Homework 6 Solutions

```
set.seed(0)
```

Problem 1: Inverse Transform Method

Consider the *Cauchy random variable* X with probability density function

$$f_X(x) = \frac{1}{\pi}\frac{1}{(1+x^2)}, \quad -\infty < x < \infty.$$

(1) [5 points] Let U be a uniform random variable over [0,1]. Find a transformation of U that allows you to simulate X from U.

Solution:

First note that

$$\int_{-\infty}^{x} \frac{1}{\pi}\frac{1}{(1+t^2)}dt = \cdots = \frac{1}{\pi}\arctan(x) + \frac{1}{2}.$$

Then

$$\frac{1}{\pi}\arctan(X) + \frac{1}{2} = U \longrightarrow X = \tan\left(\pi(U - 1/2)\right)$$

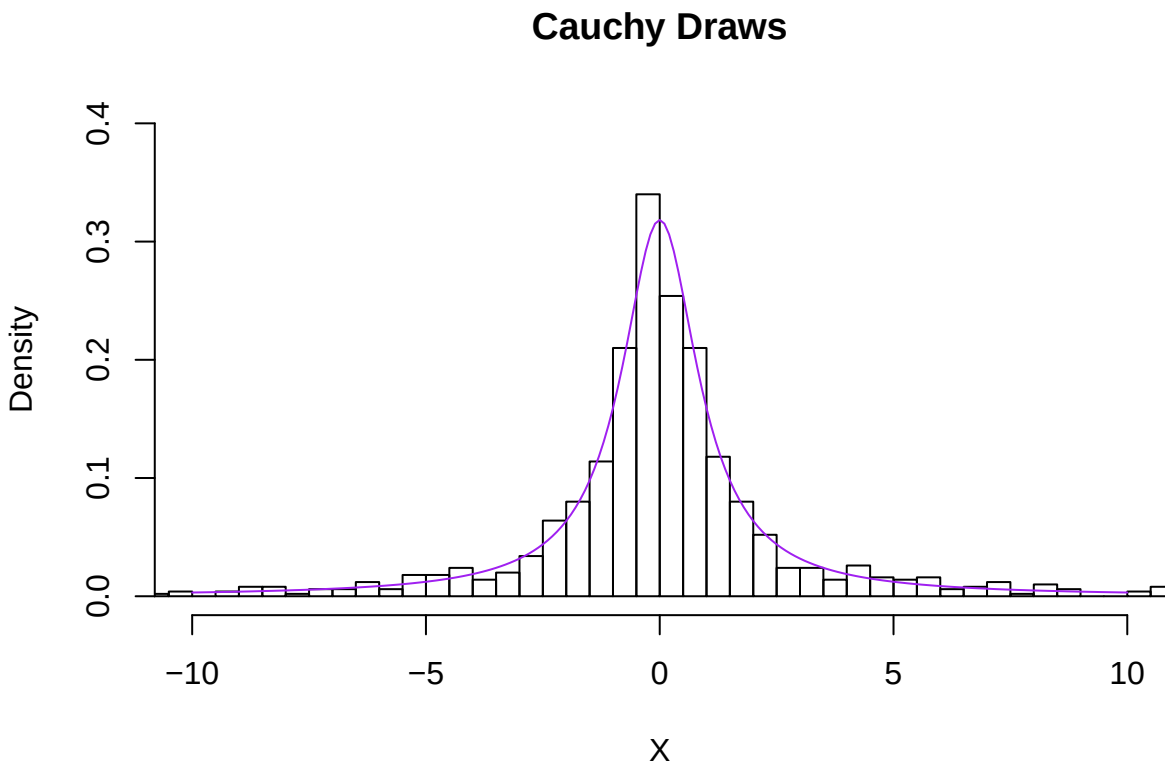
(2) [5 points] Write a **R** function called **cauchy.sim** that generates n simulated Cauchy random variables. The function should have the single input **n** and should use the inverse-transformation from Part 1. Test your function using 10 draws.

```
cauchy.sim <-function(n) {

  U <- runif(n)
  return(tan(pi*(U-1/2)))

}
cauchy.sim(10)
```

```
##  [1]  2.9723830 -0.9070131 -0.4248394  0.2329576  3.3710605 -1.3611982
##  [7]  3.0255173  5.6954298  0.5530230  0.4294381
```

(3) [5 points] Using your function **cauchy.sim**, simulate 1000 random draws from a Cauchy distribution. Store the 1000 draws in the vector **cauchy.draws**. Construct a histogram of the simulated Cauchy random variable with $f_X(x)$ overlaid on the graph. **Note:** when plotting the density curve over the histogram, include the argument **prob = T**.

```
cauchy.draws <- cauchy.sim(1000)
hist(cauchy.draws,prob = T,main="Cauchy Draws",xlab="X",breaks=10000,xlim=c(-10,10),ylim=c(0,.4))
x <- seq(-10,10,by=.1)
lines(x,1/(pi*(1+x^2)),col="purple")
```



Problem 2: Reject-Accept Method

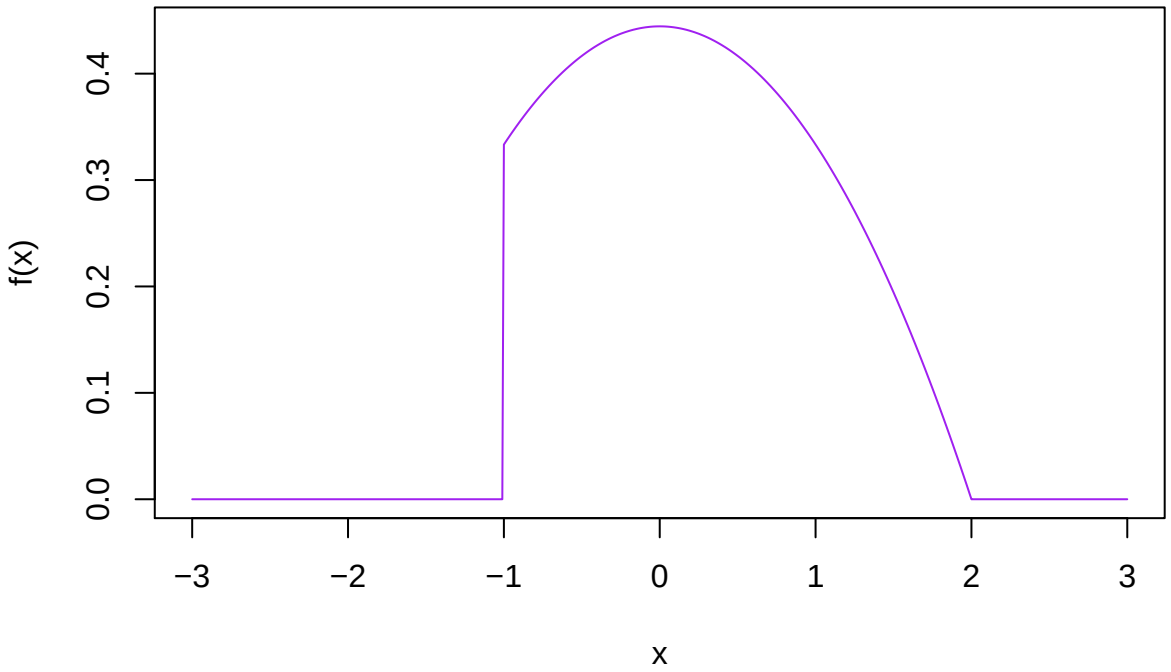
Let random variable X denote the temperature at which a certain chemical reaction takes place. Suppose that X has probability density function

$$f(x) = \begin{cases} \frac{1}{9}(4 - x^2) & -1 \leq x \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

Perform the following tasks:

(4) [5 points] Write a function `f` that takes as input a vector `x` and returns a vector of `f(x)` values. Plot the function between -3 and 3. Make sure your plot is labeled appropriately.

```
f <- function(x) {
  ifelse((x < -1 | x > 2), 0, (1/9)*(4 - x^2))
}
x <- seq(-3, 3, by = .01)
plot(x,f(x),main="Target Distribution",ylab="f(x)",type="l",col="purple")
```



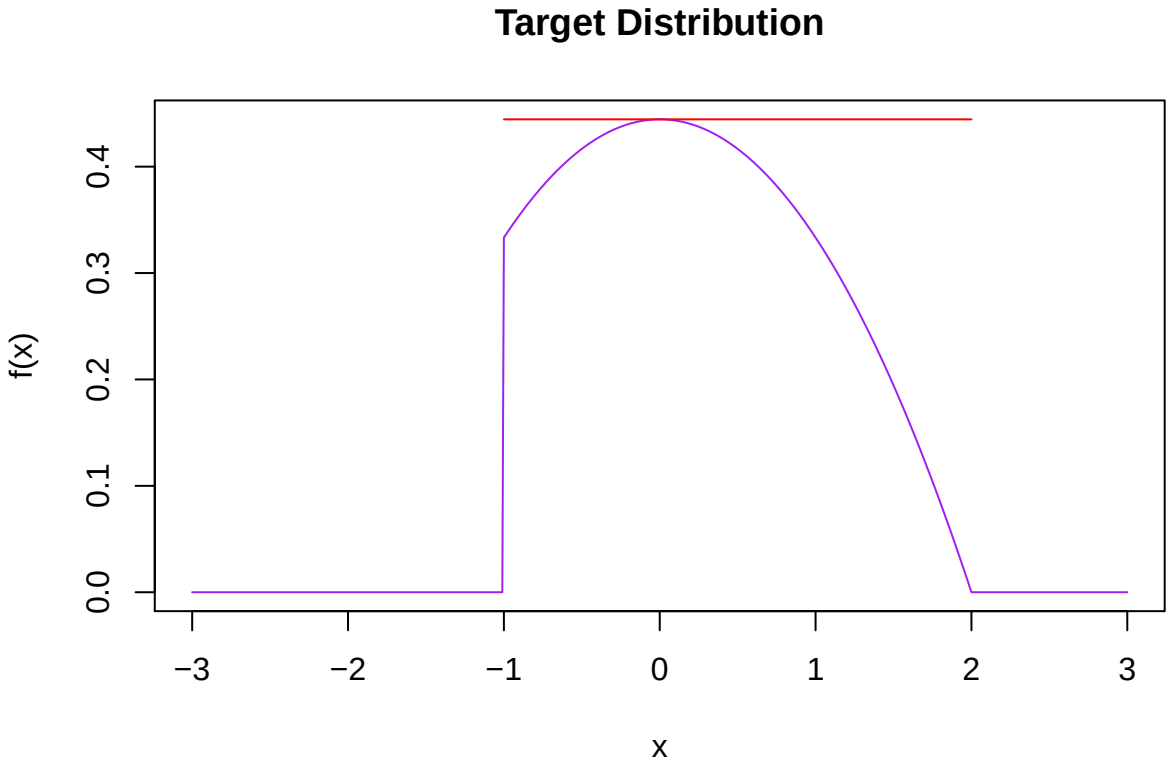
(5) [5 points] Determine the maximum of $f(x)$ and find an envelope function $e(x)$ by using a uniform density for $g(x)$ and setting $\alpha = 1/\max_x\{f(x)\}$ as in class. Write a function `e` which takes as input a vector `x` and returns a vector of `e(x)` values.

Note that

$$f'(x) = \begin{cases} -\frac{2x}{9} & -1 \leq x \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

Setting the above equal to 0 we find that the maximum occurs at $x = 0$ and $f(0) = 4/9$.

```
f.max <- 4/9
e <- function(x) {
  ifelse((x < -1 | x > 2), Inf, f.max)
}
plot(x,f(x),main="Target Distribution",ylab="f(x)",type="l",col="purple")
lines(x,e(x),col="red")
```

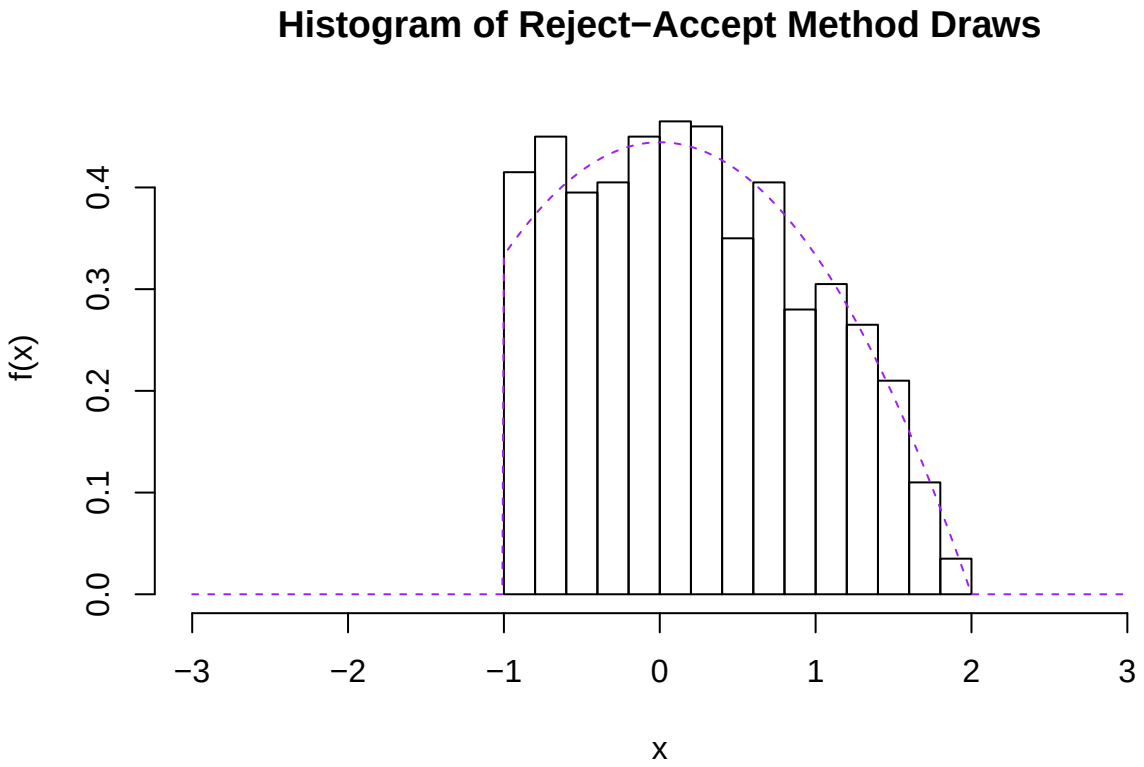


(6) [5 points] Using the *Accept-Reject Algorithm*, write a program that simulates 1000 draws from the probability density function $f(x)$ from Equation **??**.

```
n.samps <- 1000   # Samples desired
n       <- 0      # counter for number samples accepted
samps   <- numeric(n.samps) # initialize the vector of output
while (n < n.samps) {
  y <- runif(1, min = -1, max = 2)    # random draw from g
  u <- runif(1)
  if (u < f(y)/e(y)) {
    n        <- n + 1
    samps[n] <- y
  }
}
```

(7) [5 points] Plot a histogram of your simulated data with the density function f overlaid in the graph. Label your plot appropriately.

```
hist(samps, prob = T, ylab = "f(x)",breaks=20, xlab = "x", main = "Histogram of Reject-Accept Method Dra
lines(x, f(x), lty = 2,col="purple")
```



Problem 3: Reject-Accept Method Continued

Consider the standard normal distribution X with probability density function

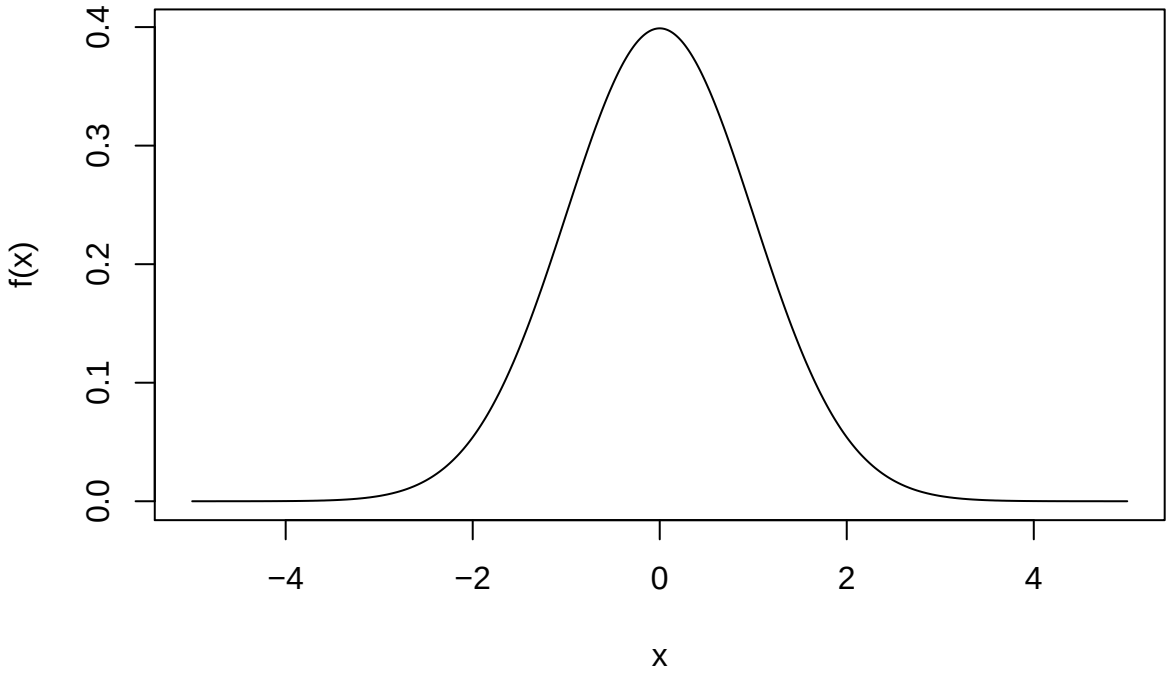
$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\big(-\frac{1}{2}x^2\big), \quad -\infty < x < \infty.$$

In this exercise, we will write a function named **normal.sim** that simulates a standard normal random variable using the **Accept-Reject Algorithm**.

Perform the following tasks:

(8) [5 points] Write a function **f** that takes as input a vector **x** and returns a vector of **f(x)** values. Plot the function between -5 and 5. Make sure your plot is labeled appropriately.

```
f <- function(x) {

  return(1/sqrt(2*pi)*exp(-x^2/2))

}
x <- seq(-5,5,by=.01)
plot(x,f(x),main="Standard Normal Density",ylab="f(x)",type="l")
```



(9) [5 points] Let the known density g be the Cauchy density defined by pdf

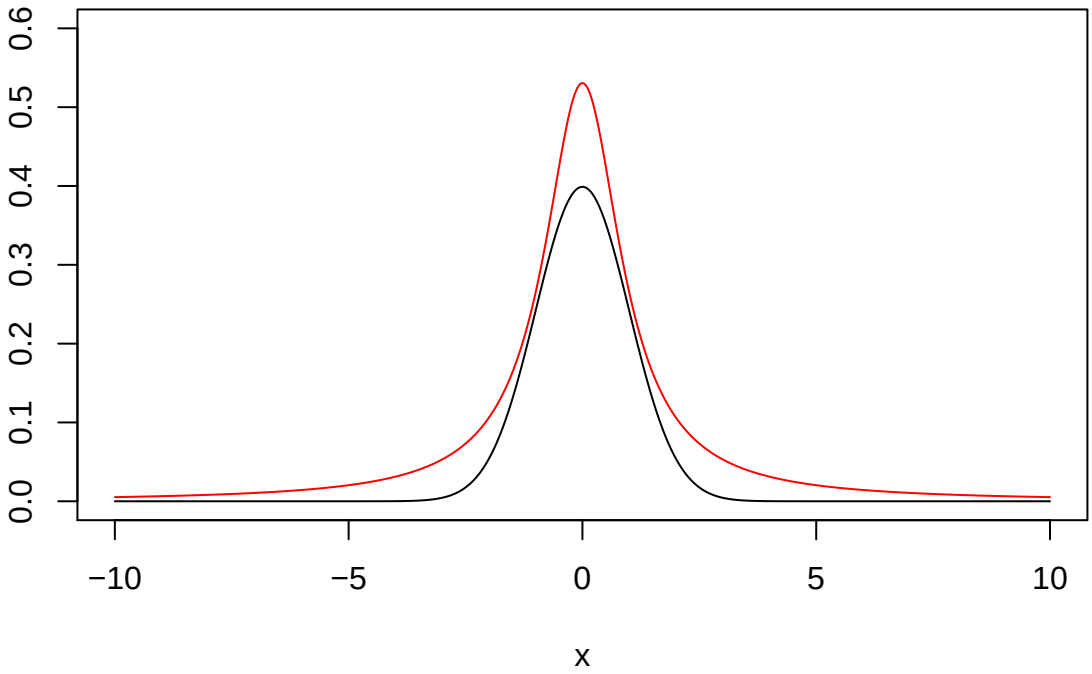
$$g(x) = \frac{1}{\pi}\frac{1}{(1+x^2)}, \quad -\infty < x < \infty.$$

Write a function **e** that takes as input a vector **x** and constant **alpha** $(0 < \alpha < 1)$ and returns a vector of **e(x)** values. The envelope function should be defined as $e(x) = g(x)/\alpha$.

```
e <- function(x,alpha=1) {

  return(1/(alpha*pi*(1+x^2)))

}
```

(10) [5 points] Determine a "good" value of α. To show your solution, plot both $f(x)$ and $e(x)$ on the interval $[-10, 10]$.

```
x <- seq(-10,10,by=.01)
plot(x,f(x),main="Cauchy Envelope Over5Standard Normal",ylab="",type="l",ylim=c(0,.60))
lines(x,e(x,alpha=.60),main="Standard Normal Density",ylab="f(x)",col="red")
```




```
all(f(x)<e(x,alpha=.65))
```

```
## [1] TRUE
```

A "good" value of α is somewhere near .6.

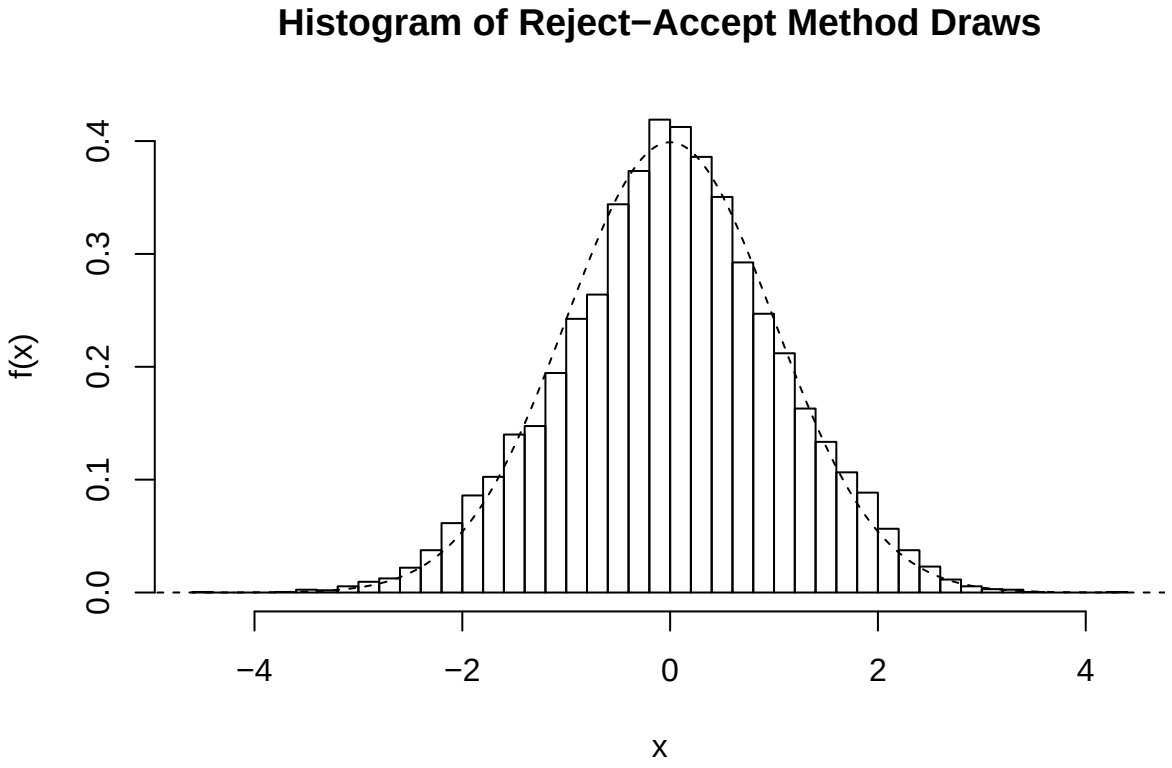
(11) [5 points] Write a function named **normal.sim** that simulates **n** standard normal random variables using the **Accept-Reject Algorithm**. The function should also use the **Inverse-Transformation** from Part 1. Test your function using **n=10** draws.

```
normal.sim <- function(n.samps) {
  n       <- 0      # counter for number samples accepted
  samps   <- numeric(n.samps) # initialize the vector of output
  while (n < n.samps) {
    y <- cauchy.sim(1) # random draw from g
    u <- runif(1)
    if (u < f(y)/e(y)) {
      n        <- n + 1
      samps[n] <- y
    }
  }
  return(samps)
}
normal.sim(10)
```

```
##  [1] -0.49721504  1.58341903 -0.27305100  0.02052886 -2.38300486
##  [6]  2.08639932  0.42237551 -0.96118393 -1.22534003 -1.68994926
```

(12) [5 points] Using your function **normal.sim**, simulate 10,000 random draws from a standard normal distribution. Store the 10,000 draws in the vector **normal.draws**. Construct a histogram of the simulated standard normal random variable with $f(x)$ overlaid on the graph. **Note:** when plotting the density curve over the histogram, include the argument **prob = T**.

```
normal.draws <- normal.sim(10000)
hist(normal.draws, prob = T, ylab = "f(x)",breaks=50, xlab = "x", main = "Histogram of Reject-Accept Me
lines(x, f(x), lty = 2)
```



Part 3: Simulation with Built-in R Functions

Consider the following "random walk" procedure:

a. Start with $x = 5$
b. Draw a random number r uniformly between -2 and 1.
c. Replace x with $x + r$
d. Stop if $x \leq 0$
e. Else repeat

Perform the following tasks:

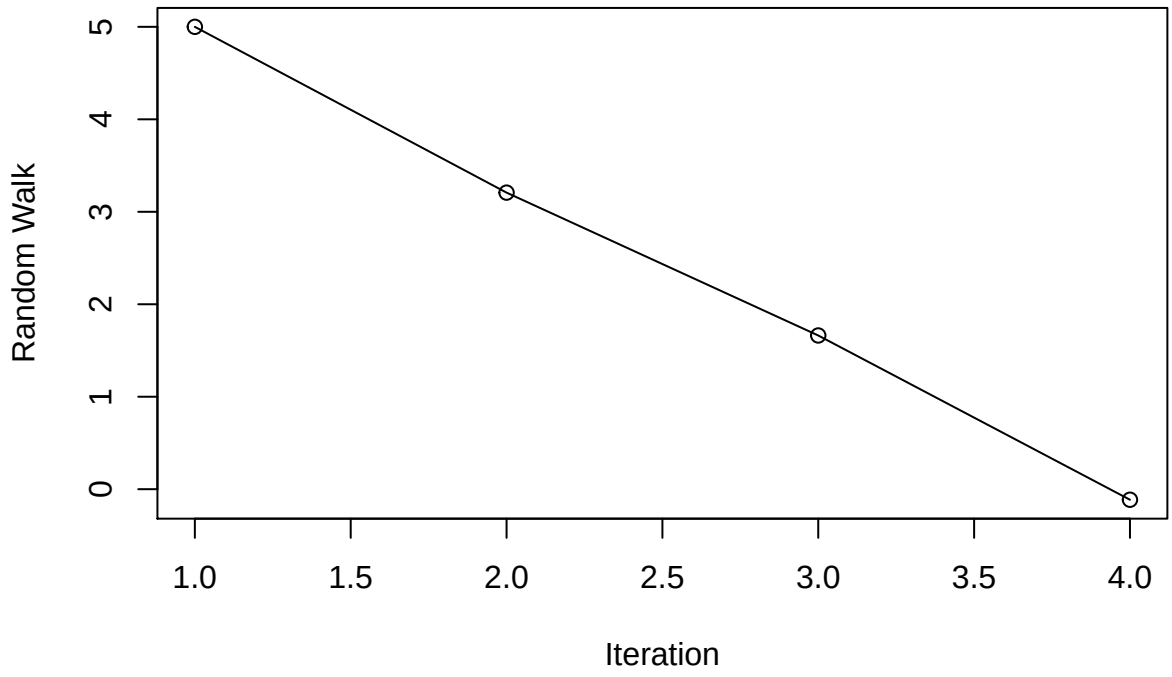
(13) [5 points] Write a `while()` loop to implement this procedure. Importantly, save all the positive values of x that were visited in this procedure in a vector called `x.vals`, and display its entries.

```
x.vals <- 5
i <- 1
while(x.vals[i] > 0) {
  r <- runif(1, min = -2, max = 1)
  x.vals <- c(x.vals, x.vals[i] + r)
  i <- i+1
}
x.vals
```

```
## [1]  5.0000000  3.2071865  1.6623219 -0.1140127
```

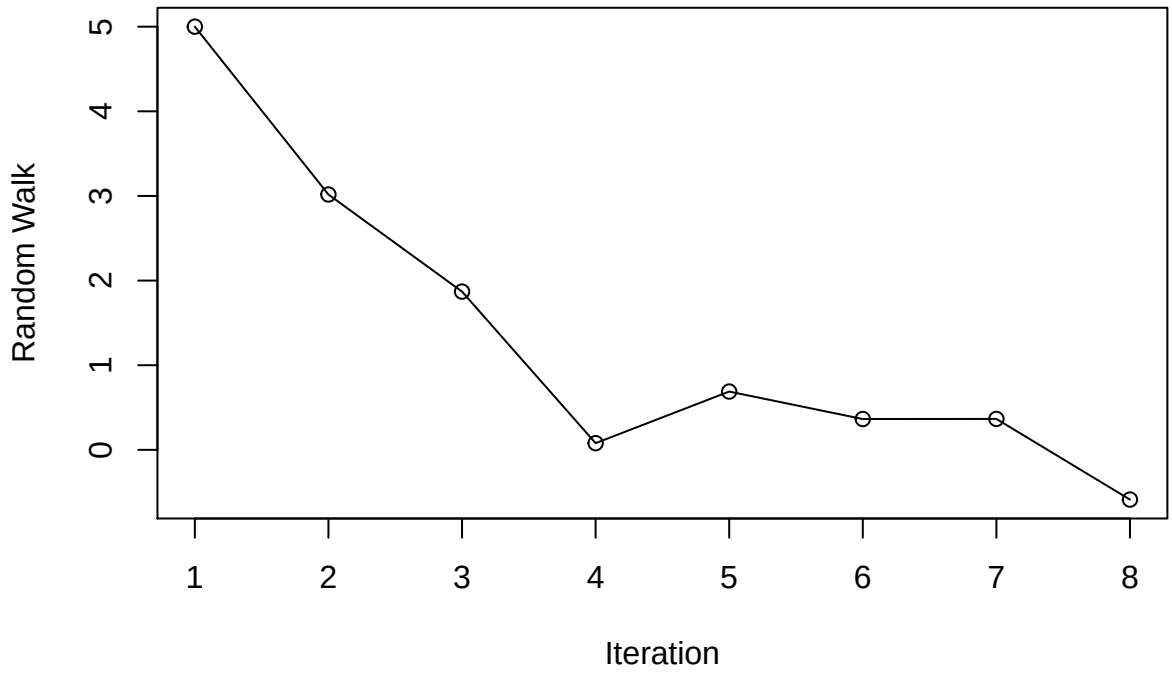
(14) [5 points] Produce a plot of the random walk values `x.vals` from above versus the iteration number. Make sure the plot has an appropriately labeled x-axis and and y-axis. Also use `type="o"` so that we can see both points and lines.

```
plot(1:length(x.vals), x.vals, type = "o", xlab = "Iteration", ylab = "Random Walk")
```



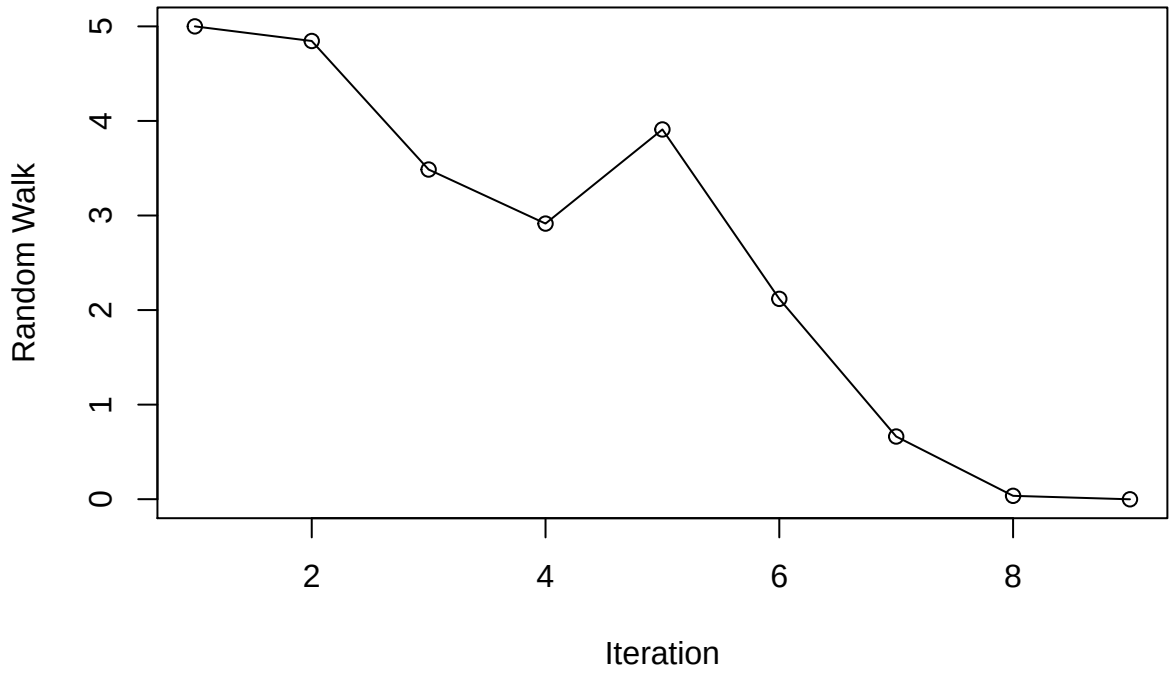
(15) [5 points] Write a function `random.walk()` to perform the random walk procedure that you implemented in question (9). Its inputs should be: `x.start`, a numeric value at which we will start the random walk, which takes a default value of 5; and `plot.walk`, a boolean value, indicating whether or not we want to produce a plot of the random walk values `x.vals` versus the iteration number as a side effect, which takes a default value of `TRUE`. The output of your function should be a list with elements: `x.vals`, a vector of the random walk values as computed above; and `num.steps`, the number of steps taken by the random walk before terminating. Run your function twice with the default inputs, and then twice times with `x.start` equal to 10 and `plot.walk = FALSE`.

```
random.walk <- function(x.start = 5, plot.walk = TRUE) {
  x.vals <- 5
  i <- 1
  while(all(x.vals[i] > 0)) {
    r <- runif(1, min = -2, max = 1)
    x.vals <- c(x.vals, x.vals[i] + r)
    i <- i+1
  }
  if (plot.walk) {plot(1:length(x.vals), x.vals, type = "o", xlab = "Iteration", ylab = "Random Walk")}
  return(list(x.vals = x.vals, num.steps = i))
}

random.walk()
```




```
## $x.vals
## [1]  5.00000000  3.01735277  1.86972172  0.07893182  0.68878691  0.36417869
## [7]  0.36533533 -0.58770197
##
## $num.steps
## [1] 8
```

```
random.walk()
```




```
## $x.vals
## [1]  5.000000000  4.845427555  3.486111703  2.914671320  3.910030388
## [6]  2.118317218  0.662596828  0.035573776 -0.001093295
```

```
##
## $num.steps
## [1] 9
```

```
random.walk(x.start = 10, plot.walk = FALSE)
```

```
## $x.vals
##  [1]  5.0000000  5.7114786  5.0586144  3.7207443  3.6068736  2.1901089
##  [7]  1.9857527  0.6998041  0.9923582 -0.8100799
##
## $num.steps
## [1] 10
```

```
random.walk(x.start = 10, plot.walk = FALSE)
```

```
## $x.vals
##  [1]  5.0000000  5.0974495  3.5438568  3.0221654  3.6899293  2.6229215
##  [7]  3.4740686  2.8401605  1.4167094  0.5729083 -1.2392308
##
## $num.steps
## [1] 11
```

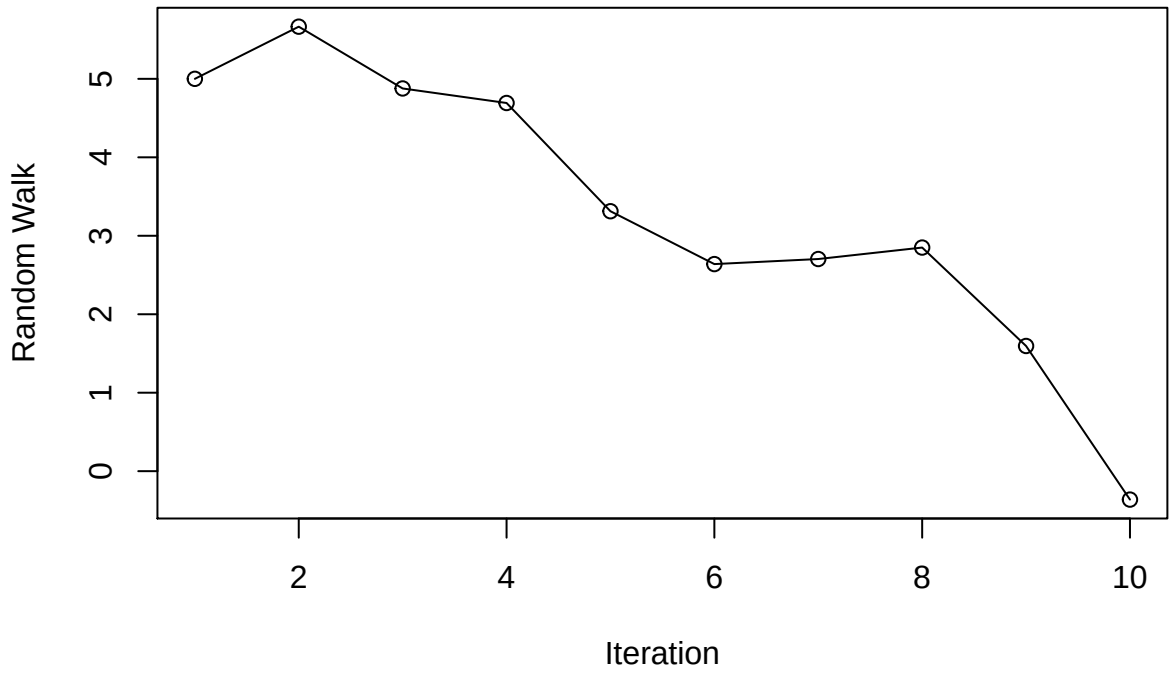
(16) [5 points] We'd like to answer the following question using simulation: if we start our random walk process, as defined above, at $x = 5$, what is the expected number of iterations we need until it terminates? To estimate the solution produce $10,000$ such random walks and calculate the average number of iterations in the $10,000$ random walks you produce. You'll want to turn the plot off here.

```
n     <- 10000
iters <- rep(NA, n)
for (i in 1:n) {
  iters[i] <- random.walk(plot.walk = FALSE)$num.steps
}
mean(iters)
```

```
## [1] 12.2637
```

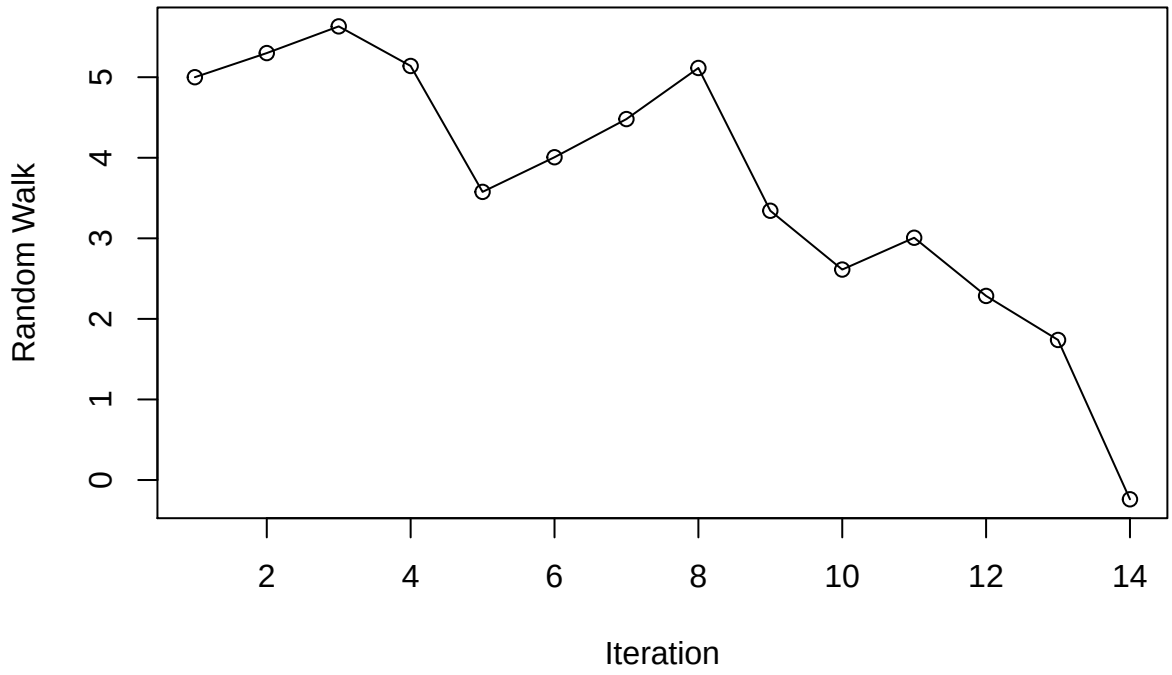
(17) [5 points] Modify your function `random.walk()` defined previously so that it takes an additional argument `seed`: this is an integer that should be used to set the seed of the random number generator, before the random walk begins, with `set.seed()`. But, if seed is `NULL`, the default, then no seed should be set. Run your modified function `random.walk()` function several times with the default inputs, then run it several times with the input seed equal to (say) 33.

```
random.walk <- function(x.start = 5, plot.walk = TRUE, seed = NULL) {
  if (!is.null(seed)) {set.seed(seed)}
  x.vals <- 5
  i <- 1
  while(all(x.vals[i] > 0)) {
    r <- runif(1, min = -2, max = 1)
    x.vals <- c(x.vals, x.vals[i] + r)
    i <- i+1
  }
  if (plot.walk) {plot(1:length(x.vals), x.vals, type = "o", xlab = "Iteration", ylab = "Random Walk")}
  return(list(x.vals = x.vals, num.steps = i))
}

random.walk()
```




```
## $x.vals
##  [1]  5.0000000  5.6647497  4.8775822  4.6920773  3.3125890  2.6390292
##  [7]  2.7038196  2.8499278  1.5953446 -0.3620453
##
## $num.steps
## [1] 10
```

```
random.walk()
```




```
## $x.vals
##  [1]  5.0000000  5.2995138  5.6306227  5.1392818  3.5765091  4.0065142
##  [7]  4.4803391  5.1143302  3.3420495  2.6121840  3.0073434  2.2848666
```

```
## [13]  1.7385892 -0.2391146
##
## $num.steps
## [1] 14
```

```
random.walk(seed = 33, plot.walk = FALSE)
```

```
## $x.vals
##  [1]  5.0000000  4.3378214  3.5217724  2.9729590  3.7295869  4.2612312
##  [7]  3.8132800  3.1246550  2.1542497  0.2008006 -1.4452259
##
## $num.steps
## [1] 11
```

```
random.walk(seed = 33, plot.walk = FALSE)
```

```
## $x.vals
##  [1]  5.0000000  4.3378214  3.5217724  2.9729590  3.7295869  4.2612312
##  [7]  3.8132800  3.1246550  2.1542497  0.2008006 -1.4452259
##
## $num.steps
## [1] 11
```

Part 4: Monte Carlo Integration

The goal of this exercise is to estimate the mathematical constant π using **Monte Carlo Integration**. Consider the function

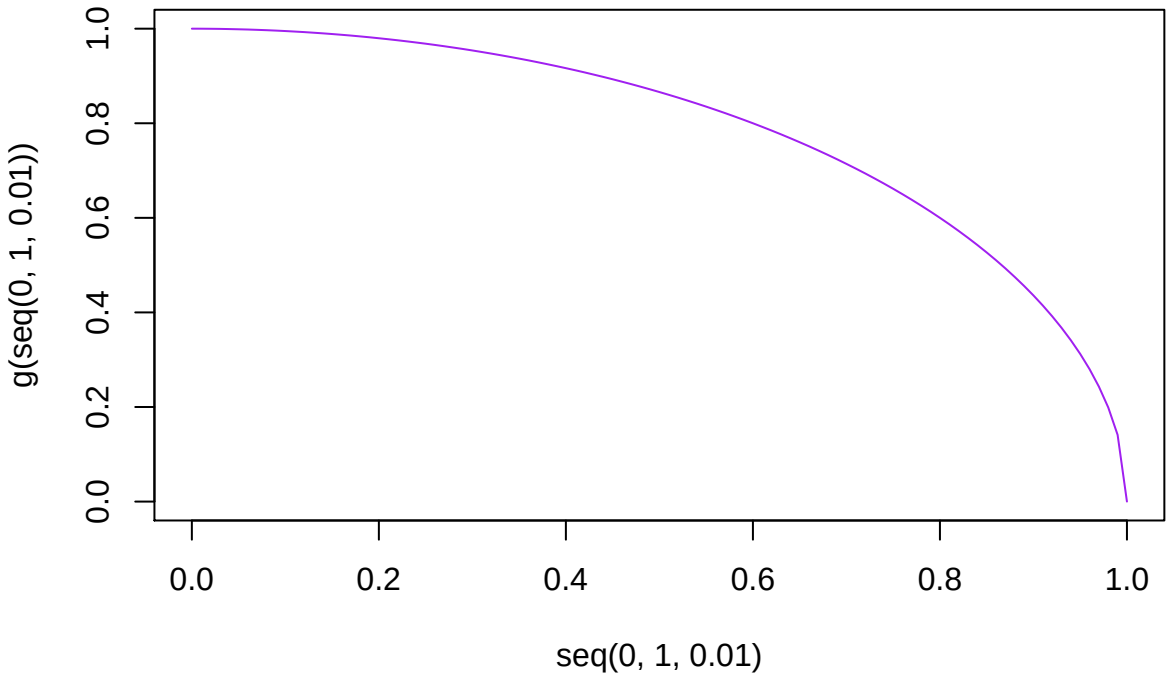
$$g(x) = \begin{cases} \sqrt{1-x^2} & 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The above function traces out a quartile circle over the interval $[0, 1]$.

Perform the following tasks:

18) [5 points] Run the following code:

```
g <- function(x) {
  return(sqrt(1-x^2))
}
plot(seq(0,1,.01),g(seq(0,1,.01)),type="l",col="purple")
```



The above code should produce the plot of a quarter circle.

19) [5 points] Identify the true area under the curve $g(x)$ by using simple geometric formulas.

$$\frac{\pi r^2}{4} = \frac{\pi * 1^2}{4} = \frac{\pi}{4}$$

20) [5 points] Using **Monte Carlo Integration**, approximate the mathematical constant π within a 1/1000 of the true value. When performing this simulation, make sure to choose a probability density function that has support over the unit interval, i.e. uniform(0,1) or beta(α,β).

```
# Choose n=1000^2
X <- runif(1000^2)
pi.est <- mean(4*sqrt(1-X^2))
pi.est
```

```
## [1] 3.142523
```

```
abs(pi-pi.est)
```

```
## [1] 0.0009306794
```

Please submit the knitted .pdf file!