Giang Vu
STAT 2301
April 5, 2021

Assignment 5

Part 1:

1) $U \sim \text{uniform } [0,1]$

$f_X(x) = \frac{1}{\pi} \frac{1}{(1+x^2)}, \quad -\infty < x < \infty$, this is Cauchy with $x_0 = 0$, $\gamma = 1$

$\Rightarrow$ CDF is $F(x) = \int_{-\infty}^{x} f_X(t)\,dt$

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

$$= \frac{1}{\pi} \arctan(t) \Big|_{-\infty}^{x}$$

$$= \frac{1}{\pi} \arctan(x) + \frac{1}{\pi} \cdot \frac{\pi}{2}$$

$$= \frac{1}{\pi} \arctan(x) + \frac{1}{2}$$

$\Rightarrow$ Invert the CDF

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

$$\Rightarrow (u - \frac{1}{2})\pi = \arctan(x)$$

$$\Rightarrow \tan(\pi(u - \frac{1}{2})) = x$$

So $x = \tan(\pi(u - \frac{1}{2}))$ is how we simulate X from U using Inverse Transform Method

Assignment 5

Giang Vu

3/31/2021

Part 1.

2.

Using the transformation of U in previous part, I defined a function in R to generate n simulated Cauchy random variables below. I also tested the function with $n = 10$

```r
#define function cauchy.sim
cauchy.sim <- function(n) {
  #generate u from uniform[0,1]
  u <- runif(n)
  # Function for the inverse transform
  return(ifelse((u<0|u>1), 0, tan(pi*(u-0.5))))
}

#test with n = 10
set.seed(0)
cauchy.sim(10)
```

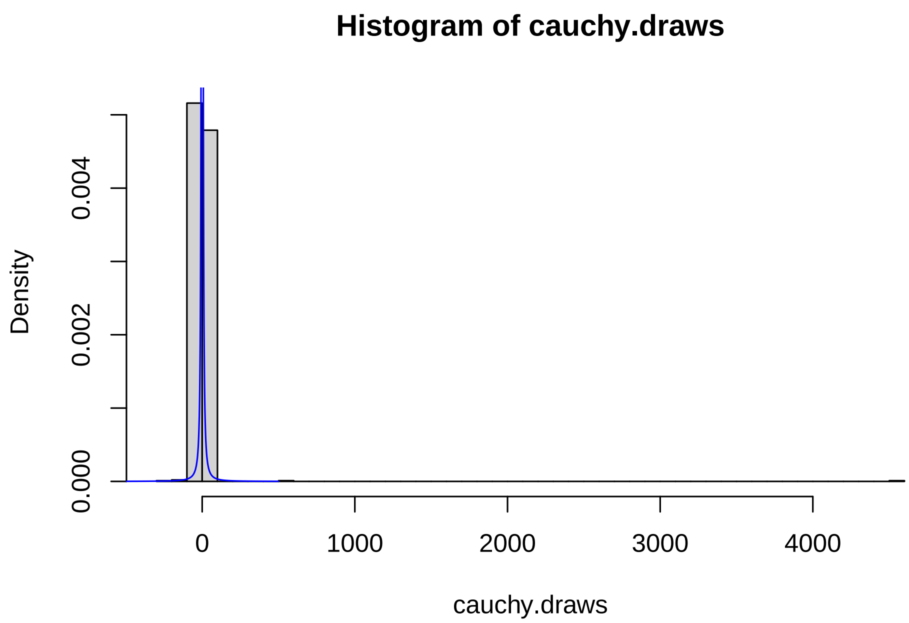
```
##  [1]  2.9723830 -0.9070131 -0.4248394  0.2329576  3.3710605 -1.3611982
##  [7]  3.0255173  5.6954298  0.5530230  0.4294381
```

3.

Using the cauchy.sim function in previous part, I simulated with $n = 1000$. Below is the histogram of this random variable with $f_X(x)$ overlaid on it.

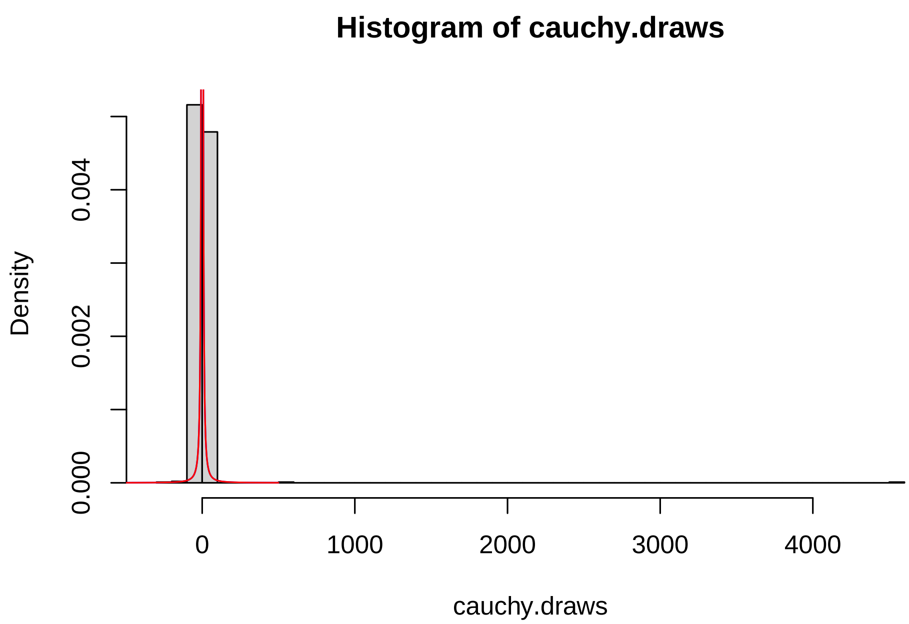
```r
#apply function with n =1000
set.seed(0)
cauchy.draws <- cauchy.sim(1000)

#histogram
hist(cauchy.draws, prob = T,breaks = 50)
y <- seq(-500, 500, 1)
lines(y, 1/(pi*(1+y^2)), col = "blue")
```



Here is the histogram of this random variable with the density of the true Cauchy(0,1) overlaid on it. I could see my simulation is very close to the true density generated using built in R function.

```
#histogram
hist(cauchy.draws, prob = T,breaks = 50)
y <- seq(-500, 500, 1)
lines(y, 1/(pi*(1+y^2)), col = "blue")
lines(y, dcauchy(y,location = 0,scale = 1),col ="red")
```



Part 2.

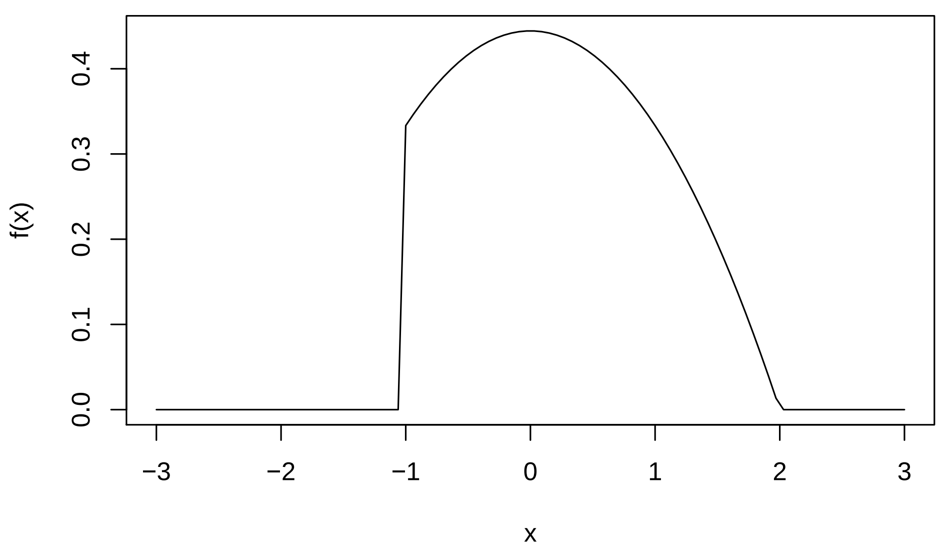
Problem 2.

4.

Function $f(x)$ is defined and a plot for it for values between -3 and 3 is generated.

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

#plot
x <- seq(-3,3,length=100)
plot(x=x,y=f(x),type = "l",
     ylab = "f(x)")
```



5.

From the plot, we can see the maximum of $f(x)$ is 0.444, which is achieved when $x = 0$. (Or we can take the derivative of $f(x)$ and set it equal to 0 to solve for x, which shows us the maximizer is x = 0).
We then form the envelope $e(x)$ with $\alpha = 1/f.max$ and $g(x)$ as the density for the uniform distribution on [-1,2] as follows.
$e(x) = g(x)/\alpha = f.max \geq f(x)$

```
#check max of f(x)
max(f(x))
```

```
## [1] 0.4443424
```

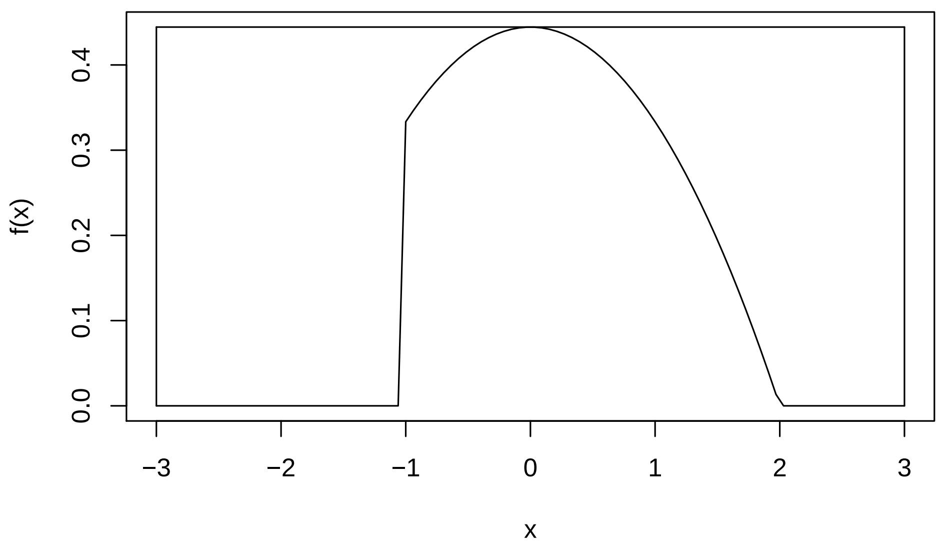
```r
f.max <- f(0)
f.max
```

```
## [1] 0.4444444
```

```r
#define envelope e(x)
e <- function(x) {
  return(ifelse((x < -1 | x > 2), Inf, f.max))
}
```

Here is the plot of the envelope function.

```r
plot(x=x,y=f(x),type = "l",
     ylab = "f(x)")
lines(c(-3, -3), c(0, e(0)), lty = 1)
lines(c(3, 3), c(0, e(1)), lty = 1)
lines(c(3,-3), c(e(0),e(0)), lty = 1)
```



6.
A program using the Accept-Reject Algorithm is written and the simulated data is saved in vector f.draws

```r
n.samps <- 10000   # number of samples desired
n       <- 0             # counter for number samples accepted
f.draws   <- numeric(n.samps) # initialize the vector of output
set.seed(0)
while (n < n.samps) {
```

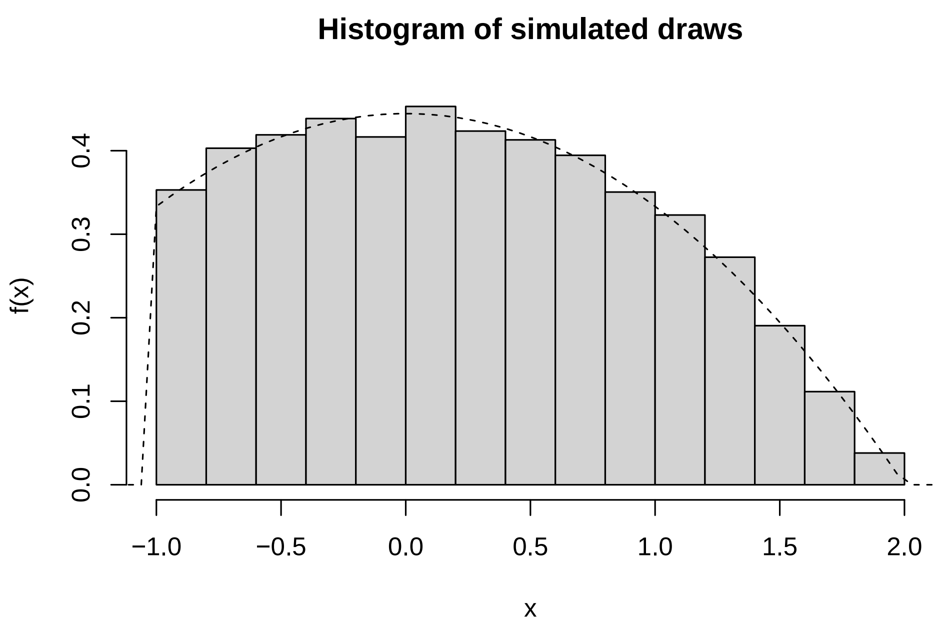
```
  y <- runif(1,min = -1,max = 2)     #random draw from g
  u <- runif(1)
  if (u < f(y)/e(y)) {
    n         <- n + 1
    f.draws[n] <- y
  }
}
```

7.

Histogram for simulated data with density f overlaid. The simulated data looks quite close to the density.

```
#histogram
x <- seq(-3,3,length=100)
hist(f.draws, prob = T, ylab = "f(x)", xlab = "x",
     main = "Histogram of simulated draws")

#add density line
lines(x, f(x), lty = 2)
```



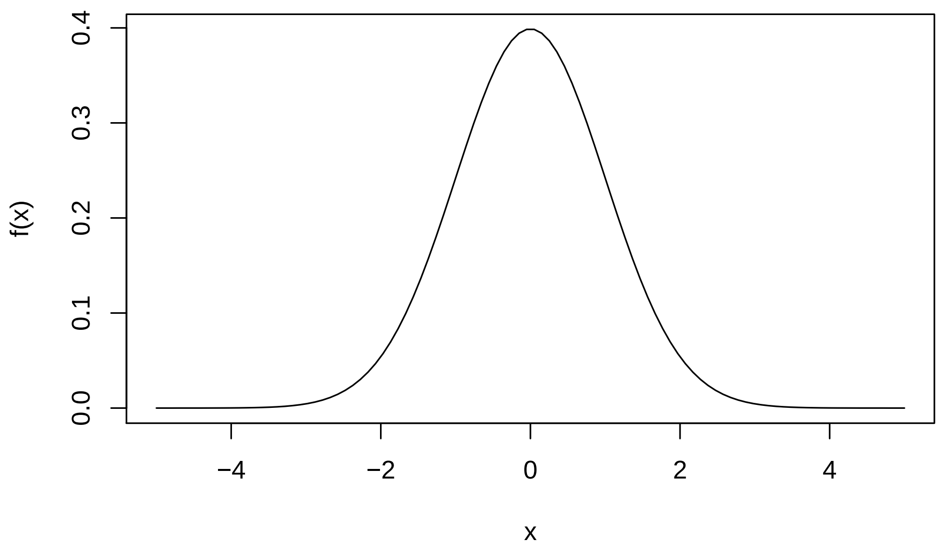
Problem 3.

8.

Function $f(x)$ is defined and a plot for it for values between -5 and 5 is generated.
I named it f1 to avoid mistaking with function f from Problem 2.

```r
#define f
f1 <- function(x){
  return((1/sqrt(2*pi))*exp((-1/2)*x^2))
}

#plot
x <- seq(-5,5,length=100)
plot(x=x,y=f1(x),type = "l",
     ylab = "f(x)")
```



9.

Function $e(x)$ is defined and I named it e1 to avoid mistaking it with e from Problem 2.

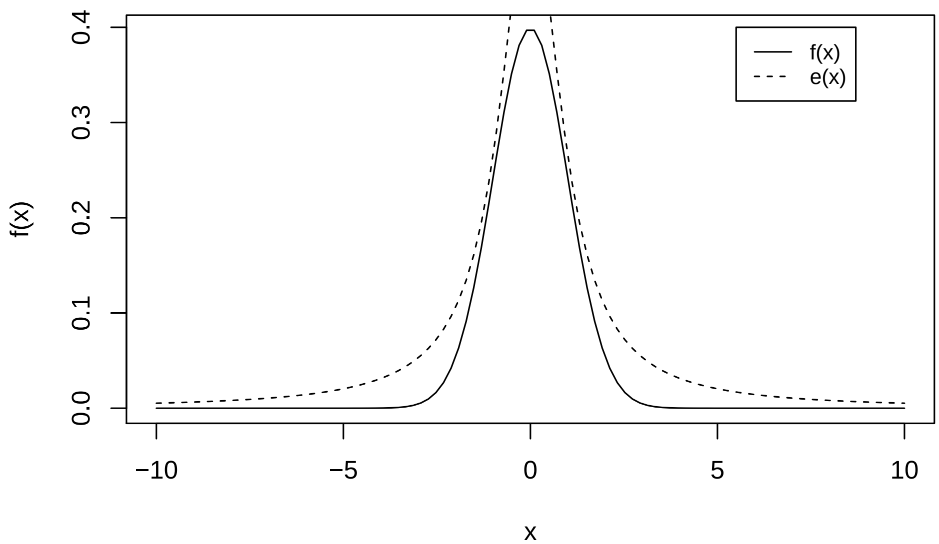
```r
#define e
e1 <- function(x, alpha){
  stopifnot(length(alpha)==1, (0 < alpha & alpha < 1))
  return((1/(pi*(1+x^2)))/alpha)
}
```

10.

After playing around and plotting for different values for α, I chose my good value to be 0.6. As seen from the plot on [-10,10] below, this envelope is very close to $f(x)$ is always right above $f(x)$ at every value of x.

```r
x <- seq(-10,10,length=100)
plot(x=x,y=f1(x),type = "l",
     ylab = "f(x)")
lines(x,e1(x,0.6), lty = 2)
```

```r
legend(5.5,0.4,legend=c("f(x)","e(x)"),lty=1:2,cex=0.8)
```



11.

A function using the Accept-Reject Algorithm is written and it also takes advantage of the Cauchy simulation function we defined in Part 1.

```r
#define function
normal.sim <- function(n){
  i <- 0 # counter for number samples accepted
  norm.draws <- numeric(n) # initialize the vector of output
  set.seed(0)
  while(i < n) {
    y <- cauchy.sim(1) #draw from g(x, which is cauchy using function in part 1
    u <- runif(1)
    if (u < f1(y)/e1(y,0.6)) {
      i <- i + 1
      norm.draws[i] <- y
    }
  }
  return(norm.draws)
}

#test with n = 10
normal.sim(10)
```

```
##  [1] -0.42483941  0.55302299 -1.61407911 -0.38108975 -0.00722817 -1.27142826
##  [7] -2.40235548 -0.37387400 -0.54832774  0.32341236
```

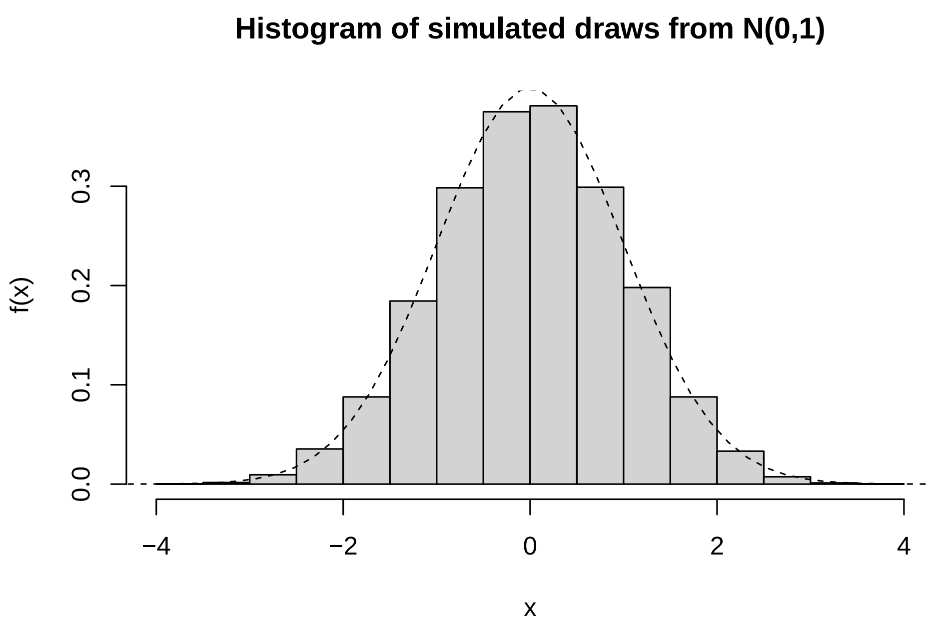
12.

Applying the function above for $n = 10000$ I obtained 10,000 draws from standard normal distribution, and made a histogram with $f(x)$ overlaid on the graph. The simulated draw is quite close to the actual density.

```
#draw 10000
normal.draws <- normal.sim(10000)

#histogram
x <- seq(-10,10,length=100)
hist(normal.draws, prob = T, ylab = "f(x)", xlab = "x",
     main = "Histogram of simulated draws from N(0,1)")

#add density line
lines(x, f1(x), lty = 2)
```



Part 3.

13.

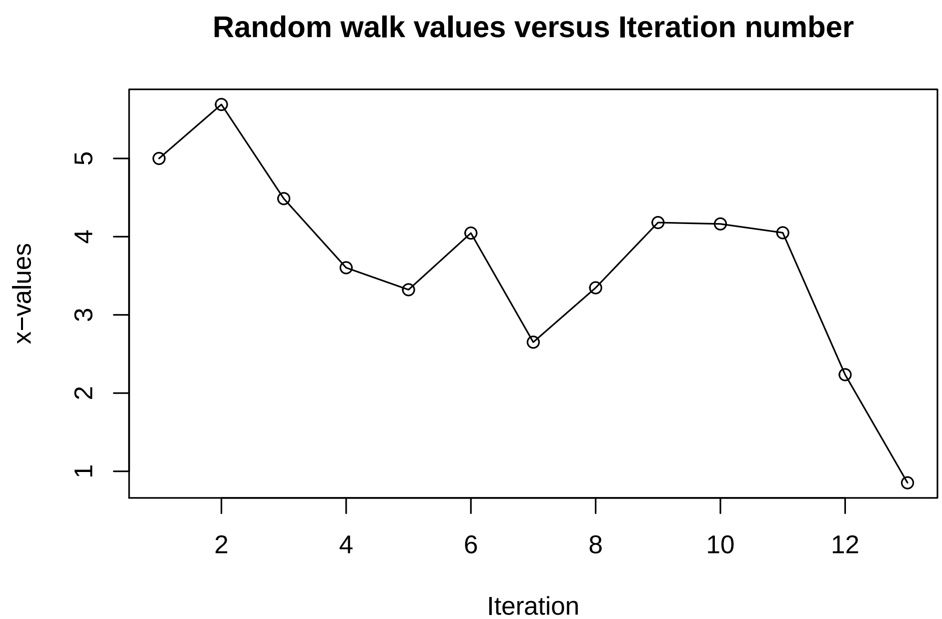
A while() loop is implemented below, and the result is a vector of 13 numbers that satisfy our requirements.

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

```
  x.vals[i] <- x
  x <- x + r
}
```

```
plot(y=x.vals,x=c(1:13),xlab = "Iteration", ylab = "x-values",
     type = "o", main = "Random walk values versus Iteration number")
```

14.



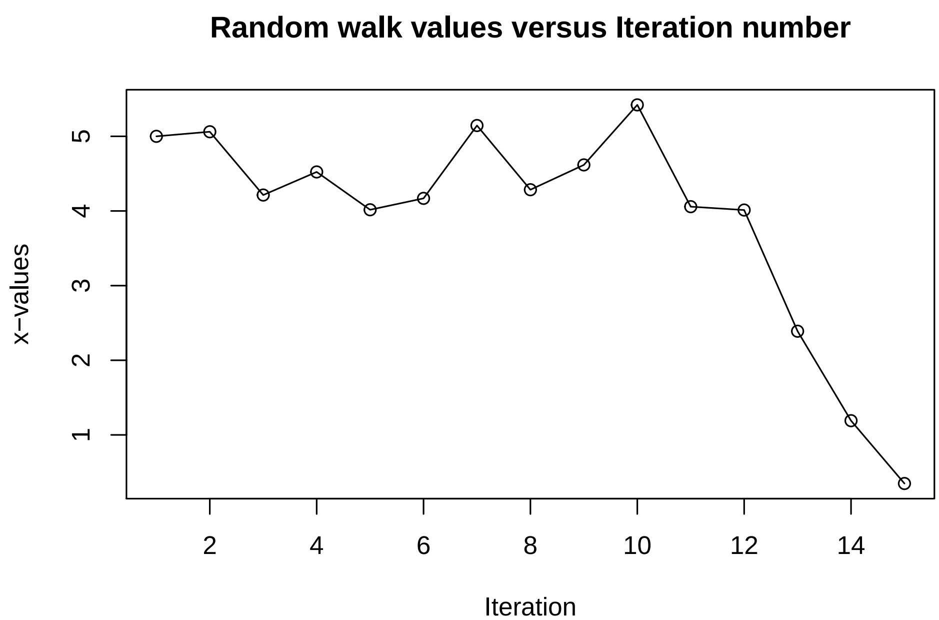
15.
The function is defined below.

```
#define function
random.walk <- function(x.start=5,plot.walk=TRUE){
  num.steps <- 0
  x.vals <- c()
  while(x.start > 0) {
    r <- runif(1, min=-2, max = 1)
    num.steps <- num.steps + 1
    x.vals[num.steps] <- x.start
    x.start <- x.start + r
  }
  if (plot.walk==TRUE){
    plot(y=x.vals,x=c(1:num.steps),xlab = "Iteration", ylab = "x-values",
         type = "o", main = "Random walk values versus Iteration number")
  }
```

```
  return(list(x.vals=x.vals,num.steps=num.steps))
}
```

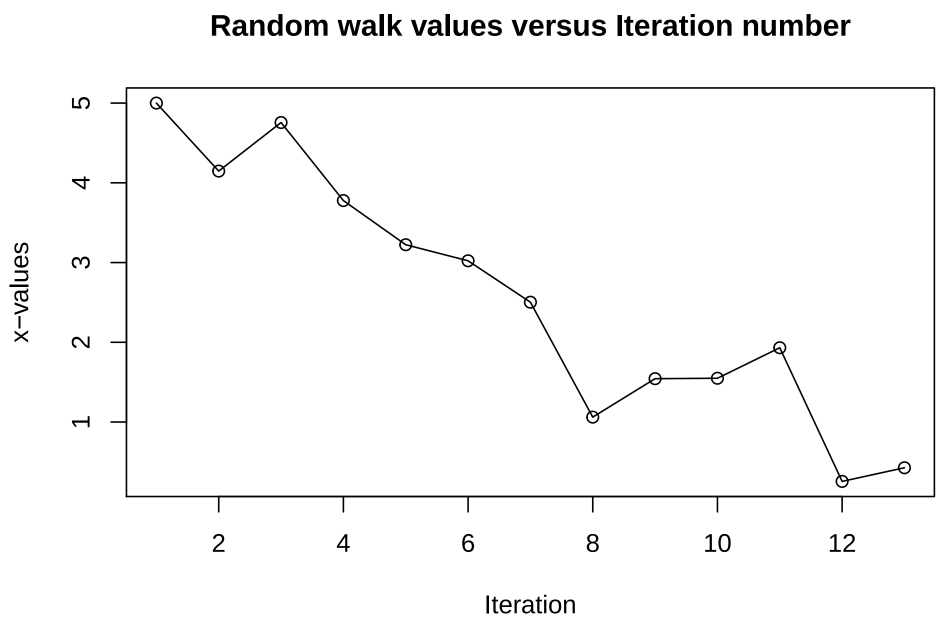
Test run twice with default values.

```
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
```

```
random.walk()
```




```
## $x.vals
##  [1] 5.0000000 4.1471639 4.7562364 3.7772834 3.2235237 3.0222212 2.5028451
##  [8] 1.0614979 1.5436179 1.5490181 1.9317377 0.2555686 0.4267014
##
## $num.steps
## [1] 13
```

Test run twice with 10 and FALSE as input.

```r
random.walk(10,F)
```

```
## $x.vals
##  [1] 10.00000000 10.46283888 10.40401946 10.75281775 10.41192669 10.00108543
##  [7] 10.36915412  8.43914773  7.87083792  8.06777914  8.14597381  7.57883268
## [13]  8.16246111  7.47675243  6.21114426  4.42318140  2.72157988  1.67039500
## [19]  1.22629779  1.21231302  0.43280358  1.17143136  0.05224147
##
## $num.steps
## [1] 23
```

```r
random.walk(10,F)
```

```
## $x.vals
##  [1] 10.000000  8.997184  8.949795  7.723846  7.159482  7.458414  5.711154
##  [8]  6.337118  5.354337  5.872658  4.912709  3.914033  3.343087  4.019682
## [15]  4.612701  3.782669  4.114631  4.996485  4.300464  4.438008  3.637991
## [22]  2.614047  2.885309  1.493386  1.626749
##
## $num.steps
## [1] 25
```

16.

By making 10,000 random walks with x = 5, I estimated the mean number of iterations to be 11.25. Essentially on average the random walk we designed carries out about 11 iterations before it terminates.

```r
#loop for 10000 random walks
iters <- numeric(10000)
for (i in 1:10000) {
  li <- random.walk(5,F)
  iters[i] <- li$num.steps #extract iteration number with each walk
}

#mean of iteration number
mean(iters)
```

```
## [1] 11.2497
```

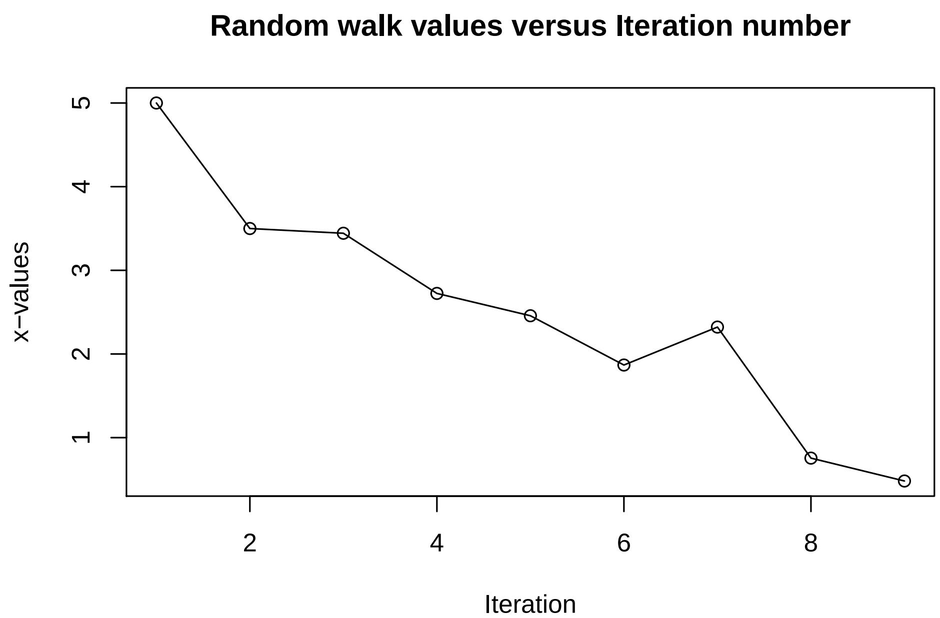
17.

The function is modified to add seed setting argument.

```r
#add seed to function
random.walk <- function(x.start=5,plot.walk=TRUE,seed=NULL){
  num.steps <- 0
  x.vals <- c()
  if (!is.null(seed)){set.seed(seed)} #set seed only when seed is specified in argument by user
  while(x.start > 0) {
    r <- runif(1, min=-2, max = 1)
    num.steps <- num.steps + 1
    x.vals[num.steps] <- x.start
    x.start <- x.start + r
  }
  if (plot.walk==TRUE){
    plot(y=x.vals,x=c(1:num.steps),xlab = "Iteration", ylab = "x-values",
         type = "o", main = "Random walk values versus Iteration number")
  }
  return(list(x.vals=x.vals,num.steps=num.steps))
}
```

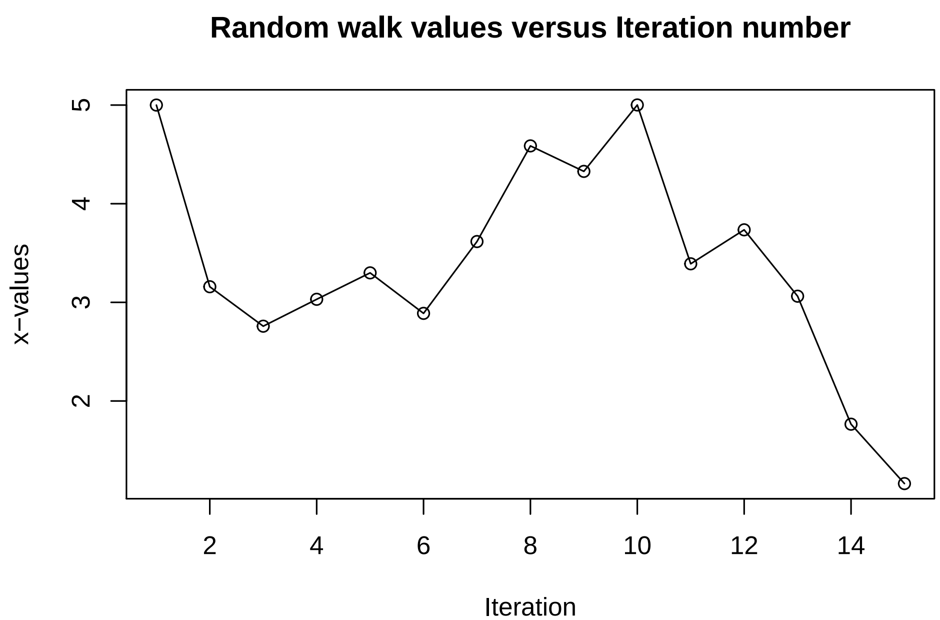
Test run with default arguments

```r
#test with default
random.walk()
```




```
## $x.vals
## [1] 5.0000000 3.4995781 3.4430442 2.7240724 2.4570026 1.8679304 2.3217466
## [8] 0.7552699 0.4817039
##
## $num.steps
## [1] 9
```

```r
#test with default again
random.walk()
```



```
## $x.vals
##  [1] 5.000000 3.158331 2.758196 3.031020 3.298819 2.888416 3.616463 4.586110
##  [9] 4.327696 5.000921 3.390694 3.735350 3.062375 1.764890 1.162725
##
## $num.steps
## [1] 15
```

Test run with seed and no plot

```
#test with seed 33
random.walk(seed = 33,plot.walk = F)
```

```
## $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
```

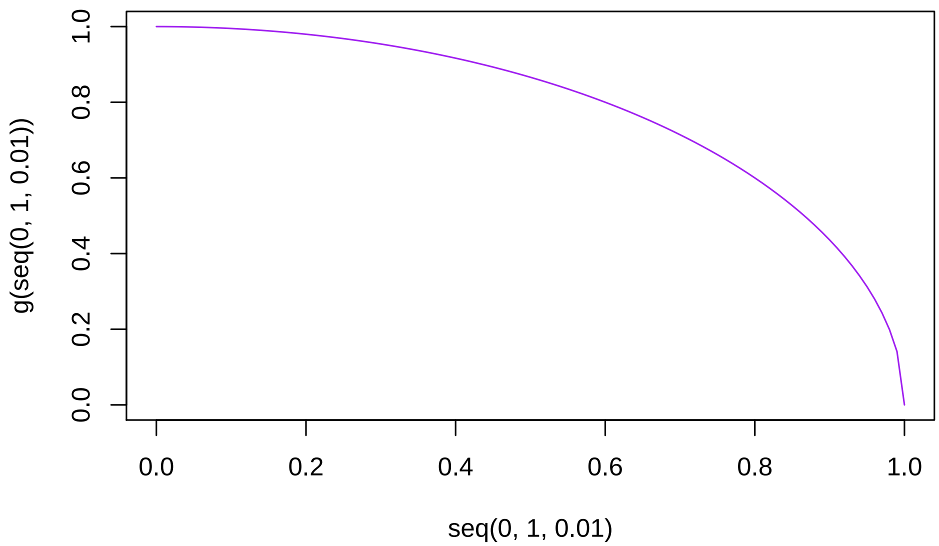
```
#test with seed 33 again
random.walk(seed = 33,plot.walk = F)
```

```
## $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.

18.
Run the given code.

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



19.

Take integral of g(x) on [0,1] and we will get the result of $\frac{\pi}{4}$, this is the area under the curve

Part 4:

19) $g(x) = \sqrt{1-x^2}$, $0 \leq x \leq 1$

True area under the curve is

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

Let $x = \sin(u) \Rightarrow u = \arcsin(x)$ $\Rightarrow \sin(2u) = \sin(2\arcsin(x))$

and $dx = \cos(u)\,du$ $= 2x\sqrt{1-x^2}$

$$\Rightarrow \int_0^1 \sqrt{1-x^2}\,dx = \int_0^1 \cos(u)\sqrt{1-\sin^2(u)}\,du$$

$$= \int_0^1 \cos(u)\cos(u)\,du$$

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

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

$$= \frac{1}{4}(\sin(2u))\Big|_0^1 + \frac{u}{2}\Big|_0^1$$

$$= \frac{x\sqrt{1-x^2}}{2}\Big|_0^1 + \frac{\arcsin(x)}{2}\Big|_0^1$$

$$= 0 + \left(\frac{\pi}{4} - 0\right) = \frac{\pi}{4}$$

20.

With $p(x)$ chosen as the pdf of a uniform distribution on [0,1], $p(x) = 1$ and thus $g(x)/p(x) = g(x) = \sqrt{1-x^2}$.

Using Monte Carlo Integration with 100,000 draws from $p(x)$, the integral is estimated to be 0.7852, which is within 1/1000 of the true value (0.7854) calculated using geometric formulas.

```r
#g(x)/p(x) = g(x) because
g.over.p <- function(x) {
  return(sqrt(1-x^2))
}

set.seed(0)
#estimate using MC integration
mean(g.over.p(runif(100000,min = 0,max = 1)))
```

```
## [1] 0.78518
```

```r
#true value
pi/4
```

```
## [1] 0.7853982
```

```r
set.seed(0)
#difference is within 1/1000
mean(g.over.p(runif(100000,min = 0,max = 1))) - (pi/4)
```

```
## [1] -0.000218197
```