RAPPORT DE SIMULATION
-CyTech Pau-

Introduction à la simulation

Projet réalisé par
Pierre NICHELE
Antonin CUYALA
Baptiste PLAUT-AUBRY
Romain CORRAL
ING 1 Groupe-2

Projet encadré par
Boris LABRADOR

Sommaire

I-Introduction au sujet

Ce rapport s'inscrit dans une démarche de simulation du cours de probabilité étudié lors du premier semestre. L'objectif de celui-ci est de réaliser différents exercices dans le but de modéliser les différentes notions vues au cours du module de modèles probabilistes.
Les exercices proposés traitent des différentes notions :

- Exercice 1 : Simulation de variables aléatoires réelles
- Exercice 2 : Simulation de variables aléatoires réelles par inversion
- Exercice 3 : Lois des grands nombres
- Exercice 4 : Théorème Central Limite
- Exercice 5 : Méthode de Box Muller

II-Choix de programmation

Pour ce TP noté, nous avons fait le choix d'utiliser R Studio, pour diverses raisons :
D'abord, dans le cadre d'un TP lié à la simulation de probabilité, l'utilisation de R semble plus pertinente afin de modéliser plus aisément des fonctions mathématiques liées aux lois de probabilités.
Enfin, nous avons pu avoir un premier contact avec R au premier semestre en Data Exploration et il nous paraissait pertinent de l'utiliser à nouveau pour ce projet, même si nous n'avons que peu d'expérience avec ce langage. Néanmoins, n'ayant dans le groupe pas d'expérience avec Python récente, nous avons choisi de ne pas coder en Python.
Par ailleurs, nous avons tenté de respecter au mieux les contraintes liées à cet exercice, en utilisant la fonction runif et en organisant nos codes autour de fonctions.

III-Difficultés rencontrées

La première difficulté rencontrée est que nous n'avions jamais réellement codé en R. Comme précisé précédemment, nous avons fait le choix de coder dans ce langage suite au cours de Data Exploration du premier semestre, mais également de notre méconnaissance de Python. Néanmoins bien que nous ayons eu un premier contact avec R, cela n'était que de l'étude de données et non de la programmation à part entière. Nous avons donc dû nous

familiariser avec la syntaxe de programmation de R, comment rédiger une fonction, faire des boucles, etc.

Nous avons aussi rencontré différents problèmes par rapport aux graphiques à réaliser. En effet, nous n'avons pas précisément vu comment réaliser de graphiques en cours et nous avons ainsi dû nous informer sur les différentes manières de tracer des graphiques/histogrammes et des courbes. Il est par ailleurs difficile de savoir si nous obtenons le résultat attendu lors de ce genre de simulations, et il a alors fallu faire confiance à nos résultats en faisant différents jeux d'essais.

IV-Analyse des résultats

Exercice 1 : Simulation de variables aléatoires réelles

L'exercice 1 est un exercice de simulation de variables aléatoires réelles selon différentes lois. Dans un premier temps nous simulons une loi de Bernoulli avec comme paramètre p.

```r
bernoulli <- function(p) { #Fonction pour simuler bernouilli
   nbr <-runif(1) #Ceci génère un nombre aléatoire uniformément distribué sur [0;1]
   if (nbr <= p){
     res <- 1 #Si le nombre tiré est plus petit ou égal à p alors res prend 1
   } else {
     res <- 0 #Sinon res prend 0
   }
   return(res) #On retourne le res
}

testber <- bernoulli(0.7)
print(testber)
```

Ici, la probabilité du tirage de Bernoulli est de 0.7, probabilité choisie en paramètre, et le programme simule un tirage. Celui-ci retourne 1 en cas de réussite si le chiffre généré aléatoirement par la fonction runif() est plus petit ou égal à notre p, 0 sinon.

Ensuite, nous avons simulé une loi binomiale de paramètres n et p.

```r
binomial <- function(n, p){ #Fonction pour simuler binomial
  nbr<- runif(n) # Va générer n nombre entre [0;1] et le mettre dans nbr[] qui seras un tableau dans ce cas là
  compt<-0 #Compteur

  for (i in 1:n) { # Pour i de 1 à n choisis en argument
    if (nbr[i]<=p) {
      compt<- compt +1 #On compte le nombre de fois que nbr[i] est plus petit ou égal à p
    }
  }
  return(compt) #Retourne le compteur
}

testbino<-binomial(10,0.7)
print(testbino)
```

Ici, la probabilité de succès est toujours de 0.7 mais on le fait n-fois afin de comptabiliser le nombre de réussites. On a donc un compteur qui va réaliser n-fois le tirage de Bernoulli de paramètre p et qui va rajouter +1 au compteur à chaque réussite afin d'obtenir à la fin le nombre total de réussites.

La 3ème question traitait de la simulation d'une loi géométrique de paramètre p.

```r
#3
geo<-function(p){ #Fonction pour simuler géométrique
  compt <- 1 #Compteur
  while (runif(1)>p){ #Tant que le nombre généré aléatoire est supérieur à p on continue
    compt<- compt +1
  }
  return(compt) #Retourne le compteur
}

testgeo<-geo(0.7)
print(testgeo)
```

Le paramètre p correspond ici encore à la probabilité de réussite du tirage. Une loi géométrique modélise le nombre d'essais nécessaires avant le premier succès du tirage. Ces multiples essais sont ici modélisés par la boucle while qui ne se terminera que lorsque le nombre tiré aléatoirement sera inférieur ou égal à p. À chaque itération de la boucle while, on ajoute +1 au compteur que l'on retourne en fin de fonction.

Les deux dernières lois à représenter sont des lois uniformes. Dans un premier temps sur un intervalle [| 1;k |] avec k un entier naturel $\geqslant 2$:

```r
uniforme<- function(k){ #Fonction uniforme1
  nbr<-round(runif(1, 1, k)) #On souhaite avoir une expérience entre 1 et k
  return(nbr) #On retourne le nombre obtenu
}

testuni<-uniforme(5)
print(testuni)
```

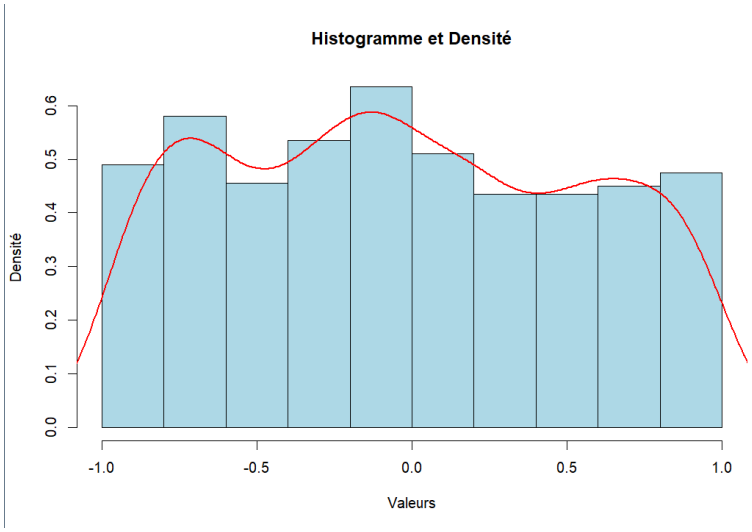
La fonction runif avec les paramètres de la capture d'écran permet de réaliser une loi uniforme avec un nombre de manière aléatoire entre 1 et k. On utilise la fonction round afin d'obtenir un arrondi et d'avoir un entier. Lorsque k = 0, la fonction runif ne s'exécute pas, et quand k = 1, on réalise sur un intervalle [| 1;1 |] donc le résultat est forcément 1.

Enfin la deuxième fonction uniforme appelée uniforme2 prend en entrée le nombre de tirages à faire k, sur un intervalle [|-1;1|].

```r
uniforme2 <- function(k) { #Fonction uniforme2
  nbr <- runif(k, -1, 1) # On veut à présent réaliser pour k fois entre -1 et 1, k pris en entré
  histo <- hist(nbr, plot = FALSE)
  densite <- density(nbr)


  # Tracer l'histogramme
  plot(histo, main = "Histogramme et Densité", col = "lightblue", xlim = c(-1, 1), freq = FALSE, xlab = "Valeurs", ylab = "Densité")

  # Tracer la densité estimée
  lines(densite, col = "red", lwd = 2)#On utilise line car on veut une courbe et lwd=2 pour la largeur

  }

# Exemple d'utilisation avec k = 1000
uniforme2(1000)
```

La fonction runif avec ces paramètres réalise k-tirages sur [|-1;1|]. L'utilisation de la fonction hist nous permet de réaliser un histogramme de nbr qui est un tableau avec les différentes valeurs de runif(k,-1,1). La fonction densité calcule la densité de nbr.

On trace ensuite sur le même graphique l'histogramme limité -1 et 1 et la densité avec lines. Voici un des résultats obtenu (puisque cela change à chaque fois que l'on exécute le programme) :

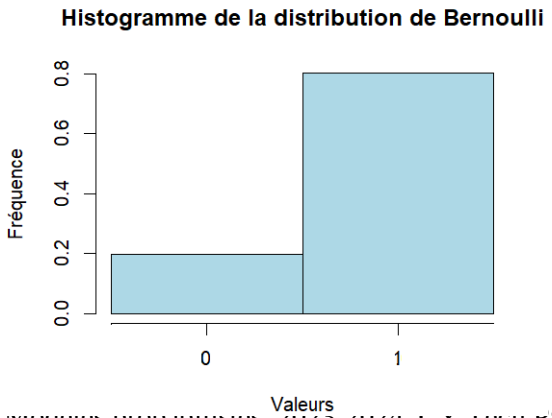


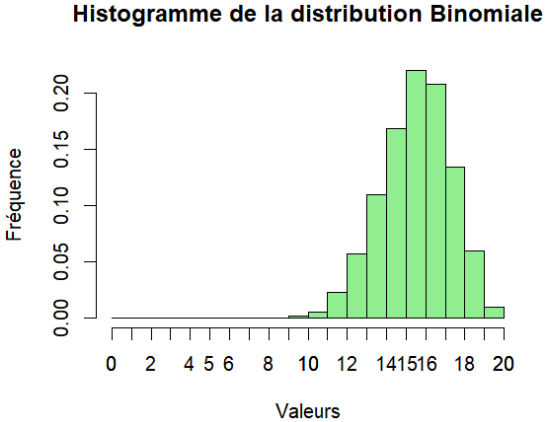
On remarque que la droite de densité suit la forme de l'histogramme ce qui est le résultat attendu par cette simulation.

En annexe, nous avons fait différents tests de modélisations pour tester nos jeux de données : Voici par exemple une modélisation pour Bernoulli et la loi Binomiale : (testé pour une probabilité p = 0,8 et n valeurs = 10 000)

```
> an_Bern(0.8, 10000)
Succès:  80.02 %
Échec:  19.98 %
```

```
> an_Bi(20, 0.8, 10000)
Fréquence:  0 0 0 0 0 0 0 0 0.01 0.02 0.23 0.69 2.34 5.54 11.41 17.18 21.74 20.1 13.94 5.73 1.07
```





Code de la distribution de Bernoulli :

```
hist(x, breaks = c(-0.5, 0.5, 1.5), col = "lightblue",
main = "Histogramme de la distribution de Bernoulli", xlab = "Valeurs", ylab = "Fréquence",
probability = TRUE)
axis(1, at = c(0, 1), labels = c(0, 1))

S <- 100 * fre / Z
E <- 100 - S
cat("Succès: ", S, "%\n")
cat("Échec: ", E, "%\n")
```

Code de la distribution Binomiale :

```
freq <- 100 * freq / Z

hist(x, breaks = seq(0, n, by = 1), col = "lightgreen",
main = "Histogramme de la distribution Binomiale", xlab = "Valeurs", ylab = "Fréquence",
probability = TRUE)
axis(1, at = li, labels = li)

cat("Fréquence: ", freq, "\n")
```

Remarque : ces tests ont été réalisés à part, car ils ne sont pas demandés dans l'exercice, ils ne sont donc pas présents dans les fichiers de code.

Exercice 2 : Simulation de variables aléatoires réelles par inversion

Cet exercice a pour but de simuler des variables aléatoires réelles grâce à la méthode d'inversion.
Commençons avec la loi de Bernoulli. Grâce à la méthode d'inversion, nous pouvons obtenir la fonction suivante :

```
simulate_bernoulli <- function(p) {
  u <- runif(1)  # Générer une variable aléatoire uniforme sur [0, 1]
  if (u < 1 - p) {
    return(0)
  } else {
    return(1)
  }
}
```

Cette fonction permet de simuler une variable aléatoire de Bernoulli et nous retourne donc une variable qui ne peut prendre comme valeur soit 1 ou 0 :

```
> cat("Variable aléatoire de Bernoulli simulée:", result, "\n")
Variable aléatoire de Bernoulli simulée: 0
```

Nous pouvons maintenant passer à la suite avec la loi géométrique. Pour obtenir la fonction de répartition inverse de cette loi, nous pouvons commencer par calculer la fonction de répartition de la loi géométrique que l'on note :

$$F(x;p) = 1 - (1-p)^x$$

L'inverse de cette fonction de répartition est obtenu en isolant x dans l'équation $F(x:p) = u$ où u est une v.a. uniforme sur l'intervalle [0;1] :

$$<=> (1-p)^x = 1-u$$
$$<=> x * log(1-p) = log(1-u)$$
$$<=> x = log(1-u)/log(1-p)$$

Ce qui nous donne bien la fonction suivante :

```r
simulate_poisson <- function(lambda) {
  u <- runif(1)  # Générer une variable aléatoire uniforme sur [0, 1)
  x <- 0
  F <- exp(-lambda)
  while (u >= F) {
    x <- x + 1
    F <- F + exp(-lambda) * (lambda^x) / factorial(x)
  }
  return(x)
}
```

Passons maintenant à la loi exponentielle où nous pouvons utiliser la même méthode pour trouver la fonction de répartition inverse :

$$F(x;p) = 1 - exp(-\lambda x)$$

Nous pouvons maintenant isoler x :

$$u = 1 - exp(-\lambda x)$$
$$1 - u = exp(-\lambda x)$$
$$log(1-u) = -\lambda x$$

$-log(1-u)/\lambda = x$

Or cela correspond bien à la fonction suivante :

```r
simulate_exponential <- function(lambda) {
  u <- runif(1)  # Générer une variable aléatoire uniforme sur [0, 1)
  x <- -log(1 - u) / lambda
  return(x)
}
```

Enfin pour la loi de Poisson, nous avons utilisé une méthode un peu différente. Nous commençons par générer une variable aléatoire uniforme u sur [0,1] et par initialiser x à 0. Ensuite, nous avons initialisé une fonction de répartition cumulative F à $exp(-\lambda)$. Grâce à cela, tant que $u >= F$, nous pouvons incrémenter x et mettre à jour F en ajoutant le terme suivant de la somme cumulative. La valeur finale de x est la v.a. de Poisson.

Nous obtenons alors la fonction suivante :

```r
simulate_poisson <- function(lambda) {
  u <- runif(1)  # Générer une variable aléatoire uniforme sur [0, 1)
  x <- 0
  F <- exp(-lambda)
  while (u >= F) {
    x <- x + 1
    F <- F + exp(-lambda) * (lambda^x) / factorial(x)
  }
  return(x)
}
```

Exercice 3: Lois des grands nombres

Cet exercice avait pour but de simuler la loi des grands nombres selon différentes variables aléatoires.

Exercice 6.4 (Lois des grands nombres). Soit $(X_n)_{n\geq 1}$ une suite de v.a.r. i.i.d. et $m = \mathrm{E}[X]$. On pose

$$S_n = \sum_{k=1}^{n} X_k.$$

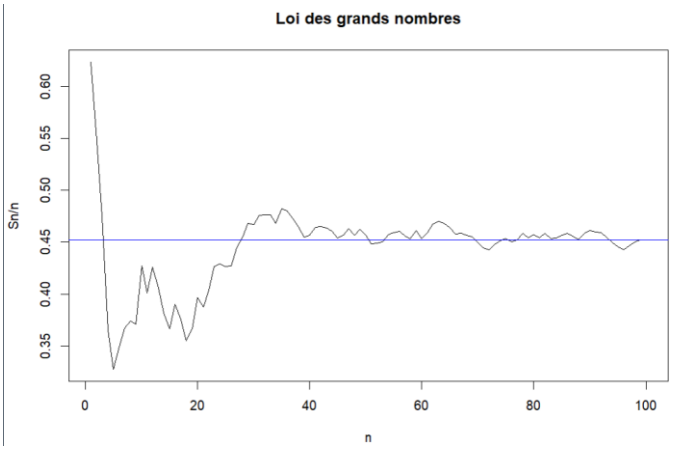
Dans un premier temps, nous devons simuler pour un grand nombre N de variables aléatoires réelles de loi uniforme sur [0;1]

```r
#EXERCICE 4
#Q1
N<-100 # Nombre de variables à sumuler
Xn<-runif(N) # Xn se présente comme un tableau dans lequel une case est un runif()
S<-cumsum(Xn) # Permet de faire une somme cumulative des Xn
m<-mean(Xn) #Fonction qui permet de calculer la moyenne de Xn
Sn_n<-S/1:N
# Tracer les courbes sur un même graphe
plot(1:N, Sn_n, type = "l", xlab = "n", ylab = "Sn/n", main = "Loi des grands nombres")
abline(h = m, col = "blue", lty = 1) #On utilise abline car on veut une ligne droite, h pour une ligne horizontale et lty=1 pour une ligne pleine
```

On trace sur le même graphique la courbe de terme général Sn_n et la courbe y=m pour m=E[x] ici représenté par mean(Xn). Avec Sn_n :

$$\frac{S_n}{n} = \frac{X_1 + \ldots + X_n}{n}$$ en fonction de n pour n variant de 1 à N.

Voici l'un des différents résultats obtenus pour N = 100 :



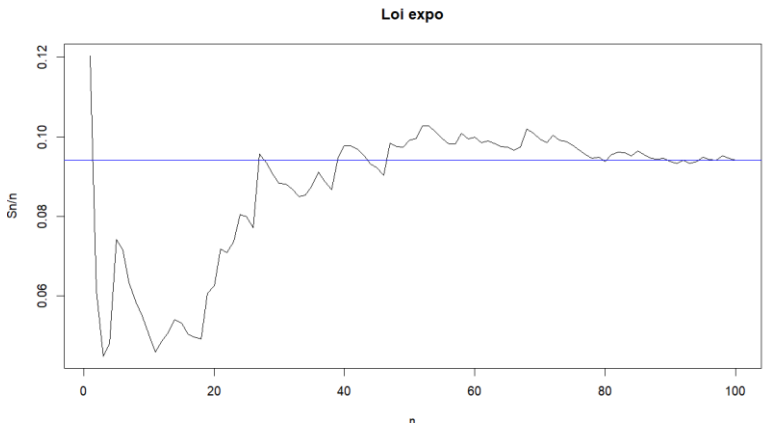
La ligne horizontale bleue correspond à y=m soit E[x]. On remarque que la suite (en gris) converge vers E[x], et c'est en effet le résultat attendu pour la loi des grands nombres.

Dans un deuxième temps, nous devons simuler une loi exponentielle pour un nombre N de variable aléatoires réelles.

```
#Q2
N<-100 # Nombre de variables à sumuler
lambda<-10
Xn<-rexp(N, lambda) #La fonction rexp(N,lambda) permet de lancer N fois une loi exponentielle de paramètre lambda
S<-cumsum(Xn)
m<-mean(Xn)
Sn_n<-S/1:N
plot(1:N, Sn_n, type = "l", xlab = "n", ylab = "Sn/n", main = "Loi expo")
abline(h = m, col = "blue", lty = 1) #On utilise abline car on veut une ligne droite, h pour une ligne horizontale et lty=1 pour une ligne pleine
```

Pour ce faire, on utilise la fonction rexp qui permet de réaliser une loi exponentielle selon N et lambda. Dans notre cas, nous avons pris un lambda = 10. De la même manière que précédemment nous avons représenté la courbe de terme général Sn_n et la courbe y=m qui correspond également à mean(Xn).

Voici l'un des différents résultats obtenus pour N = 100 et lambda = 10 :



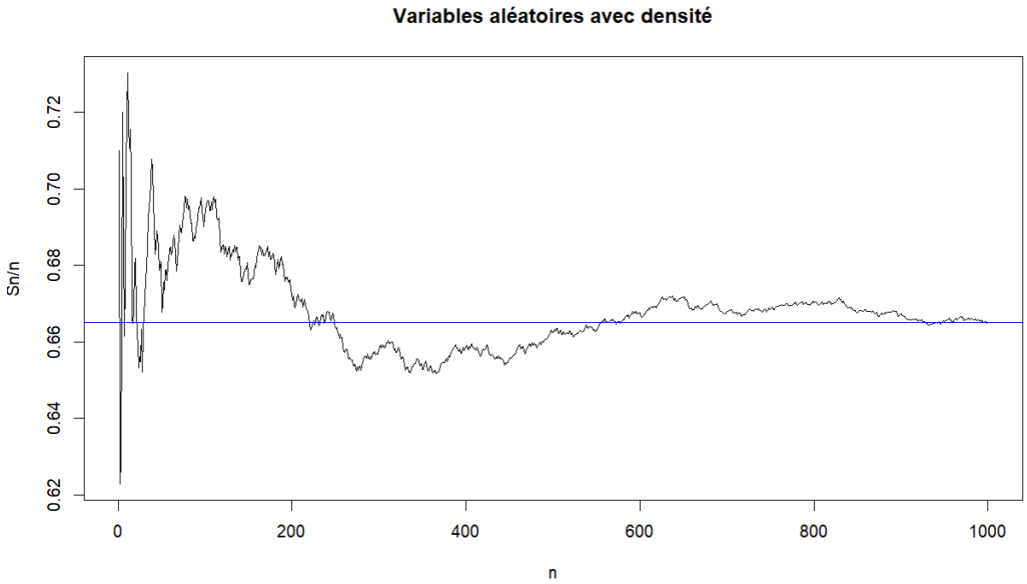
Ici également, on remarque que la courbe Sn_n en gris converge vers la courbe bleue qui correspond a E[X]. Cette convergence est encore une fois le résultat attendu d'une loi des grands nombres.

Enfin, la dernière simulation de l'exercice à réaliser était la simulation de variables aléatoires à densité qui ont pour densité : $f(x) = 2x\mathbb{1}_{[0;1]}(x)$ Afin de la simuler, on doit dans un premier temps inverser la fonction de répartition. Ici encore, on prend un grand nombre

N =1000.

```r
# Fonction de densité
f_density <- function(x) { #Fonction de densité de l'énoncé
  if (x >= 0 & x <= 1) {
    return(2 * x)
  } else {
    return(0)
  }
}
# Fonction inverse de la fonction de répartition
inverse_cdf <- function(u) {
  sqrt(u)
}
# Générer des variables aléatoires en inversant la fonction de répartition
Xn <- inverse_cdf(runif(N))
S <- cumsum(Xn)
m <- mean(Xn)
Sn_n <- S/1:N

# Tracer les courbes sur un même graphe
plot(1:N, Sn_n, type = "l", xlab = "n", ylab = "Sn/n", main = "Variables aléatoires avec densité")
abline(h = m, col = "blue", lty = 1) #On utilise abline car on veut une ligne droite, h pour une ligne horizontale et lty=1 pour une lig
```

Afin d'inverser la fonction de répartition, on utilise sqrt, suite à cela on réitère ce que l'on a fait précédemment pour obtenir ce genre de résultats :



On remarque encore une fois que les résultats obtenus convergent vers la droite correspondant a E[x]. Pour les 3 cas cela était le cas ce qui va dans le sens de la loi des grands nombres.

Exercice 4 : Théorème Central Limite

Dans cet exercice, nous simulons la convergence de différentes lois après application du théorème central limite, tout en comparant avec la loi Gaussienne grâce à un histogramme. Rappel de la densité de la loi Gaussienne :

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, \quad \forall x \in \mathbb{R}.$$

Prenons une suite de variables aléatoires qui suivent une loi uniforme, simulons la suite de terme générale :

$$Z_n = \frac{\sqrt{n}}{\sigma}(\frac{S_n}{n} - m).$$

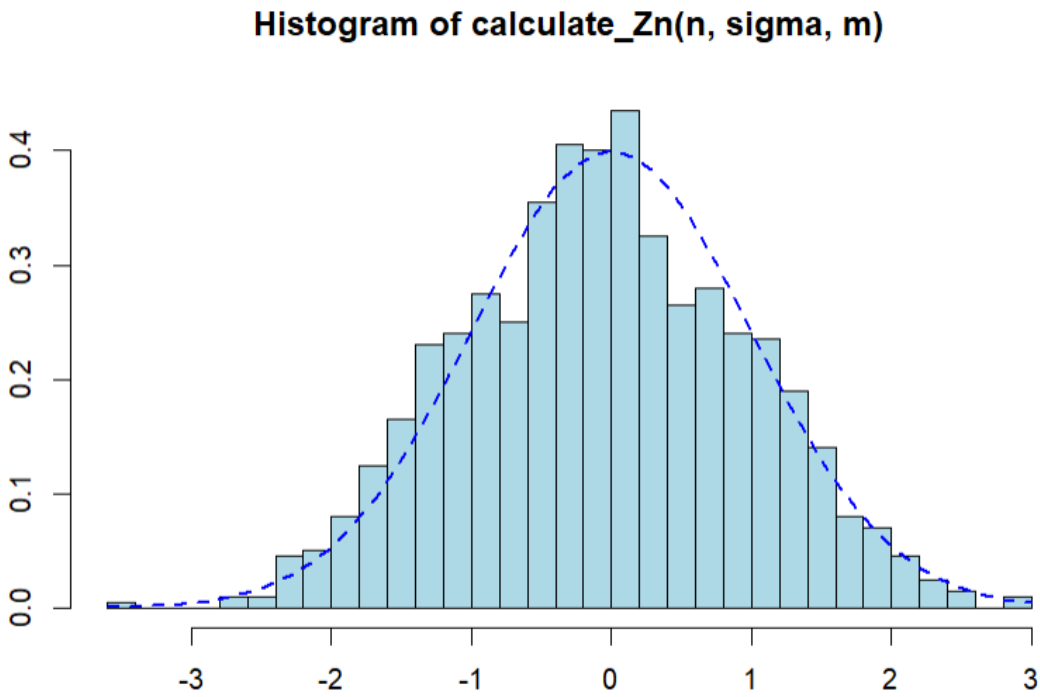
Nous pouvons donc maintenant faire la fonction qui nous permet de simuler cette suite :

```
calculate_Zn <- function(n, sigma, m) {
  for (i in 1:n) {
    Xn <- runif(n)
    Sn <- cumsum(Xn)
    Zn[i] <- (sqrt(i) / sigma) * ((Sn[i] / i) - m)
  }
  return(Zn)
}
```

avec les entrées suivantes :

```
n <- 1000     # Nombre d'itérations
Zn <- numeric(n)
sigma <- sqrt(1/12)   # Écart-type de la distribution uniforme
m <- 1/2        # Moyenne de la distribution uniforme
```

Grâce à cela nous pouvons afficher notre histogramme de notre simulation avec la loi Gaussienne pour visualiser la convergence en loi :



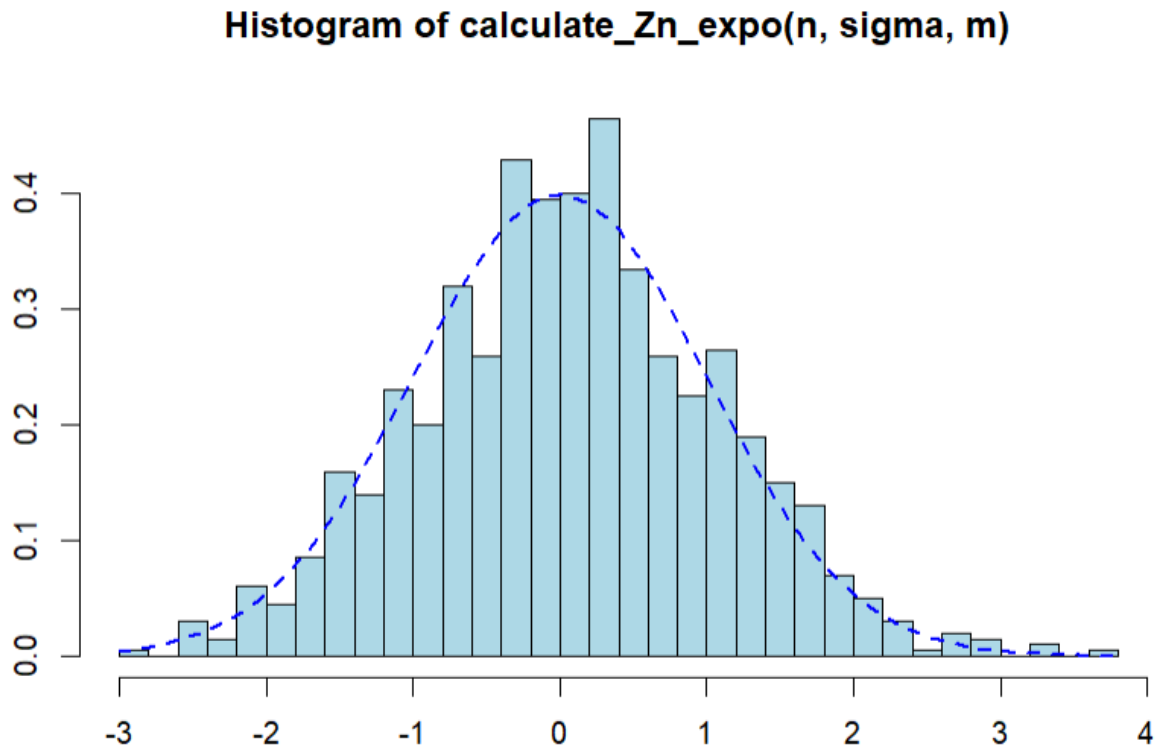
Pour la question 2, nous pouvons faire la même chose mais avec des variables aléatoires de loi exponentielle, nous pouvons alors modifier la fonction en conséquence :

```r
calculate_Zn_expo <- function(n, sigma, m) {
  for (i in 1:n) {
    Xn<-rexp(n, lambda)
    Sn <- cumsum(Xn)
    Zn[i] <- (sqrt(i) / sigma) * ((Sn[i] / i) - m)
  }
  return(Zn)
}
```

L'écart type (sigma) et la moyenne de distribution (m) change aussi :

```r
n <- 1000    # Nombre d'iterations
Zn <- numeric(n)
sigma <- sqrt(1/lambda^2)   # Écart-type de la distribution uniforme
m <- 1/lambda       # Moyenne de la distribution uniforme
```

En visualisant la convergence vers la loi gaussienne nous obtenons alors un résultat cohérent :



Enfin pour la troisième question nous devons prendre des variables aléatoires à densité, ayant pour densité la fonction : $f(x) = 2x1[0;1](x)$ et en inversant la fonction de répartition

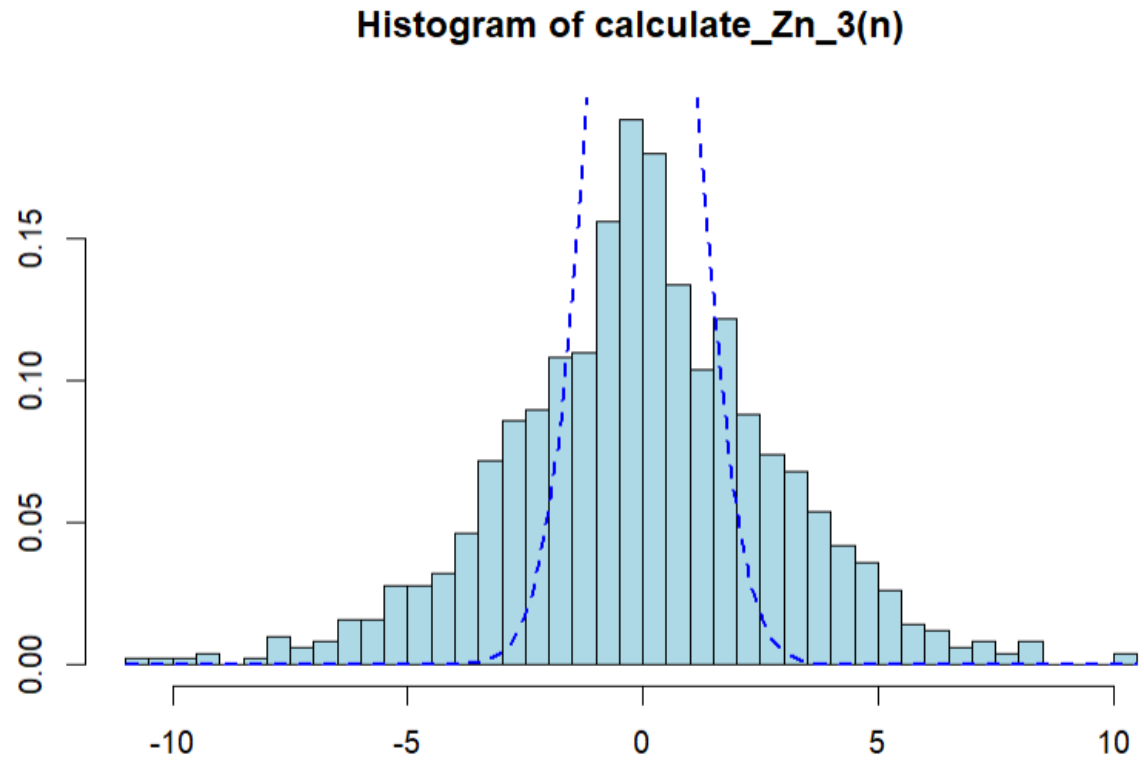
La fonction de répartition pour $f(x)$ est : $F(x) = x^2$

Donc la fonction de répartition inverse est $F(u) = sqrt(u)$

Cela nous permet donc d'obtenir la fonction suivante :

```r
calculate_Zn_3 <- function(n) {
  for (i in 1:n) {
    val <- runif(n)
    Xn <- sqrt(val)
    m <- mean(Xn) # Calcul de l'écart type
    sigma <- mean(Xn^2)-m^2 #Calcul de la moyenne de distribution
    Sn <- cumsum(Xn)
    Zn[i] <- (sqrt(i) / sigma) * ((Sn[i] / i) - m)
  }
  return(Zn)
}
```

Ainsi que l'histogramme suivant :



Exercice 5 : Box-Muller

Dans cet exercice, nous avons simulé une loi Normale N(0,1) avec la méthode de Box-Muller en la comparant avec les résultats de l'exercice 4 :

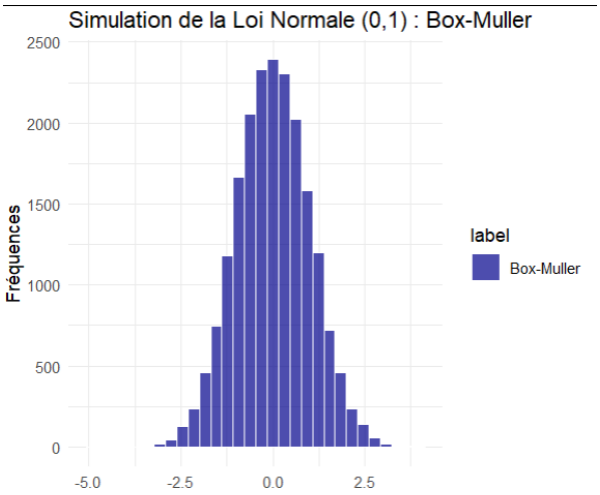
D’abord, observons la simulation de Box-Muller seule, dont les valeurs sont générées par la fonction suivante :

```r
# On utilise la formule pour Box-Muller donnée dans l'exercice
for (i in 1:n) {
  U <- runif(1)
  V <- runif(1)
  R <- sqrt(-2 * log(U))
  theta <- 2 * pi * V
  X[i] <- R * cos(theta)
  Y[i] <- R * sin(theta)
}
# Pour chaque itération, on génère une valeur aléatoire de u et v
```

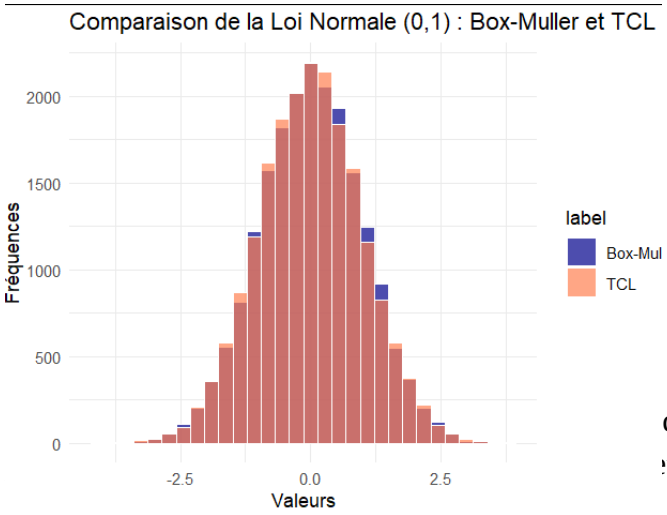
Dont voici le code de l’histogramme :

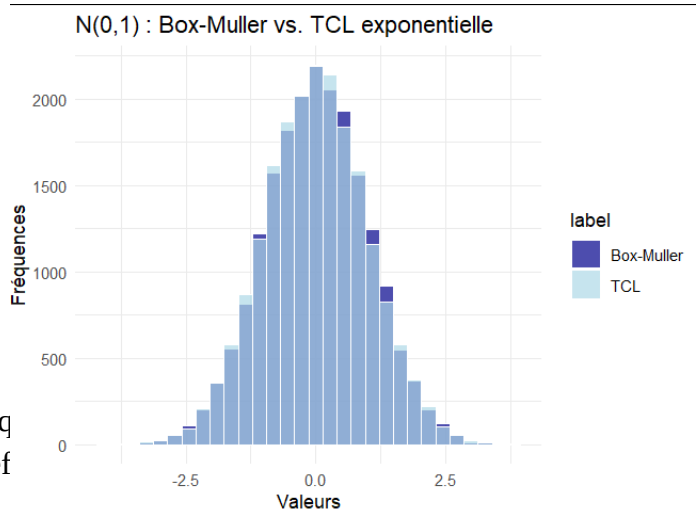
```r
# Tracé de l'histogramme, les fréquences en ordonnées, les valeurs en abscisses
# bin est le nombre de bandes choisies (= la précision, on peut augmenter son nombre pour
# diluer les fréquences des valeurs)
# ou diminuer son nombre pour avoir une idée où sont centrées les valeurs
ggplot(df, aes(x = samples, fill = label)) +
  geom_histogram(position = "identity", alpha = 0.7, bins = 20, color = "white") + theme_minimal()
  labs(title = "Simulation de la Loi Normale (0,1) : Box-Muller",
       x = "Valeurs",
       y = "Fréquences") +
  scale_fill_manual(values = c("Box-Muller" = "darkblue")) +
  theme(legend.position = "right")
```

On obtient ainsi le graphique suivant,
Pour un jeu de données n = 10 000 :



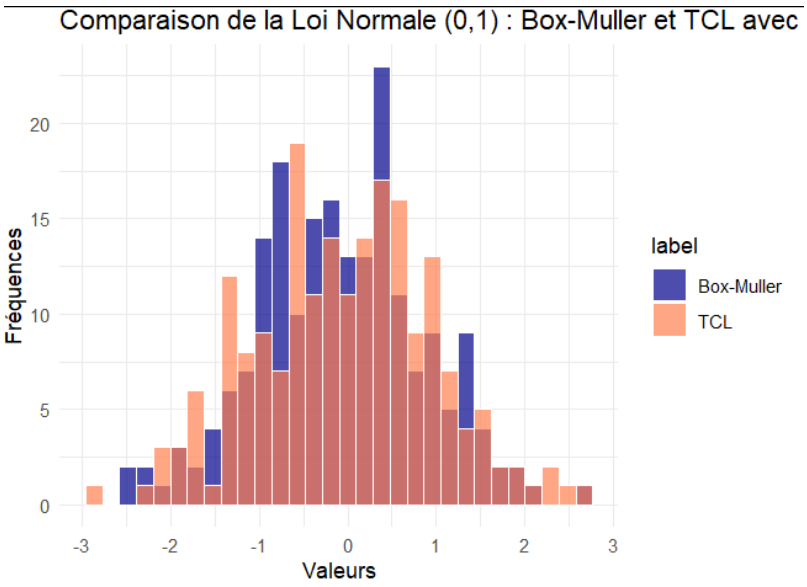
Ensuite, en comparant avec les méthodes de l’exercice 4 nous obtenons les résultats suivants :





on ne distingue pas de différence majeure entre les 2 méthodes les fréquences sont proches les unes des autres.

A noter que nous avons utilisé un jeu de 10 000 valeurs, ce qui lisse le graphique obtenu à la fin, pour 100 valeurs on a par exemple des différences plus flagrantes dues aux faibles nombre de répétitions, on voit donc l'impact de la loi des grands nombres.



Comparaison pour n = 100.

On peut donc jouer avec l'échantillon pour masquer les différences entre la méthode de Box-Muller et les autres méthodes. Sur des gros échantillons (n > 10 000) la différence de méthodes est alors quasi-invisible.

Pour obtenir ces résultats, nous avons combiné les graphiques en utilisant la fonction "rbind" permettant de regrouper 2 data-frames (ici, 2 méthodes à chaque fois pour chaque test)

```r
#On combine les 2 jeux de données pour pouvoir comparer sur le graph
df_combined <- rbind(df_boxmuller, df_tcl)


#Tracé de l'histogramme, les fréquences en ordonnées, les valeurs en abscisses
# bin est le nombre de bandes choisies (= la précision, on peut augmenter son nombre pour dilluer les fréquences
#ou diminuer son nombre pour avoir une idée de où sont centrées les valeurs

# Tracé de l'histogramme avec ggplot2
ggplot(df_combined, aes(x = samples, fill = label)) +
  geom_histogram(position = "identity", alpha = 0.7, bins = 30, color = "white") + theme_minimal() +
  labs(title = "Comparaison de la Loi Normale (0,1) : Box-Muller et TCL avec rnorm",
       x = "Valeurs",
       y = "Fréquences") +
  scale_fill_manual(values = c("Box-Muller" = "darkblue", "TCL" = "coral")) +
  theme(legend.position = "right")
```

En résumé, la méthode de Box-Muller est très proche en termes de résultat des méthodes vues précédemment, ce qui nous permet de valider sa pertinence, dans le cadre d'une simulation de loi normale.

V-Références

Hassan Maatouk PROBABILITES ET SIMULATION -ING1GI- ***Cy-Tech 2023-2024 ;***

Function reference - ggplot 2 : https://ggplot2.tidyverse.org/

"math et al" : Modélisation R Studio https://www.youtube.com/@mathetal/

Hartmann, K., Krois, J., Rudolph, A. (2023): Statistics and Geodata Analysis using R (SOGA-R). Department of Earth Sciences, Freie Universitaet Berlin
https://www.geo.fu-berlin.de/en/v/soga-r/Basics-of-statistics/Continous-Random-Variables/Students-t-Distribution/Students-t-Distribution-in-R/index.html

Documentation en ligne r studio