# Atlas-PS 4

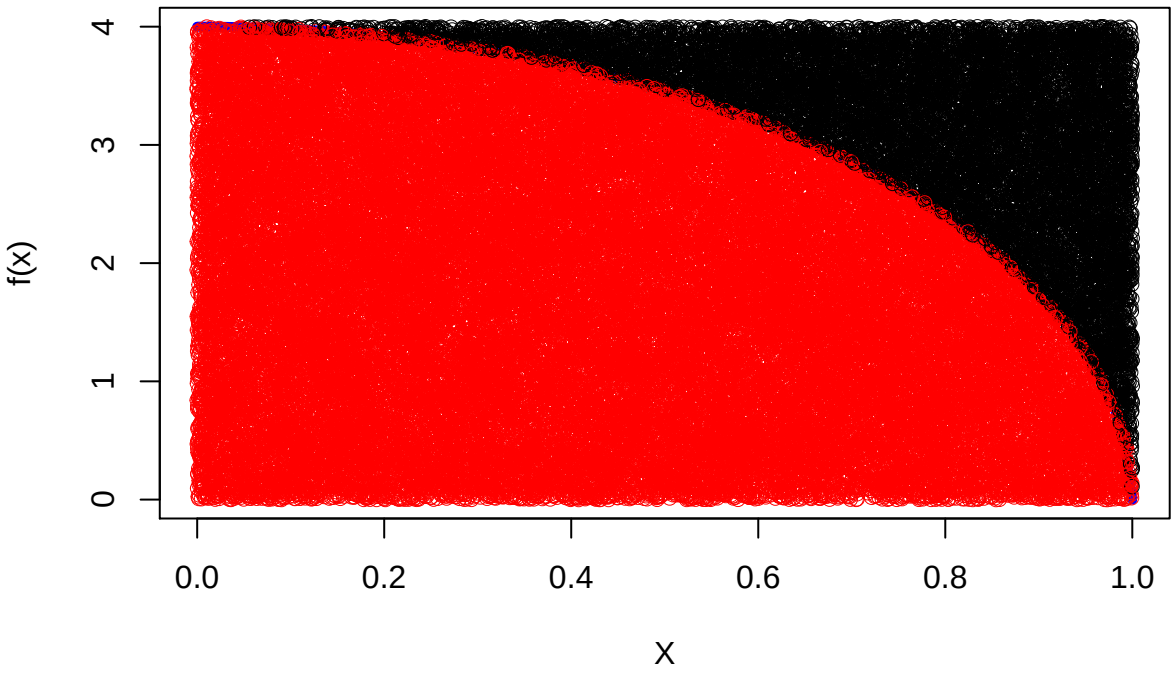
*David Atlas*

*9/21/2018*

## Problem 1

We implement a Monte Carlo Integration function and apply it to $f(x) = 4\sqrt{1-x^2}, x \in [0,1]$.

```
set.seed(73)

range0 <- function(avector){
  return(max(avector) - min(avector))
}

monte_carlo_integration <- function(f, a, b, n, plot=FALSE){

  # Get the range of the function on the space
  f_range <- range(f(seq(a, b, .01)))

  total_area <- range0(f_range) * (b - a)

  x_points <- runif(n, a, b)
  y_points <- runif(n, f_range[1], f_range[2])

  proportion_under <- mean(f(x_points) >= y_points)

  # Plot the scatter plot of points
  if(plot){
    plot(seq(a, b, .01), f(seq(a, b, .01)), 'l',
         lwd=5, col='blue', xlab='X', ylab=TeX("$f(x)$"))
    points(x_points, y_points,
           col= as.factor(ifelse(f(x_points) <= y_points, 0, 1)),
           lwd=.25)
  }

  return(total_area * proportion_under)


}


f <- function(x) 4 * sqrt(1 - x ^ 2)
monte_carlo_integration(f, 0, 1, 50000, plot=TRUE)
```




```
## [1] 3.14528
```

The solution is 3.14. This is as expected, as the integral defines the 4 times the area of a quarter unit circle, which has an area equal to $\frac{\pi}{4}$, and thus, the integral approximates the value of $\pi$.

# Problem 2

## a)

The PDF is

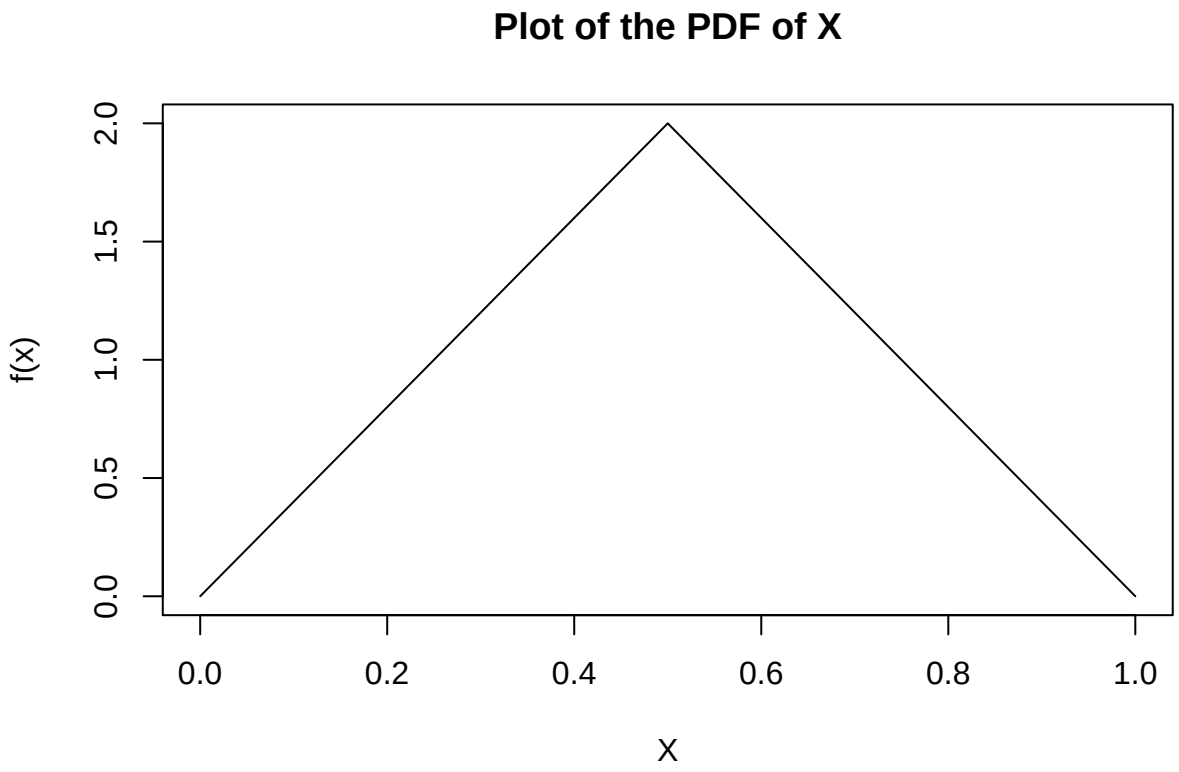
$$f(x) = \begin{cases} 4x & 0 < x \leq \frac{1}{2} \\ 4 - 4x & \frac{1}{2} \leq x \leq 1. \end{cases}$$

The CDF can be found by integrating each of the piecewise components. We subtract 1 from the second term to enforce continuity of the CDF, and to ensure that $F(\infty) = 1$

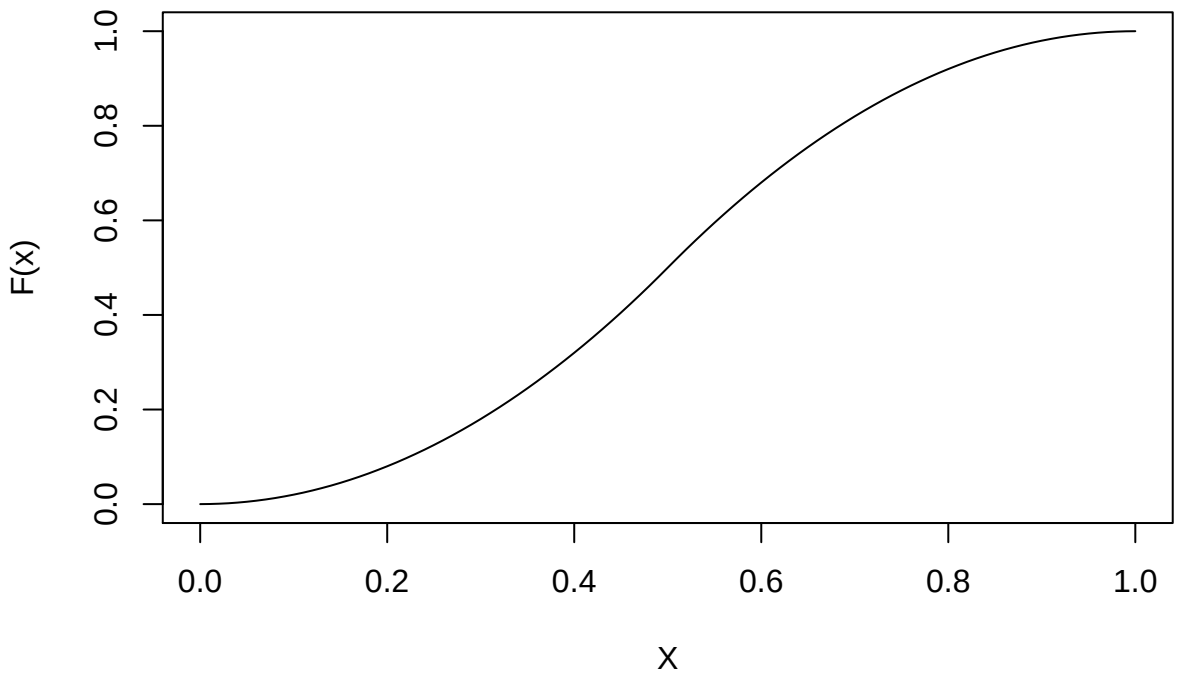
$$F(x) = \begin{cases} 2x^2 & 0 < x \leq \frac{1}{2} \\ 4x - 2x^2 - 1 & \frac{1}{2} < x \leq 1. \end{cases}$$

We plot the PDF and the CDF below.

```
pdf <- function(x) ifelse(x <= .5, 4 * x, 4 - 4*x)
cdf <- function(x) ifelse(x <= .5, 2 * x ^2, 4*x - 2*x^2 -1)

x <- seq(0, 1, .01)

plot(x, pdf(x), 'l', xlab='X', ylab=TeX('$f(x)$'), main='Plot of the PDF of X')
```




```
plot(x, cdf(x), 'l', xlab='X', ylab=TeX('$F(x)$'), main='Plot of the CDF of X')
```



## b)

We find the inverse of the CDF. The inverse of the first piecewise component is derived first:

$$\begin{aligned} x &= 2F^{-1}(x)^2 \\ \implies F^{-1}(x) &= \sqrt{\frac{x}{2}}. \end{aligned}$$

The inverse of the second piecewise component is derived second. Here, we complete the square to isolate $F^{-1}(x)$.

$$\begin{aligned} x &= 4F^{-1}(x) - 2F^{-1}(x) - 1 \\ \implies -\frac{x}{2} &= F^{-1}(x) - 2F^{-1}(x) + \frac{1}{2} \\ \implies -\frac{x}{2} + \frac{1}{2} &= F^{-1}(x) - 2F^{-1}(x) + 1 = (F^{-1}(x) - 1)^2 \\ \implies F^{-1}(x) &= 1 - \sqrt{-\frac{x}{2} + \frac{1}{2}}. \end{aligned}$$

.

Additionally, we must ensure that the domain of the inverse CDF is appropriate; however, for this example, the domains actually remain the same. Additionally, the square roots can be either positive or negative, but due to the domain, $[0, 1]$, we can disregard the terms that are not relevant.

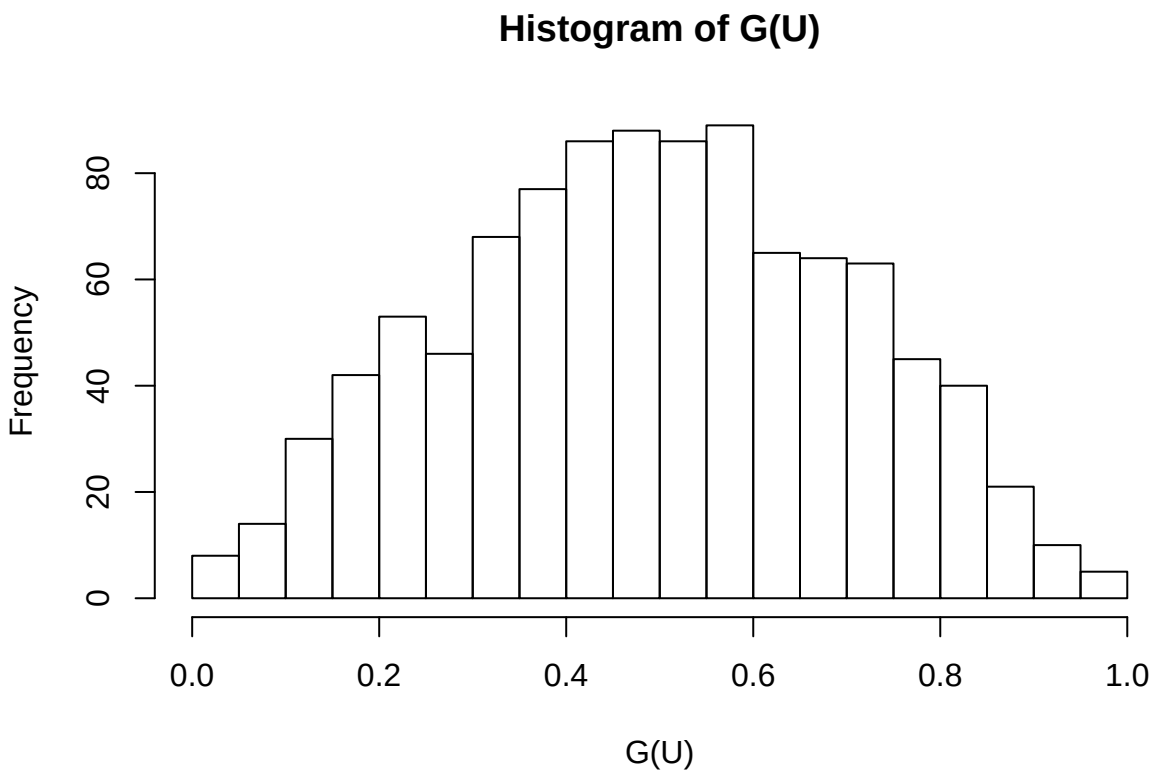
Therefore, we can write the inverse CDF as

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

## c)

We generate 1000 random points using part b and plot a histogram.

```
set.seed(73)
U <- runif(1000, 0, 1)
G <- function(u) ifelse(u <= .5, sqrt(u / 2),  1 - sqrt(-.5 * u + .5) )
hist(G(U),  breaks = seq(0, 1, .05))
```



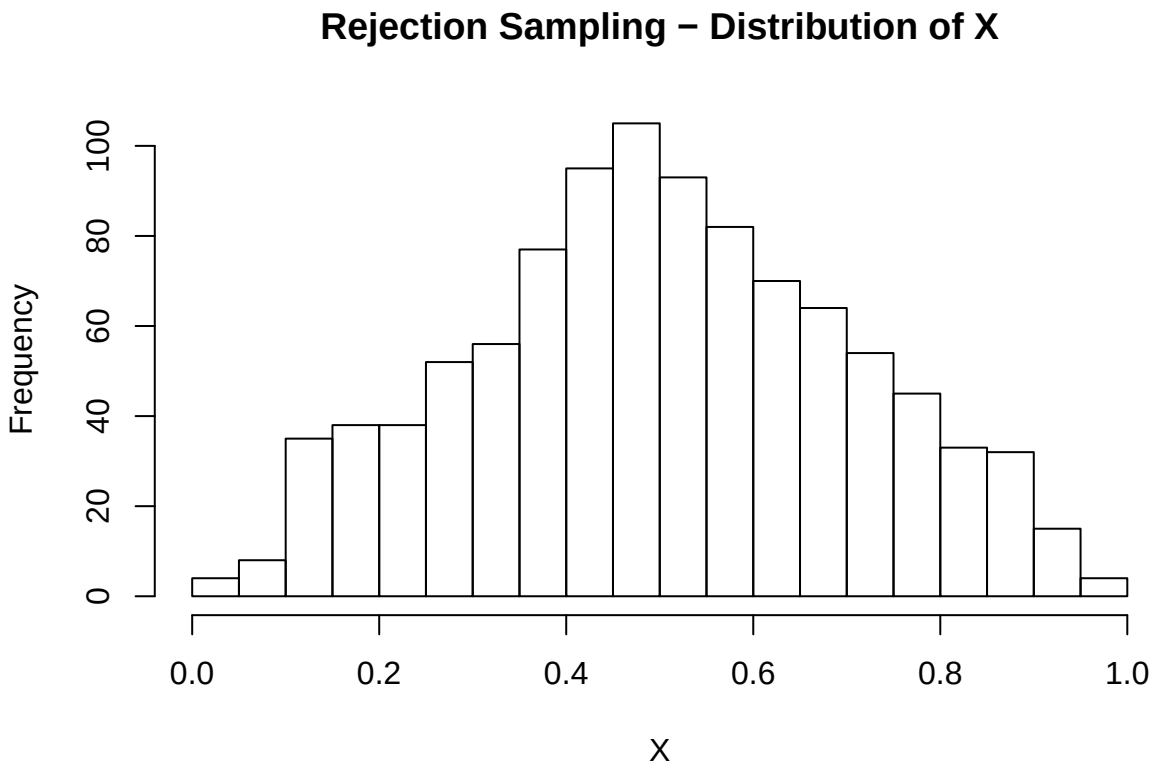

```
inverse_cdf_sample <- G(U) # save this for later
```

The distribution converges on the original density function.

## d)

We use rejection sampling to generate a random sample from the distribution. We can use a uniform distribution as the instrumental distribution, $(g(x) \sim U(0, 2))$ with an envelope $e(x) = \frac{g(x)}{1}; \alpha = 1$.

```
set.seed(73)

f <- function(x) ifelse(x <= .5, 4 * x, 4 - 4*x)

# Generate large samples so there will be 1000 non-rejected points
Y <- runif(100000, 0, 2)
U <- runif(100000, 0, 1)

# Get the first 1000 points that were not rejected
x <- U[Y<f(U)][0:1000]
hist(x, breaks = seq(0, 1, .05), main="Rejection Sampling - Distribution of X", xlab='X')
```




Here, we see that the distribution is very similar (and appears to be converging) on the original density function.

## e)

To use the Monte Carlo method to approximate $E[X^2]$, we take our sample from above and take the mean of the squared values. This is because we are trying to find $\int_0^1 g(x)f(x)dx$ where $f(x)$ is the piecewise density found above, and $g(x) = x^2$.

```
# Use the sample from above. Square the values and take the mean.
round(mean(inverse_cdf_sample^2), 3)
```

```
## [1] 0.286
```

Our simulation method yields .286. We need to find the actual value. We can write

$$\begin{aligned} E[X^2] &= \int_0^1 x^2 f(x)dx \\ &= \int_0^{\frac{1}{2}} 4x^3 dx + \int_{\frac{1}{2}}^1 (4x^2 - 4x^3)dx \\ &= \frac{1}{16} + \frac{1}{3} - (\frac{1}{6} - \frac{1}{16}) \\ &= \frac{1}{16} + \frac{22}{96} = .29166. \end{aligned}$$

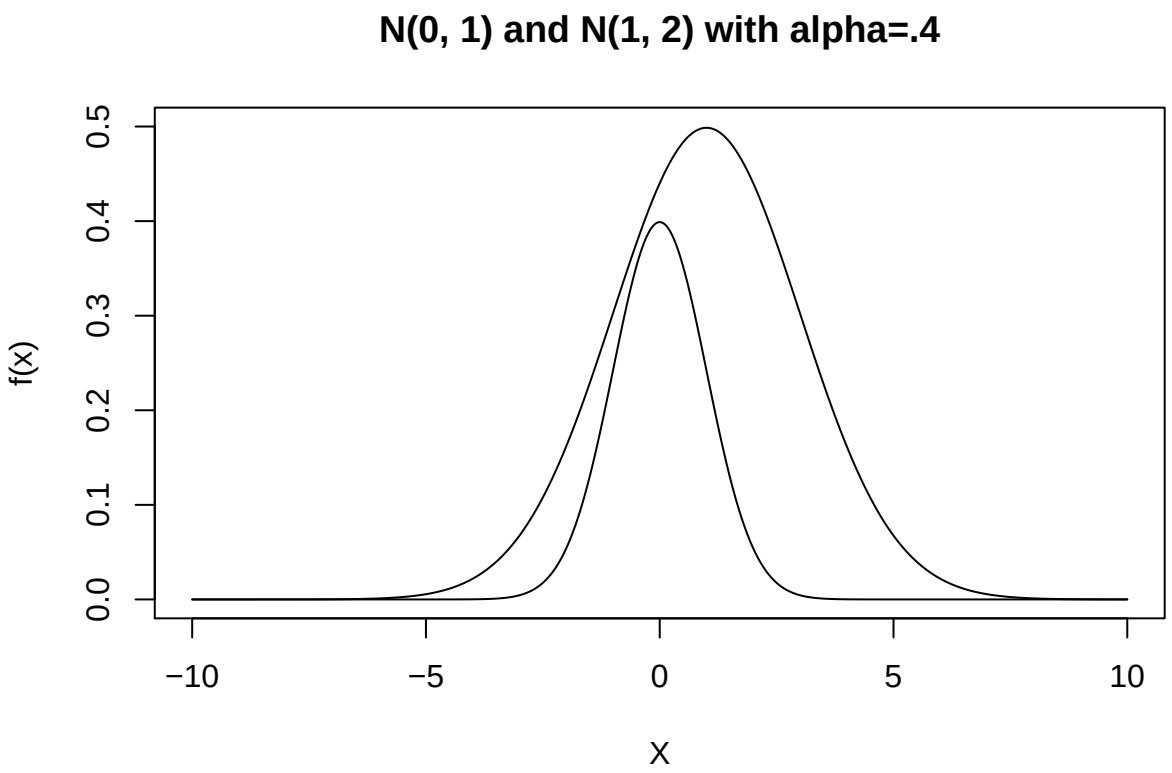
Therefore, our approximation is reasonably good. If we expanded the sample size, we could increase the accuracy of the estimate.

# Problem 3

## a)

We plot the two distributions given:

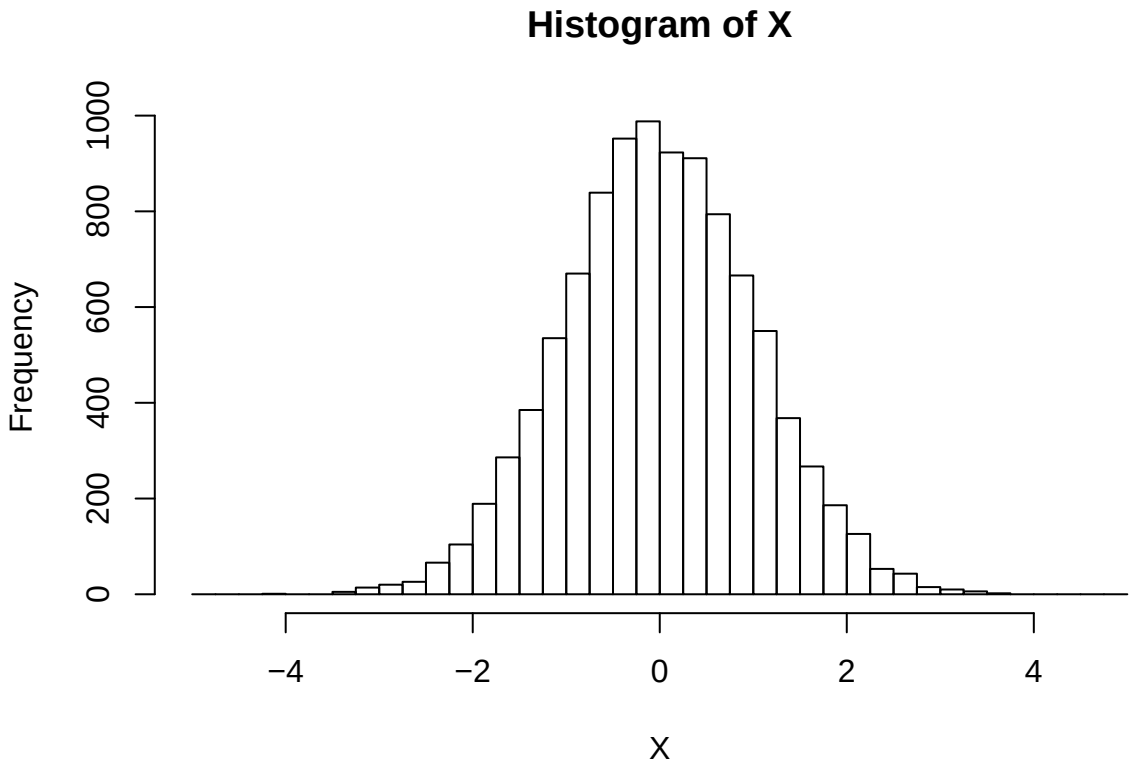
```
plot(seq(-10, 10, .01), dnorm(seq(-10, 10, .01), 0, 1), 'l',
     ylim=c(0, .5),
     main = "N(0, 1) and N(1, 2) with alpha=.4",
     xlab = 'X', ylab='f(x)')
lines(seq(-10, 10, .01), dnorm(seq(-10, 10, .01), 1, 2) / .4, 'l')
```




We can see that using $\alpha = .4$ satisfies our requirements for rejection sampling.

Next we generate a sample of 10000 points using rejection sampling.

```
n <- 0
total <- 0
X <- rep(0, 10000)
while (n < 10000){
  total <- total + 1
  Y <- rnorm(1, 1, 2)
  U <- runif(1, 0, 1)
  if (U <= dnorm(Y, 0, 1) / (dnorm(Y, 1, 2) / .4)){
    n <- n + 1
    X[n] = Y
  }
}
hist(X, breaks = seq(-5,5,.25))
```




```
print(paste0("It took ", total,
             " random draws to accumulate 10000 points, leading to efficiency ",
             round(n / total, 3)))
```

```
## [1] "It took 24684 random draws to accumulate 10000 points, leading to efficiency 0.405"
```

```
print(paste0("The mean of the sample is: ", mean(X)))
```

```
## [1] "The mean of the sample is: -0.00670776807783822"
```

```
print(paste0("The standard deviation of the sample is: ", sd(X)))
```

```
## [1] "The standard deviation of the sample is: 1.01795364958788"
```

We see that the mean is almost exactly 0 and the standard deviation is almost exactly 1.

## b)

Our choice for $\alpha$ was .4, which we derived by plotting the two functions and ensuring that the instrumental function was greater than the target function when divided by .4. Our envelope is therefore $e(x) = g(x)/.4$ where $g(x)$ is the density function for a normally distributed variable with mean 1 and standard deviation 2.

## c)

Generally, squeezed rejection sampling can help reduce the number of times that $f(x)$ needs to be computed. If it is expensive to compute, this can save time, as the squeezing function can be easy to compute. The squeezing function simply acts as a filter to accept points that are easy to determine as being less than $f(x)$ without computing $f(x)$.

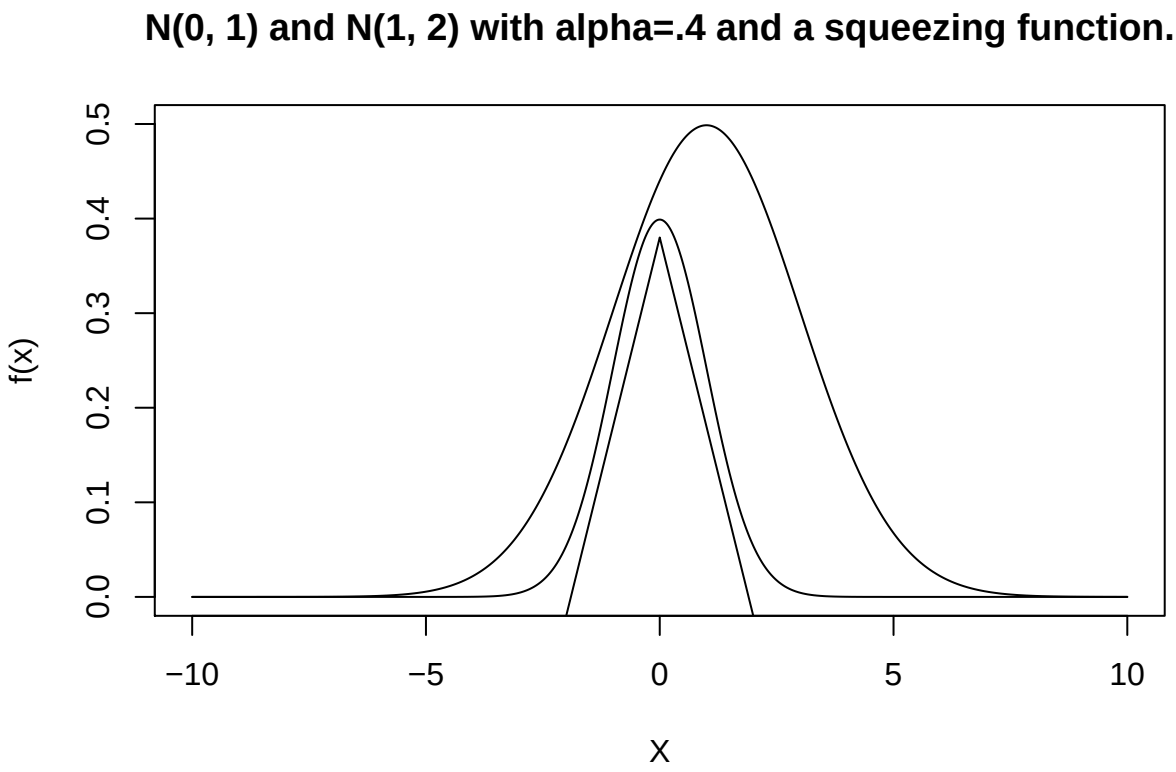
There isn't really any reason to use squeezed rejection sampling in this problem, as $f(x)$ is not computationally expensive to compute. Using the built in R function, it's a constant time evaluation.

However, if we were to want to use a squeezing function for this problem, we could use a piecewise function similar to the one in problem 2.

$$s(x) = \begin{cases} .38 + .2x & x < 0 \\ .38 - .2x & x \geq 0 \end{cases}$$

We plot the densities again with the squeezing function.

```
plot(seq(-10, 10, .01), dnorm(seq(-10, 10, .01), 0, 1), 'l',
     ylim=c(0, .5),
     main = "N(0, 1) and N(1, 2) with alpha=.4 and a squeezing function.",
     xlab = 'X', ylab='f(x)')
lines(seq(-10, 10, .01), dnorm(seq(-10, 10, .01), 1, 2) / .4, 'l')

s <- function(x) ifelse(x < 0, .38 + .2 * x, .38 - .2 * x)
lines(seq(-10, 10, .01), s(seq(-10, 10, .01)))
```




This could help, as we can evaluate the squeezing function easily, and use it to filter observations before evaluation $f(x)$.

Just for fun, I ran a quick time trial across a handful of values of $n$ (the number of time evaluating each of these functions).

```
n <- c(10, 100, 1000, 10000, 100000)
norm_times <- rep(0, length(n))
s_times <- rep(0, length(n))
for (i in 1:length(n)){
  x <- runif(n[i], -1, 1)
  norm_begin <- Sys.time()
  norm <- dnorm(x, mean=0, sd=1)
  norm_end <- Sys.time()
```

```
  s_begin <- Sys.time()
  s_ <- s(x)
  s_end <- Sys.time()

  norm_times[i] <-  norm_end - norm_begin
  s_times[i] <- s_end - s_begin
}
options(scipen = 1000000)
knitr::kable(data.frame(n=n, f_times = norm_times, s_times = s_times),
             col.names = c("n", "Time to Evaluate f(x)", "Time to Evaluate s(x)"))
```

| n | Time to Evaluate f(x) | Time to Evaluate s(x) |
|---|---|---|
| 10 | 0.0000079 | 0.0021620 |
| 100 | 0.0000081 | 0.0000219 |
| 1000 | 0.0000310 | 0.0001349 |
| 10000 | 0.0002561 | 0.0012081 |
| 100000 | 0.0023060 | 0.0217569 |

In practice, it ends up being faster to compute $f(x)$ for large samples, so we wouldn't use the squeezing function here. Not too surprising, as $f(x)$ here is not computationally expensive.