

GÉNÉRATION DE NOMBRES ALÉATOIRES ET PROBABILITÉS

Billy VILLEROY Hélène DOS SANTOS Seynabou SARR

Contents

Générateurs pseudo-aléatoires	3
Mise en place et études graphiques	3
Etudes probabilistes	6
 Etudes des files d'attentes	 9
La file M/M/1	9
 Annexes	 9
Algorithmes des générateurs	9
Algorithme de <i>binary</i>	10
Algorithmes des tests	10

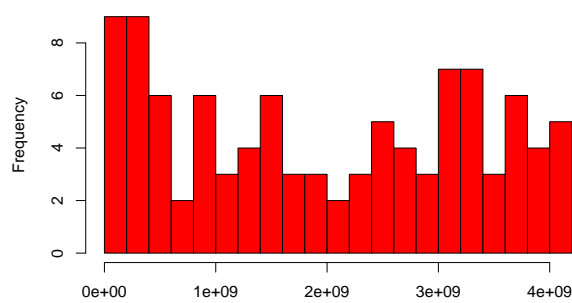
Partie I : Générateurs pseudo-aléatoires

1. Mise en place et études graphiques

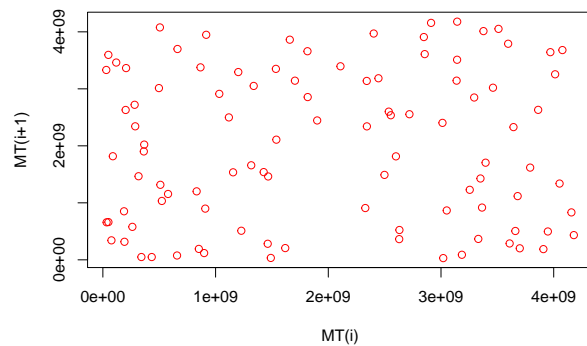
A. Exemples

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 annexes¹. 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.



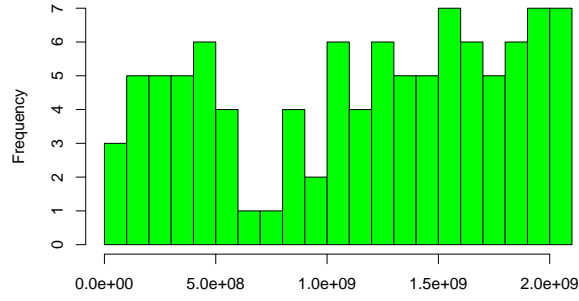
(a) Fréquence d'apparition des nombres générés



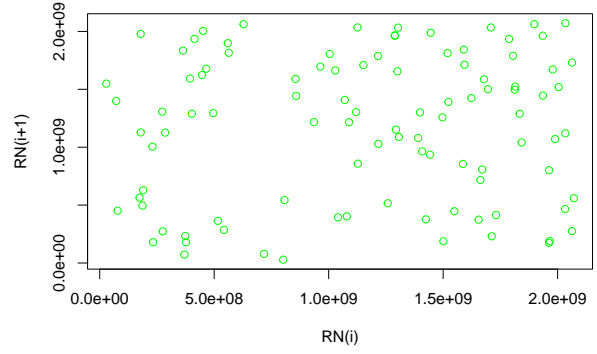
(b) Successeurs en fonction des prédécesseurs

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

¹Voir p. 9

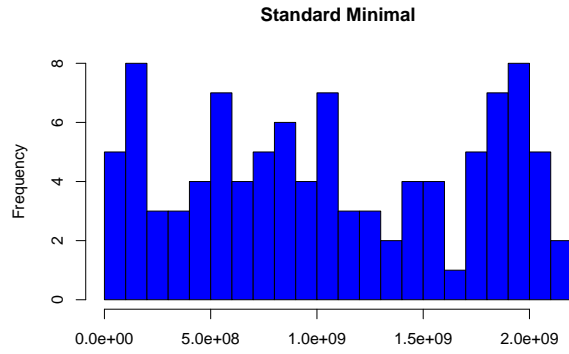


(a) Fréquence d'apparition des nombres générés

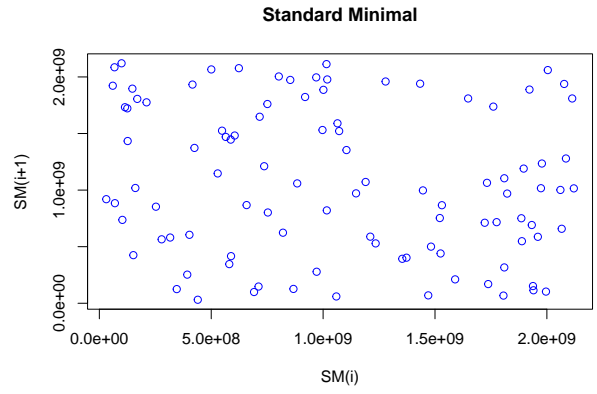


(b) Successeurs en fonction des prédécesseurs

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

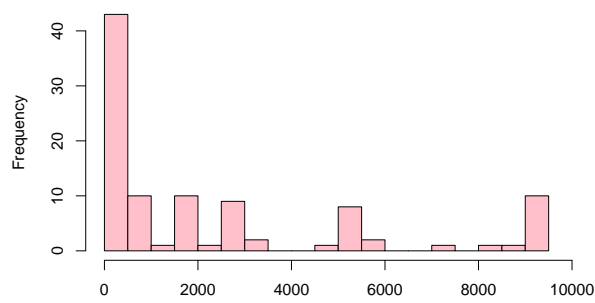


(a) Fréquence d'apparition des nombres générés

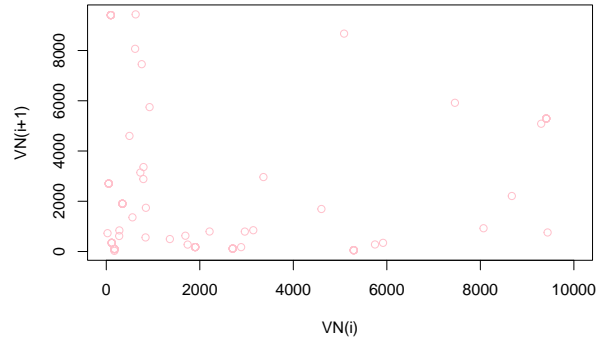


(b) Successeurs en fonction des prédécesseurs

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



(a) Fréquence d'apparition des nombres générés

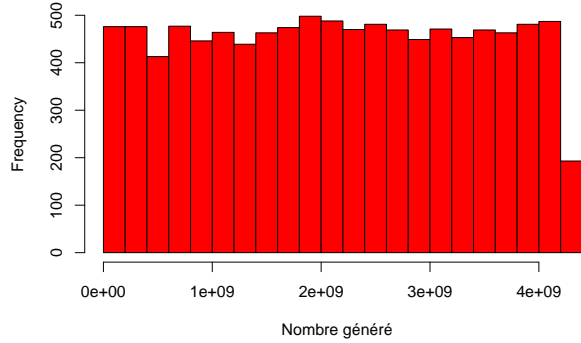


(b) Successeurs en fonction des prédécesseurs

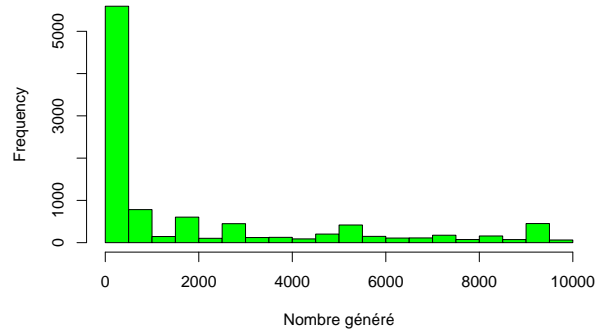
Figure 4: Générateur de Von Neuman (Graine : 3454, 100 générations)

B. Vision globale

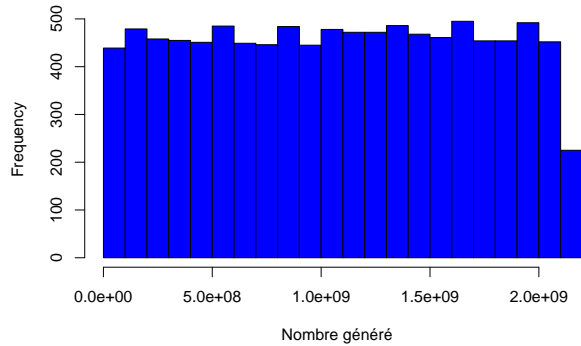
Pour donner une vision d'ensemble, nous avons généré 100 séquences avec des graines² produites par la fonction **sample** de R :



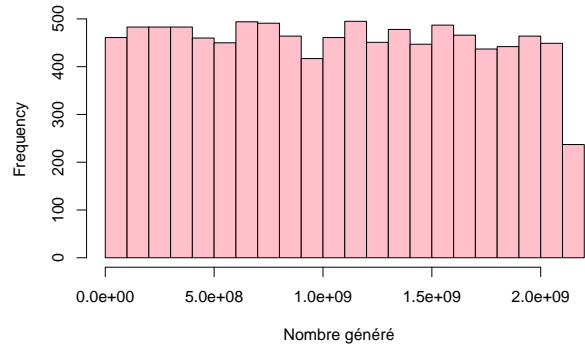
(a) Mersenne Twister



(b) Von Neuman



(c) RANDU



(d) Standard Minimal

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

Analyse graphique

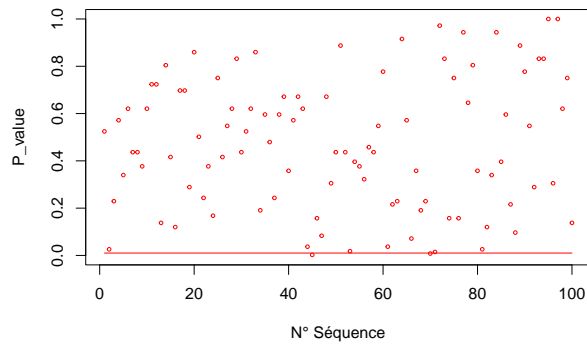
Dans l'ensemble, les nombres produits sont bien répartis sur les intervalles considérés à part pour Von Neuman, qui génère une majorité de nombres inférieurs à 2000. Mais cela n'est qu'une appréciation visuelle, par la suite nous allons effectuer divers tests sur nos générateurs afin de déterminer si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

²Vous pouvez les retrouver p. 9

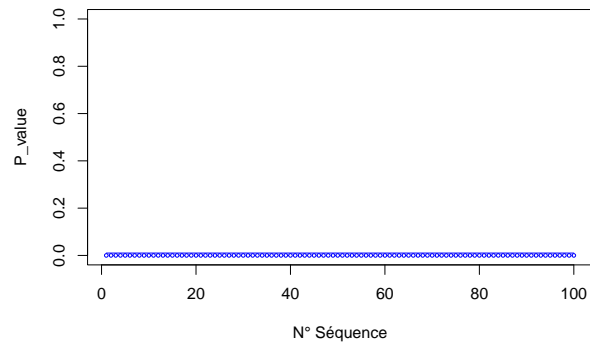
2. Etudes probabilistes

Les deux premières parties traitent de tests réalisés sur les bits³ des nombres générés.

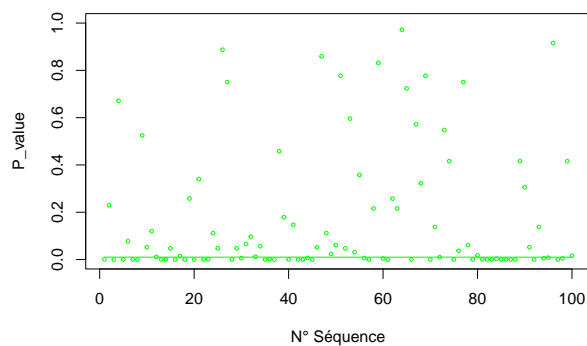
A. Tests de répartition⁴



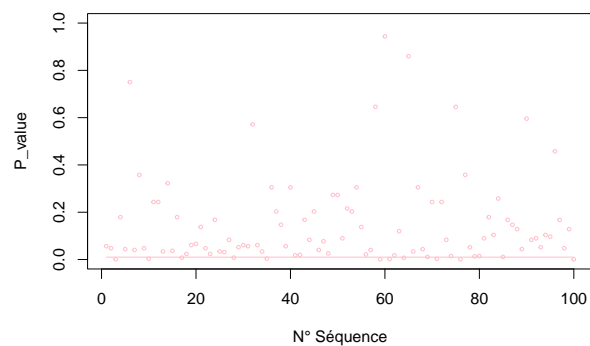
(a) Mersenne Twister



(b) Von Neuman



(c) RANDU



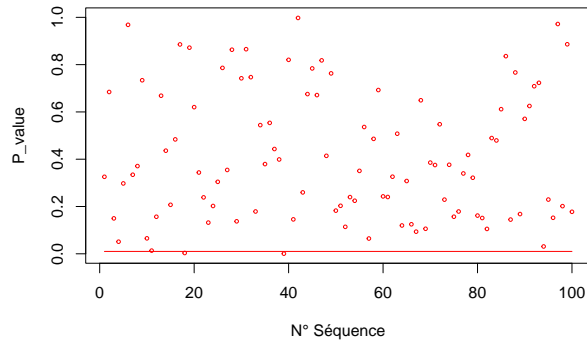
(d) Standard Minimal

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

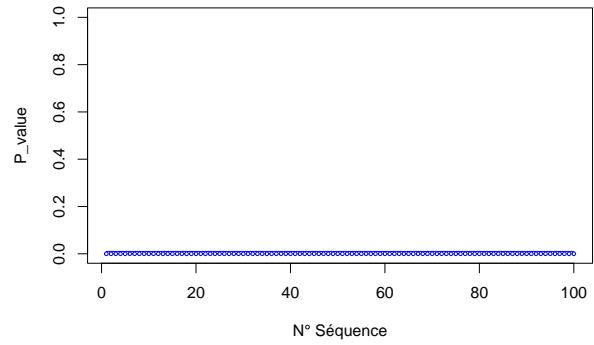
³Générés par la fonction *binary*, p. 10

⁴Voir *Frequency* p. 10

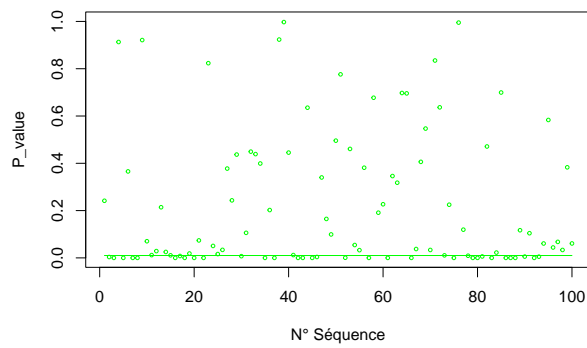
B. Tests d'ordre⁵



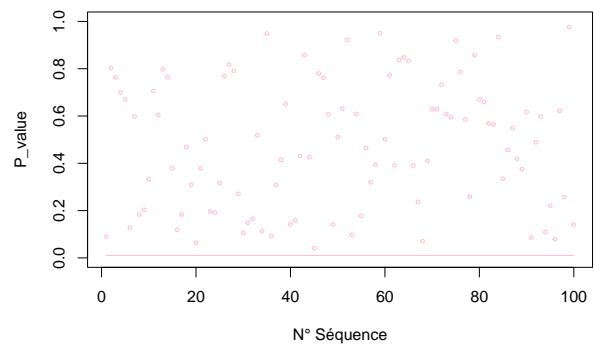
(a) Mersenne Twister



(b) Von Neuman



(c) RANDU



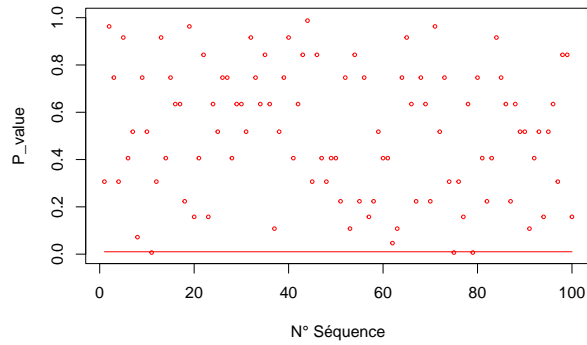
(d) Standard Minimal

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

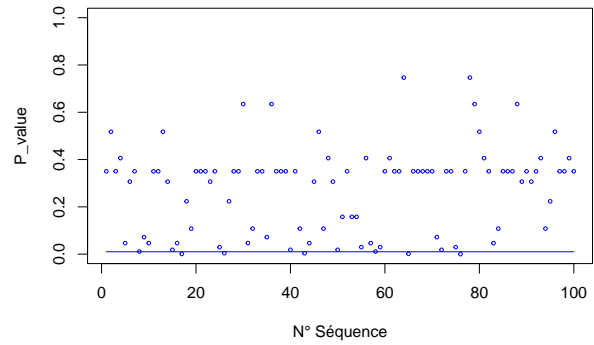
⁵Voir *Runs* p. 11

C. Tests uniformes⁶

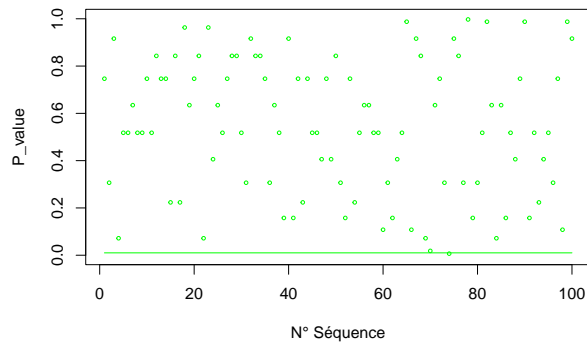
Ici nous nous intéressons aux nombres dans leur représentation décimale et vérifions si la séquence produite suit la loi uniforme (chaque nombre à autant de chance d'apparaître qu'un autre).



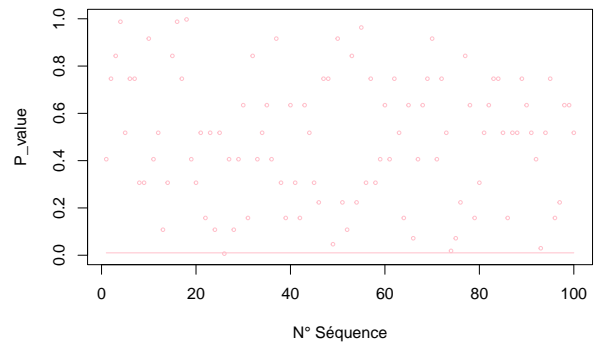
(a) Mersenne Twister



(b) Von Neuman



(c) RANDU



(d) Standard Minimal

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

⁶Voir la documentation de `order.test` : www.rdocumentation.org/packages/randtoolbox/versions/2.0.3/topics/order.test

Partie II : Etudes des files d'attentes

1. La file M/M/1

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())
    }
  }
}
```

Graines

```
## [1] 1.3e+09 1.0e+09 1.4e+09 1.7e+09 7.7e+08 1.3e+09 3.9e+08 7.2e+08 1.2e+09
## [10] 7.1e+08 1.5e+09 1.5e+08 1.5e+09 2.2e+08 1.6e+09 2.4e+07 1.9e+08 2.8e+08
## [19] 1.0e+09 1.8e+09 1.1e+08 6.5e+08 4.0e+08 1.8e+09 2.0e+09 1.5e+09 2.6e+08
## [28] 1.4e+09 7.3e+08 9.3e+08 1.2e+09 3.7e+08 1.4e+09 2.0e+09 8.1e+08 2.1e+09
## [37] 9.3e+08 4.6e+08 1.2e+09 2.0e+09 2.0e+09 1.1e+09 5.6e+08 2.0e+09 1.7e+09
## [46] 2.1e+09 8.2e+08 1.4e+09 2.1e+09 2.0e+09 3.9e+08 5.3e+08 1.6e+08 8.5e+08
## [55] 2.1e+09 2.1e+09 7.7e+08 1.1e+09 1.9e+09 2.0e+09 9.0e+08 2.1e+08 1.2e+09
## [64] 1.7e+09 2.0e+09 1.6e+09 1.3e+09 1.8e+09 1.5e+09 1.9e+09 7.2e+08 1.8e+09
## [73] 1.8e+09 1.5e+09 1.5e+08 1.3e+09 1.4e+09 1.6e+09 1.6e+09 5.1e+08 8.7e+07
## [82] 1.8e+09 6.4e+08 1.2e+09 1.8e+09 1.1e+08 1.8e+09 1.1e+09 2.1e+09 1.5e+09
## [91] 1.4e+09 3.7e+08 9.4e+08 5.3e+07 1.9e+09 4.3e+08 5.4e+08 2.1e+09 1.5e+09
## [100] 1.8e+09
```

Algorithmes des tests

Frequency ou l'étude de la répartition binaire

```

Frequency <- function(x, nb)
{
  cn <- binary(x[1])
  b <- cn[1:nb]
  for(i in 2:length(x)){
    cn <- binary(x[i])
    b <- c(b,cn[1:nb])
  }
  for(i in 1:length(b)){
    if(b[i]==0)
      b[i] <- -1
  }

  S <- sum(b)

  sObs <- abs(S)/sqrt(length(b))

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

  return(p)
}

```

Runs ou l'étude de l'ordre binaire

```

Runs <- function(x, nb)
{
  cn <- binary(x[1])
  b <- cn[1:nb]
  for(i in 2:length(x)){
    cn <- binary(x[i])
    b <- c(b,cn[1:nb])
  }
  #Pre-test
  S <- sum(b)
  pi <- S/length(b)

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

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

  return(P)
}

```