Report of Lab 4: Bootstrap

Alberto Nieto Sandino

November 2, 2018

Corrections for re-submission

All the parts that have been improved based on the comments given or new sections that have been added are written in blue. From Assignment 3.2 onwards, the sections are newly written.

Introduction

This lab revolves around the task of estimating the parameters that define a probability distribution. Whether or not to trust these estimates can be assessed by finding the distribution of our estimator. It can be derived analytically, however, in most cases it must be derived numerically. If a large number of samples N of size n is available, the distribution can be derived from the sample mean for each sample. Our aim is to be able to find this distribution with only one sample by *bootstrapping*. Using this method, we will have N resamples randomly selected with replacement from which we will calculate the estimated distribution. In order to know how accurate this estimate is, we can find its bias and variance, and the confidence intervals around it. Bootstrapping can be done with two different methods:

- *Non-parametric bootstrap*, where we use the empirical distribution of the data to draw elements uniformly random with replacement.
- *Parametric bootstrap*, where the distribution family of the data is assumed and the parameters of the distribution estimated; and then the elements are drawn from this distribution.

1 Assignment 1

1.1 Problem

First, we will work with data from a gamma distribution from which we want to estimate the mean by an estimator ($\hat{\theta}$): the sample mean. Then, we will observe how well this estimator behaves knowing the true value (θ).

1.2 Theory and implementation

We generate a dataset ($n = 100$) from a $Gamma(k, \gamma)$ distribution with shape parameter $k = 2$ and scale parameter $\gamma = 2$.

```
dataset <- rgamma (n, shape = k, scale = gamma)
```

The histogram of the dataset is created by `hist`. The theoretical values for the mean and the variance of which the samples are drawn are known to be $\mathbf{E}[x] = k\gamma$ (mean) and $\mathrm{Var}(x) = k\gamma^2$.

Let $X_1, X_2, ..., X_n$ be a random sample of size n from a population with mean μ and variance σ^2, the definition of the sample mean $\bar{X}$ is

$$\mathbf{E}[\bar{X}] = \mathbf{E}\left(\frac{X_1 + X_2 + \cdots + X_n}{n}\right). \tag{1}$$

n is a constant so it can be taken out of the expectation. Furthermore, we know that the expectation of the sum is always equal to the sum of the expectations based on the linear operator property of the expectation. Thus,

$$\mathbf{E}[\bar{X}] = \frac{1}{n}\left(\mathbf{E}[X_1] + \mathbf{E}[X_2] + \cdots + \mathbf{E}[X_n]\right) \tag{2}$$

Now, we know that $\mathbf{E}[X_1], \mathbf{E}[X_2], ..., \mathbf{E}[X_n]$ all have the same expectation, which is the mean of the population μ from which we are sampling.

$$\mathbf{E}[\bar{X}] = \frac{1}{n}(\mu + \mu + \cdots + \mu) = \frac{1}{n}(n\mu) = \mu \tag{3}$$

So we have proven that the expectation of the random variable $\bar{X}$ is equal to the mean of the population from which we sample.

The definition of the variance of the sample mean is

$$\mathbf{Var}(\bar{X}) = \mathbf{Var}\left(\frac{X_1 + X_2 + \cdots + X_n}{n}\right). \tag{4}$$

Knowing that when a random variable (e.g. $\bar{X}$) gets multiplied by a constant (n), its variance gets multiplied by the squared of that constant. We also know that the variance of the sum of independent random variables is the sum of the variances of those random variables. Thus,

$$\mathbf{Var}(\bar{X}) = \frac{1}{n^2}(\sigma^2 + \sigma^2 + \cdots + \sigma^2) = (\mathbf{Var}(X_1) + \mathbf{Var}(X_2) + \cdots + \mathbf{Var}(X_n)) \tag{5}$$

Now, we know that X_i with $i = 1, 2, ..., n$ are identically distributed, meaning they have the same variance σ^2 which is the one from the distribution that were drawn from. So,

$$\mathbf{Var}(\bar{X}) = \frac{1}{n^2}(\sigma^2 + \sigma^2 + \cdots + \sigma^2) = \frac{1}{n^2}(n\sigma^2) = \frac{\sigma^2}{n} \tag{6}$$

Then, the sample mean is given by

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i, \tag{7}$$

and the sample variance by

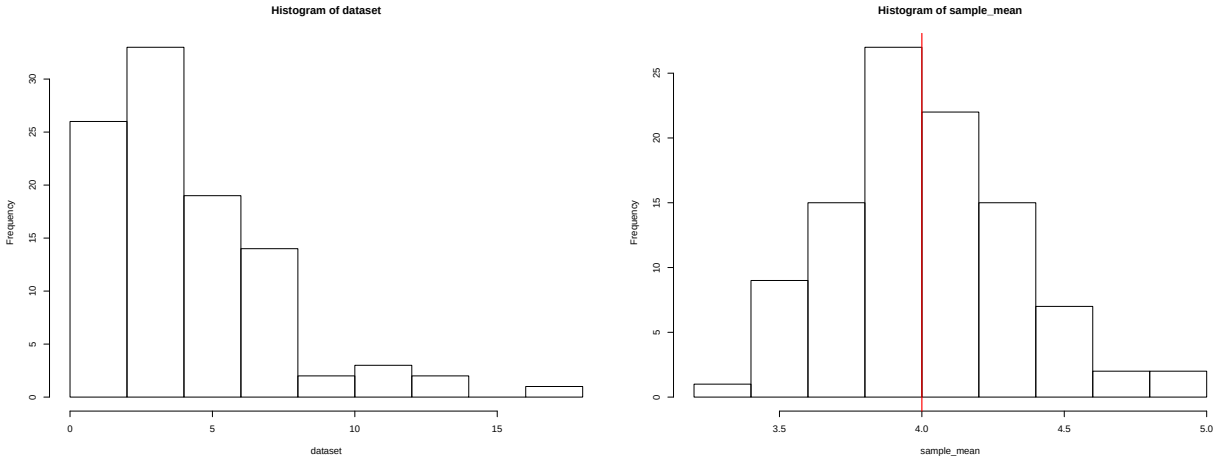
$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2 \tag{8}$$

where $i = 1, 2, ..., 100$ are the samples drawn from the gamma distribution.

Since the standard deviation of the distribution can be considered unknown (it is estimated by s), we can transform $\bar{X}$ to a variable with a *t-distribution*. The *t-distribution* with $n-1$ degrees of freedom will tend to a standard normal distribution as the degrees of freedom increase.

1.3 Results

The histogram of the data can be seen in Figure 1a. The theoretical mean and variance of this distribution are $\mu = 4$ and $\sigma^2 = 8$, respectively. For a sample size of $n = 100$, the estimator sample mean has a theoretical mean $\bar{x} = 4.0017$ and a theoretical variance $s^2 = 8.102$. The distribution of the sample mean can be seen in Figure 1. As described in the theory, it follows approximately a normal distribution. This would be more evident if the sample size would be bigger.



(a) Gamma distribution that constitutes the dataset

(b) Distribution of the sample mean

Figure 1: Histograms of the distributions

Non-parametric bootstrap

2 Assignment 2.1

2.1 Problem

With the dataset generated in assignment 1, we will estimate the mean using the sample mean as estimator ($\hat{\theta}$) with non-parametric bootstrapping. The non-parametric bootstrap approximates the distribution of the sample mean and this distribution will be compared to the original dataset comparing the estimated bias and variance.

2.2 Theory and implementation

Firstly, we generate the new big gamma distributed dataset we will use to draw the samples from. We fix the number of resamples that the bootstrap will do at $N = 1000$. Then, we construct the bootstrap sample x_i^* by drawing 100 uniformly random elements from the dataset (with replacement) using `sample`. Now that we have what is equivalent to the empirical cdf $\hat{F}$, we estimate the parameter θ (i.e. the sample mean) from the sample x_i^*. This is defined as a function `BootstrapMean` which calculates the sample mean (`mean` of the samples).

We will replicate this function as many times as specified by the bootstrap resamples N through the function `replicate`. Thus we obtain the empirical distribution of $(\hat{\theta}_1^*, ..., \hat{\theta}_N^*)$ which is the approximation of the true distribution $\hat{\theta}$ of the dataset.

The theoretical bias $\mathbf{Bias}(\hat{\theta})$ and theoretical variance $\mathbf{Var}(\hat{\theta})$ of the estimator $\hat{\theta}$ are calculated by

$$\mathbf{Bias}(\hat{\theta}) = \mathbf{E}[\hat{\theta} - \theta] = \mathbf{E}[\hat{\theta}] - \theta, \qquad \mathbf{Var}(\hat{\theta}) = \mathbf{E}\left[(\hat{\theta} - \mathbf{E}[\hat{\theta}])^2\right] \tag{9}$$

The bias is a measure of a systematic error, while the variance is a measure of a random error. To obtain the bootstrap estimate, we substitute θ for the original estimate $\hat{\theta}$. We also substitute $\hat{\theta}$ for the distribution from which we get the bootstrap $\hat{\theta}^*$. Thus, the approximations of the bias and the variance are

$$\mathbf{Bias}(\hat{\theta}) \approx \frac{1}{N}\sum_i \hat{\theta}_i^* - \hat{\theta} = \hat{\theta}_\cdot^* - \hat{\theta}, \qquad \mathbf{Var}(\hat{\theta}) \approx \frac{1}{N-1}\sum_i (\hat{\theta}_i^* - \hat{\theta}_\cdot^*)^2 \tag{10}$$

2.3 Results

Figure 2 shows the distribution of the non-parametric bootstrap algorithm. In order to compare how similar they are to the theoretical results, we can observe the bias and variance of the estimates. The bias is equal to $\mathbf{Bias}(\hat{\theta}) = 0.00465$, which is close to zero as it would be expected from the theoretical bias (i.e. from Ass. 1, we know that it is $= 0.0017 \approx 0$) since our estimator is unbiased. The variance is equal to $\mathbf{Var}(\hat{\theta}) = 0.0739$, which is very close to the theoretical value of $\frac{\sigma^2}{n} = \frac{8}{100} = 0.08$ from the gamma distribution the samples were drawn from.

As we can see the approximation given by estimator is good compared to the original since the bias is small and the variance shows that the distribution is not very spread.

3 Assignment 2.2

3.1 Problem

This assignment is aimed at exploring the already built-in function that performs the bootstrap (`boot`) and comparing it to the previous results in assignment 2.1.

3.2 Theory and implementation

The output of the boot function are the bootstrap replicates generated for the estimator. The first parameter specifies the data to sample from, the second specifies the statistic we are interested in (i.e. the sample mean) and the third parameter is the number of replicates to execute.

```
boot.out <- boot(dataset.big, fun, 3000, parallel="multicore", ncpus = no_
    cores)
```

The rest of the parameters are merely to improve the computing time of the algorithm through the use of multiple cores of the CPU.

The statistic-function is defined as

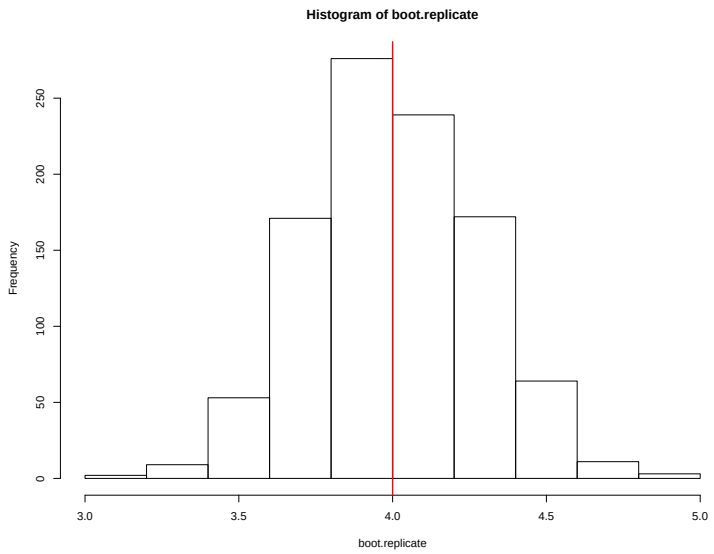


Figure 2: Histogram of the distribution of the estimator ($\hat{\theta}$), the sample mean, with non-parametric bootstrap

```
fun <- function(y = dataset.big, id) {mean(y[id])}
```

It produces the sample mean out of the elements drawn from the dataset specified by the bootstrap algorithm for every replication.

3.3 Results

Figure 3 shows histogram for the estimated sample mean with non-parametric bootsptrap. We can quantify the approximation to the original distribution knowing that $\mathbf{Bias}(\hat{\theta}) = -0.000946$ and $\mathbf{Var}(\hat{\theta}) = 0.00410$.

Both histograms (Figure 2 and Figure 3) show a very similar approximately normal distribution centered around the theoretical mean (red line). When we look at the numerical results from the bias and the variance, we see that the estimate from the `boot` function gives a more accurate estimation (i.e. smaller bias) and is also less spread (i.e. smaller variance). The results are significant enough to conclude that the `boot` function gives a better approximation to the original distribution than the own algorithm. It is expected that our function performs a little worse than the `boot` function since we are taking some approximations to do the calculations.

4 Assignment 2.3

4.1 Problem

In this task, we will construct three different types of confident intervals (CI): basic, normal and percentile. After that, we will check the results of the written function with the already implemented function `boot.ci`.

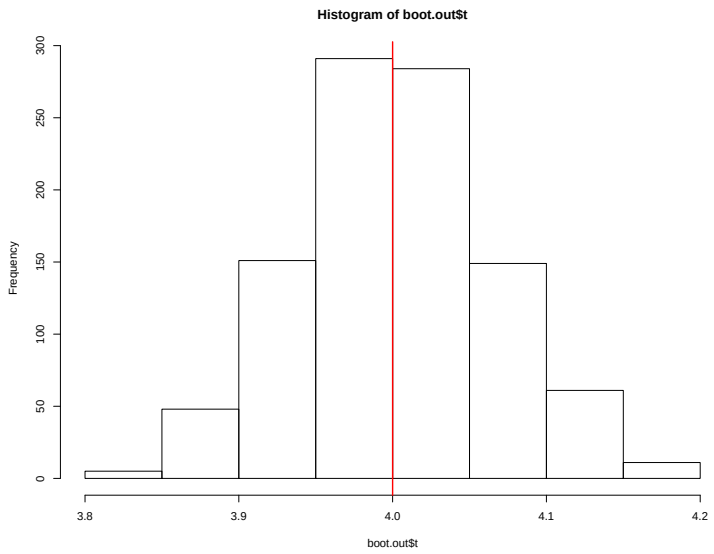


Figure 3: Histogram of the distribution of the estimator ($\hat{\theta}$) with non-parametric bootstrap using `boot`

4.2 Theory and implementation

4.2.1 Basic CI

Since the distribution of $W = \hat{\theta} - \theta$ is unknown, we must work with the approximation $W^* = \hat{\theta}^* - \theta$. Thus, the CI becomes $[\hat{\theta} - w^*_{1-\alpha/2}, \hat{\theta} - w^*_{\alpha/2}]$ where $w^*_\alpha = \hat{\theta}^*_\alpha - \hat{\theta}$ is the corresponding α-quantile, and $\hat{\theta}^*_\alpha$ is the α-quantile of the bootstrap distribution. This gives a CI in the form of $[2\hat{\theta} - \hat{\theta}^*_{1-\alpha/2}, 2\hat{\theta} - \hat{\theta}^*_{\alpha/2}]$. This is implemented in R as

```
alpha <- 0.05
l.low <- 2*orig.estimate - sort(boot.out$t)[(1-alpha/2)*1000]
l.upper <- 2*orig.estimate - sort(boot.out$t)[(alpha/2)*1000]
ci.basic <- c(l.low,l.upper)
```

4.2.2 Normal CI

If we assume that $W = \dfrac{\hat{\theta} - \theta}{\hat{se}}$ follows a normal distribution $N(0, 1)$, assuming pivotality, we obtain that the CI look like this: $[\hat{\theta} - z_{1-\alpha/2}\hat{se}, \hat{\theta} - z_{\alpha/2}\hat{se}]$, where $\hat{se}$ is the estimated standard deviation of $\hat{\theta}$ obtained from the bootstrap sample, and z_α is the α-quantile from a normal distribution. This is in implemented in R as

```
l.low <- orig.estimate + qnorm(alpha/2, mean = 0, sd = 1)*se.nonparam/sqrt
    (100)
l.upper <- orig.estimate - qnorm(alpha/2, mean = 0, sd = 1)*se.nonparam/
    sqrt(100)
ci.normal<- c(l.low,l.upper)
```

4.2.3 Percentile CI

We can define the CI as $[\hat{\theta}^*_{\alpha/2}, \hat{\theta}^*_{1-\alpha/2}]$, where $\hat{\theta}^*_\alpha$ is the α-quantile of the bootstrap distribution. It can be implemented in R as

```
l.low <- quantile(boot.out$t, probs = 0.025)
l.upper <- quantile(boot.out$t, probs = 0.975)
ci.percentile <- c(l.low,l.upper)
```

4.3 Results

The results for the constructed confidence intervals and the ones calculated with `boot.ci` are shown in Table 1.

	constructed	`boot.ci`
basic	$[3.879, 4.137]$	$[3.878, 4.140]$
normal	$[3.956, 4.065]$	$[3.878, 4.137]$
percentile	$[3.885, 4.143]$	$[3.884, 4.143]$

Table 1: Confidence interval values for the constructed CI and those given by `boot.xi`

As expected, the results are quite similar between both methods.

5 Assignment 2.4

5.1 Problem

In this task, we aim to see how reliable the bootstrap confidence intervals are when using a small sample size from $n = 10$ to $n = 100$ by increments of 10. Arbitrarily, we choose to inspect the values of the basic CI. The main goal is to see if the true theoretical value is still covered by the confidence intervals.

5.2 Theory and implementation

In this exercise, we check for every of the 1000 samples drawn and for the different sizes of n how many of 1000 CIs generated cover the true value of the mean with a confidence level of 95%. For each sample, we calculate the *normal* bootstrap CI by `boot.ci` and then check `if` the normal confidence interval covers the true value of the mean.

5.3 Results

The results for the coverage of the true mean of the normal CI can be seen in Figure 4. It can be clearly seen that there is a bigger chance of covering the true value, the larger n is. For $n = 20$, we have a decent coverage of 91%. For values of n between $n = 40$ and $n = 70$, the normal CI has approximately 94% probability of covering the true value. And for $n \geq 80$, we finally achieve a confidence over 95% which is what would be expected for a good estimation and a level that can be admitted as reliable enough.

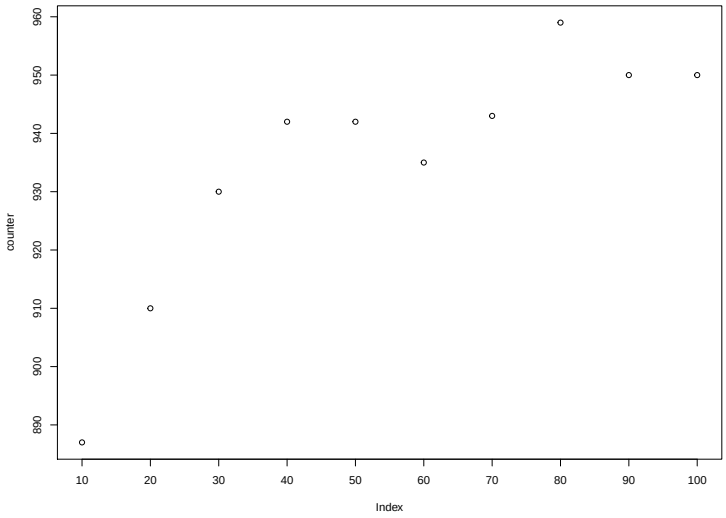


Figure 4: Evolution of the coverage of the normal CI for different sizes of n

Parametric bootstrap

6 Assignment 3.1

6.1 Problem

In this part of the lab, we aim to do the same as in assignment 2 but with a parametric bootstrap. The particularity of the parametric method is that is assumes the distribution of the data (i.e. gamma distributed in our case) and guesses the parameters of the distribution using maximum-likelihood estimation (MLE). In this task, we build a function to use MLE to find the parameters that characterize a simulated gamma distribution.

6.2 Theory and implementation

In order to do parametric bootstrap, we need to define our estimator: maximum-likelihood estimation. MLE is a method to estimate the parameter of a data set and a statistical model given observations. We first need to define the *joint density function* from a sample $x_1, x_2, ..., x_n$ of size n with i.i.d. observations drawn from a distribution function f. Thus, we have that

$$f(x_1, x_2, ..., x_n|\theta) = f(x_1|\theta)f(x_2|\theta)...f(x_n|\theta). \tag{11}$$

The parameters we are interested (i.e. k and γ) form the vector θ and define the likelihood function $\mathcal{L}(\theta; x_1, x_2, ..., x_n) = f(x_1, x_2, ..., x_n|k, \gamma) = \prod_{i=1}^{n} f(x_i)$. To simplify the calculus, we will work with the log-likelihood[1] instead: $l(\theta; x) = \ln\mathcal{L}(\theta; x_1, ..., x_n) = \sum_{i=1}^{n} \ln f(x_i|\theta)$.

[1] We will work with the negative log-likelihood in R since most optimization packages look for a minimum

In our case, we must find the estimator for the gamma distribution, which pdf looks like

$$f(x;k,\gamma) = \frac{1}{\Gamma(k)\gamma^k} x^{k-1} e^{-\frac{x}{\gamma}}, \tag{12}$$

where $\Gamma(k)$ is the gamma function. We can derived the full expression by using the properties of the logarithms, which gives

$$l(k,\gamma) = \ln \prod_{i=1}^{n} \frac{1}{\Gamma(k)\gamma^k} x^{k-1} e^{-\frac{x}{\gamma}} = (k-1)\sum_{i=1}^{n} \ln(x_i) - \sum_{i=1}^{n} \frac{x_i}{\gamma} - nk\ln(\gamma) - n\ln(\Gamma(k)) \tag{13}$$

Now that we know the log-likelihood function is defined, we can find the optimal parameters by differentiating with respect to each parameter k and γ and then equating the expression to zero. Doing this gives us $\hat{\gamma} = \frac{1}{kn}\sum_{i=1}^{n} x_i = \frac{\bar{x}}{k}$. However, the same can not be applied for $\hat{k}$, thus we need to get a solution numerically.

In R, we define the negative log-likelihood in the function `mlogl`. We give a first guess for the parameters based on the mean and the variance of the sample

```
k.guess.start <- mean(x)^2 / var(x)
gamma.guess.start <- var(x) / mean(x)
theta.start <- c(k.guess.start, gamma.guess.start)
```

Finally, we can find the fitted solution `fit` by

```
fit <- nlm (mlogl, theta.start, x = x, hessian = TRUE, fscale = length(x))
```

6.3 Results

For a test with a gamma distribution of theoretical parameters $k = 2$ and $\gamma = 2$, we obtain $\hat{k} = 1.963$ and $\hat{\gamma} = 2.042$, which are pretty good estimations of the parameters defining the gamma distribution.

7 Assignment 3.2

7.1 Problem

In this task, we write a function to approximate the distribution of $\hat{\theta}$ using parametric bootstrap. We will present the distribution of the estimator and its bias and variance.

7.2 Theory and implementation

To implement parametric bootstrap, let $x = (x_1, x_2, ..., x_n)$ be a data set. We first assume a distribution family from which the data is drawn and it is described by a set of parameters $\psi = k, \gamma$. We then estimate these parameters ψ using MLE as in Assignment 3.1 to obtain $\hat{\psi} = \hat{k}, \hat{\gamma}$. We sample with replacement a new set x^* of size n from the estimated distribution $F_{\hat{\psi}}$. From this distribution, we estimate the statistic of interest $\hat{\theta}^*$. We repeat the process N times and store the result in $\hat{\theta}_i^*$ with $i = 1, ..., N$ to get the empirical distribution of statistic of interest.

This algorithm is implemented in R by using the estimated parameters $\hat{k}, \hat{\gamma}$ from the assignment 3.1 to generate the data set. We then resample with replacement and calculate the sample mean of those samples for N times in a `for` loop to get the empirical distribution $\hat{\theta}_i^*$.

7.3 Results

The empirical distribution of the sample mean can be seen in Figure 5. It can be seen that it follows a normal distribution, as the non-parametric bootstrap estimation did. The quality of the approximation can be quantified with $\mathbf{Bias}(\hat{\theta}) = -0.000946$ and $\mathbf{Var}(\hat{\theta}) = 0.00410$. As expected, the results for the bias and the variance are similar to the ones obtained with non-parametric bootstrap.

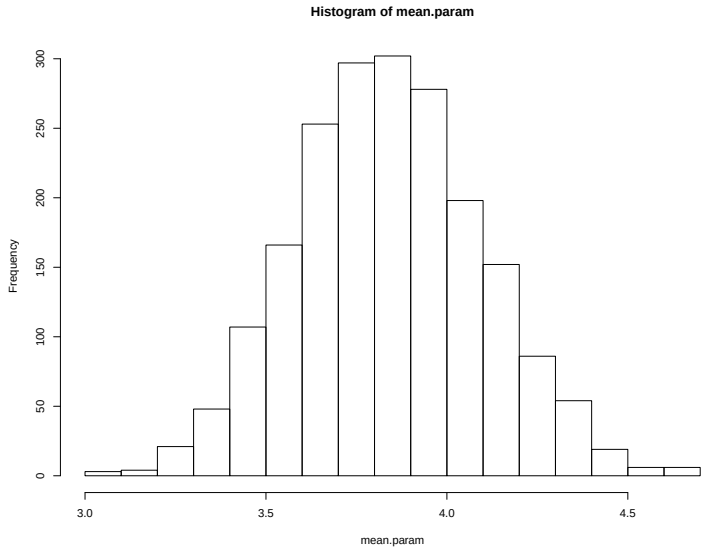


Figure 5: Histogram of the distribution of the estimator $(\hat{\theta})$, the sample mean, with parametric bootstrap

8 Assignment 3.3

8.1 Problem

Our task now is to estimate the distribution of $\hat{\theta}$ with parametric bootstrap using the built-in function `boot`. Then, we observe its histogram and see its estimated bias and variance.

8.2 Theory and implementation

In assignment 2.2, we already used the function `boot` to estimate the distribution of $\hat{\theta}$ using non-parametric bootstrap. Now, we will use the same function to do parametric bootstrap. The statistic (i.e. sample `mean`) is the same as with non-parametric bootstrap. Other same arguments are the dataset, and the number of bootstrap replicates (in order to compare the performance of both).

On the other hand, we call the parameter `sim = ''parametric''`, so as to specify the type of bootstrap that is implemented. Furthermore, we add a parameter `ran.gen` which specifies the way random number are generated in order to simulate data of the same shape as the observed data. This function has two arguments. The first argument should be the observed data and the second argument consists of the parameter estimates. Finally, `mle` contains the maximum likelihood estimates of the parameters.

8.3 Results

In Figure 6, we see the histogram of the estimated distribution of $\hat{\theta}$ on the left. It can be seen that the distribution seems to follow that of a normally distribution. On the right, we can see the qq-plot. As thought before, we can clearly see that it is normally distributed.

The quality of the approximation can be quantified with the bias and the variance: $\mathbf{Bias}(\hat{\theta}) = -0.00943$ and $\mathbf{Var}(\hat{\theta}) = 0.07861836$. As expected, the bias and the variance are very close to the theoretical values.

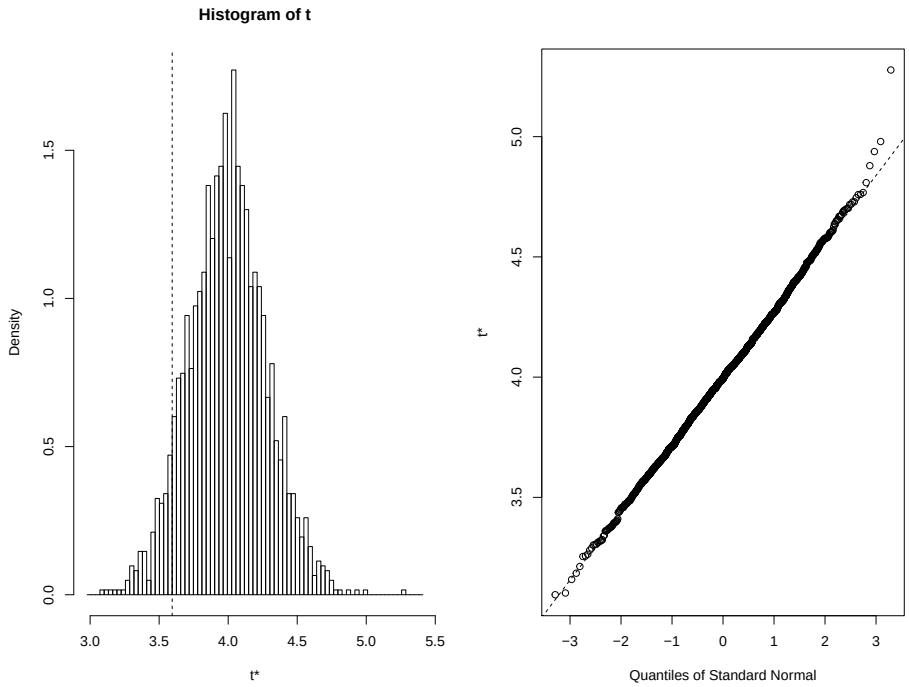


Figure 6: Histogram (left) and qq-plot (right) of the distribution of the estimated parameter, the sample mean

9 Assignment 3.4

9.1 Problem

In this task, we will observe the reliability of the CIs in the same manner as in Assignment 2.4.

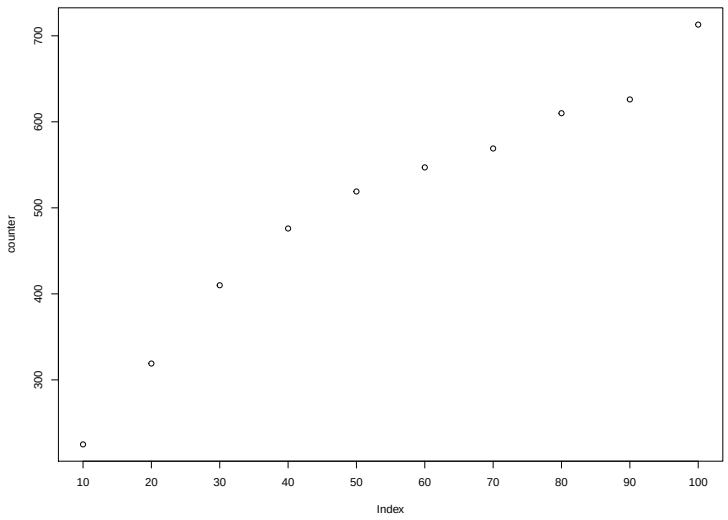


Figure 7: Evolution of the coverage of the normal CI for different sizes of n with parametric bootstrap

9.2 Theory and implementation

The overall approach for this problem is similar to that of the Assignment 2.4 with some relevant changes. The implementation of the function `boot` was adapted to perform parametric bootstrap as in Assignment 3.3. That is, we specified the parameters $sim = \text{“}parametric\text{”}, ran.gen = gengamma, mle = param_{MLE}$. However, now we must recalculate the estimation of the parameters every loop since the original samples get resampled every iteration, thus new estimated parameters are required to perform the bootstrap estimate adequately. The normal confidence intervals are calculated in the same manner as in Assignment 2.4.

9.3 Results

In Figure 7, we see the evolution of the coverage of the true value of the sample mean for the different sample sizes n. We can see the results are not as good as one could have hoped. It has a positive trend towards increasing the coverage percentage as n increases, but for the values of n inspected, the non-parametric bootstrap performs better. This could be caused by the not-so-accurate estimation of our MLE estimating the parameters for the gamma distribution. We can also see that the parametric bootstrap is much smoother than the non-parametric where there is big oscillations in the coverage of the true value (Figure 4). It is, thus, expected that for bigger sample sizes, the perfomance of the parametric bootstrap will improve and compensate the error created in the MLE algorithm.

Studentized CI

10 Assignment 4

10.1 Problem

In this task, we will estimate the true variance instead of the true mean. We will repeat Assignment 2.4 using the basic CI for the coverage of the true variance. Then, we will do the same for the studentized confidence intervals and compare both of them.

10.2 Theory and implementation

For the first part of the assignment, we need to first define the statistic we are interested on. Thus, we define the function for the vector of the variances. Besides that, the algorithm is the same since we are performing non-parametric bootstrap as in Assignment 2.4, except for the fact that in this case it is specified the type of confidence interval to perform (i.e. `basic`.

The studentized confidence interval is similar to the basic CI. However, in this case we substitute the quantiles of the normal distribution for the quantiles of the bootstrap distribution of the Student's t-test. To circumvent the pivotality assumption, the studentized CI (`type=''stud''`) considers the standardized version of $W = (\hat{\theta} - \theta)/\sigma$, instead of what the basic CI does ($W = \hat{\theta} - \theta$). We do not know the distribution of W, so it is approximated by $W^* = (\hat{\theta}^* - \theta)/\hat{se}^*$, where $\hat{se}^*$ is the standard deviation of each bootstrap sample.

The problem with this method is that the estimate of $\hat{se}^*$ is not easily obtained. We find it by performing an extra bootstrap for every original bootstrap sample. This inner bootstrap lools like the following code

```
fun_stud <- function(y, id) {
  boot.out <- boot(y, fun_var, 50,
                   parallel="multicore", ncpus = no_cores)
  return (c(var(y[id]), var(boot.out$t)))
}
```

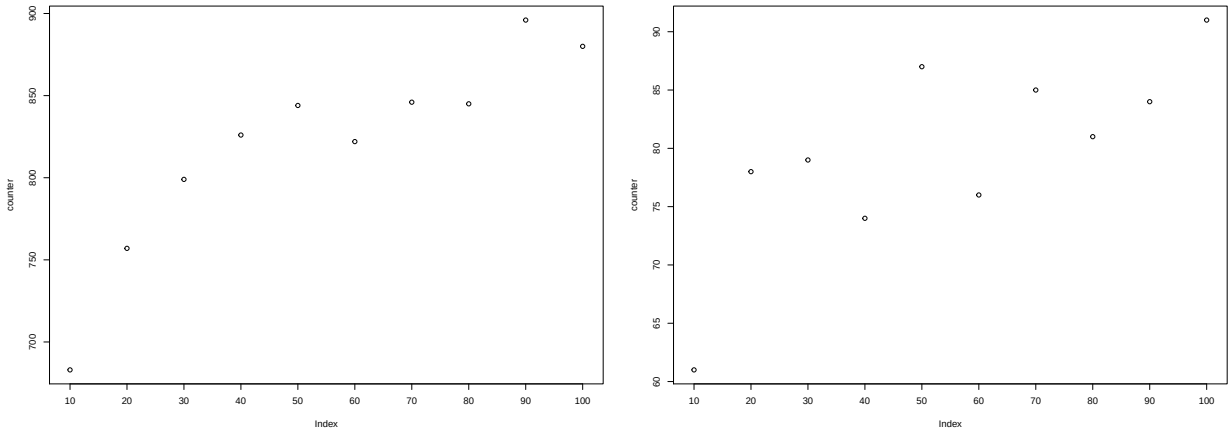
Finally, as before, we count the number of CIs that cover the true value of the variance.

10.3 Results

In Figure 8a, we see the coverage of the basic CIs. Although the results have an upward trend the higher the sample size, they do not achieve the expected 95% coverage. This can be caused by the lack of pivotality originating from our distribution $\hat{\theta}/\sigma$, since it is not independent from σ, the parameter to estimate.

Secondly, we do the studentized CI were we expect to get better results since we circumvent the pivotality assumption. It took 41.3 minutes to compute this algorithm due to the large amount of computations involved in it, especially bootstrapping inside a bootstrap. In Figure 8b, the results can be seen. The results are not as good as expected since there is not a big improvement compared to the basic CI. It could be possible that there is not enough of a large sample to get a good estimation.

The computations for the second bootstrap could be substitute for an intermediate solution like a jackknife estimate.



(a) Basic CI (b) Studentized CI

Figure 8: Evolution of the coverage of CIs for different sizes of n

Appendix - R code

```
# Lab 04 - Bootstrap
# Author: Alberto Nieto Sandino
# Date: 2018-10-09

# Clean environment
closeAllConnections()
rm(list=ls())
# Clear plots
graphics.off()
# Number of cores to use for bootstrap
library("parallel")
no_cores <- detectCores() - 1
library("boot")

##########################################################################
## Assignment 1. The gamma distribution and the estimator sample mean ##
##########################################################################

n <- 100 # number of samples
k <- 2 # shape parameter
gamma <- 2 # scale parameter
dataset <- rgamma (n, shape = k, scale = gamma)

# Histogram of the data
hist(dataset)

# Theoretical mean and variance
mean_theo <- k*gamma
```

```r
var_theo <- k*gamma^2

# Our parameter of interest theta = mean
# thus theta_hat = sample mean

# Data-generating experiment
N = 100 # repeated 100 times
sample_mean <- numeric(N) # initiate vector for sample mean values
sample_var <- numeric(N) # initiate vector for sample variance values
for (i in 1:N){
  dataset = rgamma (n, shape = k, scale = gamma)
  sample_mean[i] <- mean(dataset)
  sample_var[i] <- var(dataset)
}

# Theoretical mean and variance of the estimator (sample mean)
mean_hat <- mean(sample_mean)
var_hat <- mean(sample_var)

# Observe the distribution for the sample mean
hist(sample_mean)
abline(v=mean_theo,lwd=2, col="red")

##############################################
## Assignment 2. Non-parametric bootstrap ##
##############################################

## Assignment 2.1.

# 1) Fix the number of bootstrap re-samples N
N <- 1000
# Generate the empirical cdf
dataset.big <- rgamma (2000, shape = k, scale = gamma)
# 2) Sample a new data set x* set of size n from x with replacement (this
   is equivalent
# to sampling from the empirical cdf F^)

BootstrapMean <- function (X=dataset.big){
  x.boot <- sample(X, size = 100, replace = TRUE)
  mean(x.boot)
}

# Use replicate to perform N iterations
boot.replicate <- replicate(N, BootstrapMean())

# Plot of the histogram with the true value of the mean
hist(boot.replicate,breaks=10)
abline(v=mean_theo,lwd=2, col="red")

# Calculate estimted bias and variance of estimator
boot.estimate <- mean(boot.replicate)
```

```r
orig.estimate <- mean(dataset.big)
bias.nonparam <- boot.estimate - orig.estimate
bias.nonparam
var.nonparam <- var(boot.replicate)
var.nonparam

## Assignment 2.2

# Load library
library(boot)

# First, generate a vector of means for bootstrapping
fun <- function(y = dataset.big, id) {mean(y[id])}

# Boot function for N replicates
boot.out <- boot(dataset.big, fun, 1000, parallel="multicore", ncpus = no_
    cores)

# Plot of the histogram with the true value of the mean
hist(boot.out$t,breaks=13)
abline(v=mean_theo,lwd=2, col="red")

# Calculate estimted bias and variance of estimator
boot.estimate.boot <- mean(boot.out$t)
bias.nonparam.boot <- boot.estimate.boot - orig.estimate
bias.nonparam.boot
var.nonparam.boot <- var(boot.out$t)
var.nonparam.boot

## Assignment 2.3

# Construc CIs for the estimator

# Basic CI
alpha <- 0.05
l.low <- 2*orig.estimate - sort(boot.out$t)[(1-alpha/2)*1000] # lower
    limit of CI
l.upper <- 2*orig.estimate - sort(boot.out$t)[(alpha/2)*1000] # upper
    limit of CI
ci.basic <- c(l.low,l.upper)

hist(boot.out$t,20)
lines(rep(ci.basic[1],2),c(0,850),lty=2)
lines(rep(ci.basic[2],2),c(0,850),lty=2)

# Normal CI
se.nonparam <- sd(boot.out$t)
l.low <- orig.estimate + qnorm(alpha/2, mean = 0, sd = 1)*se.nonparam/sqrt
    (100)
l.upper <- orig.estimate - qnorm(alpha/2, mean = 0, sd = 1)*se.nonparam/
    sqrt(100)
```

```r
ci.normal<- c(l.low,l.upper)

hist(boot.out$t,20)
lines(rep(ci.normal[1],2),c(0,350),lty=2)
lines(rep(ci.normal[2],2),c(0,350),lty=2)

# Percentile CI
l.low <- quantile(boot.out$t, probs = 0.025)
l.upper <- quantile(boot.out$t, probs = 0.975)
ci.percentile <- c(l.low,l.upper)

hist(boot.out$t,20)
lines(rep(ci.percentile[1],2),c(0,350),lty=2)
lines(rep(ci.percentile[2],2),c(0,350),lty=2)

# Pre-programmed confidence intervals
ci.nonparam <- boot.ci(boot.out = boot.out, conf = 0.95, type = c("basic",
   "norm","perc"), parallel="multicore", ncpus = no_cores)

error.ci <- c(ci.basic,ci.normal,ci.percentile) -
              c(ci.nonparam$normal[2:3],ci.nonparam$basic[4:5],ci.nonparam
                 $percent[4:5])

## Assignment 2.4

N <- 1000 # number of samples to draw
counter <- rep(0,10)
for (i in 1:N){
  for (j in 1:10){
    dataset.small <- rgamma(j*10, shape = k, scale = gamma)
    boot.out <- boot(dataset.small, fun, N)
    ci <- boot.ci(boot.out, conf = 0.95, type = "norm")$norm
    if (ci[2] <= mean_theo && ci[3] >= mean_theo){
      counter[j] <- counter[j] + 1 # true value within limits
    }
  }
}

plot(counter,xaxt = "n",)
axis(1,at=seq(1,10),labels = seq(10,100,by=10))

########################################
## Assignment 3. Parametric bootstrap ##
########################################

## Assigment 3.1

# Create the distribution test for a gamma distribution
x <- rgamma (length(dataset.big), shape = 2, scale = 2)
```

```r
mlogl <- function(theta, x) {
  if (length(theta) != 2)
    stop("theta must be vector of length 2")
  k.guess <- theta[1]
  gamma.guess <- theta[2]
  if (k.guess <= 0) stop("The guess for the shape parameter, k, must be
     positive")
  if (gamma.guess <= 0) stop("The guess for the scale parameter, gamma,
     must be positive")
  return(- sum(dgamma(x, shape = k.guess, scale = gamma.guess, log = TRUE)
     ))
}

k.guess.start <- mean(x)^2 / var(x)
gamma.guess.start <- var(x) / mean(x)
theta.start <- c(k.guess.start, gamma.guess.start)

fit <- nlm (mlogl, theta.start, x = x, hessian = TRUE, fscale = length(x))
print(fit$estimate)

k_MLE <- fit$estimate[1]
gamma_MLE <- fit$estimate[2]
## Assignment 3.2
n <- 100
N <- 2000
# Generation of random samples gamma distributed with parameters from MLE
gamma_MLE <- rgamma(n, shape = k_MLE, scale = gamma_MLE)

# Boot function for N replicates
mean.param <- rep(0,N)
for (i in 1:N){
  resample_MLE <- sample(gamma_MLE, n, replace = TRUE, prob = NULL)
  mean.param[i] <- mean(resample_MLE)
}

# Plot the sample means
hist(mean.param)

# Calculate the bias and the variance
bias.param <- mean(mean.param) - mean(gamma_MLE)
bias.param
var.param <- var(mean.param)
var.param


## Assignment 3.3

# Description of how random numbers are generated in parametric bootstrap
gengamma <- function(gendata, theta){
  rgamma(100, shape = theta[1], scale = theta[2])
}
```

```
# Generate a vector of means for bootstrapping
fun_param <- function(y = dataset, id) {mean(y[id])}

boot.out.param <- boot(dataset, fun_param, 2000, sim = "parametric", ran.
   gen = gengamma,
                       mle = fit$estimate, parallel="multicore", ncpus =
                          no_cores)
boot.out.param

plot(boot.out.param)
hist(boot.out.param$t)
abline(v=mean_theo,lwd=2, col="red")

# Calculate estimted bias and variance of estimator
boot.estimate.param <- mean(boot.out.param$t)
bias.param <- boot.estimate.param - orig.estimate
bias.param
var.param <- var(boot.out.param$t)
var.param


# Assignment 3.4

N <- 1000 # number of samples to draw
counter <- rep(0,10)
for (i in 1:N){
  for (j in 1:10){
    dataset.small <- rgamma(j*10, shape = k, scale = gamma)
    # re-estimate parameters
    param_MLE <- fit$estimate
    boot.out <- boot(dataset.small, fun, N, sim = "parametric",
                     ran.gen = gengamma, mle = param_MLE)
    ci <- boot.ci(boot.out, conf = 0.95, type = "norm")$norm
    if (ci[2] <= mean_theo && ci[3] >= mean_theo){
      counter[j] <- counter[j] + 1 # true value within limits
    }
  }
}

plot(counter,xaxt = "n",)
axis(1,at=seq(1,10),labels = seq(10,100,by=10))

#################################
## Assignment 4.Studentized CI ##
#################################


## BASIC CONFIDENCE INTERVAL

fun_var <- function(y, id) {var(y[id])}
```

```r
N <- 1000 # number of samples to draw
counter <- rep(0,10)
for (i in 1:N){
  for (j in 1:10){
    dataset.small <- rgamma(j*10, shape = k, scale = gamma)
    boot.out <- boot(dataset.small, fun_var, N)
    ci <- boot.ci(boot.out, conf = 0.95, type = "basic")$basic
    if (ci[4] <= var_theo && ci[5] >= var_theo){
      counter[j] <- counter[j] + 1 # true value within limits
    }
  }
}

plot(counter,xaxt = "n",)
axis(1,at=seq(1,10),labels = seq(10,100,by=10))


## STUDENTIZED CONFIDENCE INTERVAL

fun_stud <- function(y, id) {
  boot.out <- boot(y, fun_var, 50,
                   parallel="multicore", ncpus = no_cores)
  return (c(var(y[id]), var(boot.out$t)))
}

start_time <- Sys.time() # Start measuring time
N <- 100 # number of samples to draw (small bc of computation time)
counter <- rep(0,10)
for (i in 1:N){
  for (j in 1:10){
    dataset.small <- rgamma(j*10, shape = k, scale = gamma)
    boot.out <- boot(dataset.small, fun_stud, 1000,
                     parallel="multicore", ncpus = no_cores)
    ci <- boot.ci(boot.out, conf = 0.95, type = "stud")$stud
    if (ci[4] <= var_theo && ci[5] >= var_theo){
      counter[j] <- counter[j] + 1 # true value within limits
    }
  }
}

plot(counter,xaxt = "n",)
axis(1,at=seq(1,10),labels = seq(10,100,by=10))


end_time <- Sys.time() # End of measured time
cpu_time <- end_time - start_time
cpu_time
```