Homework 6

Hanao Li hl3202

November 20, 2018

####Part 1: Inverse Transform Method 1.

If $X \sim Cauchy(\alpha, \beta)$ and $U \sim Unif(0,1)$, then $X = \alpha + \beta tan(\pi(U - 1/2))$

Since we have $X \sim Cauchy(0,1)$, then $X = tan(\pi(U - 1/2))$

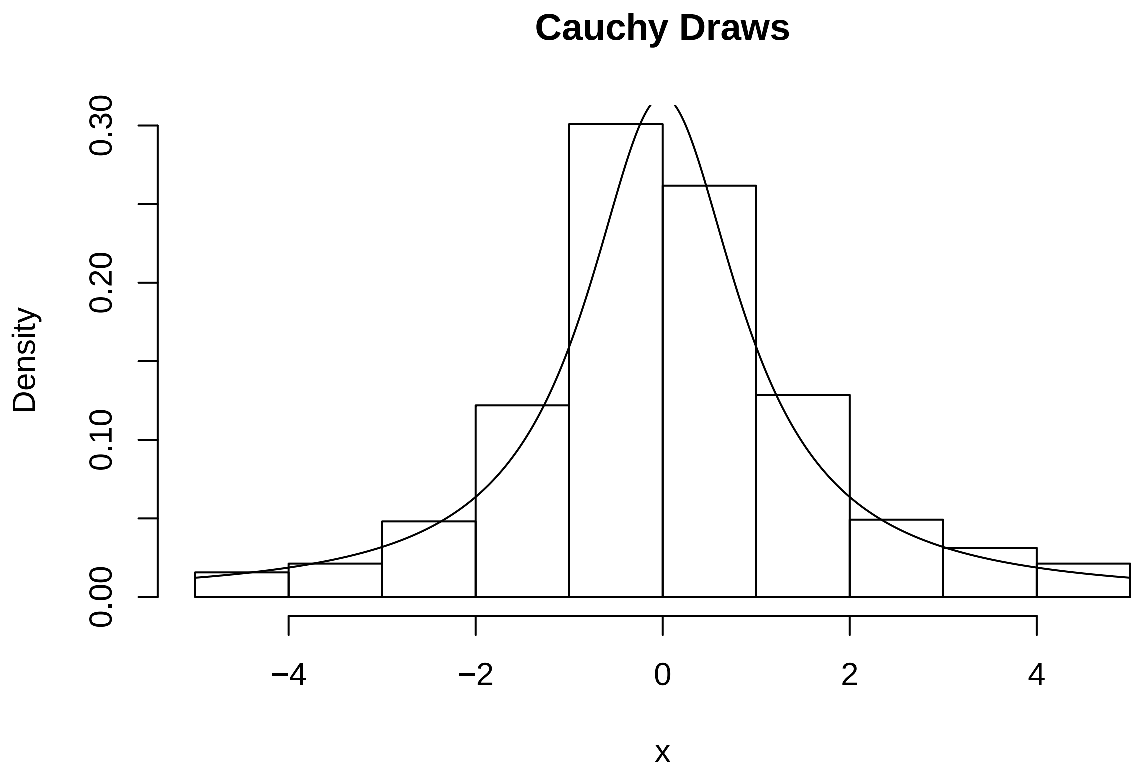
2.

```
cauchy.sim <- function(n){
  u <- runif(n)
  return(tan(pi*(u-0.5)))
}
cauchy.sim(10)
```

```
##  [1] -0.8155170  1.6239486 -2.4251509  1.4982274  0.1161945  0.1935426
##  [7] -0.2744881 -0.3201585 -1.1832637 22.6364942
```

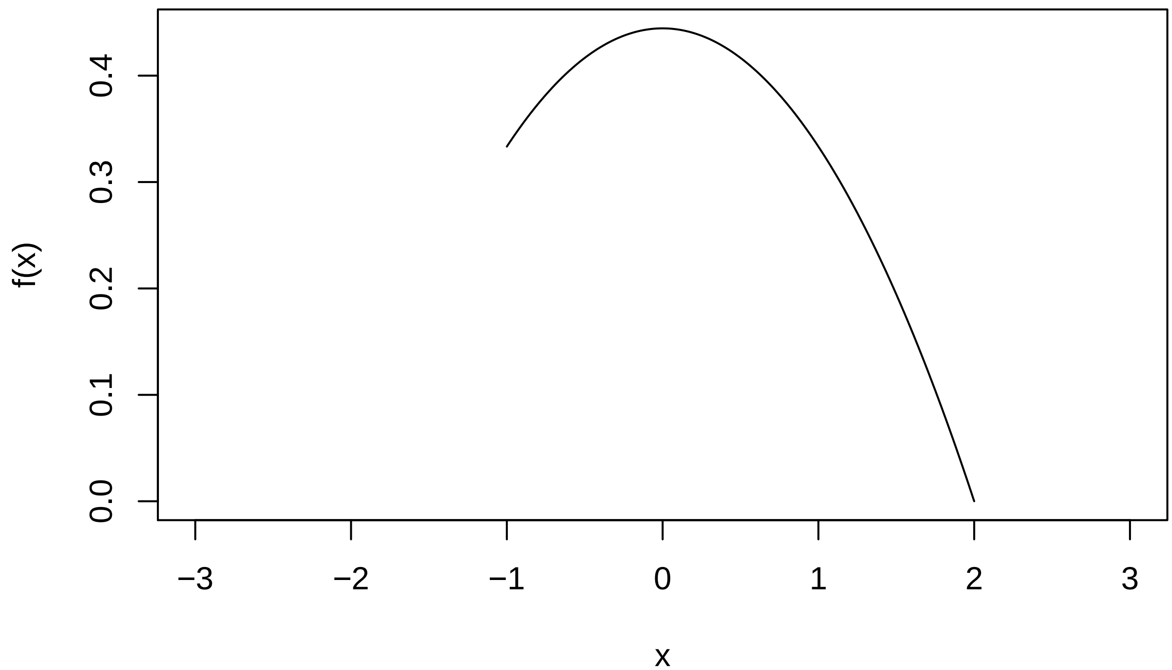
3.

```
cauchy.draws <- cauchy.sim(1000)
hist(cauchy.draws[cauchy.draws < 5 & cauchy.draws > -5], prob = T, main = "Cauchy Draws", xlab = "x")
x <- seq(-5, 5, 0.01)
lines(x,1/pi*1/(1+x^2))
```



####Part 2: Reject-Accept Method 4.

```
f <- function(x){
  return(ifelse((x < -1 | x > 2), 0, (4-x^2)/9))
}
x <- seq(-1,2,0.01)
plot(x, f(x), xlim = c(-3,3), type = 'l')
```



5.

$f'(x) = -\frac{2x}{9} = 0$, then $X_{max} = 0$, $f_{max} = \frac{4}{9}$

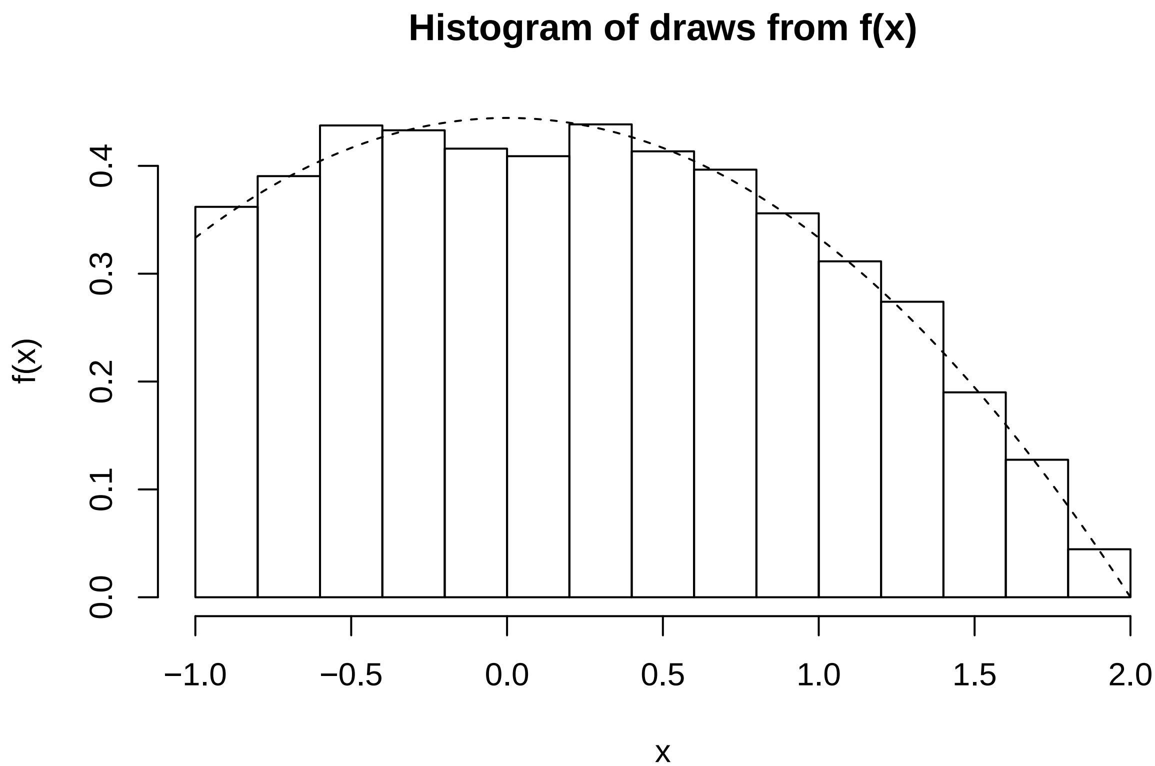
```
x.max = 0
f.max = 4/9
e <- function(x){
  return(ifelse((x < -1 | x > 2), Inf, f.max))
}
```

6.

```
n.samps <- 10000
n <- 0
samps <- numeric(n.samps)
while (n < n.samps){
  y <- runif(1,-1,2)
  u <- runif(1)
  if (u < f(y)/e(y)){
    n <- n + 1
    samps[n] <- y
  }
}
f.draws <- samps
```

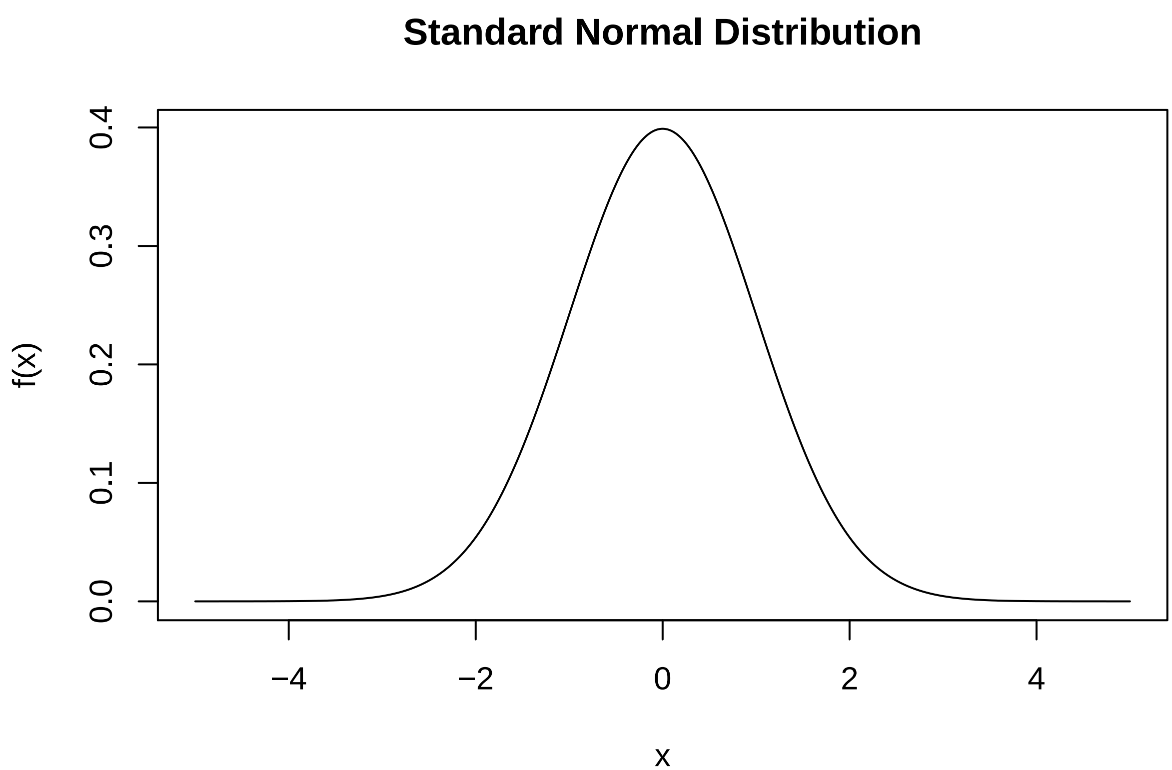
7.

```
hist(f.draws, prob = T, ylab = "f(x)", xlab = "x", main = "Histogram of draws from f(x)")
lines(x, f(x), lty = 2)
```



####Problem 3: Reject-Accpet Method Continued 8.

```
f <- function(x){
  return(1/sqrt(2*pi)*exp(-x^2/2))
}
x <- seq(-5, 5, 0.01)
plot(x, f(x), type = 'l', main = "Standard Normal Distribution")
```

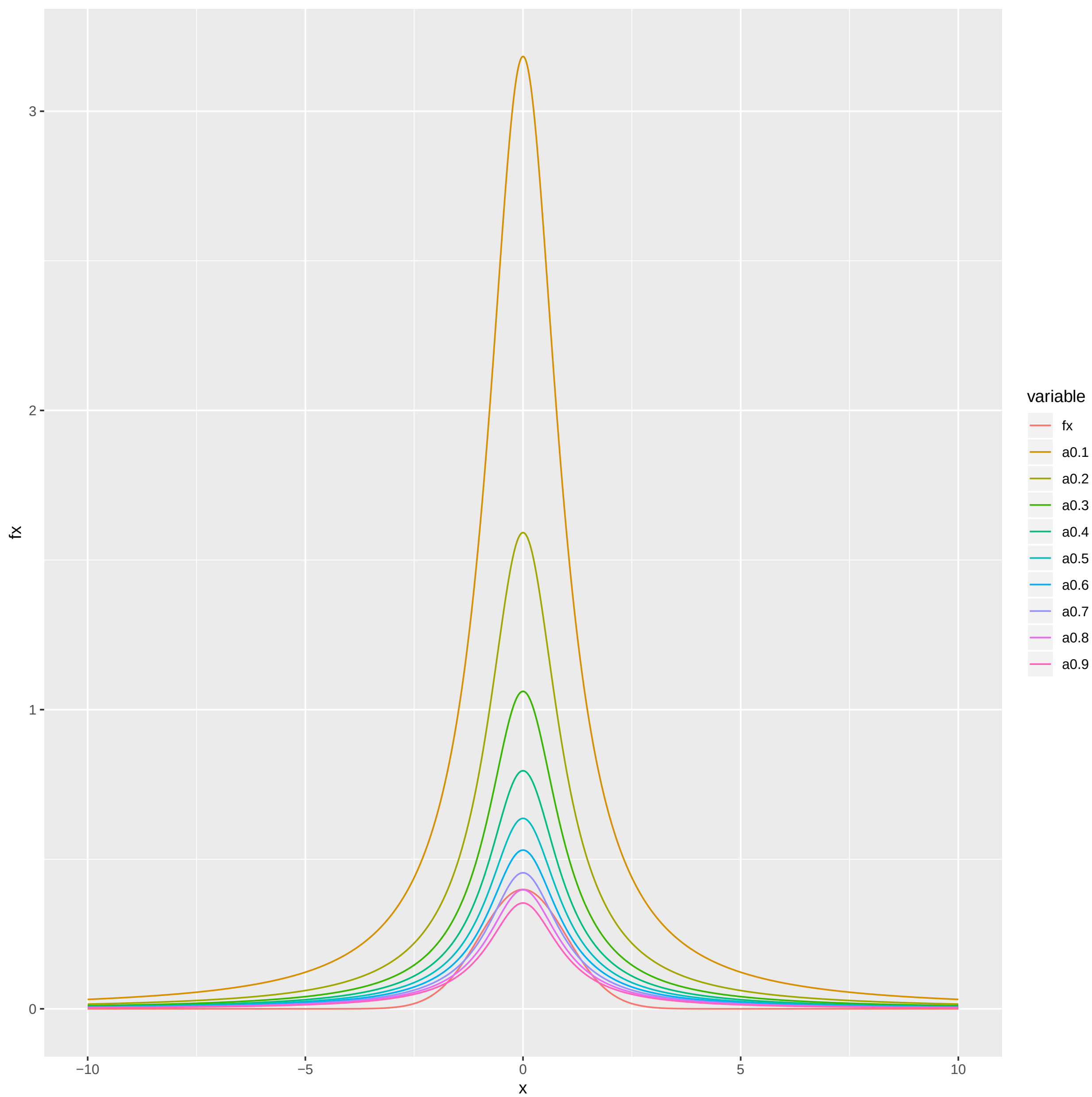


9.

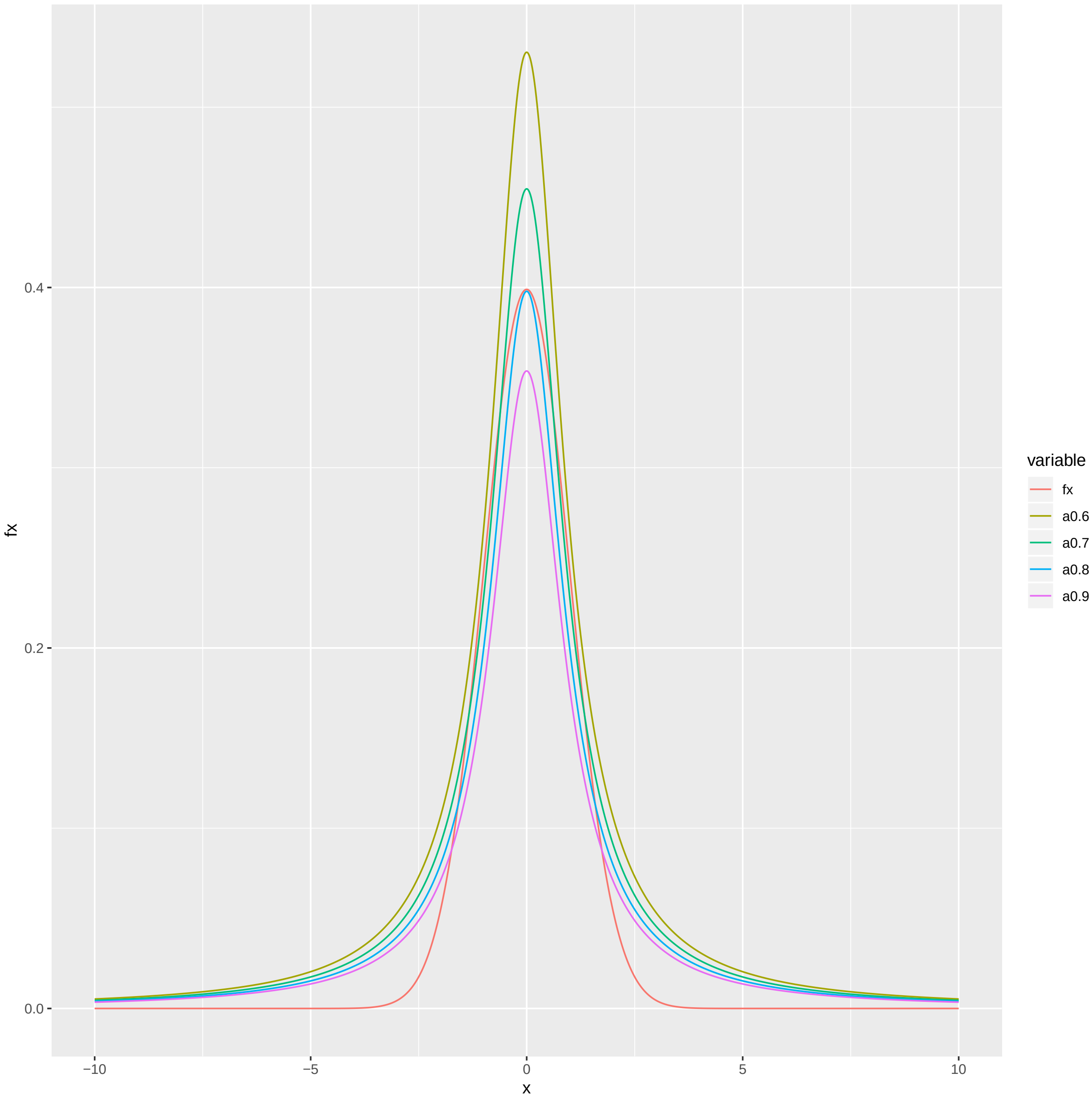
```
e <- function(x,a){
  return((1/pi*1/(1+x^2))/a)
}
```

10.

```
library(ggplot2)
suppressWarnings(library(reshape2))
# We first choose alpha value from 0.1 to 0.9
x <- seq(-10,10,0.01)
fx <- f(x)
alpha <- seq(0.1,0.9,0.1)
m <- list()
for (i in 1:length(alpha)){
  nam <- paste("a", alpha[i], sep = "")
  m[[i]] <- e(x,alpha[i])
  names(m)[i] <- nam
}
m <- do.call(cbind,m)
m <- as.data.frame(cbind(x,fx,m))
newm <- melt(m, id.vars = "x", value.name = "fx")
ggplot(newm, aes(x, fx, col = variable)) +
  geom_line()
```




```r
# Alpha seems to be good from 0.6 to 0.9

m <- list()
alpha <- seq(0.6,0.9,0.1)
for (i in 1:length(alpha)){
  nam <- paste("a", alpha[i], sep = "")
  m[[i]] <- e(x,alpha[i])
  names(m)[i] <- nam
}
m <- do.call(cbind,m)
m <- as.data.frame(cbind(x,fx,m))
newm <- melt(m, id.vars = "x", value.name = "fx")
ggplot(newm, aes(x, fx, col = variable)) +
  geom_line()
```




```
# Alpha is good at 0.8
```

11.

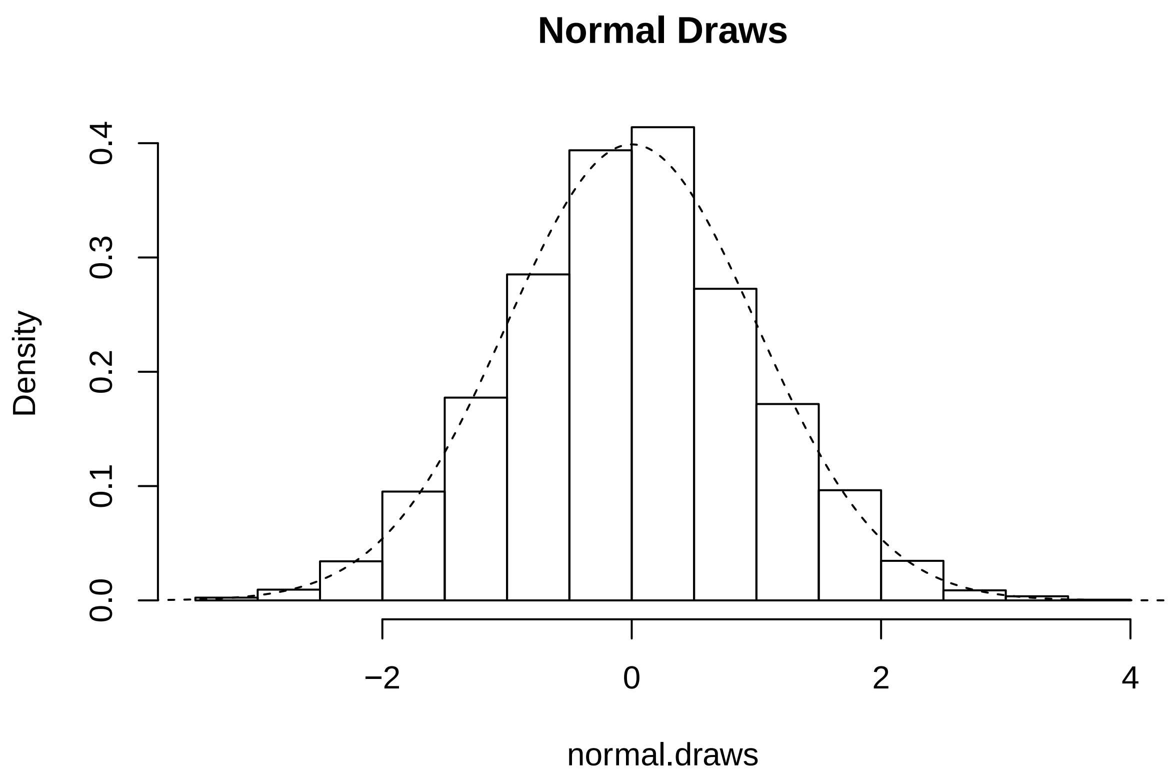
```
normal.sim <- function(n){
  n.samps <- n
  n <- 0
  samps <- numeric(n.samps)
  a = 0.8
  while (n < n.samps){
    y <- cauchy.sim(1)
    u <- runif(1)
    if (u < f(y)/e(y,a)){
      n <- n + 1
      samps[n] <- y
```

```
    }
  }
  return(samps)
}
normal.sim(10)
```

```
##  [1]  0.73607864  0.91088131 -0.06323431  0.11270693  0.02508774  0.56185361
##  [7] -1.73153476 -0.08200524 -1.23690743  1.61518216
```

12.

```
normal.draws <- normal.sim(10000)
hist(normal.draws, prob = T, main = "Normal Draws")
x <- seq(-5, 5, 0.01)
lines(x, dnorm(x), lty = 2)
```



####Part 3: Simulation with Built-in R Functions 13.

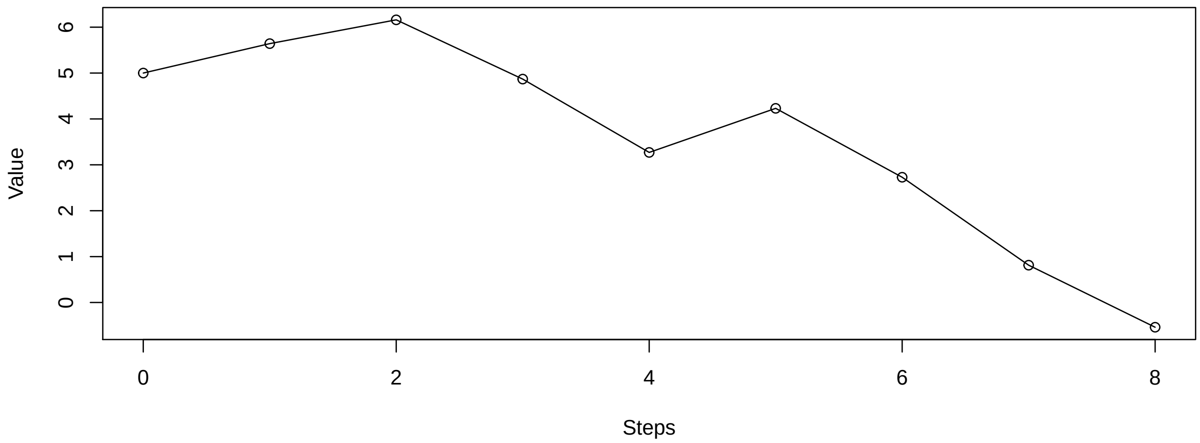
```
x <- 5
x.vals <- c(x)
while (x > 0){
  r <- runif(1, -2, 1)
  x <- x + r
  x.vals <- c(x.vals, x)
}
x.vals
```

```
## [1]  5.0000000  5.6414627  6.1594880  4.8686355  3.2703491  4.2308837  2.7291568
```

```
## [8]  0.8131343 -0.5392317
```

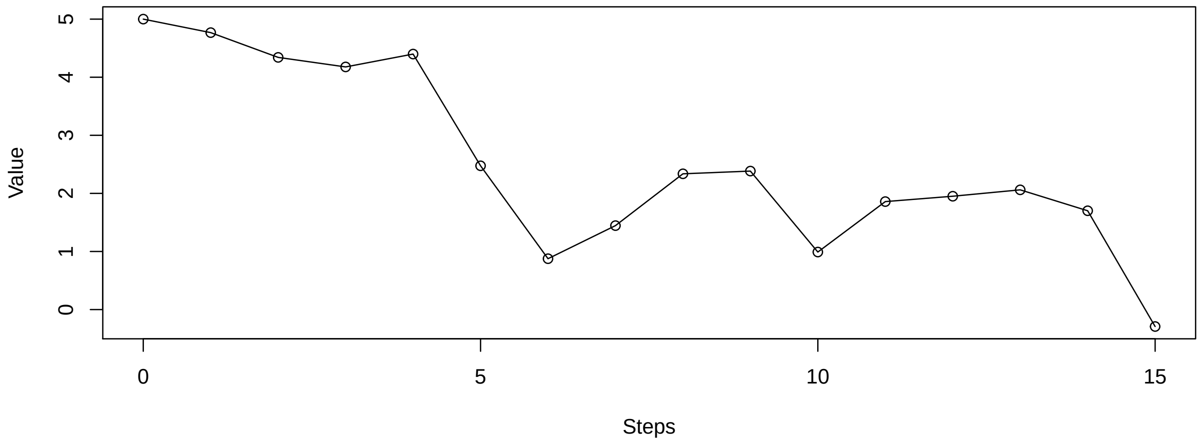
14.

```r
plot(0:(length(x.vals) - 1), x.vals, type = 'o', xlab = "Steps", ylab = "Value")
```



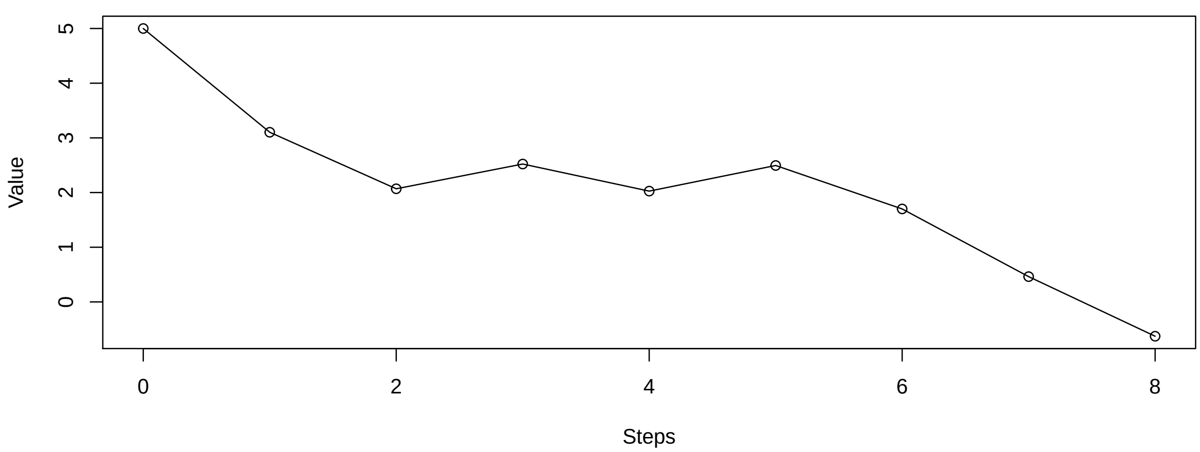
15.

```r
random.walk <- function(x.start = 5, plot.walk = TRUE){
  x.vals <- c(x.start)
  num.steps <- 0
  while (x.start > 0){
    r <- runif(1, -2, 1)
    x.start <- x.start + r
    x.vals <- c(x.vals, x.start)
    num.steps <- num.steps + 1
  }
  if (plot.walk == TRUE){
    plot(0:num.steps, x.vals, type = 'o', xlab = "Steps", ylab = "Value")
  }
  output <- list(x.vals, num.steps)
  names(output) <- c("x.vals", "num.steps")

  return(output)
}

random.walk()
```




```
## $x.vals
##  [1]  5.0000000  4.7677850  4.3413215  4.1772027  4.3985265  2.4749181
##  [7]  0.8755790  1.4453621  2.3370404  2.3839827  0.9901354  1.8588850
## [13]  1.9510849  2.0612143  1.7010838 -0.2916823
##
## $num.steps
## [1] 15
```

```r
random.walk()
```




```
## $x.vals
## [1]  5.0000000  3.1027987  2.0690005  2.5216511  2.0262637  2.4950284  1.7004206
## [8]  0.4623079 -0.6276088
##
## $num.steps
## [1] 8
```

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

```
## $x.vals
##  [1] 10.0000000  8.0129886  7.5870304  6.5092611  5.2595840  4.4866026
##  [7]  3.6251574  2.7411669  0.7556797 -0.2023972
##
## $num.steps
## [1] 9
```

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

```
## $x.vals
##  [1] 10.0000000 10.6631103  9.6757474  9.3282607  8.6160893  7.9767263
##  [7]  6.3293430  4.8584431  5.4272541  6.0685201  6.8793571  7.5588311
## [13]  7.9146708  6.5358636  6.5289751  6.0856862  4.4307243  3.7010518
## [19]  3.7430589  3.0336377  3.5115856  3.6739033  2.5657297  2.0619642
## [25]  1.1943557  0.2690885 -0.2227623
##
## $num.steps
## [1] 26
```

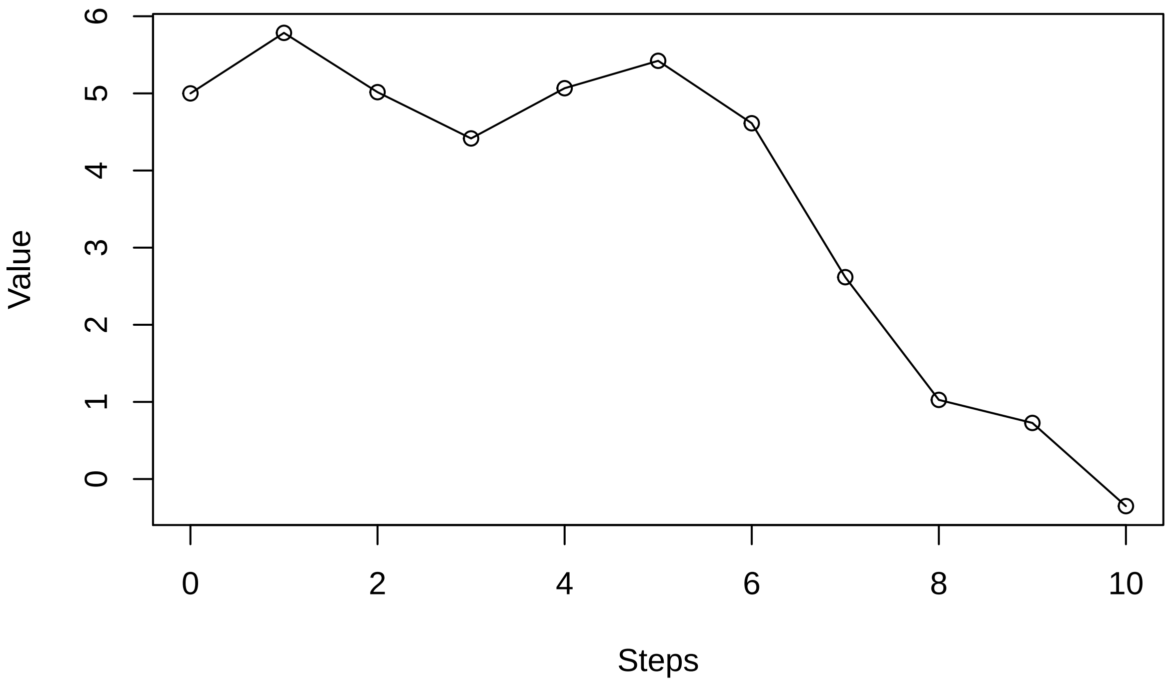
16.

```
# The expected number of iterations should be 5/(1/2) + 1 = 11
steps <- c()
for (i in 1:10000){
  random.walk(, F)
  steps[i] <- random.walk(, F)[[2]]
}
mean(steps)
```

```
## [1] 11.297
```

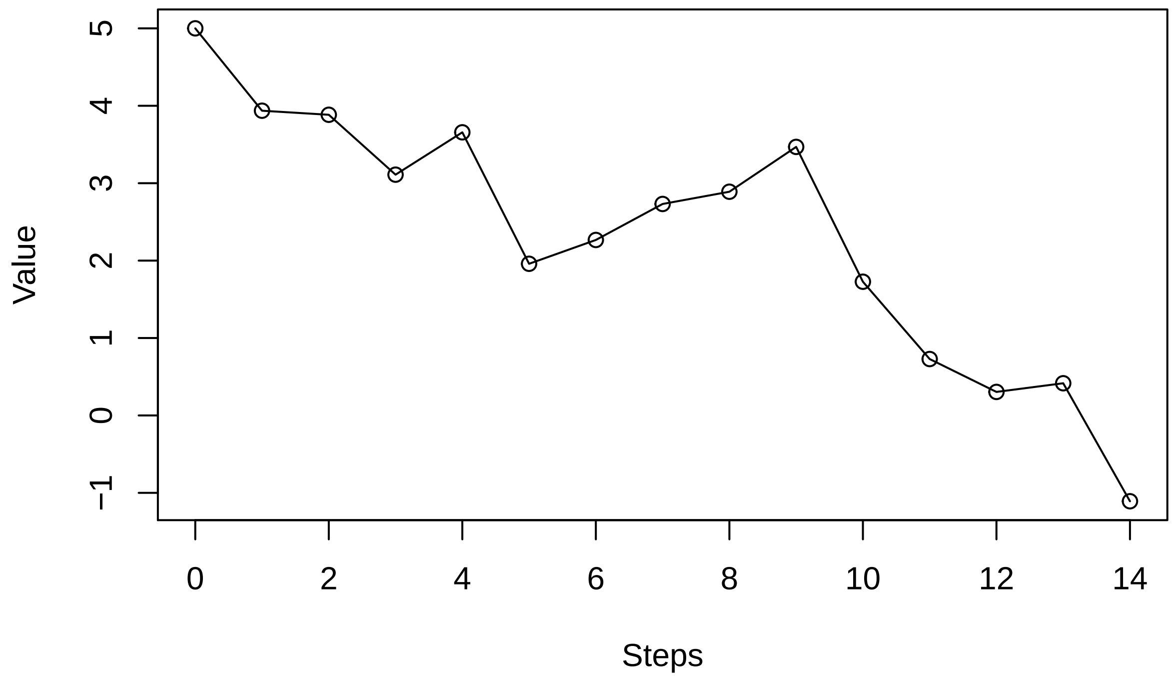
17.

```
random.walk <- function(x.start = 5, plot.walk = TRUE, seed = NULL){
  if (is.null(seed) == FALSE){
    set.seed(seed)
  }
  x.vals <- c(x.start)
  num.steps <- 0
  while (x.start > 0){
    r <- runif(1, -2, 1)
    x.start <- x.start + r
    x.vals <- c(x.vals, x.start)
    num.steps <- num.steps + 1
  }
  if (plot.walk == TRUE){
    plot(0:num.steps, x.vals, type = 'o', xlab = "Steps", ylab = "Value")
  }
  output <- list(x.vals, num.steps)
  names(output) <- c("x.vals", "num.steps")

  return(output)
}

random.walk()
```




```
## $x.vals
##  [1]  5.0000000  5.7844984  5.0157528  4.4157721  5.0668945  5.4223319
##  [7]  4.6126752  2.6171137  1.0260696  0.7270525 -0.3511712
##
## $num.steps
## [1] 10
```

```r
random.walk()
```




```
## $x.vals
##  [1]  5.0000000  3.9360707  3.8836874  3.1099632  3.6570394  1.9589386
##  [7]  2.2666793  2.7316262  2.8897985  3.4687411  1.7270853  0.7285992
## [13]  0.3037413  0.4157197 -1.1082456
##
## $num.steps
## [1] 14
```

```r
random.walk(, F, 33)
```

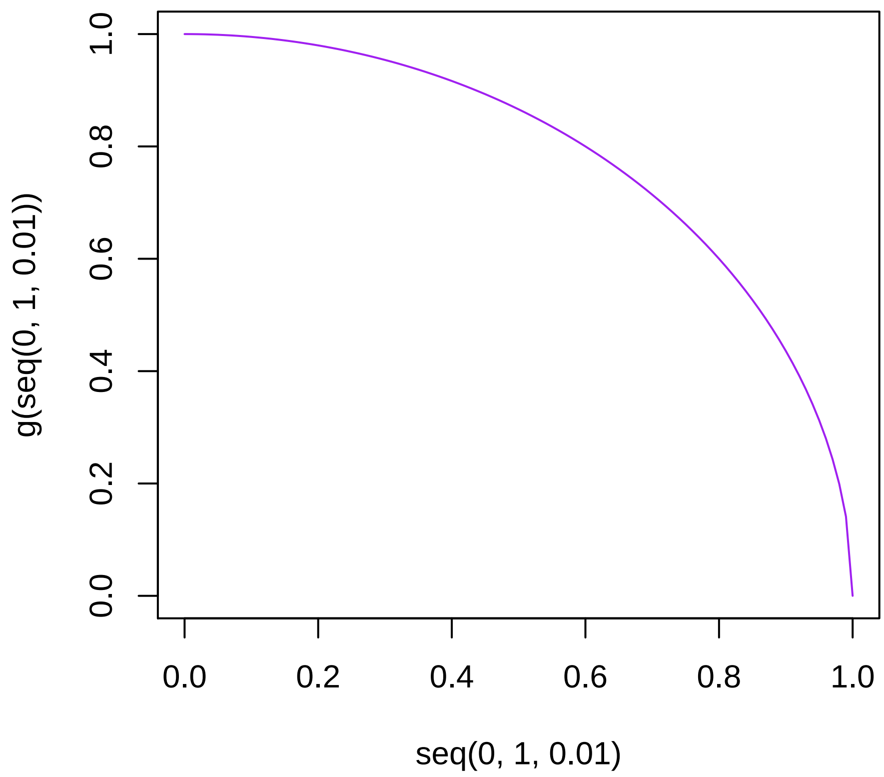
```
## $x.vals
##  [1]  5.0000000  4.3378214  3.5217724  2.9729590  3.7295869  4.2612312
##  [7]  3.8132800  3.1246550  2.1542497  0.2008006 -1.4452259
##
## $num.steps
## [1] 10
```

```r
random.walk(, F, 33)
```

```
## $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] 10
```

####Part 4: Monte Carlo Integration 18.

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



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

20.

```r
# Choose uniform distribution when perform this simulation
g.over.p <- function(x){
  return(4*sqrt(1-x^2))
}
mean(g.over.p(runif(10000000)))
```

```
## [1] 3.141666
```

```r
pi
```

```
## [1] 3.141593
```

```r
pi - mean(g.over.p(runif(10000000)))
```

```
## [1] -0.0002403507
```

```r
1/1000 - abs(pi - mean(g.over.p(runif(10000000))))
```

```
## [1] 0.0005720405
```

```r
# So we estimated the value of pi within a 1/1000 of the true value
```