTDT+LOGSALE+NFATAST+CURRAT,family=gaussian,data=ukcomp1)
#Estimation de l'erreur leave-one-out
cv.err = cv.glm(ukcomp1,uk1.glm)
#Estimation sur 5 echantillons de taille 8
cv.err.5 = cv.glm(ukcomp1,uk1.glm, K=5)
cv.err$delta[1]
```

```
## [1] 0.008822106
```

```
cv.err.5$delta[1]
```

```
## [1] 0.009106019
```

```
#Autre modele
uk1.glm = glm(RETCAP~WCFTDT+LOGSALE+LOGASST+NFATAST+FATTOT+INVTAST+QUIKRAT+CURRAT,family=gaussian,data=u
cv.err = cv.glm(ukcomp1,uk1.glm)
cv.err.5 = cv.glm(ukcomp1,uk1.glm, K=5)
cv.err$delta[1]
```

```
## [1] 0.01328485
```

```
cv.err.5$delta[1]
```

```
## [1] 0.013689
```

L'erreur estimée étant plus faible pour le premier modèle et le nombre de paramètres étant moins important pour celui-ci, ce modèle parait meilleur.

```
muhat=uk1.glm$fitted
uk1.diag= glm.diag(uk1.glm)
mean((uk1.glm$y-muhat)^2/(1-uk1.diag$h)^2)
```

```
## [1] 0.01328485
```

On vérifie que l'erreur du modèle leave-one-out est égale à celle obtenue avec la fonction cv.glm.