HW6_cq2203

Chang Qu (cq2203)

11/22/2018

Contents

Part 1: Inverse Transform Method

1. Consider the Cauchy random variable X. Let U be a uniform random variable over [0,1]. Find a transformation of U that allows you to simulate X from U.

For a Cauchy random variable X with p.d.f:

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

It has a following CDF:

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

Therefore,

$$u = \frac{1}{\pi}arctan(x) + \frac{1}{2},\ x = tan(\pi(u - \frac{1}{2}))$$

```r
set.seed(0)

F.inverse <- function(u){
  return(ifelse((u<0|u>1),0, tan(pi*(u-0.5))))
}
```

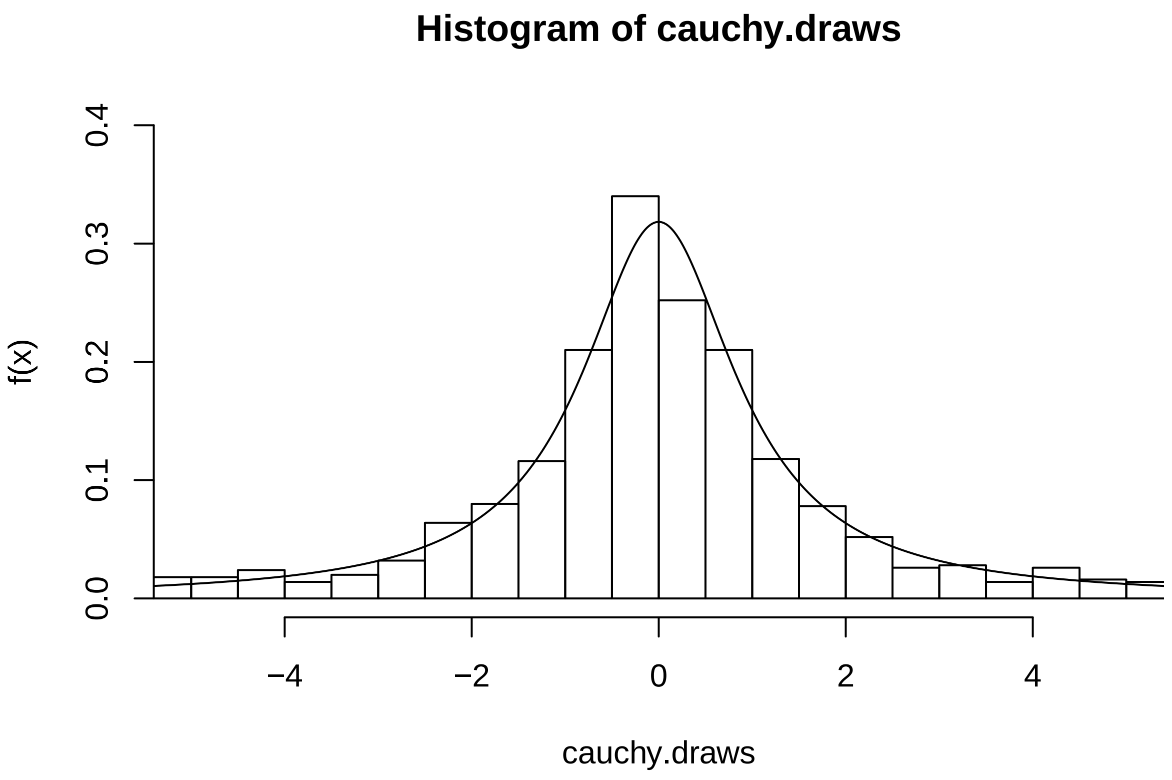
2. Write a R function called cauchy.sim that generates n simulated Cauchy random variables.

```r
set.seed(0)
cauchy.sim <- function(n){
  u <- runif(n)
  x <- F.inverse(u)
  return(x)
}

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. 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.

```
set.seed(0)
# simulate 1000 random draws
cauchy.draws <- cauchy.sim(1000)
# plot a histgram of the previous draws
hist(cauchy.draws, prob = TRUE, xlim = c(-5,5),ylim = c(0,0.4), breaks = 10000, ylab = "f(x)")
y <- seq(-10,10,0.01)
# density curve
lines(y, (1/pi)*(1/(1+y^2)))
```



Part 2: Reject-Accept Method

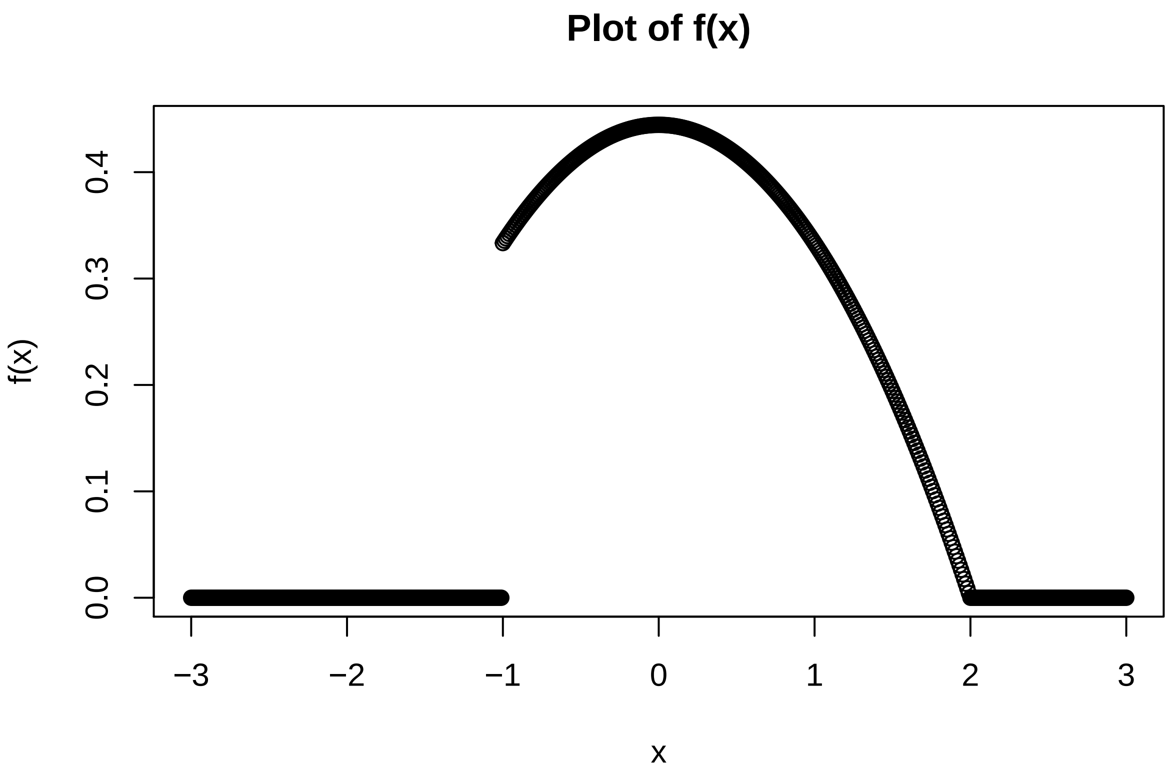
Problem 2

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

$$f(x) = 1/9(4 - x^2),\ -1 \leq x \geq 20,\ otherwise$$

4. 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.

```
f <- function(x){
  return(ifelse((x < -1|x > 2),0, (1/9)*(4-x^2)))
}

x <- seq(-3,3,0.01)
plot(x, f(x), main = "Plot of f(x)", xlab = "x", ylab = "f(x)")
```



5. Determine the maximum of $f(x)$ and find an envelope function $e(x)$ by using a uniform density for $g(x)$. Write a functione which takes as input a vector x and returns a vector of $e(x)$ values.

$$f^{`}(x) = -\frac{2}{9}x = 0 \ \rightarrow \ x = 0$$

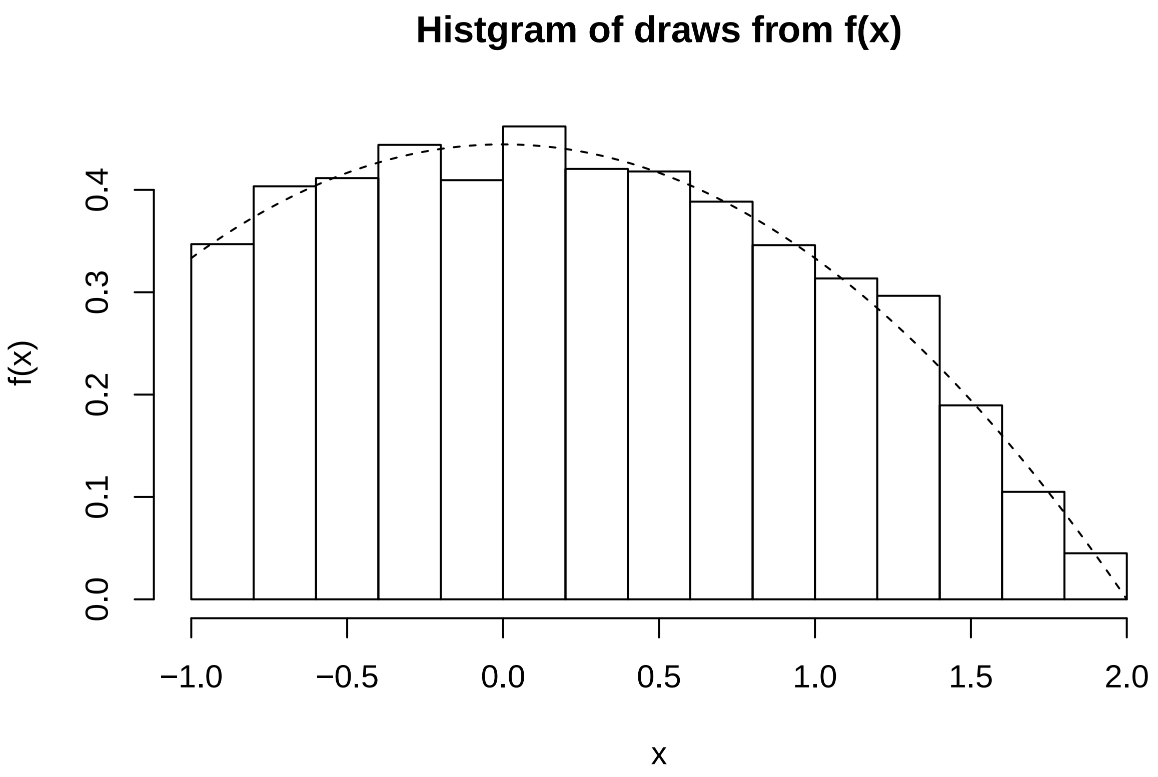
```
set.seed(0)
x.max <- 0
f.max <- f(x.max)

alpha <- f.max
e <- function(x){
  g <- dunif(x, min = -1, max = 2)
  return(ifelse((x < -1|x > 2),Inf, g/alpha))
}
```

6. Using the **Accept-Reject Algorithm**, write a program that simulates 10,000 draws from the probability density function $f(x)$ from Equation 1. Store your draws in the vector f.draws.

```
set.seed(0)
n.samps <- 10000 # number of samples wanted
n <- 0 # counter
f.draws <- numeric(n.samps) # initialize the vector of output
while(n<n.samps){
  y <- runif(1, min = -1, max = 2)
  u <- runif(1)
  if (u < f(y)/e(y)){
    n <- n+1
    f.draws[n] <- y
  }
}
```

7. Plot a histogram of your simulated data with the density function f overlaid in the graph. Label your plot appropriately.

```r
set.seed(0)
x <- seq(-1,2, 0.01)
hist(f.draws, prob=TRUE, ylab = "f(x)", xlab = "x", main = "Histgram of draws from f(x)")
lines(x, f(x),lty=2) #density curve
```

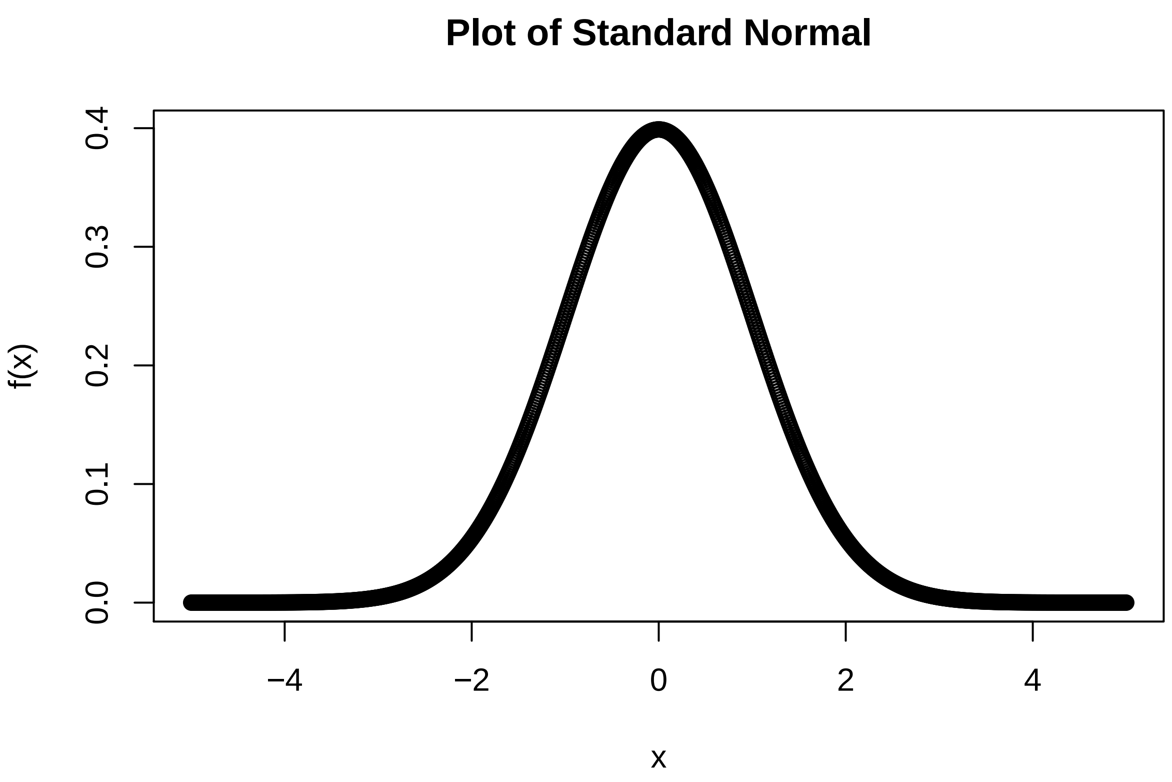


Problem 3: Reject-Accept Method Continued

Consider the standard normal distribution X with probability density function.

8. Write a functionfthat 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.

```r
f <- function(x){
  return((1/sqrt(2*pi))*exp((-1/2)*x^2))
}

x <- seq(-5,5,0.01)
plot(x, f(x), main = "Plot of Standard Normal", xlab = "x", ylab = "f(x)")
```

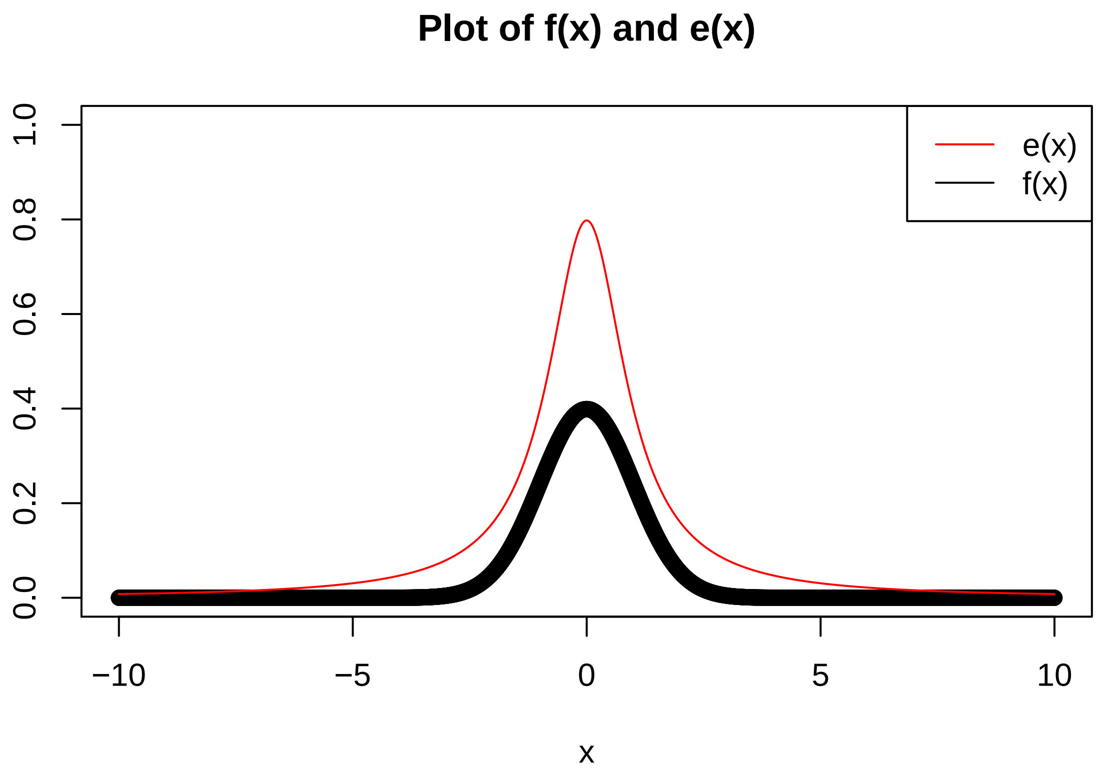


9. Let the known density g be the Cauchy density. Write a functione that takes as input a vector x and constant alpha($0 < \ < 1$) and returns a vector of $e(x)$ values. The envelope function should be defined as $e(x) = g(x)/$.

```r
set.seed(0)
# evenlope function using cauchy density
e <- function(x,alpha){
  return(((1/(pi))*(1/(1+x^2)))/alpha)
}
```

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

```r
set.seed(0)
x <- seq(-10,10,0.01)
f.max <- max(f(x))
alpha <-f.max
plot(x, f(x), main = "Plot of f(x) and e(x)", xlab = "x", ylab = "",ylim = c(0,1))
lines(x, e(x,alpha), col="2")
legend("topright", lty=1, legend=c("e(x)","f(x)"), col=c("2","1"))
```



From the plot, we can see that the desity curve of f(x) is fully covered under e(x). Therefore, f.max is a good value of alpha.

11. Write a function named normal.sim that simulates n standard normal random variables using the **Accept-Reject Algorithm**.

```r
set.seed(0)

normal.sim <- function(n){
  n.samps <- n # samples wanted
  count <- 0 #counter
  samps <- numeric(n.samps)
  while (count<n.samps){
    y <- cauchy.sim(1)
    u <- runif(1)
    if (u < f(y)/e(y,alpha)){
      count <- count+1
      samps[count] <- y
    }
  }
  return(samps)
}

normal.sim(10)
```

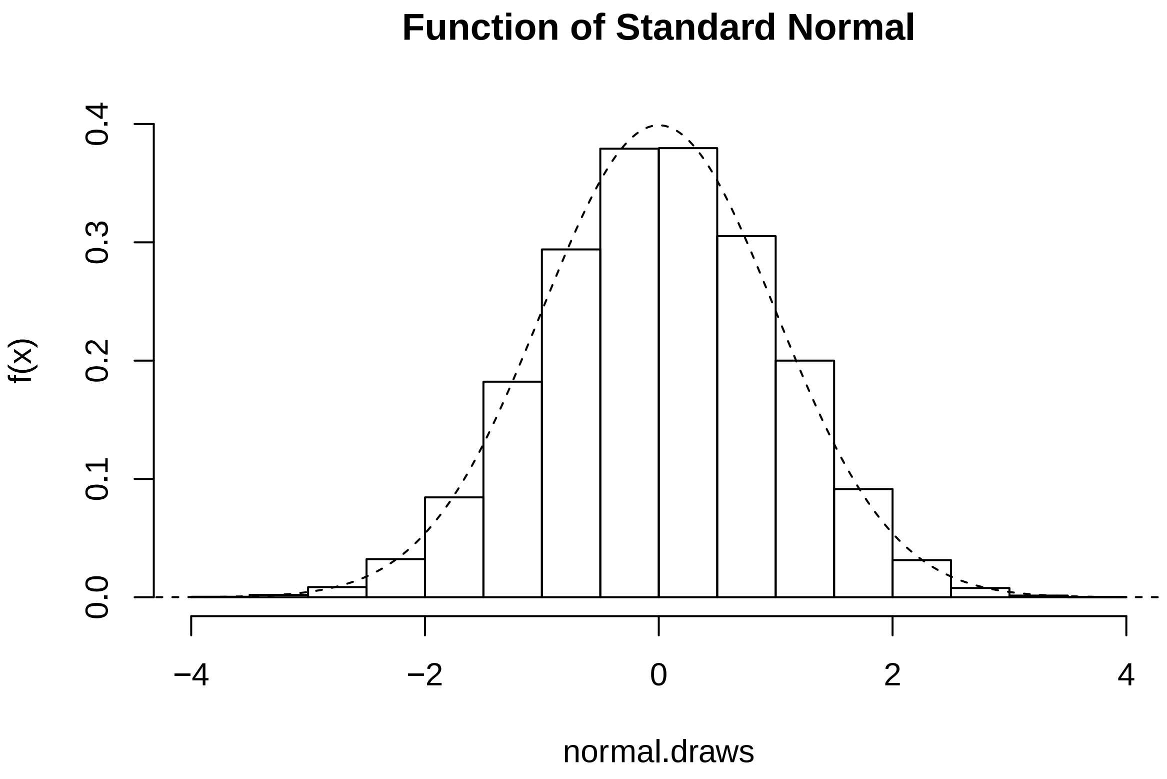
```
##  [1] -0.3738740 -0.5483277  0.3234124  1.2838224  0.6922180 -1.0332359
##  [7] -0.7577050 -0.9508564  1.1081177  1.8114418
```

12. 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.

```r
set.seed(0)
normal.draws <- normal.sim(10000)
length(normal.draws)
```

```
## [1] 10000
```

```r
# histgram of the simulated standard normal
hist(normal.draws, prob=TRUE, main = "Function of Standard Normal", ylab = "f(x)",
     ylim = c(0,0.4))
lines(x,f(x),lty=2) # density curve
```



Part 3: Simulation with Buit-in R functions

Consider the following "random walk" procedure:

- Start withx= 5
- Draw a random numberruniformly between−2 and 1
- Replacexwithx+r
- Stop ifx 0
- Else repeat

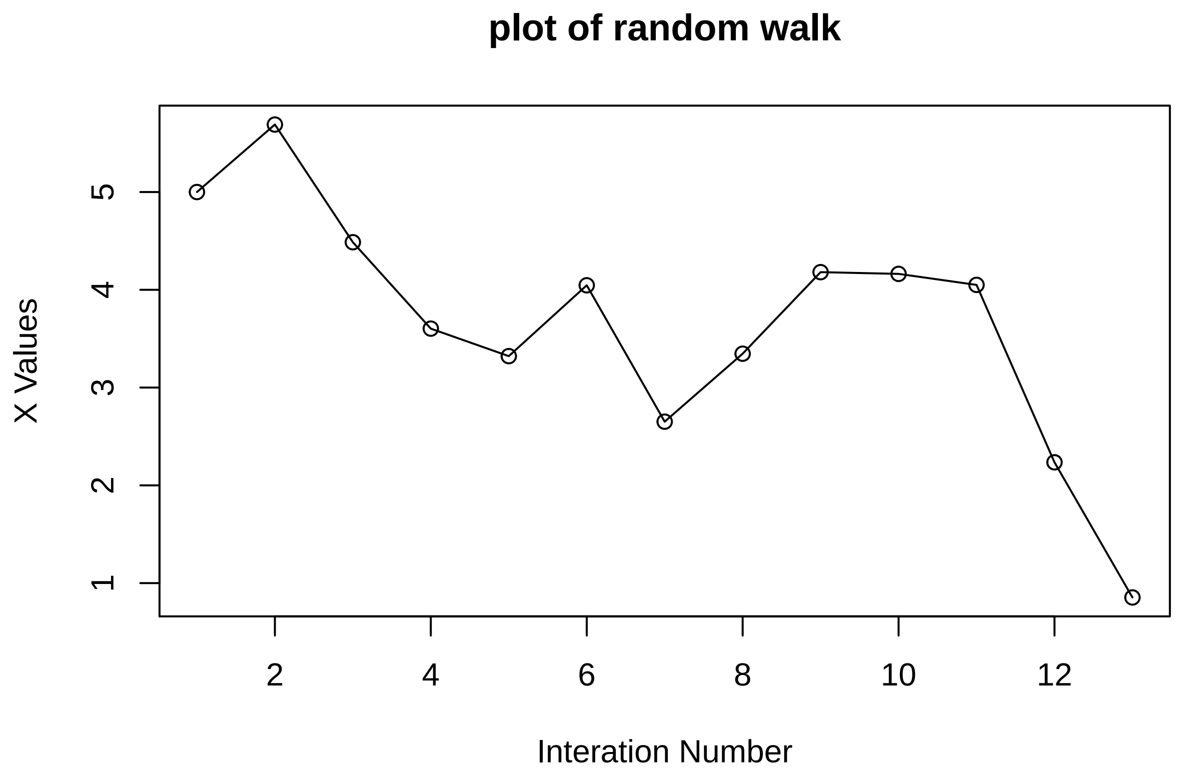
13. Write a while() loop to implement this procedure. Importantly, save all the positive values of x that were visited in this procedure in a vector called x.vals, and display its entries.

```r
set.seed(0)

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

x.vals <- x.vals[x.vals>0]
x.vals
```

```
##  [1] 5.0000000 5.6900916 4.4866176 3.6029893 3.3215494 4.0461727 2.6512185
##  [8] 3.3463876 4.1804134 4.1628068 4.0501489 2.2355077 0.8534314
```

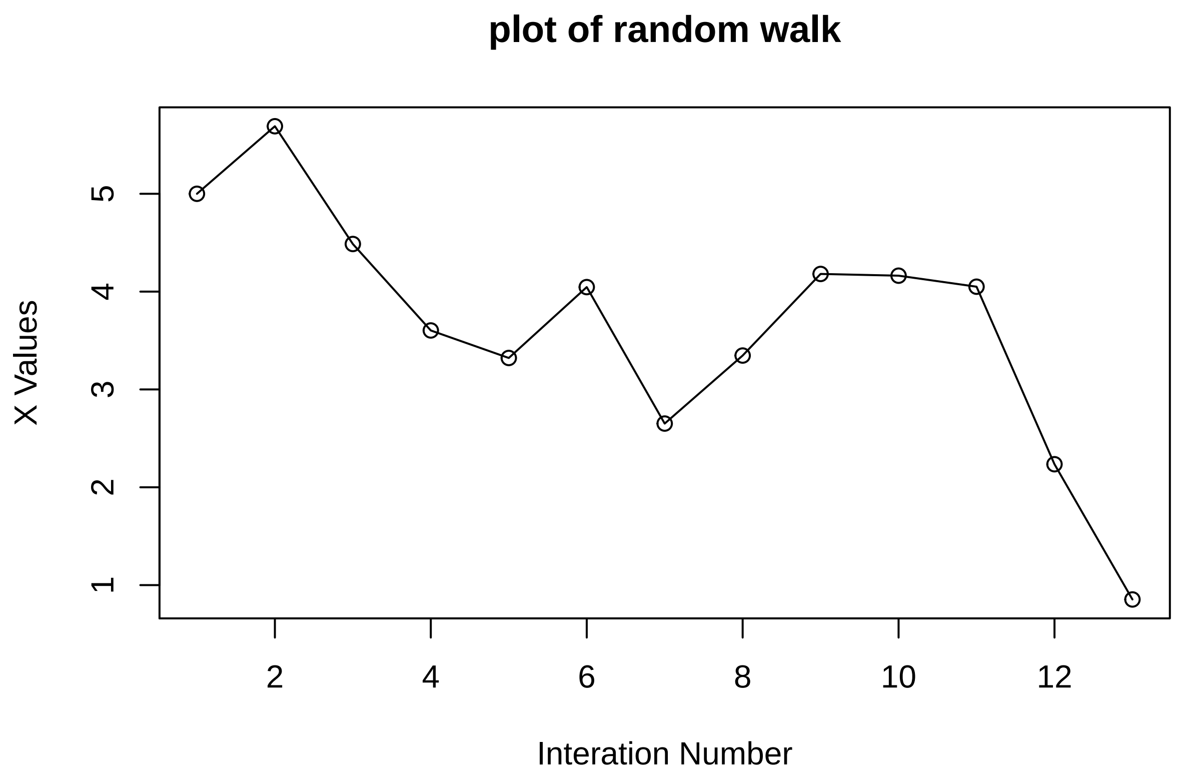
14. Produce a plot of the random walk values x.vals from above versus the iteration number.

```
set.seed(0)
count <- c(1:length(x.vals))
plot(count, x.vals, xlab = "Interation Number", ylab = "X Values",
     main = "plot of random walk", type = "o")
```



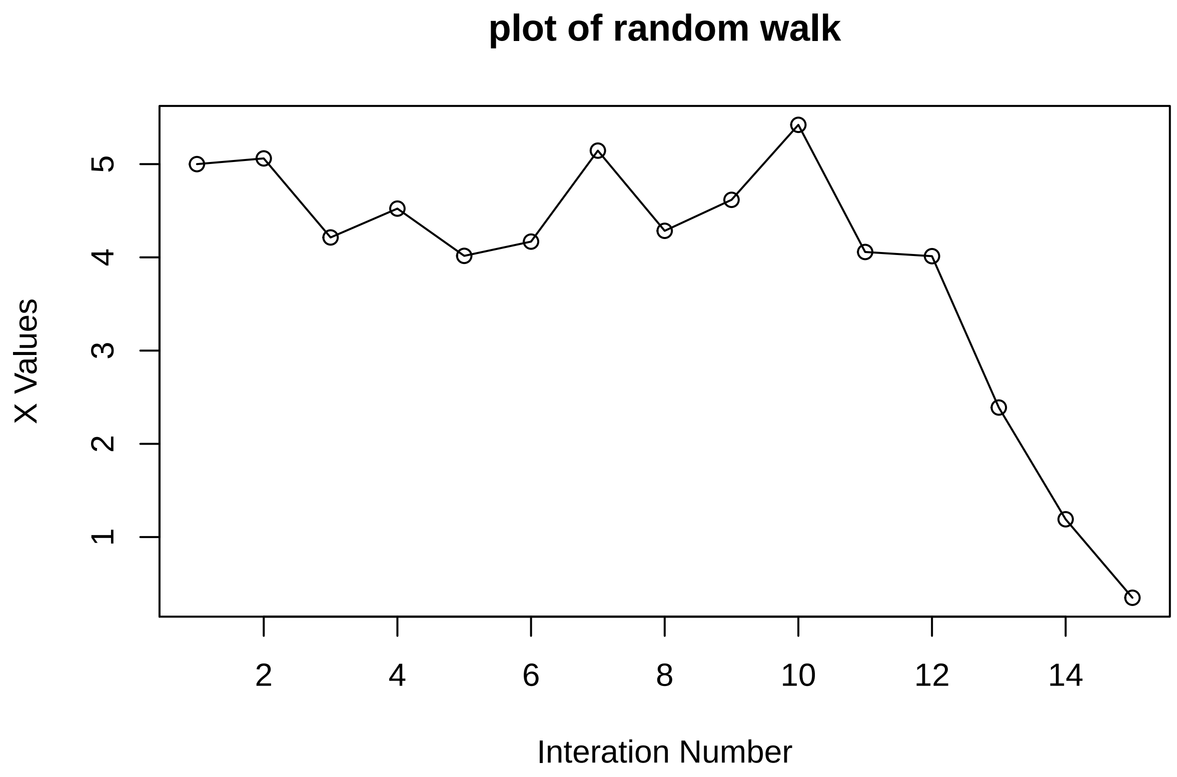
15. Write a function random.walk() to perform the random walk procedure that you implemented in question (9). Run your function twice with the default inputs, and then twice times with x.start equal to 10 and plot.walk =FALSE.

```
set.seed(0)

random.walk <- function(x.start=5, plot.walk=TRUE){
  count <- 1
  while (x.start[count]>0){
  r <- runif(1, min = -2, max = 1)
  x.start <- c(x.start, x.start[count]+r)
  count <- count+1
  }
  x.vals <- x.start[x.start>0]
  if (plot.walk==TRUE){
    count <- c(1:length(x.vals))
    plot(count, x.vals, xlab = "Interation Number", ylab = "X Values",
         main = "plot of random walk", type = "o")
  }
  return(list(x.vals=x.vals, num.steps=length(x.vals)))
}

random.walk()
```



```
## $x.vals
##  [1] 5.0000000 5.6900916 4.4866176 3.6029893 3.3215494 4.0461727 2.6512185
##  [8] 3.3463876 4.1804134 4.1628068 4.0501489 2.2355077 0.8534314
##
## $num.steps
## [1] 13
```

```
# run another time with default value
random.walk()
```



```
## $x.vals
##  [1] 5.0000000 5.0610685 4.2133797 4.5229040 4.0160017 4.1688572 5.1445755
##  [8] 4.2846810 4.6170167 5.4211324 4.0575599 4.0125812 2.3892465 1.1909085
## [15] 0.3492508
##
## $num.steps
## [1] 15
```

```
# run with x.start=10 and plot.walk=false
random.walk(x.start = 10, plot.walk = FALSE)
```

```
## $x.vals
##  [1] 10.000000  9.147164  9.756236  8.777283  8.223524  8.022221  7.502845
##  [8]  6.061498  6.543618  6.549018  6.931738  5.255569  5.426701  4.660525
## [15]  5.123364  5.064544  5.413342  5.072451  4.661610  5.029679  3.099672
## [22]  2.531363  2.728304  2.806499  2.239357  2.822986  2.137277  0.871669
##
## $num.steps
## [1] 28
```

```
# run another time with x.start=10 and plot.walk=false
random.walk(x.start = 10, plot.walk = FALSE)
```

```
## $x.vals
##  [1] 10.0000000  8.2983985  7.2472136  6.8031164  6.7891316  6.0096222
##  [7]  6.7482500  5.6290601  5.0062573  4.0034413  3.9560527  2.7301030
## [13]  2.1657388  2.4646708  0.7174115  1.3433755  0.3605943  0.8789154
##
## $num.steps
## [1] 18
```

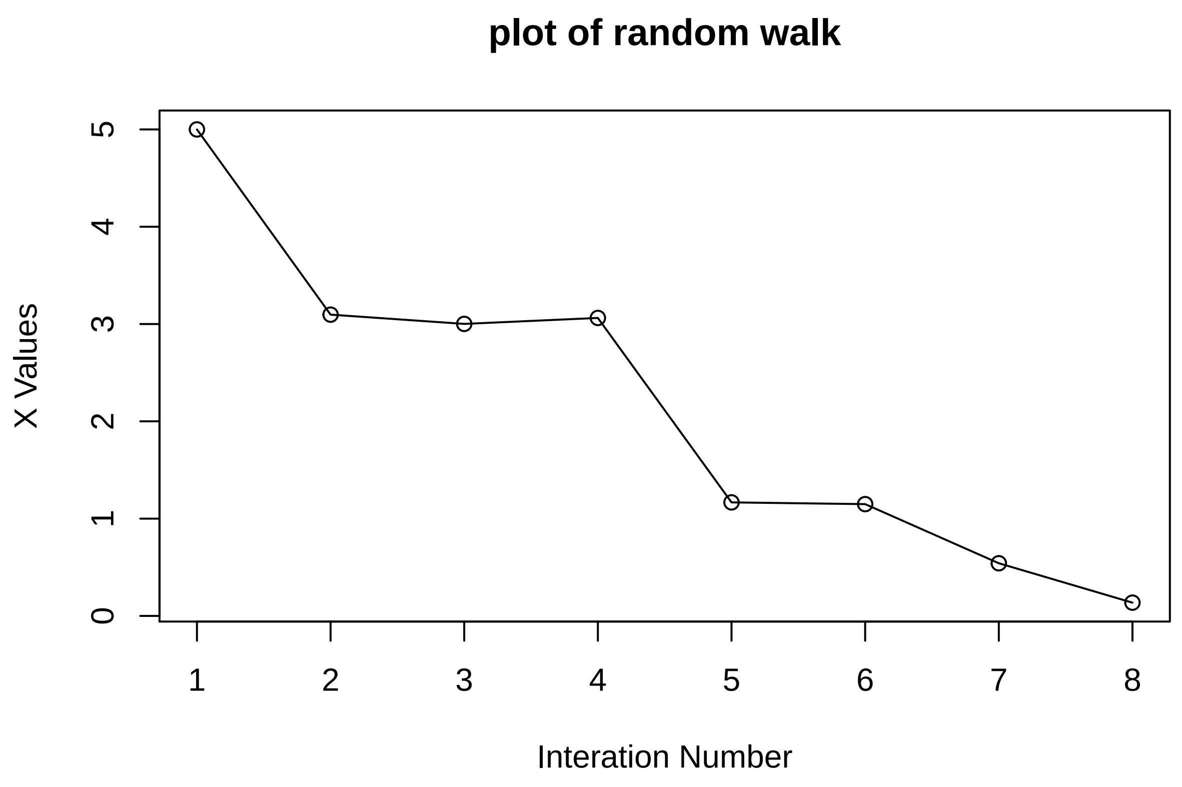
16. To estimate the solution produce 10,000 such random walks and calculate the average number of iterations in the 10,000 random walks you produce. You'll want to turn the plot off here.

```r
set.seed(0)
interation <- rep(NA, 10000)
for(i in 1:10000){
  interation[i] <- random.walk(x.start = 5, plot.walk = FALSE)$num.step
}

mean(interation)
```

```
## [1] 11.2495
```

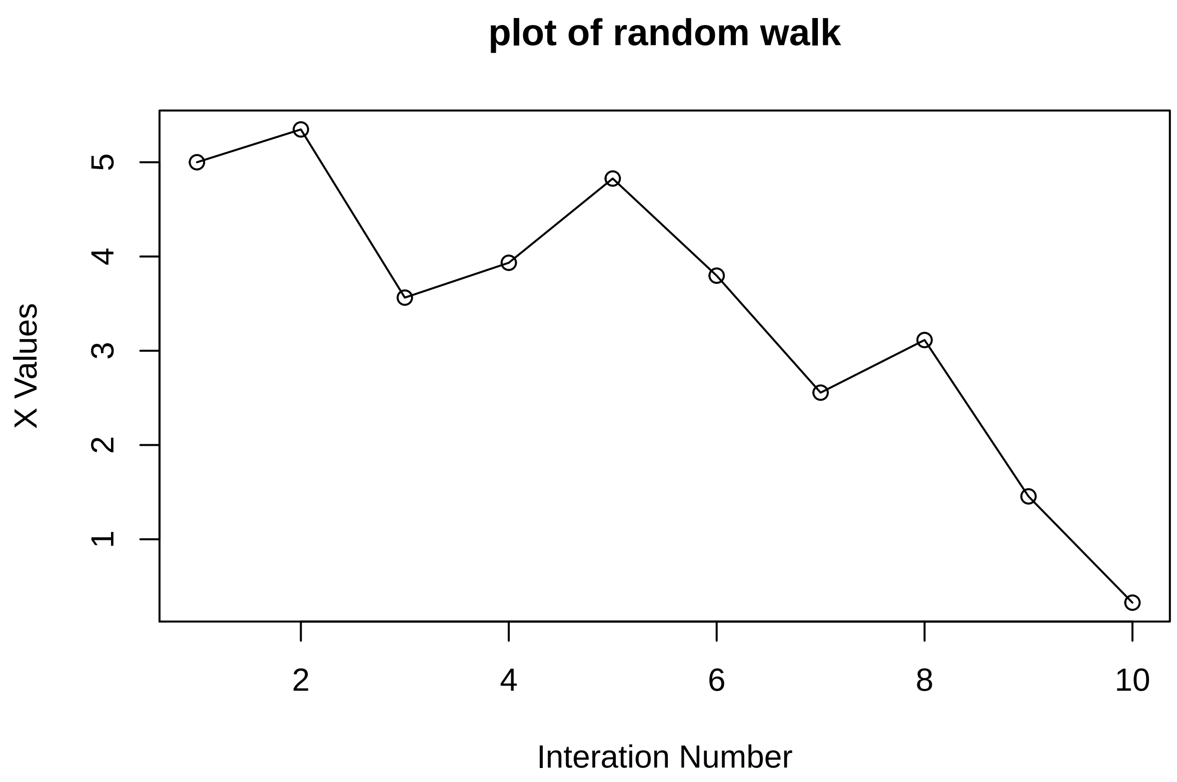
17. Modify your function random.walk() defined previously so that it takes an additional argument seed: this is an integer that should be used to set the seed of the random number generator, before the random walk begins, with set.seed(). But, if seed is NULL, the default, then no seed should be set.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

```r
random.walk <- function(x.start=5, plot.walk=TRUE, seed=NULL){
  if(is.null(seed)){}
  else{
    set.seed(seed)
  }
  count <- 1
  while (x.start[count]>0){
  r <- runif(1, min = -2, max = 1)
  x.start <- c(x.start, x.start[count]+r)
  count <- count+1
  }
  x.vals <- x.start[x.start>0]
  if (plot.walk==TRUE){
    count <- c(1:length(x.vals))
    plot(count, x.vals, xlab = "Interation Number", ylab = "X Values",
         main = "plot of random walk", type = "o")
  }
  return(list(x.vals=x.vals, num.steps=length(x.vals)))
}

random.walk()
```



```
## $x.vals
## [1] 5.0000000 3.0966701 3.0014069 3.0625444 1.1668237 1.1492436 0.5412714
## [8] 0.1367114
##
## $num.steps
## [1] 8
```

```
# run another time with defalut value
random.walk()
```



```
## $x.vals
##  [1] 5.000000 5.348049 3.563647 3.933380 4.827162 3.796786 2.555755
##  [8] 3.113828 1.455043 0.327715
##
## $num.steps
## [1] 10
```

```
# run with seed=33 and plot.walk=false
random.walk(plot.walk = FALSE, seed = 33)
```

```
## $x.vals
##  [1] 5.0000000 4.3378214 3.5217724 2.9729590 3.7295869 4.2612312 3.8132800
##  [8] 3.1246550 2.1542497 0.2008006
##
## $num.steps
## [1] 10
```

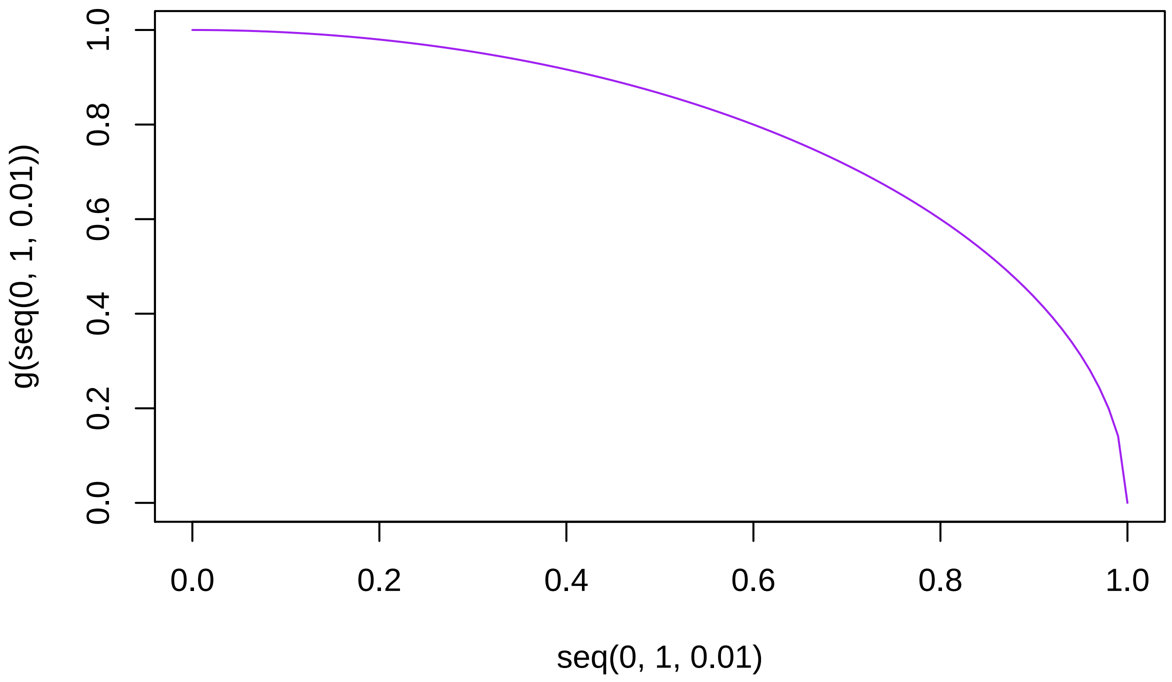
```
# run another time with seed=33 and plot.walk=false
random.walk(plot.walk = FALSE,seed = 33)
```

```
## $x.vals
##  [1] 5.0000000 4.3378214 3.5217724 2.9729590 3.7295869 4.2612312 3.8132800
##  [8] 3.1246550 2.1542497 0.2008006
##
## $num.steps
## [1] 10
```

Part 4: Monte Carlo Integration

18. Produce the plot of a quarter circle.

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



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

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

20. Using Monte Carlo Integration, approximate the mathematical constant   within a 1/1000 of the true value.

```r
set.seed(0)
h <- function(x) {4*sqrt(1-x^2)}
mean(h(runif(100000000)))
```

```
## [1] 3.141675
```