Jared Yu

Computational Statistics

Problem Set 4

NOTE: Code is at the end of the document in the Code Appendix.

0. Read Example 6.2.
1. Use the Monte Carlo Method of Integration to evaluate the following integral:

$$I = \int_0^1 4\sqrt{1-x^2}\,dx$$

Let $h(x) = 4\sqrt{1-x^2}$, then I can be rewritten as follows:

$$I = \int_0^1 (1-0)4\sqrt{1-x^2}\,\frac{1}{1-0}\,dx = \int_0^1 g(x)f(x)\,dx,$$

where $g(x) = (1-0)4\sqrt{1-x^2} = 4\sqrt{1-x^2}$ and $f(x) = \frac{1}{1-0} = 1$. Therefore, the above evaluates to:

$$\rightarrow E[g(x)] = E\left[4\sqrt{1-x^2}\right]$$

So, the above integral is the same as taking the expectation of $g(x)$ w.r.t. the uniform density $f(x)$. Using the Monte Carlo method of integration, a simulated sample of $X \sim Uniform[0,1]$ can be used to approximate $\hat{I}$. The Monte Carlo estimate can be written as follows:

$$\hat{I} = \frac{1}{n}\sum_{i=1}^{n} g(x_i),$$

where $x_1, \cdots, x_n$ are drawn from $Uniform[0,1]$. Using RStudio, and sampling 1,000,000 observations using `runif()`, the mean of $g(x_i)'s$ evaluates to 3.141308, which is roughly π.

Next, using an analytical method to compare the solutions.

$$\int_0^1 4\sqrt{1-x^2}\,dx = 4\int_0^1 (1-x^2)^{\frac{1}{2}}\,dx$$

Using u-substitution:

$$x = \sin(u)\,, u = \arcsin(x)$$
$$dx = \cos(u)\,du$$

$$\rightarrow = 4\int_0^{\frac{\pi}{2}} (1-\sin^2(u))^{\frac{1}{2}}\cos(u)du = 4\int_0^{\frac{\pi}{2}} (\cos^2(u))^{\frac{1}{2}}\cos(u)du = 4\int_0^{\frac{\pi}{2}} \cos^2(u)\,du$$

Then using the following double angle identity:

$$\cos(2u) = 2\cos^2(u) - 1 \rightarrow \cos^2(u) = \frac{1+\cos(2u)}{2}$$

$$\rightarrow = 4\int_0^{\frac{\pi}{2}} \frac{1+\cos(2u)}{2}\,du = 2\int_0^{\frac{\pi}{2}} 1+\cos(2u)\,du = \int_0^{\frac{\pi}{2}} 2\,du + \int_0^{\frac{\pi}{2}} 2\cos(2u)\,du$$

Using u-substitution again (but with v):

$$v = 2u$$
$$dv = 2du$$

$$\rightarrow = \int_0^{\pi} 1\,dv + \int_0^{\pi} \cos(v)\,dv = v|_0^{\pi} + \sin(v)\,|_0^{\pi} = \pi + 0 = \pi$$

Therefore, the solutions found for the expectation of $g(x)$ found numerically and analytically are roughly similar.

2. Consider the (triangular) probability density function defined as

$$f(x) = \begin{cases} 4x & 0 \le x \le \frac{1}{2} \\ 4-4x & \frac{1}{2} \le x \le 1 \end{cases}$$

   a. Draw plots of the density and of the CDF.

Below are plots of the p.d.f. to the left and CDF to the right (Figure 1).

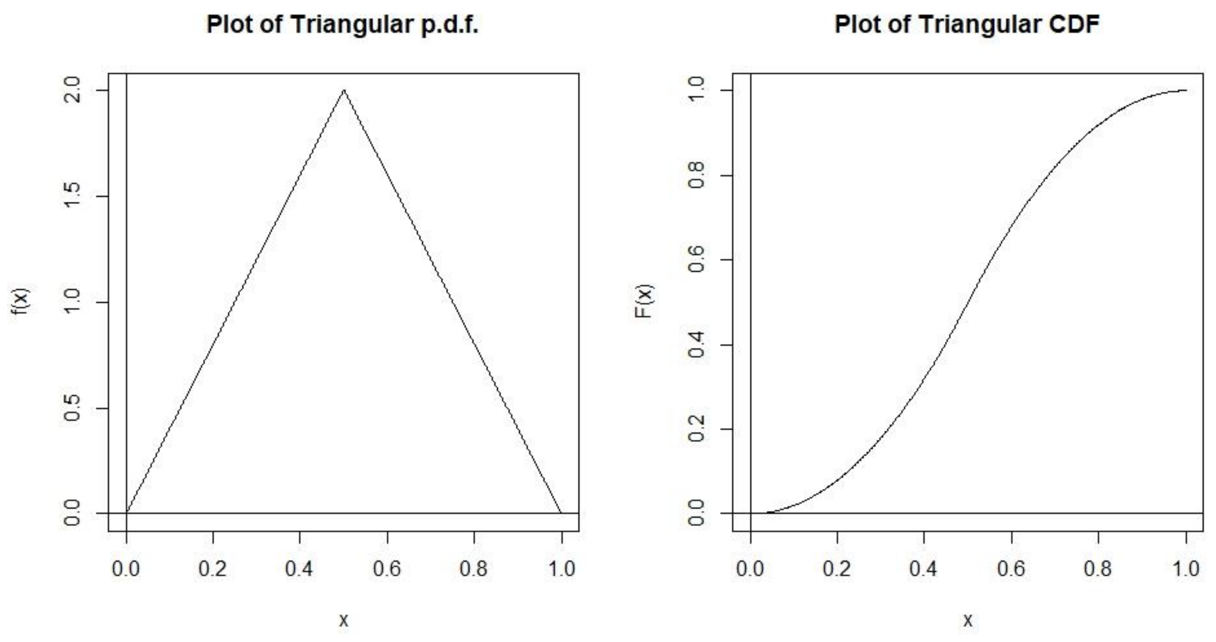

Figure 1

b. Use the Inverse CDF method to analytically find a function $G(u)$ that transforms random numbers from $U(0,1)$ distribution into random numbers from $f(x)$.

First, it's necessary to determine the CDF of $f(x)$.

If $0 \leq x \leq \frac{1}{2}$:

$$F(x) = \int_0^x 4t\,dt = \frac{4}{2}t^2|_0^x = 2x^2$$

If $\frac{1}{2} \leq x \leq 1$:

$$F(x) = \int_0^{\frac{1}{2}} 4t\,dt + \int_{\frac{1}{2}}^x 4 - 4t\,dt = 2t^2|_0^{\frac{1}{2}} + \left[4t|_{\frac{1}{2}}^x - 2t^2|_{\frac{1}{2}}^x\right] = 2\left(\frac{1}{4}\right) + 4\left(x - \frac{1}{2}\right) - 2\left(x^2 - \frac{1}{4}\right)$$

$$= \frac{1}{2} + 4x - 2 - 2x^2 + \frac{1}{2} = -2x^2 + 4x - 1$$

$$\Rightarrow F(x) = \begin{cases} 2x^2 & 0 \leq x \leq \frac{1}{2} \\ -2x^2 + 4x - 1 & \frac{1}{2} \leq x \leq 1 \end{cases}$$

Then, the next step is to take the inverse of the CDF. An important point in the range of $F(x)$ is $\frac{1}{2}$, since $F\left(\frac{1}{2}\right) = \frac{1}{2}$ for both pieces of the CDF.

$$\rightarrow y = 2x^2 \rightarrow x = \sqrt{\frac{y}{2}} \text{ for } \left(0 \leq y \leq \frac{1}{2}\right)$$

$$\rightarrow y = -2x^2 + 4x - 1 \rightarrow -2x^2 + 4x + (-1 - y) = 0$$

$$\rightarrow \frac{-4 \pm \sqrt{16 - 4(-2)(-1 - y)}}{2(-2)} = \frac{4 \pm \sqrt{8 - 8y}}{4}$$

It's important to choose the equation that allows for the output to be within the desired range of $\left[\frac{1}{2}, 1\right]$, therefore $\frac{4 - \sqrt{8 - 8y}}{4}$ is chosen. Then the inverse of the CDF is as follows:

$$F^{-1}(y) = \begin{cases} \sqrt{\frac{y}{2}} & 0 \leq y \leq \frac{1}{2} \\ \frac{4 - \sqrt{8 - 8y}}{4} & \frac{1}{2} \leq y \leq 1 \end{cases}$$

Then the function $G(u)$ that transforms random numbers from $U(0,1)$ is equal to $F^{-1}(y)$. The function is written in RStudio as `G()` and is vectorized to `G_vec()`. It can be found in the Appendix.

c. Use part (b) to generate 1000 data points from $f(x)$ and plot their histogram.

Below (Figure 2) is the plot of the histogram after generating 1,000 values using `runif()` and using them as inputs to `G_vec()`. The appearance is roughly triangular, but with some apparent "heaviness" on the left side. It's possible that with more observations generated (e.g., 1,000,000) that the histogram will appear more "triangular" in line with the shape of the p.d.f. of $f(x)$.

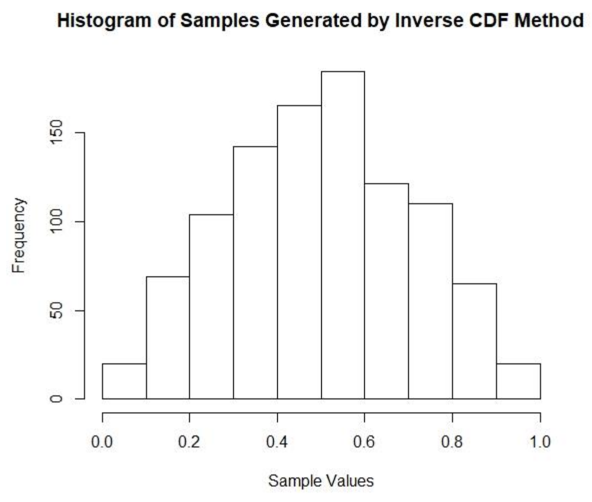

Figure 2

d. Use rejection sampling to generate 1000 data points from $f(x)$ and plot their histogram. What instrumental distribution and envelope did you use?

Below (Figure 3) is the plot of the histogram of 1,000 samples after generating them by rejection sampling.

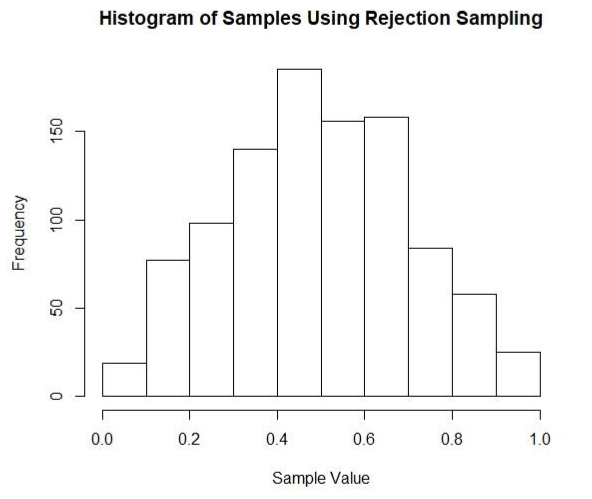

Figure 3

The envelope used was based on the following graph (Figure 4) of the Normal distribution (the chosen instrumental distribution) in red above the triangular p.d.f.

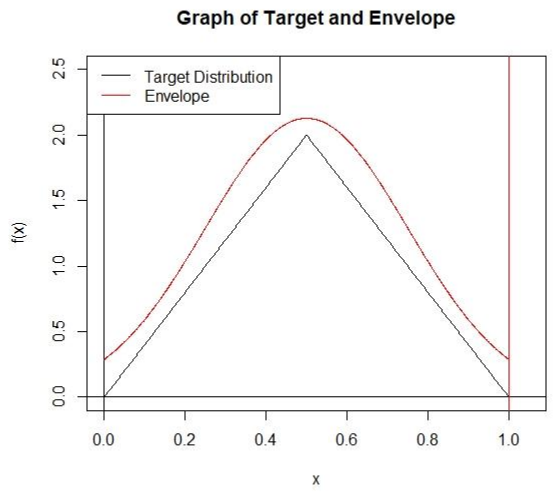

Figure 4

Different graphs were tested in RStudio with the command `dnorm()`. The chosen values for `mu` and `sd` were 0.5 and 0.25 respectively. The value of α in the equation for the envelope was chosen to be 0.7. The equation for the envelope then is as follows:

$$e(x) = \frac{g(x)}{\alpha},$$

where $g(x)$ is the Normal distribution with $\mu = 0.5$ and $\sigma = 0.25$, and $\alpha = 0.75$. The reason that the Normal distribution was selected as the instrumental distribution was that it seemed apparent that if a bell-curve of a Normal distribution had the correct σ then it could suitably cover the entire graph of the triangular p.d.f. After testing only a few values it seemed apparent that this set of parameters would act as the tightest fitting envelope.

The vertical red line indicates that values that are outside of the range of $[0,1]$ will also be rejected from the sample. When sampling values, the acceptance rate was $1 - \frac{334}{1000} = 0.666$, which is somewhat close to $\alpha = 0.75$.

e. Use the Monte Carlo method to estimate $E[X^2]$ when X is drawn from the distribution with density f and compare your estimate to the actual value. You may use the data generated in (c) or (d) for your estimation.

Solving analytically leads to the following:

$$E[X^2] = \int_0^{\frac{1}{2}} 4x^3\,dx + \int_{\frac{1}{2}}^{1} 4x^2 - 4x^3\,dx = x^4|_0^{\frac{1}{2}} + \left[\frac{4}{3}x^3|_{\frac{1}{2}}^{1} - x^4|_{\frac{1}{2}}^{1}\right]$$
$$= \left(\frac{1}{2}\right)^4 + \frac{4}{3}\left(1 - \frac{1}{8}\right) - 1 + \left(\frac{1}{2}\right)^4 = -\frac{7}{8} + \frac{7}{6} = \frac{7}{24} \approx 0.2917$$

Using the data generated from part (c), $E[X^2]$ came out to be 0.2916822 or approximately 0.2917. Therefore, from that test alone it seems that the inverse CDF method did a decent job of sampling from the target distribution.

3. Assume that you want to sample from a $N(0,1)$ distribution using a $N(1,2)$ distribution as instrumental distribution.
   a. Draw a sample of size 10000 using rejection sampling, plot the histogram and calculate the mean and variance of the sample.

Below (Figure 5) is a plot of the histogram of the data generated through rejection sampling.

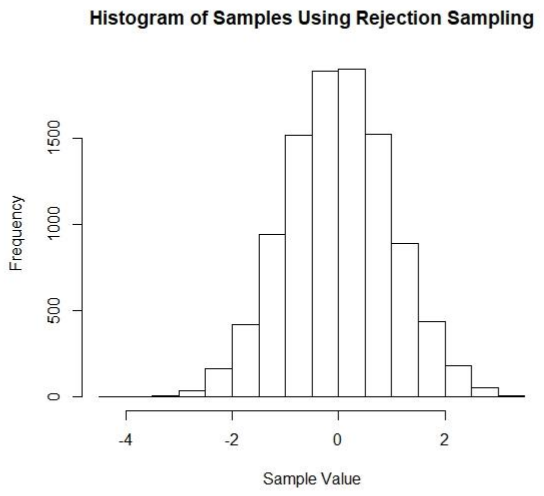

Figure 5

The mean of the sample is approximately 0.0049 and the variance is approximately 0.9987. Both the mean and variance are close to the target distribution's mean and variance of 0 and 1. Furthermore, the shape of the histogram appears similar to that of the graph of the target distribution's p.d.f.

   b. What were your choices for α and the envelope $e(x)$?

The choice for α is 0.4 and the envelope $e(x)$ then became $\frac{g(x)}{\alpha}$ where $g(x)$ is the Normal distribution with mean 1 and variance 2. This was selected after utilizing the following graph to search for an α that would fit $g(x)$ (a.k.a. the instrumental distribution) as close as possible to the target distribution (i.e., the standard Normal distribution). The graph below (Figure 6) shows the target distribution in black and the instrumental distribution in red after being divided by α (a.k.a., the envelope, or $e(x)$).

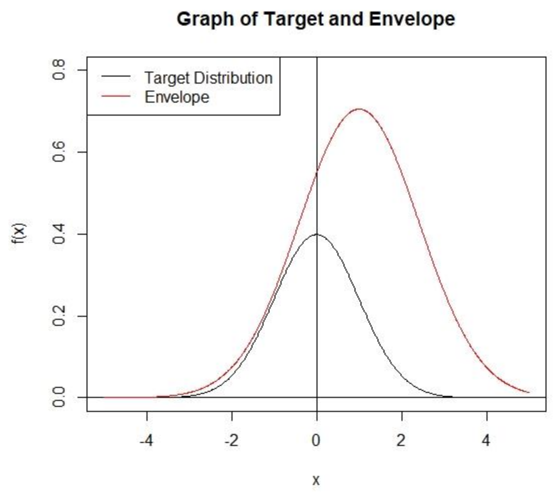

Figure 6

c. Now consider the same problem, but using squeezed rejection sampling. What advantage is there to this method over standard rejection sampling? Give an example of a squeezing function that could be used to solve this problem. Note: You do not need to generate the random samples for this part.

Squeezed rejection sampling is a useful technique to use in place of standard rejection sampling in cases where evaluating the target distribution is computationally expensive. In such cases, it's possible to use a squeezing function, $s(x)$, that fits inside (i.e., under its graph) the target distribution, $f(x)$, and is easier to calculate. By letting the algorithm first check whether the sampled point, (call it y^*) passes the comparison between the squeezing function and the envelope (i.e., $u \leq \frac{s(y^*)}{e(y^*)}$), it's possible then to skip the task in many cases of having to calculate a computationally more expensive target distribution. Only in cases when the point is rejected once already (i.e., $u > \frac{s(y^*)}{e(y^*)}$), is it checked a second time, requiring the evaluation of the point as a function of the target distribution (i.e., check if $u \leq \frac{f(y^*)}{e(y^*)}$).

A possible squeezing function $s(x)$ is as follows:

$$s(x) = \begin{cases} \frac{1}{3} & \text{if} - 0.5 \leq x \leq 0.5 \\ 0.2 & \text{if} - 1 \leq x < -0.5 \text{ or } 0.5 < x \leq 1 \\ 0.07 & \text{if} - 1.8 \leq x < -1 \text{ or } 1 < x \leq 1.8 \\ 0 & \text{otherwise} \end{cases}$$

Below (Figure 7) is a graph of the target distribution in black and the squeezing function in red. This squeezing function however is ideally simpler to compute than the standard Normal distribution. To be more technical, it would make sense if looping through the few if-statements were quicker than calculating the entirety of the standard Normal equation.

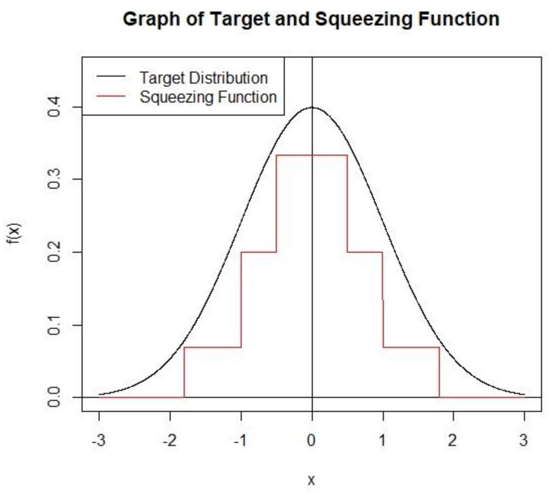
Graph of Target and Squeezing Function
Target Distribution
Squeezing Function
f(x)
0.0
0.1
0.2
0.3
0.4
-3
-2
-1
0
1
2
3
x

Figure 7

Code Appendix

```r
library(latex2exp) # Load libary
### Problem 1
set.seed(664)
x <- runif(1000000)
g <- function(x) {
  4 * sqrt(1 - x^2)
}
mean(g(x)) # 3.141308
# analytically solves to pi

### Problem 2
# part (a)
triangular_pdf <- function(x) { # target distribution pdf
  if((0 <= x) & (x <= 0.5)) {
    return(4 * x)
  } else if((0.5 < x) & (x <= 1)) {
    return(4 - 4 * x)
  } else {
    return(0)
  }
}

triangular_pdf_vec <- Vectorize(triangular_pdf)
xs <- seq(0, 1, length.out = 500)
ys <- triangular_pdf_vec(x = xs)
par(mfrow = c(1,2))
plot(xs, ys, type = 'l', main = 'Plot of Triangular p.d.f.',
     xlab = TeX('$x$'), ylab = TeX('$f(x)$'))
abline(h = 0)
abline(v = 0)

triangular_cdf <- function(x) { # target distribution CDF
  if((0 <= x) & (x <= 0.5)) {
    return(2 * x^2)
  } else if((0.5 < x) & (x <= 1)) {
    return(-2 * x^2 + 4 * x - 1)
  } else {
    return(0)
  }
}

triangular_cdf_vec <- Vectorize(triangular_cdf)
xs <- seq(0, 1, length.out = 500)
ys <- triangular_cdf_vec(x = xs)
plot(xs, ys, type = 'l', main = 'Plot of Triangular CDF',
     xlab = TeX('$x$'), ylab = TeX('$F(x)$'))
abline(h = 0)
abline(v = 0)
dev.off()

# part (b)
inverse_cdf <- function(y) { # Inverse CDF
  if ((0 <= y) & (y <= 0.5)) {
    return(sqrt(y / 2))
  } else if ((0.5 < y) & (y <= 1)) {
    return((4 - sqrt(8 - 8 * y)) / 4)
  } else {
    return(0)
  }
}

G <- function(u) inverse_cdf(u) # Function 'G(u)'
G_vec <- Vectorize(G)

# part (c)
set.seed(664)
us <- runif(n = 1e3)
inverse_cdf_samples <- G_vec(us)
hist(inverse_cdf_samples, xlab = 'Sample Values',
     main = 'Histogram of Samples Generated by Inverse CDF Method')

# part (d)
xs <- seq(0, 1, length.out = 500) # graph of target distribution
ys <- triangular_pdf_vec(x = xs)
plot(xs, ys, type = 'l', main = 'Graph of Target and Envelope',
     xlab = TeX('$x$'), ylab = TeX('$f(x)$'), ylim = c(0,2.5), xlim = c(0, 1.05))
abline(h = 0)
abline(v = 0)

bounds <- seq(0, 1, length.out = 1000) # graph of instrumental distribution
gs <- dnorm(bounds, mean = 0.5, sd = 0.25)
alpha <- 0.75
es <- gs / alpha
lines(bounds, es, col = 'red')
abline(v = 1, col = 'red')
legend("topleft", c("Target Distribution", "Envelope"),
       col = c("black", "red"), lty = c(1,1))

rejection_rule <- function(u, y, alpha = 0.75) { # Used in rejection_sampling()
  f_y <- triangular_pdf(y)
  # Reference: https://www.r-bloggers.com/normal-distribution-functions/
  envelope <- dnorm(y, mean = 0.5, sd = 0.25) / alpha
  if ((u > (f_y / envelope)) | (y < 0) | (y > 1)) {
    return(FALSE)
  } else {
    return(TRUE)
  }
}

rejection_sampling <- function(num_samples = 1000) { # Performs rejection sampling
  sampling_matrix <- matrix(NA, nrow = num_samples)
  count <- 0; rejects <- 0 # Initialize variables

  while (count != num_samples) { # Sample values
    y <- rnorm(n = 1, mean = 0.5, sd = 0.25)
    u <- runif(n = 1, min = 0, max = 1)

    if (rejection_rule(u, y)) { # Check for rejection
      sampling_matrix[count + 1] <- y
      count <- count + 1 # Accept values
    } else {
      rejects <- rejects + 1
    }
  }
  cat('Rejects:', rejects)
  return(sampling_matrix)
}

set.seed(664)
rejection_samples <- rejection_sampling()
```

```r
1 - 334 / 1000 # acceptance rate = 0.666, somewhat close to alpha = 0.75
hist(as.vector(rejection_samples), xlab = 'Sample Value', # histogram
     main = 'Histogram of Samples Using Rejection Sampling')

# part (e)
mean(inverse_cdf_samples^2) # 0.2916822
7/24 # 0.2916667

### Problem 3
# part (a)
std_normal_pdf <- function(x) dnorm(x = x, mean = 0, sd = 1)
xs <- seq(-5, 5, length.out = 500) # graph of target distribution
ys <- std_normal_pdf(x = xs)
plot(xs, ys, type = 'l', main = 'Graph of Target and Envelope',
     xlab = TeX('$x$'), ylab = TeX('$f(x)$'), ylim = c(0, 0.8))
abline(h = 0)
abline(v = 0)

bounds <- seq(-5, 5, length.out = 1000) # graph of instrumental distribution
normal_1_2 <- function(x) dnorm(x = x, mean = 1, sd = sqrt(2))
gs <- normal_1_2(bounds)
alpha <- 0.4
es <- gs / alpha # envelope
lines(bounds, es, col = 'red')
legend("topleft", c("Target Distribution", "Envelope"),
       col = c("black", "red"), lty = c(1,1))

rejection_rule2 <- function(u, y, alpha = 0.4) { # Used in rejection_sampling()
  f_y <- std_normal_pdf(x = y)
  # Reference: https://www.r-bloggers.com/normal-distribution-functions/
  envelope <- normal_1_2(x = y) / alpha
  if (u > (f_y / envelope)) {
    return(FALSE)
  } else {
    return(TRUE)
  }
}

rejection_sampling2 <- function(num_samples = 10000) { # Performs rejection sampling
  sampling_matrix <- matrix(NA, nrow = num_samples)
  count <- 0; rejects <- 0 # Initialize variables

  while (count != num_samples) { # Sample values
    y <- rnorm(n = 1, mean = 1, sd = sqrt(2))
    u <- runif(n = 1, min = 0, max = 1)

    if (rejection_rule2(u, y)) { # Check for rejection
      sampling_matrix[count + 1] <- y
      count <- count + 1 # Accept values
    } else {
      rejects <- rejects + 1
    }
  }
  cat('Rejects:', rejects)
  return(sampling_matrix)
}

set.seed(664)
rejection_samples2 <- rejection_sampling2()
1 - 15024 / 10000 # acceptance rate = -0.5024, far from alpha = 0.4
hist(as.vector(rejection_samples2), xlab = 'Sample Value', # histogram
     main = 'Histogram of Samples Using Rejection Sampling')
mean(rejection_samples2); var(rejection_samples2)

# part (b)
# no code required

# part (c)
xs <- seq(-3, 3, length.out = 5000) # graph of target distribution
ys <- std_normal_pdf(x = xs)
plot(xs, ys, type = 'l', main = 'Graph of Target and Squeezing Function',
     xlab = TeX('$x$'), ylab = TeX('$f(x)$'), ylim = c(0, 0.45))
abline(h = 0)
abline(v = 0)
legend("topleft", c("Target Distribution", "Squeezing Function"),
       col = c("black", "red"), lty = c(1,1))

squeeze_function <- function(x) { # squeezing function
  if ((-0.5 <= x) & (x <= 0.5)) {
    return(1/3)
  } else if (((-1 <= x) & (x < -0.5)) | ((0.5 < x) & (x <= 1))) {
    return(0.2)
  } else if (((-1.8 <= x) & (x < -1)) | ((1 < x) & (x <= 1.8))) {
    return(0.07)
  } else {
    return(0)
  }
}
squeeze_function_vec <- Vectorize(squeeze_function)
bounds <- seq(-3, 3, length.out = 5000) # graph the squeeze function
lines(bounds, squeeze_function_vec(bounds), col = 'red')
```