CR_B3324

Minh Phung & Hugo Grel

13/05/2022

Partie 1 : Tests de générateurs pseudo-aléatoires

1.1.4 : Génération de séquence avec 4 générateurs

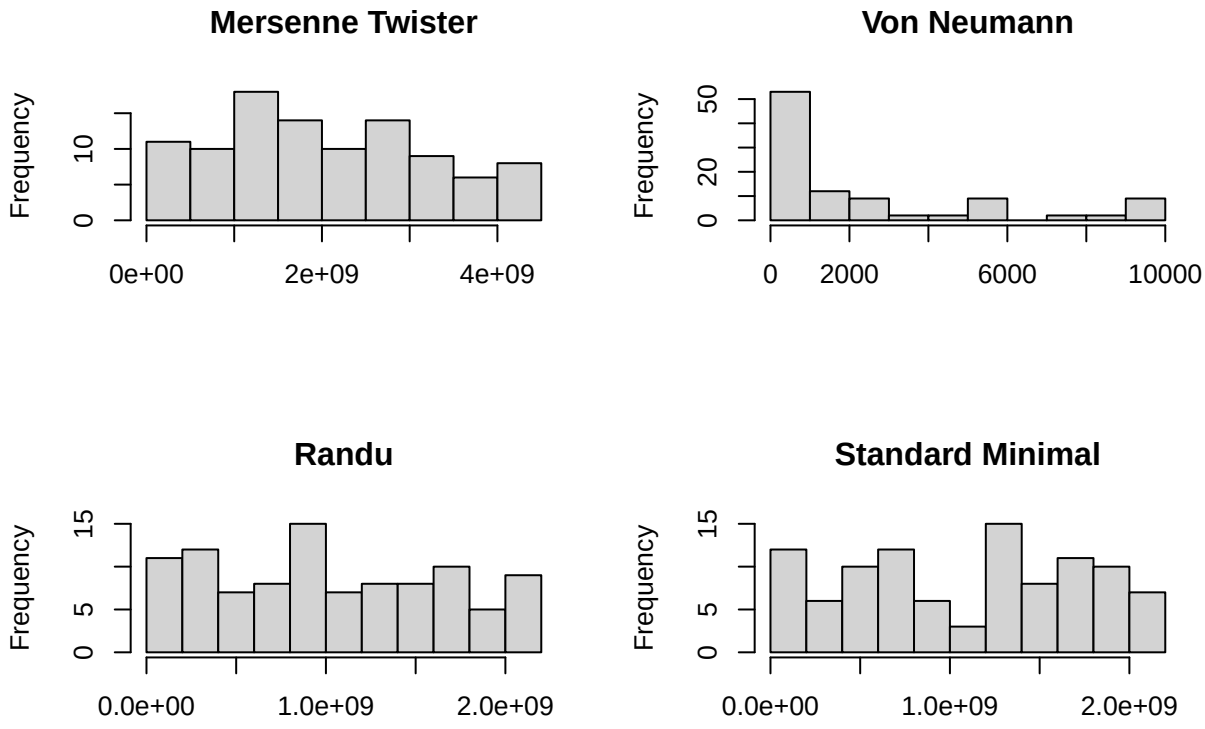
Question 1

```
library(randtoolbox)
```

```
## Loading required package: rngWELL
```

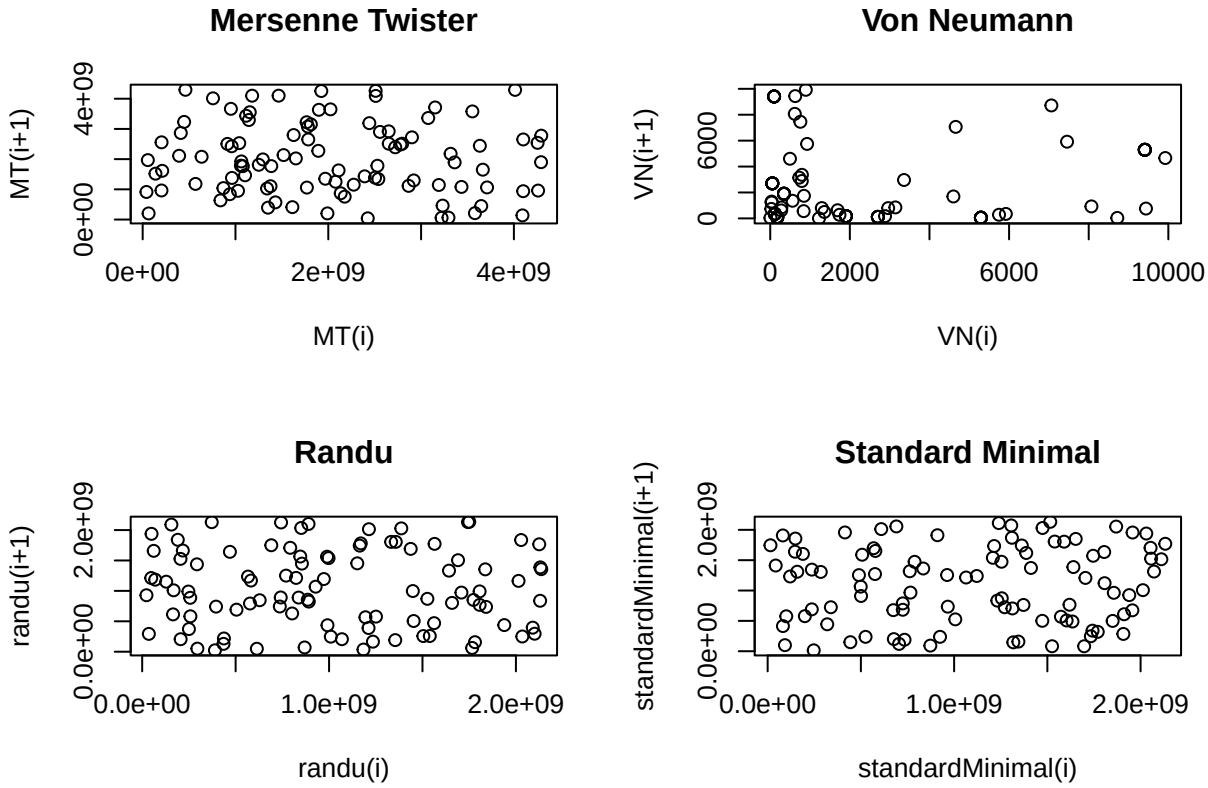
```
## This is randtoolbox. For an overview, type 'help("randtoolbox")'.
```

```
source('generateurs.R')

sVN <- 2567
sMT <- 2567
sRandu <- 2567
sSM <- 2567
Nsimu <- 100
Nrepet <- 100

vn <- VonNeumann(Nsimu,Nrepet,sVN)
mt <- MersenneTwister(Nsimu,Nrepet,sMT)
randu <- RANDU(100, 1, sRandu)
standardMinimal <- StandardMinimal(100, 1, sSM)

par(mfrow=c(2,2))
hist(mt[,1],xlab='',main='Mersenne Twister')
hist(vn[,1],xlab='',main='Von Neumann')
hist(randu[,1],xlab='',main='Randu')
hist(standardMinimal[,1],xlab='',main='Standard Minimal')
```



On constate que l'histogramme de Von Neumann présente des valeurs très ciblés et de ce fait pas suffisament aléatoires. Les trois autres histogrammes ont des valeurs mieux réparties sur la plage de valeurs.

```
par(mfrow=c(2,2))
plot(mt[1:(Nsimu-1),1],mt[2:Nsimu,1],xlab='MT(i)', ylab='MT(i+1)',
     main='Mersenne Twister')
plot(vn[1:(Nsimu-1),1],vn[2:Nsimu,1],xlab='VN(i)', ylab='VN(i+1)',
     main='Von Neumann')
plot(randu[1:(Nsimu-1),1],randu[2:Nsimu,1],xlab='randu(i)', ylab='randu(i+1)',
     main='Randu')
plot(standardMinimal[1:(Nsimu-1),1],
     standardMinimal[2:Nsimu,1],
     xlab='standardMinimal(i)', ylab='standardMinimal(i+1)',
     main='Standard Minimal')
```

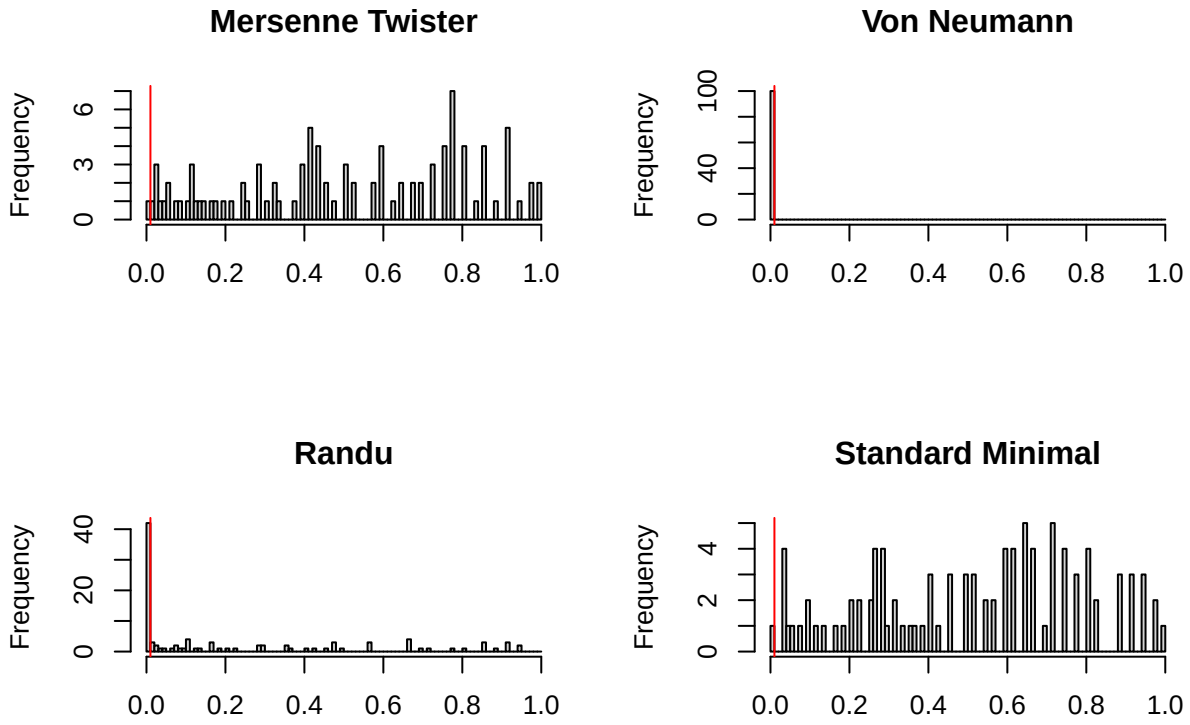


Encore une fois, le graphique concernant Von Neumann n'est pas concluant. Graphiquement, nous ne pouvons pas déterminé, pour les 3 autres, quel générateur est le meilleur.

1.2.2 Test de fréquence monobit

```
source('generateurs.R')
set.seed(4879)
graine <- sample.int(2^31, 100)

freq_vn <- rep(0,100)
freq_mt <- rep(0,100)
freq_rd <- rep(0,100)
freq_sm <- rep(0,100)

for ( i in 1:100) {
  freq_vn[i] <- Frequency(VonNeumann(Nsimu,1,graine[i]), 14)
  freq_mt[i] <- Frequency(MersenneTwister(Nsimu,1,graine[i]), 32)
  freq_rd[i] <- Frequency(RANDU(100, 1, graine[i]), 31)
  freq_sm[i] <- Frequency(StandardMinimal(100, 1, graine[i]), 31)
}

par(mfrow=c(2,2))
hist(freq_mt,xlab='',main='Mersenne Twister', breaks = seq(0,1, 0.01))
abline(v=0.01, col = 'red')
```

```
hist(freq_vn,xlab='',main='Von Neumann',breaks = seq(0,1, 0.01))
abline(v=0.01, col = 'red')
hist(freq_rd,xlab='',main='Randu', breaks = seq(0,1, 0.01))
abline(v=0.01, col = 'red')
hist(freq_sm,xlab='',main='Standard Minimal', breaks = seq(0,1, 0.01))
abline(v=0.01, col = 'red')
```



Observations des résultats :

En faisant un histogramme des valeurs de Pvaleur (valeurs renvoyées par la fonction Frequency) avec un pas de 0.01, nous pouvons observer la répartition de ces valeurs d'un côté ou de l'autre du trait rouge.

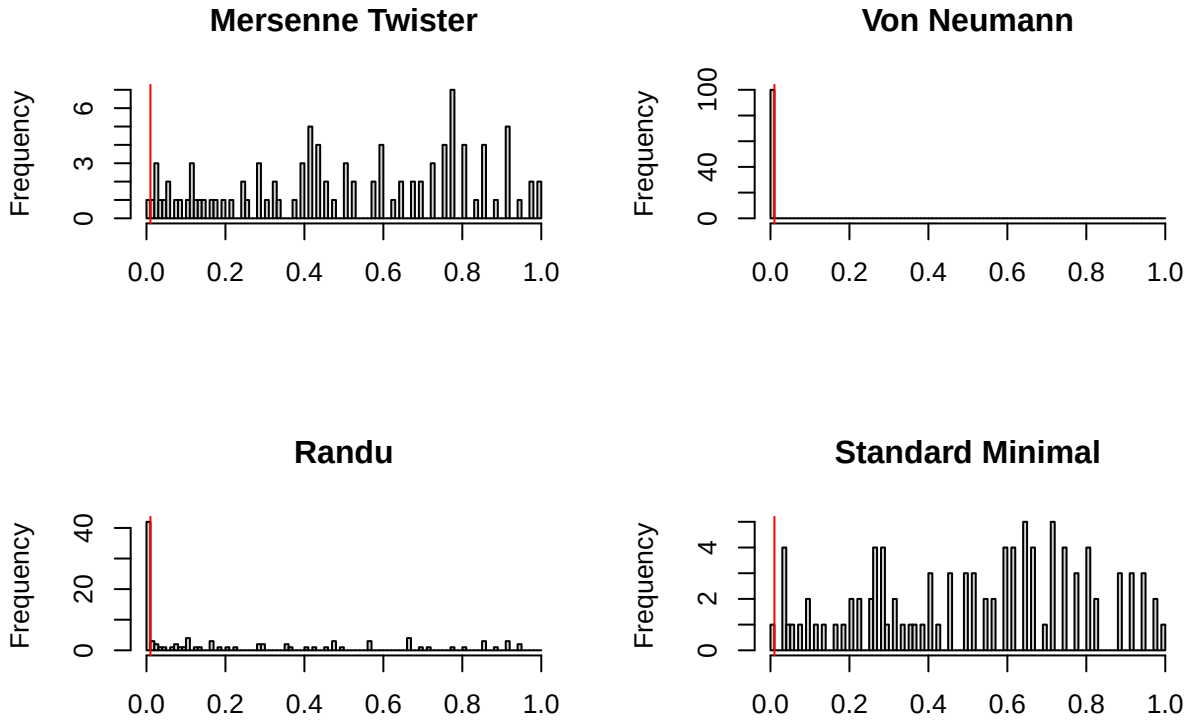
Pour 100 initialisations différentes : - Le générateur Von Neumann a 100 valeurs inférieurs à 0.01 ! - Le générateur RANDU a plus de 40 valeurs inférieurs à 0.01. - Les générateurs Mersenne Twister et Standard Minimal ont quand à eux, 1 valeur inférieur à 0.01.

- Générateurs à rejeter
  - Von Neumann
  - RANDU
- Générateurs passant la règle de décision à 1%
  - Mersenne Twister
  - Standard Minimal

Notons toutefois, qu'un générateur passant un test n'est pas un générateur valide ! C'est seulement avec un test échoué qu'on peut dire qu'un générateur n'est pas satisfaisant.

1.2.3 Test des runs

```
source('generateurs.R')

set.seed(4879)
graine <- sample.int(2^31, 100)

run_vn <- rep(0,100)
run_mt <- rep(0,100)
run_rd <- rep(0,100)
run_sm <- rep(0,100)

for ( i in 1:100) {
  run_vn[i] <- Runs(VonNeumann(Nsimu,1,graine[i]), 14)
  run_mt[i] <- Runs(MersenneTwister(Nsimu,1,graine[i]), 32)
  run_rd[i] <- Runs(RANDU(100, 1, graine[i]), 31)
  run_sm[i] <- Runs(StandardMinimal(100, 1, graine[i]), 31)
}

par(mfrow=c(2,2))
hist(freq_mt,xlab='',main='Mersenne Twister', breaks = seq(0,1, 0.01))
abline(v=0.01, col = 'red')
hist(freq_vn,xlab='',main='Von Neumann',breaks = seq(0,1, 0.01))
abline(v=0.01, col = 'red')
hist(freq_rd,xlab='',main='Randu', breaks = seq(0,1, 0.01))
abline(v=0.01, col = 'red')
hist(freq_sm,xlab='',main='Standard Minimal', breaks = seq(0,1, 0.01))
abline(v=0.01, col = 'red')
```



Remarque : Avec la règle de décision à 1%, ici le trait vertical rouge, on peut exclure ceux dont la majorité des tests sont en dessous de ce trait. Les résultats sont cohérents avec le test de fréquence monobit car on exclut ici aussi les générateurs Von Neumann et RANDU.

1.2.4 Test d'ordre

```
source('generateurs.R')

set.seed(4879)
graine <- sample.int(2^31, 100)

vn <- rep(0,100)
mt <- rep(0,100)
rd <- rep(0,100)
sm <- rep(0,100)

order_vn <- rep(0,100)
order_mt <- rep(0,100)
order_rd <- rep(0,100)
order_sm <- rep(0,100)

for ( i in 1:100) {
  vn <- VonNeumann(Nsimu,1,graine[i])
  vn = c(vn)
```

```
  order_vn[i] <- order.test(vn, d=4, echo=FALSE)$p.val

  mt <- MersenneTwister(Nsimu,1,graine[i])
  mt = c(mt)
  order_mt[i] <- order.test(mt, d=4, echo=FALSE)$p.val

  rd <- RANDU(100, 1, graine[i])
  rd = c(rd)
  order_rd[i] <- order.test(rd, d=4, echo=FALSE)$p.val

  sm <- StandardMinimal(100, 1, graine[i])
  sm = c(sm)
  order_sm[i] <- order.test(sm, d=4, echo=FALSE)$p.val
}

cptVn <- 0
cptSm <- 0
cptMt <- 0
cptRd <- 0

for (i in 1:100) {
  if(order_vn[i] < 0.01){
    cptVn <- cptVn + 1
  }
  if(order_mt[i] < 0.01){
    cptMt <- cptMt + 1
  }
  if(order_sm[i] < 0.01){
    cptSm <- cptSm + 1
  }
  if(order_rd[i] < 0.01){
    cptRd <- cptRd + 1
  }
}
```

Tableau des résultats :

Générateur	Nombre d'occurences de P-valeur $< 1\%$
Von Neumann	5
MersenneTwister	0
RANDU	0
Standard Minimal	1

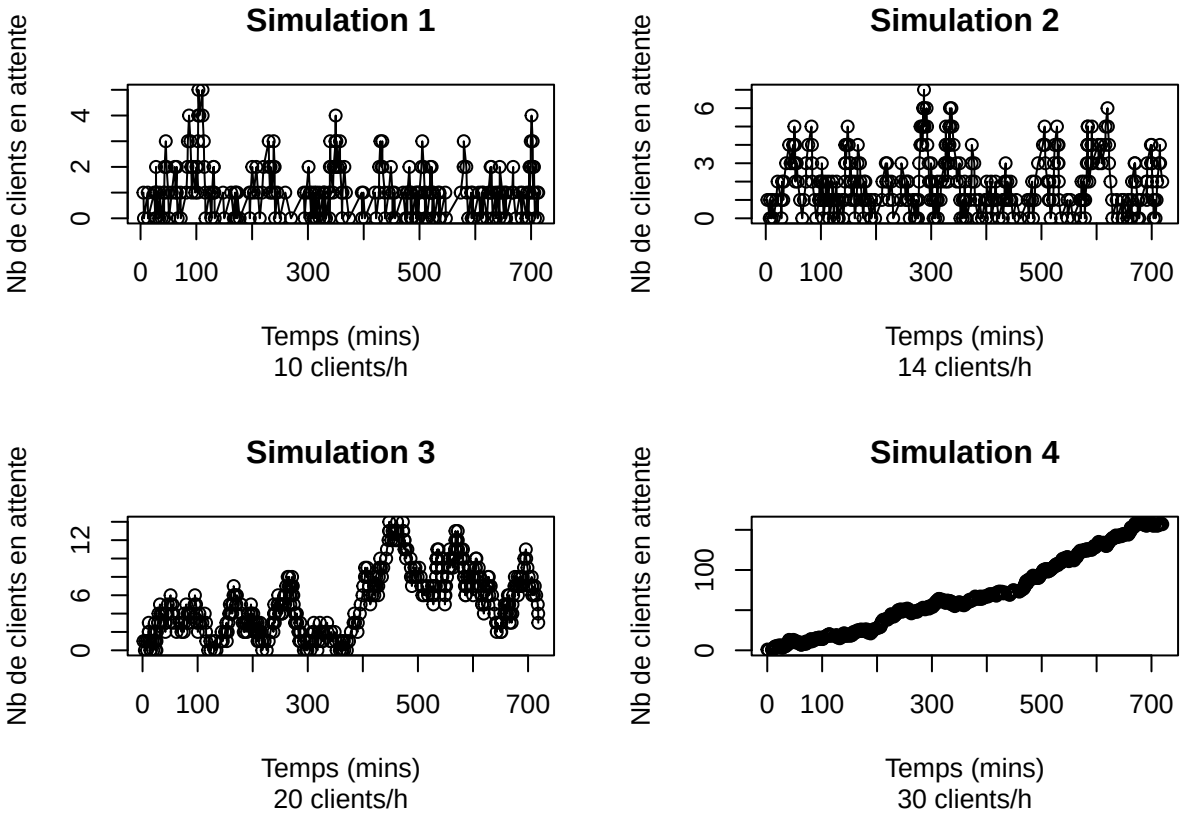
Remarque : D'après les résultats, le générateur de Von Neumann ne passe pas ce test. En effet, sur 100 répétitions, le nombre d'occurences de Pvaleur inférieur à 1% est de 5. Concernant le générateur Standard Minimal, il n'y a que 1% des Pvaleurs qui sont au dessous des 1%. Ce résultat, sans être parfait, est acceptable et n'est pas suffisant pour rejeter ce générateur.

2 Application aux files d'attentes

2.1 Files M/M/1

Question 6-7

```
source('files_dattenteMM1.R')
l1 <- 10/60
l2 <- 14/60
l3 <- 20/60
l4 <- 30/60
mu <- 20/60
a_10_20 <- FileMM1(l1, mu, 12*60)
a_14_20 <- FileMM1(l2, mu, 12*60)
a_20_20 <- FileMM1(l3, mu, 12*60)
a_30_20 <- FileMM1(l4, mu, 12*60)
par(mfrow=c(2,2))
e1 <- EvolutionMM1(a_10_20[[1]], a_10_20[[2]])
e2 <- EvolutionMM1(a_14_20[[1]], a_14_20[[2]])
e3 <- EvolutionMM1(a_20_20[[1]], a_20_20[[2]])
e4 <- EvolutionMM1(a_30_20[[1]], a_30_20[[2]])
plot(e1[,1], e1[,2], xlab="Temps (mins)", ylab = "Nb de clients en attente",
     main="Simulation 1", sub="10 clients/h", type = "o")
plot(e2[,1], e2[,2], xlab="Temps (mins)", ylab = "Nb de clients en attente",
     main="Simulation 2", sub="14 clients/h", type = "o")
plot(e3[,1], e3[,2], xlab="Temps (mins)", ylab = "Nb de clients en attente",
     main="Simulation 3", sub="20 clients/h", type = "o")
plot(e4[,1], e4[,2], xlab="Temps (mins)", ylab = "Nb de clients en attente",
     main="Simulation 4", sub="30 clients/h", type = "o")
```



Avec 10, 14 et 20 clients par heure, le nombre de client en attente dans le système fluctue dans une zone comprise entre 0 et 8 personnes pour les deux premiers cas. La simulation avec 20 clients par heure montre une évolution plus importante mais reste soutenable par le serveur. Cependant, avec 30 clients par heure, on observe une explosion dans le nombre de client en attente. Quel que soit la durée, le serveur ne peut pas suivre le rythme d'arrivée.

En utilisant le rapport $\alpha = \lambda/\mu$, on peut confirmer qu'avec un $\alpha < 1$ (simulation 1 et 2), le système va se stabiliser. Quand $\alpha = 1$ (simulation 3), dans notre cas, le système s'est stabilisé, mais on ne peut pas généraliser ce résultat. Enfin, avec $\alpha > 1$ (simulation 4), on observe bien que la file d'attente s'allonge et sature.

Question 8

```
source("files_dattenteMM1.R")
moy_nb1 <- mean(e1[,2])
moy_nb2 <- mean(e2[,2])
moy_nb3 <- mean(e3[,2])
moy_nb4 <- mean(e4[,2])
m1 <- mean(TempDAttente(a_10_20[[1]], a_10_20[[2]]))
m2 <- mean(TempDAttente(a_14_20[[1]], a_14_20[[2]]))
m3 <- mean(TempDAttente(a_20_20[[1]], a_20_20[[2]]))
m4 <- mean(TempDAttente(a_30_20[[1]], a_30_20[[2]]))
```

Observations :

Nombre de client par heure	Moyenne nombre de client dans le système	Moyenne temps de présence
10	1.14	4.07
14	2.02	6.4
20	5.36	15.97
30	71.41	123.29

Vérification de la formule de Little :

E(N)	$\lambda * E(W)$
1.14	0.68
2.02	1.49
5.36	5.32
71.41	61.64

On peut voir que nous retrouvons bien la formule de Little avec plus ou moins de précision tout de même sur certaines valeurs.