Project 4

Zihuan Qiao

11/1/2016

1. Polar Method

(a) Box-Muller Polar Method

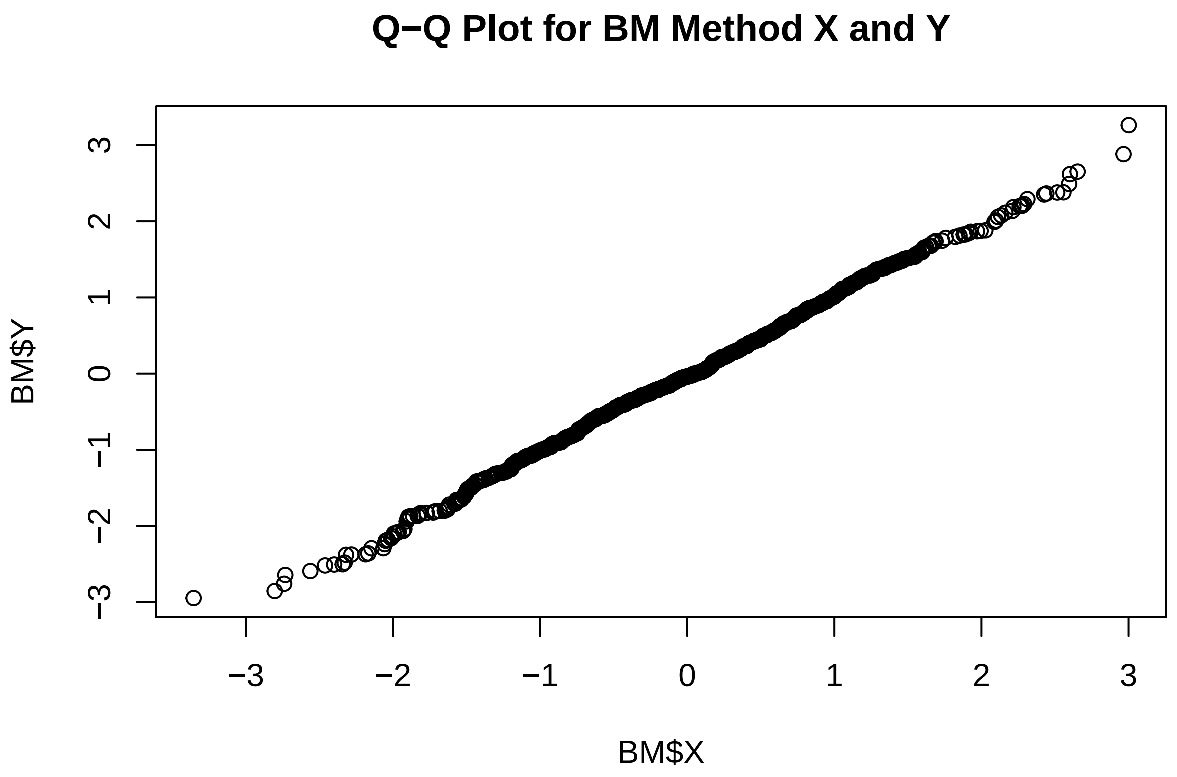
```r
BMnorm <- function(n) #sample size is n
{
  #initialize X and Y
  X <- rep(0,n); Y <- rep(0,n)

  #Sample R: inverse cdf method with latent variable Z ~ Bern(.5)
  U1 <- runif(n,0,1); U2 <- runif(n,0,1)
  Z <- (U1<.5) * 1 #latent variable Z ~ Bern(.5)
  U <- sqrt(-2 * log(U2)) #inverse cdf method
  R <- (2 * Z - 1) * U #Mixture model

  #Sample theta
  theta <- runif(n,0,2*pi)

  #Sample X & Y
  X <- R*cos(theta)
  Y <- R*sin(theta)

  out <- list(X = X, Y = Y)

  return(out)
}
```

(b) Marsaglia Polar Method

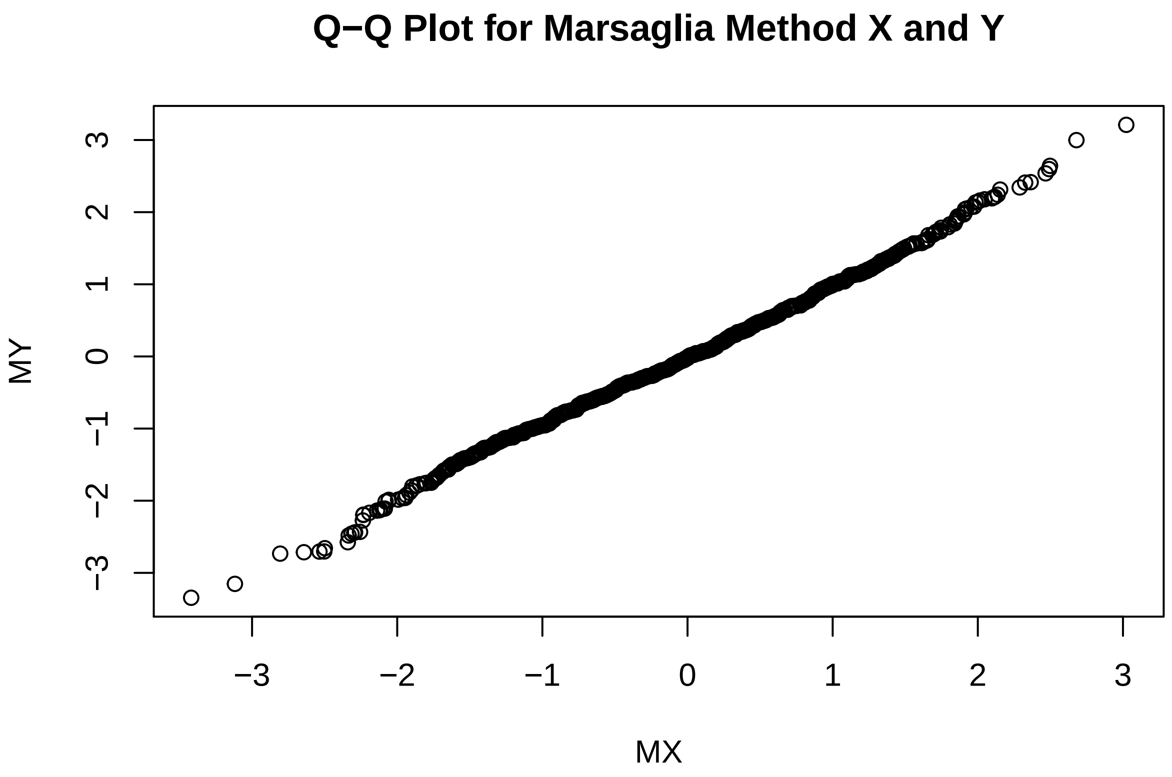
```r
Mnorm <- function()
{
  #Step 1.
  S <- 0
  while (S>=1 | S==0) {
    X <- runif(1,-1,1)
    Y <- runif(1,-1,1)
    S <- X^2+Y^2
  }
  #Step 2.
  X <- X*sqrt((-2*log(S))/S)
  Y <- Y*sqrt((-2*log(S))/S)
  result <- list(X=X,Y=Y)
  return(result)
}
```

(c)

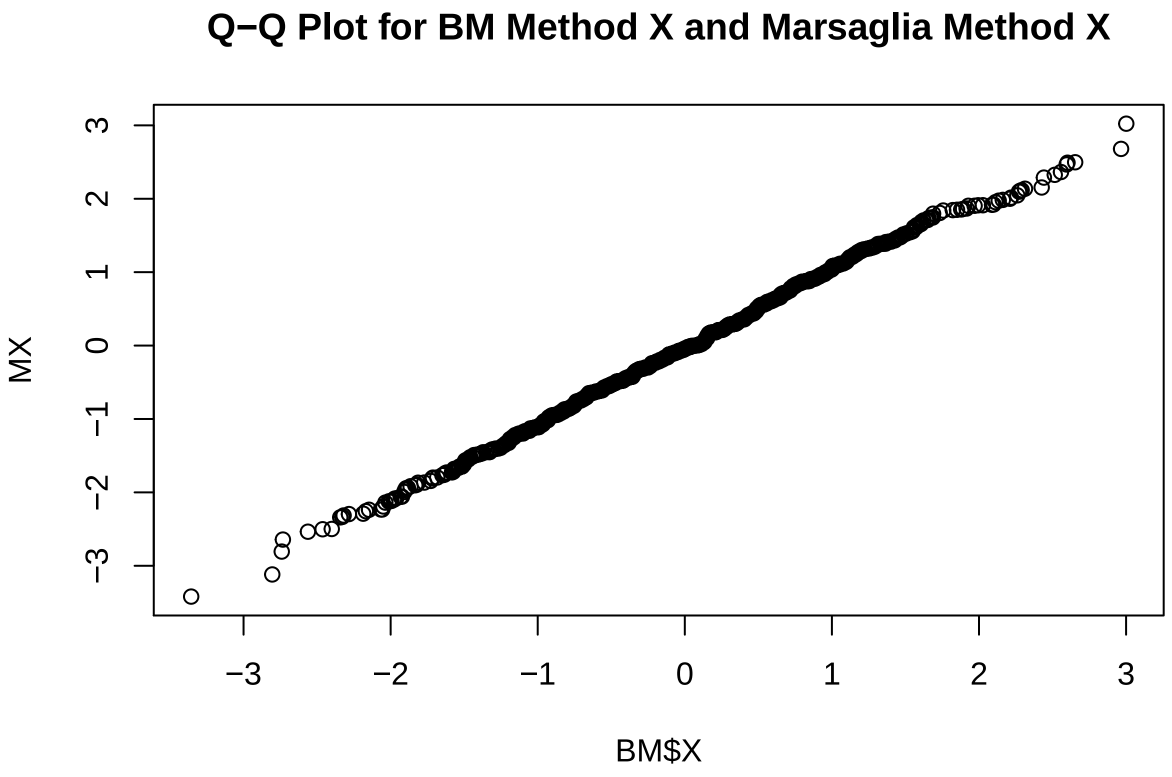
```r
BM <- BMnorm(1000)
MX <- rep(0,1000)
MY <- rep(0,1000)
for(i in 1:1000)
{
  M <- Mnorm()
  MX[i] <- M$X
  MY[i] <- M$Y
}

#QQplot of two sample sets
qqplot(BM$X,BM$Y,main="Q-Q Plot for BM Method X and Y")
```



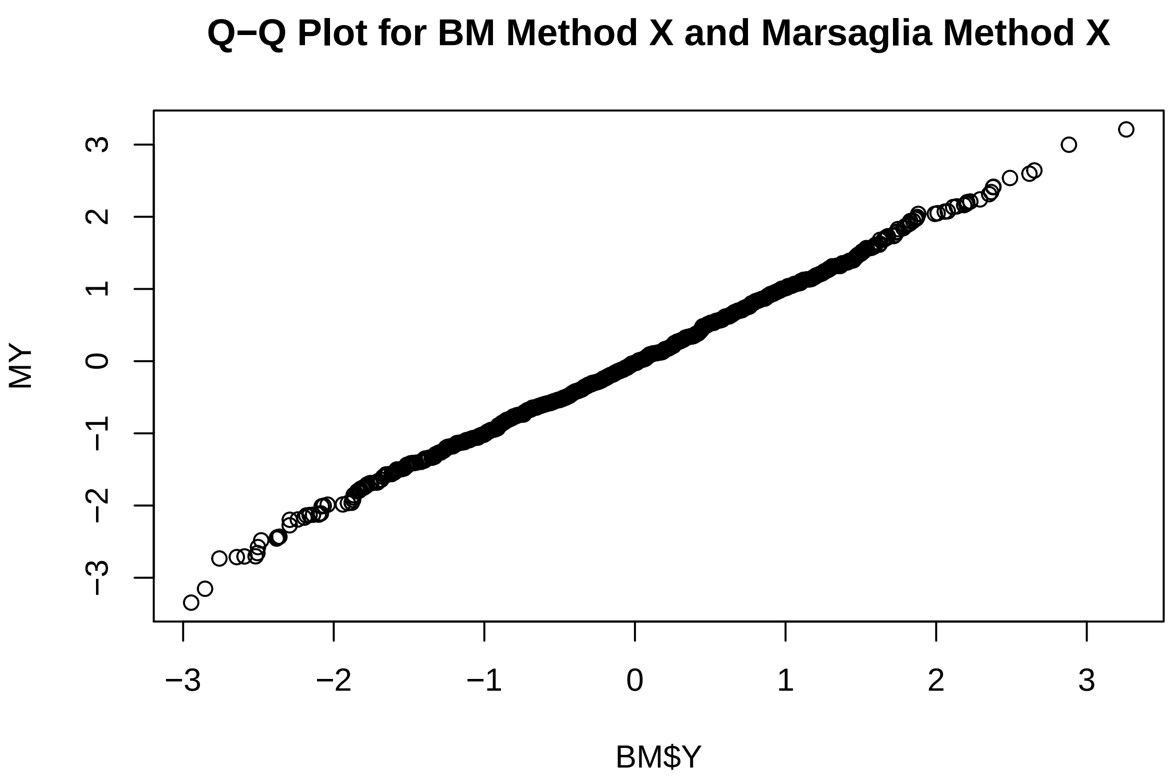
```r
qqplot(MX,MY,main="Q-Q Plot for Marsaglia Method X and Y")
```



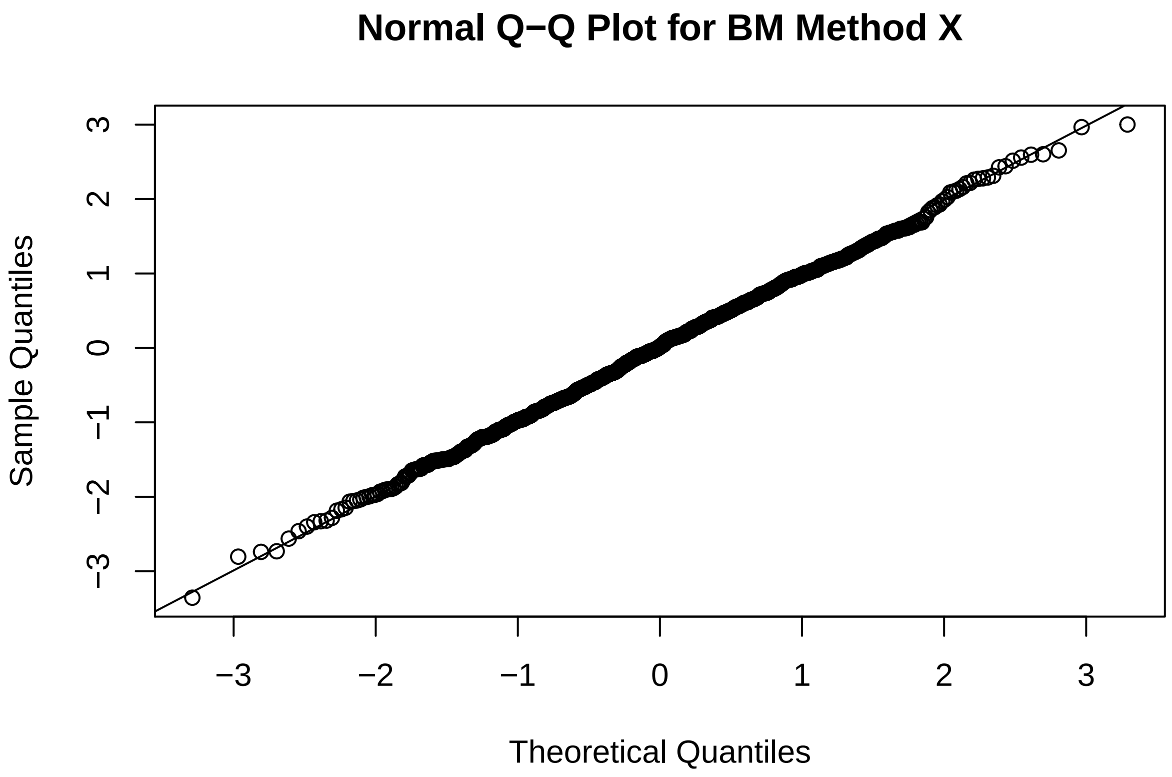
```
qqplot(BM$X,MX,main="Q-Q Plot for BM Method X and Marsaglia Method X")
```



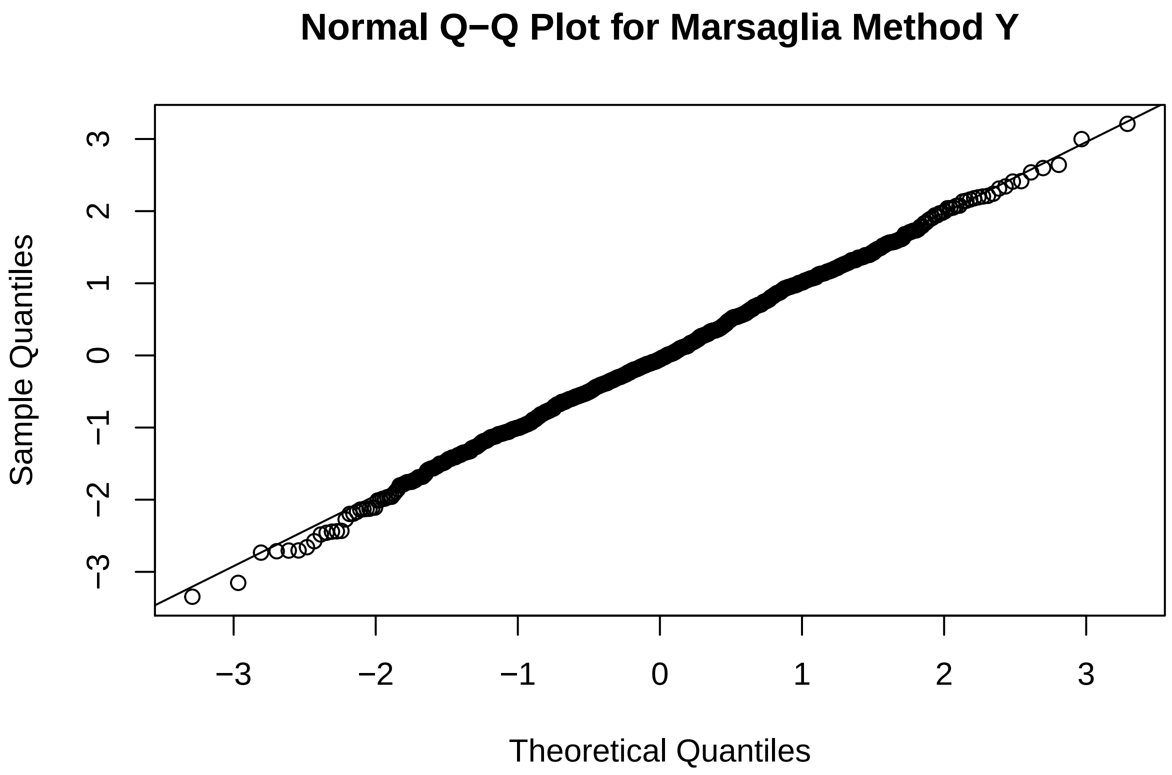
```
qqplot(BM$Y,MY,main="Q-Q Plot for BM Method X and Marsaglia Method X")
```



```r
#qqnorm to compare to the standard normal
qqnorm(BM$X,main = "Normal Q-Q Plot for BM Method X")
qqline(BM$X)
```



```r
qqnorm(MY,main = "Normal Q-Q Plot for Marsaglia Method Y")
qqline(MY)
```



From the qqplots, we can see that sample sets obtained by the BM method and the Marsaglia method should follow the same distribution because qqplot is a diagnol straight line. And according to the qqnorms, the sample sets should follow the normal distribution.

2.Sample from Truncated normal distribution

(a)

```r
rs.supp <- function(n) #sample size is n
{
  supp.x <- rep(0,n)
  k <- 0 #sample times is k
  for(i in 1:n)
  {
    #sample from Y until Y>0
    repeat{
      y <- rnorm(1,-3,1)
      k <- k+1
      if(y>0)
        break
    }
    #Take the sample for X
    supp.x[i]=y
  }
  return(list(x=supp.x, sample.time=k))
}
```

Because P(X<=x) = P(Y<=x|Y>0). So when Y>0, P(X<=x) = P(Y<=x). So when y>0, f(x)=P'(X<=x), g(x)=P'(Y<=x), f(x)=g(x). The envelope function is g(x)=P'(Y<=x) and M=1. In this case, f(x)/Mg(x)=1, so we don't need to sample U~U(0,1) here. Every y we sample when y>0 can be taken as sample from X.

(b)

```r
rs.g <- function(n) #sample size is n
{
  supp.x <- rep(0,n)
  k <- 0 #sample times is k
  for(i in 1:n)
  {
    #rejection sampling: envelop <- g(x), M <- exp(-2/9). How to find M, please see the attachment.(bas
  repeat{
    y <- rexp(1,3)
    U <- runif(1)
    k <- k+1
    if(log(U)<=-.5*(y^2))
      break
  }
    supp.x[i]=y
  }

  return(list(x=supp.x, sample.time=k))
}
```

My criterion please see the attachment.

(c)

```r
#Sample efficiency: the proportion of accepted samples
#simulations with 1000 sample results
se.supp <- 1000/rs.supp(1000)$sample.time
se.g <- 1000/rs.g(1000)$sample.time
se.supp
```

```
## [1] 0.001405306
```

```r
se.g
```

```
## [1] 0.9124088
```

```r
#Difference in sample efficiency
se.g-se.supp
```

```
## [1] 0.9110035
```

First two results show the proportion of accepted samples of the method proposed in (a) and (b) respectively. And the third result shows the difference in sample efficiency between these two methods which is very very big, bigger than 90%.

3.

```
#Given function g(x)
g <- function(x)
{
  abs(sin(2*pi*x[1]*sum(x)))*(cos(3*pi*x[2]*sum(x^2)))^2
}


#Determine I.hat by Monte Carlo Simulation
rx <- runif(100)
gx <- g(rx)
n <- 2
repeat
{
  gx <- append(gx, g(runif(100)))
  I.hat <- mean(gx)
  #determine confidence interval length with significance 95%, so Z_alpha/2 = 1.96
  ci.length <- 1.96*2*sqrt(var(gx)/n)
  n <- n+1
  if(ci.length<=0.015)
    break
}

I.hat
```

```
## [1] 0.3129459
```

4.

```
rwalk <- function (p)
{
  j <- 1 # start
  walk <- c() # store movements
  repeat {
    j <- j + (2 * rbinom(1, 1, p) - 1) # move
    walk <- append(walk, j)
    if (j == 0 || j == 20) return(walk)
  }
}
```

(a)

```
n <- 100000
bp <- c() #bp store the destination for each simulation. When bp[i]=1, ends up at BP, bp[i]=0, ends up
l <- c() #l store the length of each walk
Dugout <- c() #Dugout store the times that the student will be in front of this pub
for (i in 1:n)
{
  rw <- rwalk(.5) #Since coin is fair, p=.5
  l[i] <- length(rw)
  Dugout[i] <- length(which(rw==18))
  if(rw[l[i]]==20)
    bp[i] <- 1
  else
    bp[i] <- 0
}

p.hat <- sum(bp)/n
varbp <- var(bp)
#95% confidence interval: Z_alpha/2 = 1.96
ci.low <- p.hat-1.96*sqrt(varbp/n)
ci.high <- p.hat+1.96*sqrt(varbp/n)

p.hat
```

```
## [1] 0.05032
```

```
ci.low
```

```
## [1] 0.04896507
```

```
ci.high
```

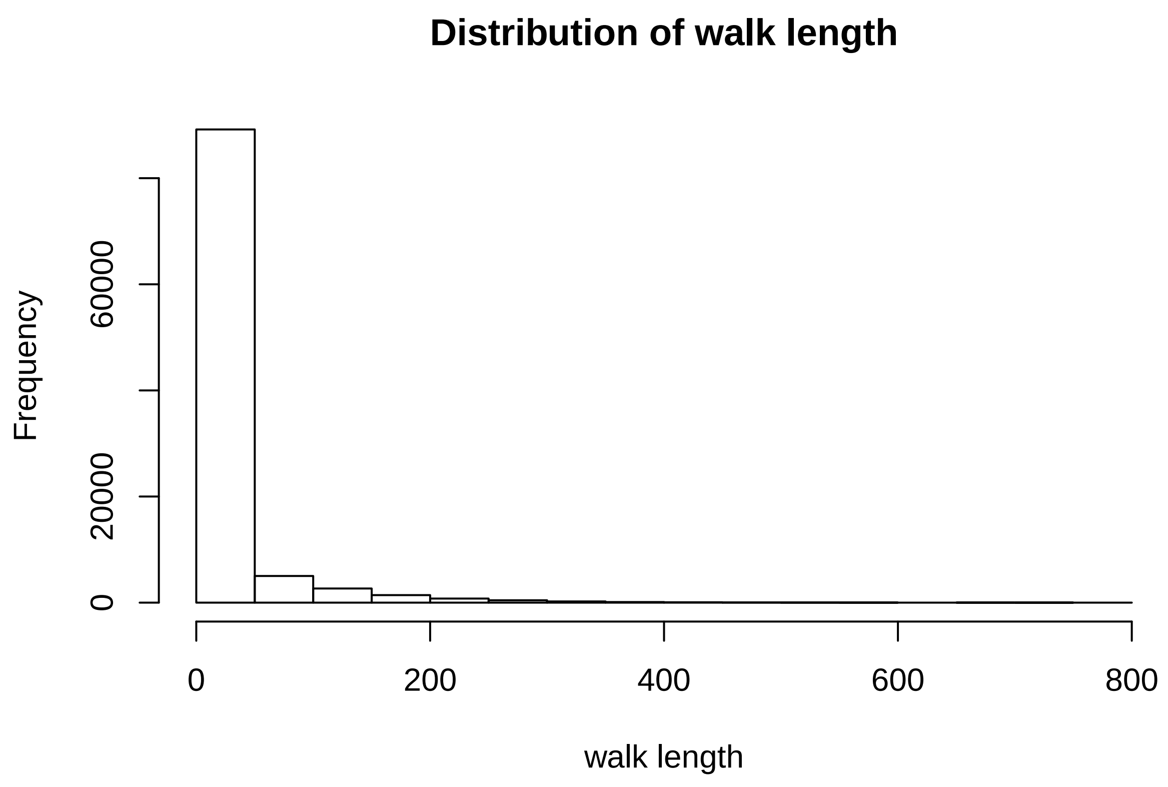
```
## [1] 0.05167493
```

```
interval.length <- ci.high-ci.low
interval.length
```

```
## [1] 0.002709862
```

(b)

```
hist(l,xlab = "walk length", main = "Distribution of walk length")
```



```r
#obtain the probability of less than 200 steps: that is F(200)=P(l<=200)=F.hat0
#obtain F.hat0 by Monte Carlo Estimate
F.hat0 <- mean((l<=200)*1)
F.hat0
```

```
## [1] 0.98297
```

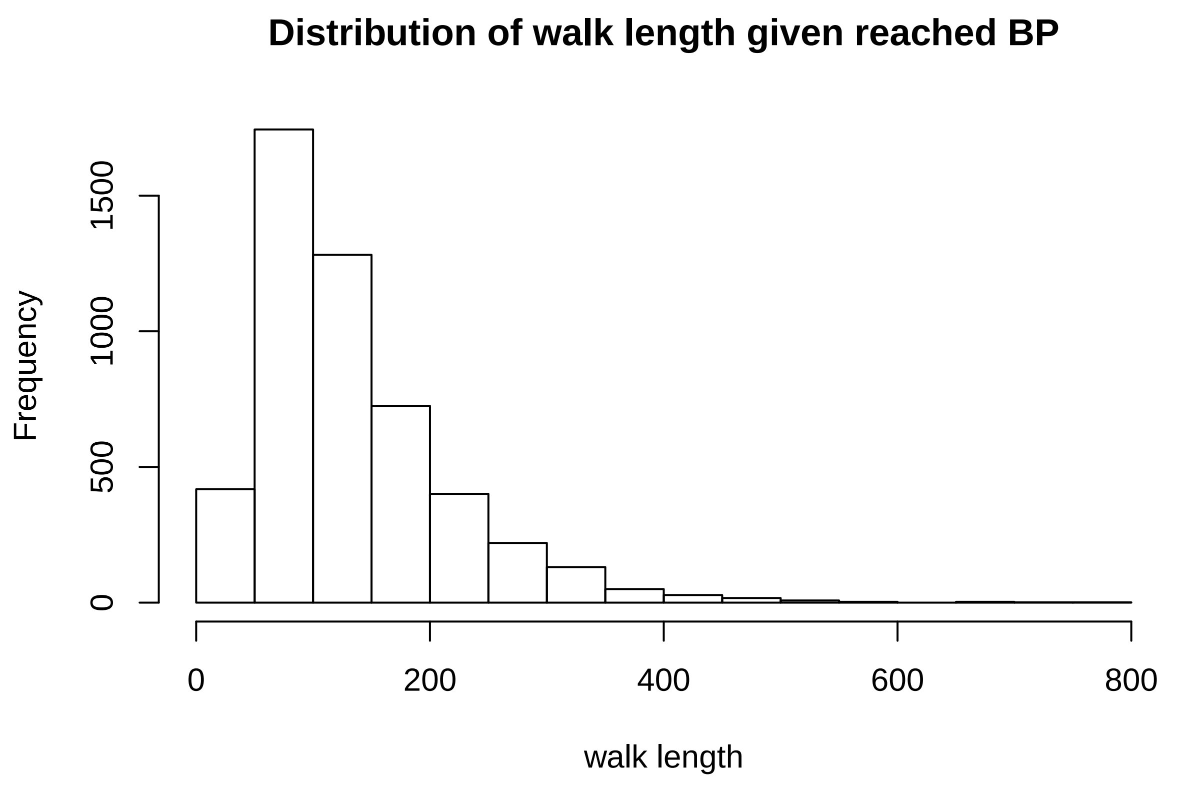
```r
#obtain F.hat: the probability of more than 200 steps
F.hat <- 1-F.hat0
F.hat
```

```
## [1] 0.01703
```

The shape of the distribution is shown in the histogram.

(c)

```r
#find sample index that reach BP
index <- which(bp==1)
#find sample length that reach BP
ll <- l[index]

hist(ll,xlab = "walk length", main = "Distribution of walk length given reached BP")
```



The shape of distribution in (c) increases first than decrease while the shape of distribution in (b) only decreases and decreases very quickly. And walk length distributes more evenly in (c) than in (b).

(d)

```
Dugout.hat <- mean(Dugout)
Dugout.hat
```

```
## [1] 0.20436
```

(e)

```
#determine g(x): Probability of ending up at BP with a loaded probability
n <- 100000
bpl <- c() #bpl store the destination for each simulation of loaded probability. When bpl[i]=1, ends up
lll <- c()
for (i in 1:n)
{
  rw <- rwalk(.55) #Since coin is loaded, p=.55
  lll[i] <- length(rw)
  if(rw[lll[i]]==20)
    bpl[i] <- 1
  else
    bpl[i] <- 0
}
```

```r
p.hatl <- sum(bpl)/n

#determine weight: f(x) is Probability of ending up at BP with a fair probability
weight <- p.hat/p.hatl
weight
```

```
## [1] 0.273404
```

(f)

```r
#reestimate using IS
ppb <- weight*p.hatl
ppb
```

```
## [1] 0.05032
```

```r
#standard deviation of MC method and IS method
se.MC <- sd(bp)
se.IS <- sd(weight*bpl)
se.MC/se.IS
```

```
## [1] 2.063262
```