Computational Statistics Lab 3

Simge Cinar & Ronald Yamashita

2023-11-19

Question 1:

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

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

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

```
# Define the function
triangle_function <- function(x){
  if (x < -1 || x > 1){
    return(0)
  }
  if (-1 <= x && x <= 0){
    return(x+1)
  }
  if (0 < x && x <= 1){
    return(1-x)
  }
}
```

part a)

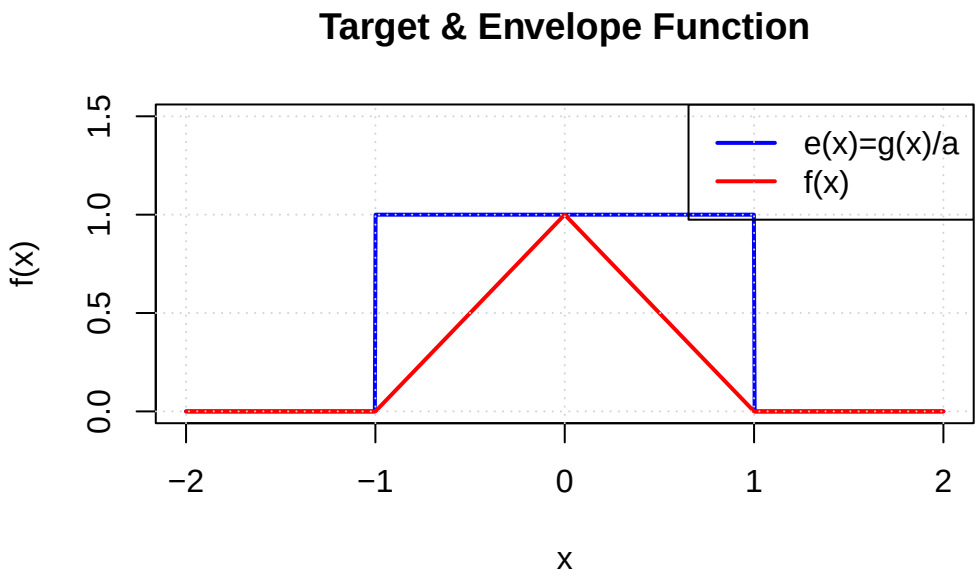
Question: Choose an appropriate and simple envelope e(x) for the density and program a random generator for X using rejection sampling.

Answer: We will use uniform distribution for g(x) in the interval [-1,1]. $f(x) = \frac{1}{b-a} = \frac{1}{1-(-1)} = 0.5$ in the given interval. The maximum value of the triangle function is 1 hence we will choose a as 0.5 to satisfy the condition below.

$$e(x) = \frac{g(x)}{a} \geq f(x), \forall x, \text{ and some a<1}$$

```
x_values <- seq(-2, 2, length.out = 1000)
a <- 0.5
pdf_uniform_values <- dunif(x_values, min = -1, max = 1) / a
pdf_triangle_values <- sapply(x_values, triangle_function)

plot(x_values, pdf_uniform_values, type = 'l', col = 'blue', lwd = 2,
     xlab = 'x', ylab = 'f(x)',
     main = 'Target & Envelope Function',
```

```
     ylim = c(0, 1.5))
lines(x_values, pdf_triangle_values, col = 'red', lwd = 2)
legend('topright', legend = c('e(x)=g(x)/a', 'f(x)'),
       col = c('blue', 'red'), lty = 1, lwd = 2)
grid()
```




```
part_a_function <- function(n){
  a <- 0.5
  count <- 0
  generated_samples <- numeric(n)
  reject <- 0

  while (count < n){
    Y <- runif(1, min = -1, max = 1) # Sample from g(x)
    U <- runif(1)
    f_y <- triangle_function(Y)
    g_y <- dunif(Y, min = -1, max = 1)
    e_y <- g_y / a

    if ((f_y/e_y) >= U){
      count <- count + 1
      generated_samples[count] <- Y
    }
    else{
      reject <- reject + 1
    }
  }
  return(generated_samples)
}
```

part b)

Question: 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.

Answer: The code can be seen below.

```
part_b_function <- function(n){
  Y <- numeric(n)
  for (i in 1:n){
    u1 <- runif(1)
    u2 <- runif(1)
    if (u2 < 0.5){
      Y[i] <- 1-sqrt(1-u1)
    } else{
      Y[i] <- -(1-sqrt(1-u1))
    }
  }
  return(Y)
}
```

part c)

Question: 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.

Answer: The code can be seen below.

```
part_c_function <- function(n){
  U1 <- runif(n)
  U2 <- runif(n)
  generated_samples <- U1 - U2
  return(generated_samples)
}
```

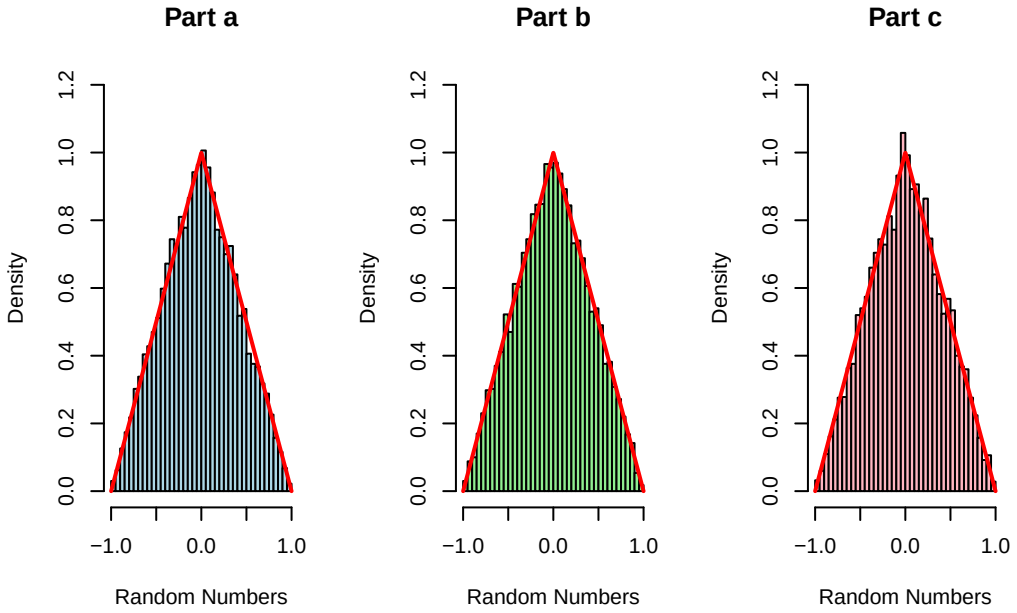
part d)

Check your random generators in each of a. to c. by generating 10,000 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.

Answer:

```
n <- 10000
samples_a <- part_a_function(n)
samples_b <- part_b_function(n)
samples_c <- part_c_function(n)
x2_values <- seq(-1, 1, length.out = 1000)
y2_values <- sapply(x2_values, triangle_function)

par(mfrow = c(1, 3))
hist(samples_a, breaks = 30, freq = FALSE, main = "Part a",
     xlab = "Random Numbers", ylim = c(0,1.2), col = "lightblue")
lines(x2_values, y2_values, col = 'red', lwd = 2)
hist(samples_b, breaks = 30, freq = FALSE, main = "Part b",
     xlab = "Random Numbers", ylim = c(0,1.2), col = "lightgreen")
lines(x2_values, y2_values, col = 'red', lwd = 2)
hist(samples_c, breaks = 30, freq = FALSE, main = "Part c",
     xlab = "Random Numbers", ylim = c(0,1.2), col = "lightpink")
```

```r
lines(x2_values, y2_values, col = 'red', lwd = 2)
```



I would choose method b. I would directly eliminate part a, rejection sampling method, because it is computationally inefficient. The expected rejection rate is 0.5. Comparing part b and c, part b offers more flexibility. In terms of shape, they all yield the similar results as it can be observed above.

```r
cat("Variance of part a:", var(samples_a), "\n")
```

```
## Variance of part a: 0.1652027
```

```r
cat("Variance of part b:", var(samples_b), "\n")
```

```
## Variance of part b: 0.1646258
```

```r
cat("Variance of part c:", var(samples_c), "\n")
```

```
## Variance of part c: 0.1641993
```

Question 2:

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

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

part 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 10,000 random numbers from this distribution, plot the histogram and comment whether the result looks reasonable.

Answer: The cumulative distribution function is as follows

$$\int_{-\infty}^{x} f(t)\,dt = F(x) = \begin{cases} \frac{1}{2} exp(\lambda(x-\mu)), & \text{if } x < \mu \\ 1 - \frac{1}{2} exp(-\lambda(x-\mu)), & \text{if } x \geq \mu \end{cases}$$

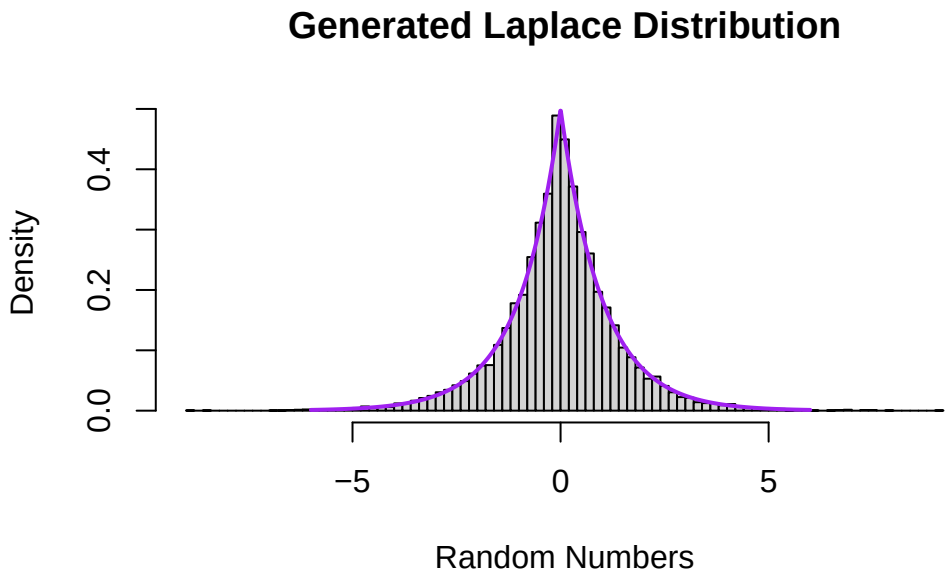
$$F(x|\lambda = 1, \mu = 0) = \begin{cases} \frac{1}{2} e^{x}, & \text{if } x < \mu \\ 1 - \frac{1}{2} e^{-x}, & \text{if } x \geq \mu \end{cases}$$

The inverse function with $\lambda = 1$ and $\mu = 0$

$$F^{-1}(u) = \begin{cases} ln(2u), & \text{if } u < 0.5 \\ ln(2-2u), & \text{if } u \geq 0.5 \end{cases}$$

```
laplace_pdf <- function(x, mu = 0, lambda = 1){
  return((lambda/2) * exp(-lambda*abs(x - mu)))
}

CDF_laplace <- function(x, mu = 0, lambda = 1){
  cdf <- ifelse(x < mu, (0.5)*exp(lambda*(x-mu)), 1-0.5*exp(-lambda*(x-mu)) )
  return(cdf)
}

inverse_CDF_laplace <- function(u){
  result <- ifelse(u < 0.5, log(2*u), -(log(2-2*u)))
  return(result)
}
```

```
set.seed(123)
U <- runif(10000)
# Use the inverse CDF to get the random variable
X <- inverse_CDF_laplace(U)
hist(X, breaks = 100, freq = FALSE, main = "Generated Laplace Distribution",
     xlab = "Random Numbers")

x_laplace_values <- seq(-6, 6, length.out = 1000)
y_laplace_values <- sapply(x_laplace_values, laplace_pdf)
lines(x_laplace_values, y_laplace_values, col = 'purple', lwd = 2)
```



The distribution above looks reasonable, it fits the actual distribution which is drawn with purple line

part 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 2,000 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 2,000 numbers from N(0,1) using standard rnorm() procedure, plot the histogram and compare the obtained two histograms.

Answer: First standard normal density function is defined

```
normal_pdf <- function(x, mean = 0, sd = 1) {
  coefficient <- 1 / (sqrt(2 * pi) * sd)
  exponent <- -((x - mean) ^ 2) / (2 * sd^2)
  density <- coefficient * exp(exponent)
  return(density)
}
```

We have the inequality below where g(x) is DE(0,1) and f(x) is N(0,1). To find the minimum a value we will minimize the left side of the inequality

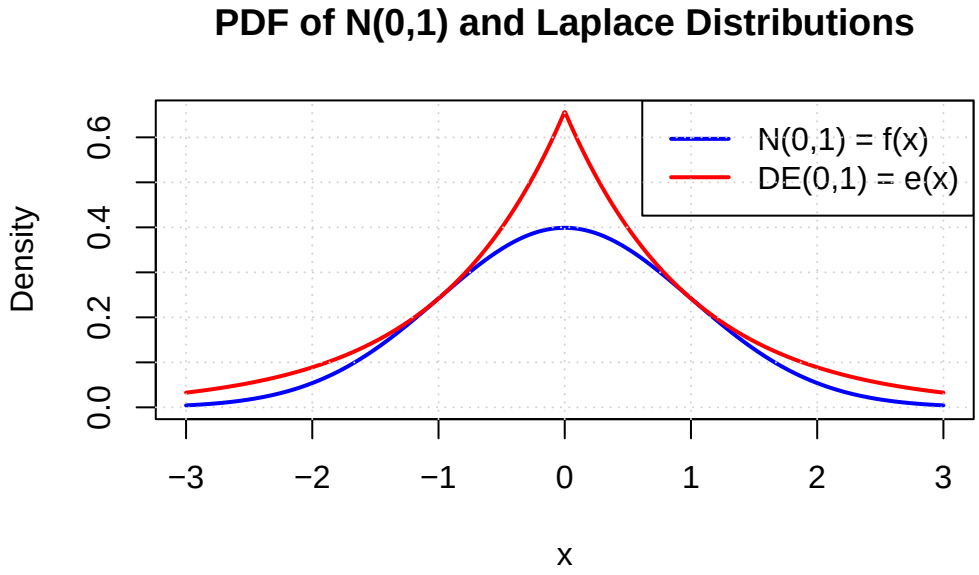
$$\frac{g(x)}{f(x)} \geq a, \forall x, \text{ and some a<1}$$

```
# Define a new function to optimize
optimize_fnc <- function(x){
  g_x <- laplace_pdf(x)
  f_x <- normal_pdf(x)
  to_optimize <- g_x / f_x
  return(to_optimize)
}

# Use optim() function to find minimum a
init_x <- c(-1)
a <- optim(init_x, fn = optimize_fnc, method = "BFGS")$value
cat("a = ", a)
```

```
## a =  0.7601735
```

The minimum a values is 0.7601735. The plot of e(x) and f(x) and can be seen below. The envelope function is above the target function in the given interval and rejection region is minimized choosing the smallest possible a as 0.7601735.

```
x_values <- seq(-3, 3, length.out = 1000)

# Calculate PDF values for Laplace and normal distributions at x_values
pdf_laplace_values <- laplace_pdf(x_values) / a
pdf_normal_values <- normal_pdf(x_values)

plot(x_values, pdf_normal_values, type = 'l', col = 'blue', lwd = 2,
     xlab = 'x', ylab = 'Density',
     main = 'PDF of N(0,1) and Laplace Distributions',
     ylim = c(0, max(pdf_normal_values, pdf_laplace_values)))
lines(x_values, pdf_laplace_values, col = 'red', lwd = 2)
legend('topright', legend = c('N(0,1) = f(x)', 'DE(0,1) = e(x)'),
       col = c('blue', 'red'), lty = 1, lwd = 2)
grid()
```




```
# Generate samples from normal distribution using laplace distribution
n <- 2000
count <- 0
generated_samples <- numeric(n)
reject <- 0

while (count < n){
  U1 <- runif(1)
  U2 <- runif(1)
  Y <- inverse_CDF_laplace(U1) #generate number from laplace dist using inverse method
  f_y <- dnorm(Y) # target function
  e_y <- laplace_pdf(Y)/a # envelope function
  if ((f_y/e_y) >= U2){
    count <- count + 1
    generated_samples[count] <- Y
  }
  else{
    reject <- reject + 1
  }
}

cat("Average Rejection Rate:", reject /(reject + count), "\n")
```

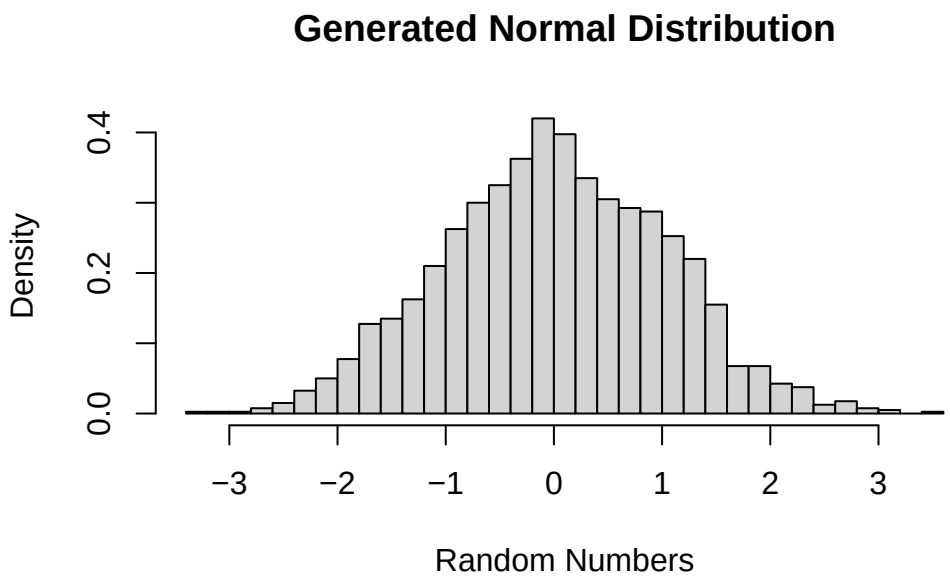
```
## Average Rejection Rate: 0.2369325
```

```
ER <- ((1/a) - 1) / (1/a)
cat("Expected Rejection Rate ER:", ER, "\n")
```

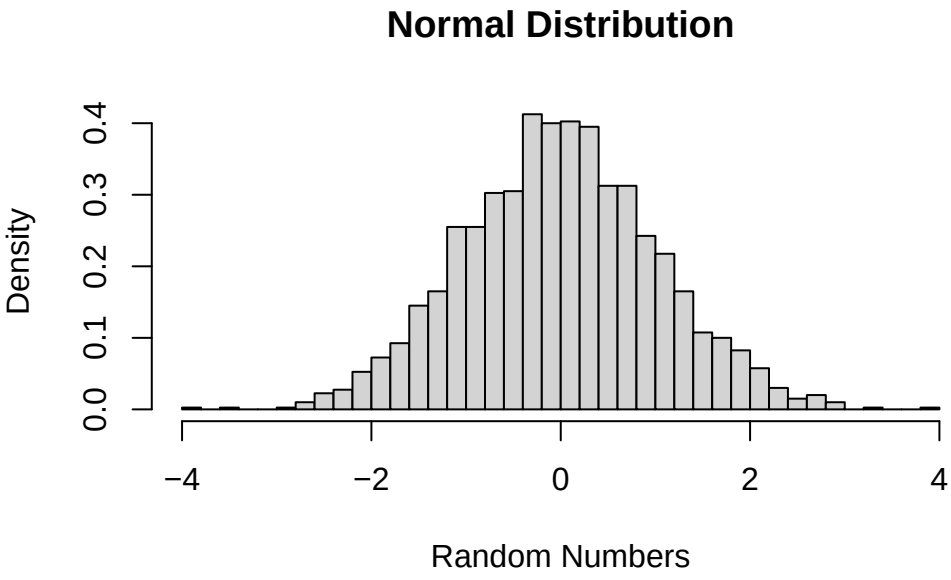
```
## Expected Rejection Rate ER: 0.2398265
```

Expected rejection rate is the area between red and blue line. We know that the area under the density function is 1 hence the area under the blue line is 1. The red line is laplace density function but it's divided by a and since a<1 the area is more than 1. We can calculate this red area dividing 1 by a which is 1.315489. To find the area between red and blue line, we subtract 1 (area under the blue line) from the area under the red line. It is calculated as 0.315489. Expected rejection rate $= \frac{(1/a)-1}{(1/a)} = \frac{0.315489}{1.315489} = 0.2398265$ is the expected rejection. Average rejection rate is 0.2369325. Average rejection rate is close the expected rejection rate.

```r
hist(generated_samples, breaks = 30, freq = FALSE,
     main = "Generated Normal Distribution", xlab = "Random Numbers")
```

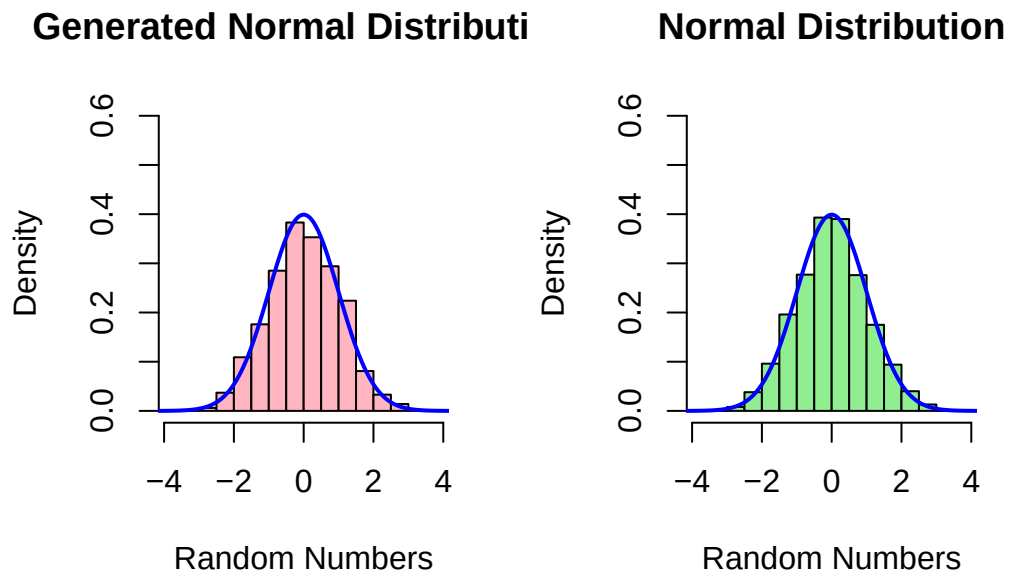



```r
actual_sample <- rnorm(n)
hist(actual_sample, breaks = 30, freq = FALSE, main = "Normal Distribution",
     xlab = "Random Numbers")
```




```r
par(mfrow = c(1, 2))
x_normal_values <- seq(-5, 5, length.out = 1000)
y_normal_values <- sapply(x_normal_values, normal_pdf)

hist(generated_samples, breaks = 20, freq = FALSE, main = "Generated Normal Distribution",
     xlab = "Random Numbers", xlim = range(c(generated_samples, actual_sample)),
     ylim = c(0, 0.6),col = "lightpink")
lines(x_normal_values, y_normal_values, col = 'blue', lwd = 2)

hist(actual_sample, breaks = 20, freq = FALSE, main = "Normal Distribution",
     xlab = "Random Numbers", xlim = range(c(generated_samples, actual_sample)),
```

```
     ylim = c(0, 0.6), col = "lightgreen")
lines(x_normal_values, y_normal_values, col = 'blue', lwd = 2)
```



The histograms can be seen above. The generated sample using rejection sampling method is similar to generate samples using dnorm() function in R. It can be concluded that using laplace distribution in rejection sampling method works successfully to generate normal distribution. Also, increasing the number of samples might make the generated distribution more like normal distribution.

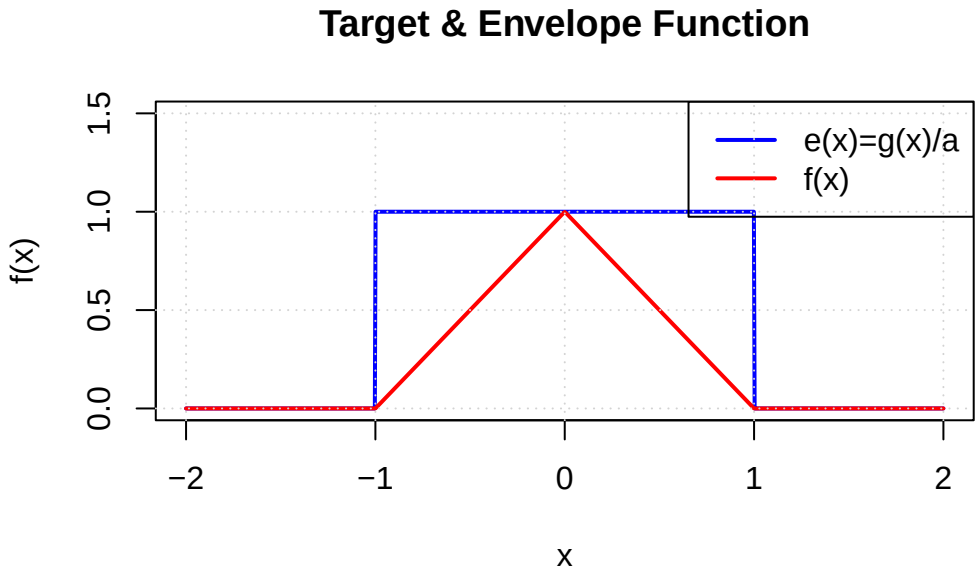
Appendix

Question 1:

part a)

```
# Define the function
triangle_function <- function(x){
  if (x < -1 || x > 1){
    return(0)
  }
  if (-1 <= x && x <= 0){
    return(x+1)
  }
  if (0 < x && x <= 1){
    return(1-x)
  }
}
```

```
x_values <- seq(-2, 2, length.out = 1000)
a <- 0.5
pdf_uniform_values <- dunif(x_values, min = -1, max = 1) / a
pdf_triangle_values <- sapply(x_values, triangle_function)

plot(x_values, pdf_uniform_values, type = 'l', col = 'blue', lwd = 2,
     xlab = 'x', ylab = 'f(x)',
     main = 'Target & Envelope Function',
     ylim = c(0, 1.5))
lines(x_values, pdf_triangle_values, col = 'red', lwd = 2)
legend('topright', legend = c('e(x)=g(x)/a', 'f(x)'),
       col = c('blue', 'red'), lty = 1, lwd = 2)
grid()
```




```
part_a_function <- function(n){
  a <- 0.5
  count <- 0
  generated_samples <- numeric(n)
```

```
  reject <- 0

  while (count < n){
    Y <- runif(1, min = -1, max = 1) # Sample from g(x)
    U <- runif(1)
    f_y <- triangle_function(Y)
    g_y <- dunif(Y, min = -1, max = 1)
    e_y <- g_y / a

    if ((f_y/e_y) >= U){
      count <- count + 1
      generated_samples[count] <- Y
    }
    else{
      reject <- reject + 1
    }
  }
  return(generated_samples)
}
```

part b)

```
part_b_function <- function(n){
  Y <- numeric(n)
  for (i in 1:n){
    u1 <- runif(1)
    u2 <- runif(1)
    if (u2 < 0.5){
      Y[i] <- 1-sqrt(1-u1)
    } else{
      Y[i] <- -(1-sqrt(1-u1))
    }
  }
  return(Y)
}
```

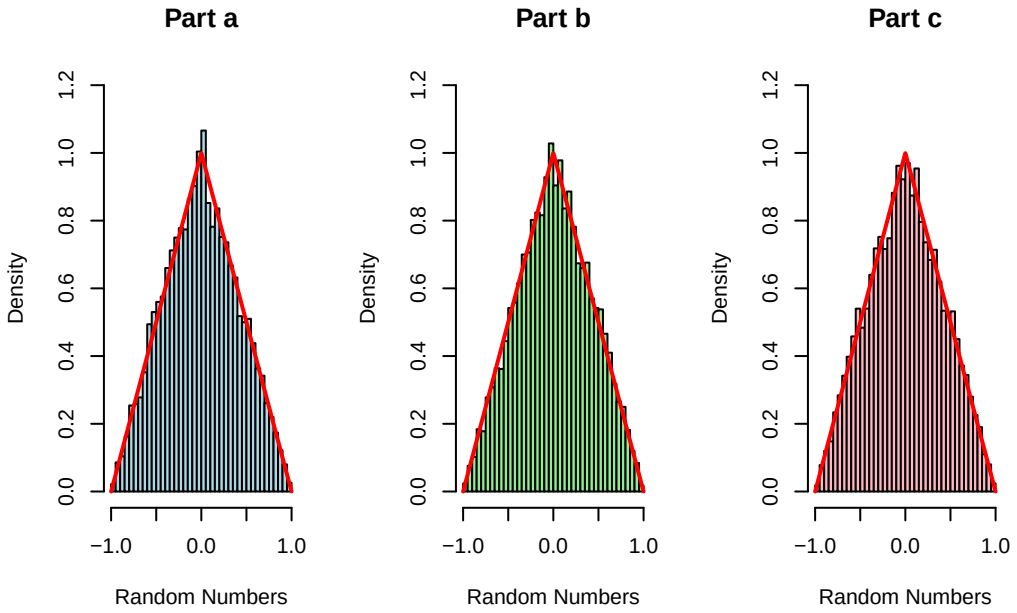
part c)

```
part_c_function <- function(n){
  U1 <- runif(n)
  U2 <- runif(n)
  generated_samples <- U1 - U2
  return(generated_samples)
}
```

part d)

```
n <- 10000
samples_a <- part_a_function(n)
samples_b <- part_b_function(n)
samples_c <- part_c_function(n)
x2_values <- seq(-1, 1, length.out = 1000)
y2_values <- sapply(x2_values, triangle_function)
```

```r
par(mfrow = c(1, 3))
hist(samples_a, breaks = 30, freq = FALSE, main = "Part a",
     xlab = "Random Numbers", ylim = c(0,1.2), col = "lightblue")
lines(x2_values, y2_values, col = 'red', lwd = 2)
hist(samples_b, breaks = 30, freq = FALSE, main = "Part b",
     xlab = "Random Numbers", ylim = c(0,1.2), col = "lightgreen")
lines(x2_values, y2_values, col = 'red', lwd = 2)
hist(samples_c, breaks = 30, freq = FALSE, main = "Part c",
     xlab = "Random Numbers", ylim = c(0,1.2), col = "lightpink")
lines(x2_values, y2_values, col = 'red', lwd = 2)
```




```r
cat("Variance of part a:", var(samples_a), "\n")
```

```
## Variance of part a: 0.166898
```

```r
cat("Variance of part b:", var(samples_b), "\n")
```

```
## Variance of part b: 0.1657791
```

```r
cat("Variance of part c:", var(samples_c), "\n")
```

```
## Variance of part c: 0.1687701
```

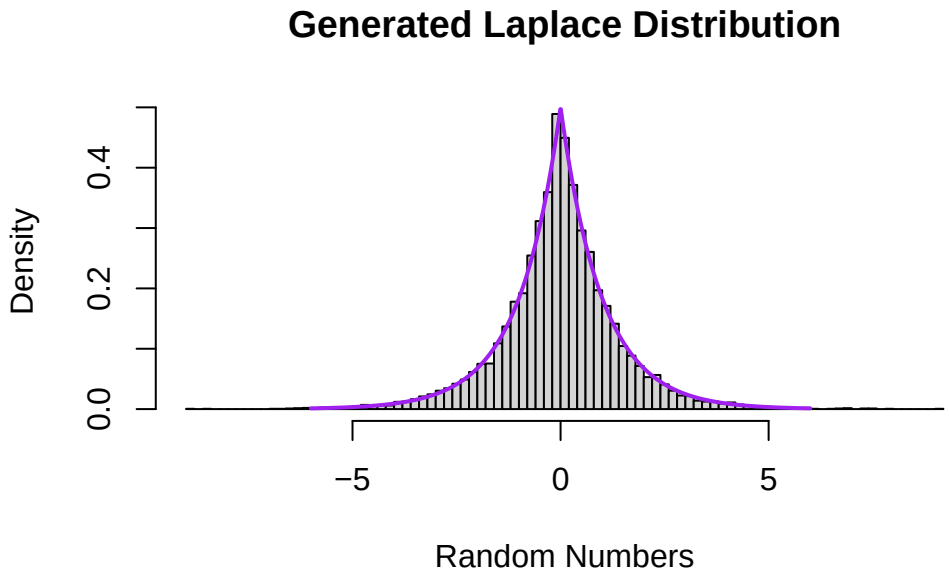
Question 2:

part a)

```r
laplace_pdf <- function(x, mu = 0, lambda = 1){
  return((lambda/2) * exp(-lambda*abs(x - mu)))
}

CDF_laplace <- function(x, mu = 0, lambda = 1){
  cdf <- ifelse(x < mu, (0.5)*exp(lambda*(x-mu)), 1-0.5*exp(-lambda*(x-mu)) )
  return(cdf)
}
```

```r
inverse_CDF_laplace <- function(u){
  result <- ifelse(u < 0.5, log(2*u), -(log(2-2*u)))
  return(result)
}
```

```r
set.seed(123)
U <- runif(10000)
# Use the inverse CDF to get the random variable
X <- inverse_CDF_laplace(U)
hist(X, breaks = 100, freq = FALSE, main = "Generated Laplace Distribution",
     xlab = "Random Numbers")

x_laplace_values <- seq(-6, 6, length.out = 1000)
y_laplace_values <- sapply(x_laplace_values, laplace_pdf)
lines(x_laplace_values, y_laplace_values, col = 'purple', lwd = 2)
```



part b)

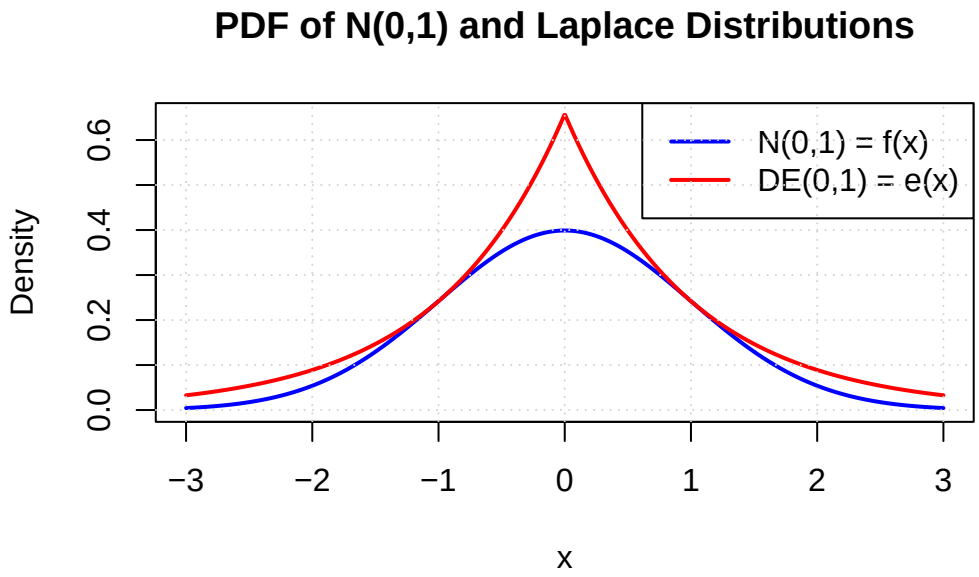
```r
normal_pdf <- function(x, mean = 0, sd = 1) {
  coefficient <- 1 / (sqrt(2 * pi) * sd)
  exponent <- -((x - mean) ^ 2) / (2 * sd^2)
  density <- coefficient * exp(exponent)
  return(density)
}
```

```r
# Define a new function to optimize
optimize_fnc <- function(x){
  g_x <- laplace_pdf(x)
  f_x <- normal_pdf(x)
  to_optimize <- g_x / f_x
  return(to_optimize)
}

# Use optim() function to find minimum a
init_x <- c(-1)
```

```r
a <- optim(init_x, fn = optimize_fnc, method = "BFGS")$value
cat("a = ", a)
```

```
## a =  0.7601735
```

```r
x_values <- seq(-3, 3, length.out = 1000)

# Calculate PDF values for Laplace and normal distributions at x_values
pdf_laplace_values <- laplace_pdf(x_values) / a
pdf_normal_values <- normal_pdf(x_values)

plot(x_values, pdf_normal_values, type = 'l', col = 'blue', lwd = 2,
     xlab = 'x', ylab = 'Density',
     main = 'PDF of N(0,1) and Laplace Distributions',
     ylim = c(0, max(pdf_normal_values, pdf_laplace_values)))
lines(x_values, pdf_laplace_values, col = 'red', lwd = 2)
legend('topright', legend = c('N(0,1) = f(x)', 'DE(0,1) = e(x)'),
       col = c('blue', 'red'), lty = 1, lwd = 2)
grid()
```




```r
# Generate samples from normal distribution using laplace distribution
n <- 2000
count <- 0
generated_samples <- numeric(n)
reject <- 0

while (count < n){
  U1 <- runif(1)
  U2 <- runif(1)
  Y <- inverse_CDF_laplace(U1) #generate number from laplace dist using inverse method
  f_y <- dnorm(Y) # target function
  e_y <- laplace_pdf(Y)/a # envelope function
  if ((f_y/e_y) >= U2){
    count <- count + 1
    generated_samples[count] <- Y
  }
```

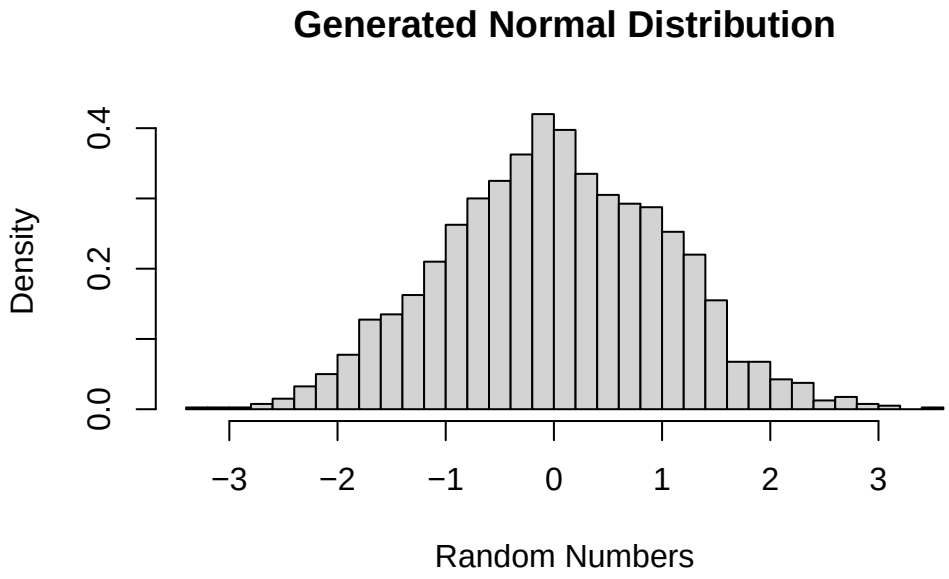
```
  else{
    reject <- reject + 1
  }
}

cat("Average Rejection Rate:", reject /(reject + count), "\n")
```

```
## Average Rejection Rate: 0.2369325
```

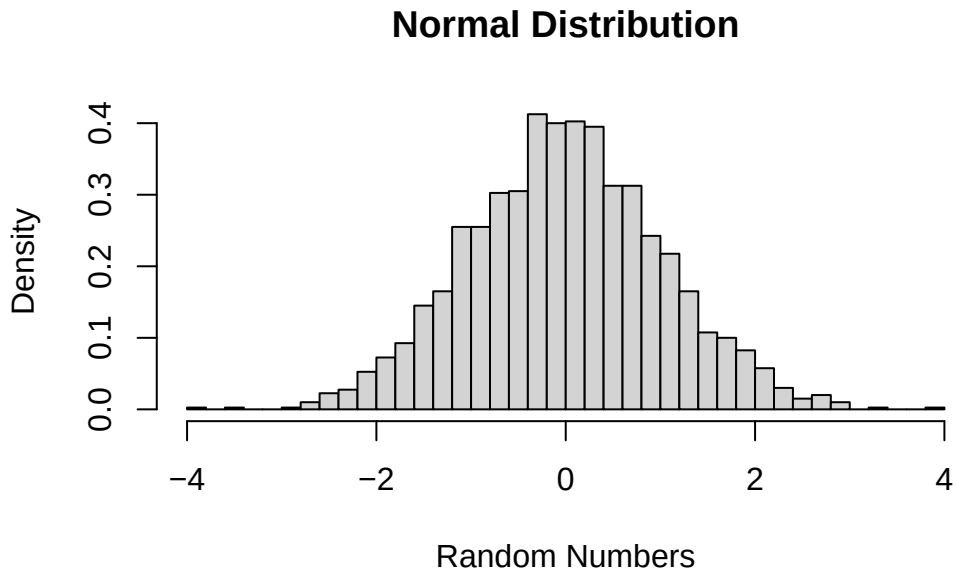
```
ER <- ((1/a) - 1) / (1/a)
cat("Expected Rejection Rate ER:", ER, "\n")
```

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

```
hist(generated_samples, breaks = 30, freq = FALSE,
     main = "Generated Normal Distribution", xlab = "Random Numbers")
```




```
actual_sample <- rnorm(n)
hist(actual_sample, breaks = 30, freq = FALSE, main = "Normal Distribution",
     xlab = "Random Numbers")
```




```
par(mfrow = c(1, 2))
x_normal_values <- seq(-5, 5, length.out = 1000)
y_normal_values <- sapply(x_normal_values, normal_pdf)

hist(generated_samples, breaks = 20, freq = FALSE, main = "Generated Normal Distribution",
     xlab = "Random Numbers", xlim = range(c(generated_samples, actual_sample)),
     ylim = c(0, 0.6),col = "lightpink")
lines(x_normal_values, y_normal_values, col = 'blue', lwd = 2)

hist(actual_sample, breaks = 20, freq = FALSE, main = "Normal Distribution",
     xlab = "Random Numbers", xlim = range(c(generated_samples, actual_sample)),
     ylim = c(0, 0.6), col = "lightgreen")
lines(x_normal_values, y_normal_values, col = 'blue', lwd = 2)
```