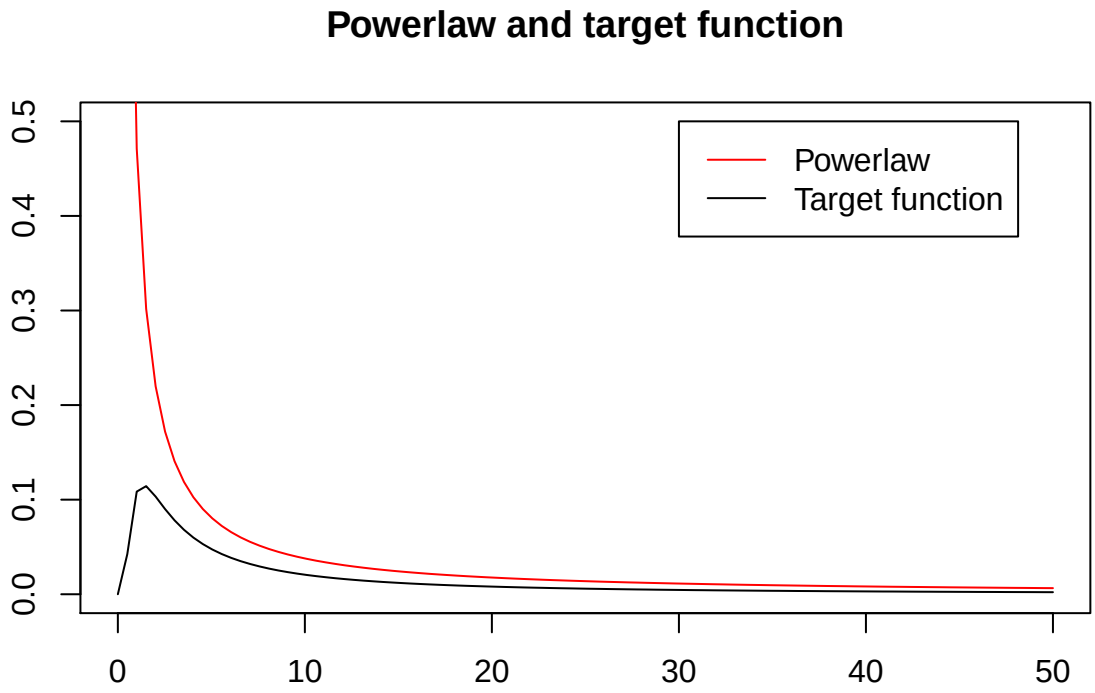
# Lab 3

Shwetha, Suhani , Hoda

11/21/2020

## 1

```
f = function(x,c){
  res = 0
  res = c * ((sqrt(2*pi))^-1) * exp(-(c^2)/(2*x)) * x^(-3/2)
  res[x<=0]<-0
  return(res)
}
fp = function(x, a, tmin) {
  return((a - 1 / tmin) * (x / tmin)^(-a))
  }
c = 2
tmin =  1.33
a = 1.1
x = seq(0,50,length.out = 100)
plot(x, f(x,c), type = "l", ylim = c(0,0.5),xlab = "", ylab = "")
lines(x,fp(x,a,tmin),col="red")
title("Powerlaw and target function")
legend(30,0.5,legend = c("Powerlaw","Target function"), lty = 1, col = c("red","black"))
```




The support for power law is from tmin to infinity , but our target function is on support from 0 to infinity. Hence we cannot use power law by itself. However we can combine it with uniform distribution for 0 to tmin support. Since the max(f(x)) is achieved when the value of x is c^2/3, we have choosen tmin value to be the same, because powerlaw is strictly decreasing funtion and maximum of it will be at tmin. Alpha has been choosen by visually deciding which value makes powerlaw function similar to our target function.

Majorizing density function is combination of uniform density and powerlaw density function.

$$Majoritydensity = (probability * uniformdensity * 1[0, tmin]) + (1 - probability * powerlawdensity * 1[tmin, \inf]) whereprobabli$$

Then we can find majorizing constant as follows

$$MajoringConstant = max\left(\frac{f(x)}{majoritydensity(x)}\right)$$

```
p = 0.0832 #for c = 2
#x = seq(0,100,length.out = 1000)
max = max(f(x,c))
nd = function(x,tmin,a,p){
  res = c()
  for(i in 1:length(x)){
    if(x[i]>=0 && x[i]<= tmin){
      res[i] = p * dunif(x[i],0,tmin)
    }
    if(x[i]>tmin){
      res[i] = (1-p) * fp(x[i],a,tmin)
    }
```

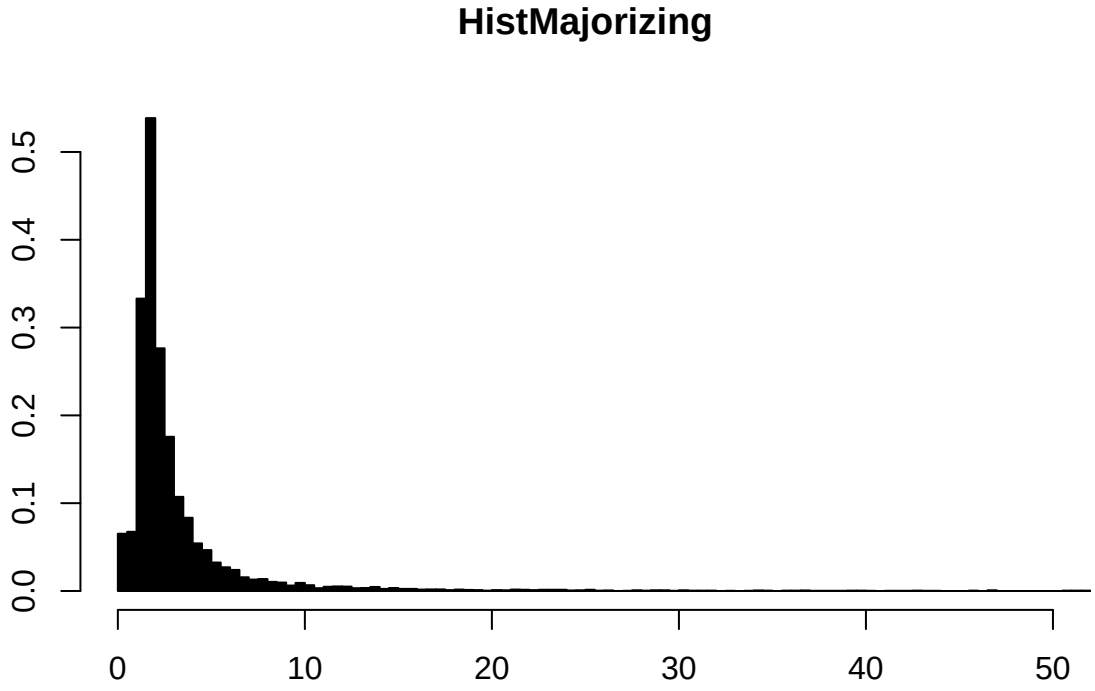
```
  }
  return(res)
}

c_maj = max(f(x,c)/as.vector(nd(x,tmin,a,p)))
```

## 2

Acceptance-rejection algorithm :

```
a =2.5
rmajorizing<-function(n){
  sapply(1:n,function(i){
    res<-NA
    component<-sample(1:2,1,prob = c(0.0833,0.9167))
    if(component==1){res<-runif(1,0,tmin)}
    if(component==2){res<-rplcon(1,tmin,a)}
    res
  })
}
Nsample<-10000
num_histbreaks<-100
hist(rmajorizing(Nsample),breaks=2000,col="black",xlab="",ylab="",main="HistMajorizing",freq=FALSE, xli
```



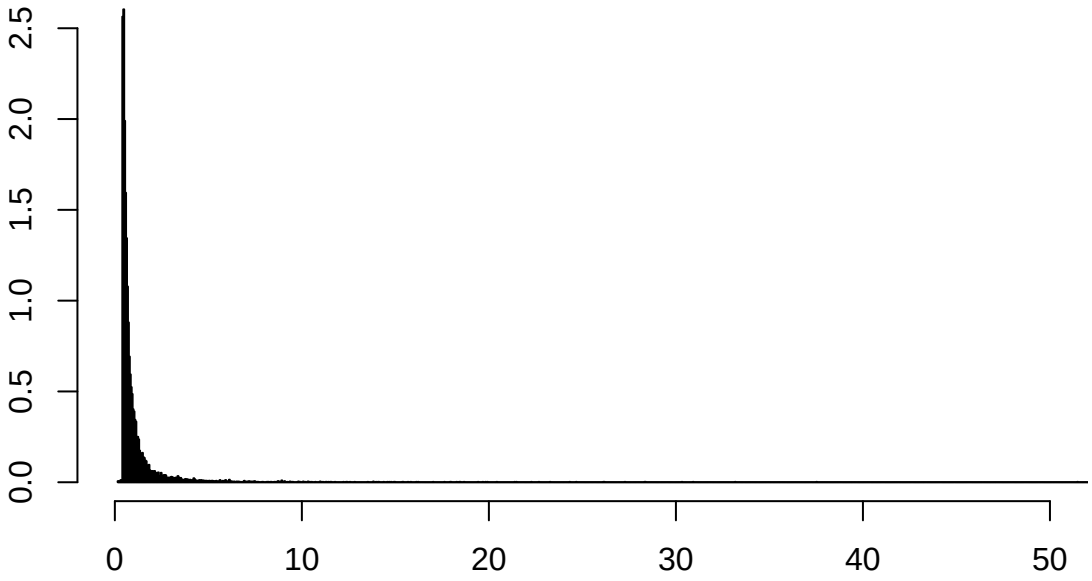
```
accept_reject<-function(c,c_maj,t){
  x<-NA
  num_reject<-0
  while (is.na(x)){
    y<-rmajorizing(1)
    u<-runif(1)
    if (u<=f(y,c)/(c_maj*nd(y,a,tmin = t,p))){
      x<-y
    } else{
      num_reject<-num_reject+1
    }
  }
 res = c(x,num_reject)
 return(res)
}
```

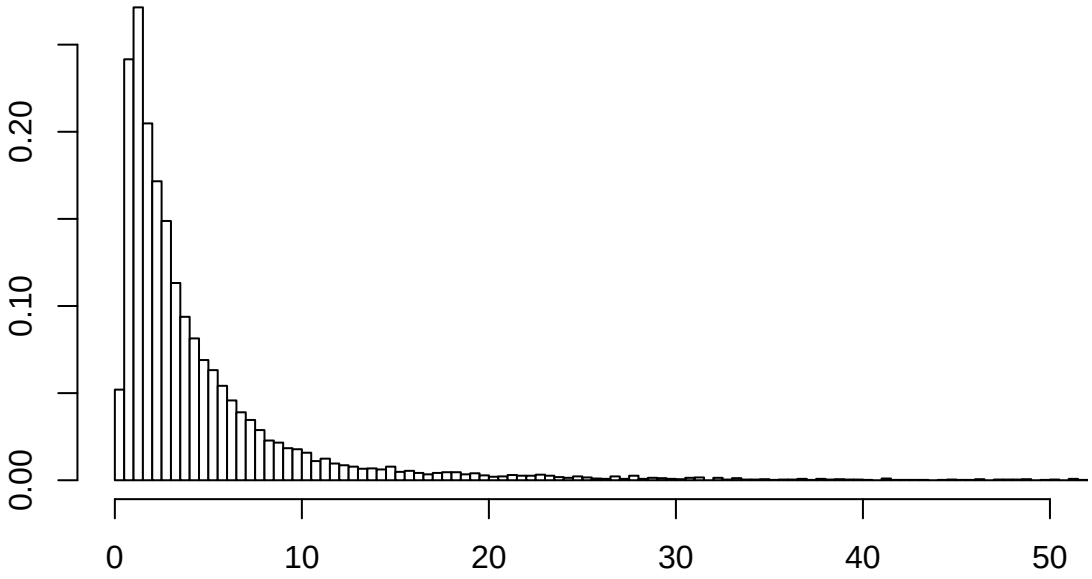
## 3

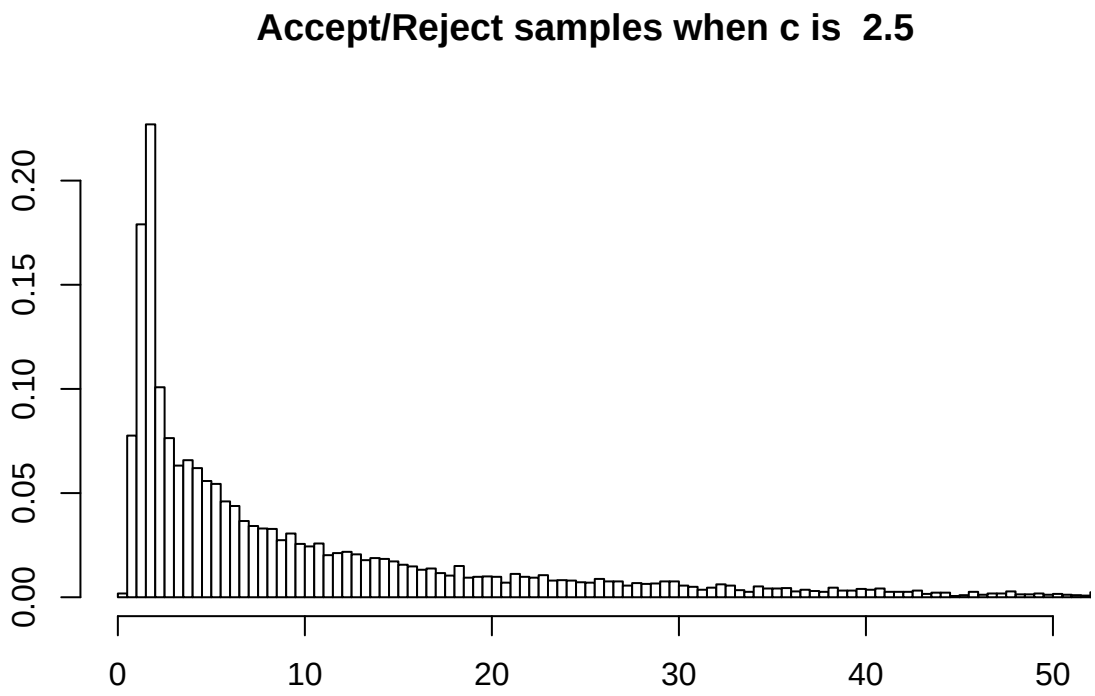
Generating large samples from above sampler to different values of c. When we compare the results of f(x) for different

```
set.seed(12345)
c = c(1.1,1.5,2.5)
p = 0.0833
t = c(0.4033333,0.75,2.0833) # tmin chosen as c^2 / 3
cm = c(max(f(x,1.1)/nd(x,a,t[1],p)),max(f(x,1.5)/nd(x,a,t[2],p)),max(f(x,2.5)/nd(x,a,t[3],p)))
mean=var=rejection_rate=c()
for(i in 1:length(c)){
tmin = t[i]
fx_acceptreject<-sapply(rep(c[i],Nsample),accept_reject,c_maj = cm[i], t = tmin)

hist(fx_acceptreject[1,],col="white",border = "black",breaks=10000,xlab="",ylab="",freq=FALSE,main="",x
title(capture.output(cat("Accept/Reject samples when c is ", c[i])))
mean =cbind(mean, mean(fx_acceptreject[,1]))
var = cbind(var,var(fx_acceptreject[,1]))
rejection_rate = cbind(rejection_rate,sum(fx_acceptreject[,2])/Nsample)
}
```


Accept/Reject samples when c is 1.1
0.0
0.5
1.0
1.5
2.0
2.5
0
10
20
30
40
50


Accept/Reject samples when c is 1.5
0.00
0.10
0.20
0
10
20
30
40
50




Mean for c = 1.1 is 0.8100103

Mean for c = 1.5 is 5.6142188

Mean for c = 2.5 is 6.0692443

Variance for c = 1.1 is 1.3122333

Variance for c = 1.5 is 11.3839036

Variance for c = 2.5 is 48.6247028

Rejection rate for c = 1.1 is $6.0560238 \times 10^{-5}$

Rejection rate for c = 1.5 is $3.4814113 \times 10^{-4}$

Rejection rate for c = 2.5 is 0.0018567

We can see that as value of c increases, the mean and variance also increases. This is because as c increases , target function peak moves further towards right on x as max of target function is achieved when x is cˆ2 / 3. As c increases , the max of target function decreases, hence reducing the sharp peaks and making it smoother. This inturn increases the values being sampled by accept reject algorithm, thus increasing mean and variance. Rejection rate also increases as the value of c increases.The proposal density goes way higher than our target density. This results in increase in number of rejections.

## Question 2: Laplace distribution

```
library(VGAM)
set.seed(12345)
```

The Laplace distribution is the distribution of the difference of two independent random variables with identical exponential distributions.

$$DE(\mu, \alpha) = \frac{\alpha}{2} \exp^{(-\alpha|x-\mu|)}$$

By integrating f(x), CDF is given as :

$$F(x) = \int_{-\infty}^{x} f(x)dx$$

$$F(x) = \int_{-\infty}^{x} \frac{\alpha}{2} e^{-\alpha(x-\mu)}dx$$

$if(x > \mu)$

$$F(x) = 1 - \int_{x}^{\infty} \frac{\alpha}{2} e^{-\alpha(x-\mu)}dx$$

$$F(x) = 1 - \frac{1}{2} e^{-\alpha(x-\mu)}$$

$$F(x) = \int_{-\infty}^{x} \frac{\alpha}{2} e^{-\alpha(x-\mu)}dx$$

$if(x <= \mu)$

$$F(x) = \frac{1}{2} e^{\alpha(x-\mu)}$$

Inverse of CDF

For $if(x > \mu)$ we got

$$F(x) = \int_{-\infty}^{x} \frac{\alpha}{2} e^{-\alpha(x-\mu)}dx$$

$$y = 1 - \frac{1}{2} e^{-\alpha(x-\mu)}$$

$$\frac{\ln(2-2y) - \alpha\mu}{-\alpha} = x$$

So for $DE(0, 1)$,

$$\frac{\ln(2-2y) - 0}{-1} = x$$

For $U \sim U(0, 1)$,

$$-\ln(2-2U) = X$$

$$-\ln 2(1-U) = X$$

$$x \geqslant 0 => \ln 2(1-U) \leqslant 0 => 0 < 2(1-U) \leqslant 1 => 0 < (1-U) \leqslant \frac{1}{2} => -1 < -U \leqslant -\frac{1}{2} => \frac{1}{2} \leqslant U < 1$$

For $if(x <= \mu)$ we got

$$F(x) = \frac{1}{2} e^{\alpha(x-\mu)}$$

$$y = \frac{1}{2} e^{\alpha(x-\mu)}$$

$$\frac{\ln(2y)}{\alpha} + \mu = x$$

So for $DE(0, 1)$,

$$\frac{\ln(2y)}{1} + 0 = x$$

For $U \sim U(0,1)$,

$$\ln(2U) = X$$

$$x < 0 => 0 < 2U < 1 => 0 < U < \frac{1}{2}$$

Hence inverse CDF equations are

$$F^{-1}(U) = ln(2U)$$

for$0 < U < \frac{1}{2}$

$$F^{-1}(U) = -ln2(1-U)$$

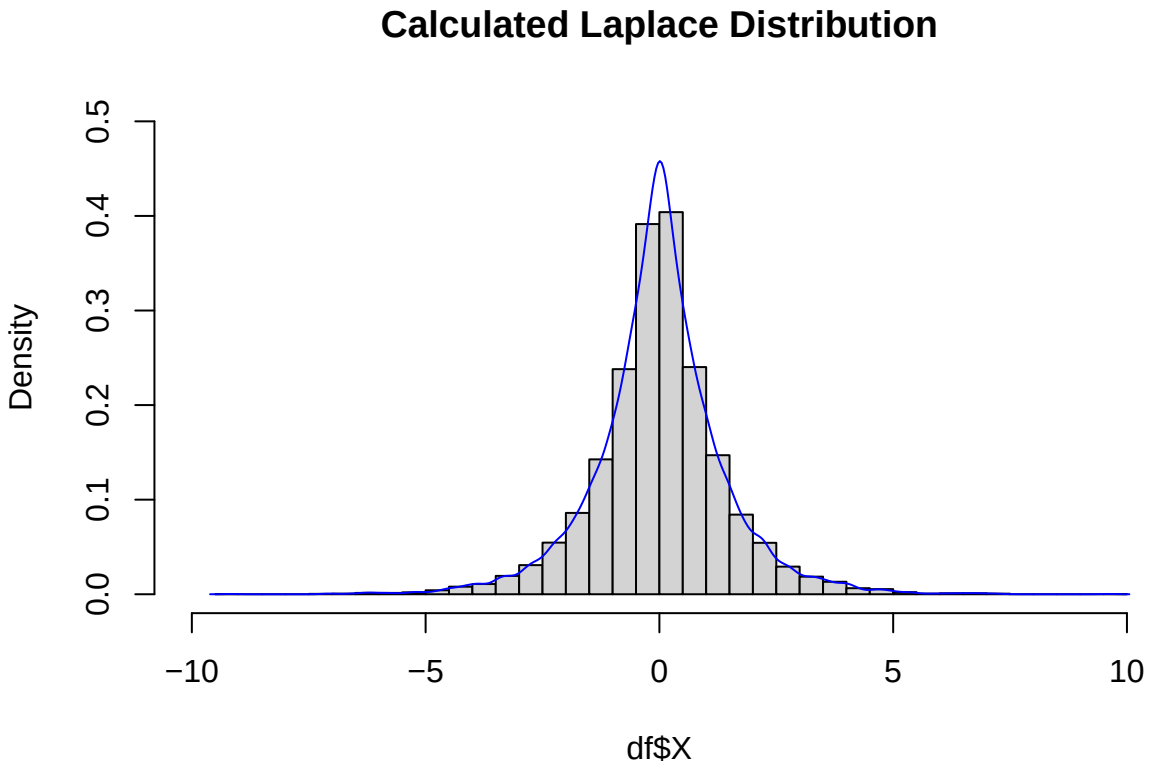
for$\frac{1}{2} \leqslant U < 1$

Generate 10000 random numbers from distribution

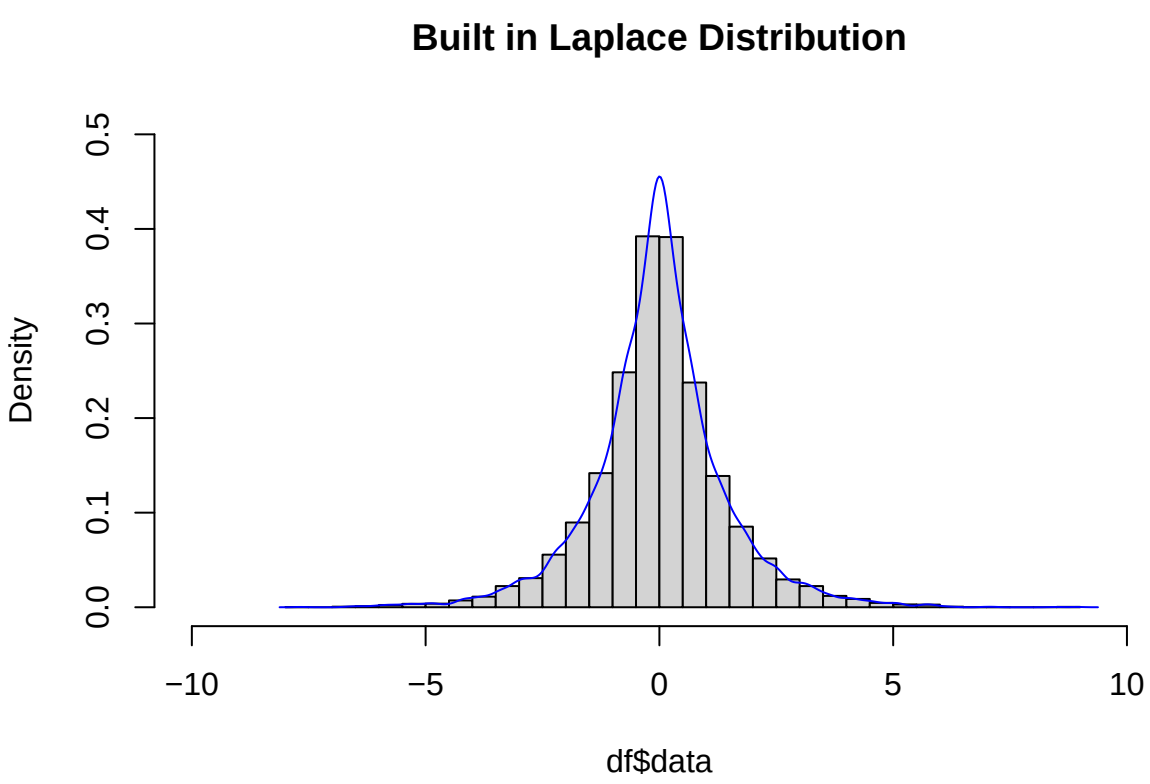
```
#U <- runif(1000,0,1)
inverse_cdf <- function(n){
  U = runif(n,0,1)                          # generate random numbers from the known distribution
  X = c()
  for(i in 1:length(U)){
    if(U[i] < 0.5){
      X[i] = log(2 * U[i])
    }
    else{
      X[i] = -log(2 * (1 - U[i]))
    }
  }
  return(X)
}
X = inverse_cdf(10000)
data = rlaplace(10000,location = 0,scale = 1)
df = data.frame(X,data)
```

Plot the generated numbers

```
plot1 = hist(df$X, prob = TRUE, ylim = c(0,0.5), xlim = c(-10,10),
breaks = 30, main = "Calculated Laplace Distribution")
lines(density(df$X), col = "blue")
```




```
plot2 = hist(df$data, prob = TRUE, ylim = c(0,0.5), xlim = c(-10,10),
breaks = 30, main = "Built in Laplace Distribution")
lines(density(df$data), col = "blue")
```




The result looks reasonable. The two plots are similar. Due to mean = 0 all the points are close to 0 as shown in the plot. Conclusion, the results obtained using CDF function is close with the result using the rlaplace function.

#2.2 Use the Acceptance/rejection method with DE(0, 1) as a majorizing density to generate N (0, 1) variables.

$$f_X(x) \sim N(0,1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

$$f_Y(x) \sim DE(0,1) = \frac{1}{2} e^{-|x|}$$

If we maximize the f_X(x)/f_Y(x) then it would be the value of c. This can be done with partial derivative of the fraction.

$$\frac{f_X(x)}{f_Y(x)} \leq c$$

$$C \geq \frac{2}{\sqrt{2\pi}} e^{|x| - \frac{x^2}{2}}$$

$$g(x) = -\frac{x^2}{2} + x$$

$$g'(x) = 1 - x$$

$$1 - x = 0 => x = 1$$

$$c = \frac{2}{\sqrt{2\pi}} e^{1 - \frac{1^2}{2}}$$

$$= \frac{2}{\sqrt{2\pi}} \sqrt{e}$$

By calculation, constant c = 1.3154

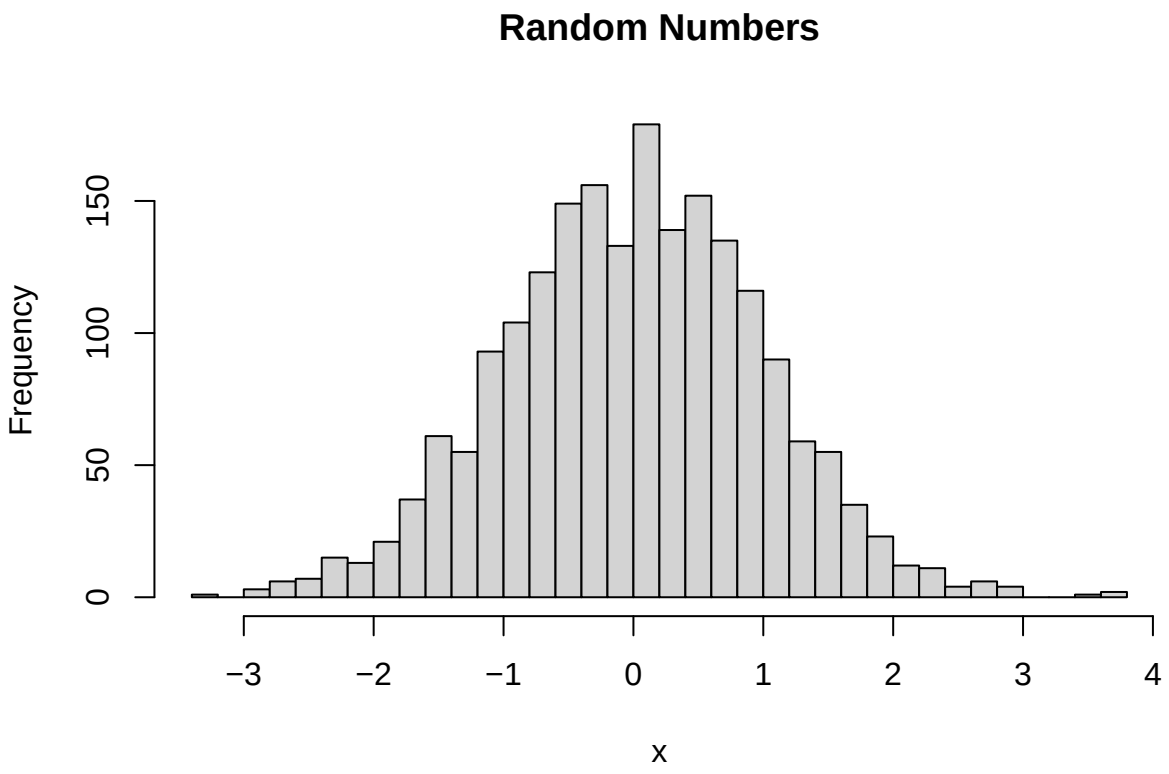
Generation of 2000 random numbers from N(0,1)

```
c = sqrt((2*exp(1))/pi)

DE <- function(x){
  exp(-abs(x)) / 2
}
accept_reject <- function(n){
  R <- 0
  Y <- vector(length = n)
  for(i in 1:n){
    repeat {
      y <- inverse_cdf(1)
      U <- runif(1)
      fy <- dnorm(y, mean = 0, sd = 1)
      gy <- DE(y)
      if(U <= fy / (c * gy)){
        Y[i] <- y
        break
        }
      else{R = R+1}
    }
  }
  return(list(Y=Y, Reject=R))
}
set.seed(12345)
Z <- accept_reject(n = 2000)
```

Plot the histogram

```
hist(Z$Y, xlab = "x",main="Random Numbers", breaks = 50)
```




```
cat("Expected Rejection Rate: ER = ", 1 - (1/c))
```

```
## Expected Rejection Rate: ER =  0.2398265
```

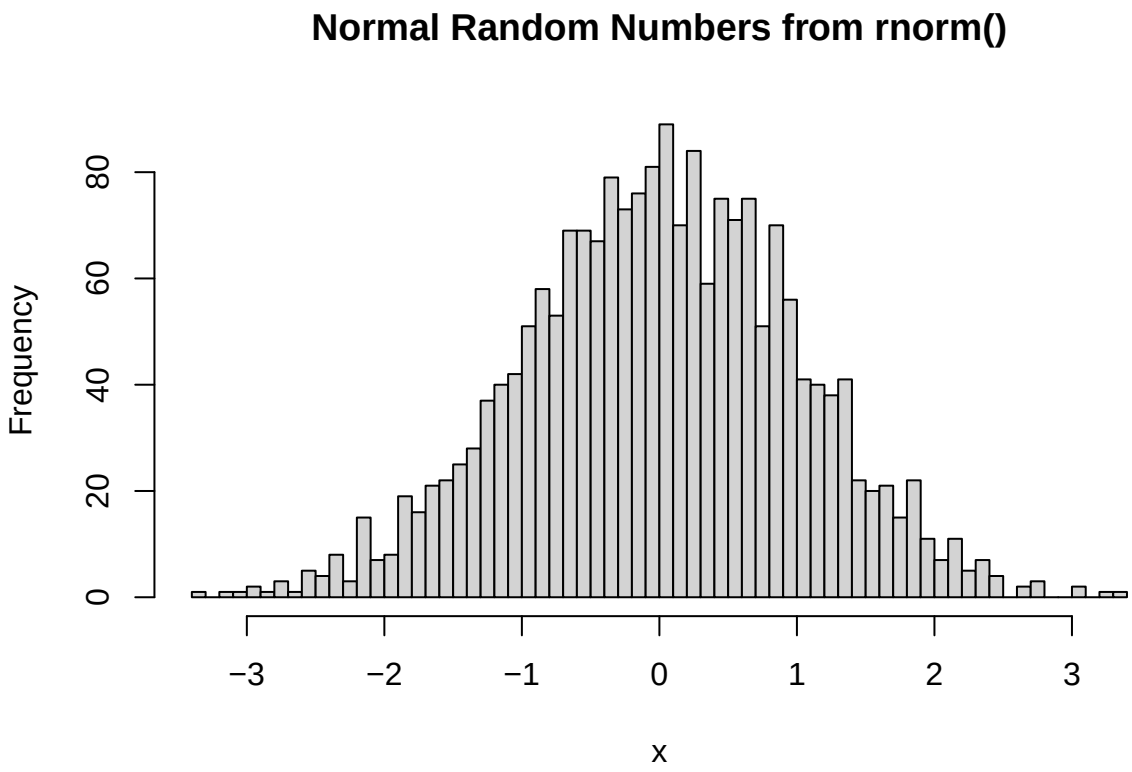
```
cat("Rejection Rate: R = ", sum(Z$Reject)/(2000 +sum(Z$Reject)))
```

```
## Rejection Rate: R =  0.2392545
```

The expected rejection rate is 0.2398265 and the average rejection rate generated in accept_reject function() is 0.2392545. The difference is only 0.000572. Hence expected rejection rate is very close to R. This means that our function for the random sample has satisfying results.

Generate 2000 numbers from N (0, 1) using standard rnorm() procedure and plot the histogram

```
set.seed(12345)
hist(rnorm(2000,0,1), main = "Normal Random Numbers from rnorm()", xlab = "x", breaks = 50 )
```




Comparing 2 histograms we can conclude that the 2 results looks similar hence the function is efficient.