Group-2

Greeshma Jeev Koothuparambil(greko370), Sangeeth Sankunny Menon(sansa237)

2023-11-21

Question 1: Sampling algorithms for a triangle distribution

Consider the following density with a triangle-shape (another triangle distribution than considered in Lecture 3):

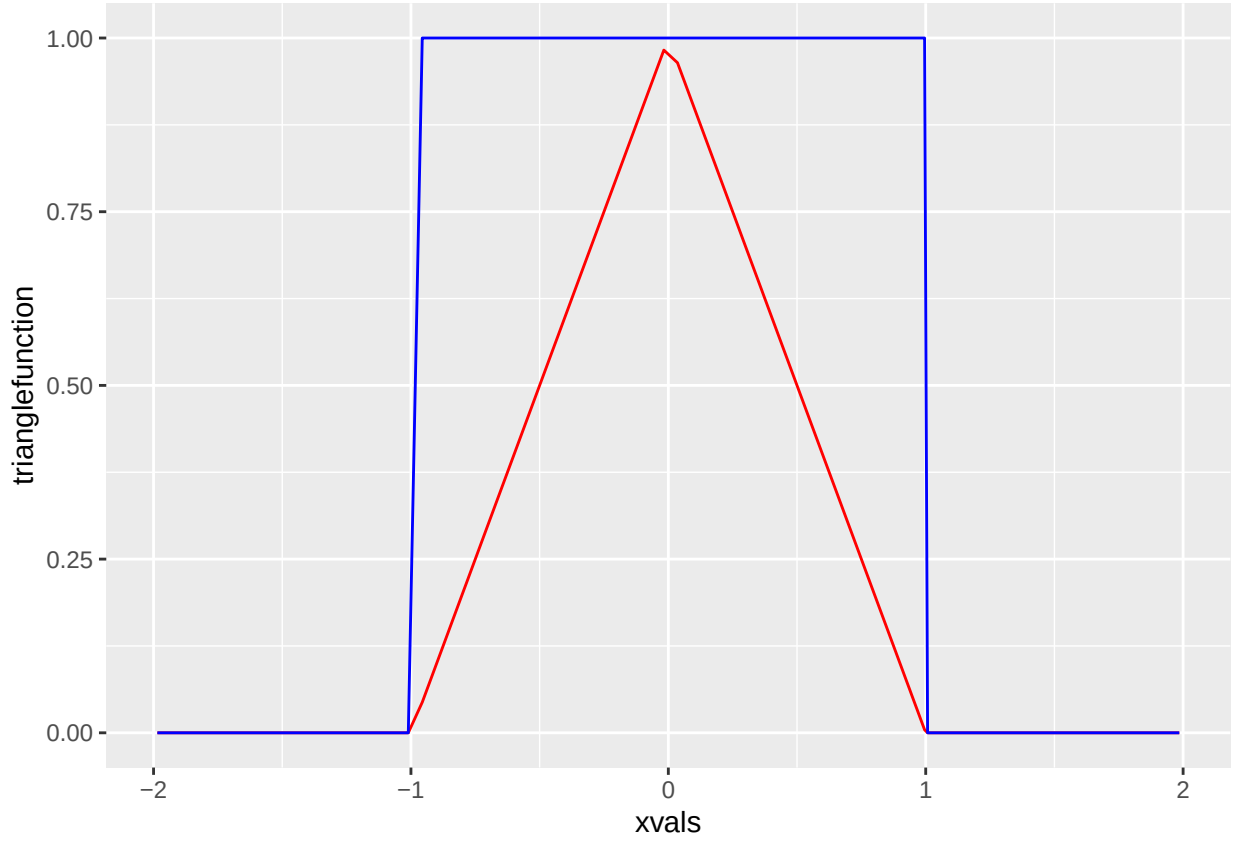
$$f(x) = \begin{cases} 0, & x < -1 \text{ or } x > 1 \\ x + 1, & -1 \leq x \leq 0 \\ 1 - x, & 0 < x \leq 1 \end{cases}$$

We are interested to generate draws of a random variable X with this density.

a. Choose an appropriate and simple envelope e(x) for the density and program a random generator for X using rejection sampling.

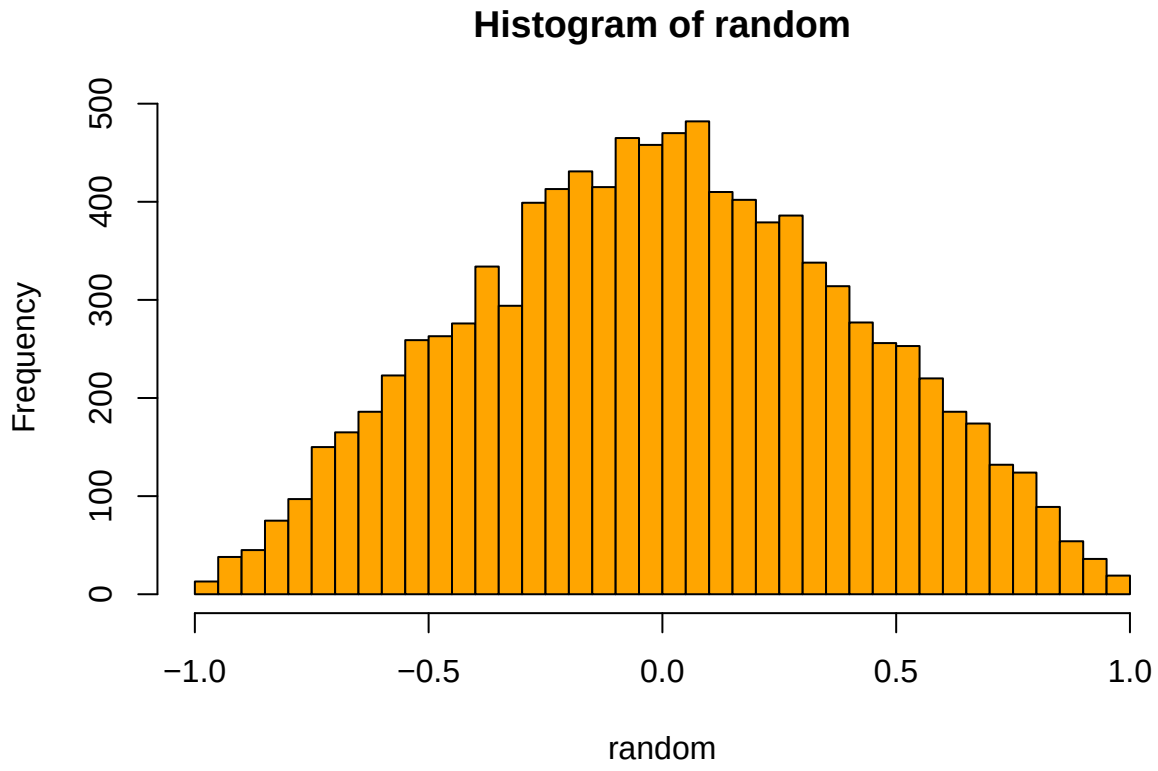
```
#1.a
library(ggplot2)
# Generating traingle function
triangle <- function(x){
  value <- 0
  if (x>= -1 && x<=0) {
    value <- x+1
  }
  if (x>0 && x<=1) {
    value <- 1-x
  }
  return(value)
}


#Taking uniform function as envelope function
envelope <- function(x){
  value <- 0
  if (x>= -1 & x<=0) {
    value <- dunif(-x, 0 , 1)
  }
  if (x>0 && x<=1) {
    value <-dunif(x, 0 , 1)
  }
  return(value)
}
```

```r
# Graphical representation of target function and envelope function
xvals <- runif(100, min = -2, max = 2)
trixvals <- c()
for (i in 1:100) {
  trixvals[i] <- triangle(xvals[i])
}

expxvals <- c()
for (i in 1:100) {
  expxvals[i] <- envelope(xvals[i])
}

fulldata <- data.frame(x= xvals,
                       trianglefunction = trixvals,
                       envelopefunction = expxvals)

ggplot(fulldata)+
  geom_line(aes(x= xvals, y = trianglefunction),color = "red")+
  geom_line(aes(x= xvals, y = envelopefunction),color = "blue")
```



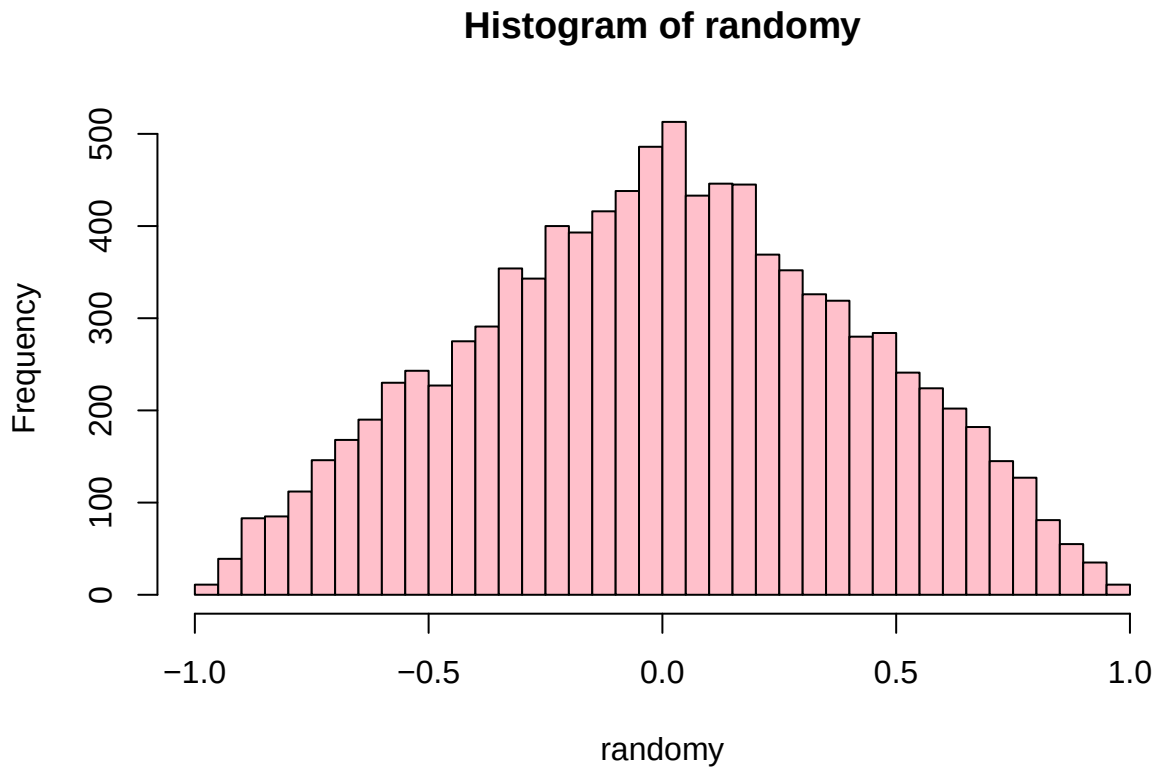

```r
#Generating random variables based on Rejection Sampling using Uniform Distribution:
randgen <- function(n){
  rands <- c()

  for (i in 1:n) {
```

```
    accept <- 0
    while (accept == 0) {
      u <- runif(1, 0,1)
      xval <- runif(1, min = -2, max = 2)
      trival <- triangle(xval)
      envval <- envelope(xval)
      if (u<= trival/(envval+1e-15)) {
        rands[i] <- xval
        accept <- 1
      }
    }
  }
  return(rands)
}

random <- randgen(10000)
#Histogram:
hist(random,col="orange", breaks = 40)
```



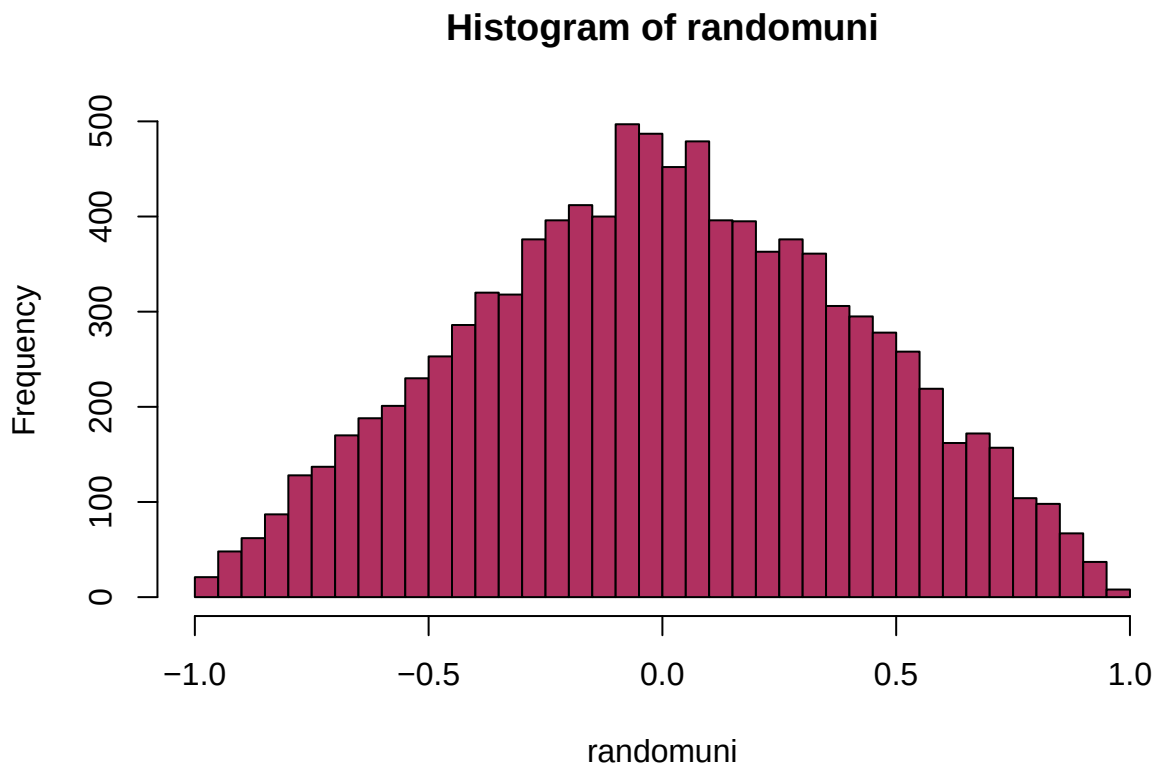
b. In Lecture 3, another triangle distribution was generated using the inverse cumulative distribution function method, see page 9-10 of the lecture notes. Let Y be a random variable following this distribution. A random variable $-Y$ has a triangle distribution on the interval $[-1, 0]$. Program a random generator for X using composition sampling based on Y and $-Y$. You can use the code from the lecture to generate Y .

```r
#1.b

#Generating random variables based on Composition Sampling using y() and -y():
randgenony <- function(n){
  rands <- c()

  for (i in 1:n) {
    u <- runif(1, -1,1)
    if (u>= -1 & u<=0) {
      rands[i] <- -(1-sqrt(1-abs(u)))
    }
    if (u>0 && u<=1) {
      rands[i] <- 1-sqrt(1-abs(u))
    }
  }
  return(rands)
}

randomy <- randgenony(10000)
#Histogram:
hist(randomy,col="pink", breaks = 40)
```



c. Sums or differences of two independent uniformly distributed variables can also have some triangle distribution. When U1,U2 are two independent Unif[0, 1] random variables, U1 −U2 has the same distribution as X. Use this result to program a generator for X.

```r
#1.c

#Generating random variables based on the difference of two Uniform Random Variables:
randgenonuni <- function(n){
  rands <- c()
  for (i in 1:n) {
    u1 <-runif(1, 0,1)
    u2 <- runif(1,0,1)
    rands[i] <- u1-u2
  }
  return(rands)
}
randomuni <- randgenonuni(10000)
#Histogram:
hist(randomuni,col="maroon" ,breaks = 40)
```



d. Check your random generators in each of a. to c. by generating 10000 random variables and plotting a histogram. Which of the three methods do you prefer if you had to generate samples of X? Use the data from one method to determine the variance of X.
Would prefer the random generator which uses difference of two uniform distribution values.
Variance of all the three models are as follows:

```r
print("Variance for random variable generated by Rejection Sampling Method :")
```

```
## [1] "Variance for random variable generated by Rejection Sampling Method :"
```

```r
var(random)
```

```
## [1] 0.1664196
```

```r
print("Variance for random variable generated by using Y and -Y functions :")
```

```
## [1] "Variance for random variable generated by using Y and -Y functions :"
```

```r
var(randomy)
```

```
## [1] 0.1702337
```

```r
print("Variance for random variable generated by U1-U2 Method :")
```

```
## [1] "Variance for random variable generated by U1-U2 Method :"
```

```r
var(randomuni)
```

```
## [1] 0.1705887
```

Question 2: Laplace distribution

The double exponential (Laplace) distribution is given by formula:

$$DE(\mu, \lambda) = \frac{\lambda}{2} exp(-\lambda|x - \mu|)$$

a. Write a code generating double exponential distribution DE(0, 1) from Unif(0, 1) by using the inverse CDF method. Explain how you obtained that code step by step. Generate 10000 random numbers from this distribution, plot the histogram and comment whether the result looks reasonable.

Before employing the inverse CDF method to generate random numbers from this distribution, the initial step involves determining the cumulative distribution function (CDF) followed by its inversion. The provided CDF is expressed as:

$$F(x; \mu, \lambda) = \begin{cases} 1/2 exp(\lambda(x - \mu)), & x < \mu \\ 1 - 1/2 exp(-\lambda(x - \mu)), & x >= \mu \end{cases}$$

Hence, the inverse CDF is given by

$$F^{-1}(u; \mu, \lambda)$$

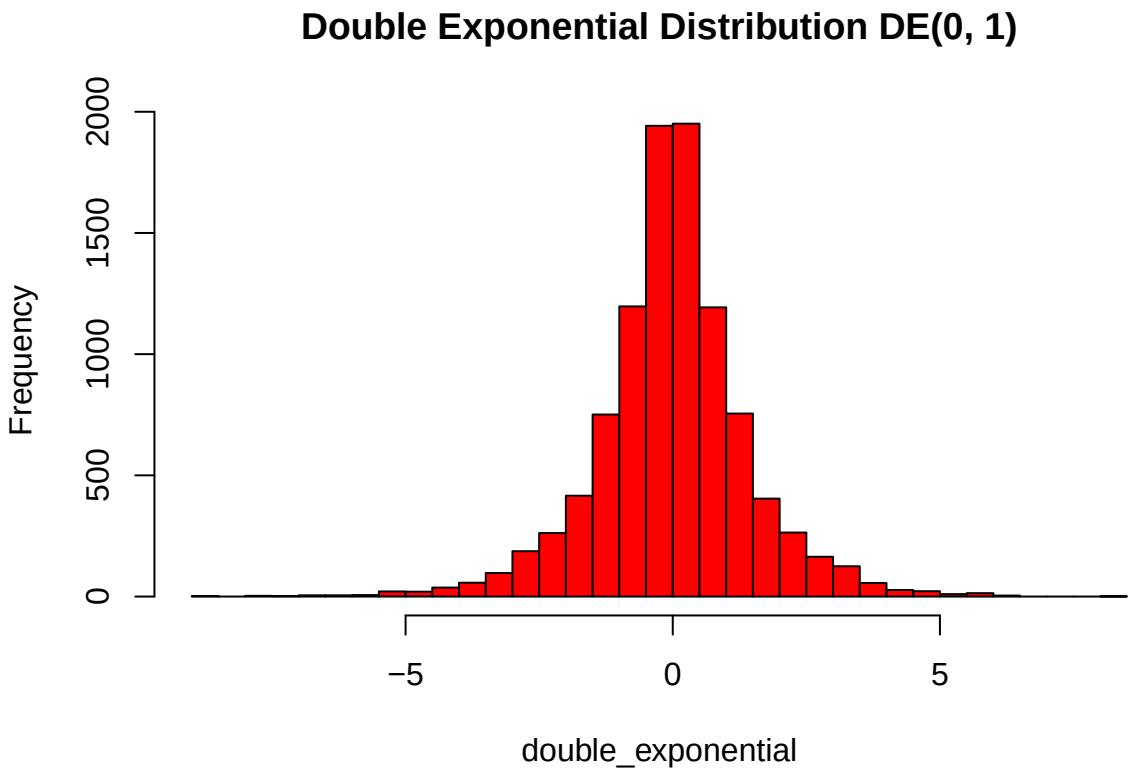
, where u is a random number from a uniform distribution between 0 and 1.

The Inverse cumulative distribution function for Laplace Distribution given $ $ = 1 and = 0

$$F^{-1}(u; \mu, \lambda) = \begin{cases} ln(2u), & u < 1/2 \\ ln(2(1 - u)), & u >= 1/2 \end{cases}$$

So the implementation is as follows in R:

```r
#2.1

#for lambda =1 and mu =0
laplace <- function(x){
  value <- 0.5* exp(-x)
  return( value)
}



inverseCDF_DE <- function(u, mu = 0, lambda = 1) {
  ifelse(u >= 0.5, -log(2*(1-u)), log(2*u))
}

uniform_random <- runif(10000)

double_exponential <- inverseCDF_DE(uniform_random)

hist(double_exponential, breaks = 30, col = "red",
     main = "Double Exponential Distribution DE(0, 1)")
```



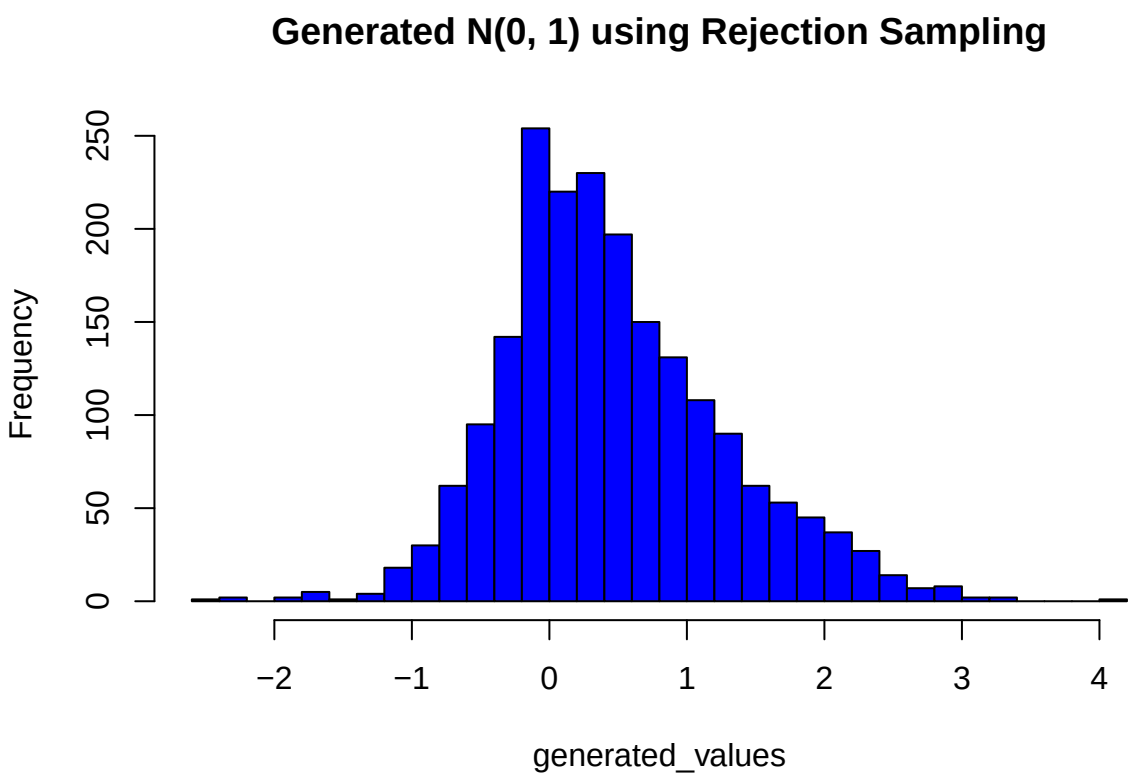
The plotted histogram looks similar to the Laplace distribution

b. Use rejection sampling with DE(0, 1) as envelope to generate N(0, 1) variables. Explain step by step how this was done. How did you choose constant a in this method? Generate 2000 random numbers N(0, 1) using your code and plot the histogram. Compute the average rejection rate R in the rejection

sampling procedure. What is the expected rejection rate ER and how close is it to R? Generate 2000 numbers from N(0, 1) using standard rnorm() procedure, plot the histogram and compare the obtained two histograms.

Find a Constant a: e(x)=g(x)/a >= f(x) Therefore, a >= f(x)/g(x)

```
#2.2



a <- sqrt(2 *exp(1)/pi)

generate_normal <- function(n) {
  result <- numeric(n)
  count_rejected <- 0

  for (i in 1:n) {
    repeat {
      x <- runif(1)
      y <- inverseCDF_DE(x)
      u <- runif(1, 0, 1)

      if (u <= dnorm(y) / (laplace(y)/a)) {
        result[i] <- y
        break
      } else {
        count_rejected <- count_rejected + 1
      }
    }
  }

  rejection_rate <- count_rejected / n
  return(list(result = result, rejection_rate = rejection_rate))
}

#set.seed(12)
generated <- generate_normal(2000)
generated_values <- generated$result
rejection_rate <- generated$rejection_rate


hist(generated_values, breaks = 30, col = "blue",
     main = "Generated N(0, 1) using Rejection Sampling")
```



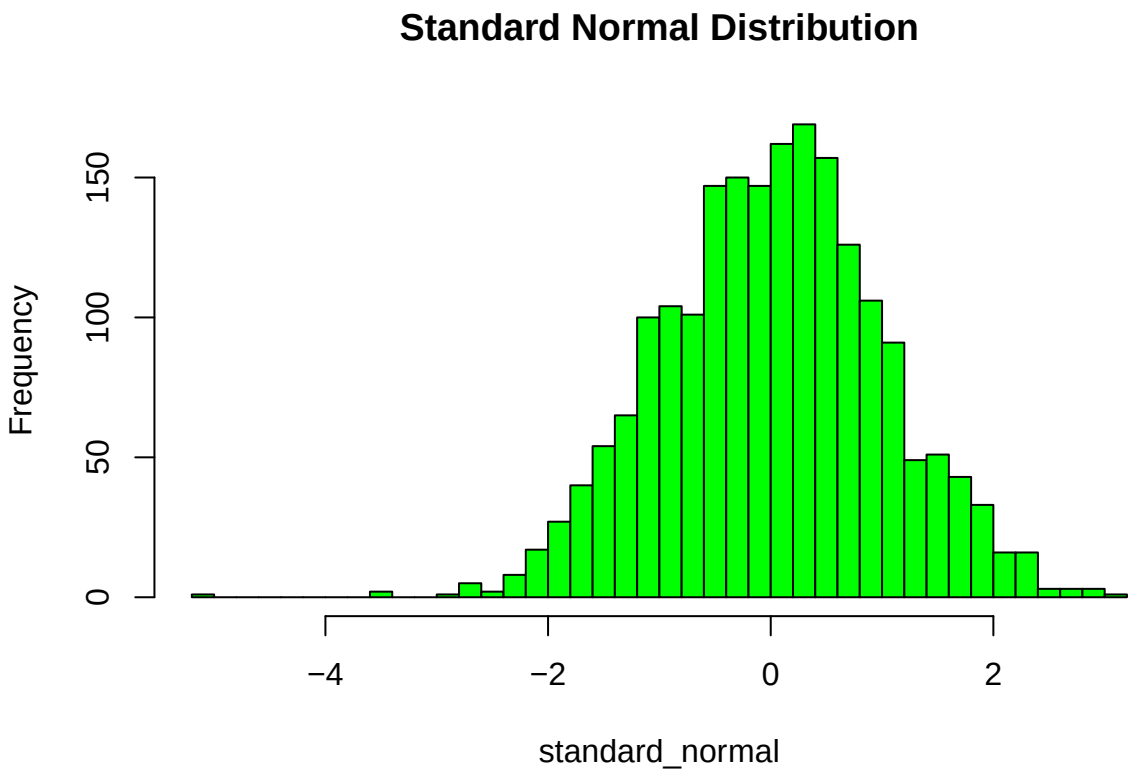

```
random2000 <- runif(2000)
invlapRN <- inverseCDF_DE(random2000)
normal2000RV <- dnorm(invlapRN)
laplace2000RV <- laplace(invlapRN)
uniform2000 <- runif(2000)
expected_rejection_rate <- 1-1/a

cat("Observed Rejection Rate (R):", rejection_rate, "\n")
```

```
## Observed Rejection Rate (R): 0.434
```

```
cat("Expected Rejection Rate (ER):", expected_rejection_rate, "\n")
```

```
## Expected Rejection Rate (ER): 0.2398265
```

```
standard_normal <- rnorm(2000)

hist(standard_normal, breaks = 30, col = "green",
     main = "Standard Normal Distribution")
```


Standard Normal Distribution
Frequency
0
50
100
150
−4
−2
0
2
standard_normal