Homework 4

Zachary Lazerick

21 February 2023

1) From An Introduction to Statistical Computing by Voss. [Examples 1.27 & 1.28 in textbook] We can use a rejection sampler to generate samples $X \sim N(0,1)$ conditioned on $X \geq a$. One option is to repeat the following two steps until enough samples are output:

   - Generate $X \sim N(0,1)$ [use the built in rnorm() function]
   - If $X \geq a$, output X. The efficiency of this method depends on the value of a.

a) Write a simulation that is able to reproduce the table at the top of page 24, which shows the average number $E(N_a)$ of samples required to generate one output sample for different values of a, rounded to the nearest integer. Generate 1000 values of X for $a = 1, 2, 3$, and 4. [It will take a while…so DON'T TEST $a = 5$ or $a = 6$]

```
NormSim <- function(n, a) {
  counts <- c();
  for (i in 1:n) {
    X <- 0; counter <- 0
    while (X < a) {
      X <- rnorm(1, 0, 1)
      counter <- counter + 1
    }
    counts <- append(counts, counter)
  }
  return(counts)
}

a.1 <- NormSim(1000, 1); a.2 <- NormSim(1000, 2); a.3 <- NormSim(1000, 3); a.4 <- NormSim(1000, 4)
round(mean(a.1), digits = 0); round(mean(a.2), digits = 0)
```

```
## [1] 6
```

```
## [1] 44
```

```
round(mean(a.3), digits = 0); round(mean(a.4), digits = 0)
```

```
## [1] 736
```

```
## [1] 31869
```

b) For even moderate values of a, the number of samples is so large that this algorithm will not be practical. Instead, we can try sampling from the shifted exponential distribution using the following steps (see the textbook for more information):

- Generate $\hat{X} \sim Exp(\lambda = a)$ and $U \sim Unif(0, 1)$
- Let $X = \hat{X} + a$
- If $U \leq exp(-\frac{(X-a)^2}{2})$, output X

Note: λ represents the "rate" parameter in the rexp() function. Your simulation should be able to reproduce the table at the bottom of page 25, which shows the average number $E(M_a)$ of samples required to generate one output sample for different values of $a = 1, 2, \ldots, 6$, rounded to three significant digits.

```
NormSim.V2 <- function(n, a) {
  counts <- c();
  for (i in 1:n) {
    X <- 0; U <- 1; counter <- 0;
    while (U > exp(-(X - a)^2/2)) {
      Y <- rexp(1, a); U <- runif(1)
      X <- Y + a; counter <- counter + 1
    }
    counts <- append(counts, counter)
  }
  return(counts)
}

a.1 <- NormSim.V2(1000, 1); a.2 <- NormSim.V2(1000, 2); a.3 <- NormSim.V2(1000, 3)
a.4 <- NormSim.V2(1000, 4); a.5 <- NormSim.V2(1000, 5); a.6 <- NormSim.V2(1000, 6)
round(mean(a.1), digits = 3); round(mean(a.2), digits = 3); round(mean(a.3), digits = 3)
```

```
## [1] 1.483

## [1] 1.177

## [1] 1.094
```

```
round(mean(a.4), digits = 3); round(mean(a.5), digits = 3); round(mean(a.6), digits = 3)
```

```
## [1] 1.048

## [1] 1.034

## [1] 1.021
```

2) [Question E1.6]. Implement the rejection method from example 1.24 (also discussed in class) to generate 1000 samples from a half-normal distribution using Exp(1)-distributed proposals. Test your program by generating a histogram of the output and by comparing the histogram with the theoretical density of the half-normal distribution. [use the hint in question #8 from last week's assignment to plot the density function].

a) Modify the code you wrote for the previous question to generate values from the standard normal distribution. To do this, take the values and negate them with probability $\frac{1}{2}$.

```r
NormSim.V3 <- function(n) {
  X <- c(); counter <- 0
  while (length(X) < n) {
    Y <- rexp(1); U <- runif(1)
    f.Y <- exp((-(Y^2)/2) + Y - (1/2))
    if (U <= f.Y) {
      X <- append(X, Y)
      counter <- counter + 1
    }
    else {
      counter <- counter + 1
    }
  }
  for (j in 1:length(X)) {
    U <- runif(1)
    if (U < .5) {
      X[j] <- X[j] * -1
    }
  }
  mylist <- list('X' = X, 'Proposals' = counter)
  return(mylist)
}
```

b) Use this approach to generate 1000 values from the normal distribution with mean 5 and variance 10. Calculate the mean and variance of your simulated values to check your work.

```r
## Store Simulated Z-scores
HalfNorm <- NormSim.V3(1000)

## Convert Z-scores to N(5, 10)
HalfNorm.X <- HalfNorm$X * sqrt(10) + 5

mean(HalfNorm.X); var(HalfNorm.X)
```

```
## [1] 5.032108
```

```
## [1] 10.47188
```

c) Simulate 25000 standard normal RVs (mean 0, variance 1) using rejection sampling and exponential reference density g(x). Use these values to calculate $P(-1.96 < X < 1.96)$

```r
Norm <- NormSim.V3(25000)

counter <- 0
for (i in 1:length(Norm$X)) {
  if (Norm$X[i] > -1.96 & Norm$X[i] < 1.96) {
    counter <- counter + 1
  }
}

counter/length(Norm$X)
```

```
## [1] 0.95004
```

d) Simulate 25000 standard normal RVs using rejection sampling in conjunction with the Cauchy reference density $g(x) = \frac{1}{\pi(1+x^2)}$. Calculate the mean and variance of your sampled values to check your work and then find $P(-1.96 < X < 1.96)$.

Notes:

- See course notes from week 3 for how to draw samples from this distribution
- The maximum value of f(x)/g(x) occurs at $X = \pm 1$, giving $A = \sqrt{2\pi} e^{\frac{-1}{2}}$.

```r
CauchySim <- function(n) {
  X <- c(); counter <- 0; A <- sqrt(2 * pi)*(0.6065306597)
  while (length(X) < n) {
    ## Generate Cauchy Variables
    U <- runif(1); Y <- tan(pi*(U - (1/2)))
    f.Y <- (1/sqrt(2*pi))*exp(-(Y^2)/2); g.Y <- 1/(pi*(1 + Y^2))
    if (U <= (f.Y/(A*g.Y))) {
      X <- append(X, Y)
      counter <- counter + 1
    }
    else {
      counter <- counter + 1
    }
  }
  mylist <- list('X' = X, 'Proposals' = counter)
  return(mylist)
}
```

```r
Cauchy <- CauchySim(25000)

counter <- 0
for (i in 1:length(Cauchy$X)) {
  if (Cauchy$X[i] > -1.96 & Cauchy$X[i] < 1.96) {
    counter <- counter + 1
  }
}

counter/length(Cauchy$X)
```

```
## [1] 0.93768
```

e) Which of the two samplers led you to reject more values? Modify your code to add a counter for the number of simulated values that were needed to obtain 25000 standard normal RVs. Print out these values and say which sampler is more efficient.

```r
Norm$Proposals; Cauchy$Proposals
```

```
## [1] 32952
```

```
## [1] 35123
```

On average, the Cauchy rejection sampler required more proposals to generate 25000 standard normal RVs. Thus, the Normal sampler is more efficient.

3) [Question E1.13] Consider a uniform distribution on the semicircle.

a) What is the value of A needed for a rejection sampler that uses a $Unif(-1,1)$ proposal distribution to simulate values from a semicircle of radius 1 centered at the origin: $f(x) = \sqrt{1-x^2}$.

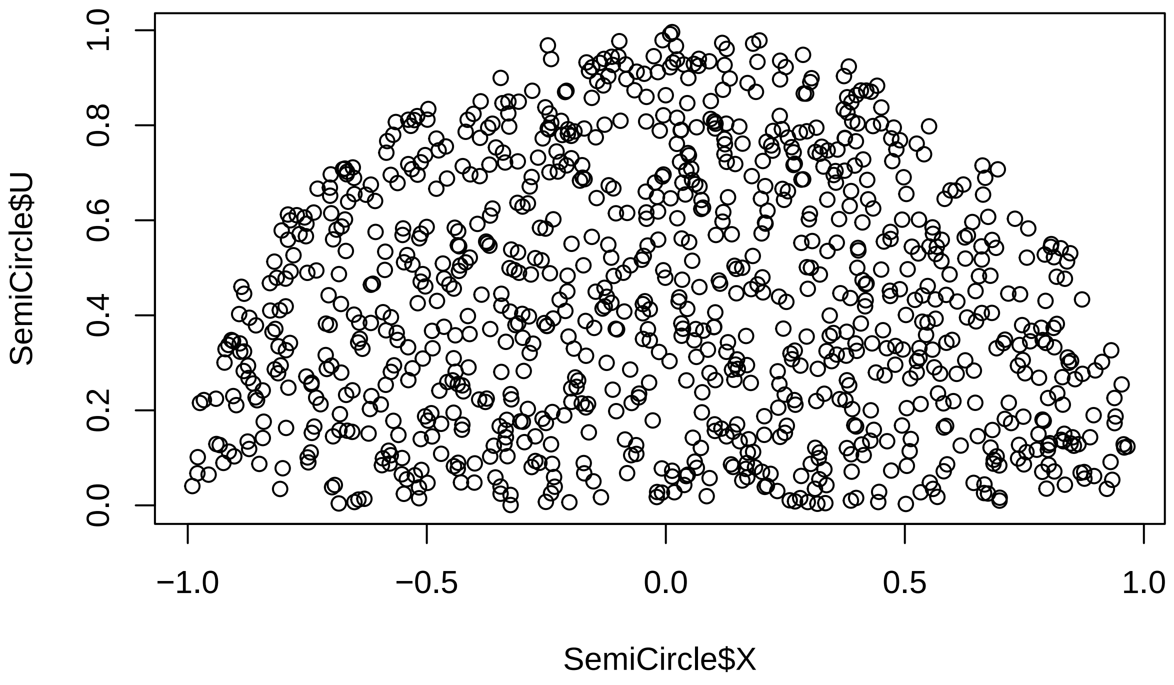
The value needed is $A = 1.0$.

$$\begin{aligned}
\frac{f(x)}{g(x)} &= \frac{(1-x^2)^{\frac{1}{2}}}{1} = (1-x^2)^{\frac{1}{2}} \\
&= \frac{1}{2} ln(1-x^2) \\
\frac{d}{dx}\frac{f(x)}{g(x)} &= \frac{1}{2}\frac{1}{(1-x^2)}(-2x) = \frac{-x}{(1-x^2)} \\
0 &= \frac{-x}{(1-x^2)} \longrightarrow x = 0 && \text{Set derivative equal to 0} \\
(1-0^2)^{\frac{1}{2}} &= 1 && \text{Plug solution back in}
\end{aligned}$$

b) Use a rejection sampler to generate 1000 samples from the uniform distribution on the semicircle, using the value of A from part (a). For each value you accept, save not only the value of X, but also the value of U you used for the acceptance probability. Create a scatterplot showing the random points (X,U). What do you see? How many proposals were needed to generate the 1000 samples?

```r
SemiRejection <- function(n) {
  X <- c(); U <- c(); counter <- 0
  while (length(X) < n) {
    Unif <- runif(1); Y <- runif(1, -1, 1)
    if (Unif <= (1 - Y^2)^(1/2)) {
      X <- append(X, Y); U <- append(U, Unif)
      counter <- counter + 1
    }
    else {
      counter <- counter + 1
    }
  }
  mylist <- list('X' = X, 'U' = U, 'Proposals' = counter)
  return(mylist)
}
```

```r
## Create Sample
SemiCircle <- SemiRejection(1000)

## Scatterplot
plot(SemiCircle$X, SemiCircle$U)
```



```r
## How many proposals needed?
SemiCircle$Proposals
```

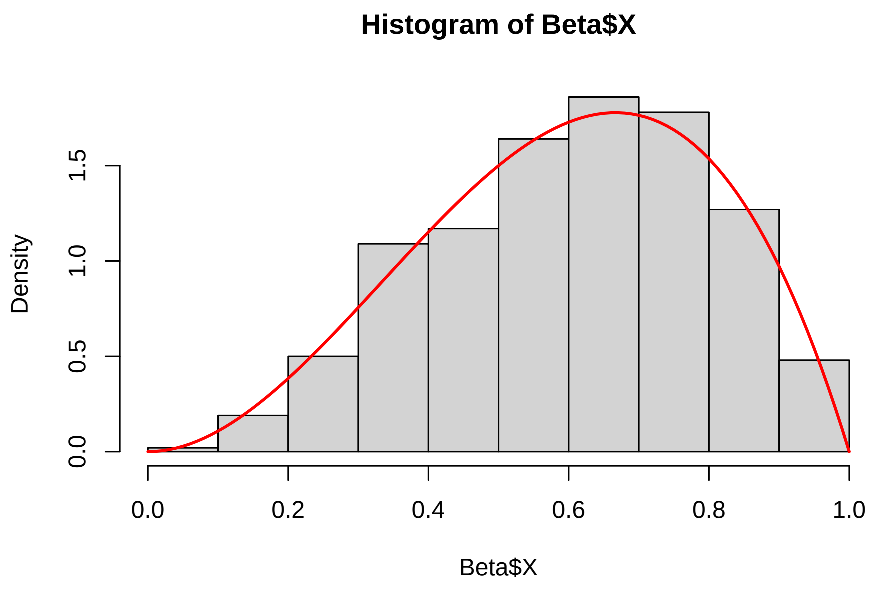
```
## [1] 1279
```

4) From Statistical Computing with R by Rizzo. [Question 3.7 in textbook] Write a function to generate a random sample of size n from the $Beta(a, b)$ distribution by the acceptance-rejection method.

a) Specifically, generate a random sample of size 1000 from the $Beta(3, 2)$ distribution. Graph the histogram of the sample with the theoretical $Beta(3, 2)$ density superimposed [use the R function dbeta(x, a, b) together with the hint in question #8 from last week's assignment].

```r
BetaSim <- function(n, alpha, beta) {
  X <- c(); counter <- 0; Y <- 0; A = (16/9)
  while (length(X) < n) {
    Y <- runif(1); U <- runif(1)
    f.Y <- 12*Y^(alpha - 1)*(1-Y)^(beta - 1)
    if (U <= (f.Y/A)) {
      X <- append(X, Y); counter <- counter + 1
    }
    else {
      counter <- counter + 1
    }
  }
  mylist = list('X' = X, 'Proposals' = counter)
  return(mylist)
}
```

```r
Beta <- BetaSim(1000, 3, 2)

hist(Beta$X, freq = FALSE)
x <- seq(0, 1, .01); DBETA <- dbeta(x, 3, 2)
lines(x, DBETA, col = "red", lwd = 2)
```



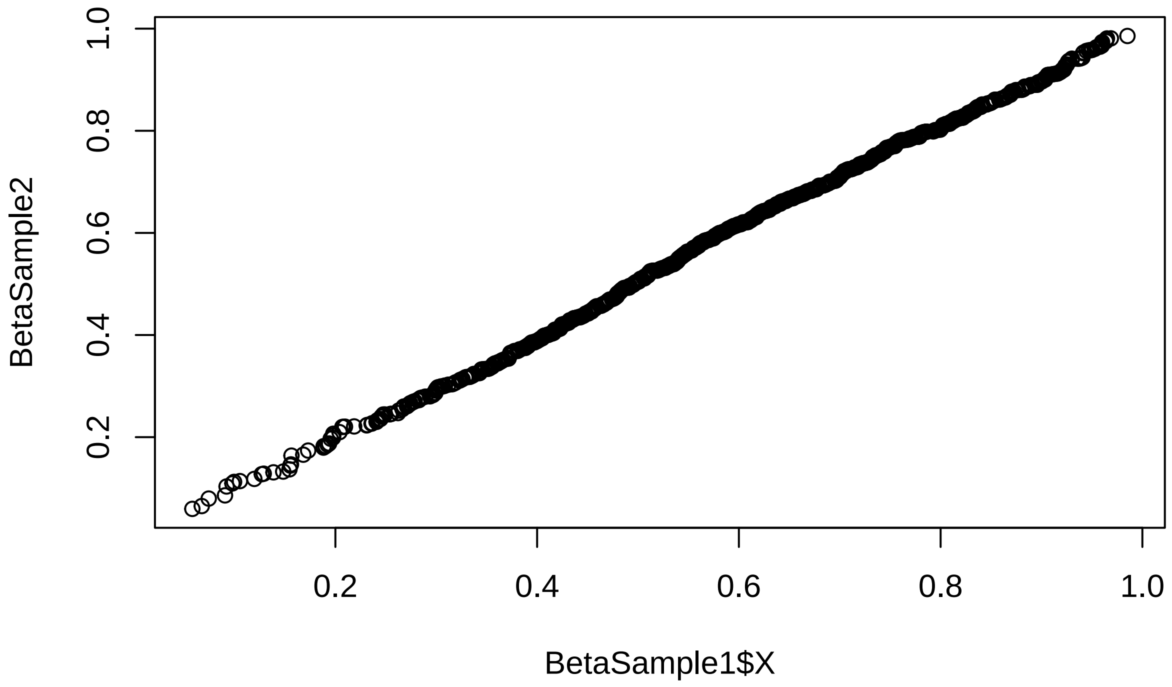
b) Find $P(X > 1/2)$.

```r
counter <- 0
for (i in length(Beta$X)) {
  if (Beta$X[i] > .5) {
    counter <- counter + 1
  }
}

## Report Probability
counter/length(Beta$X)
```

```
## [1] 0.001
```

c) Another way to compare two probability distributions is to make a quantile-quantile plot. If the two distributions are the same, you should see a straight line. Use your sample from part (a), together with a second sample drawn from the R function rbeta(n,a,b), and run the command qqplot(sample1, sample2). How did your sampler do?

```r
BetaSample1 <- BetaSim(1000, 3, 2)
BetaSample2 <- rbeta(1000, 3, 2)

qqplot(BetaSample1$X, BetaSample2)
```



Overall, the sampler did well. The Q-Q plot shows roughly a straight line, aside from a few deviations in the X < .2 range. It certainly performed better than expected.

d) Compare the performance of the Beta generator and the R generator rbeta in terms of their runtime. Fix the parameters $a = 3, b = 2$ and time each generator on 1000 iterations with sample size 500. Are the results different for different choices of a and b? Why might this be the case? [Use the R function system.time()]

```
system.time(BetaSim(1000, 3, 2)); system.time(dbeta(1000, 3, 2))
```

```
##    user  system elapsed
##    0.01    0.00    0.01
```

```
##    user  system elapsed
##       0       0       0
```

Overall, my Beta generator performed better than expected. I thought that it would be significantly slower than the built-in rbeta() function. However, for small sample sizes I can see how this slightly slower performance can compound. Despite multiple combinations of a and b, my Beta generator did not outpace the built-in function. This is probably due to the optimization of the buil-in functions to reduce runtime as low as possible.

5) From Simulation by Ross, Chapter 5. [Question 6 in textbook] Let X be an exponential random variable with mean 1. Give an efficient algorithm for simulating a random variable whose distribution is the conditional distribution of X given that $X < 0.05$. That is, its density function is

$$f(x) = \frac{e^{-x}}{1 - e^{-0.05}} 0 < x < 0.05$$

Generate 1000 such variables and use them to estimate $E[X|X < 0.05]$. (Note: The true value of $E[X|X < 0.05]$ is 0.02479). Hint: Simulating an Exp(1) RV and rejecting all values larger than 0.05 is not the most efficient way to solve this problem. Try to think of a better approach.

```r
ExpSim <- function(n) {
  X <- c();
  while (length(X) < n) {
    U <- runif(1)
    Y <- log((1 - U*(1 - exp(.05))))
    X <- append(X, Y)
  }
  return(X)
}

Exp <- ExpSim(1000)

mean(Exp)
```

```
## [1] 0.0256705
```

6) [Question 10 in textbook] A casualty insurance company has 1000 policyholders, each of whom will independently present a claim in the next month with probability 0.05. Assuming that the amount of the claims made are independent exponential random variables with mean \$ \$ 800\$, use simulation to estimate the probability that the sum of these claims exceeds \$ \$ 50,000\$.

```r
Insurance <- function(n) {
  Total_Claims <- c()
  for (i in 1:n) {
    Claims <- 0;
    U <- runif(1000); E <- rexp(1000, 1/800)
    for (j in 1:1000) {
      if (U[j] <= 0.05) {
      Claims <- Claims + E[j]
      }
    }
    Total_Claims <- append(Total_Claims, Claims)
  }
  return(Total_Claims)
}

Claims <- Insurance(1000)

count <- 0
for (i in 1:length(Claims)) {
  if (Claims[i] > 50000) {
    count <- count + 1
  }
}
count / 1000
```

```
## [1] 0.097
```

7) [Question 17 in textbook] Use rejection sampling with a uniform proposal distribution to generate a random variable having density function:

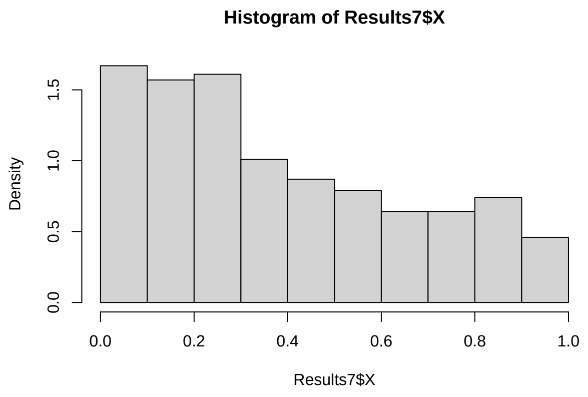
$$f(x) = \frac{1}{4} + 2x^3 + \frac{5}{4}x^4 0 < x < 1$$

Identify the value of X that maximizes the ratio $\frac{f(x)}{g(x)}$ and give the corresponding value of A used in your rejection sampler. Then make a histogram of 1000 realizations of this random variable. How many values did you actually have to simulate to get 1000 values of this random variable?

The value that maximizes the function is $X = 0$. The corresponding value for the rejection sampler is $A = 1/4$.

```r
RejectionSampler <- function(n) {
  X <- c(); counter <- 0
  while (length(X) < n) {
    Y <- runif(1); U <- runif(1)
    if ((U <= (1/4) + 2*Y^3 + (5/4) * Y^4)/(1/4)) {
      counter <- counter + 1
      X <- append(X, U)
    }
    counter <- counter + 1
  }
  mylist <- list('X'= X, 'counter' = counter)
  return(mylist)
}
```

```r
Results7 <- RejectionSampler(1000)

## Histogram
hist(Results7$X, freq = FALSE)
```



```r
## Number of Samples Generated for 1000 Accepted Samples
Results7$counter
```

```
## [1] 2641
```

8) [Question 20 in textbook] Use the rejection method to find an efficient way to generate a random variable having density function:

$$f(x) = \frac{1}{2}(1+x)e^{-x} 0 < x < \infty$$

What function did you choose for $g(x)$? Identify the value of X that maximizes the ratio $\frac{f(x)}{g(x)}$ and give the corresponding value of c used in your rejection sampler. Then make a histogram of 1000 realizations of this random variable and print out the mean and variance of X. Hint: This function has elements of both an exponential and a gamma random variable, so your proposal density should take the form of an exponential distribution. [See example 5e from Ross, Chapter 5 for a hint.

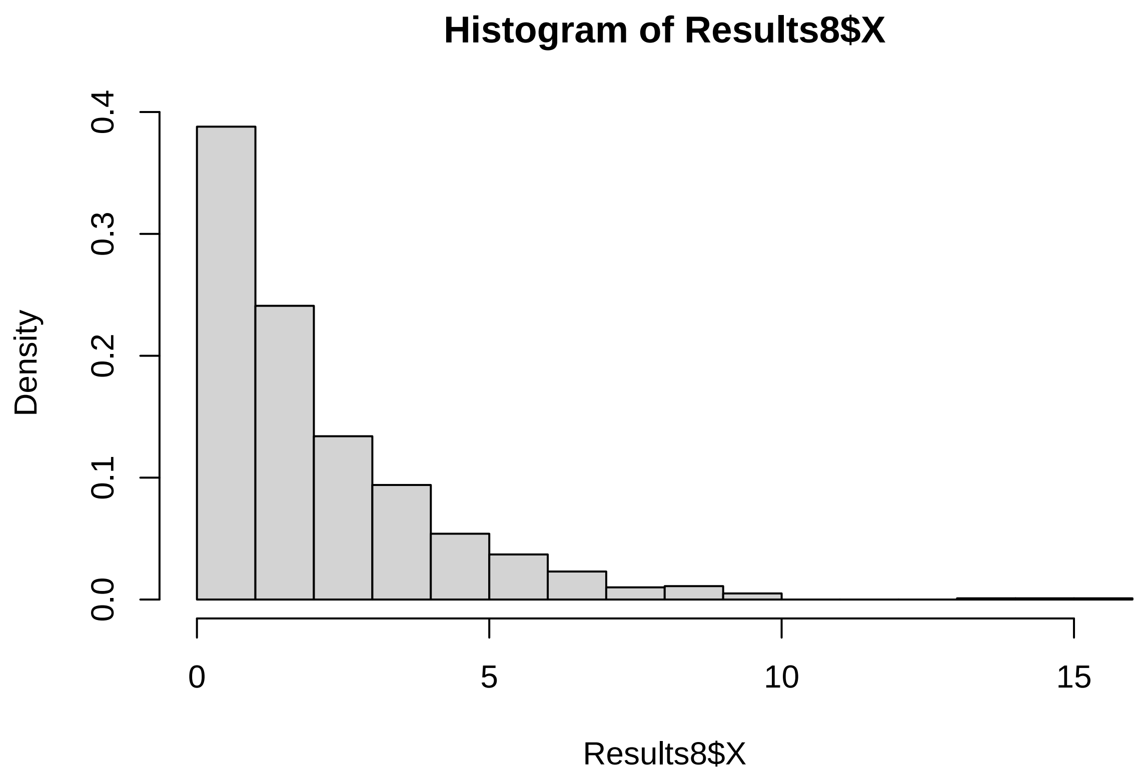
I chose to use an exponential function $g(x) = \frac{1}{2}e^{\frac{-x}{2}}$ since f(x) is not bounded above. The value that maximizes my $\frac{f(x)}{g(x)}$ is $X = 1$. This value of X yields a corresponding constant of $C = 2e^{1/2}$. That being said, the acceptance probability must be less than or equal to

$$\begin{aligned}\frac{f(x)}{Cg(x)} &= \frac{\frac{1}{2}(1+x)e^{-x}}{e^{\frac{1}{2}}\frac{1}{2}e^{\frac{-1}{2}}} \\ &= \frac{(1+x)e^{-x}}{2e^{\frac{-1-x}{2}}} \\ &= \frac{(1+x)}{2}e^{\frac{-1}{2}(x-1)}\end{aligned}$$

Now to implement the rejection sampler:

```r
RejectionSampler8 <- function(n) {
  X <- c(); counter <- 0
  while (length(X) < n ) {
    Y <- rexp(1, rate = (1/2)); U <- rexp(1)
    f.cG.Y <- ((1+Y)/2)*exp(-.5*(Y-1))
    if (U <- f.cG.Y) {
      X <- append(X, Y); counter <- counter + 1
    }
    else {
      counter <- counter + 1
    }
  }
  mylist <- list('X' = X, 'Proposals' = counter)
  return(mylist)
}
```

```r
Results8 <- RejectionSampler8(1000)

hist(Results8$X, freq = FALSE)
x <- seq(0, 10, .5); equation <- (1/2) * (1 + x) * exp(x)
lines(x, equation, col = 'red', lwd = 2)
```



```r
mean(Results8$X); var(Results8$X)
```

```
## [1] 2.017231
```

```
## [1] 4.032161
```