COMPUTATIONAL STATISTICS WEEK 3

CORRECTION DES EXERCICES 6.7, 7.1, 7.2, 7.12, 7.24, 8.4 , 9.1 ET 9.2

ECRIT PAR

VINCENT LE MEUR

PROFESSEUR :

CHRISTIAN ROBERT
ENSAE

2017

Exercice 6.7

Nous considérons la matrice

$$\mathscr{P} = \begin{bmatrix} 0 & 0.4 & 0.6 & 0 & 0 \\ 0.65 & 0 & 0.35 & 0 & 0 \\ 0.32 & 0.68 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.12 & 0.88 \\ 0 & 0 & 0 & 0.56 & 0.44 \end{bmatrix}$$

Il s'agit d'une matrice de transition d'une chaîne de Markov avec 5 états : $\{1,2,3,4,5\}$
Etudions l'irréductibilité et l'apériodicité de la chaîne.

Cette matrice nous indique qu'on ne peut pas passer des états $\{1,2,3\}$ aux états $\{4,5\}$ (et inversement). Elle n'est donc pas *irréductible*.

Etudions son apériodicité :
A partir de l'état {1} ou {2} ou {3}, le nombre d'étapes des différents lacets pour retourner à l'étape {1}(ou {2} ou {3}) sont 2,3 et 4. Le plus grand commun diviseur (PGCD) est donc 1.
Enfin, à partir des états {4} ou {5}, le nombre d'étapes des différents lacets pour retourner à l'étape {4} (ou {5}) sont 1, 2 et 3. Le PGCD est donc ici aussi de 1.

Nous avons traité tous les états donc on peut conclure que la chaîne est *apériodique*.

Exercice 7.1

L'objet de cet exercice est de générer la moyenne d'une loi $\Gamma(4.3, 6.2)$ par trois méthodes différentes. Nous allons comparer à la fin la convergence de ces trois méthodes

a) On génère la moyenne à partir d'un algorithme Accept-Reject d'une loi candidate $\Gamma(4,6)$

On estime empiriquement la constante M de notre algorithme d'accept-reject :

```
### Input :
A=rgamma(100000,4,6)
B=rgamma(100000,4.3,6.2)
M= sum(B)/sum(A)
### Output  :
M =1.04218971336212
```

Calcul de la constante M

On crée notre générateur gamma4362 à partir d'un algorithme Accept Reject :

```
### Input :
gamma4362 = function(){
    u=runif(1)*M
    y=rgamma(1,4,6)
    while (u>(dgamma(y,4.3,6.2)/dgamma(y,4,6))){
        u=runif(1)*M
        y=rgamma(1,4,6)
    }
    return(y)
    }
```

Générateur d'individu suivant la loi Γ cible avec un algorithme Accept-Reject

Et on génère enfin notre échantillon simulé de taille n :

```
### Input :
rgamma4362 = function(n){
    A=rep(0,n)
    for (k in 1:n){
        A[k]=gamma4362()
    }
    return(A)
}
```

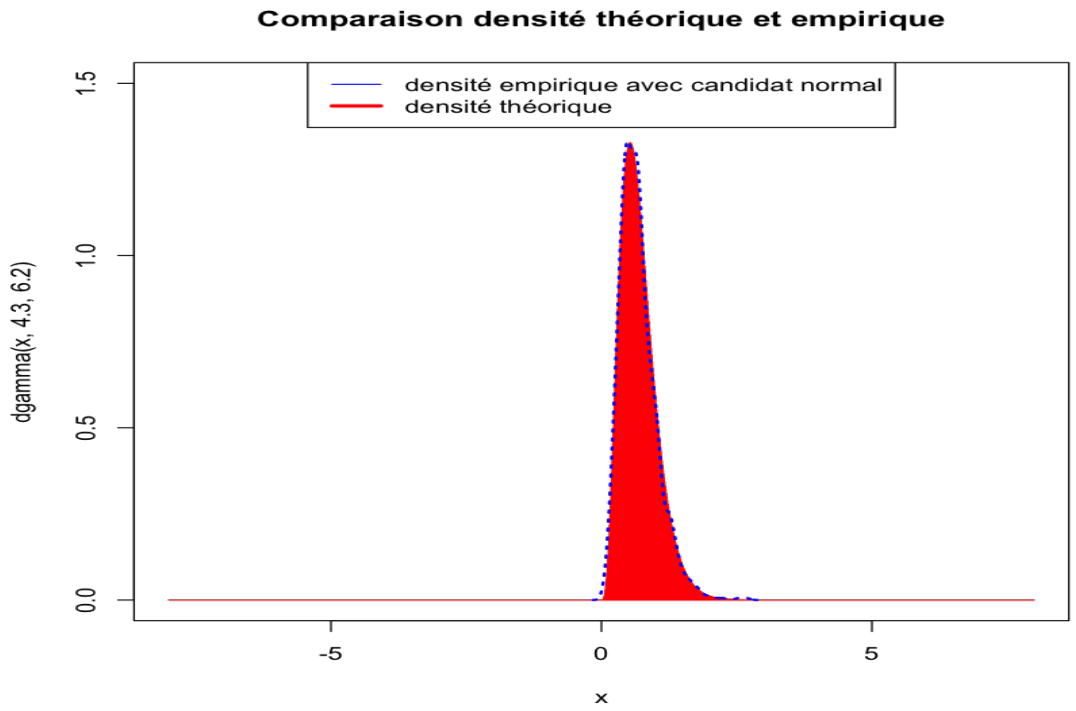
Création d'un l'échantillon de taille n

Enfin, on trace la densité empirique obtenue ainsi que la densité théorique :

```
### Input :
x=seq(from=-8,to=8,by=0.01)
plot(x,dgamma(x,4.3,6.2),type="h",col="red",main="Comparaison densité théorique et empirique",
    ylim=c(0,1.5))
lines(density(rgamma4362(length(x))),col="blue",lwd=2,lty=3)
legend("top", # la position sur le graphique
          c("densité empirique avec candidat Gamma ACCEPT_REJCT" ,"densité théorique"), # le
              texte pour chaque courbe
          col=c("blue", "red"), # La couleur de chaque courbe
          lwd=c(1,3,1), # L'épaisseur de chaque courbe
          lty=c(1,1,3) # Le type de trait de chaque courbe
       )
```

Comparaison densité théorique et empirique

On obtient le tracé suivant :



Notre densité empirique est bien proche de la densité théorique

La moyenne est alors donnée par :

```
### Input :
N=10^4
mean(rgamma4362(N))
### Output :
0.68553628919931
 )
```

Calcul de la moyenne

b) On génère la moyenne à partir d'un algorithme Metropolis-Hastings d'une loi candidate $\Gamma(4,6)$

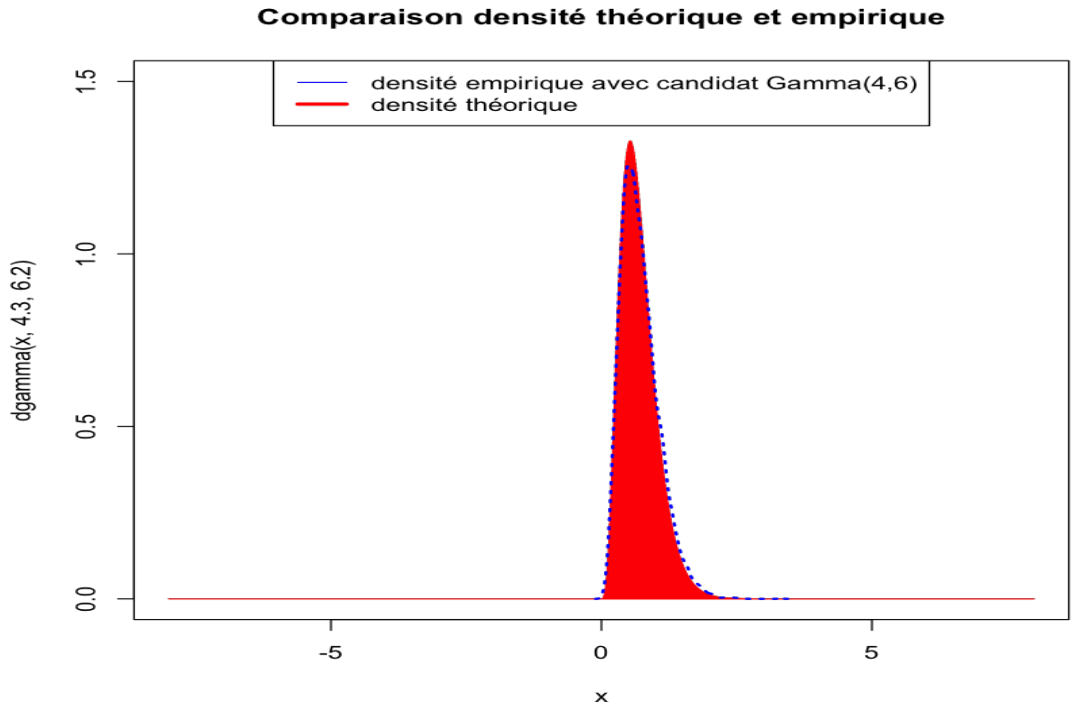
On génère notre chaîne de Markov à partir d'un algoirthme Metropolis-Hastings, puis on trace à nouveau les densité théorique et empirique :

```
### Input :
N=10^4
X71b=rep(runif(1),N)     # initialisation de la chaine
for (i in 2:N){
    Y=rgamma(1,4,6)    # On utilise un candidat Gamma(4,6)
    rho=dgamma(Y,4.3,6.2)*dgamma(X71b[i-1],4,6)/(dgamma(X71b[i-1],4.3,6)*dgamma(Y,4,6))
    X71b[i]=X71b[i-1] + (Y-X71b[i-1])*(runif(1)<min(rho,1)) }
```

Création d'un échantillon de taille N à partir d'un algorithme de Metropolis-Hastings

```
### Input :
x=seq(from=-8,to=8,by=0.01)
plot(x,dgamma(x,4.3,6.2),type="h",col="red",main="Comparaison densité théorique et empirique",
    ylim=c(0,1.5))
lines(density(X71b),col="blue",lwd=2,lty=3)
legend("top", # la position sur le graphique
          c("densité empirique avec candidat Gamma(4,6)" ,"densité théorique"), # le texte pour
              chaque courbe
          col=c("blue", "red"), # La couleur de chaque courbe
          lwd=c(1,3,1), # L'épaisseur de chaque courbe
          lty=c(1,1,3) # Le type de trait de chaque courbe
          )
```

Comparaison densité théorique et empirique



La moyenne est ici :

```
### Input :
mean(X71b)
### Output :
0.713476020907605
```

Calcul de la moyenne

c) On génère la moyenne à partir d'un algorithme Metropolis-Hastings d'une loi candidate $\Gamma(5,6)$

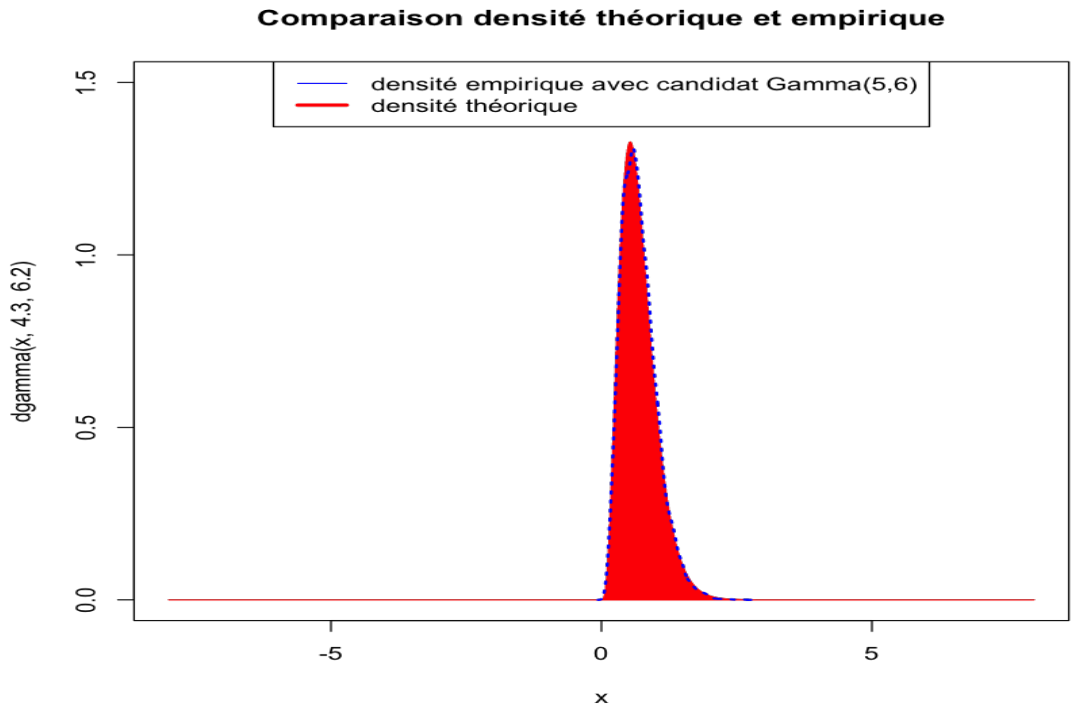
On effectue la même démarche que précédemment en partant d'une loi $\Gamma(5,6)$:

```
### Input :
N=10^4
X71c=rep(runif(1),N)     # initialisation de la chaine
for (i in 2:N){
    Y=rgamma(1,5,6)    # On utilise un candidat Gamma(5,6)
    rho=dgamma(Y,4.3,6.2)*dgamma(X71c[i-1],5,6)/(dgamma(X71c[i-1],4.3,6)*dgamma(Y,5,6))
    X71c[i]=X71c[i-1] + (Y-X71c[i-1])*(runif(1)<min(rho,1)) }
```

Création d'un échantillon de taille N à partir d'un algorithme de Metropolis-Hastings

```
### Input :
x=seq(from=-8,to=8,by=0.01)
plot(x,dgamma(x,4.3,6.2),type="h",col="red",main="Comparaison densité théorique et empirique",
    ylim=c(0,1.5))
lines(density(X71c),col="blue",lwd=2,lty=3)
legend("top", # la position sur le graphique
          c("densité empirique avec candidat Gamma(5,6)" ,"densité théorique"), # le texte pour
              chaque courbe
          col=c("blue", "red"), # La couleur de chaque courbe
          lwd=c(1,3,1), # L'épaisseur de chaque courbe
          lty=c(1,1,3) # Le type de trait de chaque courbe
       )
```

Comparaison densité théorique et empirique



On obtient la moyenne :

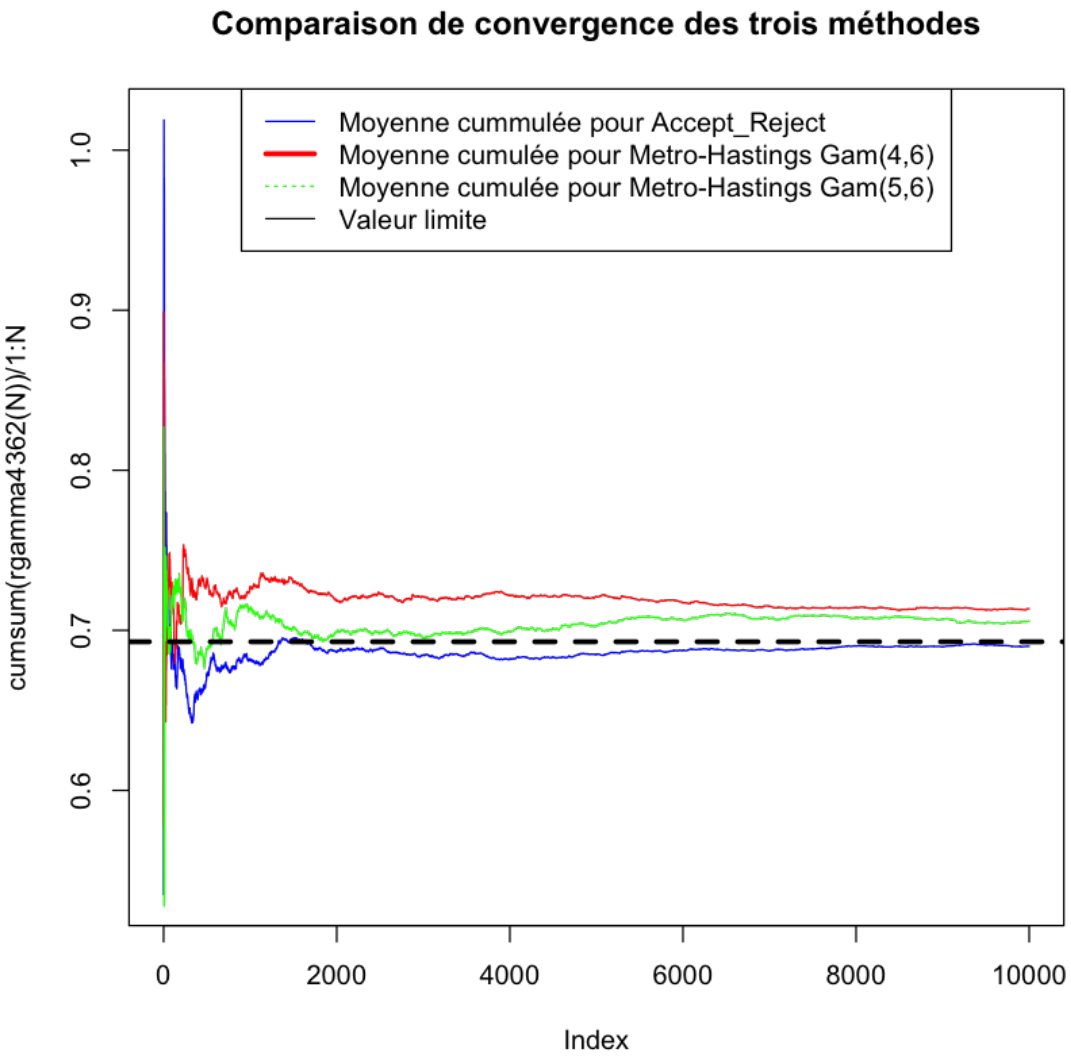
```
### Input :
mean(X71c)
### Output :
0.705873542582404
```

Calcul de la moyenne

Les trois méthodes donnent des résultats proches. Observons plus précisément la convergence de chacune des méthodes :

```
### Input :
N=10^4
plot(cumsum(rgamma4362(N))/1:N,type="l",col="blue",main="Comparaison de convergence des trois
    méthodes")
lines(cumsum(X71b)/1:N,type="l",col="red")
lines(cumsum(X71c)/1:N,type="l",col="green")
abline(a=mean(rgamma(100000,4.3,6.2)),0,col="black",lw=3,lty=2)
legend("top", # la position sur le graphique
          c("Moyenne cummulée pour Accept_Reject" ,"Moyenne cumulée pour Metro-Hastings Gam(4,6)
              ","Moyenne cumulée pour Metro-Hastings Gam(5,6)","Valeur limite"), # le texte pour
               chaque courbe
          col=c("blue", "red","green","black"), # La couleur de chaque courbe
          lwd=c(1,3,1), # L'épaisseur de chaque courbe
          lty=c(1,1,3) # Le type de trait de chaque courbe
       )
```

Comparaison des convergences des trois méthodes



En conclusion nos trois méthodes convergent globalement vers la "vraie" valeur. Seulement il semble que la convergence de l'Accept-Reject soit plus rapide et moins biaisée. Il semble également que le candidat $\Gamma(5,6)$ offre une meilleur convergence pour l'algorithme de Metropolis-Hastings

Exercice 7.2

Dans cet exercice,on cherche à calculer la moyenne d'une loi de Student à 4 degrés de liberté de densité : $f(x) = \frac{\Gamma(\frac{5}{2})}{\Gamma(\frac{4}{2})}\frac{1}{\sqrt{4\pi}}(1+\frac{x^2}{4})^{-(4+1)/2}$.

On va alors s'appuyer des propriétés de convergence de l'algorithme de Metropolis-Hastings (la vrai valeur étant ici 0).
On approxime donc la moyenne $\int_{-\infty}^{+\infty} xf(x)dx$ par $\frac{1}{T}\Sigma_{t=1}^{T} X^t$ où X^t correspond au $t^{ième}$ noeud de la chaîne de Markov associée.

a) Utilisation d'une loi $\mathcal{N}(0,1)$ comme densité candidate dans un algorithme de Metropolis Hastings

On génère notre chaîne de Markov :

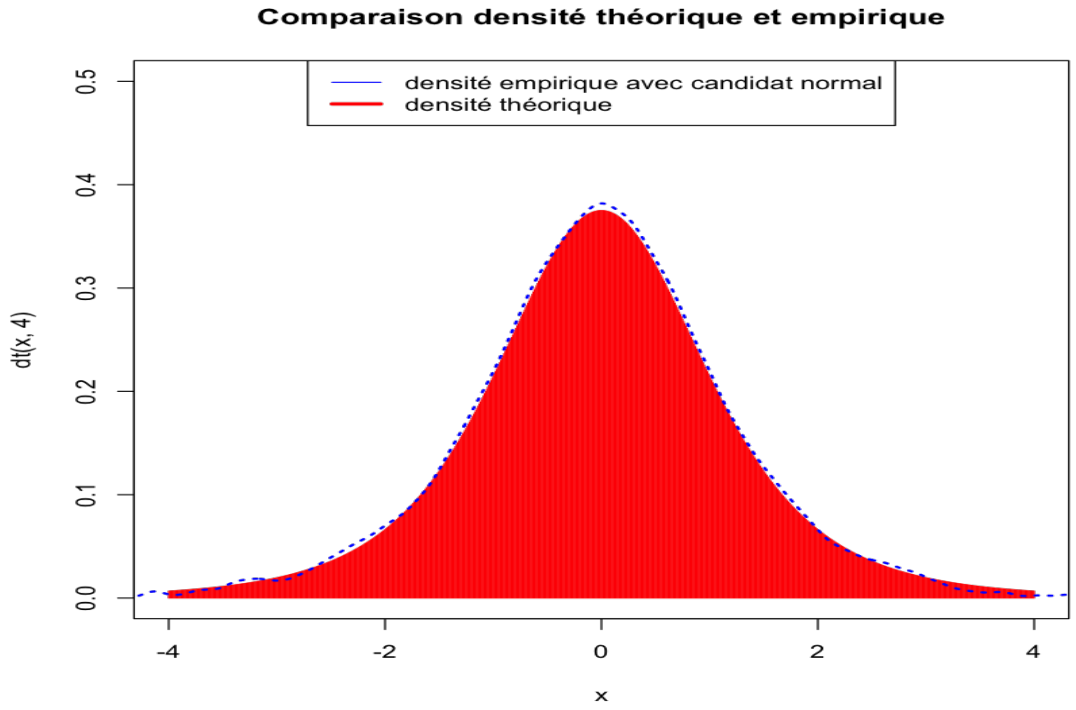
```
### Input :
N=10^5
Xa=rep(runif(1),N)     # initialisation de la chaine
for (i in 2:N){
    Y=rnorm(1,0,1)    # On utilise un candidat normal
    rho=dt(Y,4)*dnorm(Xa[i-1])/(dt(Xa[i-1],4)*dnorm(Y))
    Xa[i]=Xa[i-1] + (Y-Xa[i-1])*(runif(1)<min(rho,1)) }
```

Création d'un échantillon Markovien avec un algorithme de Metropolis-Hastings

On trace la densité obtenue et la densité théorique :

```
### Input :
x=seq(from=-4,to=4,by=0.01)
plot(x,dt(x,4),type="h",col="red",main="Comparaison densité théorique et empirique",ylim=c
    (0,0.5))
lines(density(Xa),col="blue",lwd=2,lty=3)
legend("top", # la position sur le graphique
          c("densité empirique avec candidat normal" ,"densité théorique"), # le texte pour
              chaque courbe
          col=c("blue", "red"), # La couleur de chaque courbe
          lwd=c(1,3,1), # L'épaisseur de chaque courbe
          lty=c(1,1,3) # Le type de trait de chaque courbe
      )
```

Comparaison densité empirique et densité théorique



Nous savons que la moyenne dans ce cas doit valoir 0. Notre méthode nous donne la valeur :

```
### Input :
mean(Xa)
### Output :
-0.00835036510738568
```

Calcul de la moyenne

b) Utilisation d'une loi de Student t à 2 degrès de libertés comme densité candidate dans un algorithme de Metropolis Hastings

On génère notre chaîne de Markov :

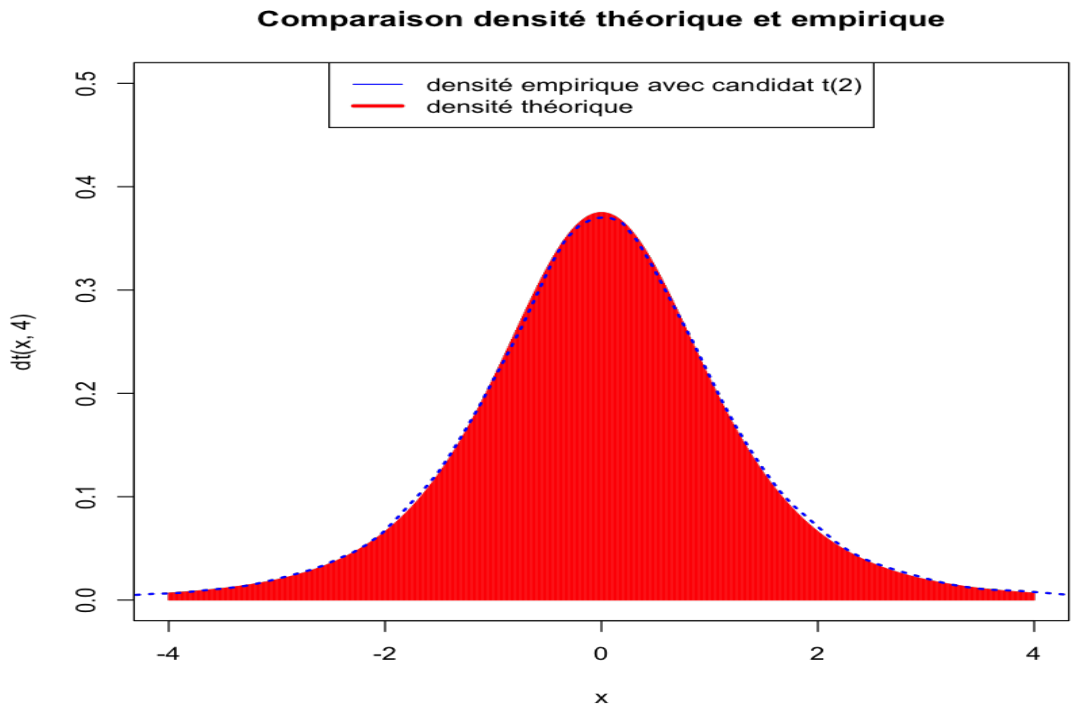
```
### Input :
N=10^5
Xb=rep(runif(1),N)     # initialisation de la chaine
for (i in 2:N){
    Y=rt(1,2)   # On utilise un candidat student à 2 degrès de liberté
    rho=dt(Y,4)*dt(Xb[i-1],2)/(dt(Xb[i-1],4)*dt(Y,2))
    Xb[i]=Xb[i-1] + (Y-Xb[i-1])*(runif(1)<min(rho,1)) }
```

Création d'un échantillon Markovien avec un algorithme de Metropolis-Hastings

On trace ici encore la densité obtenue et la densité théorique :

```
### Input :
x=seq(from=-4,to=4,by=0.01)
plot(x,dt(x,4),type="h",col="red",main="Comparaison densité théorique et empirique",ylim=c
    (0,0.5))
lines(density(Xb),col="blue",lwd=2,lty=3)
legend("top", # la position sur le graphique
          c("densité empirique avec candidat t(2)" ,"densité théorique"), # le texte pour chaque
              courbe
          col=c("blue", "red"), # La couleur de chaque courbe
          lwd=c(1,3,1), # L'épaisseur de chaque courbe
          lty=c(1,1,3) # Le type de trait de chaque courbe
       )
```

Comparaison densité empirique et densité théorique



On constate que le comportement aux queues est plus proche qu'avec le candidat normal.
Ici la moyenne vaut :

```
### Input :
mean(Xb)
### Output :
-0.000179382160827294
```

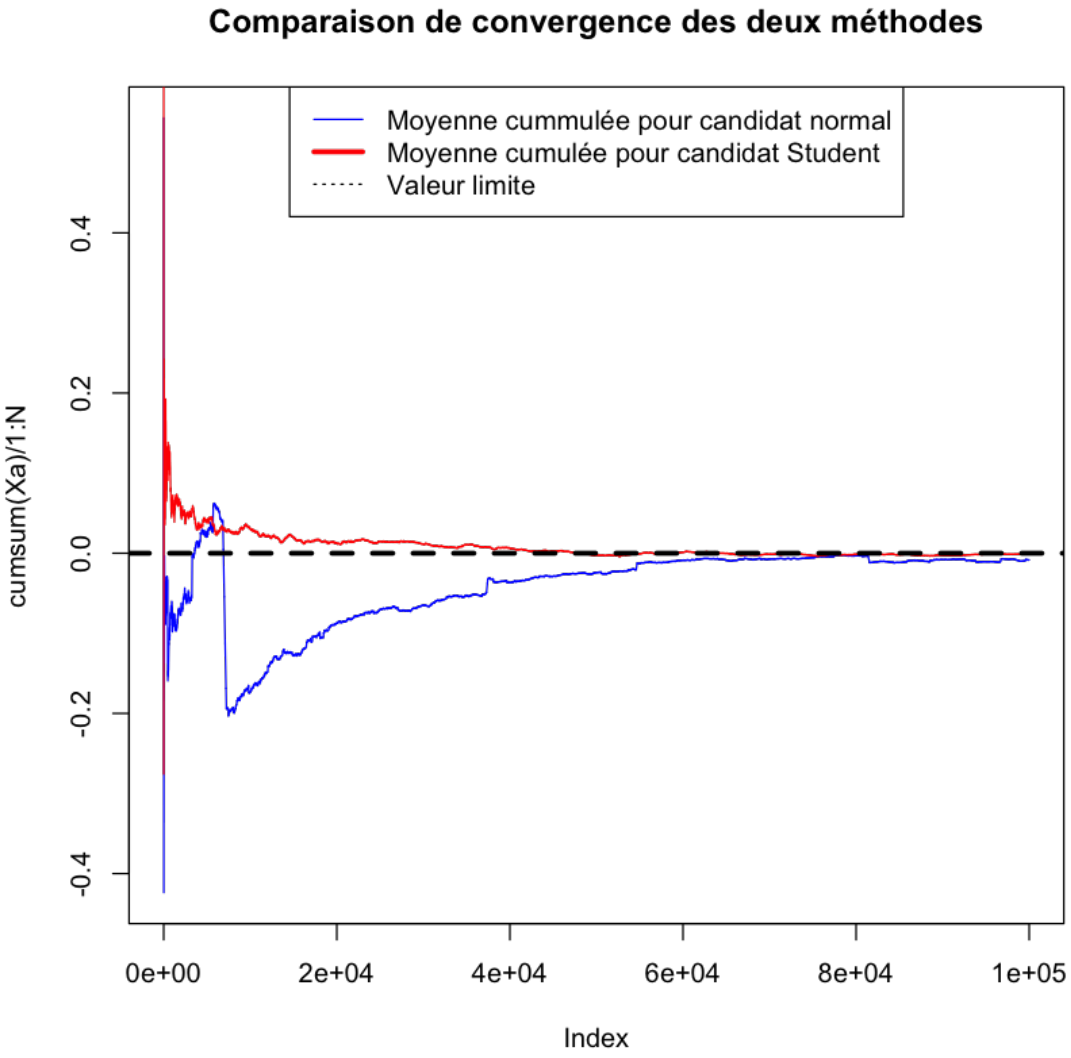
Comparaison densité empirique et densité théorique

Ce résultat est plus proche de la vraie moyenne.

Comparons à présent les différentes vitesses de convergence :

```
### Input :
plot(cumsum(Xa)/1:N,type="l",col="blue",main="Comparaison de convergence des deux méthodes")
lines(cumsum(Xb)/1:N,type="l",col="red")
abline(a=0,0,col="black",lw=3,lty=2)
legend("top", # la position sur le graphique
          c("Moyenne cummulée pour candidat normal" ,"Moyenne cumulée pour candidat Student","
              Valeur limite"), # le texte pour chaque courbe
          col=c("blue", "red","black"), # La couleur de chaque courbe
          lwd=c(1,3,1), # L'épaisseur de chaque courbe
          lty=c(1,1,3) # Le type de trait de chaque courbe
      )
```

Comparaison des convergences des deux méthodes



On conclut que le candidat Student entraîne une convergence plus rapide ainsi qu'un biais réduit.

Exercice 7.12

Dans cet exercice nous allons générer une loi normale $\mathcal{N}(0,1)$ à partir de lois uniformes sur $[-\delta,\delta]$ en utilisant un algorithme de Metropolis-Hastings.

a) On étudie l'influence de δ sur le taux de convergence et sur le taux d'acceptation

Le taux de convergence est représenté ici par la différence des quantiles à 90% et 10% pour chaque échantillon généré. On forme donc la fonction conv-rate qui va à partir de δ générer un échantillon issue d'un algorithme de Metropolis-Hastings à partir de lois uniforme sur $[-\delta,\delta]$ puis calculer les différences interquantilles (à 90% et 10%) :

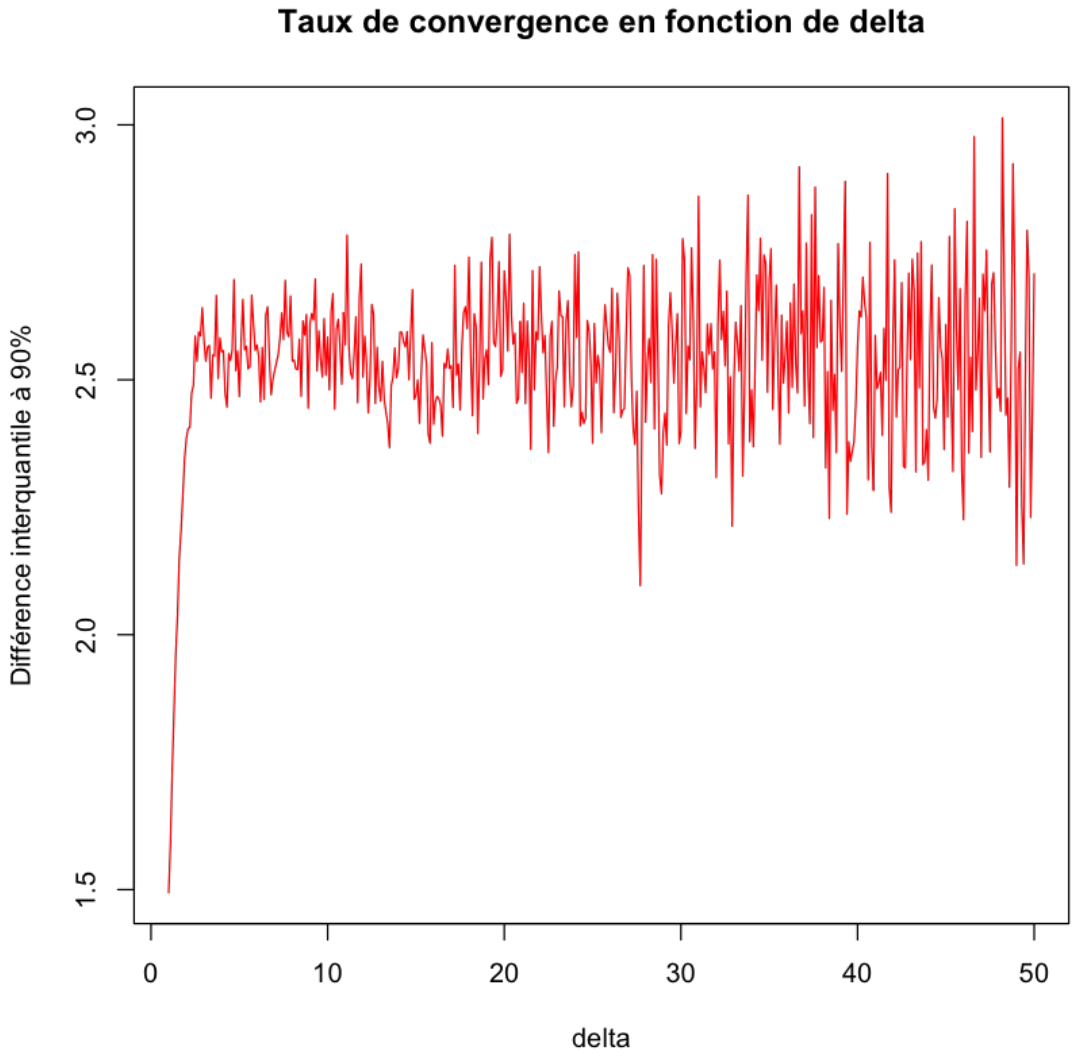
```
### Input :
conv-rate= function(delta){
    N=10^4
    X=rep(runif(1),N)    # initialisation de la chaine
      #On initialise un compteur pour le taux d'acceptation
    for (i in 2:N){
        Y=runif(1,min=-delta,max=delta)   # On utilise un candidat uniforme sur [-delta,delta]
        rho=dnorm(Y,mean=0,sd=1)/(dnorm(X[i-1],mean=0,sd=1))
        X[i]=X[i-1] + (Y-X[i-1])*(runif(1)<min(rho,1))
        }
    q10=quantile(X,c(0.1)) # calcul du quantile à 10%
    q90=quantile(X,c(0.9)) # calcul du quantile à 90%
    return(q90[[1]]-q10[[1]])
    }
```

Fonction conv-rate calculant la différence interquantilles

On trace alors le taux de convergence en fonction de δ :

```
### Input :
x=seq(from=1,to=50,by=0.1)

A<-c()
for (k in 1:length(x) ){
    A[k]=conv-rate(x[k])
}
plot(x,A,type="l",col="red",main="Taux de convergence en fonction de delta",xlab="delta",ylab="
    Différence interquantile à 90%")
```

Calcul du Taux de convergence en fonction de δ



Le taux de convergence a tendance à augmenter avec δ
On peut comparer ces valeurs au taux de convergence d'une loi $\mathcal{N}(0,1)$ qui est donnée par :

```
### Input :
b= quantile(rnorm(N,0,1),c(0.9))
a= quantile(rnorm(N,0,1),c(0.1))
c=b[[1]]-a[[1]]
c
### Output :
2.56871183890119
```

Calcul du Taux de convergence d'une loi Normale standard

En conclusion notre taux de convergence oscille autour de la "vraie" valeur. Seulement l'amplitude de ces oscillations augmente avec δ ce qui est préjudicable.

On étudie à présent le taux d'acceptation, qui représente le quotient $\frac{Nacept}{Nsim}$ avec Nacept le nombre de fois où l'algorithme de Metropolis-Hastings a accepté la proposition et Nsim le nombre total d'itérations.

Pour cela on mets en place la fonction accpt-rate qui à partir d'un delta donné va générer un échantillon issu d'un algorithme de Metropolis-Hastings à partir de lois uniforme sur $[-\delta, \delta]$ et d'en calculer le taux d'acceptation :

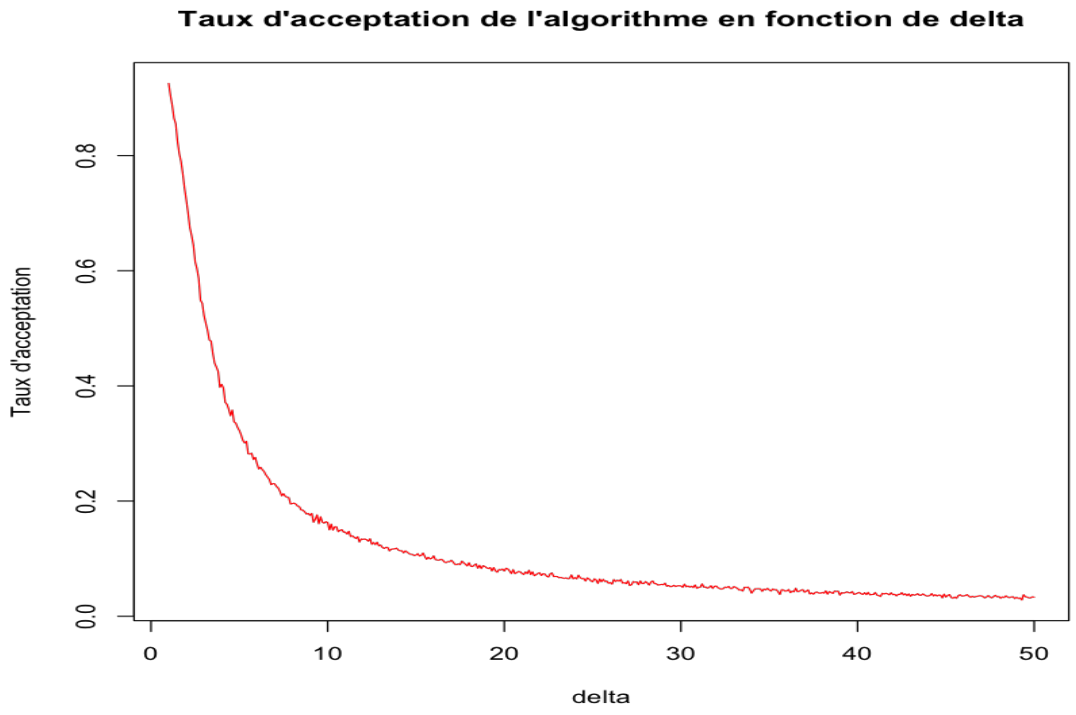
```
### Input :
accpt-rate= function(delta){
    N=10^4
    X=rep(runif(1),N)     # initialisation de la chaine
    cpt=0  #On initialise un compteur pour le taux d'acceptation
    for (i in 2:N){
        Y=runif(1,min=-delta,max=delta)    # On utilise un candidat uniforme sur [-delta,delta]
        rho=dnorm(Y,mean=0,sd=1)/(dnorm(X[i-1],mean=0,sd=1))
        X[i]=X[i-1] + (Y-X[i-1])*(runif(1)<rho)
        if (runif(1)<rho){cpt<-cpt+1 # Mise en place du compteur
        }}
    return(cpt/N) # retourne le taux d'acceptation
}
```

Calcul du Taux d'acceptation en fonction de δ

On trace alors ce Taux d'acceptation en fonction de delta :

```
### Input :
x=seq(from=1,to=50,by=0.1)

A<-c()
for (k in 1:length(x) ){
    A[k]=accpt-rate(x[k])
}
plot(x,A,type="l",col="red",main="Taux d'acceptation de l'algorithme en fonction de delta",xlab=
    "delta",ylab="Taux d'acceptation")
```

Calcul du Taux d'acceptation en fonction de δ



On constate que ce taux d'acceptation chute rapidement quand δ augmente. Cela vient du fait que la probabilité rho diminue quand delta augmente.

b) Déterminons la valeur de δ qui minimise la variance de la moyenne empirique

Tout d'abbord la variance empirique de la moyenne est donnée par : $v = \frac{1}{n-1}\sum_{j=1}^{n}[X^j - \widehat{\mu}]^2$ avec $\widehat{\mu}$ la moyenne estimée empiriquement : $\widehat{\mu} = \frac{1}{n}\sum_{j=1}^{n} X^j$

On va donc chercher pour n fixé, à minimiser cette quantité. Créons tout d'abord notre fonction vardelta qui va générer la variance à partir de δ :

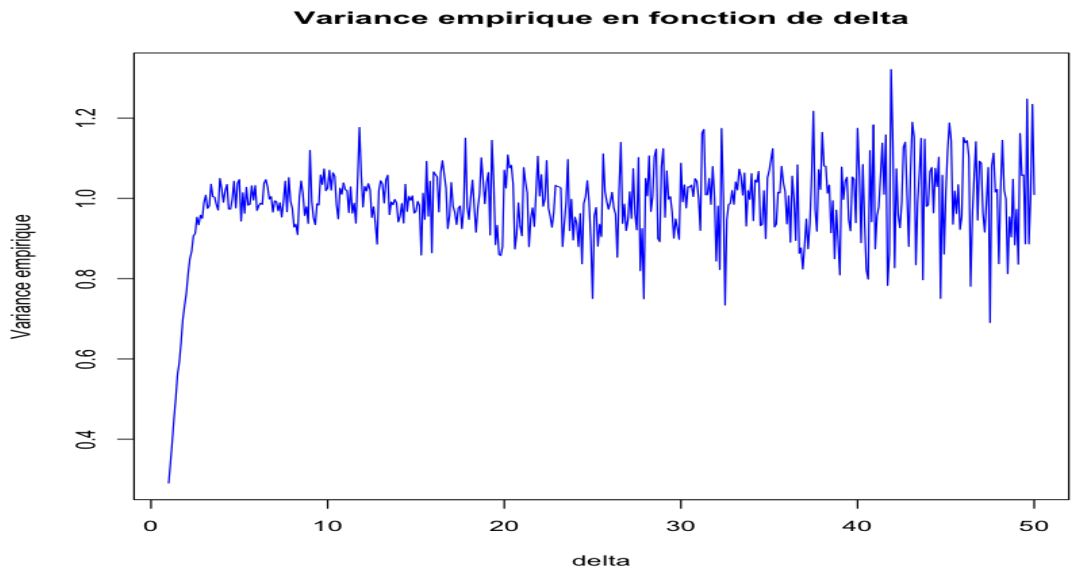
```
### Input :
vardelta = function(delta){
    N=10^4
    X=rep(runif(1),N)     # initialisation de la chaine

    for (i in 2:N){
        Y=runif(1,min=-delta,max=delta)    # On utilise un candidat uniforme sur [-delta,delta]
        rho=dnorm(Y,mean=0,sd=1)/(dnorm(X[i-1],mean=0,sd=1))
        X[i]=X[i-1] + (Y-X[i-1])*(runif(1)<rho)
        }
    mu=(1/N)*sum(X)  # On calcule la moyenne empirique
    Z=(X-mu)^2
    v=(1/(N-1))*sum(Z) # On en déduit la variance empirique
    return(v)
}
```

Calcul de la variance empirique en fonction de δ

Le tracé ci-dessous des différentes valeurs de variance en fonction de delta ne nous permet pas de conclure de manière précise :

```
### Input :
delta=seq(from=1,to=50,by=0.1)
A<-c()
for (k in 1:length(delta) ){
    A[k]=vardelta(delta[k])
}
plot(delta,A,type="l",col="blue",main="Variance empirique en fonction de delta",xlab="delta",
    ylab="Variance empirique")
```

Variance empirique en fonction de δ



Nous allons utiliser une exploration stochastique entre 0 et 50 :

```
### Input :
recherchesto = function(f,U){
    temp<-c()
    k=1
    repeat {
    if (k==length(U)) break
    temp[k]=f(U[k])
    k<-k+1
    }
    result1=min(temp) #On cherche le minimum de f
    rang=which(temp==result1) #Pour obtenir la valeur de notre échantillon uniforme minimisant f
    return(U[rang]) # La fonction retourne la valeur qui minimise f
}
```

Création d'une fonction d'exploration stochastique

Nous obtenons la valeur suivante :

```
### Input :
Udelta = runif(10000,min=5,max=50) # On commence à 5 pour être dans le comportement asymptotique
     de la variance
recherchesto(var_delta,Udelta)
### Output :
45.4583869478665
```

Calcul du delta minimisant la variance empirique

Cette valeur semble coincider avec un minima local du graph mais non à un minimum global. De plus les valeurs sont très volatiles à chaque exécution. Cela vient de notre technique d'estimation qui "n'accroche" pas nécessairement les extrema globaux.

Exercice 7.24

Ici, on cherche à implémenter un algorithme de Metropolis-Hastings pour générer une loi normale $\mathcal{N}(0,1)$ à partir de densités uniformes $q(.|x)$ de forme $U[-x-\delta,-x+\delta]$. On cherche à étudier l'influence de δ sur la corrélation (négative) des X^t.

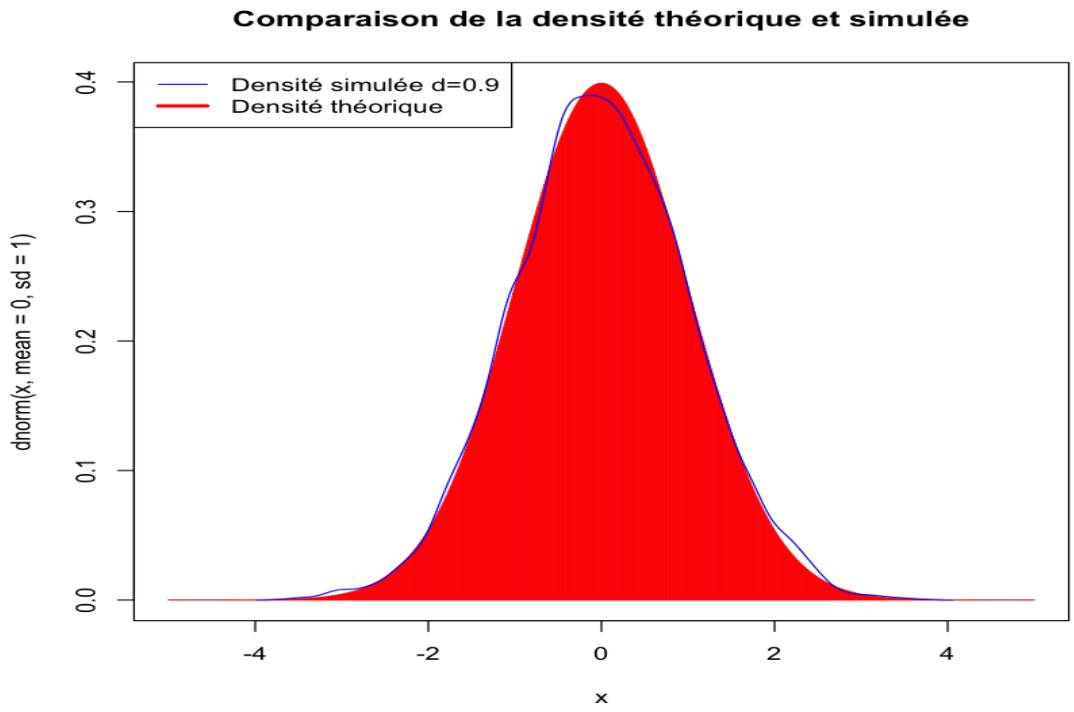
Tout d'abord formons notre Chaîne de Markov en fonction de δ grâce à la fonction Markovchain724 et traçons la densité :

```
### Input :
Markovchain724= function(delta){
    N=10^4
    X=rep(runif(1),N)     # initialisation de la chaine
    for (i in 2:N){
        Y=runif(1,min=-X[i-1]-delta,max=-X[i-1]+delta)    # On utilise un candidat uniforme sur
            [-x-delta,-x+delta]
        rho=dnorm(Y,mean=0,sd=1)/(dnorm(X[i-1],mean=0,sd=1))
        X[i]=X[i-1] + (Y-X[i-1])*(runif(1)<min(rho,1))
        }
    return(X)
}
```

Création d'un échantillon Markovien avec un algorithme Metropolis-Hastings

```
### Input :
x=seq(from=-5,to=5,by=0.01)
plot(x,dnorm(x,mean=0,sd=1),type="h",col="red",main="Comparaison de la densité théorique et
    simulée")
lines(density(Markovchain724(0.9)),col="blue")
legend("topleft", # la position sur le graphique
          c("Densité simulée d=0.9" ,"Densité théorique"), # le texte pour chaque courbe
          col=c("blue", "red"), # La couleur de chaque courbe
          lwd=c(1,3,1), # L'épaisseur de chaque courbe
          lty=c(1,1,3) # Le type de trait de chaque courbe
      )
```

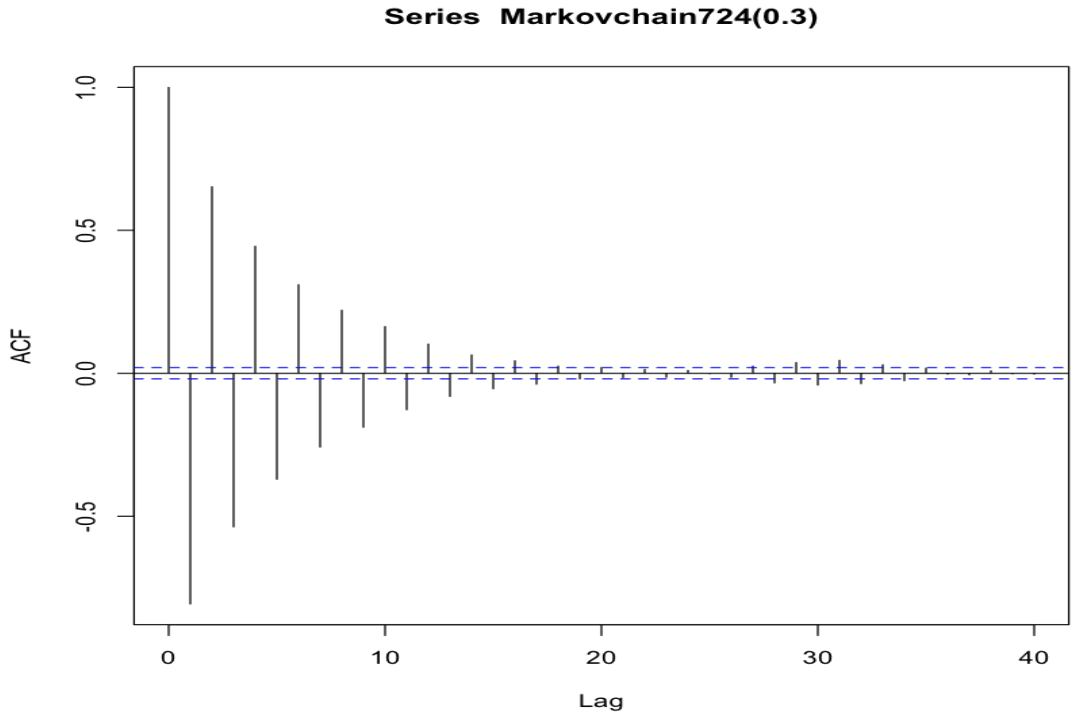
Comparaison densité théorique et empirique à delta fixé

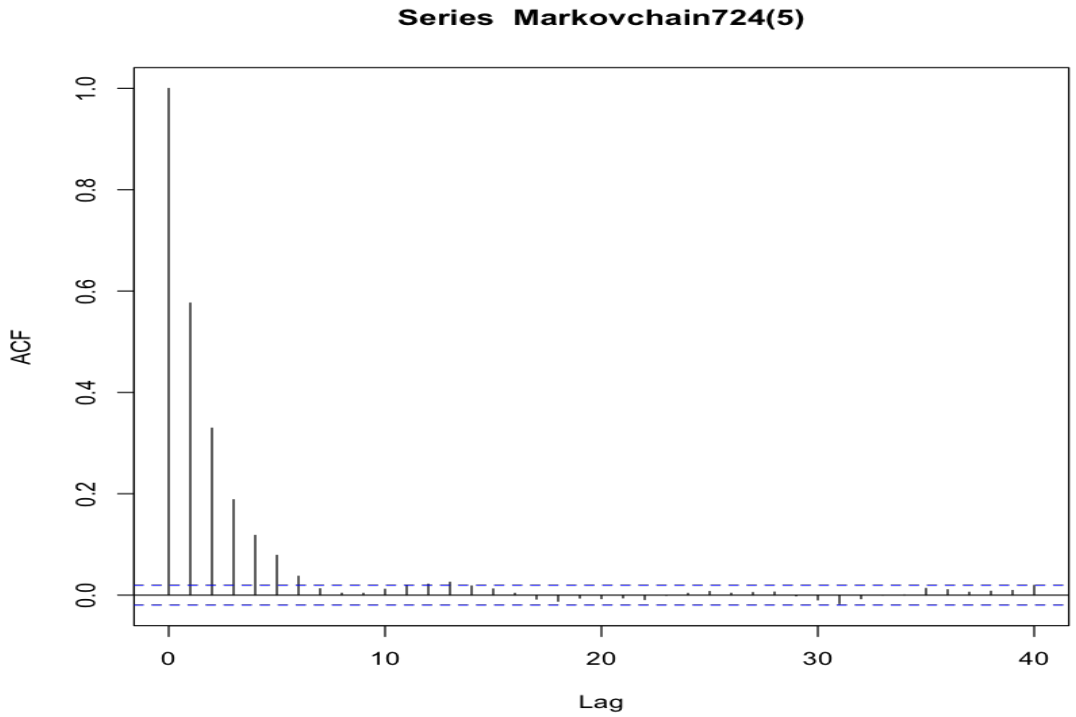


Nous allons à présent observer la corrélation existant entre les X^t en utilisant des ACF.
Voici deux ACF prises pour $\delta = 0.3$ et $\delta = 5$:

```
### Input :
acf(Markovchain724(0.3))
acf(Markovchain724(5))
```

AutoCorrelation Function pour deux valeurs de delta





On constate pour la première ACF une corrélation négative entre les coefficients alors qu'elle est positive pour l'autre. Cela vient du fait que notre fonction Markovchain724 initialise la chaîne de Markov pour x sur [0,1]. Si $\delta >> 1$ alors le passage à l'étape d'après utilisant une loi uniforme sur [-x-delta,-x+delta] ne modifie pas l'intervalle (-x et x sont alors déjà dans le même intervalle) d'où l'absence de corrélation négative.

Nous allons à présent tracer de manière plus précise le 2ième coefficient de l'ACF pour chaque delta à partir de la fonction correl :

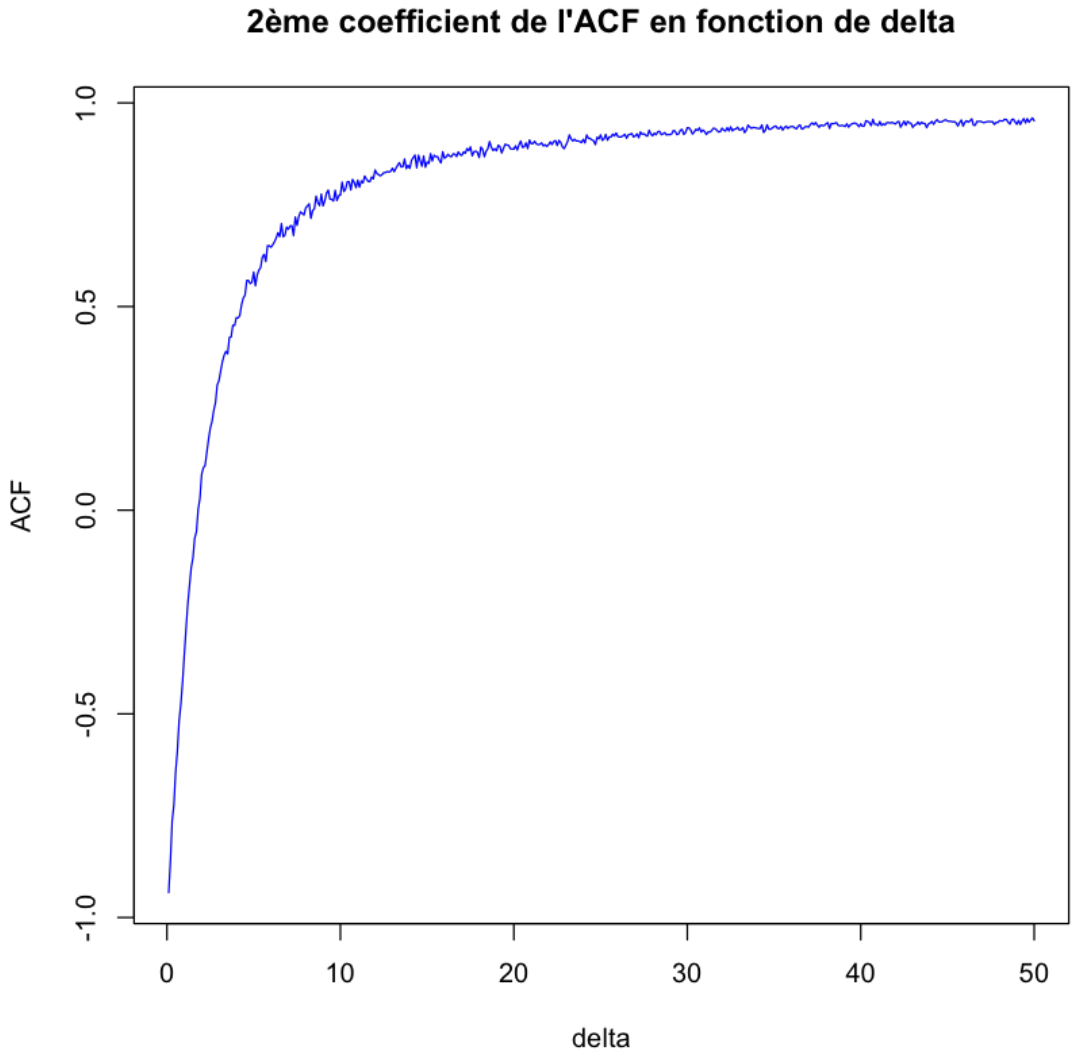
```
### Input :
correl = function(delta){
    X=Markovchain724(delta) # On génère notre chaîne de Markov
    Y=acf(X,plot=FALSE)[[1]][2]  # On récupère le 2ième coefficient de l'ACF
    return(Y)
}
```

Création de la fonction correl qui retourne le 2ème coefficient de l'ACF

Traçons alors l'évolution de ce 2ème coefficient de l'ACF en fonction de δ pour $\delta \in [0.1,50]$:

```
### Input :
delta=seq(from=0.1,to=50,by=0.1)
B<-c()
for (k in 1:length(delta) ){
    B[k]=correl(delta[k])
}
plot(delta,B,type="l",col="blue",main="2ème coefficient de l'ACF en fonction de delta",xlab="
    delta",ylab="ACF")
```

Evolution du 2ème coefficient de l'ACF avec delta



En conclusion on observe bien une corrélation négative pour des valeurs de $\delta < 1.79$ car notre algorithme de Metropolis-Hastings initie sa première valeur sur $[0,1]$. Au delà, l'algorithme reste dans un même intervalle ce qui explique l'absence de corrélation négative. x et -x sont alors dans le même intervalle. On pourrait initialiser notre chaîne de Markov sur un intervalle plus grand pour palier cette situation

Exercice 8.4

Dans cet exercice, on considère une densité $f(x) \propto \exp^{-x^d}$ avec $d < 1$ et on cherche à mettre en place un échantillonneur par tranches et évaluer sa performance pour $d \in \{0.1, 0.25, 0.4\}$

Tout d'abord créons notre fonction f notée ici fd :

```
### Input :
fd = function (x,d){
    return(exp(-(x^d)))
}
```

Création de la fonction fd

Nous allons, pour chaque valeur de d, générer des échantillons issus d'un échantillonneur de Gibbs avec la fonction sampler84 suivante :

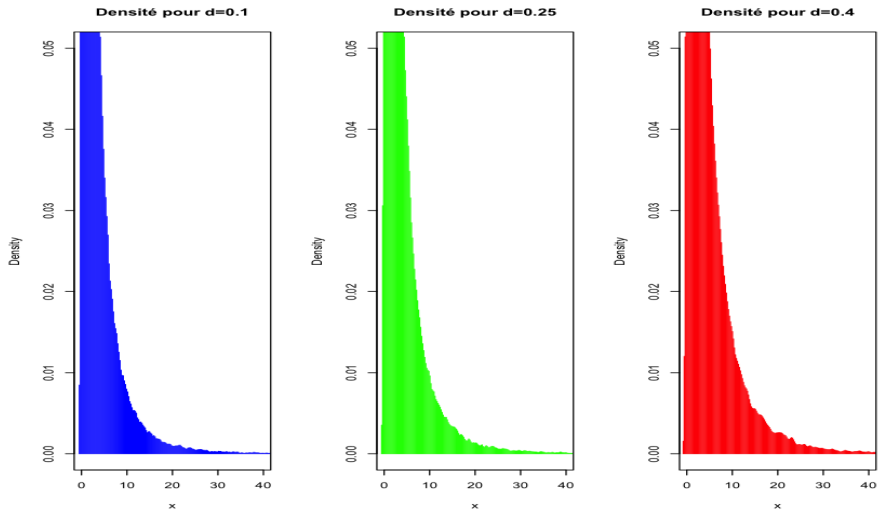
```
### Input :
sampler84 = function(d,Nsim) {
x=rep(1,Nsim) #initialisation de la chaîne
u=rep(0,Nsim) #initialisation de la chaîne
for (j in 2:Nsim){
    u[j]=runif(1,min=0,max=(0.5)*fd(x[j-1],d))
    x[j]=runif(1,min=0,max=(log(2*u[j]))^2) }
    return(x)
    }
```

Générateur d'échantillon de Gibbs

Traçons alors les densités correspondantes :

```
### Input :
par(mfcol=c(1,3))
plot(density(sampler84(0.1,Nsim)),type="h",col="blue",xlim=c(0,40),ylim=c(0,0.05),main="Densité
    pour d=0.1",xlab="x")
plot(density(sampler84(0.25,Nsim)),type="h",col="green",xlim=c(0,40),ylim=c(0,0.05),main="
    Densité pour d=0.25",xlab="x")
plot(density(sampler84(0.4,Nsim)),type="h",col="red",xlim=c(0,40),ylim=c(0,0.05),main="Densité
    pour d=0.4",xlab="x")
```

Tracé des densités

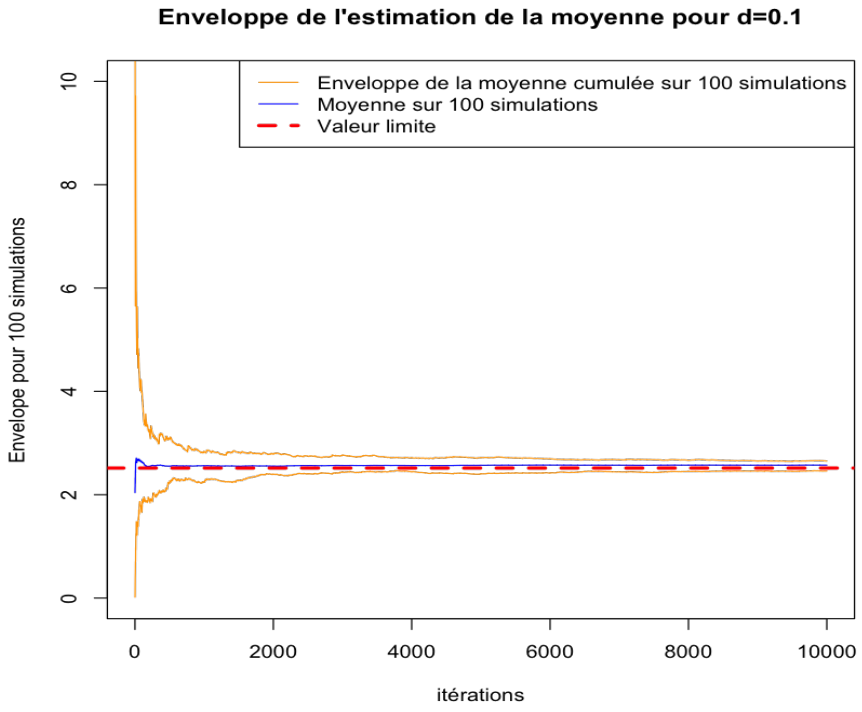


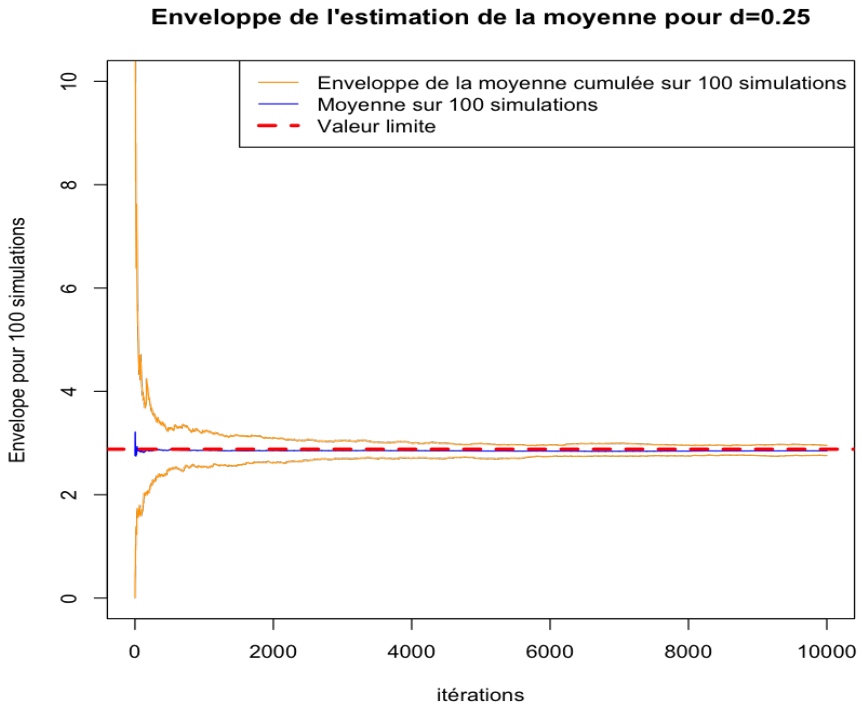
Nous allons évaluer la performance de l'algorithme pour chaque valeur de d. Pour cela nous allons créer la fonction plotsample84 qui va pour une valeur de d fixé tracer le maximum, le minimum et la moyenne sur 100 simulations de la moyenne cumulée de chaque série :

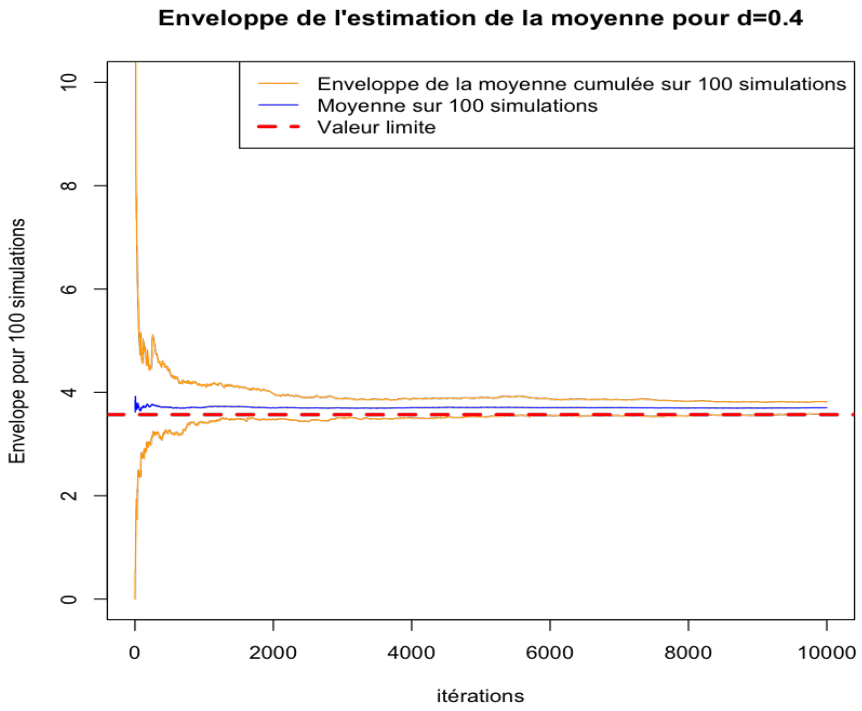
```
### Input :
plotsampler84 = function(d,titre){
simulations = 100
iterations = 10000
nb_iterations = 1:iterations
y1= sampler84(d, iterations * simulations)
mat_y = matrix(y1,nrow = simulations,ncol = iterations)
cummean_y = t(apply(mat_y, 1, cumsum) / nb_iterations)
  min_y  = apply(cummean_y, 2, min)
  max_y  = apply(cummean_y, 2, max)
  mean_y = apply(cummean_y, 2, mean)
plot(nb_iterations, max_y, type = "l", col = "orange",ylim=c(0,10) ,
       main = titre, ,
       xlab = "itérations", ylab = "Envelope pour 100 simulations")
  lines(nb_iterations, min_y, col = "orange")
  lines(nb_iterations, mean_y, col = "blue")
  abline(a = mean(sampler84(d,iterations)), b =0 , col = "red", lw = 3, lty = 2)
  legend("topright", legend = c("Enveloppe de la moyenne cumulée sur 100 simulations", "Moyenne
      sur 100 simulations", "Valeur limite"),
         lwd = c(1, 1, 3), lty = c(1,1,2), col = c("orange", "blue", "red"))
         }
plotsampler84(0.1,"Enveloppe de l'estimation de la moyenne pour d=0.1")
plotsampler84(0.25,"Enveloppe de l'estimation de la moyenne pour d=0.25")
plotsampler84(0.4,"Enveloppe de l'estimation de la moyenne pour d=0.4")
```

Fonction génératrice du tracé de moyenne cumulée sur 100 simulations

Nous obtenons les tracés suivant :







On constate que la convergence de notre algorithme semble à peu près uniforme selon les différentes valeurs de d.

Exercice 9.1

On considère l'échantillonneur de GIBS suivant :

Algorithm A.33 –Two-stage Gibbs sampler–

Take $X_0 = x_0$
For $t = 1, 2, \ldots,$ generate

1. $Y_t \sim f_{Y|X}(\cdot|x_{t-1})$; [A.33]
2. $X_t \sim f_{X|Y}(\cdot|y_t)$.

a) Montrons que les séquences (X_i) , (Y_i) et (X_i, Y_i) sont des chaînes de Markov.

Dans cet algorithme :

$$Y_t \sim f_{Y|X}(.|x_{t-1})$$
$$X_{t+1} \sim f_{X|Y}(.|y_t)$$

Ainsi à chaque t, Y_t et X_t ne dépendent que de Y_{t-1} et X_{t-1}.
Nous avons donc la même chose pour le couple (X^t, Y^t) qui ne dépend que de (X^{t-1}, Y^{t-1}) et non de X_{t-2},Y_{t-2},X_{t-3},Y_{t-3}....

Ainsi (X_i) , (Y_i) et (X_i, Y_i) sont bien des chaînes de Markov.

b) Montrons que $f_X(.)$ et $f_Y(.)$ sont les densités invariantes de X et Y

On considère le noyau de transition de Xt :

$$K(x, x^*) = \int_y f_{Y|X}(y|x) f_{X|Y}(x^*|y) dy$$

Et on calcule alors la quantité :

$$\int_x K(x, x^*) f_X(x) dx$$

$$\int_x K(x, x^*) f_X(x) dx = \int_x \int_y f_{Y|X}(y|x) f_{X|Y}(x^*|y) f_X(x) dx dy$$
$$= \int_y f_{X|Y}(x^*|y) dy \int_x f_{Y|X}(y|x) f_X(x) dx$$
$$= \int_y f_{X|Y}(x^*|y) \int_x f_{X,Y}(x, y) dx dy$$
$$= \int_y f_{X,Y}(x^*, y) dy = f_X(x^*)$$

Ainsi $f_X(.)$ est bien la densité invariante de X_t
Une démonstration analogue permet de montrer que $f_Y(.)$ est bien la densité invariante de Y_t

Exercice 9.2

L'objet de cet exercice est de générer (X,Y) une loi normale bivariée de moyenne 0 de variance 1 et de corrélation ρ à partir d'un échantillonneur de Gibbs. On étudie ensuite la variable $Z = X^2 + Y^2$ et on cherche à déterminer $P(Z > 2)$

On génère tout d'abord (X,Y) avec les fonction sampler92X et sampler92Y :

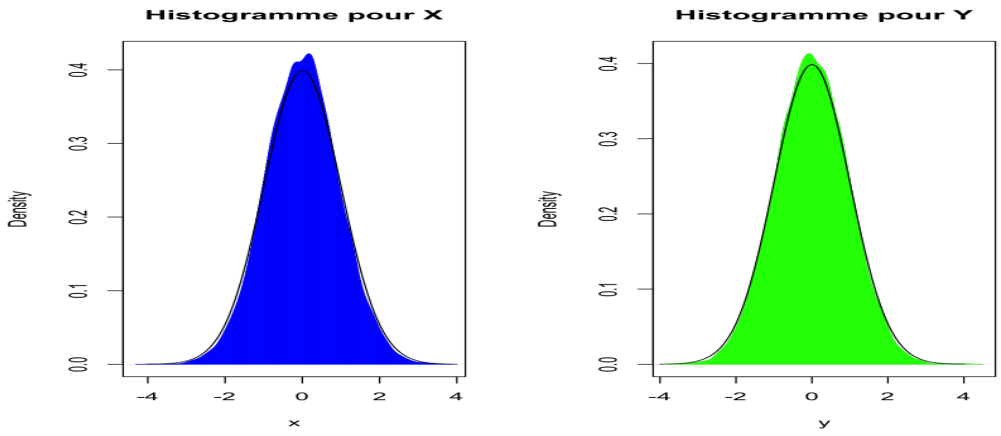
```
### Input :
sampler92X = function(rho,Nsim){
X=rep(0,Nsim)
Y=rep(0,Nsim)
for (t in 2:Nsim){
  X[t]=rnorm(1,rho*Y[t-1],1-(rho*rho))
  Y[t]=rnorm(1,rho*X[t],1-(rho*rho))

}
return(X)}
sampler92Y = function(rho,Nsim){
X=rep(0,Nsim)
Y=rep(0,Nsim)
for (t in 2:Nsim){
  X[t]=rnorm(1,rho*Y[t-1],1-(rho*rho))
  Y[t]=rnorm(1,rho*X[t],1-(rho*rho))

}
return(Y)}
```

Création des échantillons de Gibbs bivariés

On obtient les tracés suivant pour les densités obtenues (la densité théorique étant superposée en noir) :

```
### Input :
x=seq(from=-4,to=4,by=0.1)
par(mfcol=c(1,2))
plot(density(sampler92X(0.3,Nsim)),type="h",col="blue",main="Histogramme pour X",xlab="x")
lines(x,dnorm(x))
plot(density(sampler92Y(0.3,Nsim)),type="h",col="green",main="Histogramme pour Y",xlab="y")
lines(x,dnorm(x))
```

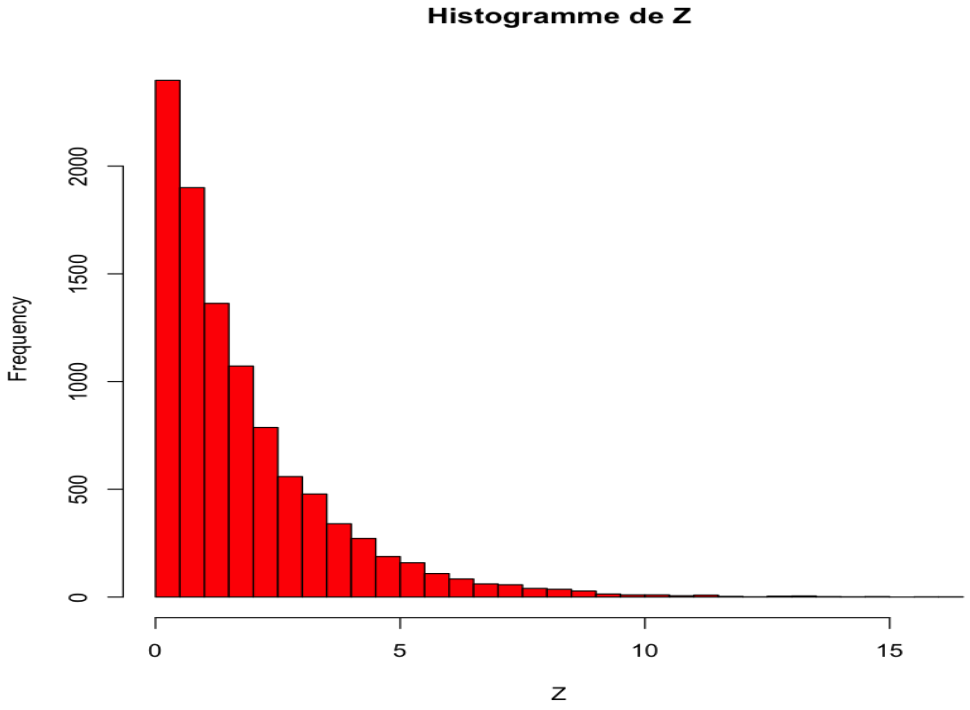
Tracés des densités empirique et théorique



On étudie à présent la variable $Z = X^2 + Y^2$ pour laquelle on trace la densité :

```
### Input :
Z=(sampler92X(0.3,Nsim))^2+(sampler92Y(0.3,Nsim))^2
hist(Z,col= "red",breaks = 50,main="Histogramme de Z")
```

Tracés de l'histogramme de la densité de Z



On cherche à présent à estimer $P(Z > 2)$

Nous pouvons estimer empiriquement cette probabilité en calculant le quotient $\frac{Card[Z[i] > 2]}{Card[Z[i]]}$. On moyenne sur 100 itérations pour obtenir :

```
### Input :
Az <-c()
for (k in 1:100){
Z=(sampler92X(0.3,Nsim))^2+(sampler92Y(0.3,Nsim))^2
Z2=Z[Z>2]
temp = length(Z2)/length(Z)
Az[k]=temp
}
mean(Az)
### Output :
0.332419
```

Estimation de P(Z>2)