Génération de nombres aléatoires et probabilités

Billy VILLEROY Hélène DOS SANTOS Seynabou SARR

3IF 2022-2023

Contents

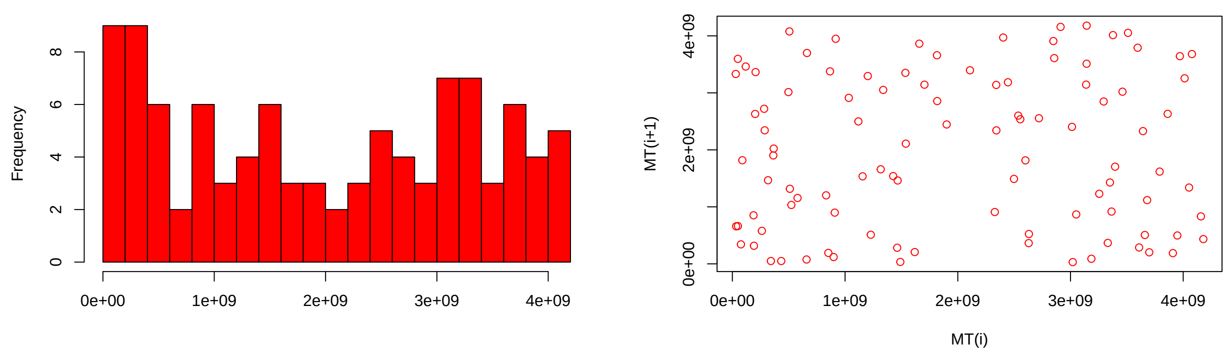
Partie I : Générateurs pseudo-aléatoires

1. Mise en place et études graphiques

A. Exemples

Dans un premier temps, nous avons généré des nombres pseudo-aléatoires à l'aide de l'ensemble des générateurs dont vous retrouverez l'implémentation dans les annexes[1]. Voici les résultats obtenus :

- L'histogramme représente la fréquence d'apparition des nombres sur l'intervalle en ordonnée.
- Le graphique représente chaque nombre en fonction de celui qui lui précède, il permet d'évaluer l'étendue, entre les valeurs, générée par l'algorithme.

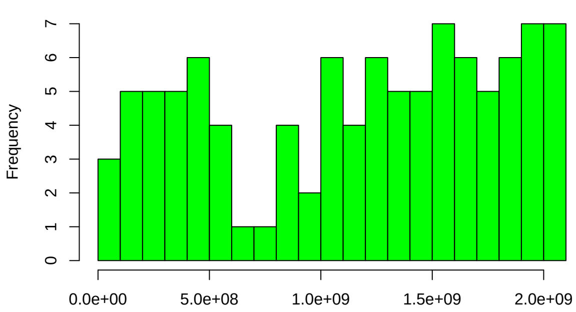


(a) Fréquence d'apparition des nombres générés

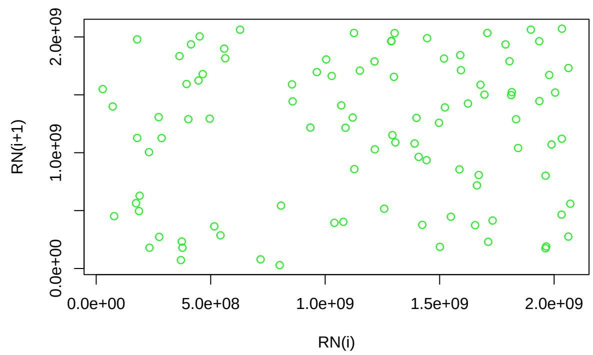
(b) Sucesseurs en fonction des prédecesseurs

Figure 1: Générateur de Mersenne Twister (Graine : 1502, 100 générations)

[1]Voir p. 10

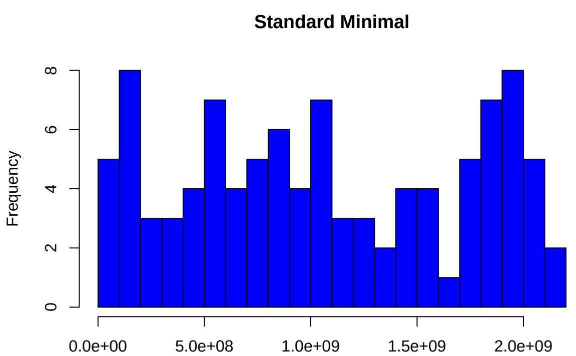


(a) Fréquence d'apparition des nombres générés

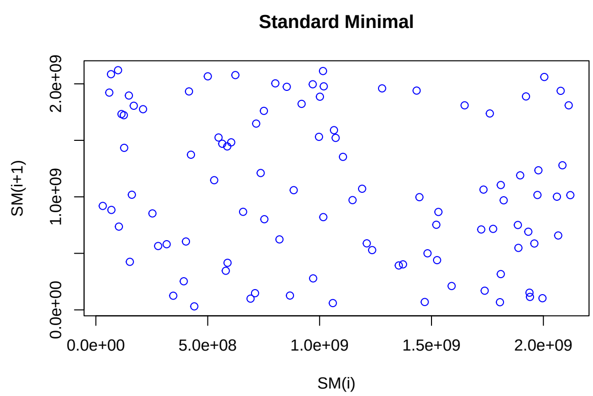


(b) Sucesseurs en fonction des prédecesseurs

Figure 2: Générateur RANDU (Graine : 5645, 100 générations)

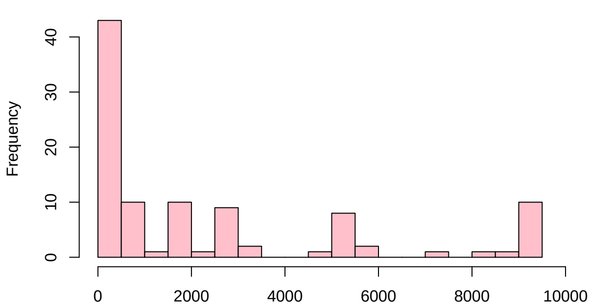


(a) Fréquence d'apparition des nombres générés

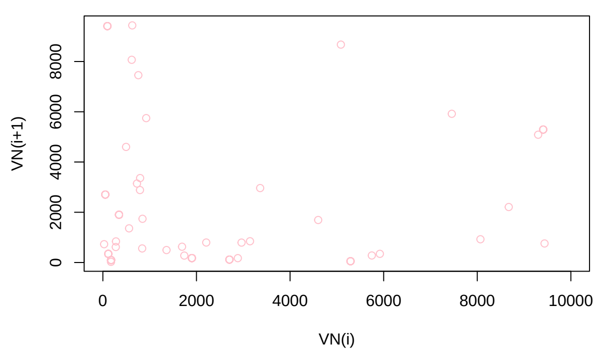


(b) Sucesseurs en fonction des prédecesseurs

Figure 3: Générateur Standard Minimal (Graine : 9575, 100 générations)



(a) Fréquence d'apparition des nombres générés

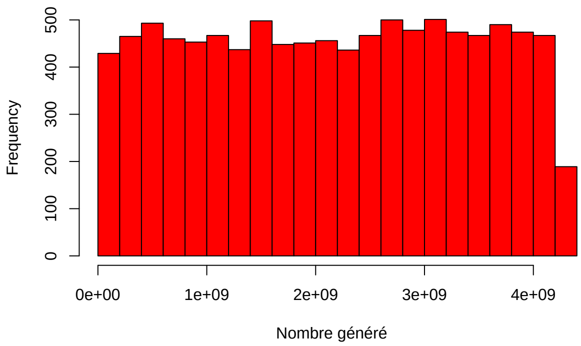


(b) Sucesseurs en fonction des prédecesseurs

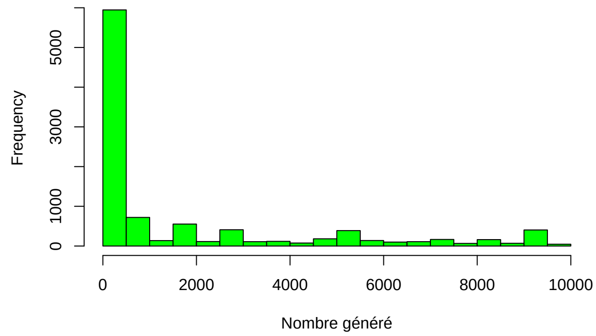
Figure 4: Générateur de Von Neuman (Graine : 3454, 100 générations)

B. Vision globale

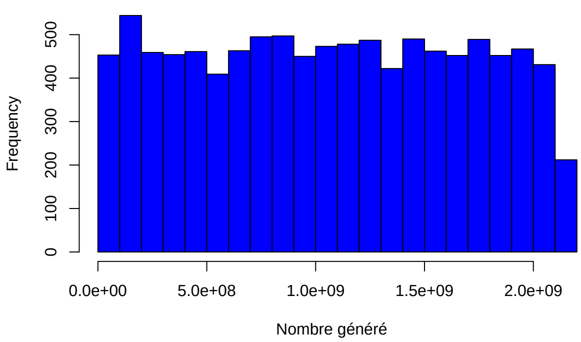
Pour donner une vision d'ensemble, nous avons généré 100 séquences avec des graines\footnote{Les valeurs ont été divisées par 2 afin de rentrer dans l'intervalle {0,2ˆ31} produites par Mersenne Twister avec la graine 4675 :



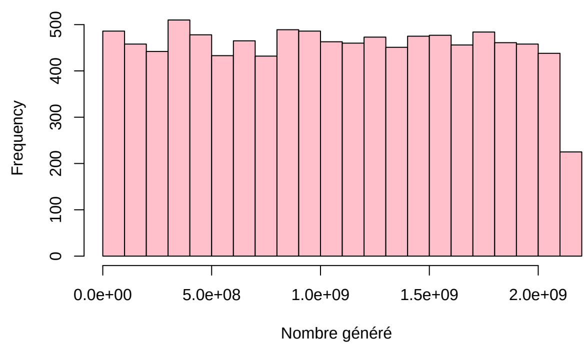
(a) Mersenne Twister



(b) Von Neuman



(c) RANDU



(d) Standard Minimal

Figure 5: Etude sur 100 séquences de cardinal 100, Graine aléatoire

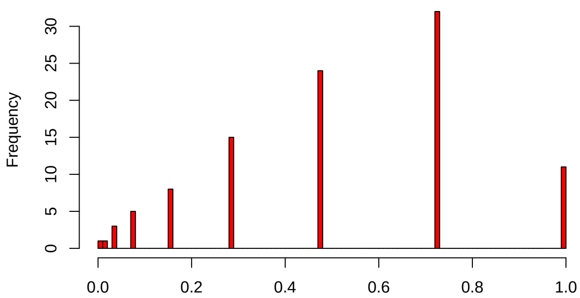
Analyse graphique

Dans l'ensemble, les nombres produits sont bien répartis sur les intervalles considérés à part pour Von Neuman, qui génère une majorité de nombres inférieurs à 2000. Mais cela n'est qu'une apréciation visuelle, par la suite nous allons effectuer divers tests sur nos générateurs afin de déterminer si les nombres et séquences produites peuvent êtres considérés comme aléatoires.

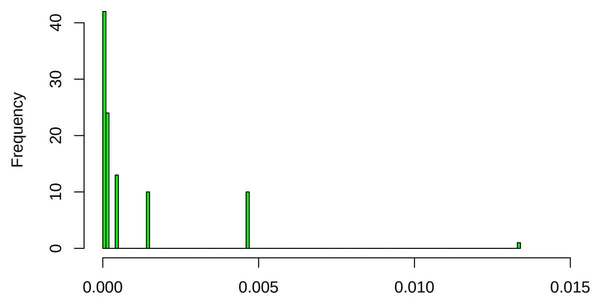
2. Etudes probabilistes

Les deux premières parties traitent de tests réalisés sur les bits[2] des nombres générés.

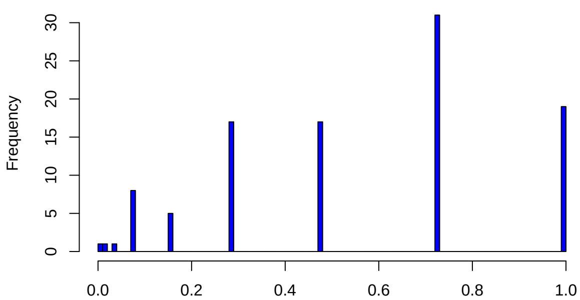
A. Tests de répartition[3]



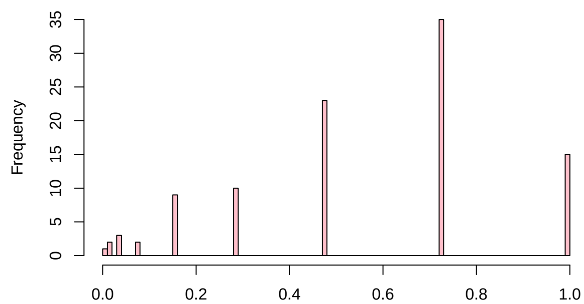
(a) Mersenne Twister



(b) Von Neuman



(c) RANDU

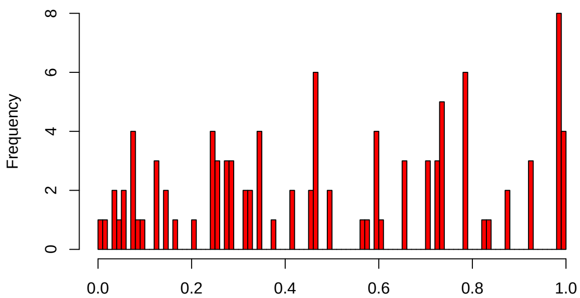


(d) Standard Minimal

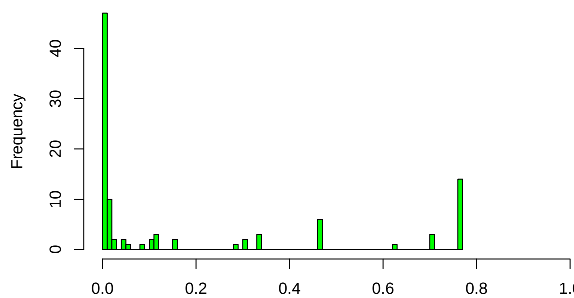
Figure 6: Etude de la répartition des bits sur 100 séquences de cardinal 100, Graines aléatoires

[2]Générés par la fonction *binary*, p. 11
[3]Voir *Frequency* p. 11

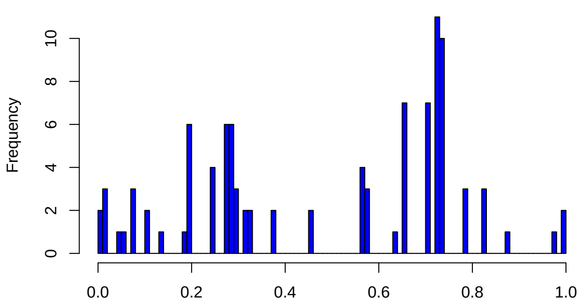
B. Tests d'ordre[4]



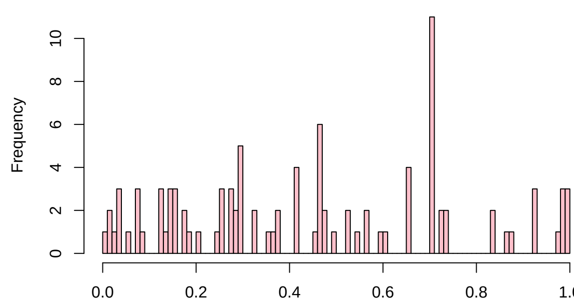
(a) Mersenne Twister



(b) Von Neuman



(c) RANDU



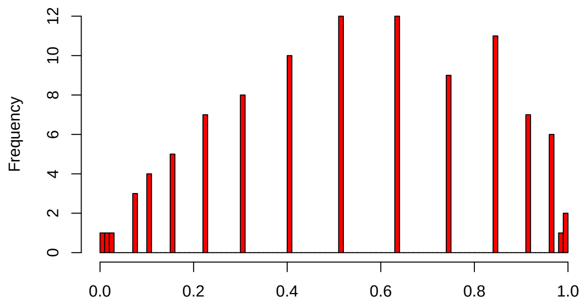
(d) Standard Minimal

Figure 7: Etude de l'ordonnancement des bits sur 100 séquences de cardinal 100, Graines aléatoires

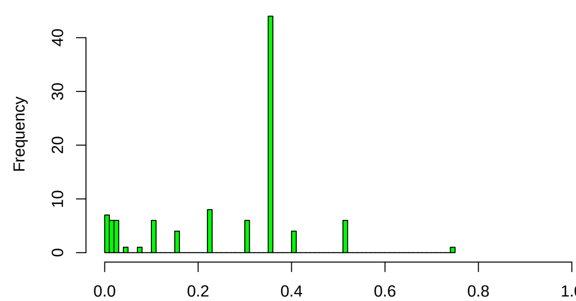
[4]Voir *Runs* p. 11

C. Tests uniformes[5]

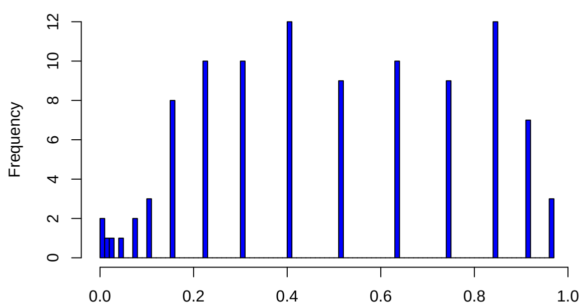
Ici nous nous intéressons aux nombres dans leur représentation décimale et vérifions si la séquence produite suit la loi uniforme (chaque nombre à autant de chance d'apparaître qu'un autre).



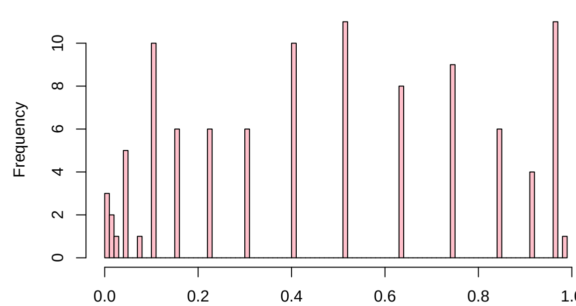
(a) Mersenne Twister



(b) Von Neuman



(c) RANDU



(d) Standard Minimal

Figure 8: Etude de la loi uniforme sur 100 séquences de cardinal 100, Graines aléatoires

[5]Voir la documentation de order.test : www.rdocumentation.org/packages/randtoolbox/versions/2.0.3/topics/order.test

Partie II : Etudes des files d'attentes

Annexes

Algorithme des générateurs

Von Neuman

```
VonNeumann <- function(n, p=1, graine)
{
  x <-  rep(graine,n*p+1)
  for(i in 2:(n*p+1))
  {
    numbers <- strsplit(format(x[i-1]^2,scientific=FALSE),'')[[1]]
    while(length(numbers)>4){
        numbers <- numbers[2:(length(numbers)-1)]
    }
    x[i] <- as.numeric(numbers)%*%(10^seq(length(numbers)-1,0,-1))
  }
  x <- matrix(x[2:(n*p+1)],nrow=n,ncol=p)
  return(x)
}
```

Mersenne Twister

```
MersenneTwister <- function(n, p=1, graine)
{
  set.seed(graine,kind='Mersenne-Twister')
  x <- sample.int(2^32-1,n*p)
  x <- matrix(x,nrow=n,ncol=p)
  return(x)
}
```

RANDU

```
RANDU <- function(n=1,k = 10,graine)
{
  x <-  rep(graine,k*n+1)
  for (i in 2:(k*n+1)) {
    x[i] <- (65539*x[i-1])%%(2^31)
  }
  x <- matrix(x[2:(k*n+1)],nrow=n,ncol=k)

  return(x)
}
```

Standard Minimal

```
StandardMinimal <- function(n=1,k = 10,graine)
{
  x <-  rep(graine,k*n+1)

  for (i in 2:(k*n+1)) {
```

```
    x[i] <- (16807*x[i-1])%% (2^31-1)
  }

  x <- matrix(x[2:(k*n+1)],nrow=n,ncol=k)

  return(x)
}
```

Binary

```
binary <- function(x)
{
  if((x<2^31)&(x>=0))
    return( as.integer(intToBits(as.integer(x))) )
  else{
    if((x<2^32)&(x>0))
      return( c(binary(x-2^31)[1:31], 1) )
    else{
      cat('Erreur dans binary : le nombre etudie n est pas un entier positif en 32 bits.\n')
      return(c())
    }
  }
}
```

Algorithmes des tests

Frequency ou l'étude de la répartition binaire

```
Frequency <- function(x, nb)
{
  for(i in 1:nb){
    if(x[i]==0)
      x[i] <- -1
  }

  S <- sum(x)

  sObs <- abs(S)/sqrt(nb)

  p <- 2*(1-pnorm(sObs,0,1))

  return(p)
}
```

Runs ou l'étude de l'ordre binaire

```
Runs <- function(x, nb)
{
```

```r
  #Pre-test
  S <- 0
  for(i in 1:nb){
    if(x[i]==1)
      S <- S+1
  }
  pi <- S/nb

  if(abs(pi-1/2)>=(2/sqrt(nb))){
    return(0.0)
  }

  #Test
  VnObs <- 0
  for(i in 2:nb-1){
    if(x[i+1]!=x[i])
      VnObs <- VnObs + 1
  }
  VnObs <- VnObs + 1
  frac <- (abs(VnObs-2*nb*pi*(1-pi))/(2*sqrt(nb)*pi*(1-pi)))
  P <- 2*(1-pnorm(frac,0,1))

  return(P)
}
```