The bootstrap

Here is a series of exercises on the non-parametric bootstrap. We will first empirically derive the probability that a given observation is part of a bootstrap sample.

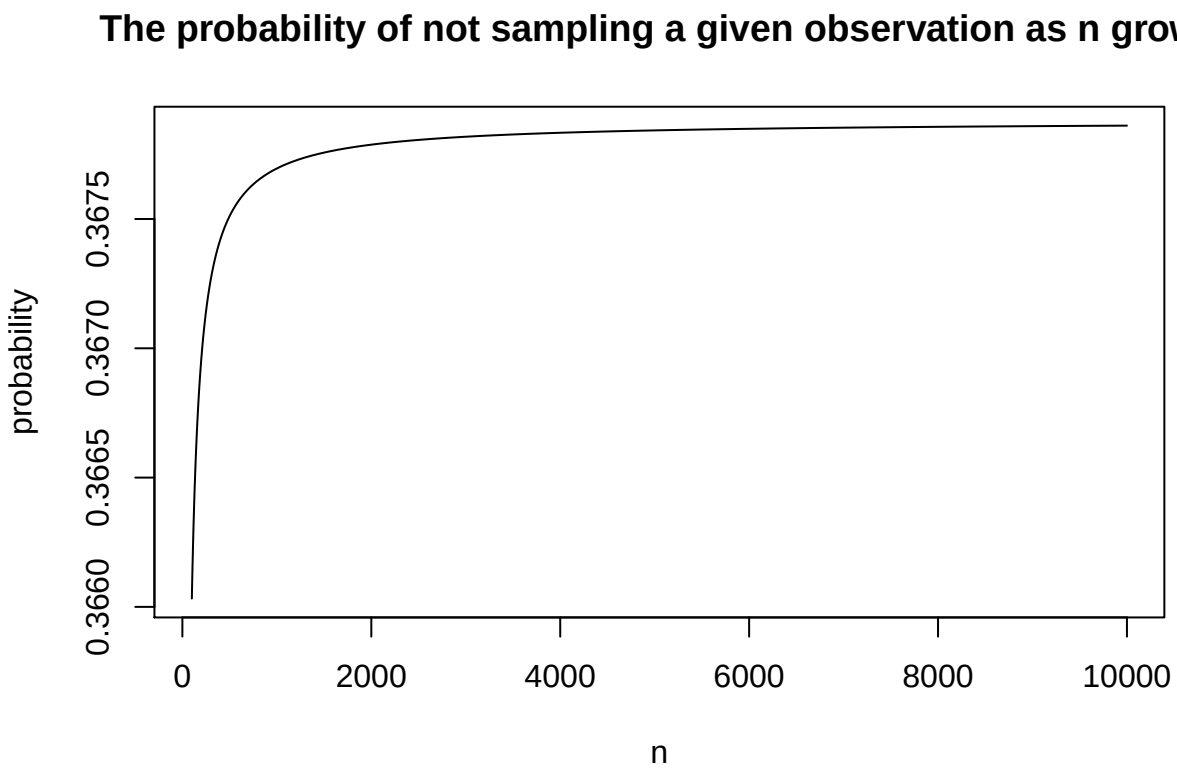
```
# sampling with replacement
p.not.in.sample <- function(n){
  return((1-(1/n))**n)
}
```

Example: what is the probability that a given observation is not in the sample, if it has size 100?

```
p.not.in.sample(100)
```

```
## [1] 0.3660323
```

```
# now let's simulate and plot
plot(100:10000, p.not.in.sample(100:10000), main="The probability of not sampling a given observation as
```



Looking at the above result we can answer the following question: what proportion of the original observations do you expect to be in a bootstrap sample of size n?

On average, weighting each observation as 1/n, we would have that the number of observations left out would be approximately equal to 1/3 (take the expectation of the indicator variable to see it). Hence,on average, we

would expect to have 2/3 of the original dataset in a bootstrap sample.

Empirical coverage of Bootstrap confidence intervals

We want to estimate the trimmed mean of the Gamma distribution where the 10% largest and 10% smallest observations are trimmed.

```
set.seed(0)
# approximate true parameter value with a huge sample
true.tm <-mean(rgamma(100000000, shape = 2, rate = 1), trim = 0.1)
true.tm
```

```
## [1] 1.820736
```

So true.tm is our ground truth. We're now going to draw a small sample from the true distribution to use it to do inference.

```
n<-40
small.sample <- rgamma(n,shape=2,rate=1)
# our sample estimate
hat.tm <- mean(small.sample, trim = 0.1)
hat.tm
```

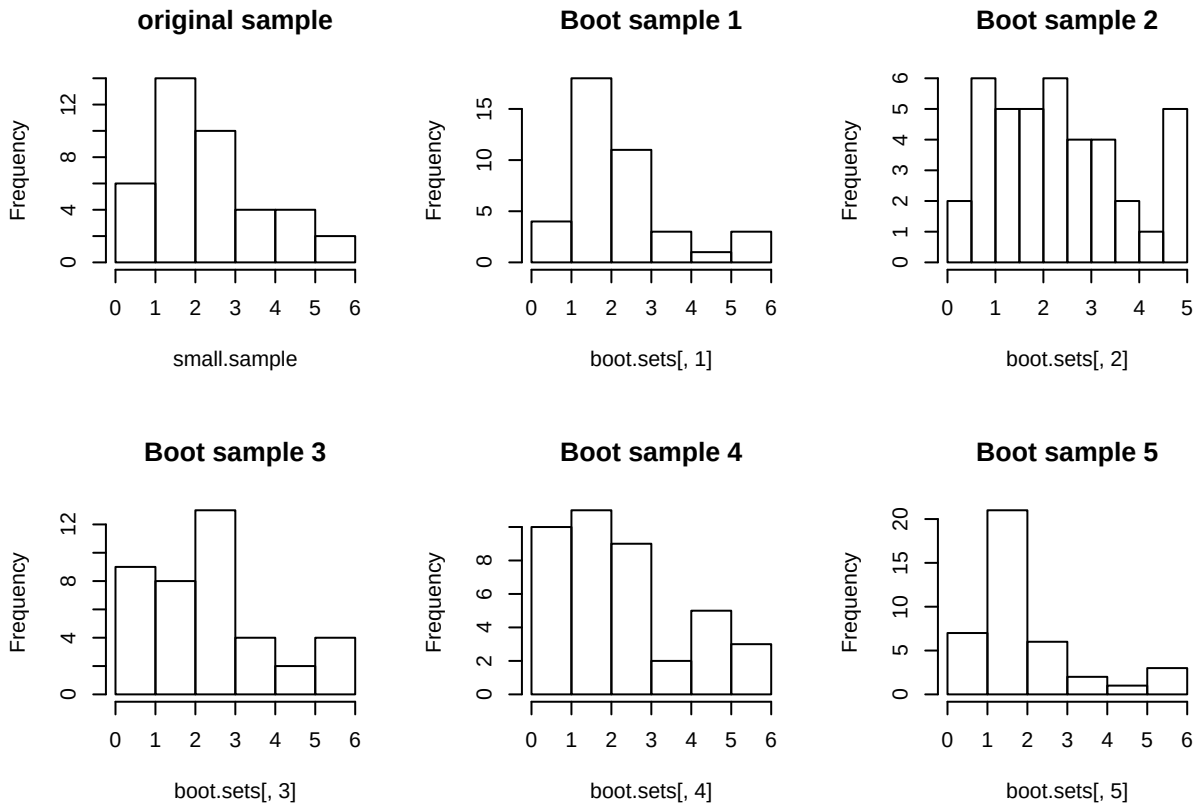
```
## [1] 2.171702
```

We'll now use Bootstrap to build 95% confidence intervals around our estimate.

```
# First of all, we need to create B bootstrap sets.
B <- 50
boot.sets <- matrix(nrow=n,ncol=B)
for(b in 1:B){
  b.set <- sample(small.sample,size=n, replace = T)
  boot.sets[,b]<- b.set
}
```

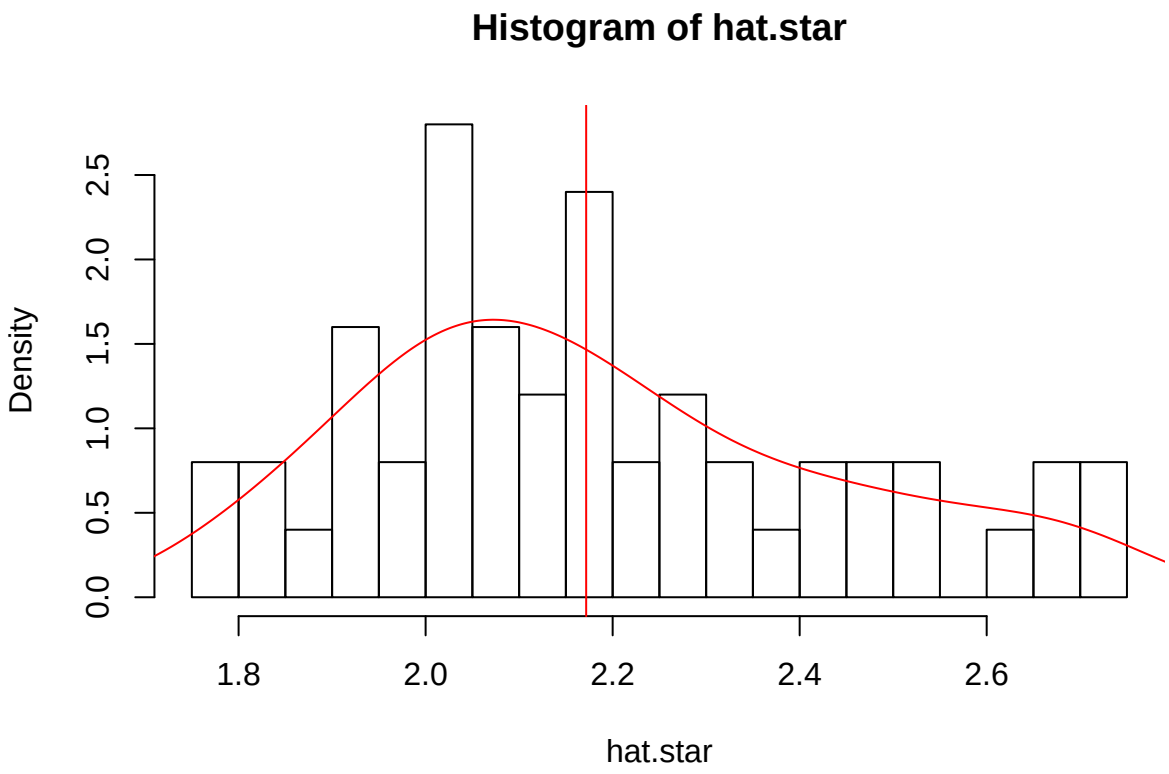
Let's visualize the first five datasets

```
par(mfrow=c(2,3))
hist(small.sample, main="original sample")
hist(boot.sets[,1], main="Boot sample 1")
hist(boot.sets[,2], main="Boot sample 2")
hist(boot.sets[,3], main="Boot sample 3")
hist(boot.sets[,4], main="Boot sample 4")
hist(boot.sets[,5], main="Boot sample 5")
```



Okay now we're ready to create the confidence intervals. We're going to create four different CIs.

```
hat.star <- matrix(nrow = B, ncol=1)
for(b in 1:B){
  hat.star[b]<-mean(boot.sets[,b],trim=0.1)
}
hist(hat.star, probability = T, breaks=20)
abline(v=hat.tm, col="red")
lines(density(hat.star), col="red")
```

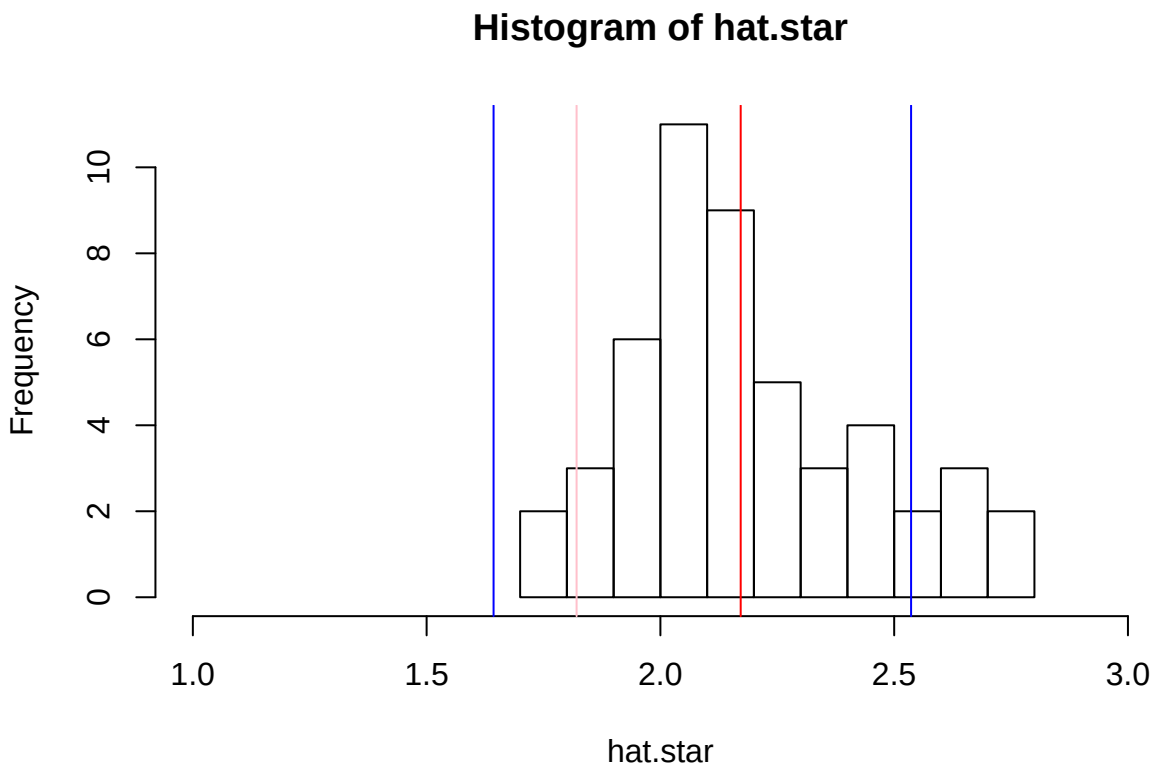


The reversed quantile CI

```
# find the empirical quantiles of the bootstrap distribution
p.star <- hat.star - hat.tm
lower.q <- quantile(p.star, probs = 0.025)
upper.q <- quantile(p.star, probs = 0.975)
lower <- hat.tm - upper.q
upper <- hat.tm - lower.q
c(lower,upper)
```

```
##    97.5%     2.5%
## 1.643181 2.536170
```

```
hist(hat.star, xlim = c(1,3))
abline(v=lower, col="blue")
abline(v=upper, col="blue")
abline(v=hat.tm, col="red")
abline(v=true.tm, col="pink")
```



Let's check our results against R results.

```
require("boot")
```

```
## Loading required package: boot
```

```
# statistic functions
tm.fun <- function(x, ind){return(mean(x[ind], trim=0.1))}
tm.var <- function(x, ind){
  tm.value <- tm.fun(x[ind])
  # second level bootstrap to obtain variance of our estimate
  tm.variance <- var(boot(data=x[ind],R=50,statistic=tm.fun)$t)
  return(c(tm.value,tm.variance))
}
```

```
boot.res <- boot(data=small.sample,statistic=tm.fun, R=50, sim="ordinary")
boot.ci(boot.res, conf=0.95, type="perc")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 50 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.res, conf = 0.95, type = "perc")
##
## Intervals :
## Level     Percentile
## 95%   ( 1.818,  2.714 )
## Calculations and Intervals on Original Scale
```

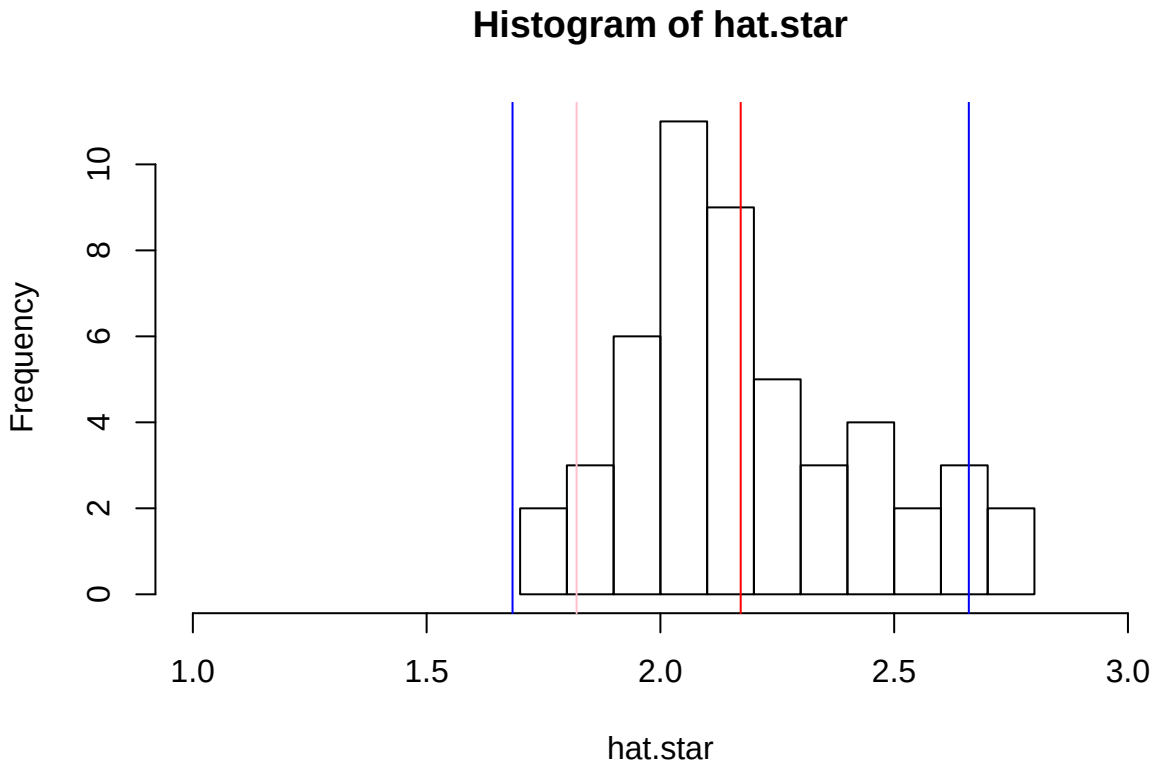
```
## Some percentile intervals may be unstable
```

The normal approximation CI

```
# this one is base on the assumption that the estimate distribution tends to a Gaussian as n grows
sd.hat.tm <- sqrt(var(hat.star))
lower.q <- qnorm(0.025)*sd.hat.tm
lower <- hat.tm + lower.q
upper <- hat.tm - lower.q
c(lower,upper)
```

```
## [1] 1.683751 2.659654
```

```
hist(hat.star, xlim = c(1,3))
abline(v=lower, col="blue")
abline(v=upper, col="blue")
abline(v=hat.tm, col="red")
abline(v=true.tm, col="pink")
```



Now let's check with the R results:

```
boot.ci(boot.res, conf=0.95, type="norm")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 50 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.res, conf = 0.95, type = "norm")
```

```
##
## Intervals :
## Level      Normal
## 95%   ( 1.742,  2.534 )
## Calculations and Intervals on Original Scale
```

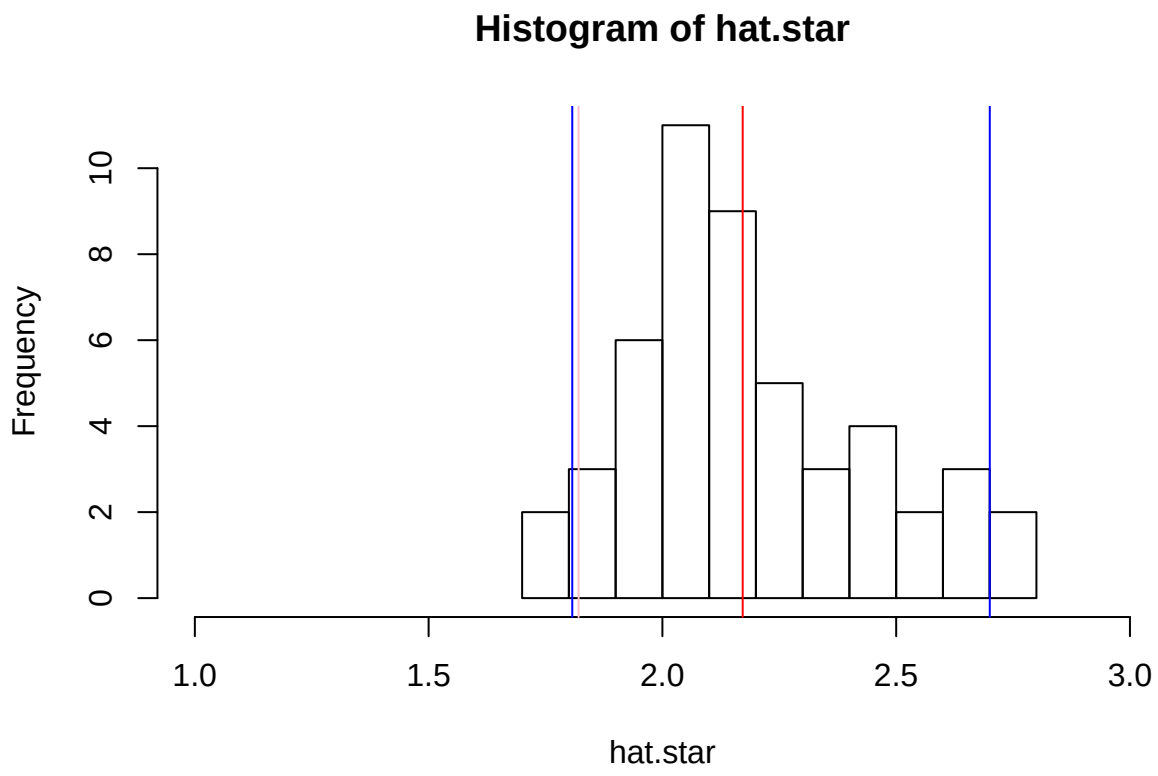
The naive CI

The naive CI directly uses the hat.star quantiles. Note: no theoretical justification unless it's distribution is symmetric.

```
lower.q <- quantile(hat.star, probs = 0.025)
upper.q <- quantile(hat.star, probs = 0.975)
lower <- lower.q
upper <- upper.q
c(lower,upper)
```

```
##     2.5%    97.5%
## 1.807235 2.700224
```

```
hist(hat.star, xlim = c(1,3))
abline(v=lower, col="blue")
abline(v=upper, col="blue")
abline(v=hat.tm, col="red")
abline(v=true.tm, col="pink")
```



Again let's check with the R results:

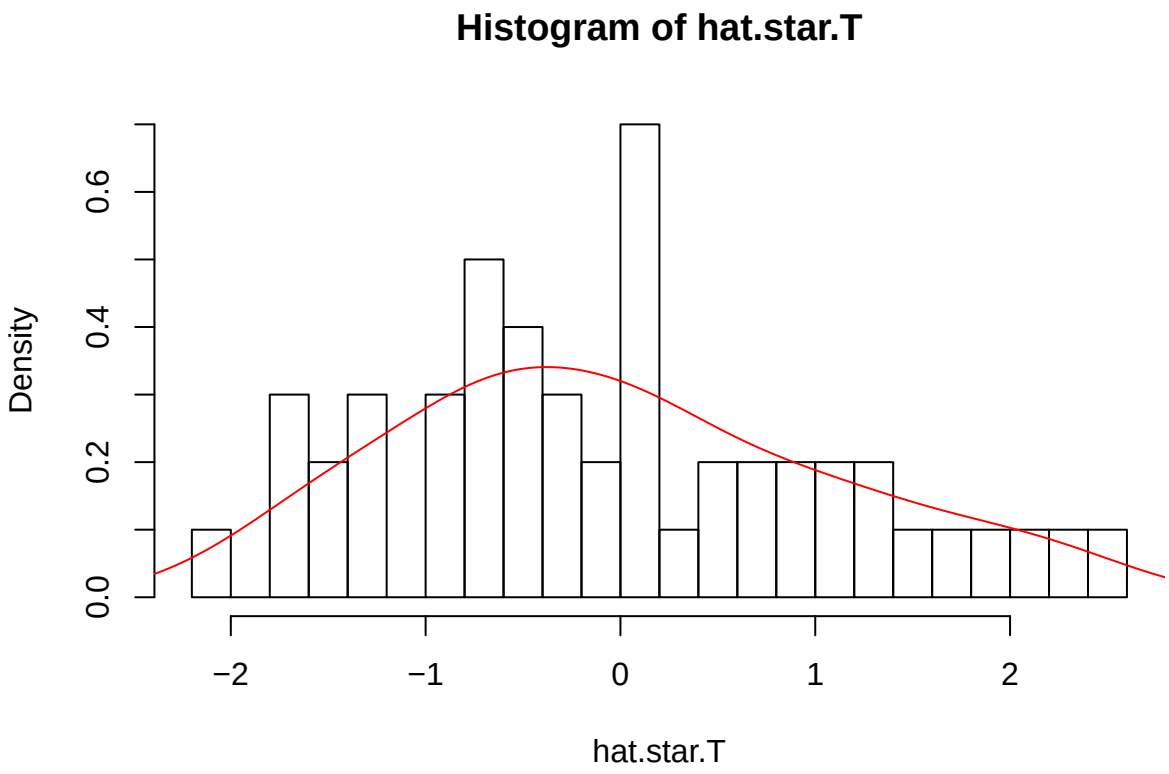
```r
boot.ci(boot.res, conf=0.95, type="basic")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 50 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.res, conf = 0.95, type = "basic")
##
## Intervals :
## Level      Basic
## 95%   ( 1.629,  2.525 )
## Calculations and Intervals on Original Scale
## Some basic intervals may be unstable
```

The bootstrap T CI

Now, to obtain the bootstrap T CI we need a second level bootstrap.

```r
C <- 50
hat.star.T <- matrix(nrow=B, ncol=1)
for(b in 1:B){
  hat.star.b<-matrix(nrow=C,ncol=1)
  for(c in 1:C){
    c.set <- sample(boot.sets[,b],size=n, replace = T)
    hat.star.b[c]<-mean(c.set,trim=0.1)
  }
  sd.b<-sqrt(var(hat.star.b))
  hat.star.T[b]<-(hat.star[b]-hat.tm)/sd.b
}
hist(hat.star.T, probability = T, breaks=20)
lines(density(hat.star.T), col="red")
```

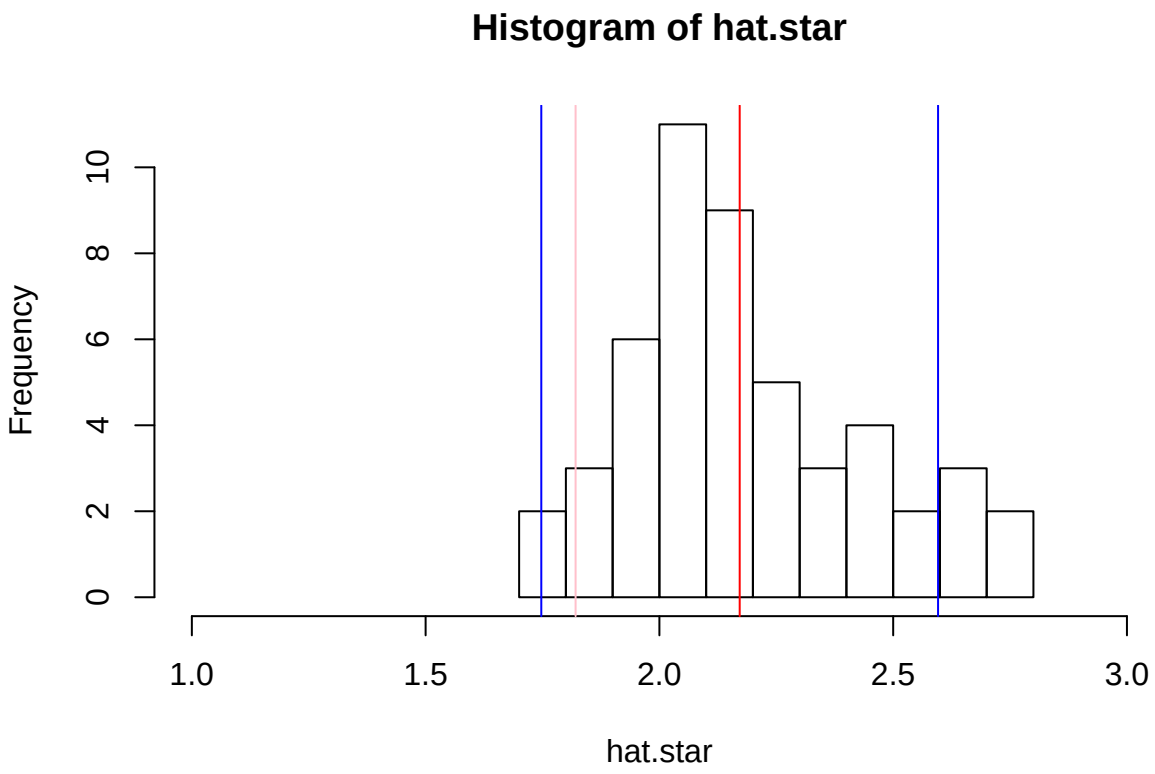


We'll now use the empirical quantiles of this simil-T distribution to estimate the CIs.

```
lower.q <- quantile(hat.star.T, probs = 0.025)*sd.hat.tm
upper.q <- quantile(hat.star.T, probs = 0.975)*sd.hat.tm
lower <- hat.tm + lower.q
upper <- hat.tm - lower.q
c(lower,upper)
```

```
## [1] 1.747416 2.595989
```

```
hist(hat.star, xlim = c(1,3))
abline(v=lower, col="blue")
abline(v=upper, col="blue")
abline(v=hat.tm, col="red")
abline(v=true.tm, col="pink")
```



And finally, let's check with the R results:

```
boot.res <- boot(data=small.sample,statistic=tm.var, R=50, sim="ordinary")
boot.ci(boot.res, conf=0.95, type="stud", index = c(1,2))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 50 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.res, conf = 0.95, type = "stud", index = c(1,
##     2))
##
## Intervals :
## Level    Studentized
## 95%   ( 1.807,  2.628 )
## Calculations and Intervals on Original Scale
## Some studentized intervals may be unstable
```

Simulations

Okay, as you have probably noticed, the bootstrap CI are variable. We'll now study their covarage exploiting this variability with a simulation.

```
nsim<-50
Bsim<-50
get.coverage <- function(ci){
  if(true.tm < ci[1]){return(c(1,0))}
  if(true.tm > ci[2]){return(c(0,1))}
```

```
  return(c(0,0))
}
```

```
coverages <- matrix(nrow=nsim,ncol=2*4) #each coverage has 2 columns
names.covrg<- names(coverages)<-c("norm.low","norm.high",
                    "basic.low","basic.high",
                    "percent.low","percent.high",
                    "student.low","student.high")
for(s in 1:nsim){
  new.sample <- rgamma(n,shape=2,rate=1)
  boot.res <- boot(data=new.sample,statistic=tm.var, R=Bsim,
                   sim="ordinary",parallel = "multicore", ncpus=20)
  b.ci<- boot.ci(boot.res, conf=0.95, type = c("basic", "norm", "perc", "stud"),
                 index = c(1,2))
  coverages[s,1:2] <- get.coverage(b.ci$normal[2:3])
  coverages[s,3:4] <- get.coverage(b.ci$basic[4:5])
  coverages[s,5:6] <- get.coverage(b.ci$percent[4:5])
  coverages[s,7:8] <- get.coverage(b.ci$student[4:5])
}

coverages<- data.frame(coverages)
names(coverages) <- names.covrg
```

Let's look at the results of the simulation.

```
#first: a check on the output
head(coverages)
```

```
##   norm.low norm.high basic.low basic.high percent.low percent.high student.low
## 1        0         0         0          0           0            0           0
## 2        0         0         0          0           0            0           0
## 3        0         0         1          0           0            0           0
## 4        0         0         0          1           0            0           0
## 5        0         0         0          0           0            0           0
## 6        0         0         0          0           0            0           0
##   student.high
## 1            0
## 2            0
## 3            0
## 4            0
## 5            0
## 6            0
```

```
coverages.frac <- colMeans(coverages)
coverages.frac
```

```
##     norm.low    norm.high    basic.low   basic.high  percent.low percent.high
##         0.02         0.04         0.02         0.08         0.04         0.04
##  student.low student.high
##         0.02         0.02
```

Let's now simulate for multiple sample sizes, with more simulations.

```
nsim <- 200
Bsim <- 200
ns <- c(10,50,100,500,1000)
coverages.ns <- matrix(nrow=length(ns),ncol=2*4) #each coverage has 2 columns
```

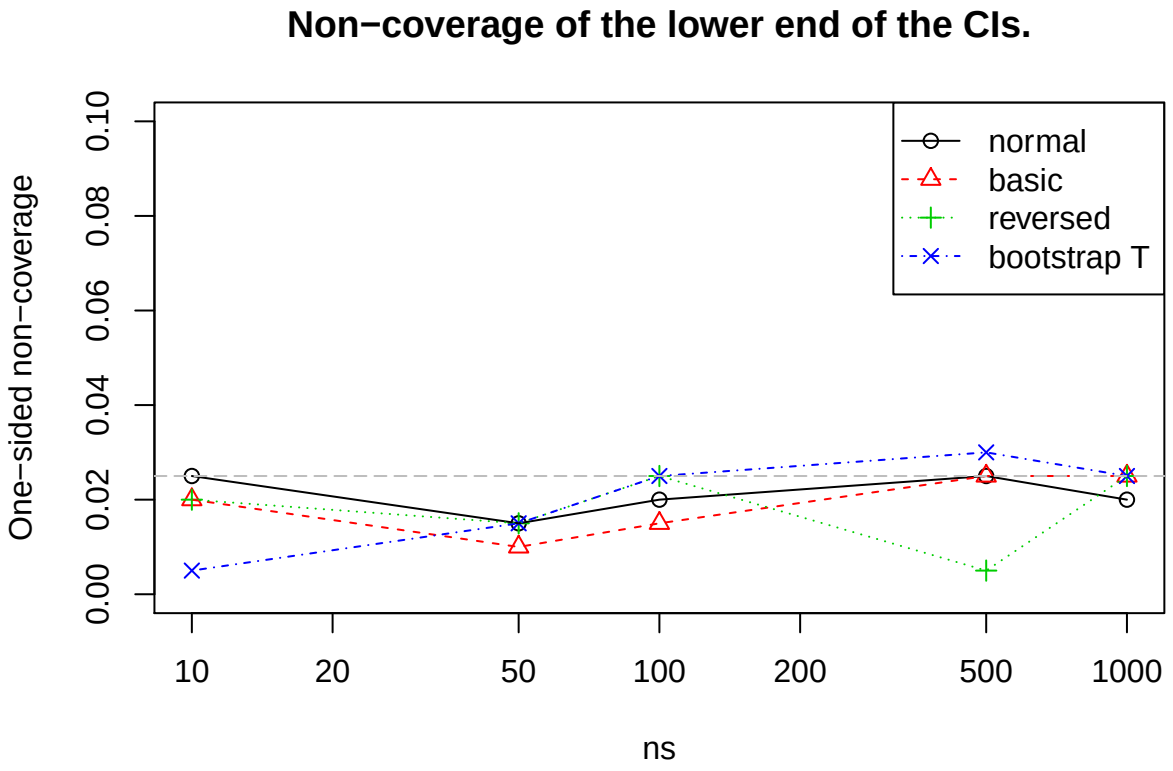
```
for(i in 1:length(ns)){
  n <- ns[i]
  coverages <- matrix(nrow=nsim,ncol=2*4) #each coverage has 2 columns
  for(s in 1:nsim){
    new.sample <- rgamma(n,shape=2,rate=1)
    boot.res <- boot(data=new.sample,statistic=tm.var, R=Bsim,
                     sim="ordinary",parallel = "multicore", ncpus=20)
    b.ci<- boot.ci(boot.res, conf=0.95, type = c("basic", "norm", "perc", "stud"),
                   index = c(1,2))
    coverages[s,1:2] <- get.coverage(b.ci$normal[2:3])
    coverages[s,3:4] <- get.coverage(b.ci$basic[4:5])
    coverages[s,5:6] <- get.coverage(b.ci$percent[4:5])
    coverages[s,7:8] <- get.coverage(b.ci$student[4:5])
  }
  coverages.ns[i,] <- colMeans(coverages)
}

coverages.ns<- data.frame(coverages.ns, row.names=ns)
names(coverages.ns) <- names.covrg
head(coverages.ns)
```

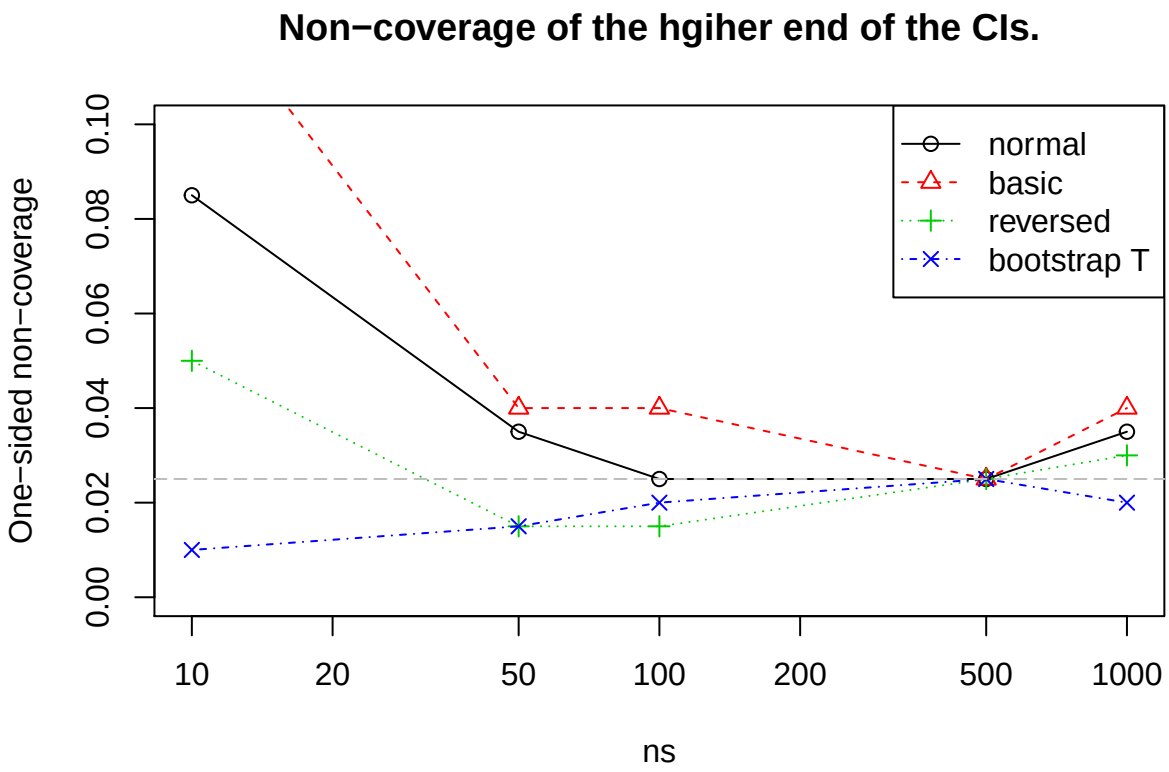
```
##      norm.low norm.high basic.low basic.high percent.low percent.high
## 10      0.025     0.085     0.020      0.130       0.020        0.050
## 50      0.015     0.035     0.010      0.040       0.015        0.015
## 100     0.020     0.025     0.015      0.040       0.025        0.015
## 500     0.025     0.025     0.025      0.025       0.005        0.025
## 1000    0.020     0.035     0.025      0.040       0.025        0.030
##      student.low student.high
## 10         0.005        0.010
## 50         0.015        0.015
## 100        0.025        0.020
## 500        0.030        0.025
## 1000       0.025        0.020
```

Let's visualize the results.

```
plot(norm.low~ ns, data = coverages.ns, col = 1, pch = 1, ylim = c(0, 0.1),
     log = "x", ylab = "One-sided non-coverage",
     main = "Non-coverage of the lower end of the CIs.")
points(basic.low ~ ns, data = coverages.ns, col = 2, pch = 2, xlog = TRUE)
points(percent.low ~ ns, data = coverages.ns, col = 3, pch = 3, xlog = TRUE)
points(student.low ~ ns, data = coverages.ns, col = 4, pch = 4, xlog = TRUE)
lines(norm.low ~ ns, data = coverages.ns, col = 1, lty = 1, xlog = TRUE)
lines(basic.low ~ ns, data = coverages.ns, col = 2, lty = 2, xlog = TRUE)
lines(percent.low ~ ns, data = coverages.ns, col = 3, lty = 3, xlog = TRUE)
lines(student.low ~ ns, data = coverages.ns, col = 4, lty = 4, xlog = TRUE)
abline(h = 0.025, lty = 5, col="gray")
legend("topright", legend = c("normal","basic", "reversed", "bootstrap T"),
       pch = 1:4, lty = 1:4, col = 1:4)
```




```
plot(norm.high~ ns, data = coverages.ns, col = 1, pch = 1, ylim = c(0, 0.1),
     log = "x", ylab = "One-sided non-coverage",
     main = "Non-coverage of the hgiher end of the CIs.")
points(basic.high ~ ns, data = coverages.ns, col = 2, pch = 2, xlog = TRUE)
points(percent.high ~ ns, data = coverages.ns, col = 3, pch = 3, xlog = TRUE)
points(student.high ~ ns, data = coverages.ns, col = 4, pch = 4, xlog = TRUE)
lines(norm.high ~ ns, data = coverages.ns, col = 1, lty = 1, xlog = TRUE)
lines(basic.high ~ ns, data = coverages.ns, col = 2, lty = 2, xlog = TRUE)
lines(percent.high ~ ns, data = coverages.ns, col = 3, lty = 3, xlog = TRUE)
lines(student.high ~ ns, data = coverages.ns, col = 4, lty = 4, xlog = TRUE)
abline(h = 0.025, lty = 5, col="gray")
legend("topright", legend = c("normal","basic", "reversed", "bootstrap T"),
       pch = 1:4, lty = 1:4, col = 1:4)
```



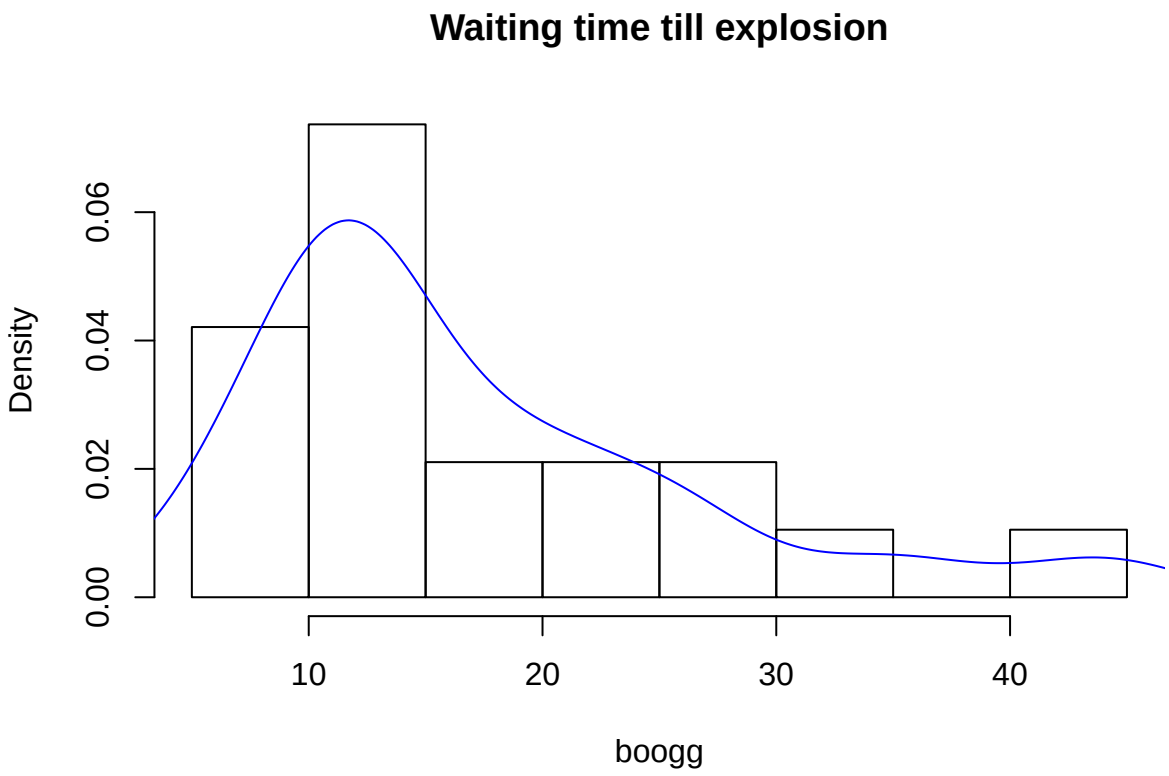
Notice the difference between the two tested sides, which could be due to asymmetry of the shape of the gamma distribution.

Parametric bootstrap

We'll now switch to parametric bootstrap techniques and analyse them in comparison with the non-parametric equivalent.

We're going to work with the Boogg data. The variable boogg which measures the waiting time till the head of the Boogg explodes on Sechsel¨auten in Zurich between the year 2000 and 2018.

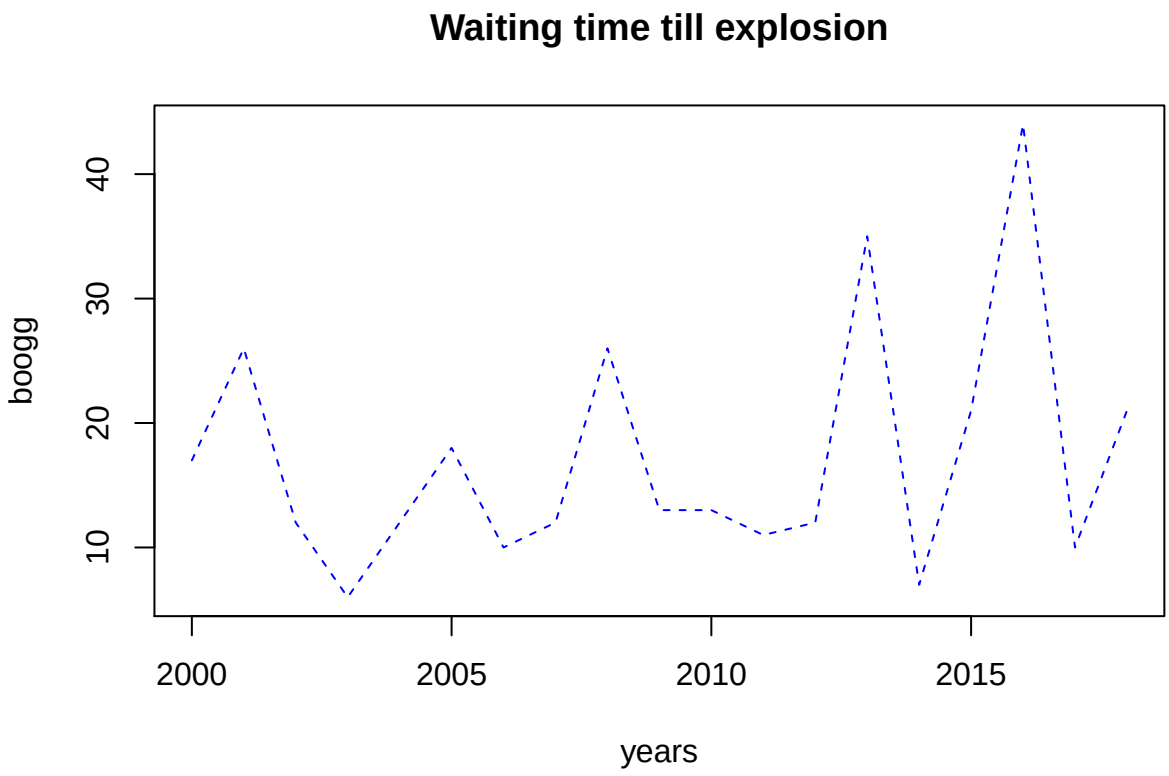
```
# The values are rounded to minutes (from 2000 to 2018).
years <- 2000:2018
boogg <- c(17, 26, 12, 6, 12, 18, 10, 12, 26, 13, 13, 11, 12, 35, 7, 21, 44, 10, 21)
hist(boogg, main = "Waiting time till explosion", probability = T, breaks=10)
lines(density(boogg), col="blue")
```




```
stem(boogg)
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   0 | 67
##   1 | 00122223378
##   2 | 1166
##   3 | 5
##   4 | 4
```

```
plot(years, boogg,  main = "Waiting time till explosion", lty="dashed", col="blue", type="l")
```



Let's assume that the data comes from a Gamma distribution, and let's use maximum likelihood to fit the parameter theta.

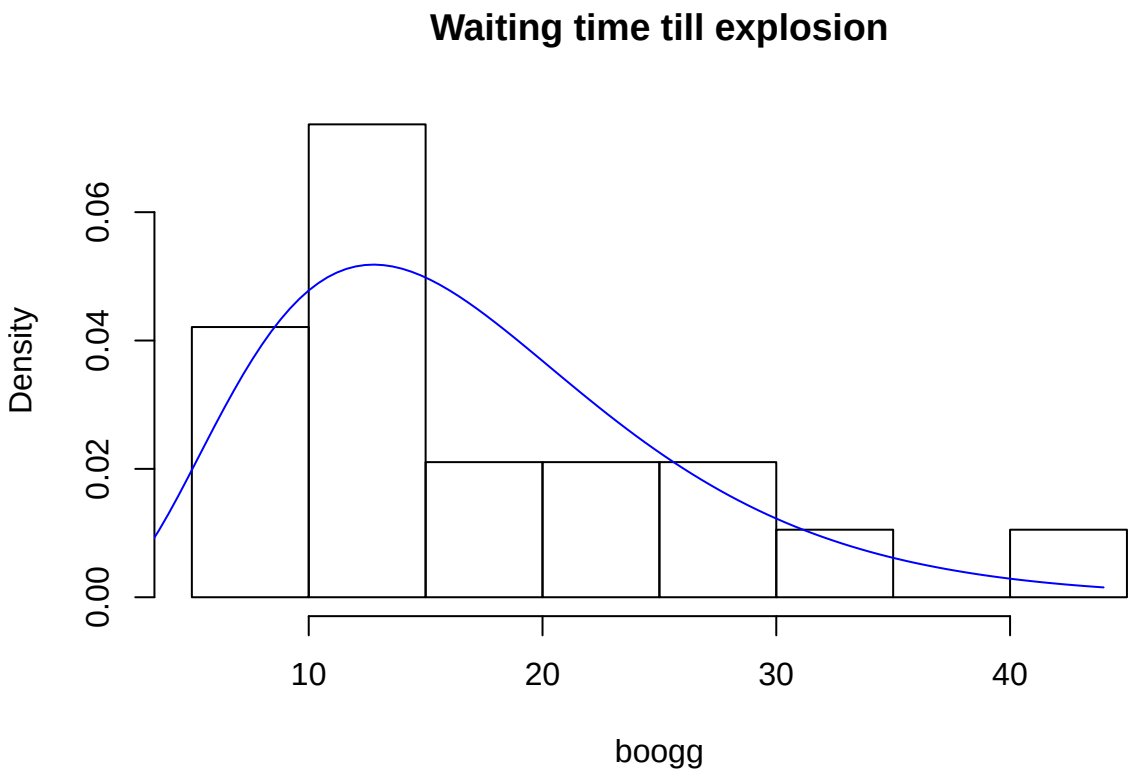
```
require(MASS)
```

```
## Loading required package: MASS
```

```
gamma.MLE <- fitdistr(boogg, densfun = "gamma")
gamma.MLE
```

```
##      shape         rate
##   3.91681011   0.22828203
##  (1.22046966) (0.07589377)
```

```
# let's plot the gamma density on our data
boogg.prob <- dgamma(x=seq(from=0, to = max(boogg),by = 0.4),
                     shape=gamma.MLE$estimate["shape"],
                     rate=gamma.MLE$estimate["rate"])
hist(boogg, main = "Waiting time till explosion", probability = T, breaks=10)
lines(x=seq(from=0, to = max(boogg),by = 0.4), y=boogg.prob, col="blue")
```



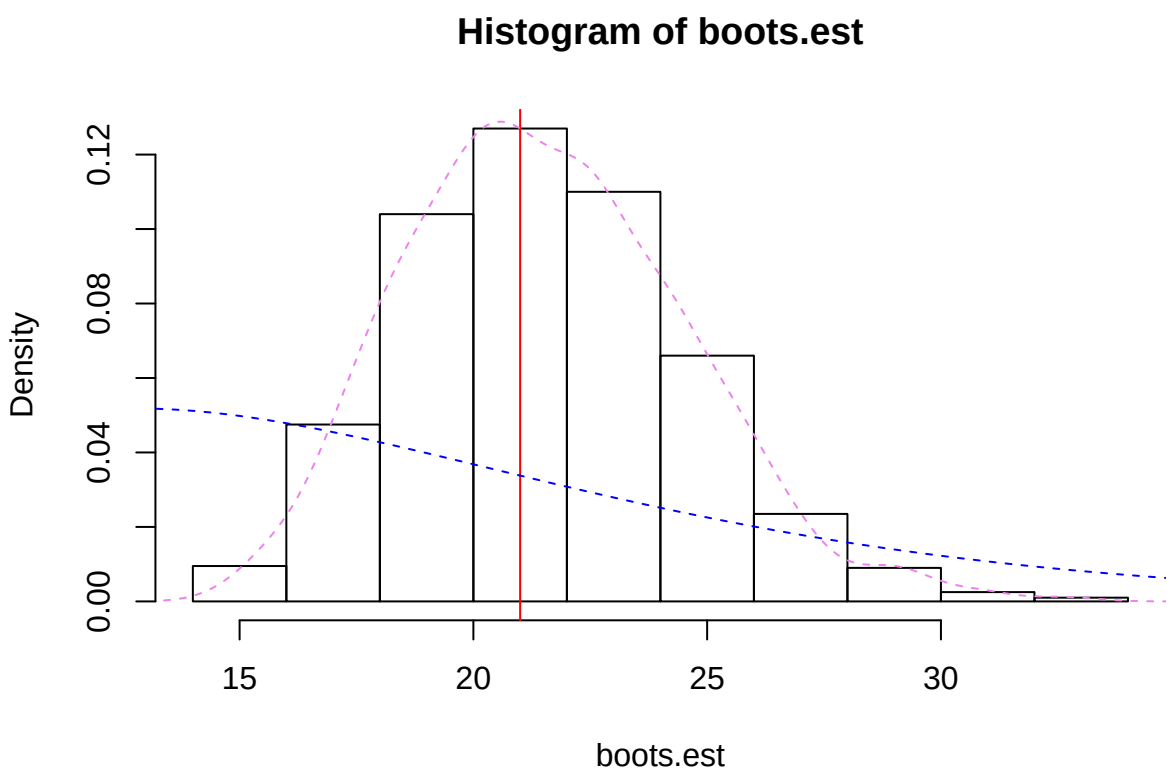
Let's now compute the quantity of interest and use bootstrap to evaluate its variance.

```
theta.hat <- quantile(boogg, probs = 0.75)
theta.hat
```

```
## 75%
##  21
```

```
B <- 1000
n <- length(boogg)
theta.fun <- function(data){
  # calculate the 75% quantile of the given data
  theta<-quantile(data, probs=0.75)
  return(theta)
}
generate.data <-function(n, shape, rate){
  data<-rgamma(n, shape, rate)
  return(data)
}
# we now have everything we need for our parametric bootstrap
boots.est <- matrix(nrow=B, ncol=1)
for(b in 1:B){
  data <- generate.data(n, shape=gamma.MLE$estimate["shape"],
                        rate=gamma.MLE$estimate["rate"])
  estimate <- theta.fun(data)
  boots.est[b]<-estimate
}
```

```r
hist(boots.est, probability = T)
abline(v=theta.hat, col="red")
lines(density(boots.est), col="violet", lty="dashed")
lines(x=seq(from=0, to = max(boogg),by = 0.4), y=boogg.prob, col="blue", lty="dashed")
```



Now let's use the result of the above bootstraping to construct confidence intervals for our estimate.

```r
#the quantile (or naive) confidence interval
naive.CI <- quantile(boots.est, probs =c(0.025,0.975))
naive.CI
```

```
##     2.5%    97.5%
## 16.48404 27.84308
```

```r
# the reversed quantile
lower <- theta.hat - quantile(boots.est-theta.hat, probs = 0.975)
upper <- theta.hat - quantile(boots.est-theta.hat, probs = 0.025)
reversed.quantile <- c(lower, upper)
reversed.quantile
```

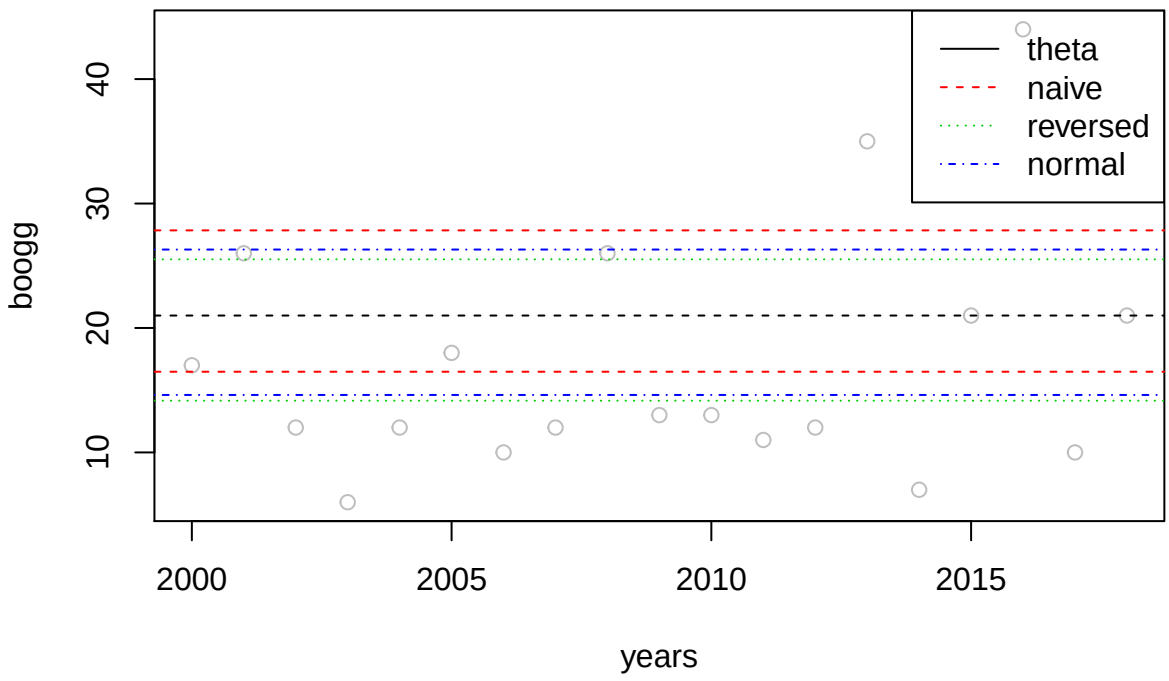
```
##      75%      75%
## 14.15692 25.51596
```

```r
# the normal quantile
sd.theta <- sqrt(var(boots.est))
lower <- (2*theta.hat - mean(boots.est)) - qnorm(0.975)*sd.theta
upper <- (2*theta.hat - mean(boots.est)) + qnorm(0.975)*sd.theta
normal.quantile <- c(lower, upper)
normal.quantile
```

```
## [1] 14.61687 26.30336
```

Let's now look at our results and check them with the output of the boot package.

```
plot(years,boogg, col="gray")
abline(h=theta.hat, col=1, lty="dashed")
abline(h=naive.CI[1], col=2, lty=2)
abline(h=naive.CI[2], col=2, lty=2)
abline(h=reversed.quantile[1], col=3, lty=3)
abline(h=reversed.quantile[2], col=3, lty=3)
abline(h=normal.quantile[1], col=4, lty=4)
abline(h=normal.quantile[2], col=4, lty=4)
legend("topright", legend = c("theta","naive", "reversed", "normal"),
       lty = 1:4, col = 1:4)
```



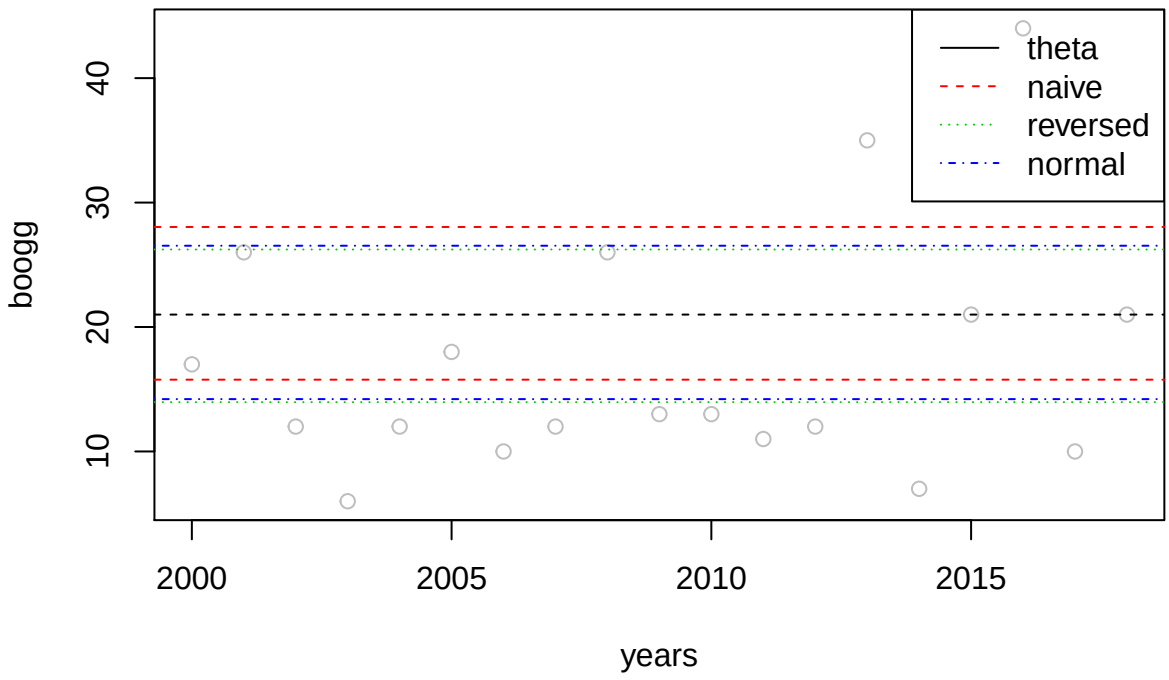
Note from the above plot how the naive is not equivalent to the reversed in this case, since the distribution of theta.boot - theta.hat is not symmetric around 0.

```
# wrapping previously used function to be compatible with boot package APIs
statistic <- function(data, indices){
  return(theta.fun(data[indices]))
}
random.generator <- function(data, parameters){
  n <- length(data)
  shape <- parameters[1]
  rate <- parameters[2]
  data<- generate.data(n, shape, rate)
```

```
  return(data)
}
require("boot")
boot.res <- boot(data=boogg,statistic=statistic, R = B, sim="parametric",ran.gen =
                  random.generator, mle = c(gamma.MLE$estimate["shape"],
                                            gamma.MLE$estimate["rate"]))
boot.cis <- boot.ci(boot.res, conf=0.95, type = c("basic","perc","norm"))
boot.cis
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.res, conf = 0.95, type = c("basic", "perc",
##     "norm"))
##
## Intervals :
## Level      Normal              Basic              Percentile
## 95%   (14.21, 26.53 )   (13.96, 26.23 )   (15.77, 28.04 )
## Calculations and Intervals on Original Scale
```

```
plot(years,boogg, col="gray")
abline(h=theta.hat, col=1, lty="dashed")
abline(h=boot.cis$percent[4], col=2, lty=2)
abline(h=boot.cis$percent[5], col=2, lty=2)
abline(h=boot.cis$basic[4], col=3, lty=3)
abline(h=boot.cis$basic[5], col=3, lty=3)
abline(h=boot.cis$normal[2], col=4, lty=4)
abline(h=boot.cis$normal[3], col=4, lty=4)
legend("topright", legend = c("theta","naive", "reversed", "normal"),
       lty = 1:4, col = 1:4)
```

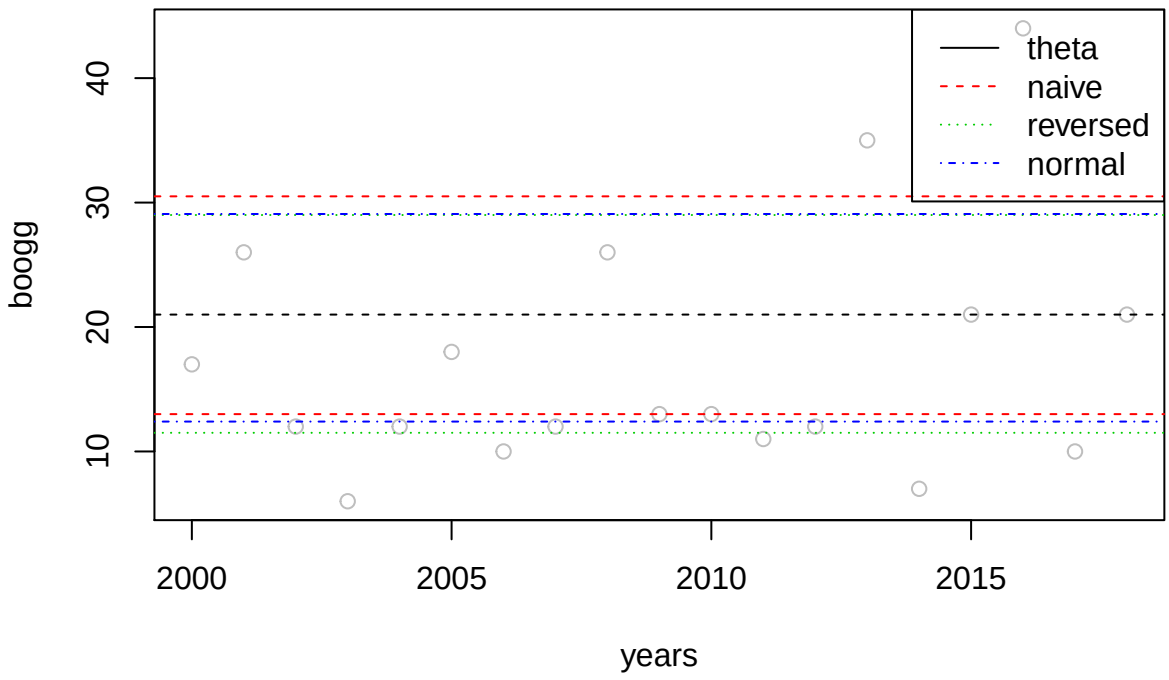


The results are very similar to the ones we computed by hand. What if we use a non-parametric approach?

```
boot.res <- boot(data=boogg,statistic=statistic, R = B)
boot.cis <- boot.ci(boot.res, conf=0.95, type = c("basic","perc","norm"))
boot.cis
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.res, conf = 0.95, type = c("basic", "perc",
##     "norm"))
##
## Intervals :
## Level      Normal              Basic              Percentile
## 95%   (12.40, 29.08 )   (11.50, 29.00 )   (13.00, 30.50 )
## Calculations and Intervals on Original Scale
```

```
plot(years,boogg, col="gray")
abline(h=theta.hat, col=1, lty="dashed")
abline(h=boot.cis$percent[4], col=2, lty=2)
abline(h=boot.cis$percent[5], col=2, lty=2)
abline(h=boot.cis$basic[4], col=3, lty=3)
abline(h=boot.cis$basic[5], col=3, lty=3)
abline(h=boot.cis$normal[2], col=4, lty=4)
abline(h=boot.cis$normal[3], col=4, lty=4)
legend("topright", legend = c("theta","naive", "reversed", "normal"),
       lty = 1:4, col = 1:4)
```



The confidence intervals considerably widen up in the non-parametric case. Note that if the gamma assumption is wrong, the parametric approach is not computing the right estimates.

Bootstrap for regression

We'll now use Bootstrap to build standard error estimates for the least squares estimator. We'll work on the Boston dataset and try to estimate the variable crim from the others.

```
require(MASS)
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

```
fit <- lm(crim~., data=Boston)
summary(fit)
```

```
##
## Call:
## lm(formula = crim ~ ., data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -9.924 -2.120 -0.353  1.019 75.051
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  17.033228   7.234903   2.354 0.018949 *
## zn            0.044855   0.018734   2.394 0.017025 *
## indus        -0.063855   0.083407  -0.766 0.444294
## chas         -0.749134   1.180147  -0.635 0.525867
## nox         -10.313535   5.275536  -1.955 0.051152 .
## rm            0.430131   0.612830   0.702 0.483089
## age           0.001452   0.017925   0.081 0.935488
## dis          -0.987176   0.281817  -3.503 0.000502 ***
## rad           0.588209   0.088049   6.680 6.46e-11 ***
## tax          -0.003780   0.005156  -0.733 0.463793
## ptratio      -0.271081   0.186450  -1.454 0.146611
## black        -0.007538   0.003673  -2.052 0.040702 *
## lstat         0.126211   0.075725   1.667 0.096208 .
## medv         -0.198887   0.060516  -3.287 0.001087 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.439 on 492 degrees of freedom
## Multiple R-squared:  0.454,  Adjusted R-squared:  0.4396
## F-statistic: 31.47 on 13 and 492 DF,  p-value: < 2.2e-16
```

```
boot.fun<-function(data,index){
  fit <- lm(crim~., data=Boston, subset = index)
  return(coefficients(fit))
}
```

```
B = 1000
boot.res <- boot(Boston, boot.fun, R=B)
boot.res
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston, statistic = boot.fun, R = B)
##
##
## Bootstrap Statistics :
##           original        bias    std. error
## t1*   17.033227523 -2.486674e-01 8.251326470
## t2*    0.044855215 -4.075068e-04 0.010265916
## t3*   -0.063854824  4.149376e-04 0.038906120
## t4*   -0.749133611  1.235256e-02 0.543523698
## t5*  -10.313534912  4.032615e-02 4.066176641
```

```
## t6*    0.430130506  3.937616e-02 1.071374316
## t7*    0.001451643 -6.845418e-04 0.013020378
## t8*   -0.987175726 -3.578959e-03 0.257762316
## t9*    0.588208591 -4.449790e-03 0.071574280
## t10*  -0.003780016  9.837212e-05 0.001968108
## t11*  -0.271080558  2.356978e-04 0.098989319
## t12*  -0.007537505  2.408705e-04 0.008690765
## t13*   0.126211376 -1.005198e-04 0.090226926
## t14*  -0.198886821 -2.739708e-03 0.110499571
```

Compare the standard error calculated by bootstrap with that calculated by the lm function: if there are any differences those are probably due to the linearity and Gaussianity assumptions made by the lm model. Using non-parametric bootstrap we avoid falling in the trap of un-realistic assumptions and always get reliable (and asymptotically right) estimates for the standard deviation. Note that to get an estimate for the standard deviation of each coefficient estimate using lm we need homoskedasticity and indipendence assumptions to hold, while to estimate the coefficient itself we don't need homoskedasticity (while we still need linearity and indipendence).