TP3 Statistique

February 1, 2024

0.1 Travaux pratiques 3 Statistique

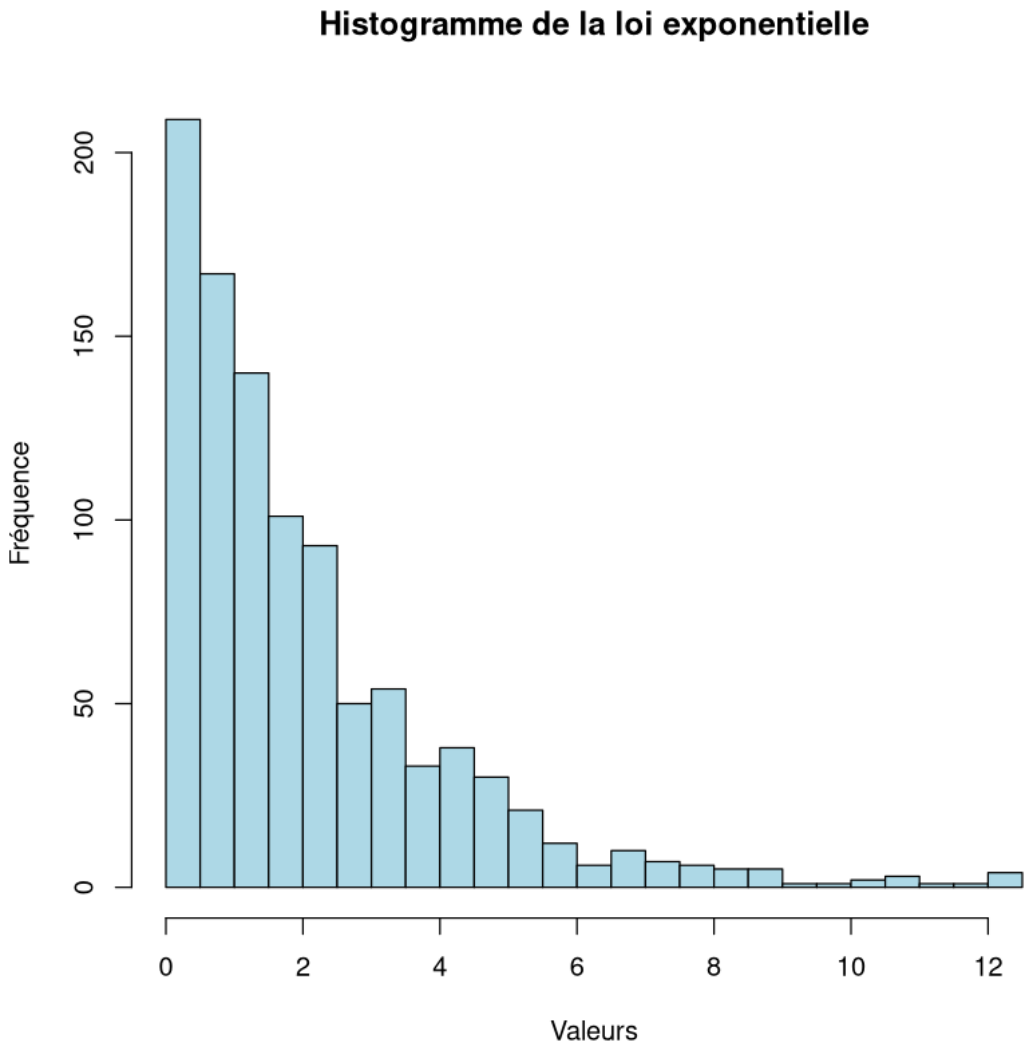
0.2 Loi exponentielle

1°)

[2]:
```r
# Fonction pour générer des échantillons aléatoires selon la loi exponentielle␣
↪Exp( )
generate_exponential <- function(n, lambda) {
  # Générer des échantillons aléatoires selon une distribution uniforme sur [0,␣
↪1]
  u <- runif(n)
  # Appliquer la méthode de l'inverse de la fonction de répartition
  x <- -log(1 - u) / lambda
  return(x)
}

# Paramètres de la loi exponentielle
lambda <- 0.5  # Paramètre lambda

# Générer 1000 échantillons aléatoires selon la loi exponentielle Exp( )
samples <- generate_exponential(1000, lambda)
```

2°)

[3]:
```r
# Afficher les échantillons aléatoires générés
hist(samples, breaks = 30, col = "lightblue", main = "Histogramme de la loi␣
↪exponentielle", xlab = "Valeurs", ylab = "Fréquence")
```



3°)

[3]:
```
# Calcul de l'intégrale en utilisant la fonction d'intégration numérique de R
integrate_result <- integrate(function(x) cos(x) * exp(-x), lower = 0, upper =␣
 ↪Inf)
estimated_value_method1 <- integrate_result$value
print(estimated_value_method1)
```

```
[1] 0.5
```

[8]:
```
# Utilisation de la méthode de Monte Carlo pour estimer l'intégrale
monte_carlo_integration <- function(n) {
  x <- rexp(n)  # Générer des échantillons aléatoires selon la loi␣
 ↪exponentielle Exp(1)
```

```r
  estimated_value_method2 <- mean(cos(x) * exp(-x))
  return(estimated_value_method2)
}

# Estimation de l'intégrale par la méthode de Monte Carlo avec 100000␣
 ↪échantillons
set.seed(123)  # Pour la reproductibilité
estimated_value_method2 <- monte_carlo_integration(100000)
print(estimated_value_method2)
```

```
[1] 0.4005153
```

0.3 Loi de Cauchy

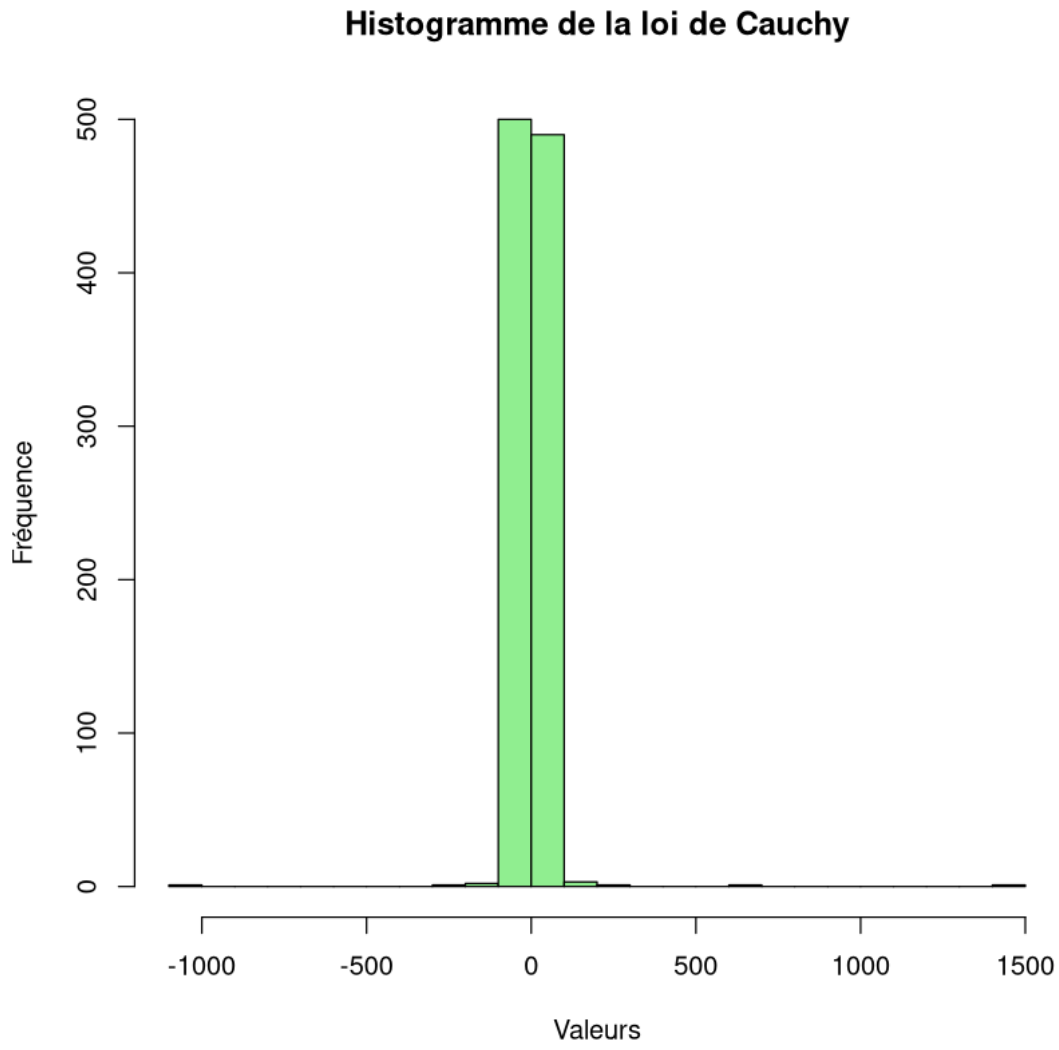
1°)

[5]:
```r
# Fonction pour générer des échantillons aléatoires selon la loi de Cauchy
generate_cauchy <- function(n, lambda) {
  # Générer des échantillons aléatoires selon une distribution uniforme sur [0,␣
 ↪1]
  u <- runif(n, 0, 1)
  # Appliquer la méthode de l'inverse de la fonction de répartition
  x <- lambda * tan(pi * (u - 0.5))
  return(x)
}

# Paramètres de la loi de Cauchy
lambda <- 1  # Paramètre lambda

# Générer 1000 échantillons aléatoires selon la loi de Cauchy
samples_cauchy <- generate_cauchy(1000, lambda)
```

2°)

[8]:
```r
# Fonction pour générer des échantillons aléatoires selon la loi de Cauchy
generate_cauchy <- function(n, lambda) {
  # Générer des échantillons aléatoires selon une distribution uniforme sur [0,␣
 ↪1]
  u <- runif(n, 0, 1)
  # Appliquer la méthode de l'inverse de la fonction de répartition
  x <- lambda * tan(pi * (u - 0.5))
  return(x)
}

# Paramètres de la loi de Cauchy
lambda <- 1  # Paramètre lambda

# Générer 1000 échantillons aléatoires selon la loi de Cauchy
```

```r
samples_cauchy <- generate_cauchy(1000, lambda)

# Afficher un histogramme des échantillons générés
hist(samples_cauchy, breaks = 30, col = "lightgreen", main = "Histogramme de la␣
 ↪loi de Cauchy", xlab = "Valeurs", ylab = "Fréquence")
```



3°)

[13]:
```r
# Calcul de l'intégrale en utilisant la fonction d'intégration numérique de R
integrate_result <- integrate(function(x) x^2/(1 + x^2), lower = -Inf, upper =␣
 ↪Inf)
estimated_value_method1 <- integrate_result$value
print(estimated_value_method1)
```

```
Error in integrate(function(x) x^2/(1 + x^2), lower = -Inf, upper = Inf): the␣
 ↪integral is probably divergent
Traceback:

1. integrate(function(x) x^2/(1 + x^2), lower = -Inf, upper = Inf)
2. stop(res$message)
```

[12]:
```r
# Utilisation de la méthode de Monte Carlo pour estimer l'intégrale
monte_carlo_integration <- function(n) {
  x <- generate_cauchy(n, 1)  # Générer des échantillons aléatoires selon la␣
 ↪loi de Cauchy avec lambda = 1
  estimated_value_method2 <- mean(x^2 / (1 + x^2))
  return(estimated_value_method2)
}

# Estimation de l'intégrale par la méthode de Monte Carlo avec 10000␣
 ↪échantillons
set.seed(123)  # Pour la reproductibilité
estimated_value_method2 <- monte_carlo_integration(10000)
print(estimated_value_method2)
```

```
[1] 0.4944481
```

0.3.1 COMMENTAIRE

L'intégration numérique directe, donne un résultat divergent pour l'intégrale de la fonction $(x^{2)/(1+x}2)$ sur l'intervalle $[-\infty,\infty]$.

Une somme renormalisée de variables i.i.d. suivant une loi de Cauchy ne converge pas vers une loi limite lorsque n tend vers l'infini. Les propriétés de convergence habituelles ne s'appliquent pas aux sommes renormalisées de variables i.i.d. suivant une loi de Cauchy, et des méthodes alternatives doivent être utilisées pour étudier le comportement asymptotique de telles sommes.

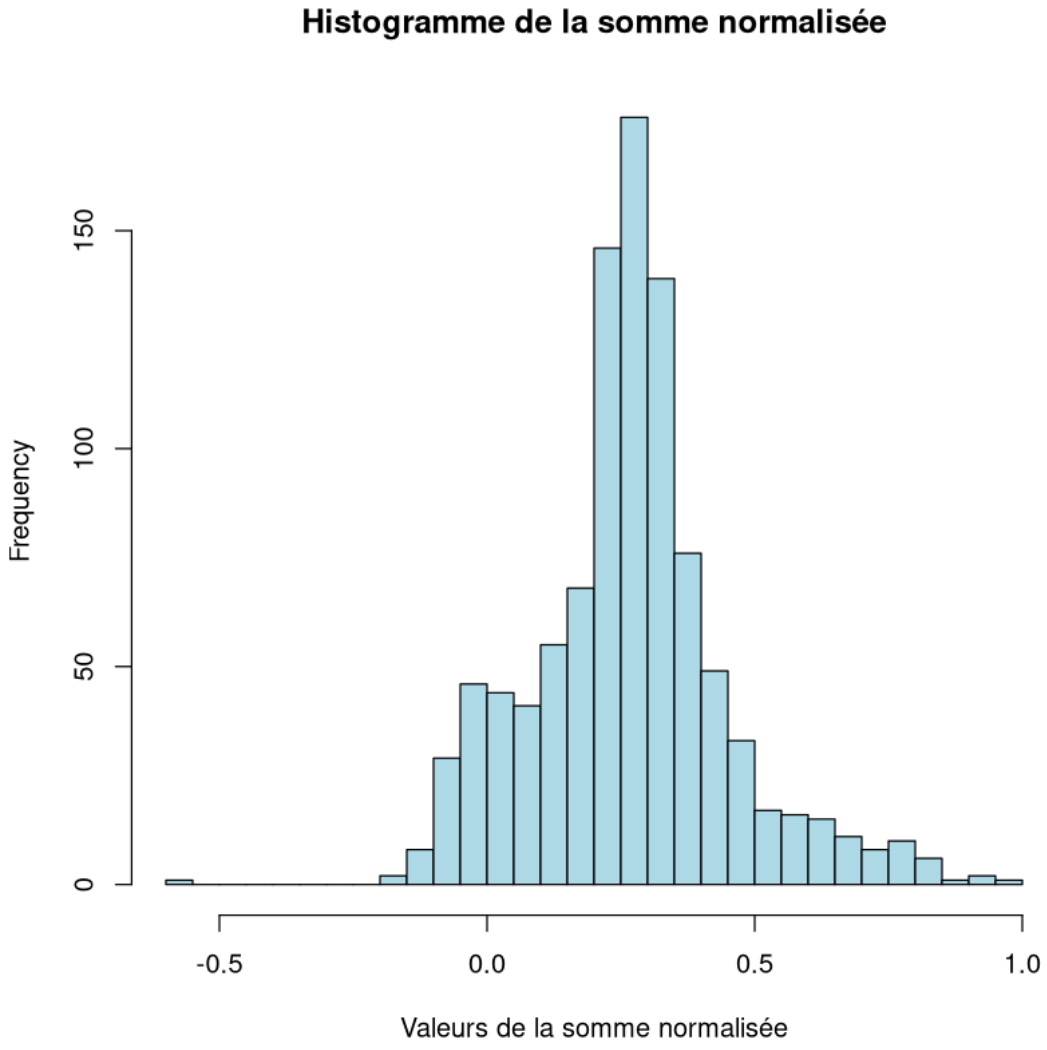
0.4 Méthode de Box-Muller

1°)

Nos deux variables convergent vers la loi normale de variance 1 et d'espérance 0.

[17]:
```r
# Nombre d'échantillons à générer
n <- 1000

# Générer les échantillons de X_k suivant une loi uniforme sur [-1, 1]
X <- runif(n, min = -1, max = 1)

# Calculer la somme cumulée de X_k
cumulative_sum <- cumsum(X)
```

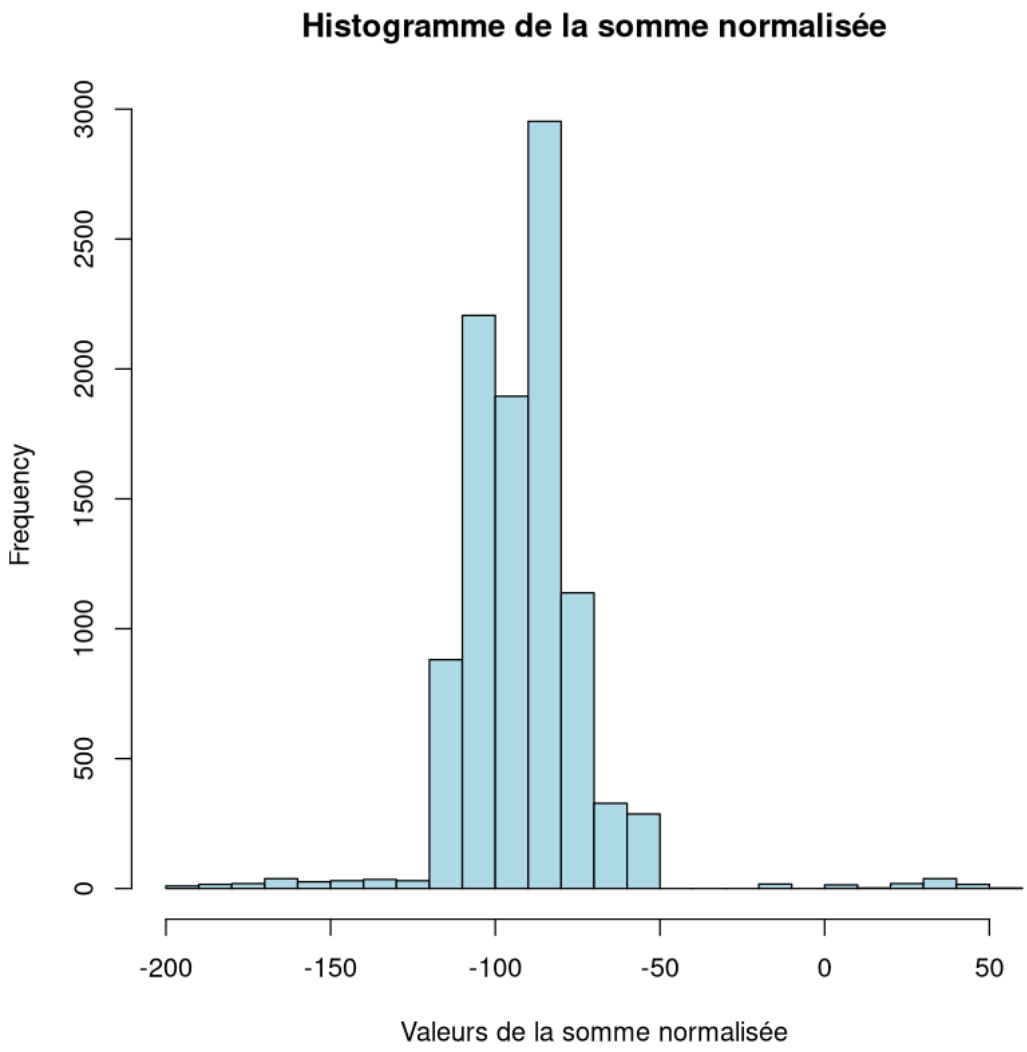
```r
# Calculer les valeurs de la variable aléatoire de la somme normalisée
normalized_sum <- cumulative_sum / sqrt(1:n)

# Créer un histogramme pour représenter les réalisations de la somme normalisée
hist(normalized_sum, breaks = 30, col = "lightblue", main = "Histogramme de la␣
 ↪somme normalisée", xlab = "Valeurs de la somme normalisée")
```



2°)

[28]:
```r
# Nombre d'échantillons à générer
n <- 10000

# Générer les échantillons de X_k suivant une loi de Cauchy avec lambda = 1
```

```r
X <- rcauchy(n, location = 0, scale = 1)

# Calculer la somme cumulée de X_k
cumulative_sum <- cumsum(X)

# Calculer les valeurs de la variable aléatoire de la somme normalisée
normalized_sum <- cumulative_sum / sqrt(1:n)

# Créer un histogramme pour représenter les réalisations de la somme normalisée
hist(normalized_sum, breaks = 30, col = "lightblue", main = "Histogramme de la␣
  ↪somme normalisée", xlab = "Valeurs de la somme normalisée")
```



0.4.1 COMMENTAIRE

Lorsque n est assez grand, le théorème central limite nous indique que notre somme converge en distribution vers une loi normale

3°)

[15]:
```r
# Nombre d'échantillons à générer
n <- 100000

# Générer des échantillons de la variable aléatoire X suivant la distribution␣
 ↪N(0,1)
X <- rnorm(n)

# Calculer la moyenne de la fonction cos(X)*exp(-X^2/2)
mean_value <- mean(cos(X) * exp(-X^2/2))

# Estimer l'intégrale de la fonction en multipliant la moyenne par la largeur␣
 ↪de l'intervalle
estimated_integral <- mean_value * sqrt(2 * pi)

# Afficher l'estimation de l'intégrale
print(estimated_integral)
```

```
[1] 1.378808
```

0.4.2 COMMENTAIRE

Oui la méthode de Monte Carlo est efficace.

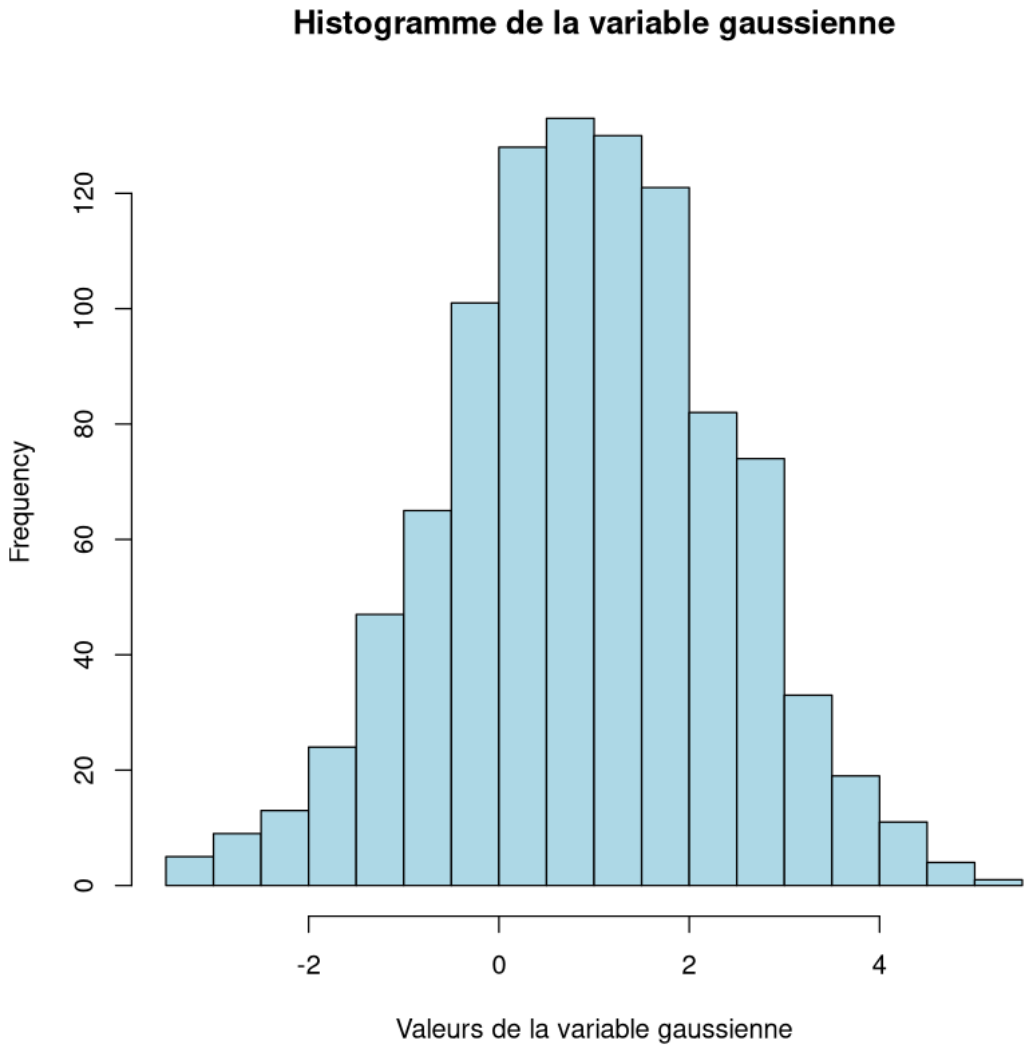
La méthode de Monte Carlo est efficace pour estimer des intégrales, en particulier dans des cas où les méthodes analytiques ou déterministes sont difficiles à appliquer. En utilisant un grand nombre d'échantillons aléatoires, la méthode de Monte Carlo permet d'obtenir des estimations précises de l'intégrale avec une erreur qui diminue à mesure que le nombre d'échantillons augmente.

4°) Sur papier

5°)

[16]:
```r
# Nombre d'échantillons à générer
n <- 1000

# Générer des échantillons de variables aléatoires uniformes U1 et U2
U1 <- runif(n)
U2 <- runif(n)

# Calculer les variables aléatoires Z0 et Z1 par la méthode de Box-Muller
Z0 <- sqrt(-2 * log(U1)) * cos(2 * pi * U2)
Z1 <- sqrt(-2 * log(U1)) * sin(2 * pi * U2)
```

```r
# Obtenir des réalisations d'une variable gaussienne de moyenne 1 et de␣
 ↪variance 2
X <- sqrt(2) * Z0 + 1

# Créer un histogramme pour illustrer les réalisations de la variable gaussienne
hist(X, breaks = 30, col = "lightblue", main = "Histogramme de la variable␣
 ↪gaussienne", xlab = "Valeurs de la variable gaussienne")
```



[ ]: