Assignment2

Masroor Hossain(45765758)

5/4/2021

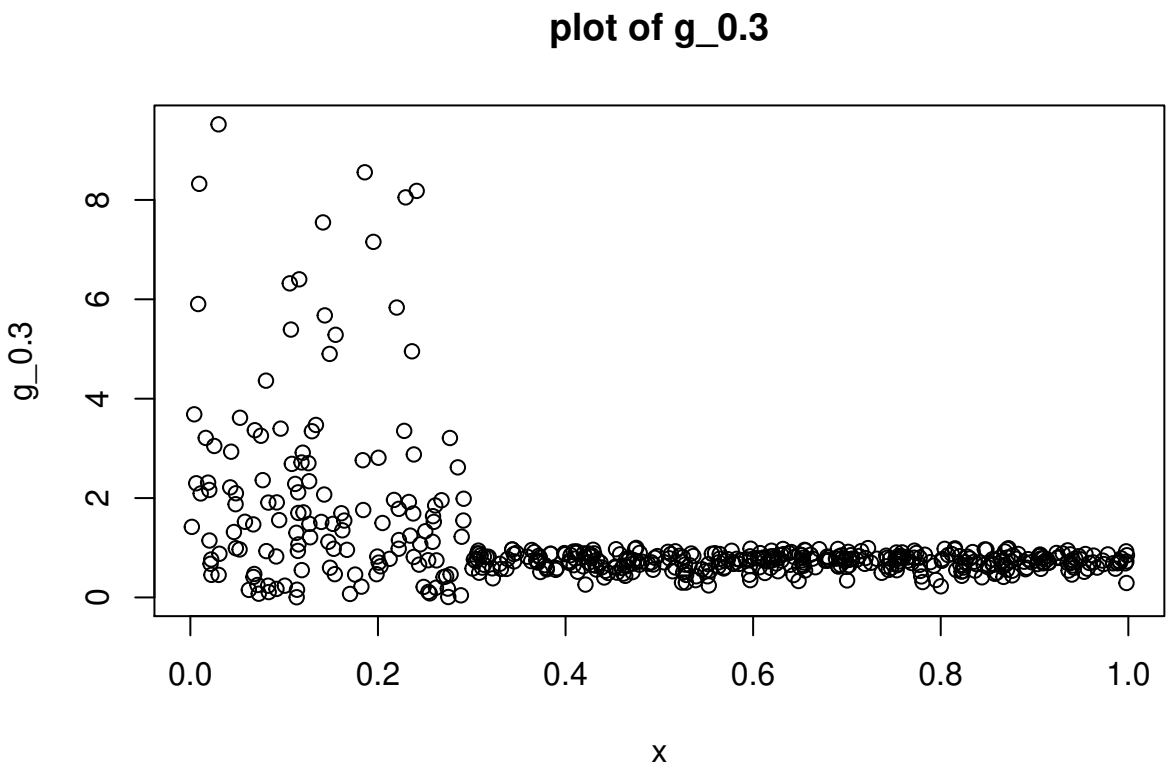
1 a)

Since it is a mixture of exponential and beta distribution, I used this following approach to solve the problem taking idea from this link(https://stats.stackexchange.com/questions/70855/generating-random-variables-from-a-mixture-of-normal-distributions/435130)

```
N = 500

#Sample N random uniforms U                                    1.1
U =runif(N)

#Variable to store the samples from the mixture distribution
g_0.3 = rep(NA,N)

#Sampling from the mixture
for(i in 1:N){
    if(U[i]<=.3){
        g_0.3[i] = rexp(1,rate=0.5)
    }else{
        g_0.3[i] = rbeta(1,5,2)
    }
}
```

This is the code for the plot of g_0.3.

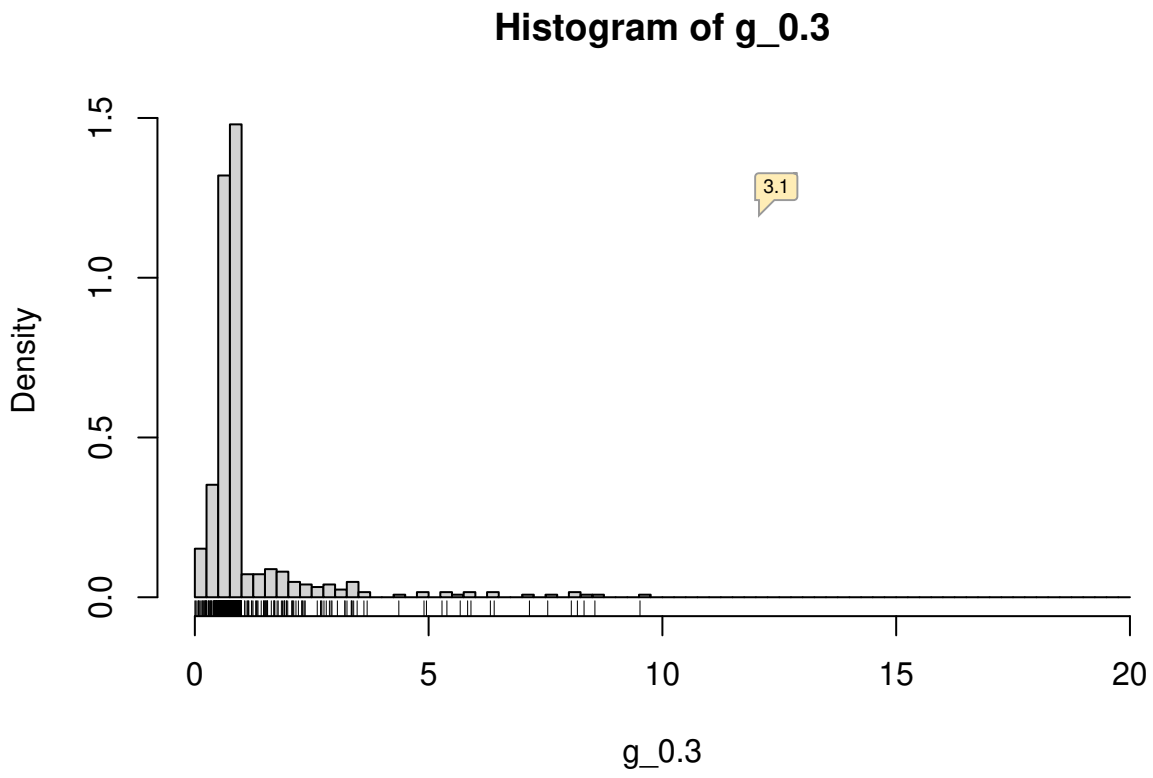
```
plot(U, g_0.3,
main="plot of g_0.3",xlab="x",
ylab="g_0.3")
```



1 b)

Plotting the histogram estimator, I get

```
h <- 0.25
x <- seq(0, 20, by = h)
hist(g_0.3, probability = TRUE,breaks=x)
rug(g_0.3) # The sample
```



1 c)

```
#install.packages("dvmisc")
library(dvmisc)
```

```
## Loading required package: rbenchmark
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(dplyr)
#install.packages("tidyverse")
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v stringr 1.4.0
## v tidyr   1.1.3     v forcats 0.5.1
## v readr   1.4.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::expand_grid() masks dvmisc::expand_grid()
## x dplyr::filter()      masks stats::filter()
## x dplyr::lag()         masks stats::lag()
```

As there are 500 random observations, I initialized n as 500. The value of M is used in the main for loop of Monte carlo estimation. So I initialized M as 1000. As I used the same histogram estimator shown in 1 b, I used the same variable i.e. g_0.3. The variable bt is the bin counts of a histogram

```
n <- 500
a=-4
b=4
M <- 1000

#sigxbar <- sd/sqrt(n)#
h<-hist(g_0.3,plot=FALSE)
bt<-h$counts
```

4.1

In the following code, I plotted MSE against the values of x.

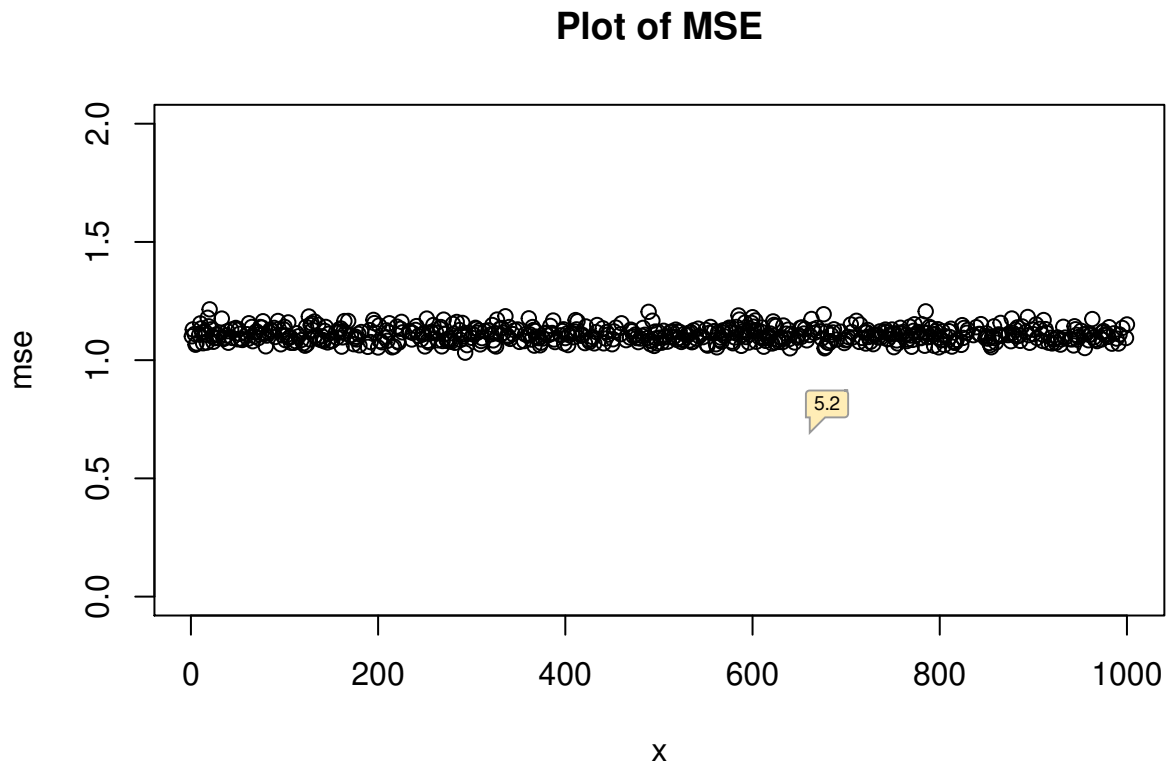
```
True <- 0.3*dexp(bt/n,rate=0.5)+0.7*dbeta(bt/n,5,2)

d=array(dim=M)
for(i in 1:M){
# Decide on a pseudo-population that reflects the characteristics
# of the true population under the null hypothesis.
# Obtain a random sample of size n from the pseudo-population.

#h1=bw.nrd(x = x)
data=sample(g_0.3,size=n,replace=TRUE)

ho=hist(data,plot=FALSE)
bkct=ho$counts
fhat=bkct/n
tmp <-0

for(k in 1:length(True)){
  tmp <- tmp+(fhat[k]-True[k])^2


}
#m2 <- dbeta(x,5,2)
#m <- 0.3*m1+0.7*m2
#l <- hist(m)

d[i]=sum(tmp)
```

4.2

4.3

```r
# Calculate the value of the test statistic using the random
# sample in step above and record it
#crit.val.result[i]=t.test(x, mean)$statistic
}

#curve((h-l)^2,a,b,lwd=2,col=3)

plot(d,xlab="x",ylab="mse",ylim=c(0, 2),main="Plot of MSE")
```

5.1




```r
#cv1 = quantile(crit.val.result,alpha/2)
#cv2 = quantile(crit.val.result,1-alpha/2)

#n <- get_mse(l)
#n
#cv1

#cv2
```

1 d)

```r
a=-4
b=4
M <- 1000
alpha<-0.05
```

```r
lambda <- 0.05
#sigxbar <- sd/sqrt(n)#
histt<-hist(g_0.3,plot=FALSE)
bt1<-histt$counts
```

```r
#install.packages("sfsmisc")
library(sfsmisc)
```

```
##
## Attaching package: 'sfsmisc'

## The following object is masked from 'package:dplyr':
##
##     last
```

```r
#install.packages("splines2")
#library(splines2)
#install.packages("stats")
library(stats)
```

```r
histestanother <- density(g_0.3)
M <- 1000
mise50=array(dim=M)
mise100=array(dim=M)
mise250=array(dim=M)
mise500=array(dim=M)
for(i in 1:M){
 data2=sample(g_0.3,size=50,replace=TRUE)

 honew=density(data2)
 spxynew <- splinefun(honew$x,(honew$y-histestanother$y)^2)
 mise50[i]=integrate(spxynew,lower=min(data2),upper=max(data2))$value

}
for(i in 1:M){
 data3=sample(g_0.3,size=100,replace=TRUE)

 honew1=density(data3)
 spxynew1 <- splinefun(honew1$x,(honew1$y-histestanother$y)^2)
 mise100[i]=integrate(spxynew1,lower=min(data3),upper=max(data3))$value

}
for(i in 1:M){
 data4=sample(g_0.3,size=250,replace=TRUE)

 honew2=density(data3)
 spxynew2 <- splinefun(honew2$x,(honew2$y-histestanother$y)^2)
 mise250[i]=integrate(spxynew2,lower=min(data4),upper=max(data4))$value

}
for(i in 1:M){
 data5=sample(g_0.3,size=500,replace=TRUE)
```

6.1

```
 honew3=density(data5)
 spxynew3 <- splinefun(honew3$x,(honew3$y-histestanother$y)^2)
 mise500[i]=integrate(spxynew3,lower=min(data5),upper=max(data5))$value

}

print(mean(mise50))
```

7.1

```
## [1] 0.6877481
```

```
print(mean(mise100))
```

7.2

```
## [1] 0.4972386
```

```
print(mean(mise250))
```

```
## [1] 0.01746503
```

7.3

```
print(mean(mise500))
```

```
## [1] 0.08473004
```

7.4

The MISE for sample size 50 is 0.82 which is greater than MISE for sample size 100. On the other hand, the MISE for sample size 250 and 500 are small values which are -18.278 and 0.09 respectively.

7.5

2 a)

It is known,

H0: $\mu = 600$

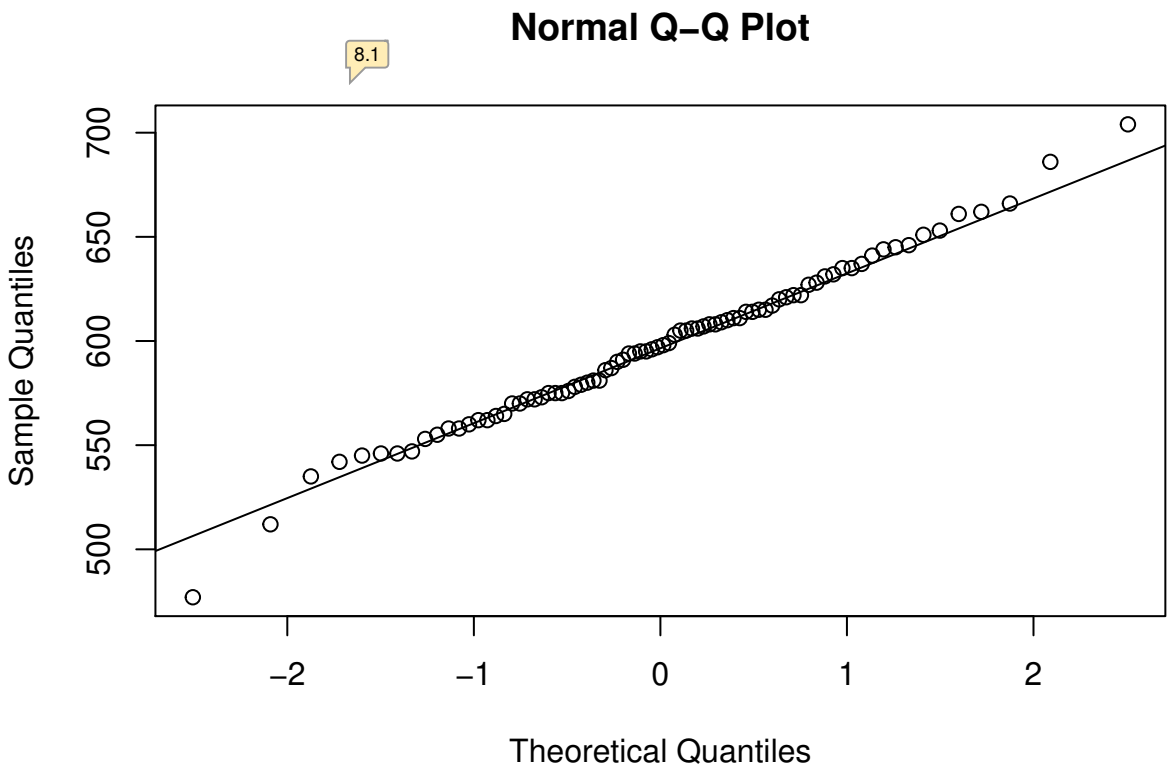
H1: $\mu < 600$

This is a case of left tailed one sample t-test.

```
options(scipen=999)
lsat <- read.csv("C:/Users/Dell/Downloads/lowlsat.csv",header=TRUE)
head(lsat)
```

```
##   lsat
## 1  622
## 2  542
## 3  579
## 4  653
## 5  606
## 6  576
```

```
qqnorm(lsat$lsat)
qqline(lsat$lsat)
```



It can be seen that the normal Q-Q plot is a straight line showing that it is normally distributed.

```
n <- nrow(lsat)
n
```

```
## [1] 82
```

```
x <- mean(lsat$lsat)
x
```

```
## [1] 597.5488
```

```
sigma <- 40
sigma
```

```
## [1] 40
```

```
sigxbar <- sigma/sqrt(n)#Population sigma is known.
alpha<-0.05
mu <- 600
Tobs = (mean(x)-mu)/sigxbar
Tobs
```

8.2

```
## [1] -0.5549184
```

```
# Apply Monte Carlo Simulation
M=1000

# Critical value Result
T.Statistic=array(dim=M)

for(i in 1:M){

# Decide on a pseudo-population that reflects the characteristics
# of the true population under the null hypothesis.

# Obtain a random sample of size n from the pseudo-population.
x=rnorm(n,mu,sigma)

# Calculate the value of the test statistic using the random
# sample in step above and record it
T.Statistic[i]=(mean(x)-mu)/sigxbar

}
#T.Statistic
# critiacal values
#cv1 = quantile(T.Statistic,alpha/2)
#cv2 = quantile(T.Statistic,1-alpha/2)
#cv1
#cv2
cv1 = quantile(T.Statistic,alpha)
cv2 = quantile(T.Statistic,1-alpha)
cv1
```

```
##        5%
## -1.740759
```

```
cv2
```

```
##      95%
## 1.660995
```

```
# Test Statistic
test.statistic <- Tobs
test.statistic
```

```
## [1] -0.5549184
```

#Result: Estimated critical values are -1.674506 and 1.685063. Since the observed value of the test statistic -0.5549184 is greater than the lower critical value we do not reject H0.

Pvalue is calculated

```
ind1 <- which(T.Statistic<=Tobs)

pvalhat = length(ind1)/M
pvalhat
```

```
## [1] 0.304
```

Since the p-value is greater t 0.05, we reject H1.

3.

First, we load the data

```
library(rmatio)
library(dplyr)
```

```
#install.packages("moments")
library(moments)
#install.packages("boot")
library(boot)
#install.packages("purr")
library(purrr)
```

```
quake = read.mat("C:/Users/Dell/Downloads/quakes.mat")
head(quake)
```

```
## $quakes
##  [1]  840  157  145   44   33  121  150  280  434  736  584  887  263 1901  695
## [16]  294  562  721   76  710   46  402  194  759  319  460   40 1336  335 1354
## [31]  454   36  667   40  556   99  304  375  567  139  780  203  436   30  384
## [46]  129    9  209  599   83  832  328  246 1617  638  937  735   38  365   92
## [61]   82  220
```

```
x <- quake$quakes
n <- length(x)
#number of bootstrap replicates
B <- 100
thetahat = skewness(x)
# Generate boot samples
boot.samples = matrix(sample(x, size = B * n, replace = TRUE),B,n)
```

```
thetab = apply(boot.samples, 1, skewness)
seb=sd(thetab)
seb
```

```
## [1] 0.3839222
```

```
# Evaluate the mean.
meanb = mean(thetab)
meanb
```

```
## [1] 1.426162
```

```
# Now estimate the bias.
biasb = meanb - thetahat
biasb
```

```
## [1] -0.07299058
```

```
# Now the bias-corrected estimate:
bctheta = thetahat - biasb # =2*theta-meanb
bctheta
```

```
## [1] 1.572143
```

Bootstrap standard confidence interval

```
alpha = 0.05
Lower = thetahat - qnorm(1- alpha/2)*seb
Upper = thetahat - qnorm(alpha/2)*seb
c(Lower, Upper)
```

11.1

```
## [1] 0.7466786 2.2516261
```

The bootstrap-t interval

```
#estimated standard error of skewness of that particular bootstrap sample.
shat=array(dim=B)
zstarb=array(dim=B)
for(i in 1:B) {
boot.samples2 = matrix(sample(boot.samples[i,], size = B * n, replace = TRUE),B,n)
thetab = apply(boot.samples2, 1, skewness)
shat[i]=sd(thetab)
}
# Compute Z*b=(thetahatb-thetahat)/shat
zstarb=array(dim=B)
for(i in 1:B)
{ zstarb[i]=(thetab[i]-thetahat)/(shat[i])}
#Get the quantiles.
k = (B*alpha)/2
szval = sort(zstarb);
tlo = szval[k]
thi = szval[B-k]
# Get the endpoints of the interval.
blo = thetahat - thi*seb
bhi = thetahat - tlo*seb
c(blo, bhi)
```

11.2

11.3

```
## [1] 0.9972584 2.7109895
```

Bootstrap percentile interval

```
quantile(thetab, probs = c(alpha/2, 1- alpha/2))
```

```
##      2.5%     97.5%
## 0.7423792 1.8992442
```

The bootstrap standard confidence interval is between 0.884895 and 2.113410 which is much wider than the bootstrap-t interval that covers between 0.8882429 and 2.0138840 and the bootstrap percentile interval that covers between 0.9979889 and 2.0730595.

Index of comments

Index of comments