Compte rendu TP - B3246

Baptiste PAULETTO & Louis UNG

9 et 17 mai 2019

Partie 1 : Tests de générateurs pseudos aléatoires

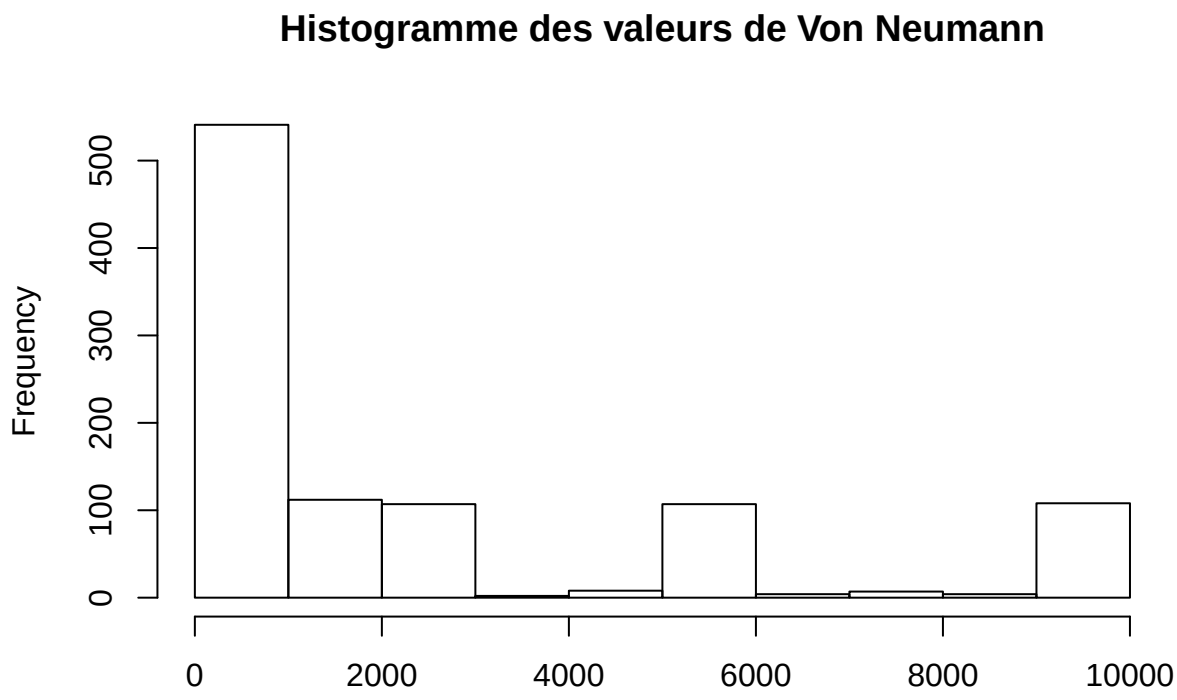
Test visuel

Pour ce premier test, il sera réalisé sur une séquence de 1000 valeurs.

Question 2.1 :

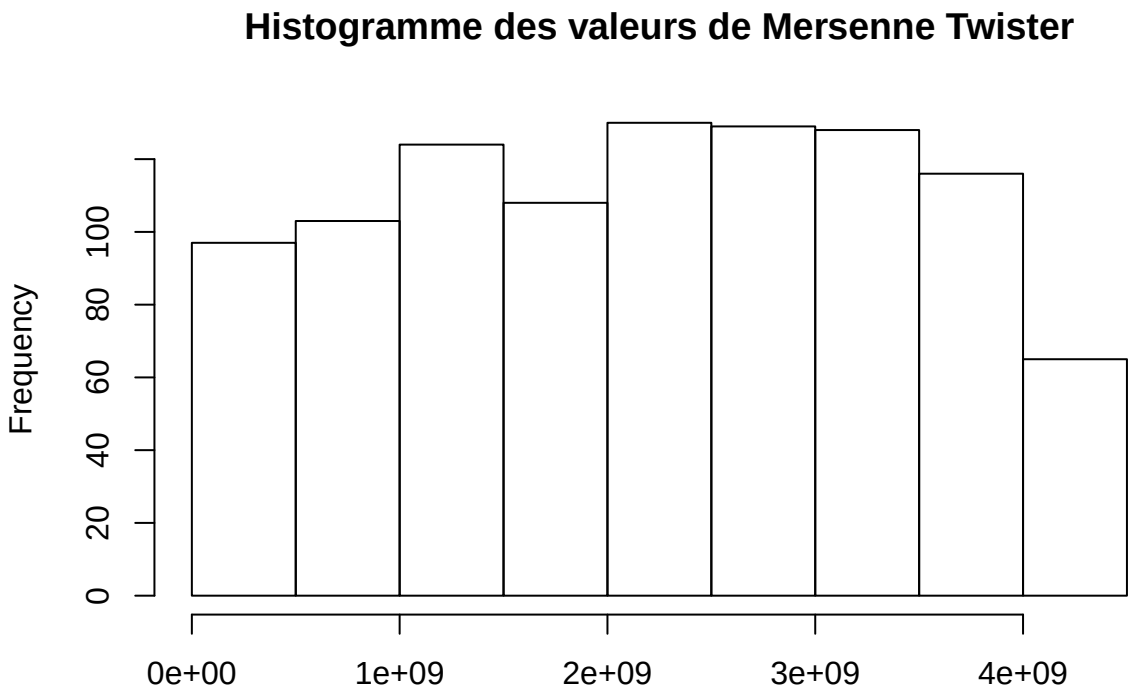
– Von Neumann : La répartition des valeurs avec cette méthode est trés hétérogène, l'essentiel des résultats se trouvent en dessous de 1000 (plus de la moitié d'entre eux).
Explication : La manière de calculer les valeurs est incorrecte, ôter des deux côtés du nombre des chiffres jusqu'à être dans l'intervalle [0,9999] n'est pas une bonne idée, en effet, tous les nombres composés d'un nombre de chiffres impair se retrouvent dans l'intervalle {0, 999}, ce qui explique la présence d'autant de résultats dans cet intervalle.

```
hist(vn[,1],xlab='',main='Histogramme des valeurs de Von Neumann')
```



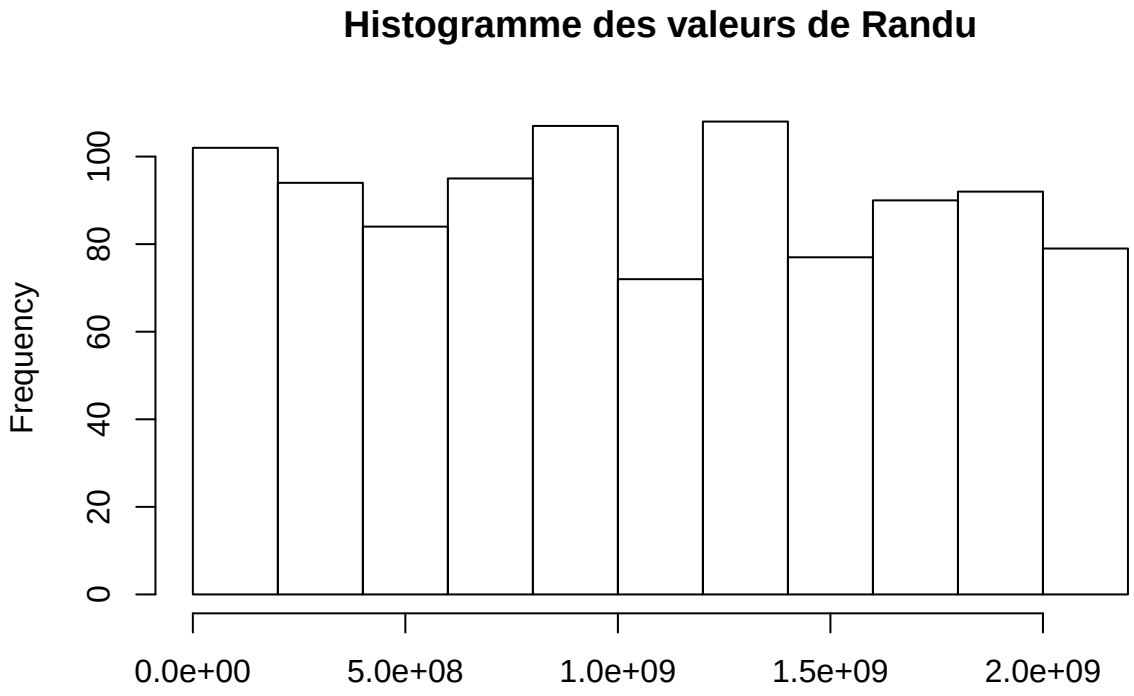
– Mersenne-Twister: La répartition des valeurs avec cette méthode est homogène et étalée dans un grand intervalle, [0 4*10^9], ce qui nous laisse à penser que l'on peut obtenir des valeurs allant de 0 à 2^{32} et qui se révéle particuliérement intéressant pour les questions suivantes.
Explication : Nous pensons que cela vient de la définition et du fonctionnement de Mersenne Twister, qui est uniformément distribué ce qui en fait un générateur de nombres pseudo-aléatoire particulièrement efficace.

```
hist(mt[,1],xlab='',main='Histogramme des valeurs de Mersenne Twister')
```



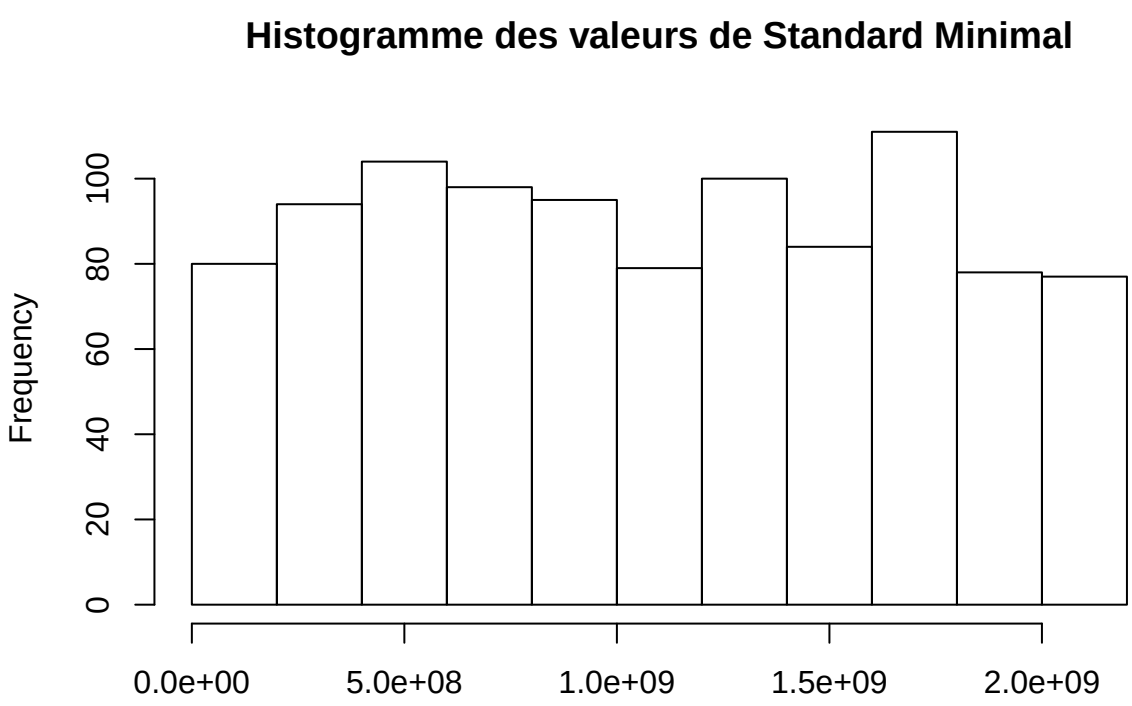
– Randu : Répartition assez homogéne même si l'on retrouve régulièrement des trous, étalée dans l'intervalle [0; 2*10^9].
Explication : Les résultats fournis semblent uniformément distribués mais on remarque notamment sur cet histogramme qu'il posséde un certain nombre de biais, engendrant alors une génération de valeurs qui ne correspondent pas à ce que l'on pourrait attendre d'un générateur pseudo-aléatoire. En effectuant quelques recherches à son sujet, nous remarquons qu'il est particuliérement décrié à cause de son manque de qualité dû aux choix des variables a, c et m.

```
hist(rnd[,1],xlab='',main='Histogramme des valeurs de Randu')
```



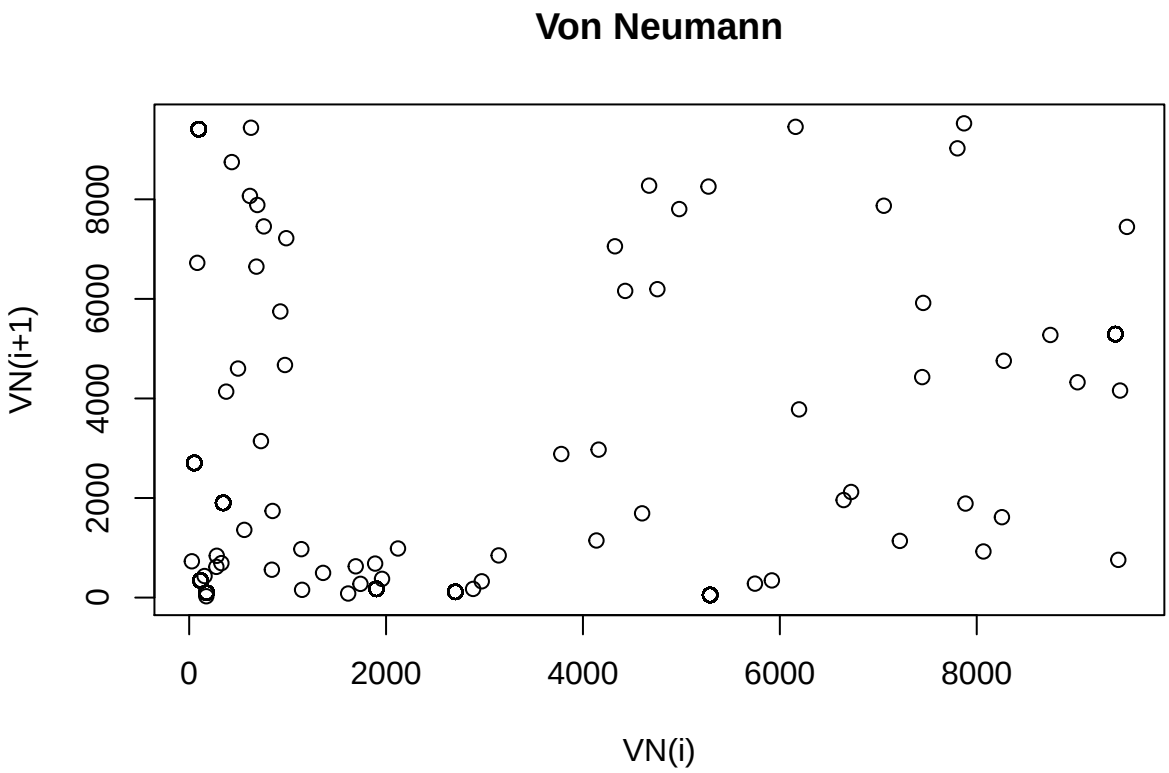
– StandardMinimal: Répartition assez homogène et disposant de peu de “trous” comparé à RANDU, elle est également étalée dans l’intervalle [0;2*10^9].
Explication : Les résultats obtenus avec Standard Minimal semblent plus cohérents et représentatifs de ce que pourrait renvoyer un générateur de congruence linéaire homonyme. En effet, c’est de par sa définition (soit les valeurs a, c et m choisies) qu’il nous permet d’obtenir des valeurs exploitables pour du pseudo-aléatoire.

```
hist(std[,1],xlab='',main='Histogramme des valeurs de Standard Minimal')
```



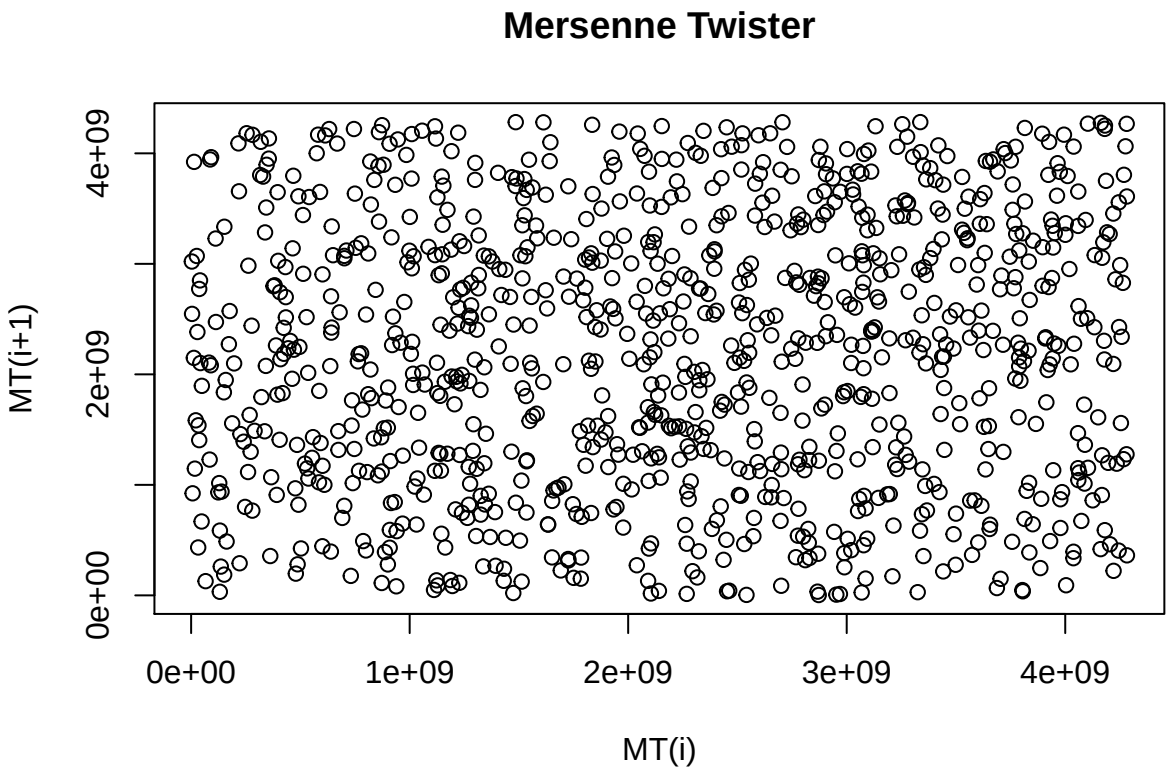
Question 2.2 :

```
plot(vn[1:(Nsimu-1),1],vn[2:Nsimu,1],xlab='VN(i)', ylab='VN(i+1)', main='Von Neumann')
```



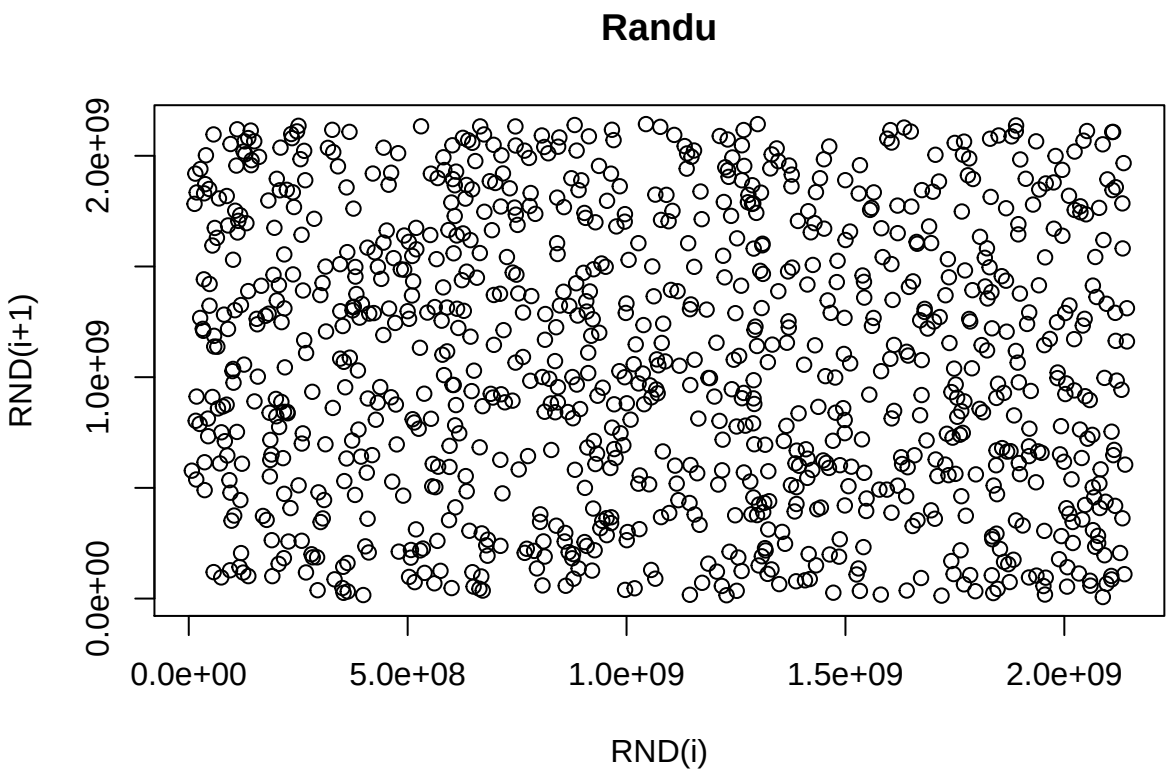
– Commentaire : On observe des “blancs” dans la représentation générée, traduisant ainsi une répartition non homogéne des valeurs dans l’espace des valeurs disponibles (l’intervalle [0;9999]). De plus, on peut aussi en déduire que V(n) et V(n+1) ne sont pas totalement indépendants l’un de l’autre : on le constate par la présence d’amas de points.

```
plot(mt[1:(Nsimu-1),1],mt[2:Nsimu,1],xlab='MT(i)', ylab='MT(i+1)', main='Mersenne Twister')
```



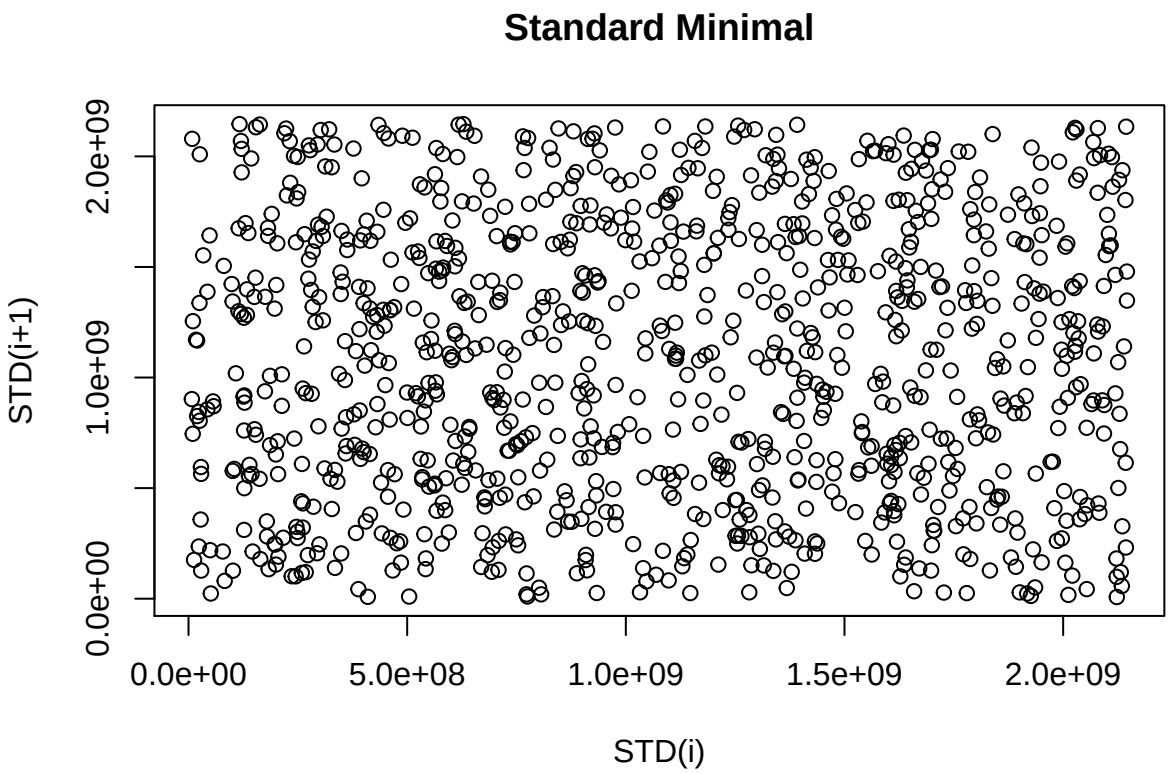
– Commentaire : Contrairement à la représentation précédente, nous avons tous les points qui sont répartis de maniére homogène dans l'espace des valeurs disponibles (l'intervalle [0;2^{32}-1]). On remarque donc une indépendance entre MT(i) et MT(i+1).

```r
plot(rnd[1:(Nsimu-1),1],rnd[2:Nsimu,1],xlab='RND(i)', ylab='RND(i+1)', main='Randu')
```



– Commentaire : D'aprés les résultats de ce test, Randu semble a priori être un générateur de pseudo-aléatoire de bonne qualité puisque la répartition des points est homogéne dans l'espace des valeurs disponibles (l'intervalle $[0;2^{31}]$). On pourrait donc supposer une indépendance entre RND(i) et RND(i+1).

```
plot(std[1:(Nsimu-1),1],std[2:Nsimu,1],xlab='STD(i)', ylab='STD(i+1)', main='Standard Minimal')
```



– Commentaire : Nous avons tous les points qui sont répartis de manière homogène dans l'espace des valeurs disponibles (l'intervalle [0;2^{31}-1]). On remarque donc une indépendance entre STD(i) et STD(i+1).

Test de fréquence monobit

Pour ce test de fréquence monobit ainsi que pour les deux suivants, la séquence sera de 1000 valeurs comme le précédent, mais avec 100 initialisations différentes.

Question 3 :

Fonction utilisée	Nombre de bits considérés	Fréquence obtenue	Observation du test
Von Neumann	14	0.0082607	Proportion 0/1 trés trés inégale
Mersenne Twister	32	0.49102	Proportion 0/1 trés satisfaisante
Randu	31	0.19066	Proportion 0/1 trés peu égale
Standard Minimal	31	0.46116	Proportion 0/1 satisfaisante

– Commentaire : Dans l'ensemble, les résultats obtenus confirment nos attentes par rapport au test précédemment réalisé, en effet, les fonctions Mersenne Twister et Standard Minimal sont en tête du classement avec respectivement 0.49102 et 0.46116 soit quasiment une proportion de bit é 0 et 1 équivalente.
De plus, notre hypothése concernant randu se confirme, la proportion est tout à fait inégale et on se rend véritablement compte qu'il ne peut jouer le rôle de générateur de pseudo-aléatoire convenablement.
Le cas de Von Neumann était déjà écarté, mais ce n'est qu'une confirmation de plus concernant son incapacité à produire du pseudo-aléatoire de qualité.

Test des runs

Question 4 :

Fonction utilisée	Nombre de bits considérés	Fréquence obtenue	Observation du test
Von Neumann	14	4.658169e-08	Inférieur à 0.01
Mersenne Twister	32	0.5111088	Supérieur à 0.01
Randu	31	0.3616877	Supérieur à 0.01
Standard Minimal	31	0.5863881	Supérieur à 0.01

– Commentaire : Lors de ce test des runs, on cherche à connaître la longueur des "runs", soit les suites consécutives de 0 ou de 1. Afin de s'assurer de la qualité de l'aléatoire dans la séquence de bits observée, on doit obtenir un résultat supérieur à 0.01.
A nouveau, Mersenne Twister ainsi que Standard Minimal sont en tête du classement mais Randu retourne lui aussi une valeur bien supérieure à 0.01 ce qui ne nous permet alors pas de l'eloigner dans le cadre de ce test. Von Neumann, fidèle à lui même, prouve une nouvelle fois la mauvaise qualité des séquences qu'il produit avec un résultat trés inférieur à 0.01.

Test d'ordre

Question 5 :

Fonction utilisée	Fréquence obtenue	Observation du test
Von Neumann	NA / 0	Inférieur à 0.01
Mersenne Twister	0.49	Supérieur à 0.01
Randu	0.46	Supérieur à 0.01
Standard Minimal	0.48	Supérieur à 0.01

– Commentaire : Dans ce test d'ordre qui vise à étudier directement la suite de nombre obtenus et non les bits générés, nous fixons la valeur de d à 4. De manière conforme au test précédent, nous retrouvons des valeurs supérieures à 1% pour Mersenne Twitser, Randu et Standard Minimal et une valeur non attribuée pour Von Neumann et parfois 0. Il est intéressant de remarquer que Randu passe également ce test et pourrait nous laisser penser qu'il est, en réalité, capable de produire des suite de nombres conforme au pseudo-aléatoire si nous n'avions pas réalisé le premier test de fréquence monobit.

Partie 2

Files d'attente

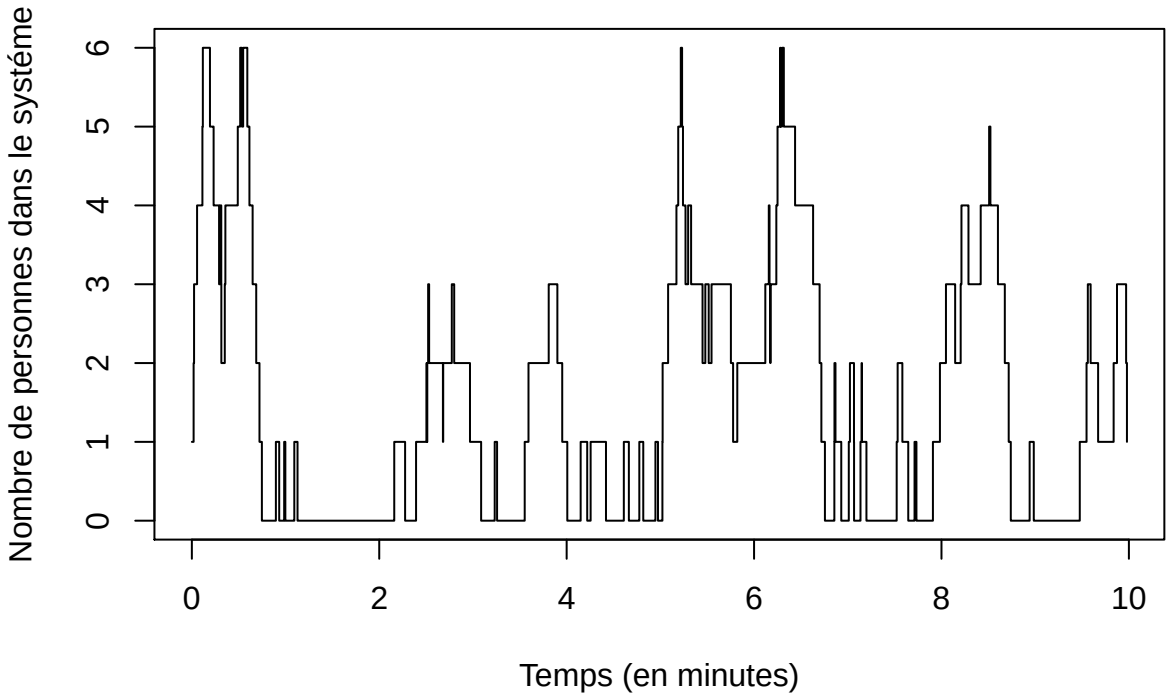
Question 6 :

La fonction FileMM1 construite a pour but de modéliser une file d'attente avec entrée et sortie. Trois principes sont alors fixés :
– Les entrées et sorties sont réalisées en file, une personne aprés l'autre.
– La loi qui représente l'évolution du systéme est considérée comme exponentielle.
– Les temps sont exprimés en minutes.

Question 7 :

La représentation qui suit est celle de l'évolution du nombre de clients en fonction du temps qui passe. Le temps entre deux personnes qui arrivent dans le système est choisi aléatoirement (de manière exponentielle) grâce à la fonction rexp().
Pour des valeurs choisies arbitrairement d'alpha (8), mu (12) et D (10), nous obtenons l'affichage suivant :

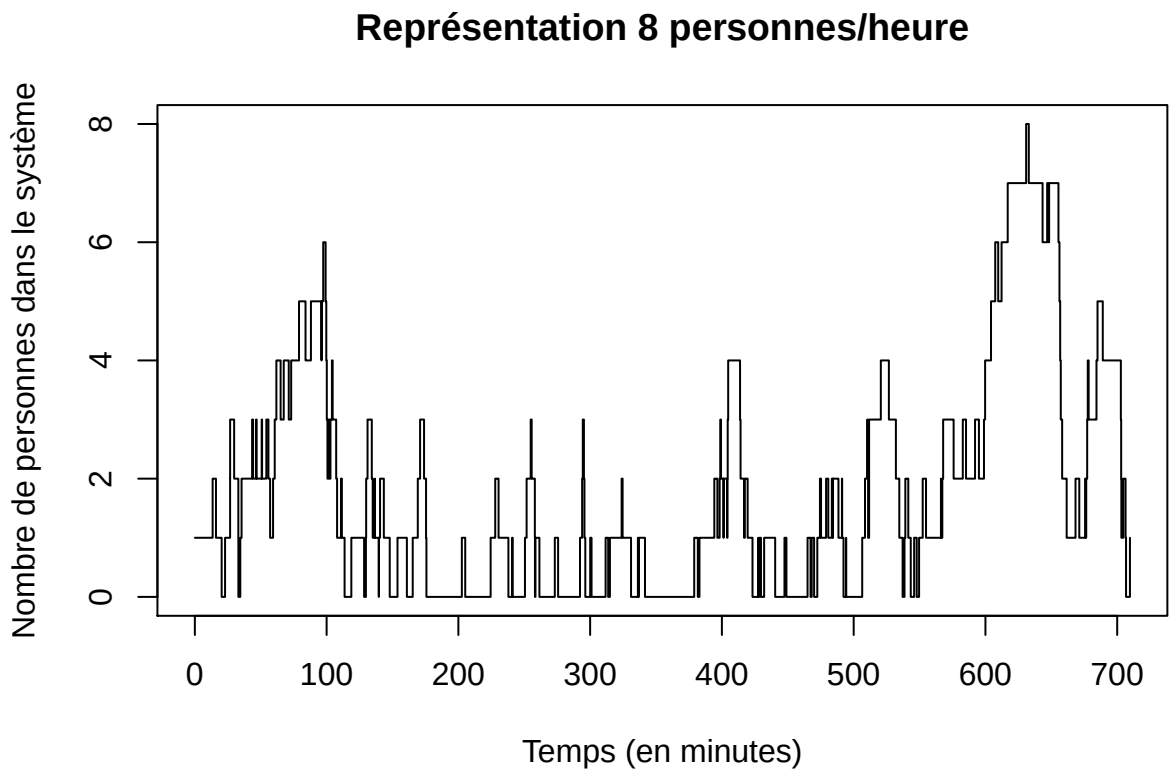
```
plot(EvolClients(liste),type='s',xlab='Temps (en minutes)', ylab='Nombre de personnes dans le systéme
```



Application

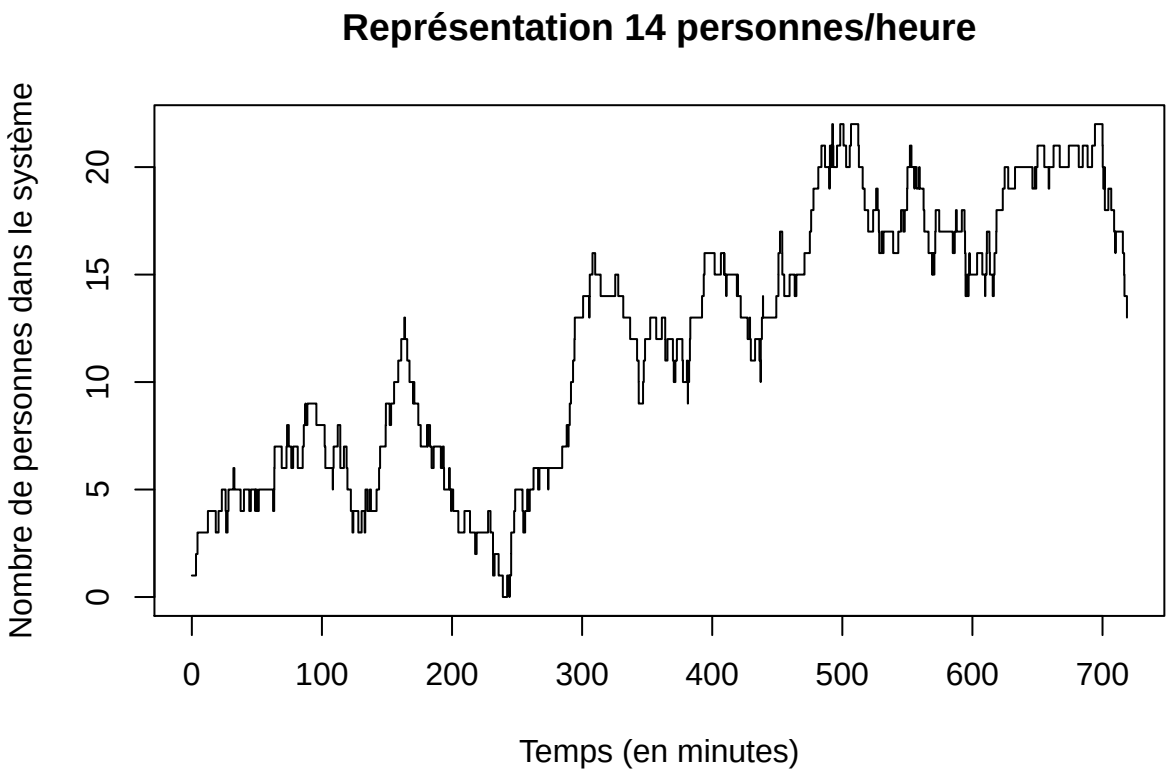
Les quatres graphes suivants représententent tous l'activité de la file d'attente pendant 12 heures avec, en moyenne 15 personnes qui partent par heure et un nombre de personnes/heure qui arrivent différent, respectivement les nombres de personnes sont 8, 14, 15 et 20.

```
plot(evol,type='s',xlab='Temps (en minutes)', ylab='Nombre de personnes dans le système', main='Représe
```



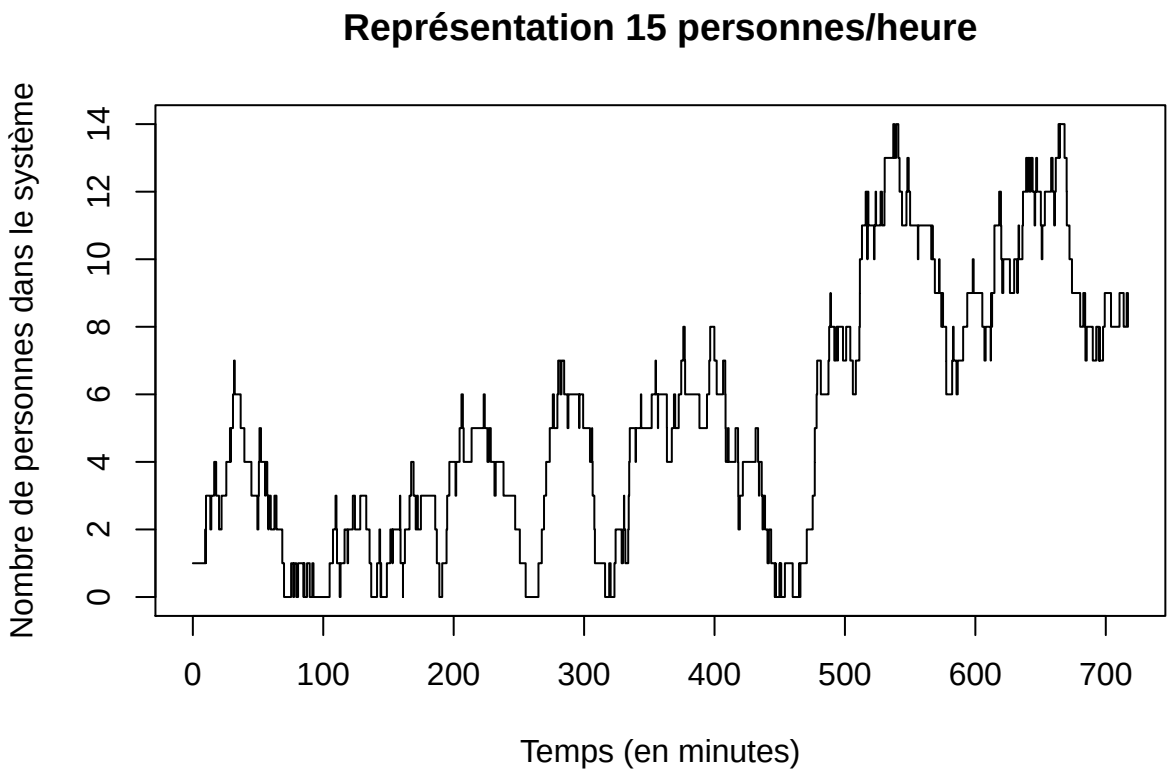
– Commentaire :
Après avoir réalisé plusieurs observations des graphiques obtenus avec ces paramètres, nous remarquons une convergence en probabilité de l'évolution du nombre de client en fonction du temps.

```
plot(evol,type='s',xlab='Temps (en minutes)', ylab='Nombre de personnes dans le système', main='Représe
```



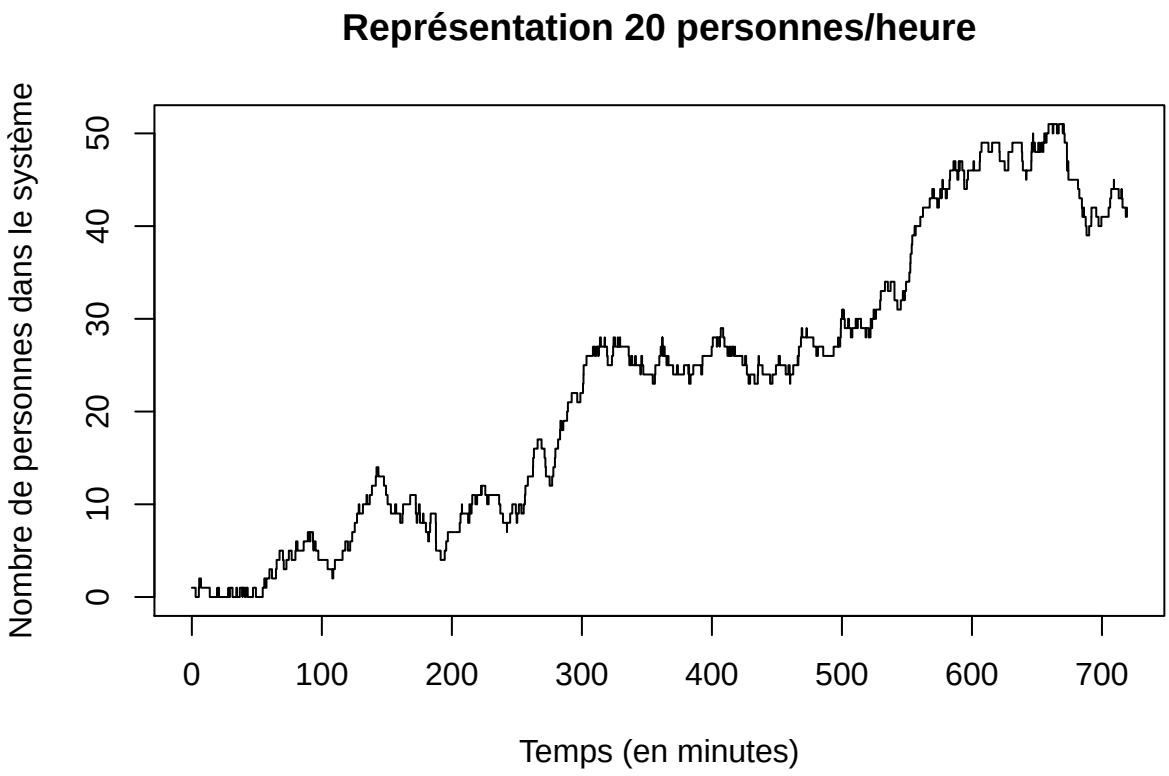
– Commentaire : Après avoir réalisé plusieurs observations des graphiques obtenus avec ces paramètres, nous remarquons, comme pour le précédent, une convergence en probabilité de l'évolution du nombre de client en fonction du temps. Cependant, le nombre de client maximal est souvent beaucoup plus élevé que pour les paramétres précédents (mais on pouvait s'y attendre puisqu'on ne fait qu'augmenter le nombre de personnes qui arrivent chaque heure).

```
plot(evol,type='s',xlab='Temps (en minutes)', ylab='Nombre de personnes dans le système', main='Représe
```



– Commentaire : Après avoir réalisé plusieurs observations du graphique obtenu avec ces paramètres, nous ne sommes pas parvenus à en déduire soit une convergence, soit une divergence en probabilité, nous pensons alors qu'il est impossible de prédire comment va varier cette évolution. Nous nous retrouvons alors dans l'incapacité d'estimer le nombre de clients à chaque instant t, ce qui était possible avec les deux représentations précédentes.

```r
plot(evol,type='s',xlab='Temps (en minutes)', ylab='Nombre de personnes dans le système', main='Représe
```



– Commentaire : Après avoir réalisé plusieurs observations du graphique obtenu avec ces paramètres, nous sommes arrivés à la conclusion que l'évolution du nombre de clients en fonction du temps diverge en probabilité. En effet, il y a beaucoup plus d'arrivées que de départs et la file d'attente finit par être saturée, ce qui augmente de maniére significative le nombre de clients présents dans le systéme au cours du temps.

Question 8

Lambda	E(N) exp	E(N) th	E(W) exp	E(W) th
8	1.11	1.14	7.19	8.55
14	14.27	14.00	61.37	60.00

– Commentaire : Nous ne calculons pas les valeurs pour *lambda* >= 15 car cela impliquerait que alpha soit supérieur ou égal à 1. Cela ne nous intéresse pas car nous quitterions le régime stationnaire. Les valeurs théoriques sont calculées à partir de la formule de Little : E(N) = alpha/(1-alpha) et E(W) = E(N)/lambda. Aussi, pour calculer les valeurs expérimentales, nous avons augmenté la période de temps sur 1200 heures afin d'avoir des valeurs plus fiables (plus le nombre de répétitions augmente, plus les valeurs expériementales convergent vers les valeurs théoriques).