Université Claude Bernard Lyon 1 (*ucbl*)
Villeurbanne
Rhone-Alpes-France

Mathématiques Appliquées

PROJET DE PROCESSUS STOCHASTIQUES

Pour obtenir le diplôme de Master 1 en :
Mathématiques Appliquées : statistique

Auteur :
KALY Bauvary

Simulation des mouvements browniens

Projet encadré par : PERRUT Anne

Table des matières

0.1 Simulation d'une échantillon de loi normale

Ici nous présentons deux méthodes de simulation d'une échantillon de loi normale , dont une naturelle et l'autre utilisant la méthode de décomposition de Cholesky. Ci-dessous, le code permettant les deux simulations ainsi que la représentation des nuages de points correspondant.

```
n=1000
M=(c(-1,5))
G=matrix(c(4,2,2,3),nrow=2)
Z=matrix(rnorm(2*n),ncol=2)
L=t(chol(G)) #ici, L est la transposée
                                    #de la décomposition de Cholesky de G
X=t(M+L%*%t(z))
plot(X,cex=0.2,main="Représentation du nuage de point de X")
plot(Z,cex=0.2,main="Représentation du  na de point de Z")
```

Ce code fait une simulation de taille mille, de loi normale $\mathcal{N}(M, G)$, avec :

$$M = \begin{pmatrix} -1 \\ 5 \end{pmatrix} \qquad et \qquad G = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix};$$

dont la représentation des nuages de points est ci-dessous.

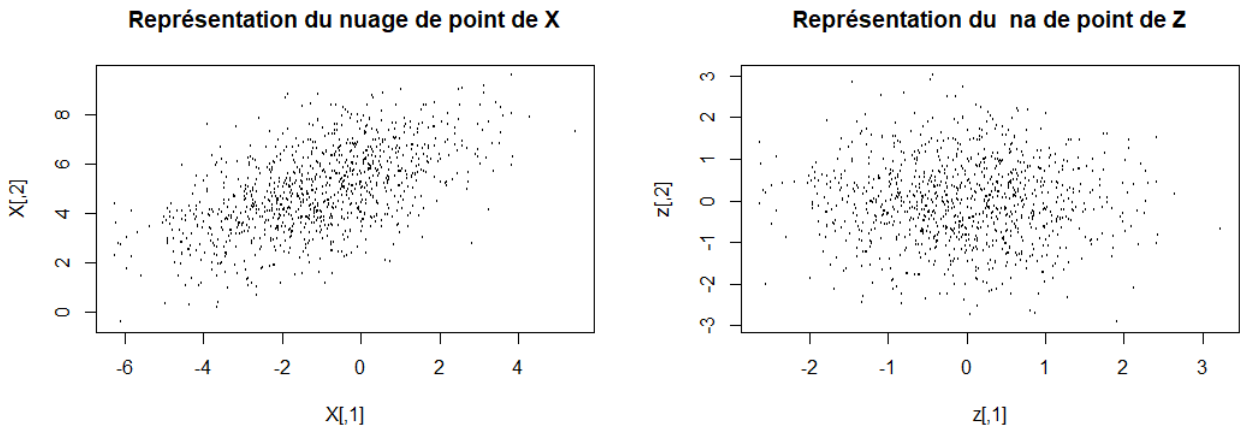

FIGURE 1 – Représentation des nuages de points des échantillons X et Z .
Le nuage de points de l'échantillon X (simulation utilisant la méthode de Cholesky) a une forme d'ellipse, témoin d'une grande variance de X, et dont de la dépendance des données.
Par contre l'échantillon Z, a des nuage de points sous forme de cercle, témoin d'une variance plus petite que celle de X, et donc de l'indépendance des variables.

0.2 Simulation d'un mouvement brownien

Ici nous allons présenter discuter sur l'efficacité de deux méthodes de simulation de mouvement brownien ; la méthode directe et celle itérative.

0.2.1 Simulation directe

Ci-dessous le code permettant la simulation directe.

```
t=5
n =500
instants = seq(t/n,t,t/n) # Les instant du mouvement brownien
sigma = matrix(t/n,ncol = n,nrow = n)
for (i in 2:n-1) {
        sigma[i,1:i]=instants[1:i]
        sigma[i,(i+1):n]=instants[i]
}
sigma[n,]=instants
Z=rnorm(n)
MB=t(chol(sigma))%*%Z
plot(instants,MB,type = 'l',main = 'Graphe du mouvement pour t')

# Autre méthode
# Elle permet permet d'éviter la boucle

t=5
n=500
instantsbiss=seq(t/n,t,by=t/n)
A=matrix(instantsbiss,ncol = n,nrow = n)
Sigma=pmin(A,t(A)) # pour les valeurs min et max
Z=rnorm(n)
MB=t(chol(Sigma))%*%Z
plot(instantsbiss, MB,type='l',main='Graphe du mouvement pour t.')
```

Ci-dessous nous avons la représentation du mouvement créé par le code ci-dessus.

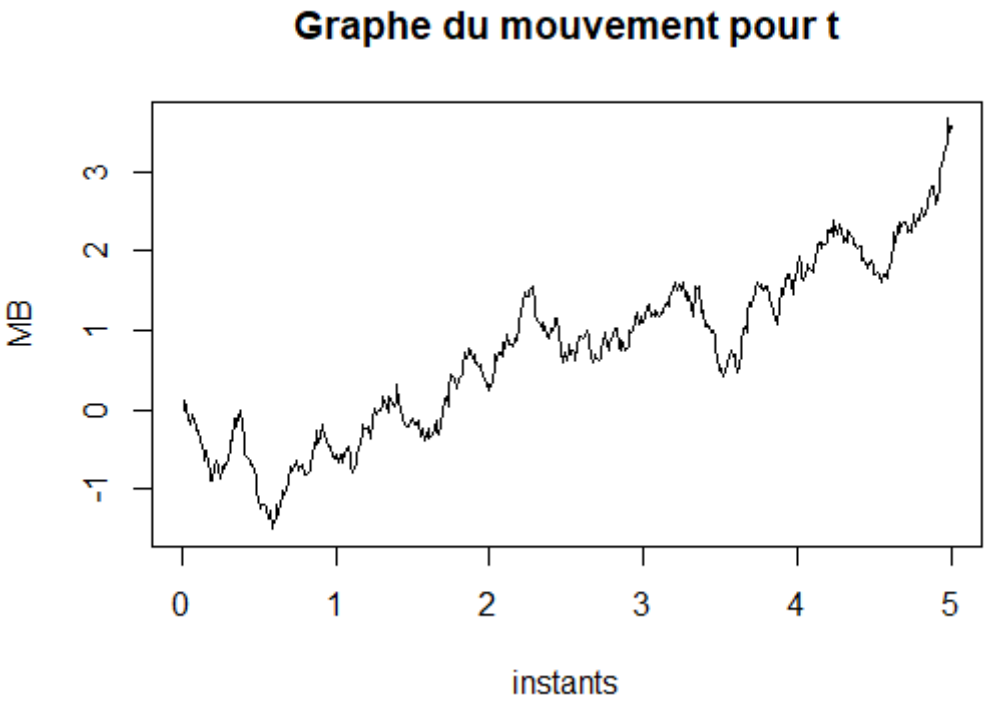

FIGURE 2 – Représentation du trajectoire du mouvement .

0.2.2 Simulation par la méthode itérative

Cette méthode construit le mouvement en utilisant son passé, d'où son "itérative". Ci-dessous le code R pour la simulation.

```
t=5
n=1000
instants=seq(t/n,t,by=t/n)
MB = rep(NA,length(instants))
MB = rnorm(1,0,instants[1])
for (i in 2:length(instants)) {
        MB = c(MB, MB[length(MB)] + rnorm(1,0,instants[i]-instants[i-1]))
}
plot(instants,MB,type='l',main='Graphe du mouvement brownien itérative.')
```

Ci-dessous une représentation de ce mouvement.

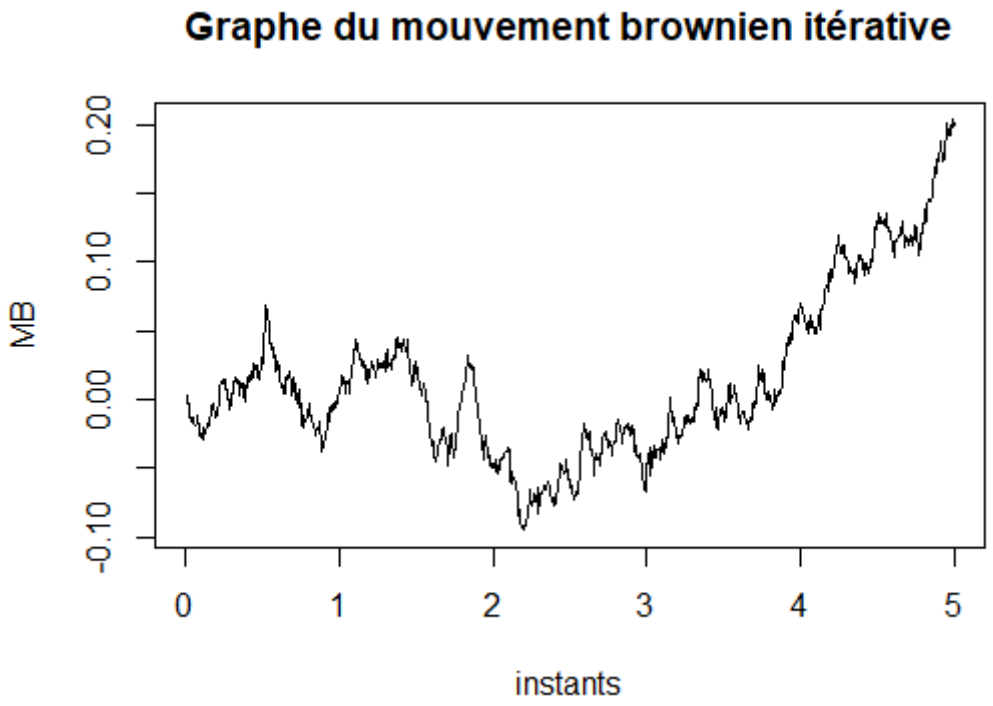

FIGURE 3 – Représentation du trajectoire du mouvement .

0.2.3 Étude de l'efficacité des méthodes

Pour l'étude de l'efficacité des méthodes on va comparer leur temps d'exécution.
Pour ce faire, nous présenter ci-dessous les programmes R permettant d'obtenir les temps d'exécution pour chaque méthode.

```
# Pour la méthode directe

tps1 = proc.time()
t=5
```

```
n=5000
instantsbiss=seq(t/n,t,by=t/n)
A=matrix(instantsbiss,ncol = n,nrow = n)
Sigma=pmin(A,t(A))
Z=rnorm(n)
MB=t(chol(Sigma))%*%Z
plot(instantsbiss,MB,cex=0.2, main = 'Graphe du mouvement pour t')
tps1 = proc.time() - tps1

# Pour la méthode itérative

tps2 = proc.time()
t=5
n=5000
instants=seq(t/n,t,by=t/n)
MB = rep(NA,length(instants))
MB = rnorm(1,0,instants[1])
for (i in 2:length(instants)) {
        MB = c(MB, MB[length(MB)] + rnorm(1,0,instants[i]-instants[i-1]))
}
plot(instants, MB, type = 'l')
tps2 = proc.time() - tps2

tps1
tps2
```

Voici le tableau récapitulatif des temps.

Type / *Temps*	utilisateur	systeme	écoulé
méthode directe	26.53	0.88	31.08
méthode itérative	0.06	0.42	0.56

TABLE 1 – Tableau récapitulatif de temps d'exécution des programmes.
Ici il est claire que la méthode itérative (voir 0.2.2) est la plus rapide que la méthode directe (voir 0.2.1).
La lenteur de la méthode directe est du à cause de la méthode Cholesky.

0.2.4 Complexité de la méthode itérative

Nous essayons dans cette section d'estimer la complexité de la méthode. Ci-dessous le code R pour le simulation de plusieurs mouvements avec la méthode directe, et ainsi permettant de représenter les temps d'exécution.

```
teps1=rep(NA,10)
for(i in 100*(1:10)){
  tc=proc.time();
  t=5;
  tps=seq(t/(i),t,by=t/(i));
```

```
  A=matrix(tps,ncol=i,nrow=i);
  Sigma=pmin(A,t(A));
  Z=rnorm(i);

  MB=t(chol(Sigma)%*%Z)
  teps1[i/100]=proc.time()-tc;
}
temps=log(teps1)
plot(100*(1:10),temps, type="l",main="Graphe de la complexité")
abline(-6.3,0.005,col='red') ## Ajout de la droite linaire y=0.005x - 6.3
```

Voici le graphe représentant les temps en fonction de n.

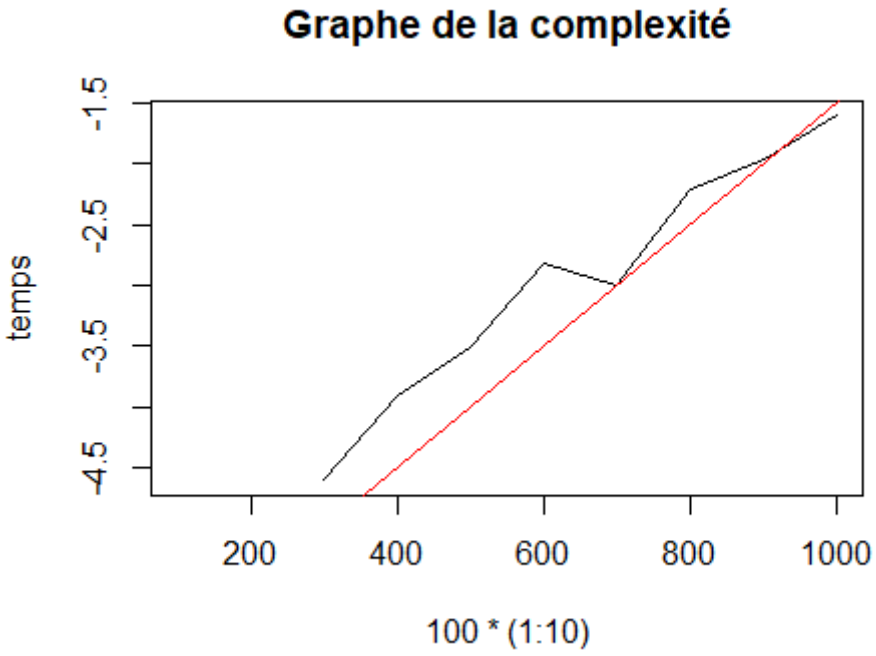

FIGURE 4 – Représentation du trajectoire du mouvement .

0.3 Excursion brownienne

Dans cette section, il est question de simuler un mouvement brownien en le conditionnant d'être nul en $t_0 = 0$ et en $t_f = 1$. Cette simulation va se faire en utilisant la méthode récursive. Le code R permettant cette simulation est mentionné ci-dessous :

```
n = 5
## initialisation du mouvement brownien respectant les conditions
B = c(0,0)
for (k in 1:n) {
  Bold = B
  Bmoy = 0.5*(B[1:2^(k-1)] + B[2:(2^(k-1)+1)])
  Bsim = Bmoy + rnorm(2^(k-1))/(2*sqrt(2^(k-1)))
  Bnew = rep(0,2^k+1)
```

```
  Bnew[seq(1,2^k+1,by=2)] = Bold
  Bnew[seq(2,2^k+1,by=2)] = Bsim
  B=Bnew
  print(B)
}

plot(B, type= 'l', main="Excursion browniennne")
## Bold: représente le passé du mouvement.
## Bmoy: représente la moyenne des élément successifs de B.
## Bsim: représente la partie de la récursivité dans le code.
## Bnew: est utilisé pour récupérer le passé et la simulation;
##              pour reconstituer un nouveau mouvement.
## D'où son non: méthode récursive.
```

Ci-dessous, la figure donnée par le plot du programme ci-dessus :

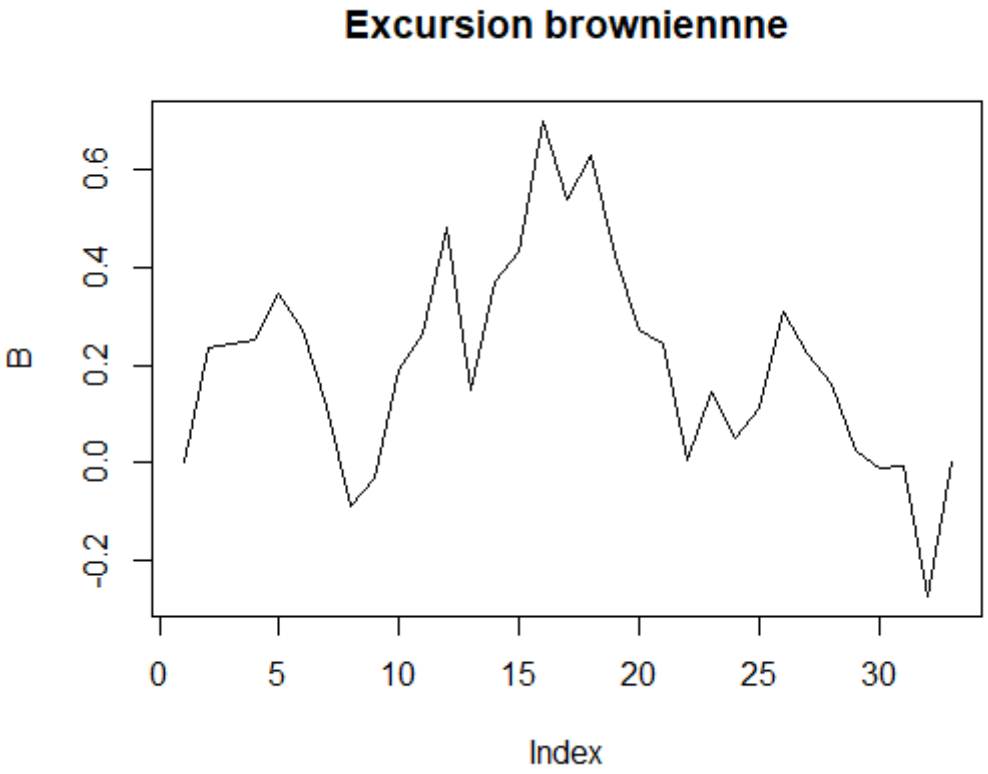

FIGURE 5 – Représentation du trajectoire du mouvement.
On observe bien que le mouvement débute par 0 et finit par 0 respectivement en $t_0 = 0$ et en $t_f = 1$

0.4 Probabilité de passé par 0 entre s et t

Ici il s'agit d'estimer la probabilité qu'un mouvement passe par 0 entre s et t.
Pour cela, on utilise le fait que le mouvement passe par 0 si le produit entre deux instants

successifs est négative. le code R que l'on propose ci-dessous le fera pour tous les point de l'espace temps.

```
t=2
s=1
n=1000
instants=seq(s,t,l=n)
compteur=0
for (k in 1:n) {
Bs = rnorm(1,0,sqrt(s))
N=1000
B = Bs + cumsum(rnorm(n,0,sqrt((t-s)/n)))
compteur = compteur + (sum(B[1:(n-1)]*B[2:(n)]<0)>0)
}
plot(instants,B,type = 'l', main="mouvement brownienne")
proba_0 =compteur/n

prop.test(compteur,n)$conf.int
```

On trouve une probabilité $P = 0.448$ et l'intervalle de confiance à 95% est $I_c = [0.4169408, 0.4794636]$. Ci-dessous le graphe représentatif dur mouvement.

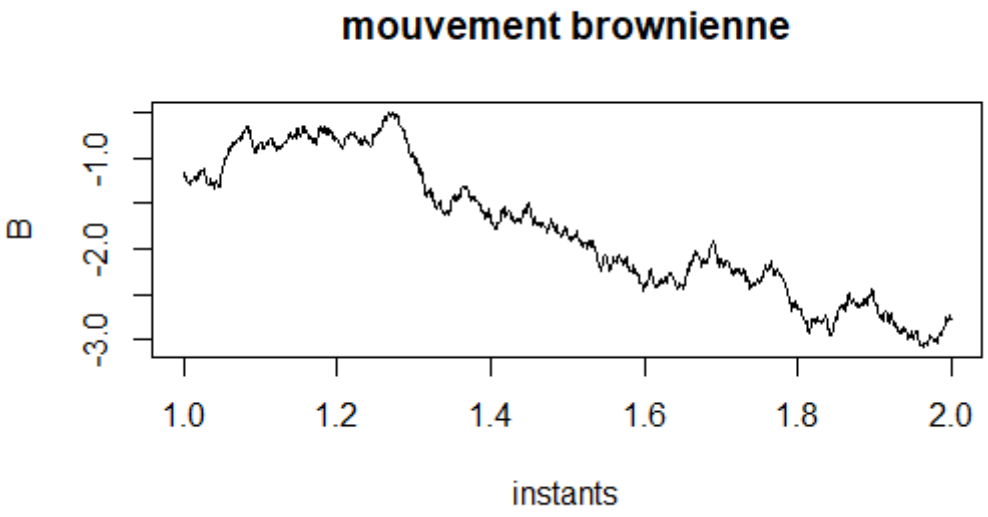

FIGURE 6 – Représentation du trajectoire du mouvement.
On observe bien que le mouvement débute par 0 et finit par 0 respectivement en $t_0 = 0$ et en $t_f = 1$