Homework 8. Due Tuesday, November 2nd. All of these questions have computational solutions. Your solution must be in a '.R' or other similar file.

Preamble

In this HW you will think about uncertainty in model fitting and estimation in two ways.

First, you will repeatedly sample from a population and use this to assess the uncertaintly inherent in estimation that comes from the random sampling. (1) Each sample you will produce point estimates of the parameters of a model for the population. (2) Produce plots of the density and distribution functions for the sample estimates.

Second, you will assess the uncertainty in point estimates using the bootstrap. You cannot actually do the first set of uncertainty quantifications in practice because you do not get to repeatedly sample from a population in the real world (you get one sample). The bootstrap creates replicates from the observed sample by sampling from the observved sample with replacement a sample of the same size as the observed sample. (1) Each bootstrap replicate will give you a point estimate. (2) You will "bias correct" the bootstrap point estimates to make an approximation of the density and the distribution of the estimator from repeatedly sampling from the population. Basically, you will do the same operations you did on point estimates from repeatedly sampling the population, but instead you will do the operations on the point estimates from the bootstrap replicates.

Instead of considering real data in this assignment, we will work completely with randomly generated data. Our populations will not be finite populations that we sample from using the `sample` function. Instead, they will be random number generation functions for specific distributions. This means we are considering infinite populations in this HW (mostly for ease of computational burden). At the top of your solution file, include the line

```
set.seed(0123456789)
```

the integer `0123456789` in `set.seed` with your 10-digit Student ID to set the seed of `R`'s random number generator.

Example Statistic Functions

In order to use the `boot` library and efficiently do the bootstrap, we need to write functions. The statistic functions that are used by the `boot` function have one small modification from standard statistic functions. They take in the data (which I will call `x`) as their first argument (which standard statistic functions do). The modification is that the functions that need to be used by `boot` need a second argument is an vector of indices. The function needs to access the data using the vector of indices to make a local version of the data(which I will call `x_loc`) and do a calculation. Setting the default value for the index makes these functions easy to also use when doing estimation outside of the context of using the `boot` package. Here are four example statistic functions

```
mean_sample_stat_fun = function(x,index=c(1:length(x))){
  # x is a data vector
  # index is an integer vector of indices to access
  # default is c(1,2,...,length(x)) and for it x_loc=x
```

```
  x_loc = x[index] # access elements corresponding to index
  return(mean(x_loc)) # return statistic for local x
}

sd_sample_stat_fun = function(x,index=c(1:length(x))){
  # x is a data vector
  # index is an integer vector of indices to access
  # default is c(1,2,...,length(x)) and for it x_loc=x
  x_loc = x[index] # access elements corresponding to index
  return(sd(x_loc)) # return statistic for local x
}

mean_and_sd_sample_stat_fun = function(x,index=c(1:length(x))){
  # x is a data vector
  # index is an integer vector of indices to access
  # default is c(1,2,...,length(x)) and for it x_loc=x
  x_loc = x[index] # access elements corresponding to index
  return(c(mean=mean(x_loc), sd=sd(x_loc))) # return statistic for local x
}

median_and_iqr_sample_stat_fun = function(x,index=c(1:length(x))){
  # x is a data vector
  # index is an integer vector of indices to access
  # default is c(1,2,...,length(x)) and for it x_loc=x
  x_loc = x[index] # access elements corresponding to index
  return(c(median=median(x_loc), iqr=IQR(x_loc))) # return statistic for local x
}
```

Notice that the second two functions return not scalars, but vectors with named arguments. In the return line, the `c()` is constructing the vector, the string before the `=` is the name of the element, and the value after the `=` is the value assigned to the named element. For these functions, we are outputting more than one statistic (number computed from the data vector) and returning them both at once. In these instances, we wrote functions that merely wrapped the use of available functions. In other instances, we might have to do more intricate calculations.

Examples using matrices and `apply`

When using any random number generation function and then statistic function, it is always faster if we can avoid looping as much as possible. Consider the situation where we are several generating data vectors (say `N_samples` of them) of the same length (say `n`) from a random number generator with fixed parameters. We can generate all of the samples at one time as one big vector (of length `n*N_samples`). We can in the same line organize the draws into a matrix (with dimensions `n` by `N_samples`), making each column of the matrix one data vector. Here is an example of sampling many normal data vectors using a for loop and then sampling using a single line.

```
# set parameters of the simulation
N_samples = 10000 # number of samples
n = 123 # sample size

# distribution parameters
```

```
mean_dist = 4.5
sd_dist = 2.3

# first using a for loop
sample_mat_1 = matrix(NA,nrow=n,ncol=N_samples) # make a container
for(j in 1:N_samples){ # loop over columns
  # replace j-th column with random draws
  sample_mat_1[,j] = rnorm(n,mean=mean_dist,sd=sd_dist)
}

# second in a single line
sample_mat_2 = matrix(rnorm(n*N_samples,mean=mean_dist,sd=sd_dist),nrow=n,ncol=N_samples)
```

Using a single line is more compact, easier to read and interpret, and more computationally efficient.

Now, when we want to compute statistics on many data vectors that we are sampling, we have three options (well, many options, but we will discuss three). The first is generating a sample and computing its statistic inside of a for loop. The containers we make will need to store all the values that are returned by whatever statistic function we are using. The second is to generate all of the samples and organize them into a matrix and then do the for loop for computing the statistics. We still need to make containers for storing the output of the statistic function. The third is to use the `apply` function. This function applies (hence the name) a function to a margin of a matrix (more generally to margins of an array). We will use it with three inputs: the data matrix, the fixed margin (we will use `2` to do computations for each column), and the function (one of our statistic functions). Here are two examples of all three methods.

```
# example 1, scalar statistic of mean

## first method, sample and compute in for loop
## store only statistic output
sample_stat_1 = rep(NA,N_samples) # make a container
for(j in 1:N_samples){ # loop
  # generate sample and use it as input to statistic function
  sample_stat_1[j] = mean_sample_stat_fun(rnorm(n,mean_dist,sd_dist))
}

## second method, use existing data matrix
## compute statistic in loop
sample_stat_2 = rep(NA,N_samples) # make a container
for(j in 1:N_samples){ # loop
  # access sample in column and use it as input to statistic function
  sample_stat_2[j] = mean_sample_stat_fun(sample_mat_1[,j])
}

## third method, using apply
sample_stat_3 = apply(sample_mat_2,2,mean_sample_stat_fun)

# second example, vector statistics function output

## first method, sample and compute in for loop
## store only statistic output
## we need to know the length of the output of the function
```

```
L_out = 2 # length of output from statistic function
sample_stat_4 = matrix(NA,L_out,N_samples) # make a container
for(j in 1:N_samples){ # loop
  # generate sample and use it as input to statistic function
  sample_stat_4[,j] = mean_and_sd_sample_stat_fun(rnorm(n,mean_dist,sd_dist))
}

## second method, use existing data matrix
## compute statistic in loop
## we need to know the length of the output of the function
L_out = 2 # length of output from statistic function
sample_stat_5 = matrix(NA,L_out,N_samples) # make a container
for(j in 1:N_samples){ # loop
  # generate sample and use it as input to statistic function
  sample_stat_5[,j] = mean_and_sd_sample_stat_fun(sample_mat_1[,j])
}

## third method, using apply
sample_stat_6 = apply(sample_mat_2,2,mean_and_sd_sample_stat_fun)
```

So, let's just do the generation and computation without for loops.

```
# generation
sample_mat = matrix(rnorm(n*N_samples,mean=mean_dist,sd=sd_dist),nrow=n,ncol=N_samples)
# scalar statistic
sample_stat_mean = apply(sample_mat,2,mean_sample_stat_fun)
# vector statistic
sample_stat_mean_and_sd = apply(sample_mat,2,mean_and_sd_sample_stat_fun)
```

So compact and so simple. Let's look at the dimensions of each object.

```
# looking at the dimensions
dim(sample_mat)
```

```
## [1]   123 10000
```

```
length(sample_stat_mean)
```

```
## [1] 10000
```

```
dim(sample_stat_mean_and_sd)
```

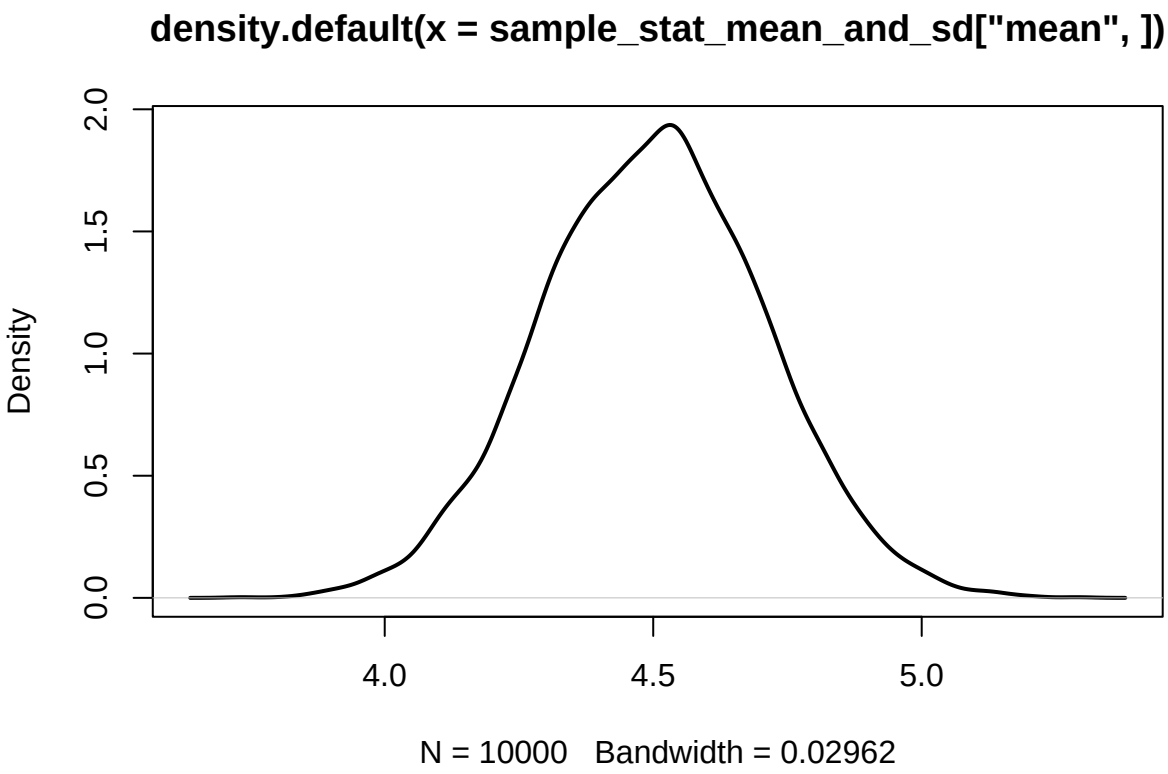
```
## [1]     2 10000
```

So the elements of the vector output from `apply` (for a scalar statistic) or the columns of the output from `apply` (for a vector statistic) correspond to the data vectors in the columns of `sample_mat`. Because we used named elements in the vector output from the stat function, we get row names in the matrix output from `apply` when using the stat fun with vector output. This is very useful for keeping track of things, we could also use these names to access the rows.

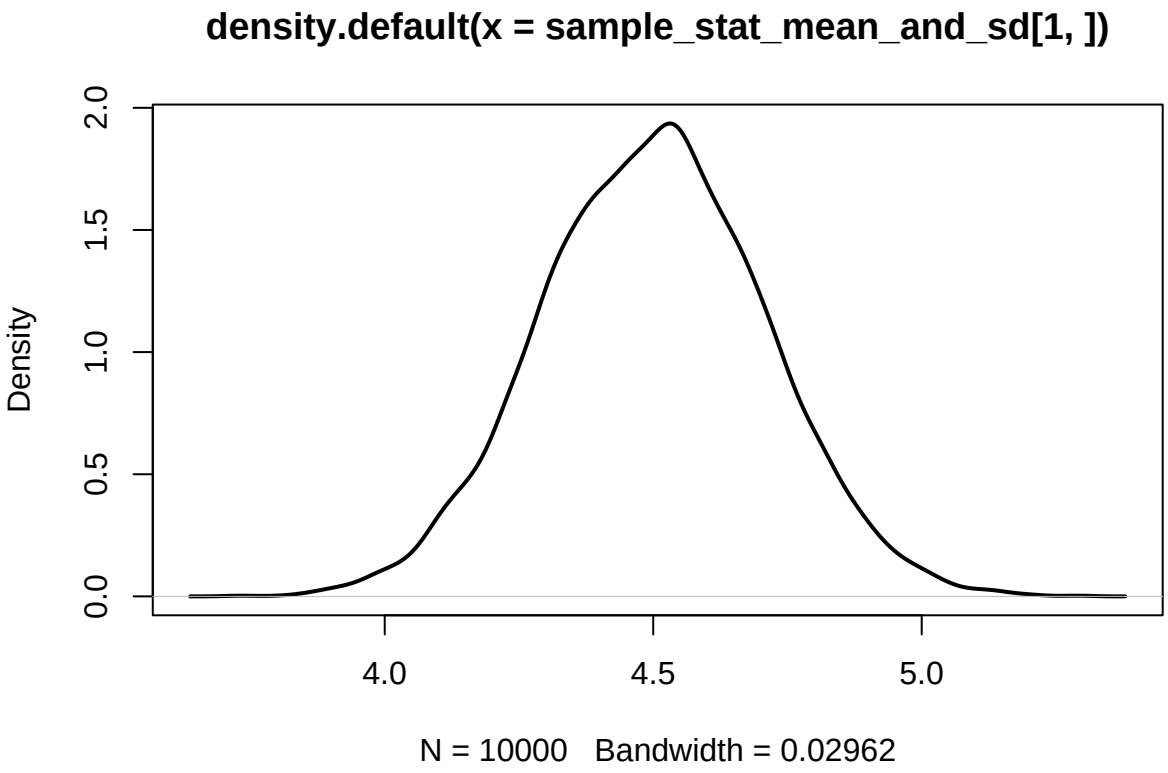
```
rownames(sample_stat_mean_and_sd)
```

```
## [1] "mean" "sd"
```

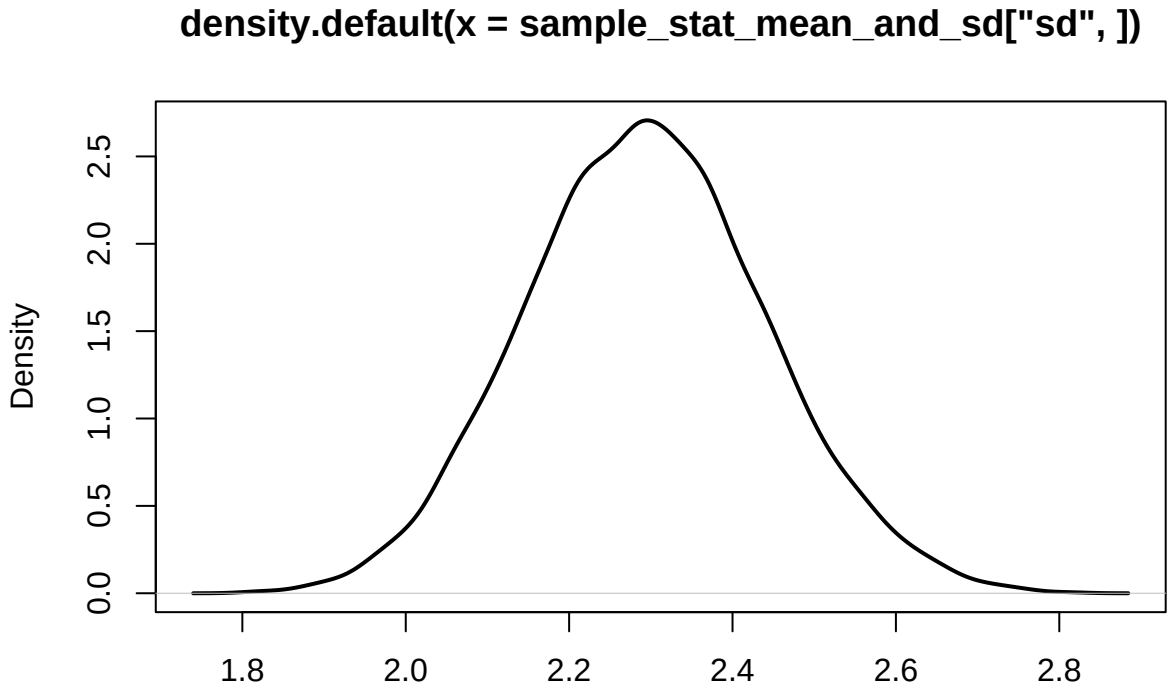
```r
plot(density(sample_stat_mean_and_sd['mean',]),lwd=2) # access by name
```



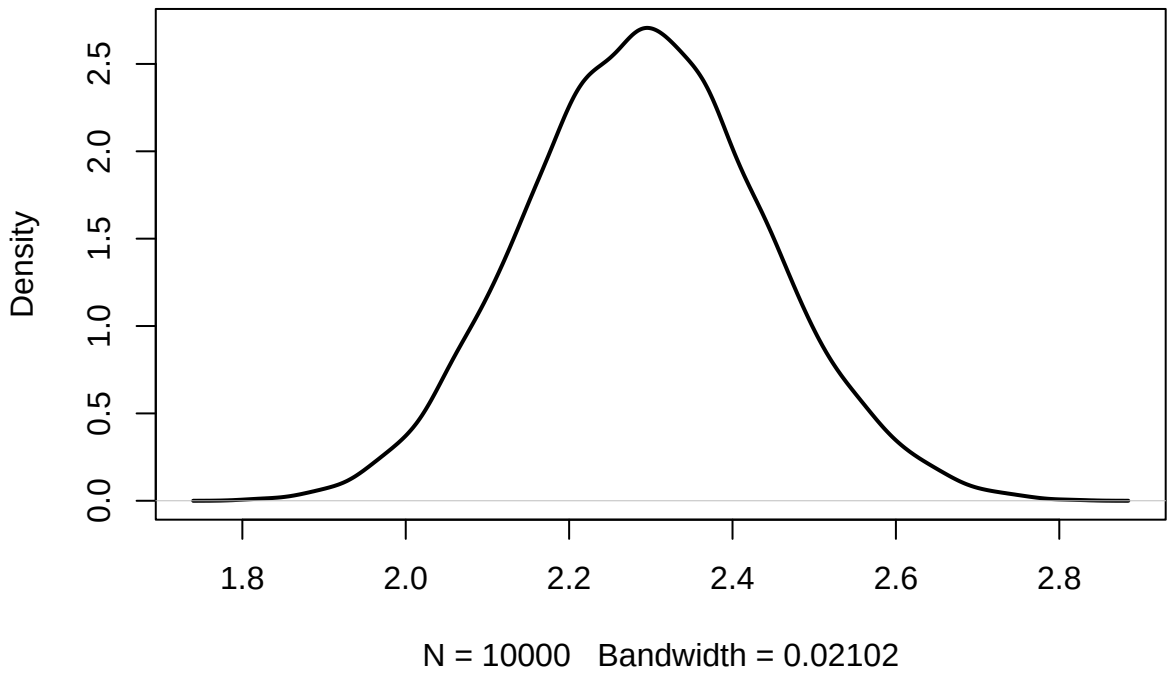

```r
plot(density(sample_stat_mean_and_sd[1,]),lwd=2) # access by row number
```




```r
plot(density(sample_stat_mean_and_sd['sd',]),lwd=2) # access by name
```




```
plot(density(sample_stat_mean_and_sd[2,]),lwd=2) # access by row number
```



The Bootstrap

The `boot` function performs the boostrap for us by taking samples from our observed, fixed sample. It takes the samples by sampling with replacement. Each of these is a called a bootstrap replicate. For each bootstrap replicate, we compute an estimate of the parameter (in the same manner as for the fixed sample). These bootstrap estimates are what we use to approximate the distribution of the estimate that would come from repeatedly sampling fixed samples from the population. Suppose that θ is a quantity that we care about, that θ_P^* is the true population parameter value of the quantity, that $\hat{\theta}_S$ is an estimate from a sample from the population, and that $\tilde{\theta}_B$ is an estimate from a bootstrap replicate from our fixed sample.

The bootstrap theory says that the random variable $\left(\hat{\theta}_S - \theta_P^*\right)$, the error of an estimate from the truth due to sampling from the population, can be approximated by the random variable $\left(\tilde{\theta}_B - \hat{\theta}_S\right)$, the error of a bootstrap estimate from the fixed population estimate. A decent notation for representing this is

$$\left(\tilde{\theta}_B - \hat{\theta}_S\right) \overset{Dist}{\approx} \left(\hat{\theta}_S - \theta_P^*\right)$$

On the left hand side, the $\tilde{\theta}_B$ is random and the $\hat{\theta}_S$ is fixed. On the right hand side, the $\hat{\theta}_S$ and the θ_P^* is fixed.

Bias correction

We can estimate the bias (expected value minus target) of the sample estimate, given by

$$\text{Bias}\left(\hat{\theta}_S\right) = E\left[\hat{\theta}_S\right] - \theta_P^*$$

by using the bootstrap. The bootstrap estimate of the bias is given by

$$\widetilde{\text{Bias}}\left(\hat{\theta}_S\right) = E\left[\tilde{\theta}_B\right] - \hat{\theta}_S$$

We can bias correct the bootstrap sample to get an approximation to the sampling distribution of the estimator. While it might look a little counterintuitive, the correct way to do this to compute the values $2\hat{\theta}_S - \tilde{\theta}_B$. Formally rearranging the distribution approximation (though this is definitely not math), we get $2\hat{\theta}_S - \tilde{\theta}_B \overset{Dist}{\approx} \theta_P^*$. So the values $2\hat{\theta}_S - \tilde{\theta}_B$ "look like" θ_P^*, which is what we want.

Examples of using the `boot` function

In order to use the `boot` function, you must install the `boot` package. This can be achieved with the line

```
install.packages('boot')
```

which only needs to be run once to install the package. When you start `R` and want to use the functions in the `boot` package, you have to run the line

```
library(boot)
```

With the library installed, we can get point estimates from bootstrap replicates easily from a fixed sample.

```
fixed_sample = rnorm(n,mean_dist,sd_dist)
N_replicates = 10000
boot_out_mean = boot(fixed_sample,mean_sample_stat_fun,R=N_replicates)
class(boot_out_mean)
```

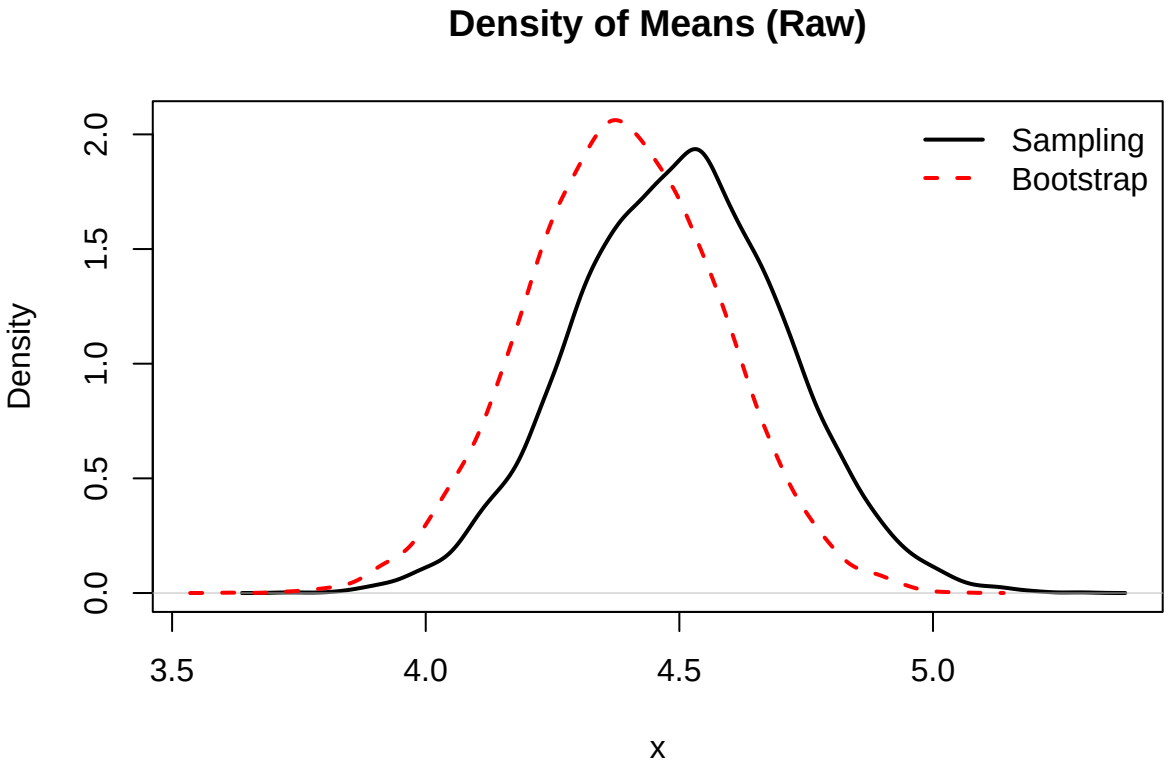
```
## [1] "boot"
```

```
names(boot_out_mean)
```

```
##  [1] "t0"        "t"         "R"         "data"      "seed"      "statistic"
##  [7] "sim"       "call"      "stype"     "strata"    "weights"
```

In the output `t0` is the estimate from the fixed sample and `t` is a vector of the estimates from the bootstrap replicates. We can use bootstrap replicates, `t`, in a number of ways. First let's plot the density of these replicates as well as the density that we got earlier from sampling the population.

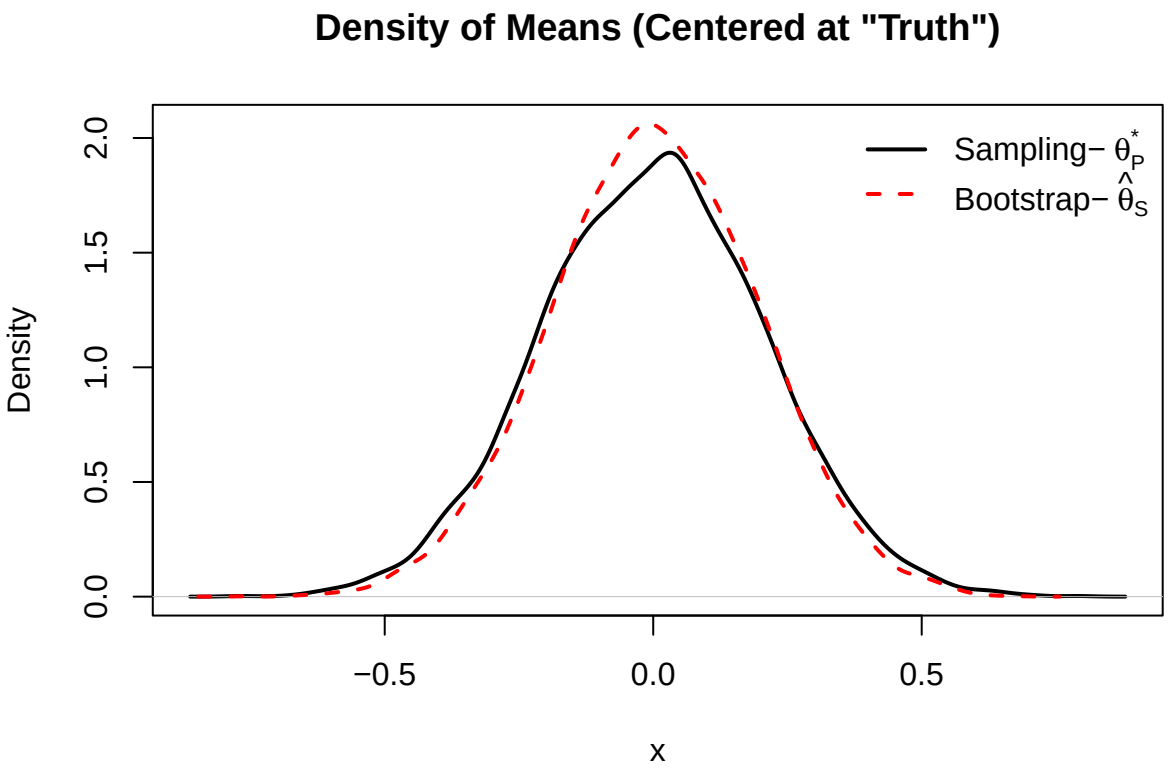
```
density_boot_out_mean = density(boot_out_mean$t)
density_sample_stat_mean = density(sample_stat_mean)
ylim = c(0,max(c(density_boot_out_mean$y,density_sample_stat_mean$y)))
xlim = c(min(c(density_boot_out_mean$x,density_sample_stat_mean$x)),
         max(c(density_boot_out_mean$x,density_sample_stat_mean$x)))
plot(density_sample_stat_mean,ylim=ylim,xlim=xlim,main="Density of Means (Raw)",
     ylab="Density",xlab="x",type="l",lwd=2)
lines(density_boot_out_mean,col="red",lty=2,lwd=2)
legend("topright",c("Sampling","Bootstrap"),col=c("black","red"),lty=c(1,2),bty="n",lwd=2)
```



We can see that the the centers of the distributions are different, which makes sense because the sample estimates should be "around" the population parameter and the bootstrap estimates should be "around" the fixed sample estimate. second, let's do this plot again, centering the sample estimates at the population parameter and the bootstrap estimates at the fixed sample estimate.

```
density_boot_out_mean_centered = density(boot_out_mean$t-boot_out_mean$t0)
density_sample_stat_mean_centered = density(sample_stat_mean - mean_dist)
ylim = c(0,max(c(density_boot_out_mean_centered$y,density_sample_stat_mean_centered$y)))
xlim = c(min(c(density_boot_out_mean_centered$x,density_sample_stat_mean_centered$x)),
         max(c(density_boot_out_mean_centered$x,density_sample_stat_mean_centered$x)))
```

```
plot(density_sample_stat_mean_centered,ylim=ylim,xlim=xlim,
     main='Density of Means (Centered at "Truth")',
     ylab="Density",xlab="x",type="l",lwd=2)
lines(density_boot_out_mean_centered,col="red",lty=2,lwd=2)
legend("topright",
       c(expression('Sampling-'~theta[P]^'*'),expression('Bootstrap-'~hat(theta)[S])),
       col=c("black","red"),lty=c(1,2),bty="n",lwd=2)
```

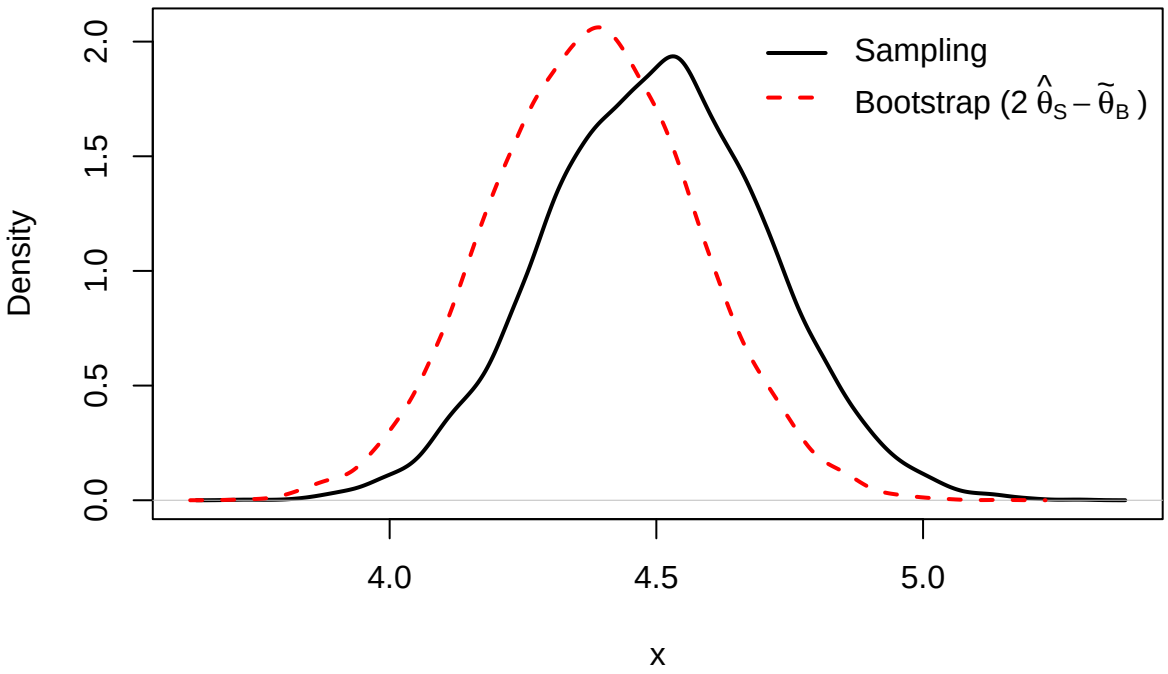


These two plots correspond to each other well. This confirms the distributional approximation

$$\left(\tilde{\theta}_B - \hat{\theta}_S\right) \overset{Dist}{\approx} \left(\hat{\theta}_S - \theta^*_P\right)$$

We do not get to know θ^*_P, so we can't really use the plots that correct the center to do inference. Thirs, we can do the bias corrected bootstrap plot and see how well it corresponds to the sampling distribution.

```
boot_out_mean_bias_corrected = 2*boot_out_mean$t0-boot_out_mean$t
density_boot_out_mean_bias_corrected = density(boot_out_mean_bias_corrected)
density_sample_stat_mean = density(sample_stat_mean)
ylim = c(0,max(c(density_boot_out_mean_bias_corrected$y,density_sample_stat_mean$y)))
xlim = c(min(c(density_boot_out_mean_bias_corrected$x,density_sample_stat_mean$x)),
         max(c(density_boot_out_mean_bias_corrected$x,density_sample_stat_mean$x)))
plot(density_sample_stat_mean,ylim=ylim,xlim=xlim,
     main='Density of Means (bootstrap is bias corrected)',
     ylab="Density",xlab="x",type="l",lwd=2)
lines(density_boot_out_mean_bias_corrected,col="red",lty=2,lwd=2)
legend("topright",
       c(expression('Sampling'),expression('Bootstrap (2'~hat(theta)[S]-tilde(theta)[B]~')')),
       col=c("black","red"),lty=c(1,2),bty="n",lwd=2)
```

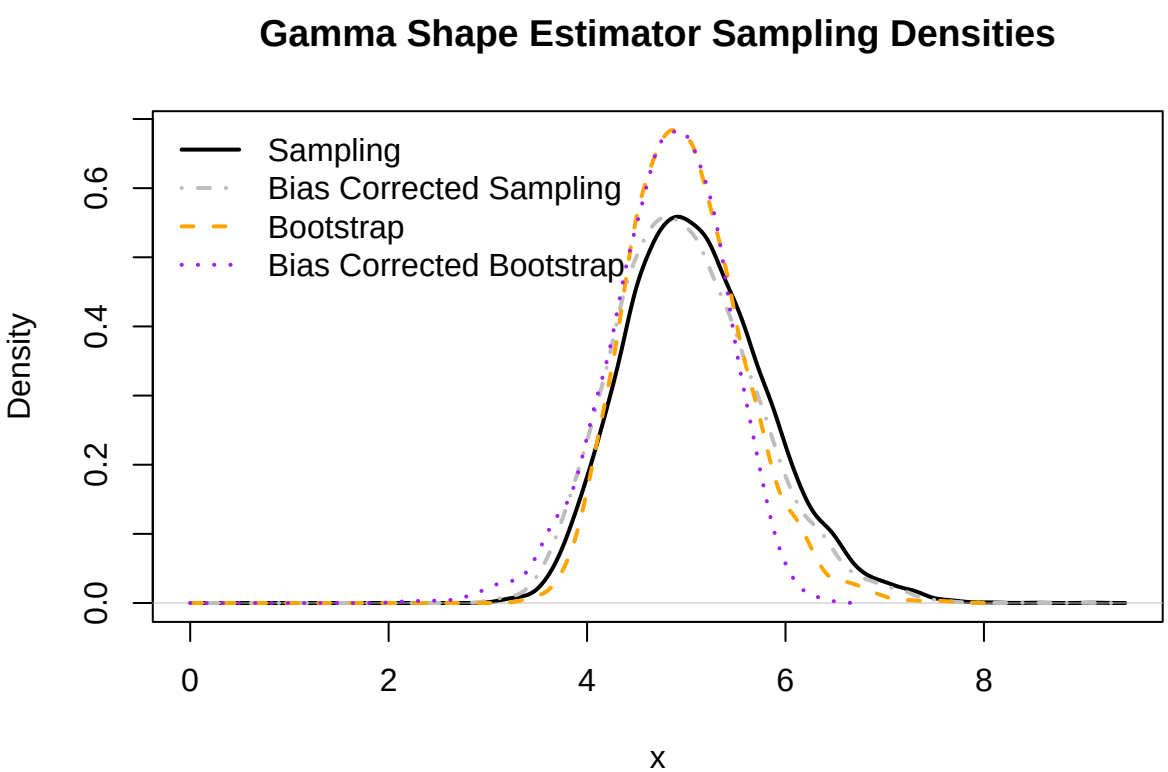


This plot does not look very different from the first plot that we me made using bootstrap replicates. Why could that be? Well, we cannot get anything for free. We do not know the true parameter value θ_P^*, so it is impossible for us to force the center of our guess for the sampling distribution to be at θ_P^*. The best we can do is (1) get the spread right (which the second plot confirms the bootstrap is doing) and (2) make our best guess at the center. In the first plot, our best guess at the center is $\hat{\theta}_S$. In the second plot, our best guess at the center is $2\hat{\theta}_S - \mathrm{E}\left[\tilde{\theta}_B\right]$. This second guess at the center is a bias corrected guess at the center (recall, bias is just the difference between the expected value of the estimator and the true population value).

Let's do this all with a different statistic, an estimate of the shape parameter of a gamma distribution.

```
gamma_shape_est_fun = function(x,ind=c(1:length(x))){
  x_loc = x[ind]
  return(mean(x_loc)^2/var(x_loc))
}
shape = 5
rate = 1
N_samples = 10000
N_replicates = 10000
n = 123
sample_shape_ests = apply(matrix(rgamma(n*N_samples,shape,rate),n,N_samples),
                          2,gamma_shape_est_fun)
sample_shape_ests_bias_corrected = sample_shape_ests-mean(sample_shape_ests)+shape
# we can't actually do bias correction using the true sampling distribution
# and true shape parameter value in practice, but we can use it here for a reference

fixed_sample_gamma = rgamma(n,shape,rate)

boot_out_shape = boot(fixed_sample_gamma,gamma_shape_est_fun,R=N_replicates)
boot_shape_ests = boot_out_shape$t
```

```
boot_shape_ests_bias_corrected = 2*boot_out_shape$t0 - boot_shape_ests

dens_sample_shape_ests = density(sample_shape_ests,from=0)
dens_sample_shape_ests_bias_corrected = density(sample_shape_ests_bias_corrected,from=0)
dens_boot_shape_ests = density(boot_shape_ests,from=0)
dens_boot_shape_ests_bias_corrected = density(boot_shape_ests_bias_corrected,from=0)

ylim = c(0,max(c(dens_sample_shape_ests$y,
                 dens_sample_shape_ests_bias_corrected$y,
                 dens_boot_shape_ests$y,
                 dens_boot_shape_ests_bias_corrected$y)))
xlim = range(c(dens_sample_shape_ests$x,
               dens_sample_shape_ests_bias_corrected$x,
               dens_boot_shape_ests$x,
               dens_boot_shape_ests_bias_corrected$x))
plot(dens_sample_shape_ests,xlim=xlim,ylim=ylim,
     main="Gamma Shape Estimator Sampling Densities",
     xlab='x',ylab='Density',lwd=2)
lines(dens_sample_shape_ests_bias_corrected,col='gray',lty=4,lwd=2)
lines(dens_boot_shape_ests,col='orange',lty=2,lwd=2)
lines(dens_boot_shape_ests_bias_corrected,col='purple',lty=3,lwd=2)

legend('topleft',
       c('Sampling',"Bias Corrected Sampling","Bootstrap", "Bias Corrected Bootstrap"),
       bty='n',lty=c(1,4,2,3),col=c('black','gray','orange','purple'),lwd=2)
```



There are more complicated versions of the bootstrap than the raw version and bias corrected version that have been plotted here. When we do intervals later in the semester, we will use one of these more complicated methods (which will come to use for free).

When we have a vector statistic, the bootstrap outputs a vector for `t0` and a matrix for `t`.

```
boot_out_mean_and_sd = boot(fixed_sample_gamma,mean_and_sd_sample_stat_fun,10000)
boot_out_mean_and_sd$t0
```

```
##     mean       sd
## 4.842999 2.189217
```

```
class(boot_out_mean_and_sd$t)
```

```
## [1] "matrix" "array"
```

```
dim(boot_out_mean_and_sd$t)
```

```
## [1] 10000     2
```

```
head(boot_out_mean_and_sd$t)
```

```
##          [,1]     [,2]
## [1,] 4.716960 2.031584
## [2,] 4.633978 2.137637
## [3,] 4.657974 2.224836
## [4,] 5.115352 2.170459
## [5,] 4.754140 2.088971
## [6,] 4.859002 2.241720
```

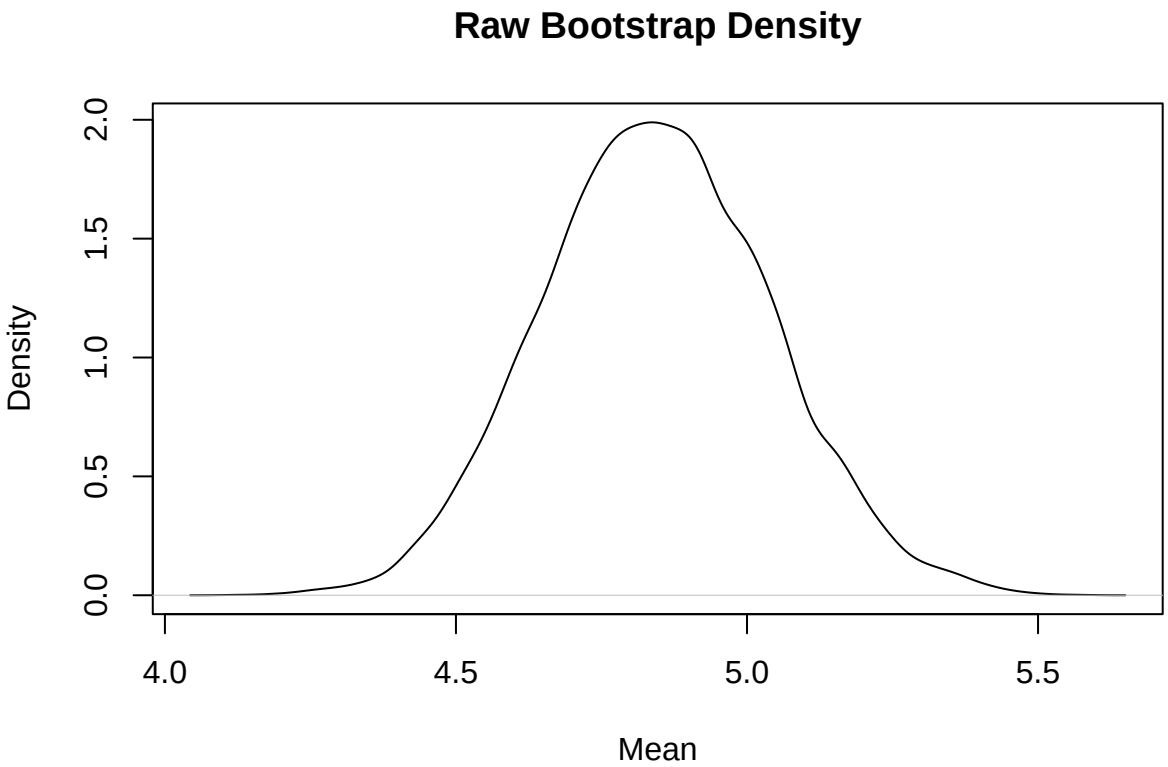
In contrast to our use of `apply` before, these are organized so that each column corresponds to a statistic. Unforunately, `boot` does not pass our names in the output from the statistic function to `t`. However, it does to `t0`. We can use these to attach column names to `t`

```
colnames(boot_out_mean_and_sd$t) = names(boot_out_mean_and_sd$t0)
head(boot_out_mean_and_sd$t)
```

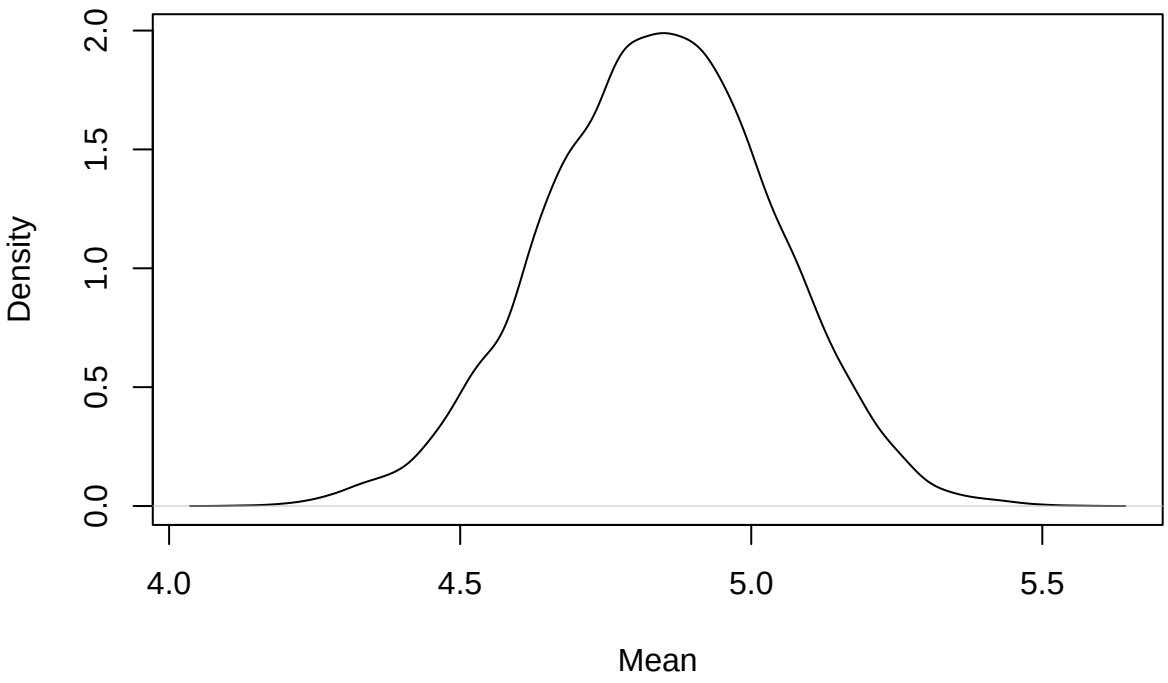
```
##          mean       sd
## [1,] 4.716960 2.031584
## [2,] 4.633978 2.137637
## [3,] 4.657974 2.224836
## [4,] 5.115352 2.170459
## [5,] 4.754140 2.088971
## [6,] 4.859002 2.241720
```

After setting these column names, we can access these elements using column numbers or the column names.

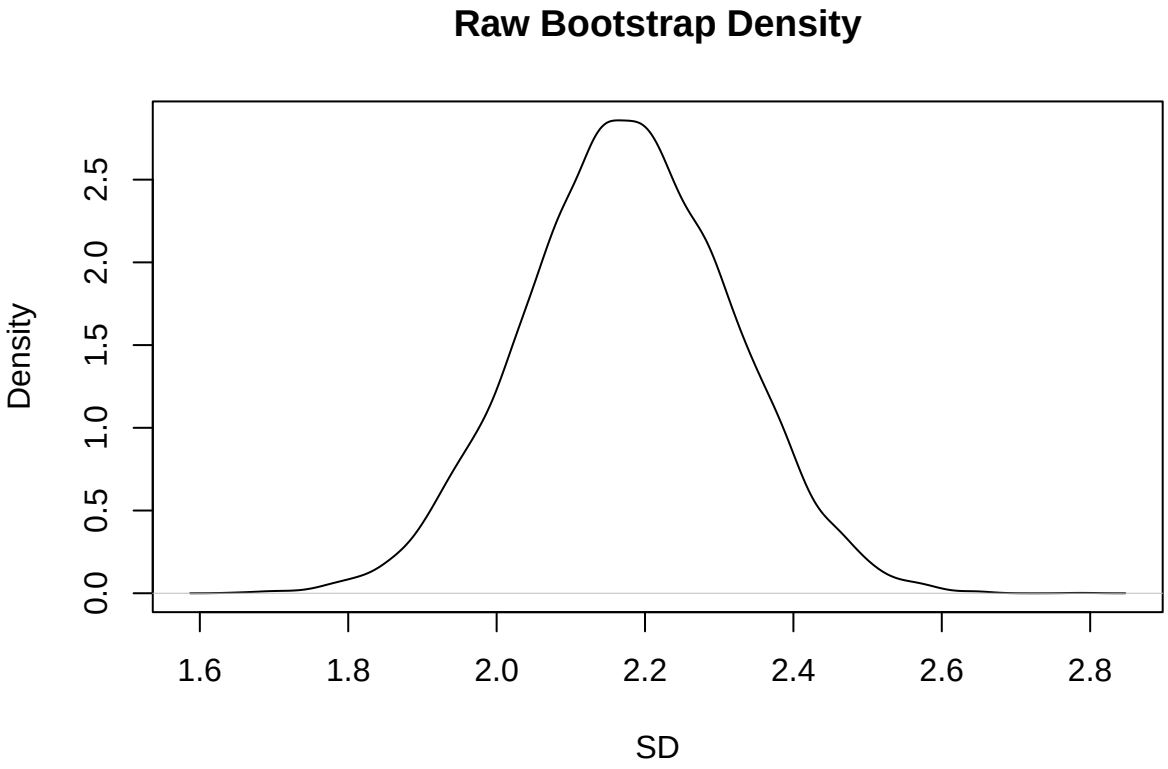
```
plot(density(boot_out_mean_and_sd$t[,'mean']),
     main="Raw Bootstrap Density",xlab='Mean')
```



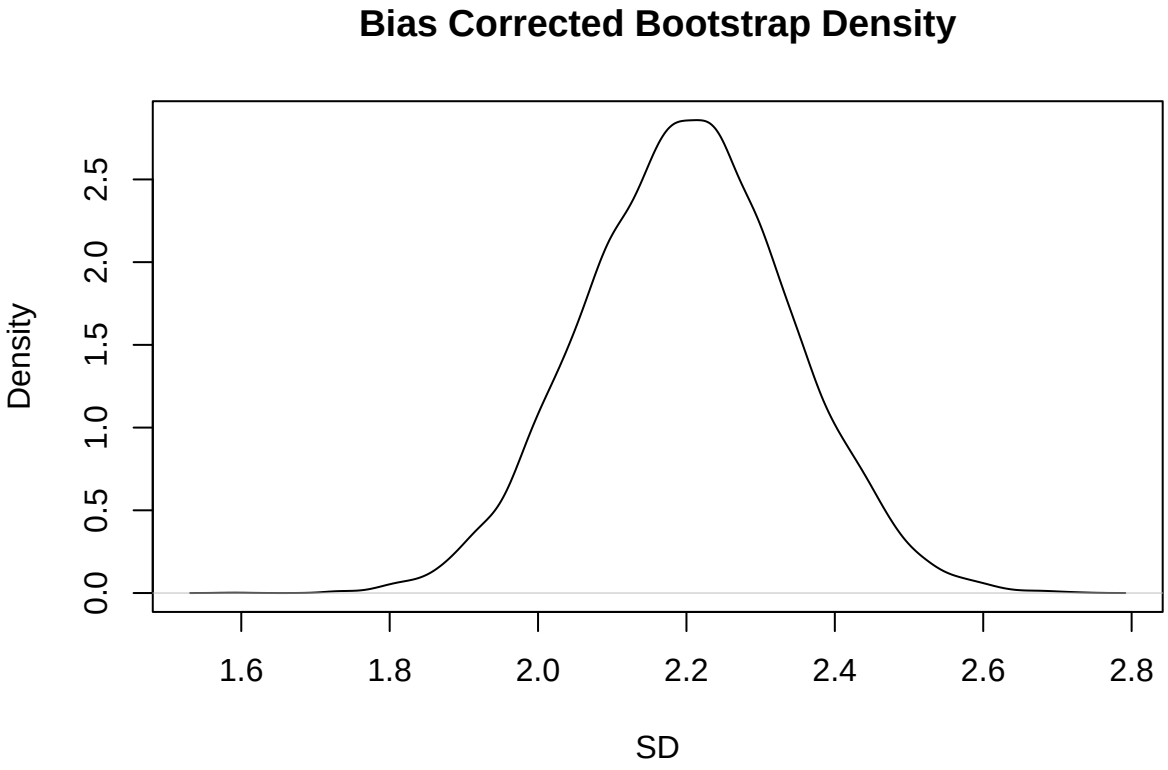

```
plot(density(2*boot_out_mean_and_sd$t0['mean']-
               boot_out_mean_and_sd$t[,'mean']),
     main="Bias Corrected Bootstrap Density",xlab='Mean')
```




```
plot(density(boot_out_mean_and_sd$t[,'sd']),
     main="Raw Bootstrap Density",xlab='SD')
```




```
plot(density(2*boot_out_mean_and_sd$t0['sd']-
               boot_out_mean_and_sd$t[,'sd']),
     main="Bias Corrected Bootstrap Density",xlab='SD')
```



Question 1

In this question, you will compare the shifted sampling distribution to the shifted bootstrap resampling distribution and discuss whether they coincide well. The recentering of the sampling distribution can't be done in practice, but we can do it here because we are generating the samples from a fixed population that we get to design. For each question, you will have six tasks. (1) Write a statistic function that can be used by the `boot` function. (2) Generate samples from the population and compute the statistic for each sample. (3) Generate a fixed sample from the population. (4) Generate bootstrap replicates from the fixed sample and compute that statistic for each bootstrap replicate. (5) Plot the densities of the sample statistic (minus the population value) and the bootstrap replicate statistic (minus the fixed sample estimate) on the same plot. (6) Discuss the plot. There are examples of all of these in the Preamble to this assignment.

Question 1a

A random variable X that follows an exponential distribution with rate parameter $\lambda > 0$ has density on $x > 0$ given by

$$f_X(x) = \lambda \exp(-\lambda x).$$

The expectation of X is $\mathrm{E}[X] = 1/\lambda$ and so a plug-in estimator of the rate for sample values $x_1, \ldots, x_n$ is

$$\hat{\lambda} = \frac{n}{x_1 + \cdots + x_n}.$$

The `R` function for generating random exponential variates is `rexp`.

(1) Write a statistic function for the rate of an exponential distribution that can be used by the `boot` function.

(2) Generate 10000 samples of size 87 from a population following an exponential distribution with rate $\lambda = 0.5$ and compute the rate estimate for each sample.

(3) Generate a fixed sample of size 87 from the population.

(4) Generate 10000 bootstrap replicates from the fixed sample and compute the rate estimate for each bootstrap replicate.

(5) Plot the densities of the sample rate estimates (minus the population rate) and the bootstrap replicate rate estimates (minus the fixed sample rate estimate) on the same plot.

(6) Do the recentered densities correspond well to each other? Are there any major discrepencies that between the two densities?

Question 1b

A random variable X that follows a Poisson distribution with rate parameter $\lambda > 0$ has density on $x = 0, 1, 2, \ldots$ given by

$$f_X(x) = \frac{\lambda^x}{x!} \exp(-\lambda).$$

The expectation of X is $\mathrm{E}[X] = \lambda$ and so a plug-in estimator of the rate for sample values $x_1, \ldots, x_n$ is

$$\hat{\lambda} = \frac{x_1 + \cdots + x_n}{n}.$$

The `R` function for generating random exponential variates is `rpois`.

(1) Write a statistic function for the rate of a Poisson distribution that can be used by the `boot` function.

(2) Generate 10000 samples of size 104 from population following a Poisson distribution with rate $\lambda = 10$ and compute the rate estimate for each sample.

(3) Generate a fixed sample of size 104 from the population.

(4) Generate 10000 bootstrap replicates from the fixed sample and compute the rate estimate for each bootstrap replicate.

(5) Plot the densities of the sample rate estimates (minus the population rate) and the bootstrap replicate rate estimates (minus the fixed sample rate estimate) on the same plot.

(6) Do the recentered densities correspond well to each other? Are there any major discrepencies that between the two densities?

Question 1c

A random variable X that follows a negative-binomial distribution with size parameter $r > 0$ and probability of success parameter $0 < p < 1$ has density on $x = 0, 1, 2, \ldots$ given by

$$f_X(x) = \frac{p^r}{\Gamma(r)} \frac{\Gamma(r+x)}{x!} (1-p)^x.$$

The expectation of X is $\mathrm{E}[X] = r(1-p)/p$ and the variance of X is $\mathrm{Var}[X] = r(1-p)/p^2$. These lead to $p = \mathrm{E}[X]/\mathrm{Var}[X]$. A plug-in estimator of the probability of success for sample values $x_1, \ldots, x_n$ is

$$\hat{p} = \bar{x} / \left(\overline{x^2} - \bar{x}^2 \right)$$

where $\bar{x} = (x_1 + \cdots + x_n)/n$ and $\overline{x^2} = (x_1^2 + \cdots + x_n^2)/n$ The `R` function for generating random negative-binomial variates is `rnbinom`.

(1) Write a statistic function for the probability of success of a negative binomial distribution that can be used by the `boot` function.

(2) Generate 10000 samples of size 72 from population following a negative binimial distribution with size $r = 8$ and probability of success $p = 0.4$ and compute the rate estimate for each sample.

(3) Generate a fixed sample of size 72 from the population.

(4) Generate 10000 bootstrap replicates from the fixed sample and compute the probability of success estimate for each bootstrap replicate.

(5) Plot the densities of the sample probability of success estimates (minus the population probability of success) and the bootstrap replicate probability of success estimates (minus the fixed sample probability of success estimate) on the same plot.

(6) Do the recentered densities correspond well to each other? Are there any major discrepencies that between the two densities?

Question 1d

A random variable X that follows a gamma distribution with shape parameter $\alpha > 0$ and rate parameter $\beta > 0$ has density on $x > 0$ given by

$$f_X(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} \exp(-\beta x).$$

The expectation of X is $\mathrm{E}[X] = \alpha/\beta$ and the variance of X is $\mathrm{Var}[X] = \alpha/\beta^2$. These lead to $\beta = \mathrm{E}[X]/\mathrm{Var}[X]$. A plug-in estimator of the rate for sample values $x_1, \ldots, x_n$ is

$$\hat{\beta} = \bar{x} / \left(\overline{x^2} - \bar{x}^2 \right)$$

where $\bar{x} = (x_1 + \cdots + x_n)/n$ and $\overline{x^2} = (x_1^2 + \cdots + x_n^2)/n$ The `R` function for generating random gamma variates is `rgammma`.

(1) Write a statistic function for the rate of a gamma distribution that can be used by the `boot` function.

(2) Generate 10000 samples of size 65 from population following a gamma distribution with shape $\alpha = 7.5$ and rate $\beta = 2.5$ and compute the rate estimate for each sample.

(3) Generate a fixed sample of size 65 from the population.

(4) Generate 10000 bootstrap replicates from the fixed sample and compute the rate estimate for each bootstrap replicate.

(5) Plot the densities of the sample rate estimates (minus the population rate) and the bootstrap replicate rate estimates (minus the fixed sample rate estimate) on the same plot.

(6) Do the recentered densities correspond well to each other? Are there any major discrepencies that between the two densities?

Question 1e

A random variable X that follows a logistic distribution with location parameter $\mu \in \mathbb{R}$ and scale parameter $s > 0$ has density on $x \in \mathbb{R}$ given by

$$f_X(x) = \frac{1}{s \left(\exp\left(\frac{x-\mu}{2s} \right) + \exp\left(-\frac{x-\mu}{2s} \right) \right)^2}.$$

The expectation of X is $\mathrm{E}[X] = \mu$ and the variance of X is $\mathrm{Var}[X] = s^2\pi^2/3$. The plug-in estimator of the location for sample values $x_1, \ldots, x_n$ is

$$\hat{\mu} = \frac{x_1 + \cdots + x_n}{n}$$

The `R` function for generating random logistic variates is `rlogis`.

(1) Write a statistic function for the location of a logistic distribution that can be used by the `boot` function.

(2) Generate 10000 samples of size 91 from population following a logistic distribution with location $\mu = -2.7$ and scale $s = 3$ and compute the location estimate for each sample.

(3) Generate a fixed sample of size 91 from the population.

(4) Generate 10000 bootstrap replicates from the fixed sample and compute the location estimate for each bootstrap replicate.

(5) Plot the densities of the sample location estimates (minus the population location) and the bootstrap replicate location estimates (minus the fixed sample location estimate) on the same plot.

(6) Do the recentered densities correspond well to each other? Are there any major discrepencies that between the two densities?

Question 2

In this question you will compute the bias corrected bootstrap distribution from a fixed sample. You will plot this distribution, report a summary of it (using the `summary` function), and discuss your assessment of the uncertainty of the point estimate based on this distribution. The idea of the bias correction is to take the bootstrap replicates $\tilde{\theta}_B$ and the fixed sample estimate $\hat{\theta}_S$ and estimate the bias in estimation using the bootstrap by computing

$$\widetilde{\text{Bias}}\left(\hat{\theta}_S\right) = \frac{1}{N_{\text{replicates}}} \sum \tilde{\theta}_B - \hat{\theta}_S$$

This leads to the bias corrected bootstrap estimates given by

$$2\hat{\theta}_S - \tilde{\theta}_B$$

The point estimates of $\hat{\theta}_S - \widetilde{\text{Bias}}\left(\hat{\theta}_S\right)$ have expected value θ_P^* when the sample size is large. There are more interesting (and complicated) corrections than this simple bias correction, but we will keep things simple for this course.

Question 2a

A random variable X follows a Pareto distribution with lower bound $x_0 > 0$ and shape parameter $\alpha > 0$ has density

$$f_X(x) = \frac{\alpha x_0^\alpha}{x^{\alpha-1}}$$

for $x > x_0$. The expected value of $Y = \log(X/x_0)$ is

$$\mathrm{E}[Y] = \mathrm{E}\left[\log\left(\frac{X}{x_0}\right)\right] = \frac{1}{\alpha}$$

which leads to the shape parameter estimator

$$\hat{\alpha} = \frac{n}{\sum \log\left(\frac{x_i}{\widehat{x_0}}\right)}$$

for sample $x_1, \ldots, x_n$ where $\widehat{x_0} = \min\{x_1, \ldots, x_n\}$.

(1) Generate a sample of size 200 from a Pareto distribution with lower bound $x_0 = 3$ and shape $\alpha = 1.5$ using the function

```
rpareto = function(n,x_0,shape){
  return(exp(1/shape*runif(n))*x_0)
}
```

You have to read this function into memory (add it to your code and run it in console) and call it with the right parameters. An example call after reading the function into memory is

```
x = rpareto(200,x_0=3,shape=1.5)
```

(2) Write a function to compute the shape parameter estimator that can be used with `boot`.

(3) Get 10000 bootstrap replicate estimates of the shape parameter using the Pareto sample that you drew. Use these replicate estimates and the point estimate from the sample to get the bias corrected bootstrap estimates.

(4) Plot the density of the bias corrected bootstrap estimates and use the `summary` function on this vector. From these outputs, describe what you think is a reasonable guess at the true population value and how much uncertainty you would want to express in that guess. Plot the empirical distribution function of the bias corrected bootstrap estimates. Does the distribution look smooth to you? Why or why not?

Question 2b

A random variable X follows a Rayleigh distribution with scale parameter $s > 0$ has density

$$f_X(x) = \frac{x}{s} \exp\left(-\frac{x^2}{2s^2}\right)$$

for $x > 0$. This distribution has positive skewness given by

$$\text{Skew}\,[X] = \frac{\text{E}\left[(X - \text{E}[X])^3\right]}{\text{E}\left[(X - \text{E}[X])^2\right]^{\frac{3}{2}}} = \frac{2\sqrt{\pi}(\pi - 3)}{(4 - \pi)^{\frac{3}{2}}} \approx 0.631$$

This skewness does not depend on the scale s. We can estimate this skewness from a sample $x_1, \ldots, x_n$ using the formula

$$\widehat{\text{Skew}} = \sqrt{n}\frac{\sum(x_i - \bar{x})^3}{(\sum(x_i - \bar{x})^2)^{\frac{3}{2}}}$$

where $\bar{x} = (x_1 + \cdots + x_n)/n$.

(1) Generate a sample of size 200 from a Rayleigh distribution with scale $s = 5$ using the function

```
rrayleigh = function(n,scale){
  return(sqrt(rchisq(n,2))*scale)
}
```

You have to read this function into memory (add it to your code and run it in console) and call it with the right parameters. An example call after reading the function into memory is

```
x = rrayleigh(200,scale=5)
```

(2) Write a function to compute the skewness estimator that can be used with `boot`.

(3) Get 10000 bootstrap replicate estimates of the skewness parameter using the Rayleigh sample that you drew. Use these replicate estimates and the point estimate from the sample to get the bias corrected bootstrap estimates.

(4) Plot the density of the bias corrected bootstrap estimates and use the `summary` function on this vector. From these outputs, describe what you think is a reasonable guess at the true population value and how much uncertainty you would want to express in that guess. Does the actual true value of the skewness lie within what you would describe as the bulk of the density you have plotted? Plot the empirical distribution function of the bias corrected bootstrap estimates. Does the distribution look smooth to you? Why or why not?

Question 2c

A random variable X that follows a Laplace distribution with center μ and scale $s > 0$ has density

$$f_X(x) = \frac{1}{2s} \exp\left(-\frac{|x - \mu|}{s}\right)$$

for $x \in \mathbb{R}$. The scale s satisfies

$$s = \mathrm{E}\left[|X - \mu|\right]$$

A common estimator for s based on a data vector $x_1, \ldots, x_n$ is the Mean Absolute Deviation (MAD)

$$\hat{s} = \frac{1}{n} \sum |x_i - \hat{\mu}|$$

where $\hat{\mu} = \text{median}\{x_1, \ldots, x_n\}$.

(1) Generate a sample of size 200 from a Laplace distribution with center $\mu = -10$ and scale $s = 4$ using the function

```
rlaplace = function(n,center,scale){
  return(rexp(n)*scale*(2*rbinom(n,1,0.5)-1)+center)
}
```

You have to read this function into memory (add it to your code and run it in console) and call it with the right parameters. An example call after reading the function into memory is

```
x = rlaplace(200,center=-10,scale=4)
```

(2) Write a function to compute the scale estimator that can be used with `boot`.

(3) Get 10000 bootstrap replicate estimates of the scale parameter using the Laplace sample that you drew. Use these replicate estimates and the point estimate from the sample to get the bias corrected bootstrap estimates.

(4) Plot the density of the bias corrected bootstrap estimates and use the `summary` function on this vector. From these outputs, describe what you think is a reasonable guess at the true population value and how much uncertainty you would want to express in that guess. Plot the empirical distribution function of the bias corrected bootstrap estimates. Does the distribution look smooth to you? Why or why not?

Question 2d

A random variable X that follows a Yule distribution with shape $\rho > 0$ has density

$$f_X(x) = \frac{\rho \Gamma(\rho + 1) \Gamma(x)}{\Gamma(x + \rho + 1)}$$

for $x = 1, 2, 3, \ldots$. The shape ρ satisfies

$$\frac{\rho}{\rho - 1} = \mathrm{E}\left[X\right]$$

when $\rho > 1$ A possible estimator for ρ based on a data vector $x_1, \ldots, x_n$ is

$$\hat{\rho} = \frac{\bar{x}}{\bar{x} - 1}$$

where $\bar{x} = (x_1 + \cdots + x_n)/n$.

(1) Generate a sample of size 200 from a Yule distribution with shape $\rho = 5$ using the function

```
ryule = function(n,shape){
  return(rgeom(n,exp(-rexp(n,shape)))+1)
}
```

You have to read this function into memory (add it to your code and run it in console) and call it with the right parameters. An example call after reading the function into memory is

```
x = ryule(200,shape=5)
```

(2) Write a function to compute the shape estimator that can be used with `boot`.

(3) Get 10000 bootstrap replicate estimates of the shape parameter using the Yule sample that you drew. Use these replicate estimates and the point estimate from the sample to get the bias corrected bootstrap estimates.

(4) Plot the density of the bias corrected bootstrap estimates and use the `summary` function on this vector. From these outputs, describe what you think is a reasonable guess at the true population value and how much uncertainty you would want to express in that guess. Plot the empirical distribution function of the bias corrected bootstrap estimates. Does the distribution look smooth to you? Why or why not?

Question 2e

A random variable X that follows a negative-log-exponential distribution with shape $\alpha > 0$ has density

$$f_X(x) = \alpha x^{\alpha-1}$$

for $0 < x < 1$. If we compute $Y = -\log(X)$ then Y follows and exponential distribution with rate α. The shape α satisfies

$$\frac{\alpha}{\alpha+1} = \mathrm{E}\left[X\right]$$

A possible estimator for α based on a data vector $x_1, \ldots, x_n$ is

$$\hat{\alpha} = \frac{\bar{x}}{1-\bar{x}}$$

where $\bar{x} = (x_1 + \cdots + x_n)/n$.

(1) Generate a sample of size 200 from a negative-log-exponential distribution with shape $\alpha = 2$ using the function

```
rneglogexp = function(n,shape){
  return(rbeta(n,shape,1))
}
```

You have to read this function into memory (add it to your code and run it in console) and call it with the right parameters. An example call after reading the function into memory is

```
x = rneglogexp(200,shape=2)
```

(2) Write a function to compute the shape estimator that can be used with `boot`.

(3) Get 10000 bootstrap replicate estimates of the shape parameter using the negative-log-exponential sample that you drew. Use these replicate estimates and the point estimate from the sample to get the bias corrected bootstrap estimates.

(4) Plot the density of the bias corrected bootstrap estimates and use the `summary` function on this vector. From these outputs, describe what you think is a reasonable guess at the true population value and how much uncertainty you would want to express in that guess. Plot the empirical distribution function of the bias corrected bootstrap estimates. Does the distribution look smooth to you? Why or why not?