TP de Probabilités

Chanèle Jourdan, Corentin Laharotte, Jorge Terreu

12/05/2019

Ce Tp a pour but de créer des générateurs pseudo-aléatoires et de leur faire passer des tests statistiques afin de comparer la qualité des séquences produites.

Partie 1 : Test de générateurs pseudo-aléatoires

Question 1:

Les générateurs à congruence linéaire consiste à appliquer une transformation linéaire suivie d'une opération de congruence : $S = U = \{0, \ldots, m-1\}, S_n = f(S_{n-1}) = a * S_{n-1} + b \mod m, X_n = g(S_n) = S_n$

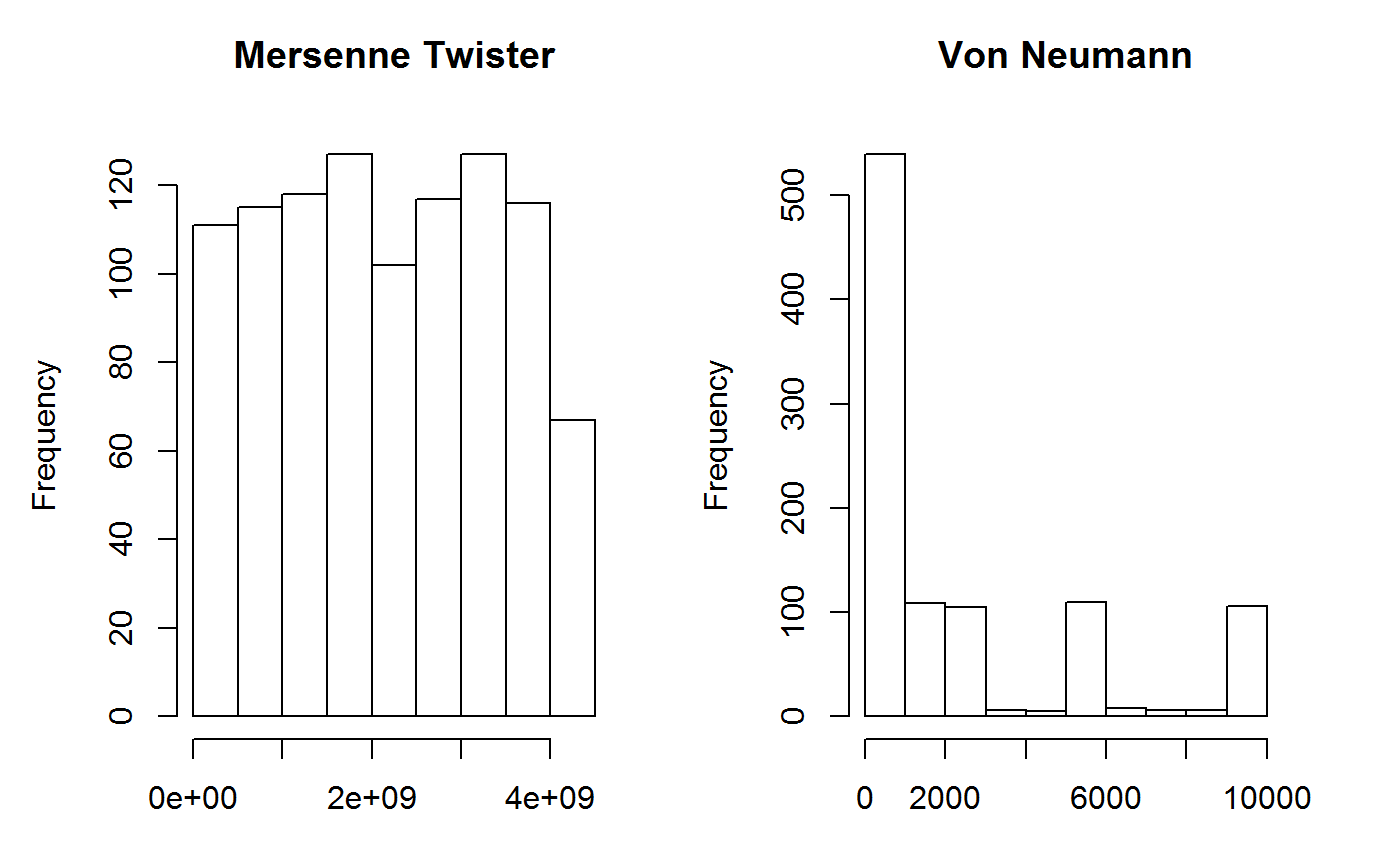
Nous avons implémenté 2 générateurs à congruence linéaire:

- Un generateur à congruence linéaire RANDU : a=65539, b=0, $m = 2^{31}$
- Un generateur à congruence linéaire Standard Minimal : a=16807, b=0,$m = 2^{31} - 1$

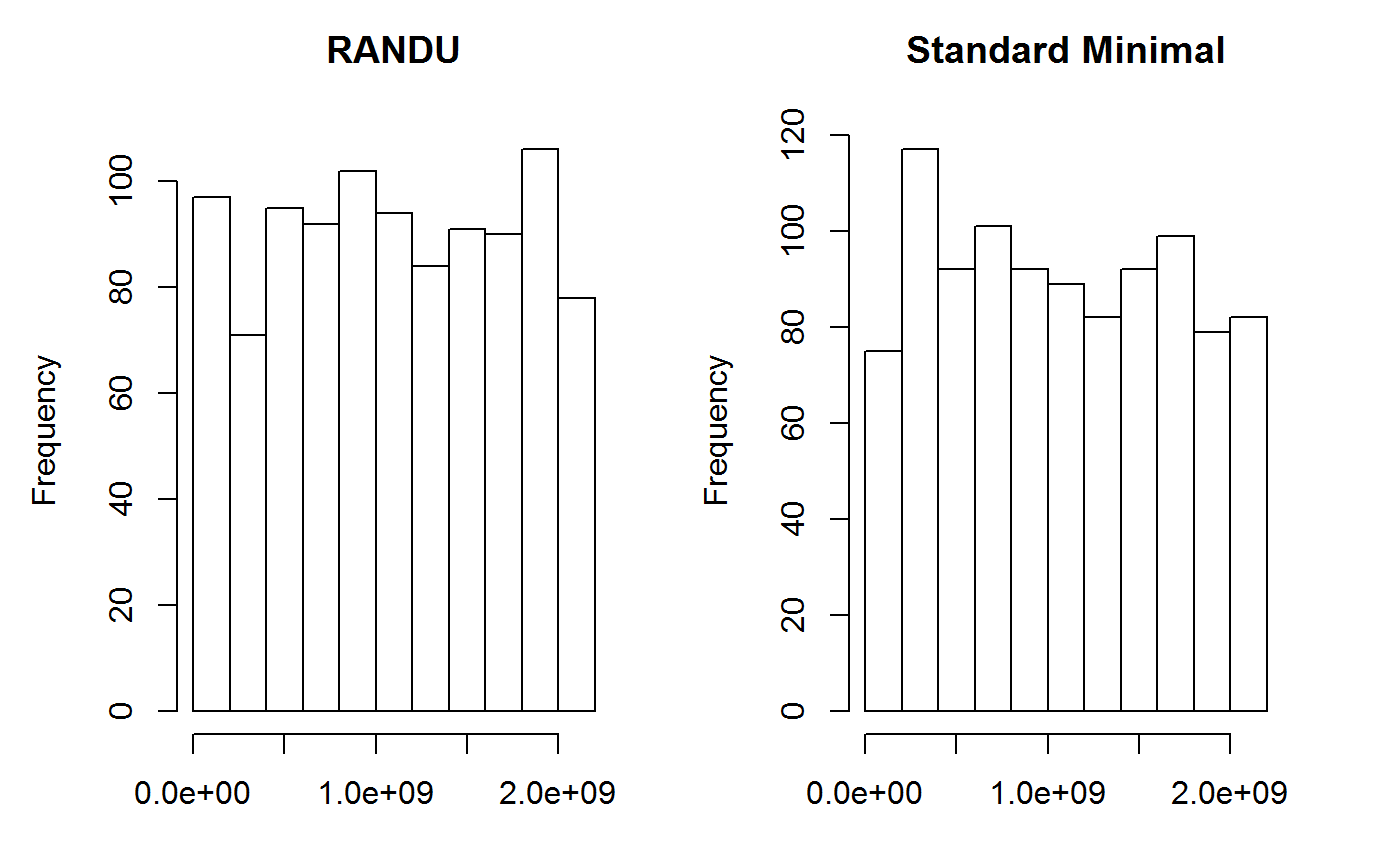
Question 2.1 Test Visuel:

Nous allons comparer ces générateurs pour une suite de 1000 valeurs. Pour cela nous générons une graine aléatoire avec un nombre compris entre 1000 et 1.

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




```
hist(rd[,1],xlab='',main='RANDU')
hist(sm[,1],xlab='',main='Standard Minimal')
```

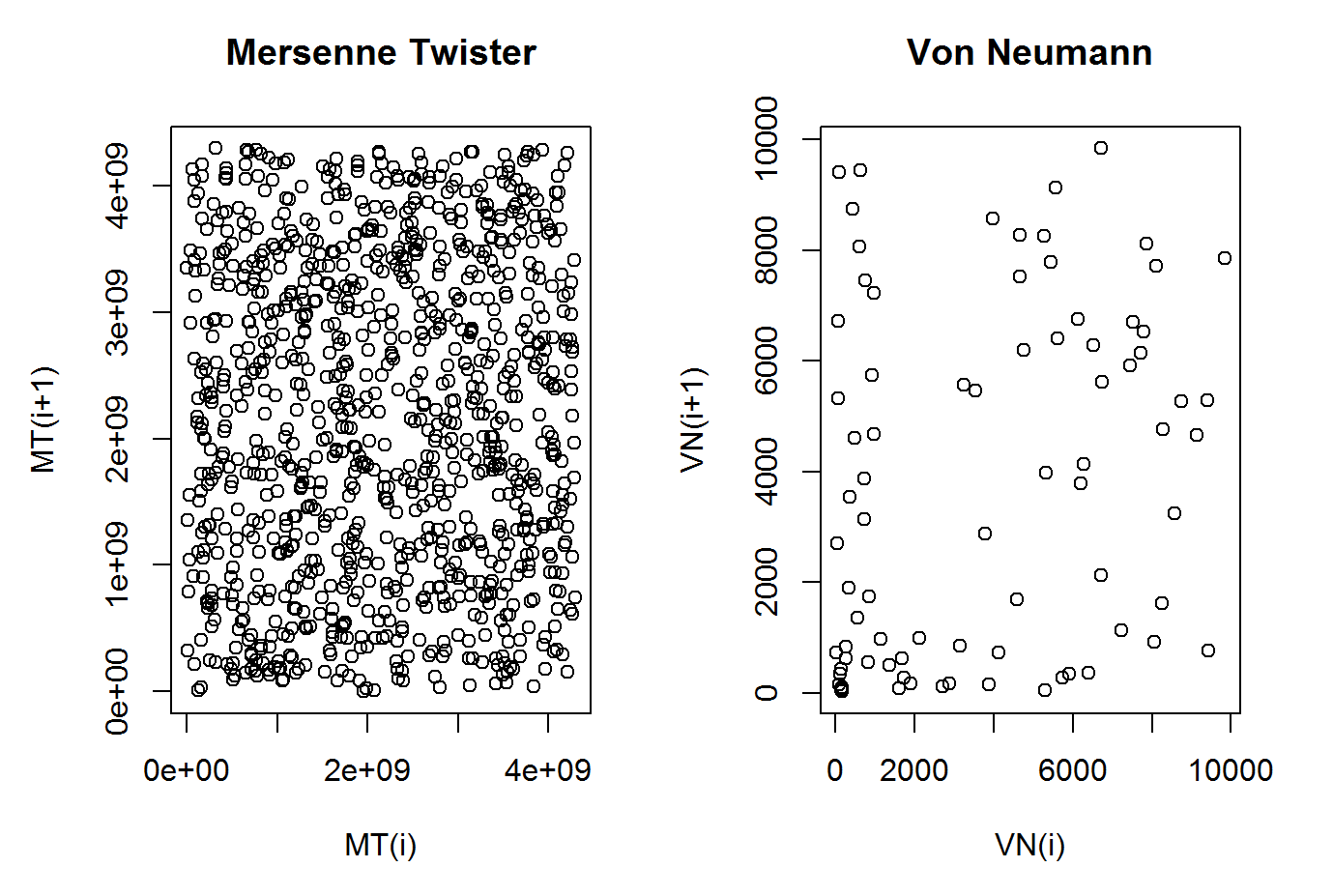


Nous pouvons remarquer que pour 1000 valeurs, les répartitions des résultats obtenus grâce aux générateurs Mersenne Twister, RANDU et Standard minimal sont globalement uniformes. Même si les histogrammes obtenus ne sont pas lisses, les répartitions sont visuellement assez satisfaisantes pour être considérées comme uniformes. En augmentant le nombre de valeurs générées, les histogrammes obtenus tendent à se lisser. Seul le générateur de Von Neumann n'est visuellement pas satisfaisant, en effet, la majorité des valeurs sont regroupées dans l'intervalle [0,2000]. La répartition ne semble donc pas uniforme.

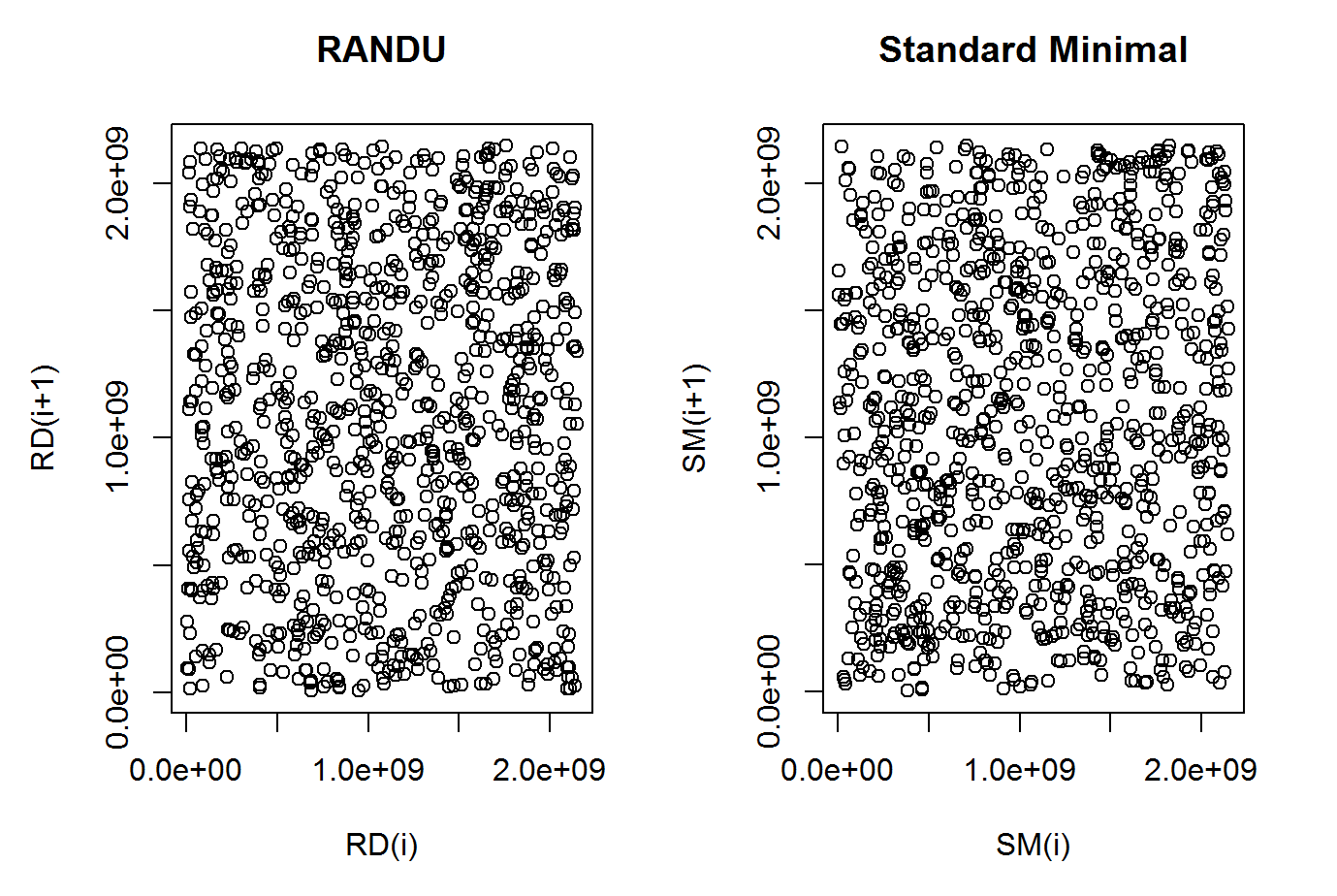
Question 2.2:

Nous pouvons nous intéresser à la répartition des valeurs obtenues en fonction de la valeur précédente:

```
par(mfrow=c(1,2))
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(rd[1:(Nsimu-1),1],rd[2:Nsimu,1],xlab='RD(i)', ylab='RD(i+1)', main='RANDU')
plot(sm[1:(Nsimu-1),1],sm[2:Nsimu,1],xlab='SM(i)', ylab='SM(i+1)', main='Standard Minimal')
```



Nous pouvons voir que, sauf pour le générateur de Von Neumann, les valeurs obtenues sont réparties assez uniformément, il n'y a pas de réel lien entre la valeur obtenue et la valeur précédente. On ne peut pas prédire la valeur obtenue à l'intant t en connaissant la valeur obtenue à l'instant $t-1$.

Question 3 Test de fréquence monobit:

On s'intéresse au nombre de 0 et de 1 dans les bits d'une séquence entière. On teste si les nombres de 1 et de 0 d'une séquence sont approximativement les mêmes, comme attendu dans une séquence vraiment aléatoire.

Pour cela, on attribut aux 1 de notre séquence de bits la valeur 1, et aux 0 la valeur -1. Puis nous sommons les valeurs des bits de notre séquence. Et nous divisons le tout par le nombre de bits à considérer. On obtient donc :$S_{obs} = \frac{|S_n|}{\sqrt(n)}$

Puis on regarde la valeur (appelée p_{valeur}) de la fonction de répartition de la loi $N(0,1)$ pour la valeur S_{obs} obtenue. Si p_{valeur} est petite, cela signifie qu'il est peu probable d'avoir obtenu la séquence générée, donc que la séquence n'est pas aléatoire.

les p_{valeur} des différents générateurs sont:

```
cat("Mersenne Twister | Von Neumann | RANDU | Standard Minimal\n",pvaleurmtF," | ",pvaleurvnF," | ",pvaleurrdF,
" | ",pvaleursmF)
```

```
## Mersenne Twister | Von Neumann | RANDU | Standard Minimal
##  0.8405061  |  0  |  1.866294e-09  |  0.9095609
```

Nous pouvons remarquer que les p_{valeur} des générateurs de Von Neumann et de RANDU sont inférieures à 0.01, donc les séquences obtenues ne sont pas aléatoires. Pour les générateurs Mersenne Twister, Standard Minimal, les p_{valeur} obtenues sont supérieures à 0.01, nous ne pouvons donc rien conclure sur l'aspect aléatoire de la séquence obtenue.

Question 4 Test des runs:

On s'intéresse maintenant à la longueur des suites successives de 0 et de 1 dans la séquence obtenue. Avec un raisonnement assez similaire à la question précédente, on essaye de savoir s'il est probable d'obtenir une séquence avec cette même longueur de suites successives de 0 et de 1. Plus la p_{valeur} obtenue est petite, moins il est probable d'obtenir la séquence observée, donc la séquence obtenue n'est pas aléatoire.

les p_{valeur} des différents générateurs sont:

```
cat("Mersenne Twister | Von Neumann | RANDU | Standard Minimal\n",pvaleurmtR," | ",pvaleurvnR," | ",pvaleurrdR,
" | ",pvaleursmR)
```

```
## Mersenne Twister | Von Neumann | RANDU | Standard Minimal
##  0.3343823  |  0  |  2.961478e-06  |  0.8290666
```

Nous pouvons remarquer que les p_{valeur} des générateurs de Von Neumann et de RANDU sont inférieures à 0.01, donc les séquences obtenues ne sont pas aléatoires. Pour les générateurs Mersenne Twister et Standard Minimal, les p_{valeur} obtenues sont supérieures à 0.01, nous ne pouvons donc rien conclure sur l'aspect aléatoire de la séquence obtenue.

Question 5 Test d'ordre:

On s'intéresse maintenant à la suite de nombres obtenus. On veut compter le nombre d'apparition d'un ordre de séquence donné (exemple : le nombre de fois où le premier nombre obtenu est inférieur au deuxième pour une séquence de 2 nombres (exemple simpliste)). Ce test d'ordre retourne une p_{valeur}.

les p_{valeur} des différents générateurs sont:

```
cat("Mersenne Twister | Von Neumann | RANDU | Standard Minimal\n",pvaleurmtO," | ",pvaleurvnO," | ",pvaleurrdO,
" | ",pvaleursmO)
```

```
## Mersenne Twister | Von Neumann | RANDU | Standard Minimal
##  0.91  |  2.5e-42  |  0.79  |  0.61
```

Nous pouvons remarquer que la p_{valeur} du générateur de Von Neumann est inférieure à 0.01, donc la séquence obtenue n'est pas aléatoire. Pour les générateurs Mersenne Twister, RANDU et Standard Minimal, les p_{valeur} obtenues sont supérieures à 0.01, nous ne pouvons donc rien conclure sur l'aspect aléatoire de la séquence obtenue.

Nous pouvons remarquer que, suivant le Test choisi, les séquences de nombres rejetées ne sont pas issues des mêmes générateurs. Il est donc nécessaire de faire plusieurs tests différents pour "s'assurer" de l'aspect aléatoire d'une séquence (en réalité on peut seulement s'assurer du non-rejet de la séquence, nous ne pouvons pas confirmer l'aspect aléatoire de la séquence).

Partie 2 : Application aux files d'attentes

Nous souhaiterions réaliser une file d'attente de Type FCFS(First Come First Served).

Files M/M/1

Pour un serveur particulier, il y a lambda arrivées et mu départs par unité de temps. Nous nous intéressons à l'état de cette file au bout d'un instant D.

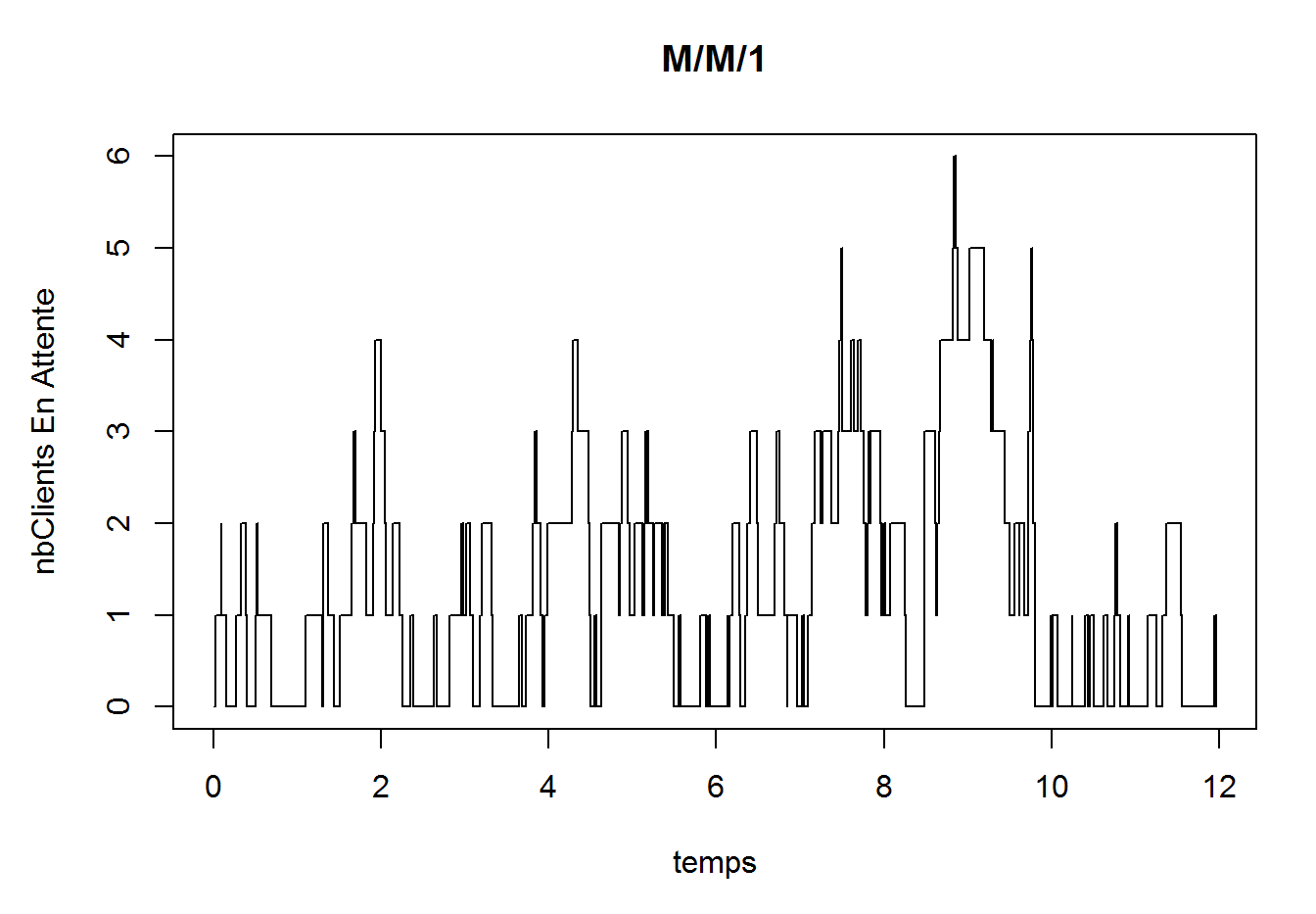
Question 6 et 7:

Nous aimerions connaître l'évolution de la file d'attente, c'est à dire, savoir combien il y a de clients dans la file d'attente à un instant donné.

Pour ce premier exemple nous fixons les paramètres:

- lambda=8
- mu=15
- D=12

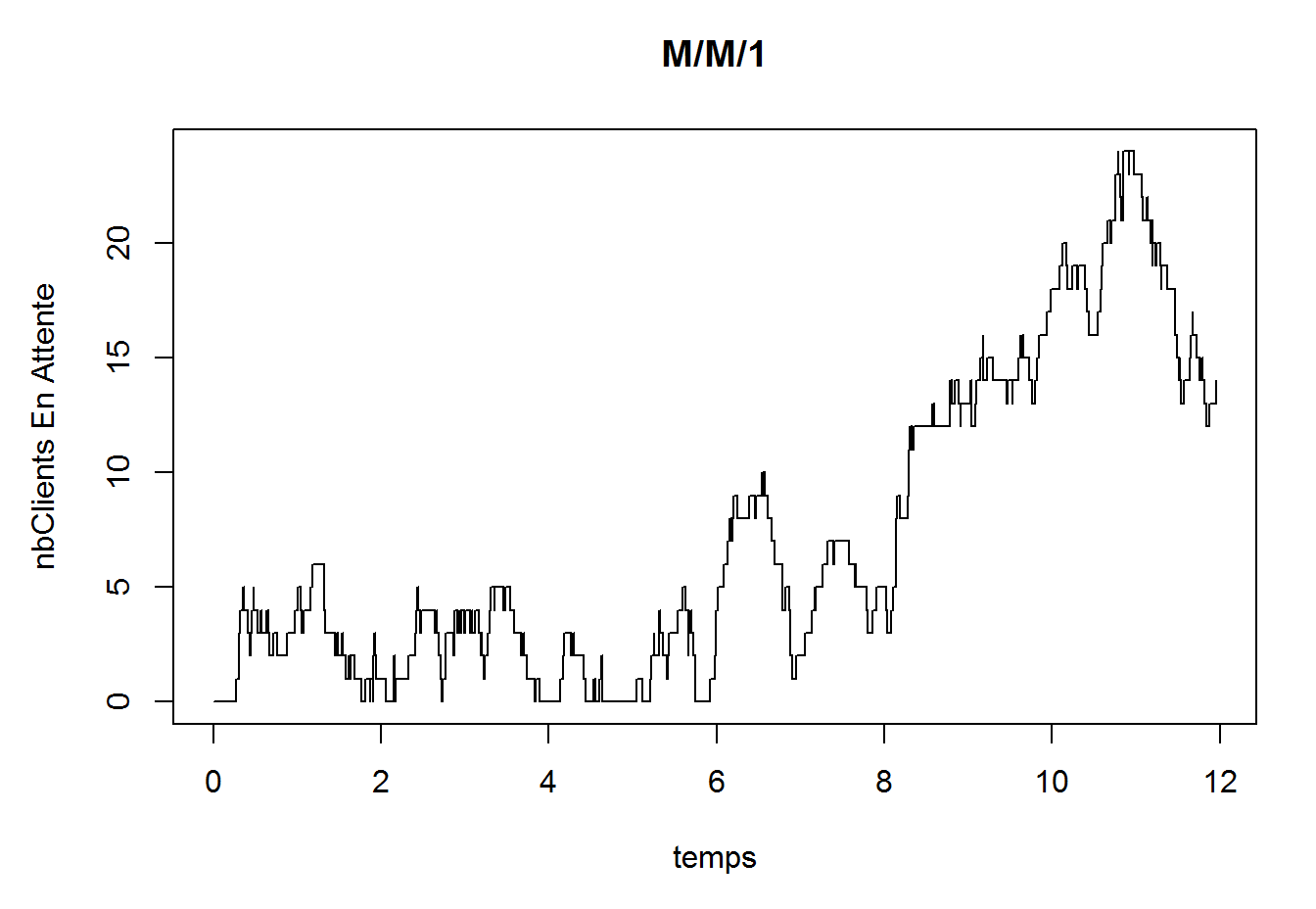
```
plot(unlist(temps),unlist(client),xlab='temps', ylab='nbClients En Attente', main='M/M/1',type='s')
```



Pour cet exemple nous fixons les paramètres:

- lambda=14
- mu=15
- D=12

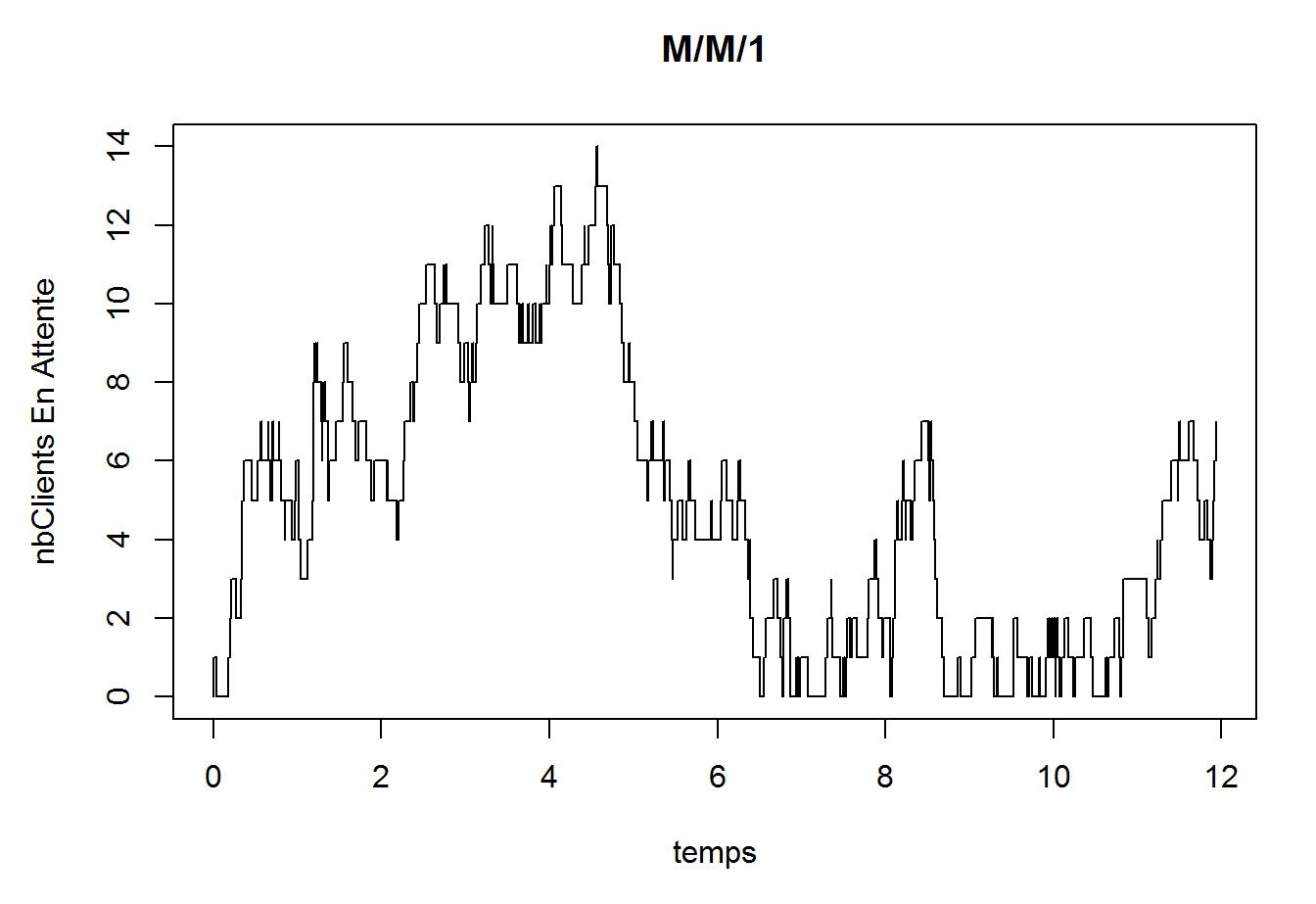
```
plot(unlist(temps),unlist(client),xlab='temps', ylab='nbClients En Attente', main='M/M/1',type='s')
```



Pour cet exemple nous fixons les paramètres:

- lambda=15
- mu=15
- D=12

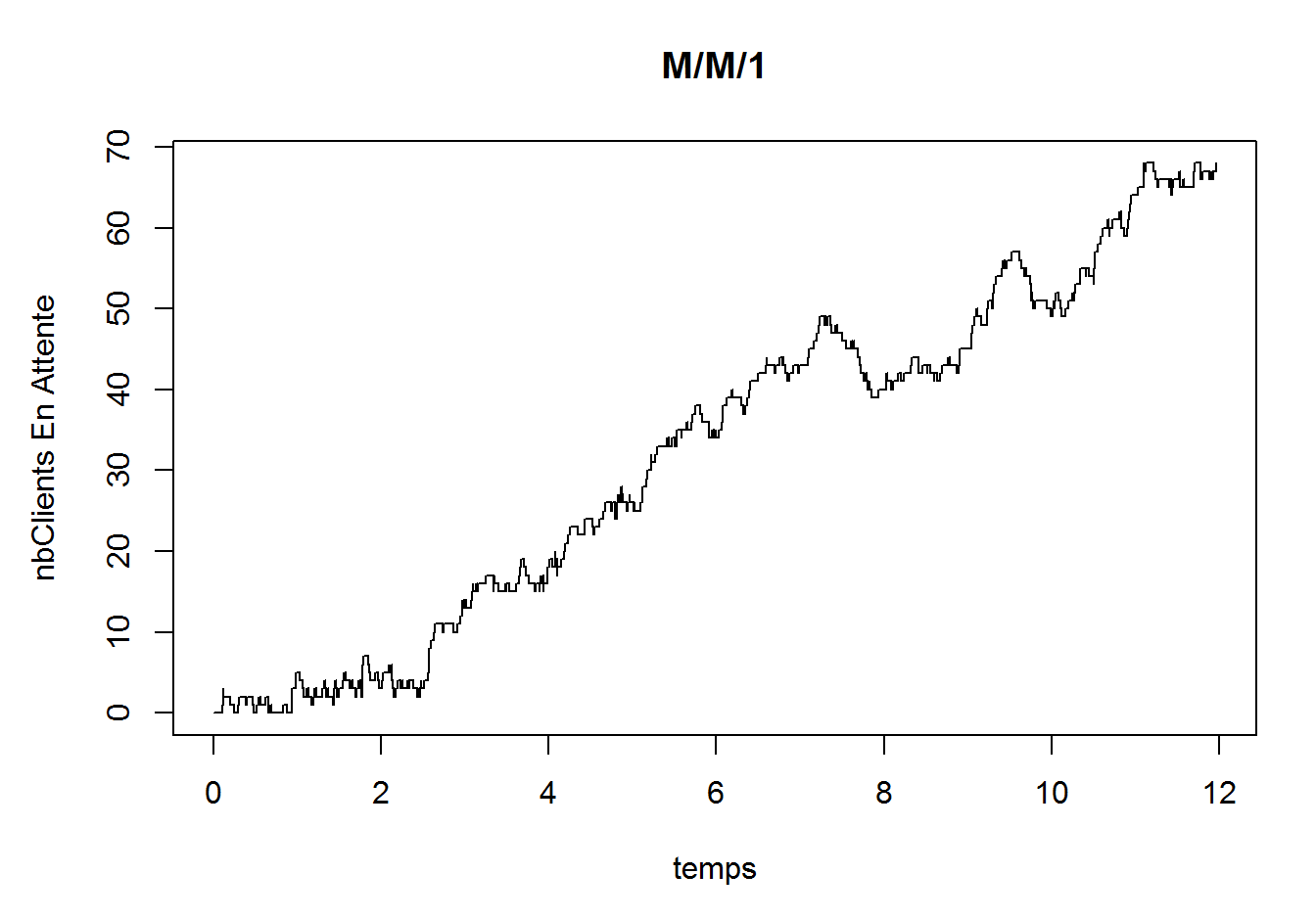
```
plot(unlist(temps),unlist(client),xlab='temps', ylab='nbClients En Attente', main='M/M/1',type='s')
```



Pour cet exemple nous fixons les paramètres:

- lambda=20
- mu=15
- D=12

```
plot(unlist(temps),unlist(client),xlab='temps', ylab='nbClients En Attente', main='M/M/1',type='s')
```



Nous pouvons remarquer que pour un même nombre de départ, plus le nombre d'arrivées par heures augmente, plus le réseau a tendance à se saturer. Lorque le rapport $\frac{lambda}{mu}$ est supérieur à 1, le réseau ne se stabile plus et sature.

Question 8:

On calcule désormais le nombre moyen de clients dans le système et le temps de présence d'un client dans le système après 12 heures de fonctionnement (les temps d'attente sont affichés en heures).

Pour ce premier exemple nous fixons les paramètres:

- lambda=8
- mu=15
- D=12

```
cat("attente moyenne :",attenteMoyenne)
```

```
## attente moyenne : 0.2
```

```
cat("nombre moyen de clients :",nbClientMoyen)
```

```
## nombre moyen de clients : 1.6
```

```
cat("nombre moyen de clients(calculé avec la forme de Little): ",formeLittle)
```

```
## nombre moyen de clients(calculé avec la forme de Little):  1.6
```

Pour cet exemple nous fixons les paramètres:

- lambda=14
- mu=15
- D=12

```
cat("attente moyenne :",attenteMoyenne)
```

```
## attente moyenne : 0.52
```

```
cat("nombre moyen de clients :",nbClientMoyen)
```

```
## nombre moyen de clients : 7.7
```

```
cat("nombre moyen de clients(calculé avec la forme de Little): ",formeLittle)
```

```
## nombre moyen de clients(calculé avec la forme de Little):  7.3
```

Pour cet exemple nous fixons les paramètres:

- lambda=15
- mu=15
- D=12

```
cat("attente moyenne :",attenteMoyenne)
```

```
## attente moyenne : 0.79
```

```
cat("nombre moyen de clients :",nbClientMoyen)
```

```
## nombre moyen de clients : 13
```

```
cat("nombre moyen de clients(calculé avec la forme de Little): ",formeLittle)
```

```
## nombre moyen de clients(calculé avec la forme de Little):  12
```

Pour cet exemple nous fixons les paramètres:

- lambda=20
- mu=15
- D=12

```
cat("attente moyenne :",attenteMoyenne)
```

```
## attente moyenne : 1.5
```

```
cat("nombre moyen de clients :",nbClientMoyen)
```

```
## nombre moyen de clients : 28
```

```
cat("nombre moyen de clients(calculé avec la forme de Little): ",formeLittle)
```

```
## nombre moyen de clients(calculé avec la forme de Little):  29
```

Nous pouvons constater qu'à partir du moment où il y a plus d'arrivée que de départ, le système tend à se saturer. On retrouve globallement la forme de Little : $E(N) = lambda * E(W)$

Conclusion :

Ce TP nous a permis de comparer les générateurs de séquences pseudo-aléatoires à l'aide de tests statistiques. Nous avons remarqué que ce n'est pas parce qu'une séquence de nombres passe un test qu'elle est aléatoire. Et quand bien même la séquence passerait tous les tests statistiques, on ne pourrait pas affirmer que cette séquence est aléatoire (seulement que l'on n'a pas pu réfuter cette hypothèse).