Problem set 3 - CSCI E-88b Spring 2020

Due date: Friday, April 17 2020, 10pm EST.

Please write down the last 5 digits of your Harvard ID: 70196

Please write down the URL of your Renku project: https://renkulab.io/projects/stephen.t.knapp14/problem3 (https://renkulab.io/projects/stephen.t.knapp14/problem3)

Please indicate who you may have worked with on this problem and which section you worked with someone on. If you did not work with anyone then you can leave this blank: N/A

This problem set corresponds to 1/6 of your final grade. It is graded out of 10.

Question 3.0: 1pt

Question 3.1: 3pts

Question 3.2: 3pts

Question 3.3: 3pts

Question 3.0: Keep it private!

If your problem set is not private, your grade will be 0/10.

Question: Please confirm that your project is private. You can edit the markdown cell from `[ ]` to `[x]` to check a box.

Your answer (3.0):

- [X] Yes, my project is private.
- [ ] No, my project is not private.

Objectives

We want to use vectorization, parallel computing and `Rcpp` to calculate confidence intervals with the bootstrap.

You can read more about the bootstap in `notebooks/intro_bootstrap.ipynb` .

We use an example from Wasserman (*All of Statistics*, 2004) and adapt the pseudocode he provides on p. 112, Example 8.5 to reimplement his Example 8.7 on p. 113 on bioequivalence:

> When drug companies introduce new medications, they are sometimes required to show bioequivalence. This means that the new drug is not substantially different than the current treatment.

In [2]:
```
placebo <- c(9243, 9671, 11792, 13357, 9055, 6290, 12412, 18806)
old <- c(17649, 12013, 19979, 21816, 13850, 9806, 17208, 29044)
new <- c(16449, 14614, 17274, 23798, 12560, 10157, 16570, 26325)

Z <- old - placebo
Y <- new - old
```

We provide an R version of the pseudocode but adapted to the bioequivalence example.

In [3]:
```
x1 <- Y
x2 <- Z

n1 <- length(x1)
n2 <- length(x2)
B <- 1000
Tboot <- rep(NA, B)

for (i in 1:B) {
  xx1 <- sample(x1, n1, replace = TRUE) # sample of size n1 with replacement f
rom x1
  xx2 <- sample(x2, n2, replace = TRUE) # sample of size n2 with replacement f
rom x2
  Tboot[i] <- mean(xx1) / mean(xx2)
}
```

In [4]:
```
mean(sample(x1, n1, replace = TRUE))
```

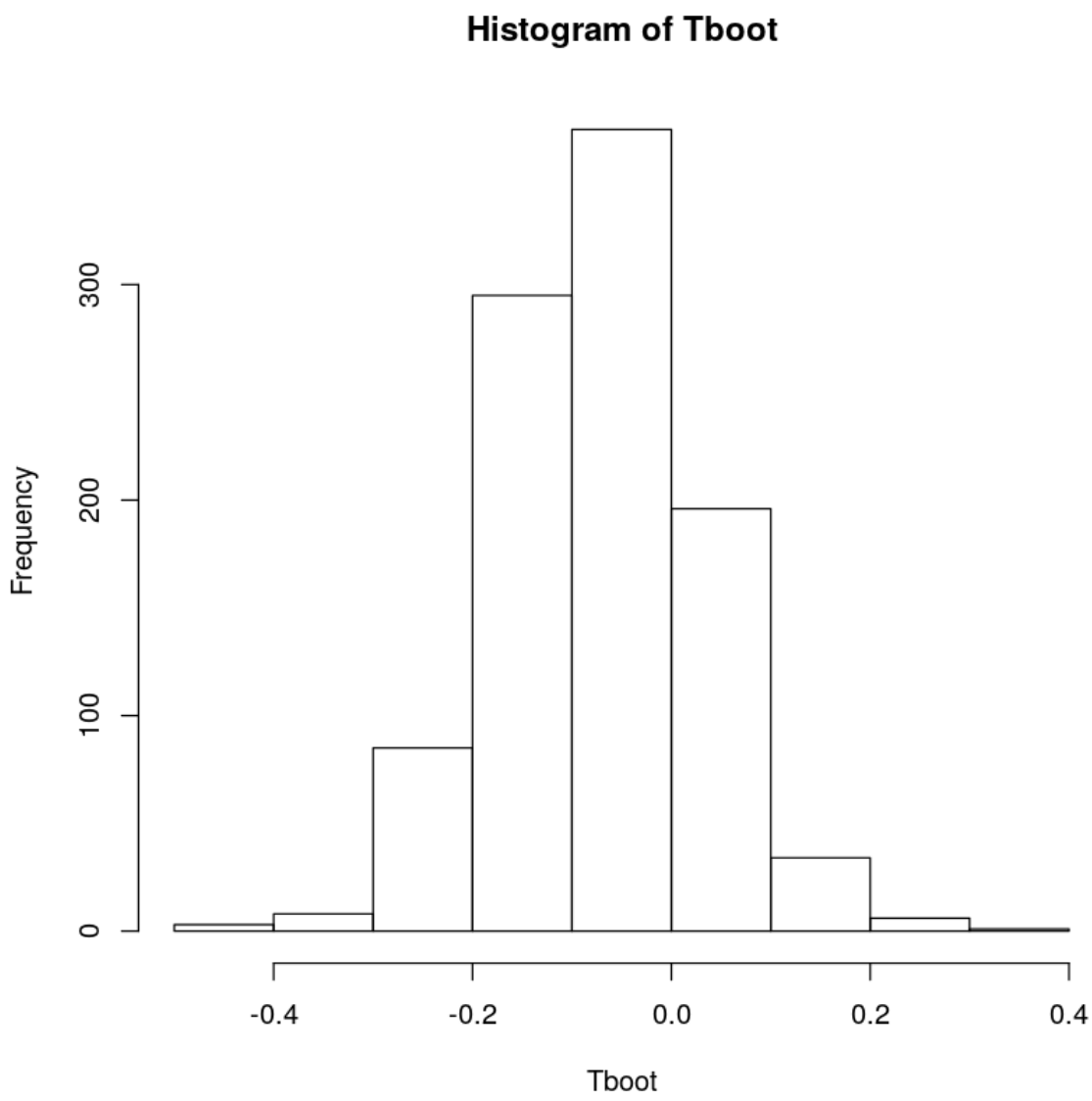
-347.875

We can calculate the 95% bootstrap interval.

In [5]:
```
percentile <- c(quantile(Tboot,.025), quantile(Tboot,.975))
percentile
```

2.5%	-0.27535802421457
97.5%	0.124127213527134

In [6]:
```
hist(Tboot)
```



Let's refactor as a function!

A function will prove way more convenient to write loops. We can refactor the code as follow.

In [7]:
```
ratio_sim_loop <- function(x1, x2, nrep = 1000) {
    n1 <- length(x1)
    n2 <- length(x2)
    B <- nrep
    Tboot <- rep(NA, B)

    xx1 <- rep(NA, n1)
    xx2 <- rep(NA, n2)

    for (i in 1:B) {
        xx1 <- sample(x1, n1, replace = TRUE) # sample of size n1 with replace
ment from x1
        xx2 <- sample(x2, n2, replace = TRUE) # sample of size n2 with replace
ment from x2
        Tboot[i] <- mean(xx1) / mean(xx2)
    }
    return(Tboot)
}
```

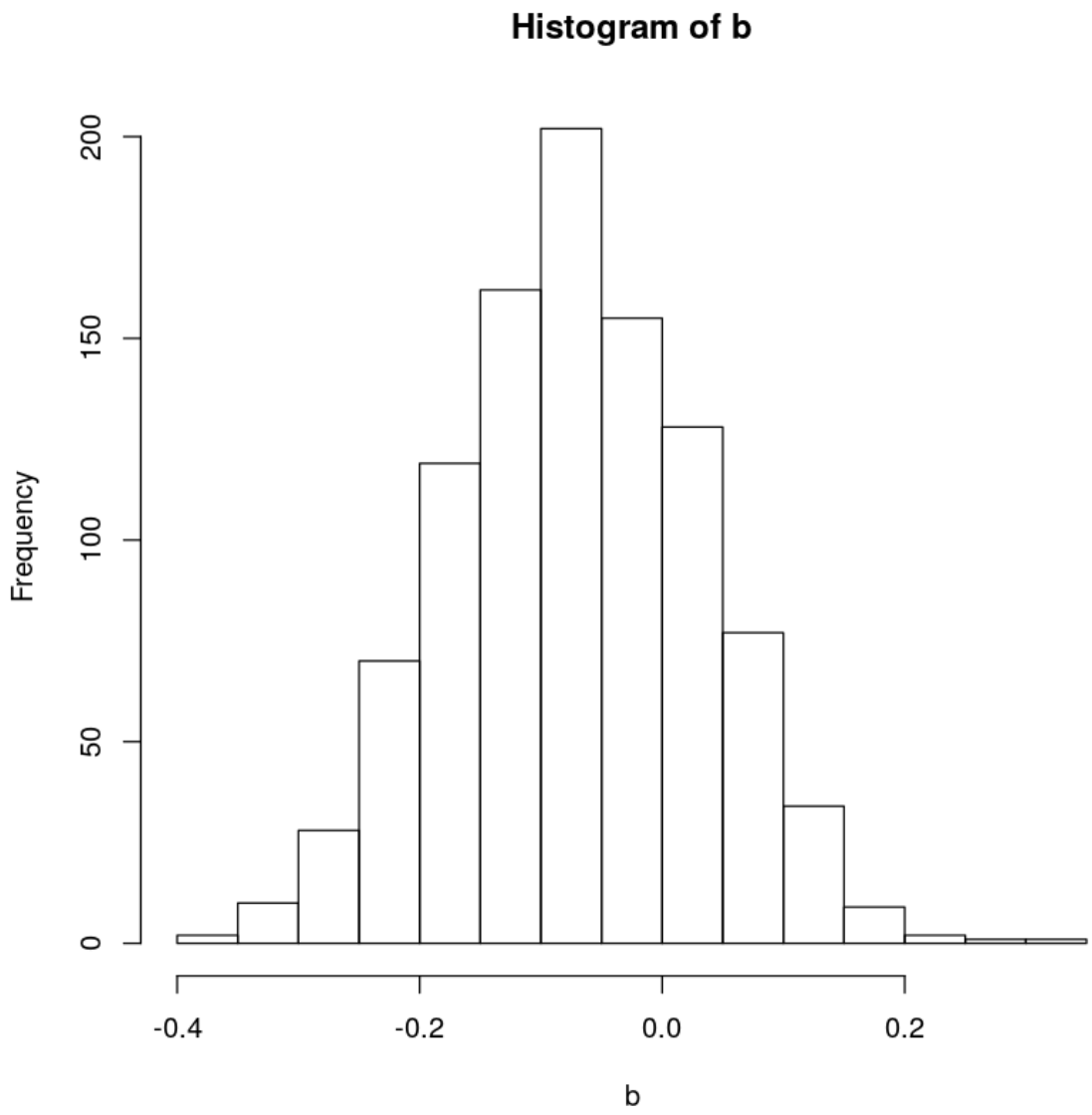
We can also wrap the 95% confidence interval and the histogram in functions.

In [8]:
```
confint_sim <- function(b)
    c(quantile(b,.025), quantile(b,.975))

plot_sim <- function(b) {
    hist(b)
}
```

Our simulation code becomes:

In [9]:
```
b <- ratio_sim_loop(x1, x2, 1000)
confint_sim(b)
plot_sim(b)
```

2.5%	-0.266236250480739
97.5%	0.132693202049222



Question 3.1: Let's vectorize with R!

Question: Write a `ratio_sim_vec` function that vectorizes the simulation and lets you specify the `nrep` number of replications. Your function will return a vector of size with `nrep` rows (similar to `ratio_sim_loop`).

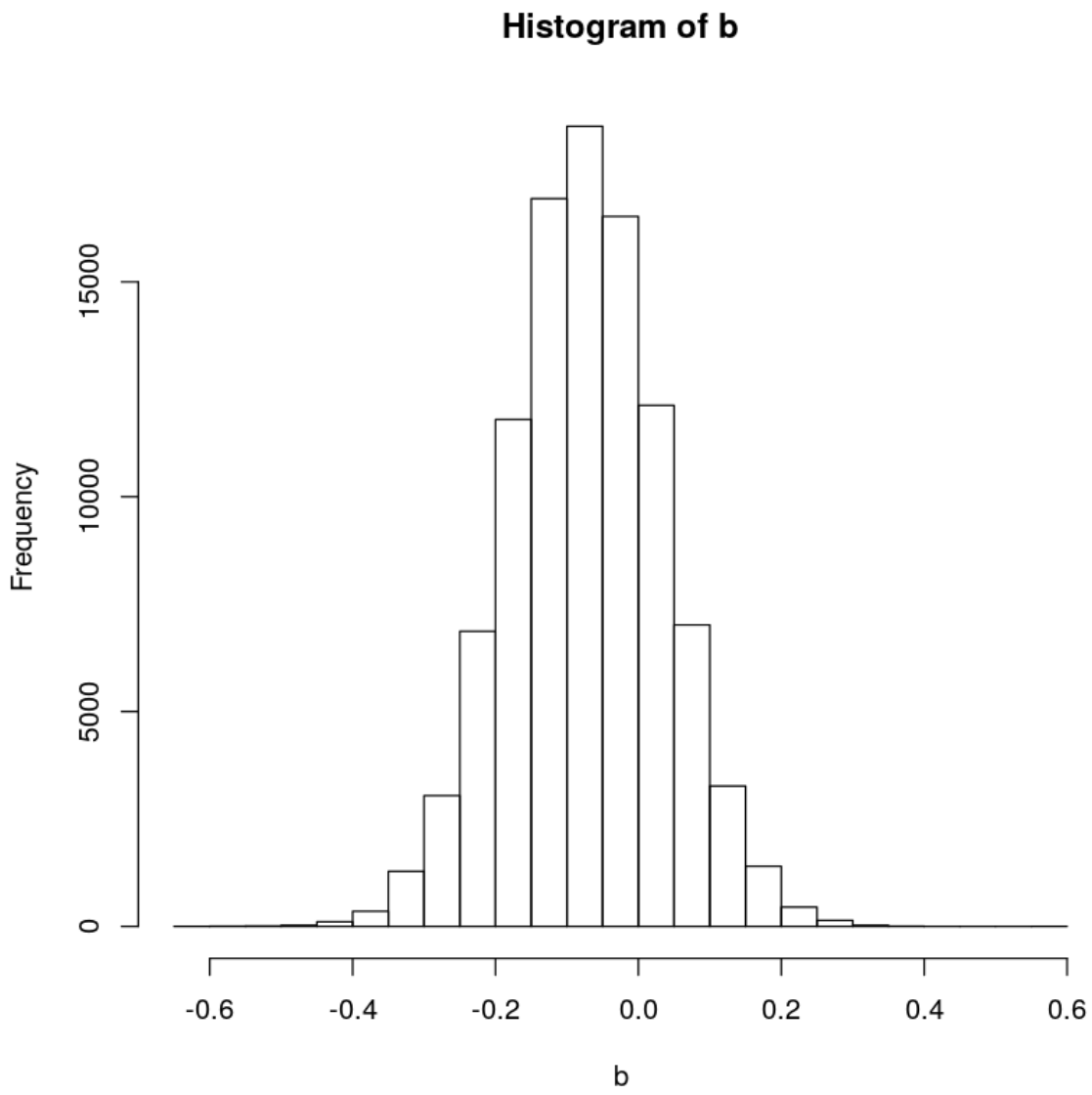
Your answer (3.1):

In [10]:
```
## Because R is vector based many of the algebriac steps in the earlier code c
an be eliminated.
## Instead of creating 3 empty vectors at full size and then constantly rewrit
ing to them with each iteration of the for loop,
## I was able to create vectorized operations to accomplish the same thing.
## B was really just an alias for nrep, so that was eliminated completely.
## I was able to effectively duplicate the resampling of each iteration of the
for loop by applying the replicate function to x1 and x2.
## I replicated each nrep times and I now have two giant vectors of size lengt
h(x1)*nrep and length(x2)*nreps which represent the entire
## sampling population over all nreps.
## Originally each sampling population was sampled length(x1) and length(x2) t
imes, now with the larger pool, I will sample each giant vector
## length(x1) * nrep and length(x2) * nrep times.  This is effectively the sam
e as taking nrep samples repeatedly of x1 and x2.
## Next the vectors were "bent" into the shape of matrices so that I can now t
ake row means using rowMeans()
## Tboot is simply the row means from x1 divided by the row means from x2

ratio_sim_vec <- function(x1, x2, nrep = 1000) {
    n1 <- length(x1)
    n2 <- length(x2)
    xx1 <- sample(rep(x1, nrep), n1 * nrep, replace = TRUE) #combine all x1 sa
mpling iterations into one pool & sample same GRAND TOTAL
    xx2 <- sample(rep(x2, nrep), n2 * nrep, replace = TRUE) #combine all x2 sa
mpling iterations into one pool & sample same GRAND TOTAL
    xx1means <- .rowMeans(xx1, nrep, n1)
    xx2means <- .rowMeans(xx2, nrep, n2)
    Tboot <- xx1means / xx2means #calculate Tboot for each row
    return(Tboot) #return vector
}
```

Please make sure you run the following code snippet:

In [11]:
```
bvec <- ratio_sim_vec(x1, x2, 100000)
confint_sim(bvec)
plot_sim(bvec)
```

2.5%	-0.283404112805753
97.5%	0.139257007882527



Question 3.2: Let's go parallel with R!

We can use the `parallel` package (https://stat.ethz.ch/R-manual/R-devel/library/parallel/html/detectCores.html (https://stat.ethz.ch/R-manual/R-devel/library/parallel/html/detectCores.html)) and we will use detectCores() from the parallel package below to see the number of CPU's on our current host. We expect to see 8 (these is the typical value for the machines that run Renku, it would be different if you were running the snippet locally on your computer).

In [12]:
```
library(parallel)
detectCores() # should be 8
```

8

Question: Using the set of R tools of your choice, write a `ratio_sim_par` function that runs the simulation in parallel and lets you specify the `nrep` number of replications and the `ncore` number of cores (we chose sensible default values of nrep = 1000 and ncore = 3, but of course other default values are possible). Your function will return a vector of size with `nrep` rows (similar to `ratio_sim_loop`).

Your answer (3.2):

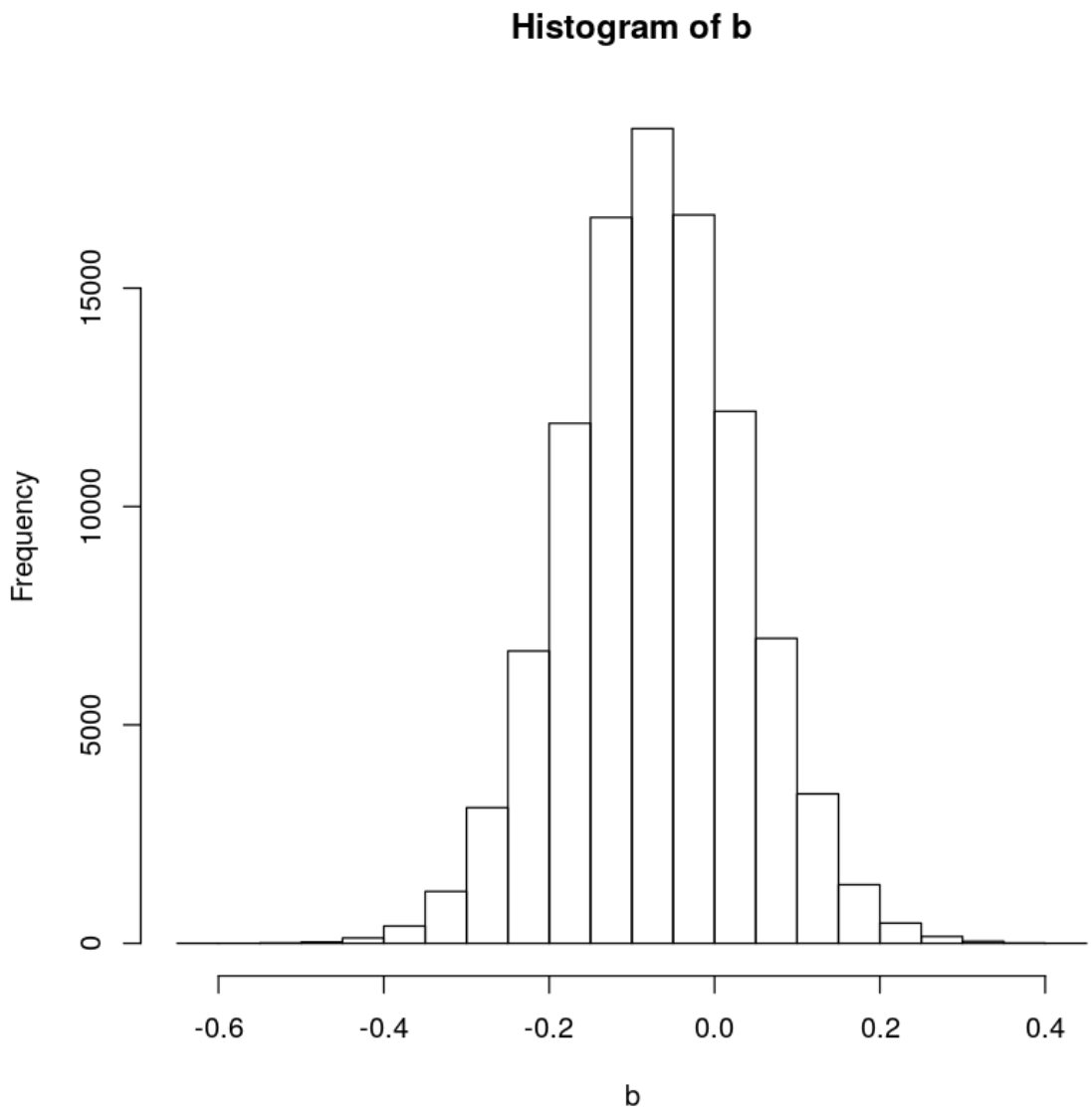
In [13]:
```
library(foreach)
```

In [18]:
```
## In this method I've chosen to use the foreach and %dopar% functions to farm
out different loops of my for loop to different cores
ratio_sim_par <- function(x1, x2, nrep = 1000, ncore=3) { #added ncore as a fu
nction input with a default value of 3
    cl <- makeCluster(ncore) #created 'ncore' clusters
    n1 <- length(x1) #set sample sizes
    n2 <- length(x2) #set sample sizes
    Tboot <- foreach(i = seq(1,nrep), .combine=c) %dopar% { #called foreach fu
nction to do in parallel 1 to 'nrep' times and then
                                                            #combine into a si
ngle vector at the end
        xx1 <- sample(x1, n1, replace = TRUE) #sample x1 n1 times
        xx2 <- sample(x2, n2, replace = TRUE) #sample x2 n2 times
        mean(xx1) / mean(xx2) #calculate Tboot for 1 run
    }
    stopCluster(cl) #stopping cluster
    return(Tboot) #returning the Tboot array as the function output
}
```

Please make sure you run the following code snippet:

In [19]:
```
bpar <- ratio_sim_par(x1, x2, 100000, 3)
confint_sim(bpar)
plot_sim(bpar)
```

2.5%	-0.283590949645176
97.5%	0.139886082082391



Question 3.3: Let's use `Rcpp`!

Question: Write a `Rcpp` version of `ratio_sim` that you will call `ratio_sim_c`. Make sure to run the call after your function definition.

Hint: You can port the code from R to `Rcpp` line by line. In C++ (like in Python), array index starts at 0 (when it starts at 1 with R). So, make sure your loop index goes from 0 to <B. Also, `sample` is available in Rcpp. Booleans are `true` and `false` (when they are `TRUE` and `FALSE` in R, and `True` and `False` in Python).

Your answers (3.13:

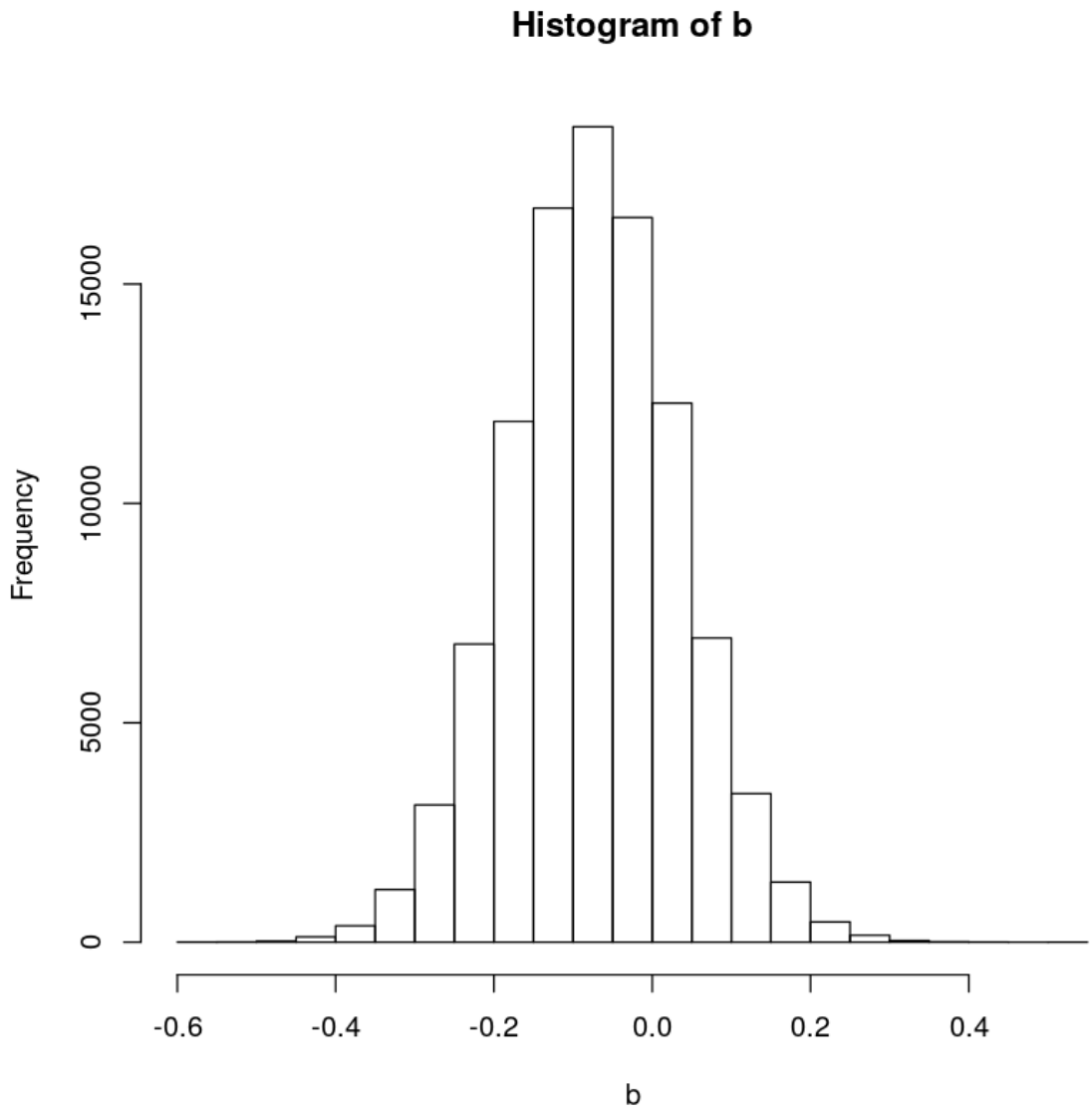
In [20]:
```
library(Rcpp)

## INSERT YOUR (COMMENTED) ANSWER HERE

cppFunction('Rcpp::NumericVector ratio_sim_c(NumericVector x1, NumericVector x
2, int nrep) {
    int n1 = x1.size();
    int n2 = x2.size();
    NumericVector Tboot(nrep);
    for(int i = 0; i < nrep; ++i) {
        NumericVector xx1 = NumericVector(sample(x1, n1, true));
        NumericVector xx2 = NumericVector(sample(x2, n2, true));
        Tboot[i] = mean(xx1) / mean(xx2);
    }
    return Tboot;
}')

ratio_sim_c(x1, x2, 10)
```

```
0.0190709864495623  -0.305802075506184  0.169717286847692  -0.0317738210534158
-0.121446360603355  0.0118506050404877  -0.232868205727645  0.0411896680665018
-0.0544163280975079  -0.118746420948514
```

Please make sure you run the following code snippet:

In [21]:
```
bc <- ratio_sim_c(x1, x2, 100000)
confint_sim(bc)
plot_sim(bc)
```

2.5%	-0.282241435857557
97.5%	0.139807058641711

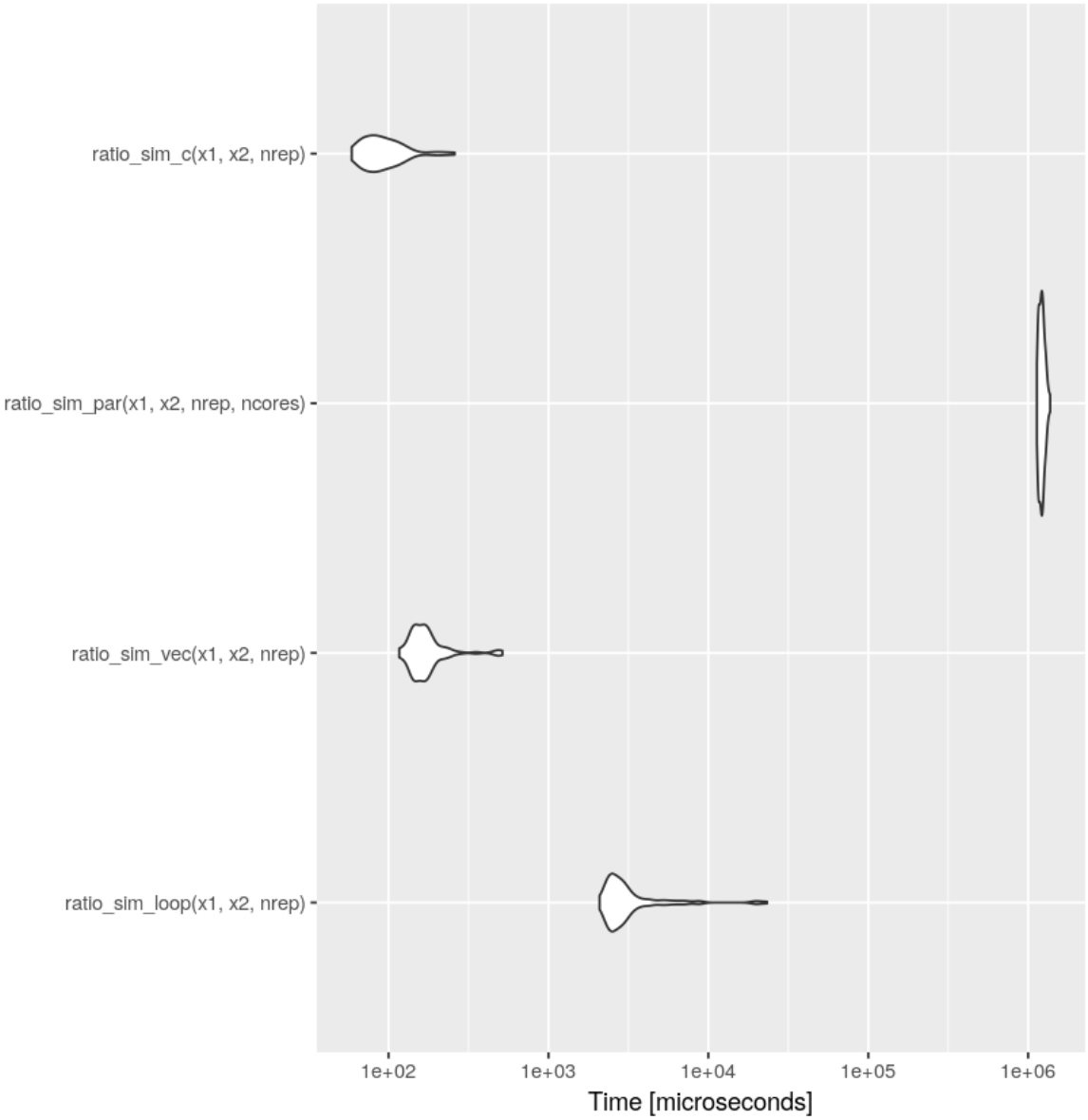


Let's benchmark!

We could benchmark `ratio_sim_loop`, `ratio_sim_vec` and `ratio_sim_c` with the `microbenchmark` package. You can play with the `nrep`, `times` and `n cores` parameters. You may (or not) notice that performance decreases when you increase the number of cores. It may seem paradoxical, but remember your interative session only has 0.25 CPU reserved... We will observe a more more stable and predictible behavior on the Harvard Cannon cluster (https://www.rc.fas.harvard.edu/about/cluster-architecture/ (https://www.rc.fas.harvard.edu/about/cluster-architecture/)).

In [25]:
```
library(microbenchmark)
nrep <- 100
ncores <- 4
bench <- microbenchmark(times = 100,
                        ratio_sim_loop(x1, x2, nrep),
                        ratio_sim_vec(x1, x2, nrep),
                        ratio_sim_par(x1, x2, nrep, ncores),
                        ratio_sim_c(x1, x2, nrep))
ggplot2::autoplot(bench)
```

```
Coordinate system already present. Adding new coordinate system, which will r
eplace the existing one.
```



Self-assessment (not graded)

Questions

1. What do you think I was hoping for you to learn through this homework?
2. Did you find anything particularly challenging?

Your answer

1. I think we were supposed to learn the methods and advantages and disadvantes to parallization, vectorization, and changing to a faster language.
2. I found the C++ code very challenging because I have not touched C++ for 15 years.

Submitting your work

To submit the problem set, export the notebook to HTML and upload the file to Canvas.

[`File -> Export Notebook As... -> Export Notebook to HTML`]