Bootstrap methods Part 1

Bootstrap: Example of the effects of aspirin

Classical approach

```
n11 = 119
n21 = 98

n12 = 11037-119
n22 = 11034-98

teta = (n11*n22)/(n12*n21)

SE = sqrt(1/n11 + 1/n12 + 1/n21 + 1/n22)
LSup = log(teta) + qnorm(1-0.05/2)*SE
LInf = log(teta) - qnorm(1-0.05/2)*SE

exp(LInf)
```

```
[1] 0.929726
```

```
exp(LSup)
```

```
[1] 1.591174
```

Bootstrap approach

Original data

```
n1 = 11037       # Sample size 1
s1 = 119         # Number of sucesses

n2 = 11034      # Sample size 2
s2 = 98         # Number of sucesses
```

However, we do not have the *original* data file, only the table with **counts**.

We then recreate the original file:

```
p1pre = c(rep(1,s1), rep(0,n1-s1))
p2pre = c(rep(1,s2), rep(0,n2-s2))

# We take permutations of the above data
p1 = sample(p1pre, n1)   # Sample 1
p2 = sample(p2pre ,n2)   # Sample 2
```

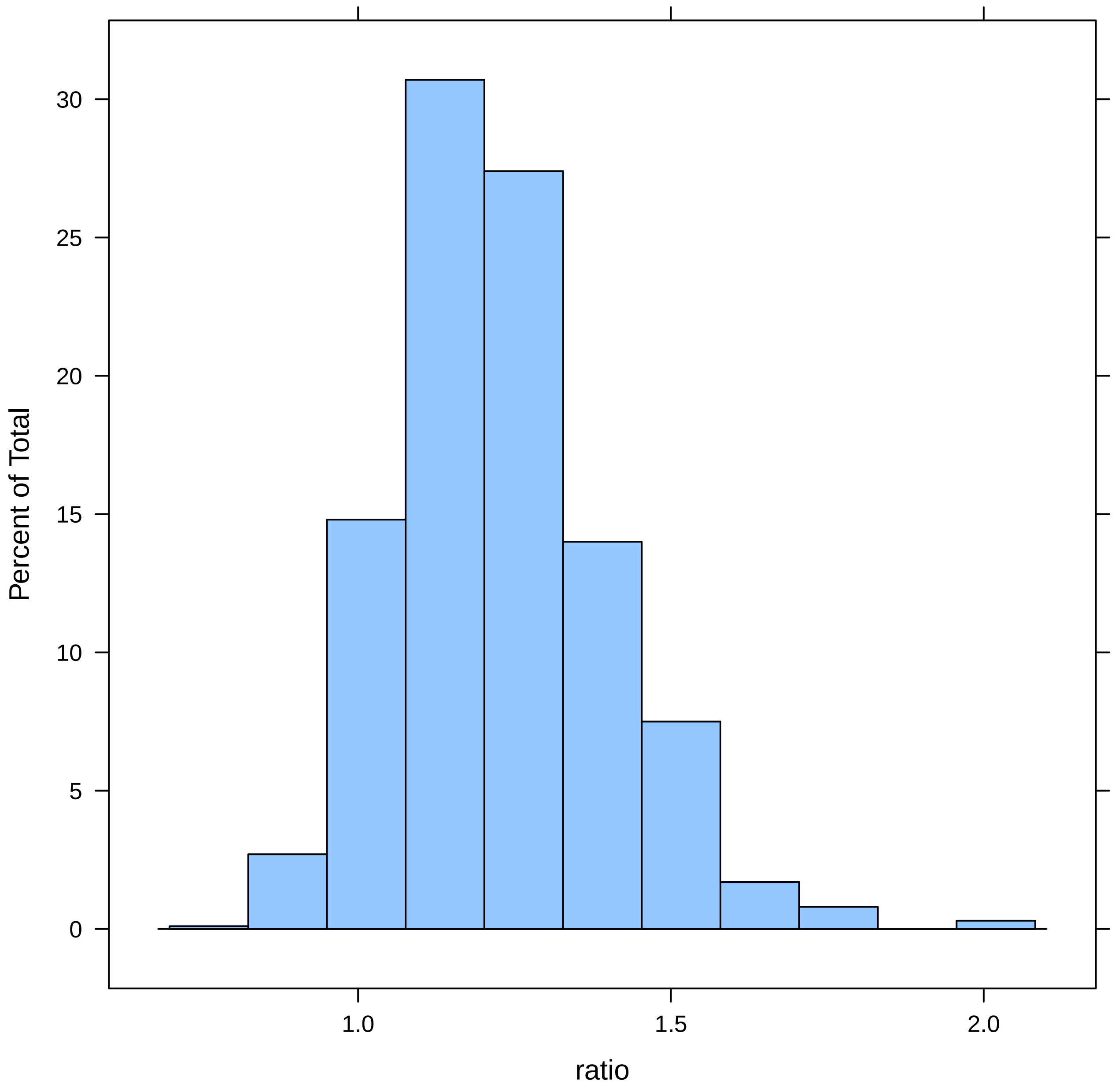
That is, we assume that $p1$ and $p2$ are the original observed data.

A bootstrap resampling method is applied.

```
n.bs = 1000      # Take n.bs bootstrap samples

bs1 = rep(0, n.bs)
bs2 = rep(0, n.bs)

for (i in 1:n.bs) {
   # Proportion of successes in bootstrap samples 1 and 2
   bs1[i] = sum(sample(p1, n1, replace=TRUE))/n1
   bs2[i] = sum(sample(p2, n2, replace=TRUE))/n2
}

# Replications of the bootstrap estimation of the ratio
ratio = bs1/bs2
```

Histogram of ratio estimates:

```
lattice::histogram(ratio)
```



```r
mean(ratio)
```

```
[1] 1.229073
```

```r
median(ratio)
```

```
[1] 1.210996
```

The bootstrap confidence interval corresponds to the 0.025 and 0.975 quantiles of the ordered sample.

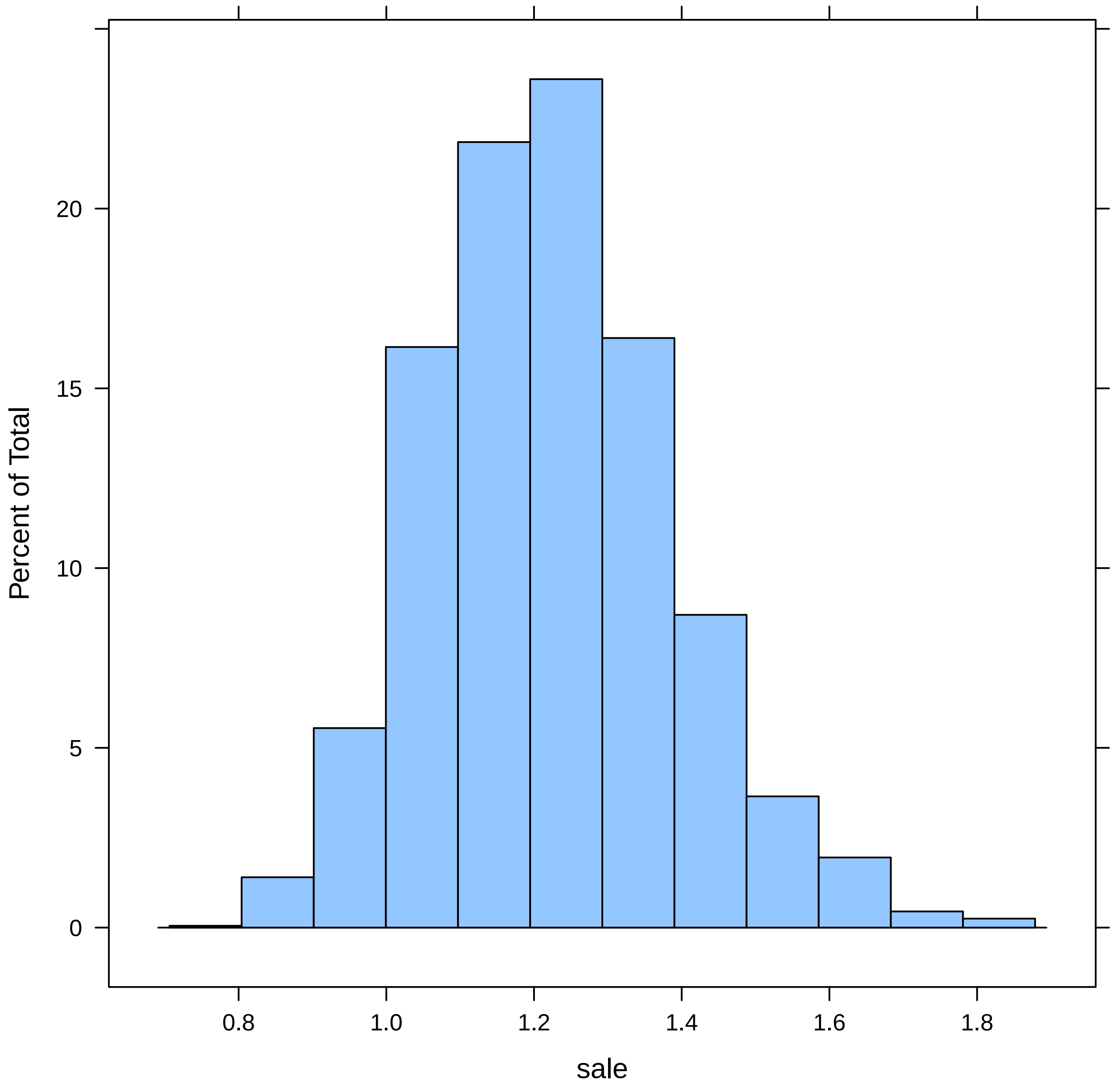
```r
quantile(ratio, probs=c(0.025, 0.975))
```

```
     2.5%     97.5% 
0.9441061 1.5923448 
```

In Rcpp:

```r
n1 = 11037       # Sample size 1
s1 = 119         # Number of sucesses

n2 = 11034      # Sample size 2
s2 = 98         # Number of sucesses

p1pre = c(rep(1,s1), rep(0,n1-s1))
p2pre = c(rep(1,s2), rep(0,n2-s2))

p1 = sample(p1pre, n1)
p2 = sample(p2pre ,n2)

library(Rcpp)

sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp ;
// [[Rcpp::export]]

NumericVector boot_ratio2prop(NumericVector p1,
NumericVector p2, int replicas=1000) {

    int n1 = p1.size();
    int n2 = p2.size();

    NumericVector bs1(replicas);
    NumericVector bs2(replicas);
    NumericVector ratio(replicas);
    bool replace = true;

    for(int i=0; i<replicas; i++) {
       bs1[i] = sum(sample(p1, n1, replace))/n1;
       bs2[i] = sum(sample(p2, n2, replace))/n2;
       ratio[i] = bs1[i]/bs2[i];
    }

return ratio;
}
'
)
```

```
replica = 2000

sale = boot_ratio2prop(p1, p2, replica)

lattice::histogram(sale)
```



Alternative using dplyr

```r
library(dplyr)
library(purrr)
library(tibble)

# Create the dataset
trial_data = tibble(
  patient = 1:22071,
  group = ifelse(patient <= 11037, "aspirin", "control"),
  heartAttack = c(
    rep(TRUE, 119), rep(FALSE, 10918),
    rep(TRUE, 98), rep(FALSE, 10936)
  )
)
```

```r
# Calculate summary statistics for each group
summary_stats = trial_data %>%
  group_by(group) %>%
  summarise(
    num_attacks = sum(heartAttack),
    num_people = n(),
    attack_rate = (num_attacks / num_people) * 100,
    .groups = "drop"
  )
print(summary_stats)
```

```
# A tibble: 2 x 4
  group   num_attacks num_people attack_rate
  <chr>         <int>      <int>       <dbl>
1 aspirin         119      11037       1.08
2 control          98      11034       0.888
```

```r
# Compute the ratio of attack rates (aspirin / control)
rate_ratio = summary_stats %>%
  summarise(ratio = attack_rate[group == "aspirin"] /
                    attack_rate[group == "control"]) %>%
  pull(ratio)
print(rate_ratio)
```

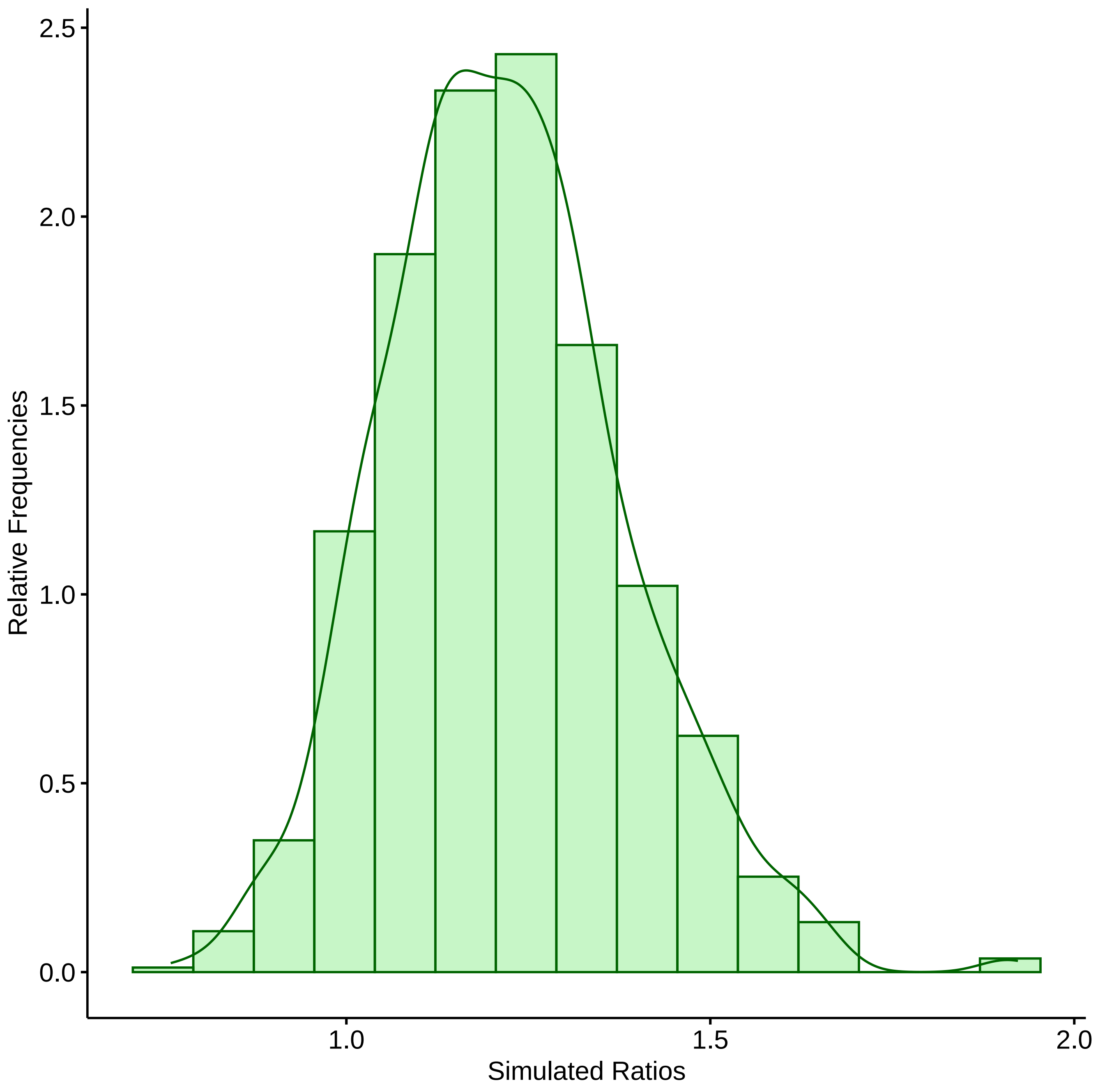
```
[1] 1.213956
```

```r
# Define a bootstrap function that resamples within each group
bootstrap_rate_ratio = function() {
  # Bootstrap sample: resample each group with replacement
  boot_sample = trial_data %>%
    group_by(group) %>%
    sample_frac(replace = TRUE) %>%
    summarise(rate = mean(heartAttack), .groups = "drop")

  # Calculate the ratio explicitly using group names
  ratio = boot_sample %>%
    filter(group == "aspirin") %>%
    pull(rate) /
    boot_sample %>%
    filter(group == "control") %>%
    pull(rate)

  ratio
}
```

```
# Run the bootstrap 1000 times and calculate the standard error
bootstrap_ratios = map_dbl(1:1000, ~ bootstrap_rate_ratio())
se = sd(bootstrap_ratios)
cat("Standard error: ", se, "\n")
```

```
Standard error:  0.1651527
```

```
library(ggpubr)
gghistogram(bootstrap_ratios, y="..density..",
ylab="Relative Frequencies", xlab="Simulated Ratios",
bins = 15, fill = "lightgreen", color="darkgreen", add_density = TRUE)
```


Relative Frequencies
0.0
0.5
1.0
1.5
2.0
2.5
1.0
1.5
2.0
Simulated Ratios

Example of the effect of a surgical treatment on mice

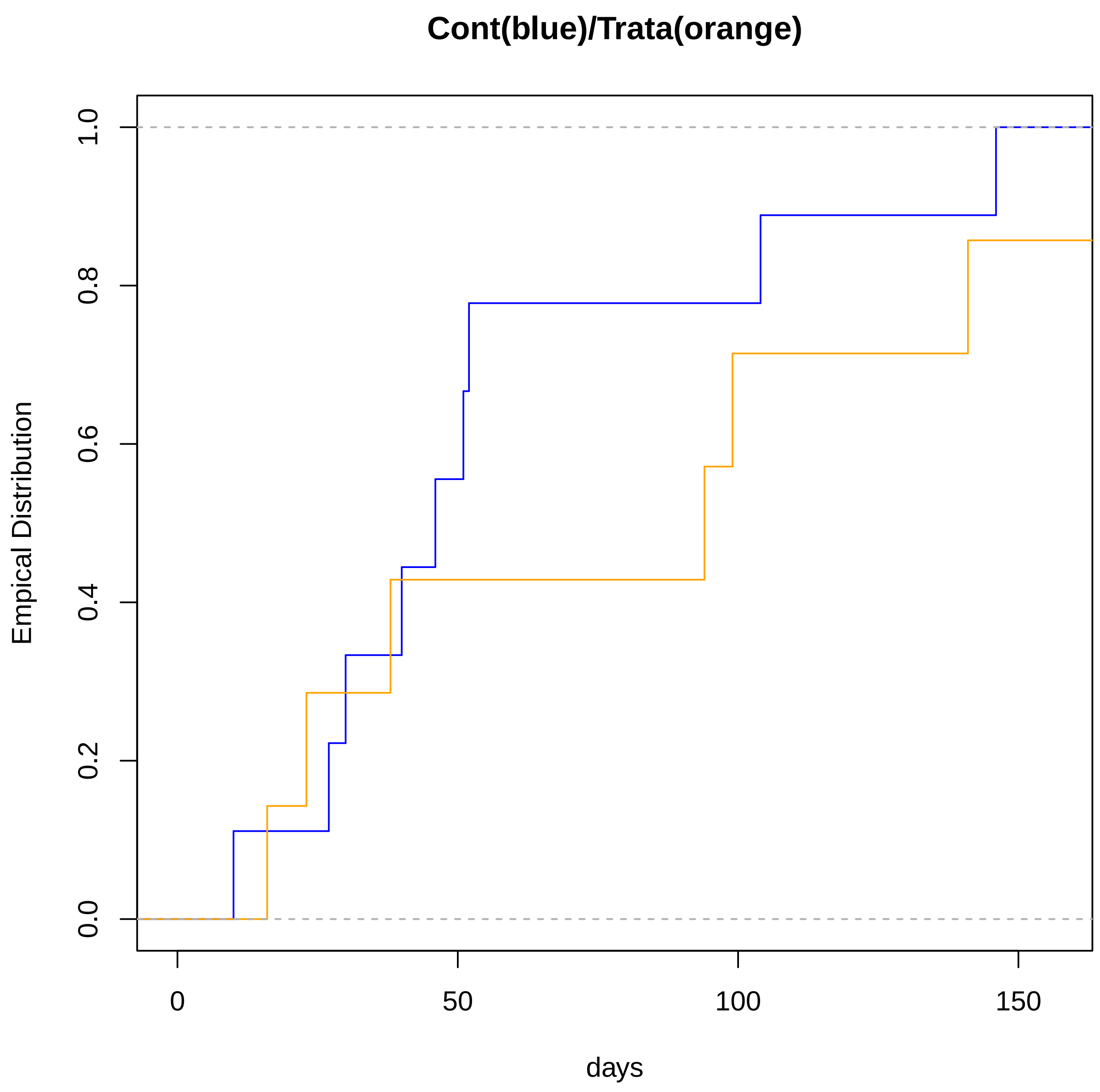
```
Trata = c(94, 197, 16, 38, 99, 141, 23)
Cont = c(52, 104, 146, 10, 51, 30, 40, 27, 46)

# Stem-and-leaf graph of treatment group
stem(Trata,scale=2)
```

```
  The decimal point is 1 digit(s) to the right of the |

   0 | 6
   2 | 38
   4 |
   6 |
   8 | 49
  10 |
  12 |
  14 | 1
  16 |
  18 | 7
```

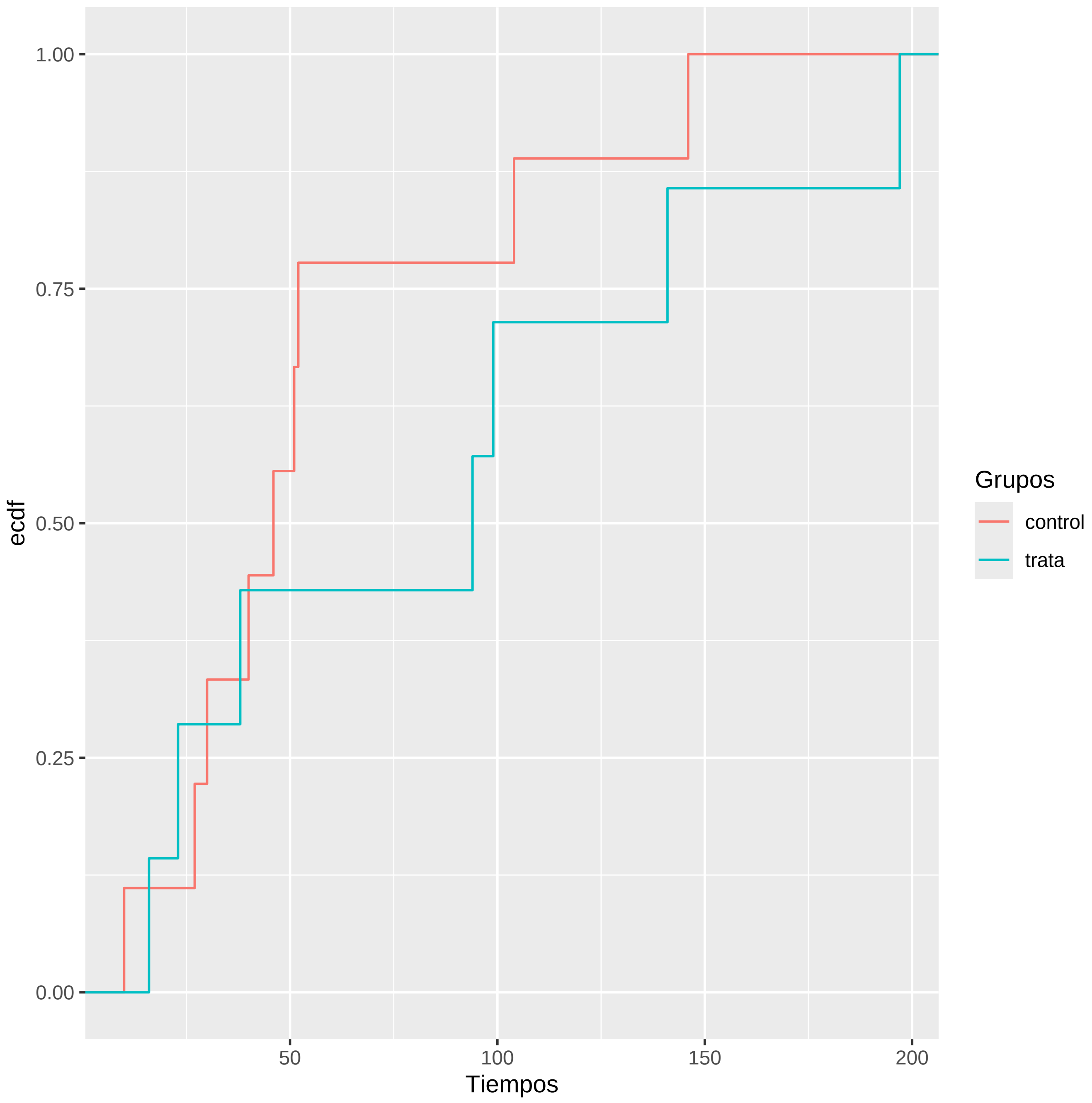
```
# Stem-and-leaf plot of control group
stem(Cont,scale=2)
```

```
  The decimal point is 1 digit(s) to the right of the |

   0 | 0
   2 | 70
   4 | 0612
   6 |
   8 |
  10 | 4
  12 |
  14 | 6
```

```
ecdf1 = ecdf(Trata)
ecdf2 = ecdf(Cont)

plot(ecdf2, verticals=TRUE, do.points=FALSE, col='blue', xlab="days",
ylab="Empical Distribution",
main="Cont(blue)/Trata(orange)" )

plot(ecdf1, verticals=TRUE, do.points=FALSE, add=TRUE, col='orange')
```



```r
library(ggplot2)
Tiempos = c(Trata, Cont)
Grupos = as.factor(c(rep("trata",length(Trata)),
rep("control",length(Cont))))

Losdatos = data.frame(Tiempos, Grupos)

ggplot(Losdatos, aes(Tiempos, colour = Grupos)) +
stat_ecdf()
```



As *Efron y Tibshirani* write:

```
mean(Trata)
```

```
[1] 86.85714
```

```r
mean(Cont)
```

```
[1] 56.22222
```

```r
(sdDiff = sqrt(var(Trata)/length(Trata)+var(Cont)/length(Cont)))
```

```
[1] 28.93607
```

We can apply the **Central Limit Theorem**.

```r
(t = (mean(Trata) - mean(Cont)) / sdDiff)
```

```
[1] 1.058711
```

We put the values in a single vector and define another vector of 1's and 2's according to their group membership:

```r
x = matrix(c(Trata, Cont, rep(1,length(Trata)), rep(2, length(Cont))), ncol=2)

# Student t-test
t.test(x[,1] ~ x[,2])
```

```

    Welch Two Sample t-test

data:  x[, 1] by x[, 2]
t = 1.0587, df = 9.6545, p-value = 0.3155
alternative hypothesis: true difference in means between group 1 and group 2 is not equal to 0
95 percent confidence interval:
 -34.15279  95.42263
sample estimates:
mean in group 1 mean in group 2 
       86.85714        56.22222 
```

Using Rcpp

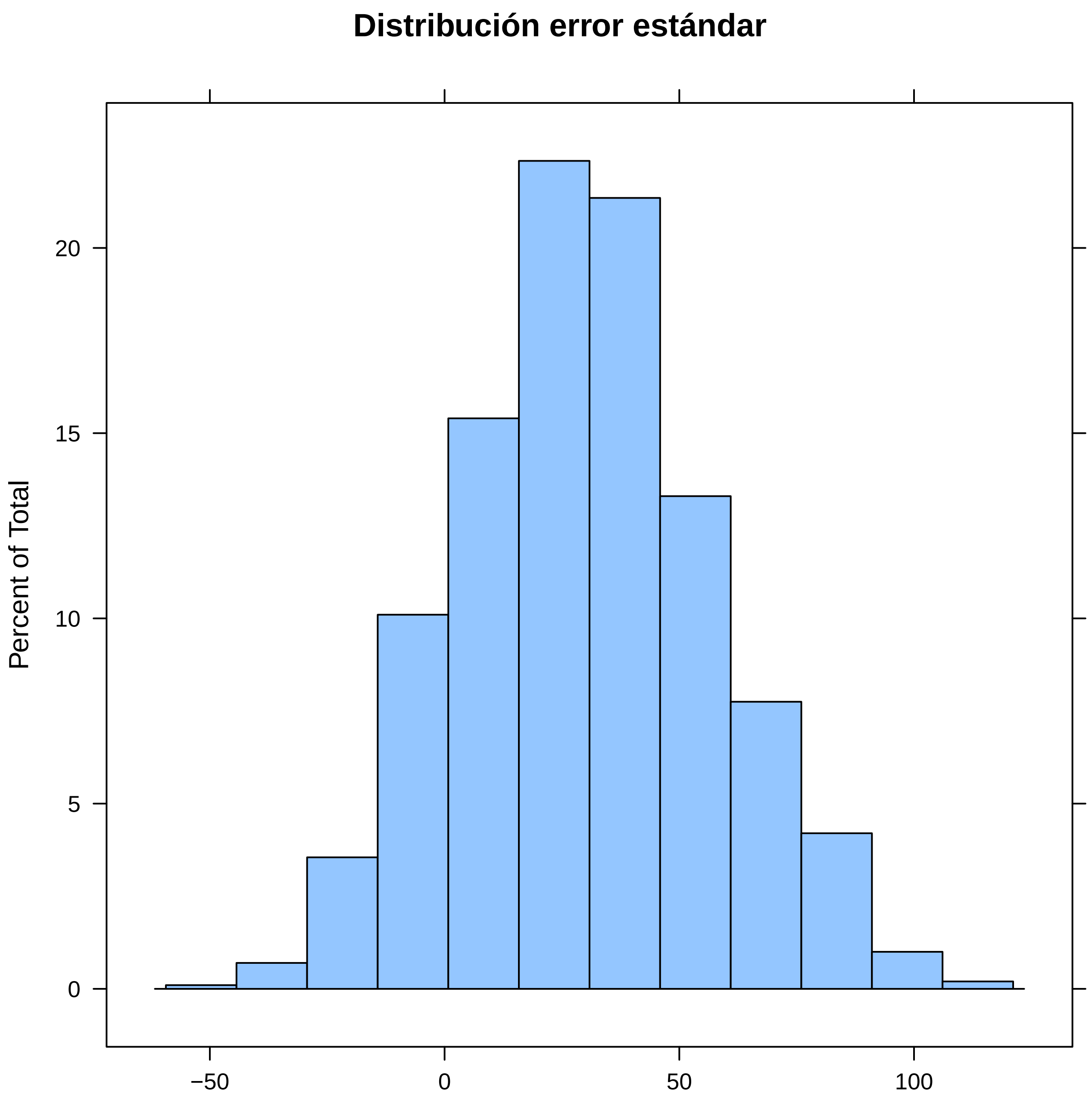
```
# install.packages("Rcpp")
if (!require(Rcpp)) install.packages("Rcpp")

library(Rcpp)

sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp ;
// [[Rcpp::export]]

List boot_MediaRatones(NumericVector trata, NumericVector control, int replicas=1000) {

    int n1 = trata.size();
    int n2 = control.size();
    double sale;

    NumericVector bs1(replicas);
    NumericVector bs2(replicas);
    NumericVector diff(replicas);

    bool replace = true;

    for(int i=0; i<replicas; i++) {

       bs1[i] =
       mean(sample(trata, n1, replace));

       bs2[i] =
       mean(sample(control, n2, replace));

       diff[i] = bs1[i]-bs2[i];
    }

    sale = sd(diff);

List saletodo;
    saletodo["sd"] = sale;
    saletodo["vector"] = diff;
    return saletodo;
}
'
)
```

```
Trata = c(94, 197, 16, 38, 99, 141, 23)
Cont = c(52, 104, 146, 10, 50, 31, 40, 27, 46)
```

```
sale = boot_MediaRatones(Trata, Cont, replica)

sale["sd"]
```

```
$sd
[1] 26.82791
```

```
lattice::histogram(unlist(sale["vector"]), main="Distribución error estándar", xlab="")
```



Calculate medians

The standard error of the *sample median* statistic is calculated.

This would be similar to the previous programs by replacing the `mean` command with `median`.

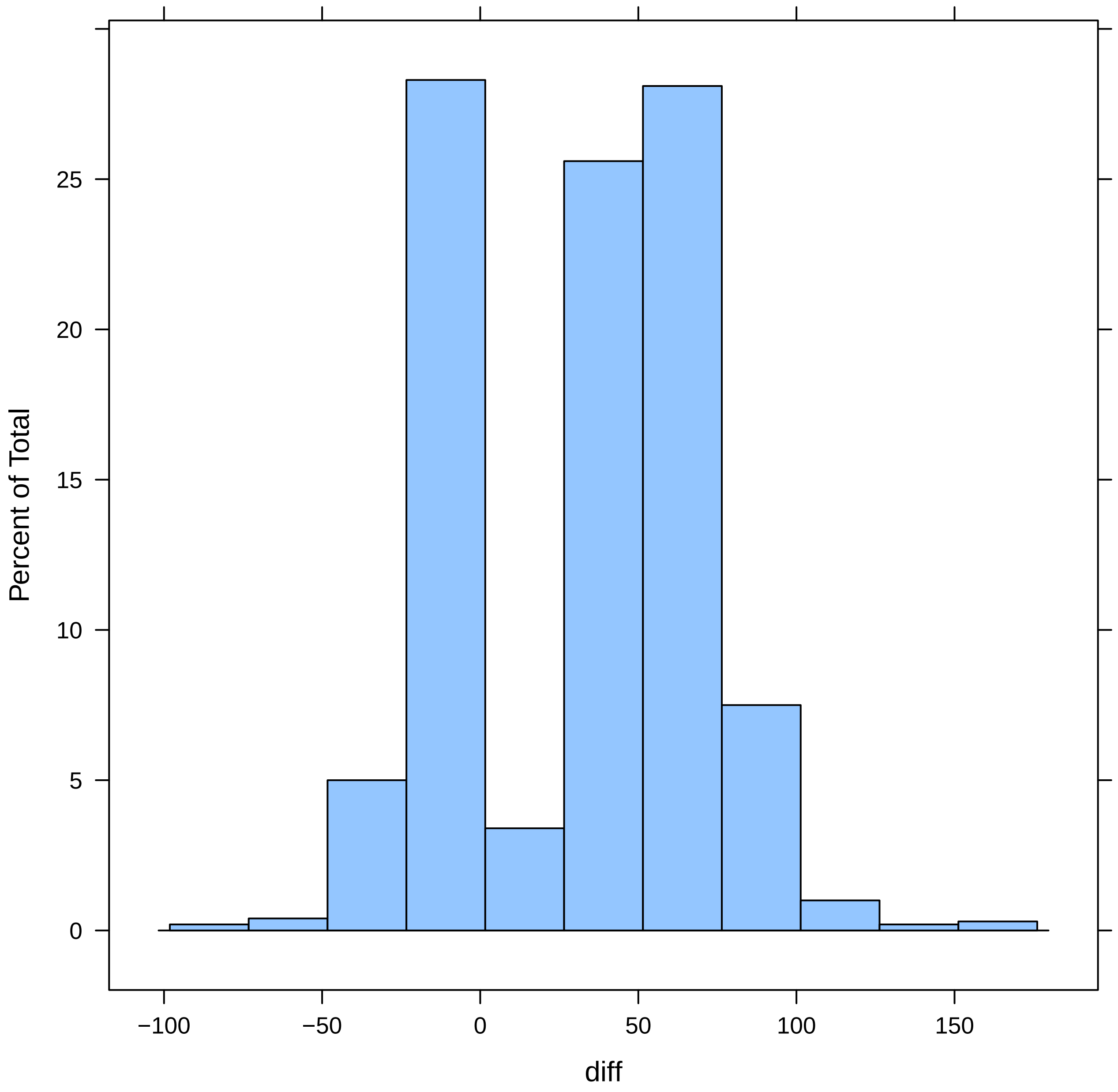
```r
n1 = length(Trata)
n2 = length(Cont)

n.bs = 1000

bs1 = rep(0, n.bs)
bs2 = rep(0, n.bs)

for (i in 1:n.bs) {
   bs1[i] = median(sample(Trata, n1, replace=TRUE))
   bs2[i] = median(sample(Cont, n2, replace=TRUE))
}
```

```r
# Bootstrap replications of difference estimators
diff = bs1-bs2
sd(diff)       # standard error estimate
```

```
[1] 38.24044
```

Histogram of the estimates of the differences:

```r
lattice::histogram(diff)
```



Mice example: Alternative programs

Means

```
B = 2000

mean(Trata) - mean(Cont)
```

```
[1] 30.63492
```

```
sd(replicate(B, mean(sample(Trata,replace=TRUE)) -
mean(sample(Cont, replace=TRUE))))
```

```
[1] 27.16673
```

```
# Medians
median(Trata) - median(Cont)
```

```
[1] 48
```

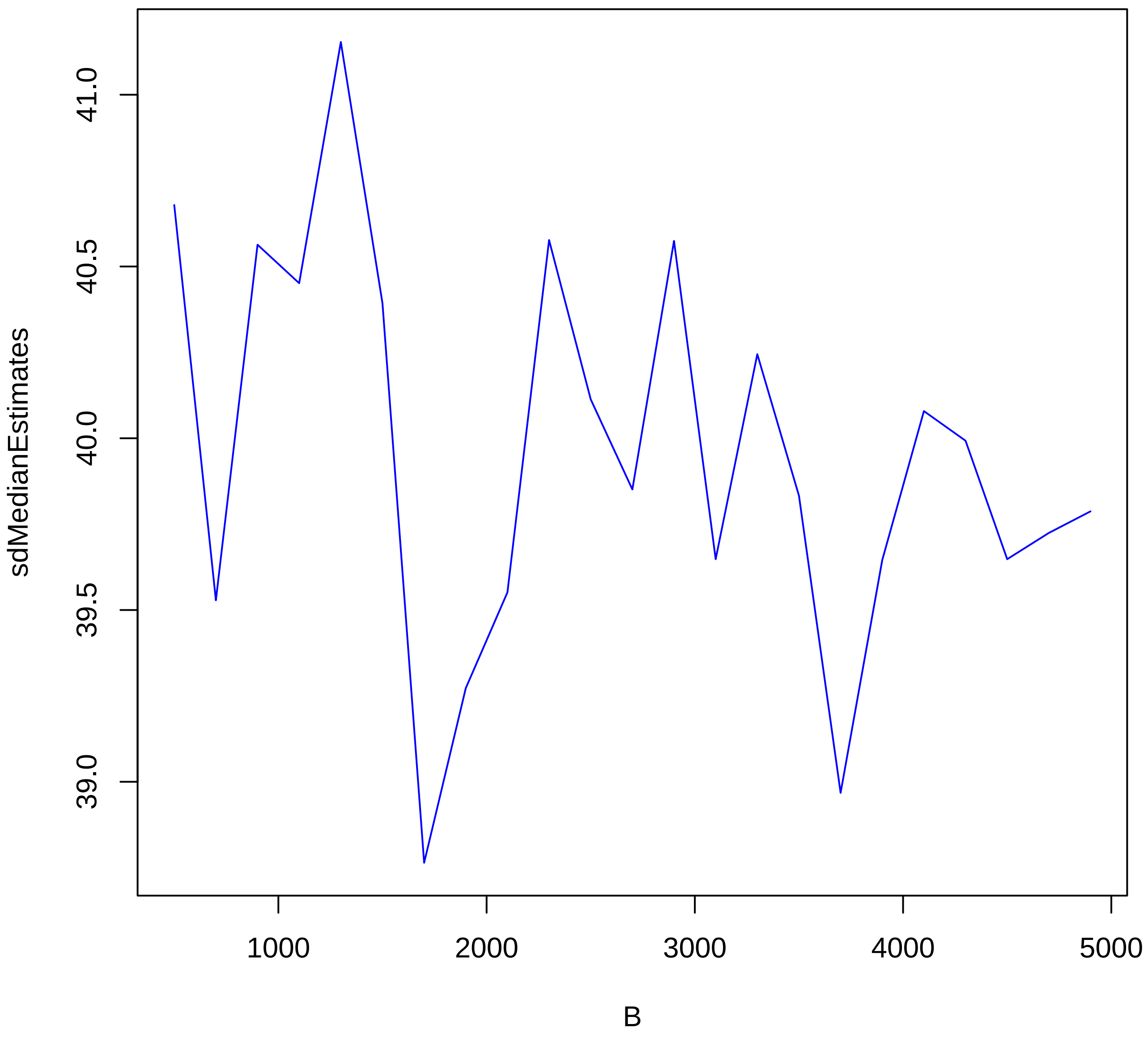
```
sd(replicate(B, median(sample(Trata,replace=TRUE)) -
median(sample(Cont, replace=TRUE))))
```

```
[1] 38.90387
```

Program for medians with different bootstrap sample sizes

```
sdMedian = function(B){
    sd(replicate(B,
    median(sample(Trata, replace=TRUE)) -
    median(sample(Cont, replace=TRUE))))
    }

# Different numbers of replicates are tested
B = seq(500, 5000, 200)
sdMedianEstimates = sapply(B, sdMedian)
```

```
plot(sdMedianEstimates ~ B, type="l", col="blue")
```

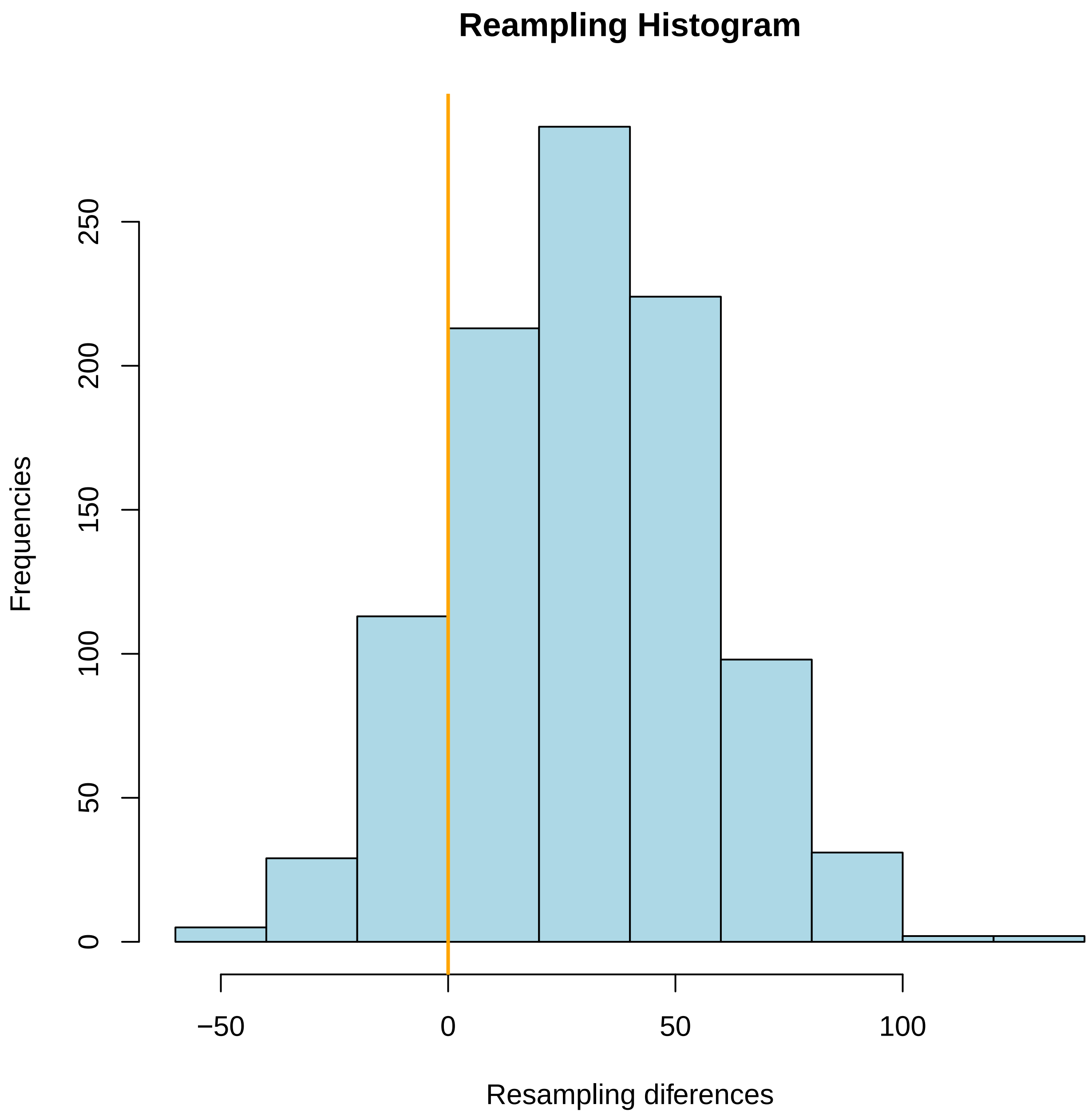

41.0
40.5
40.0
39.5
39.0
sdMedianEstimates
1000
2000
3000
4000
5000
B

Using libraries boot and bootstrap.

```r
library(bootstrap)

mouse.boot.c = bootstrap(mouse.c, 1000, mean)
mouse.boot.t = bootstrap(mouse.t, 1000, mean)

mouse.boot.diff = mouse.boot.t$thetastar -
mouse.boot.c$thetastar

sd(mouse.boot.diff)
```

```
[1] 27.59966
```

```r
hist(mouse.boot.diff, main='Reampling Histogram',
col='lightblue', xlab='Resampling diferences',
ylab='Frequencies')
abline(v=0, col="orange", lwd=2)
```



With library `boot`:

```r
library(boot)
trat = c(94, 197, 16, 38, 99, 141, 23)
control = c(52, 104, 146, 10, 51, 30, 40, 27, 46)

# Define a dataframe that assigns values 1 and 2 depending
# of whether it is control or treatment

bichos = data.frame(sobrevive=c(control,trat),
grupo = c(rep(1,length(control)), rep(2,length(trat))))
```

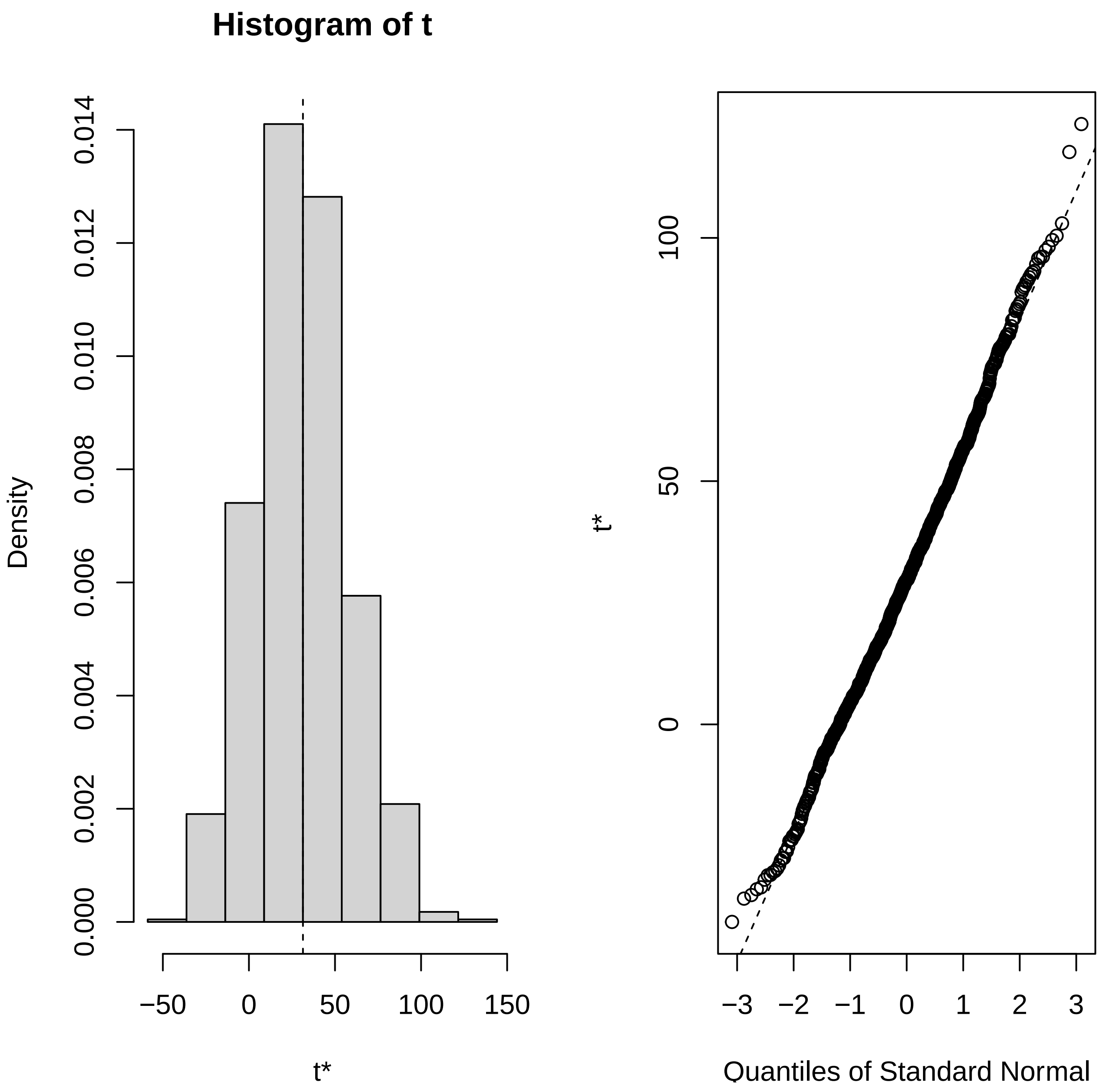
```
bichos
```

```
   sobrevive grupo
1         52     1
2        104     1
3        146     1
4         10     1
5         51     1
6         30     1
7         40     1
8         27     1
9         46     1
10        94     2
11       197     2
12        16     2
13        38     2
14        99     2
15       141     2
16        23     2
```

```
lafuncion = function(x, i) {

# Resample the observations within each group
booty = tapply(x$sobrevive, x$grupo,
FUN=function(x) {sample(x,length(x),TRUE)} )

# Calculate the differences in the means of each group
diff(sapply(booty, mean))
}

# Apply the function
bbichos = boot(data=bichos, lafuncion, R=1000)

sd(bbichos$t)
```

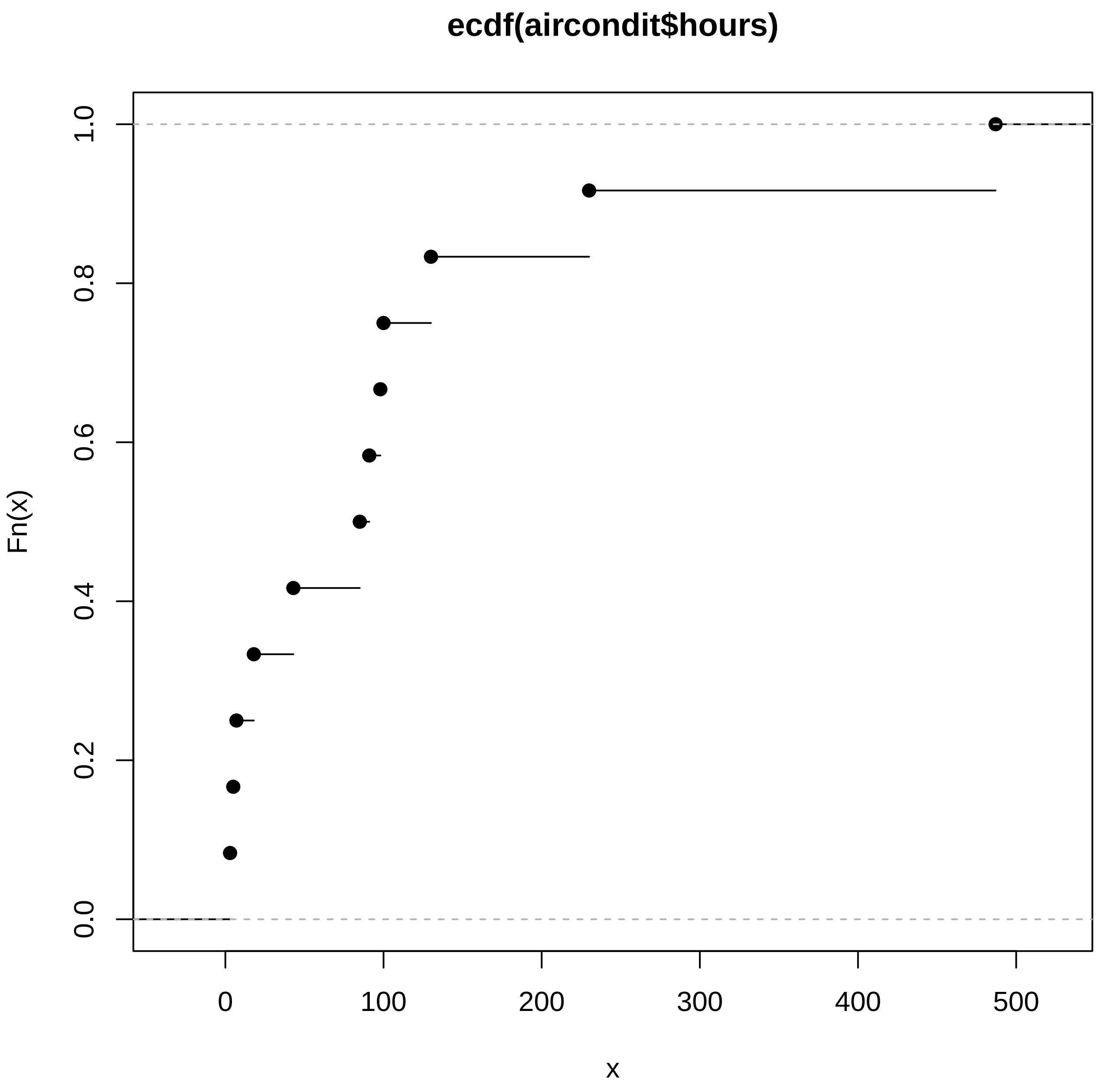
```
[1] 26.39139
```

```
plot(bbichos, nclass=10)
```


Histogram of t
Density
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
−50
0
50
100
150
t*
t*
0
50
100
−3
−2
−1
0
1
2
3
Quantiles of Standard Normal

Empirical distribution function

```r
data(aircondit, package="boot")
plot(ecdf(aircondit$hours))
```



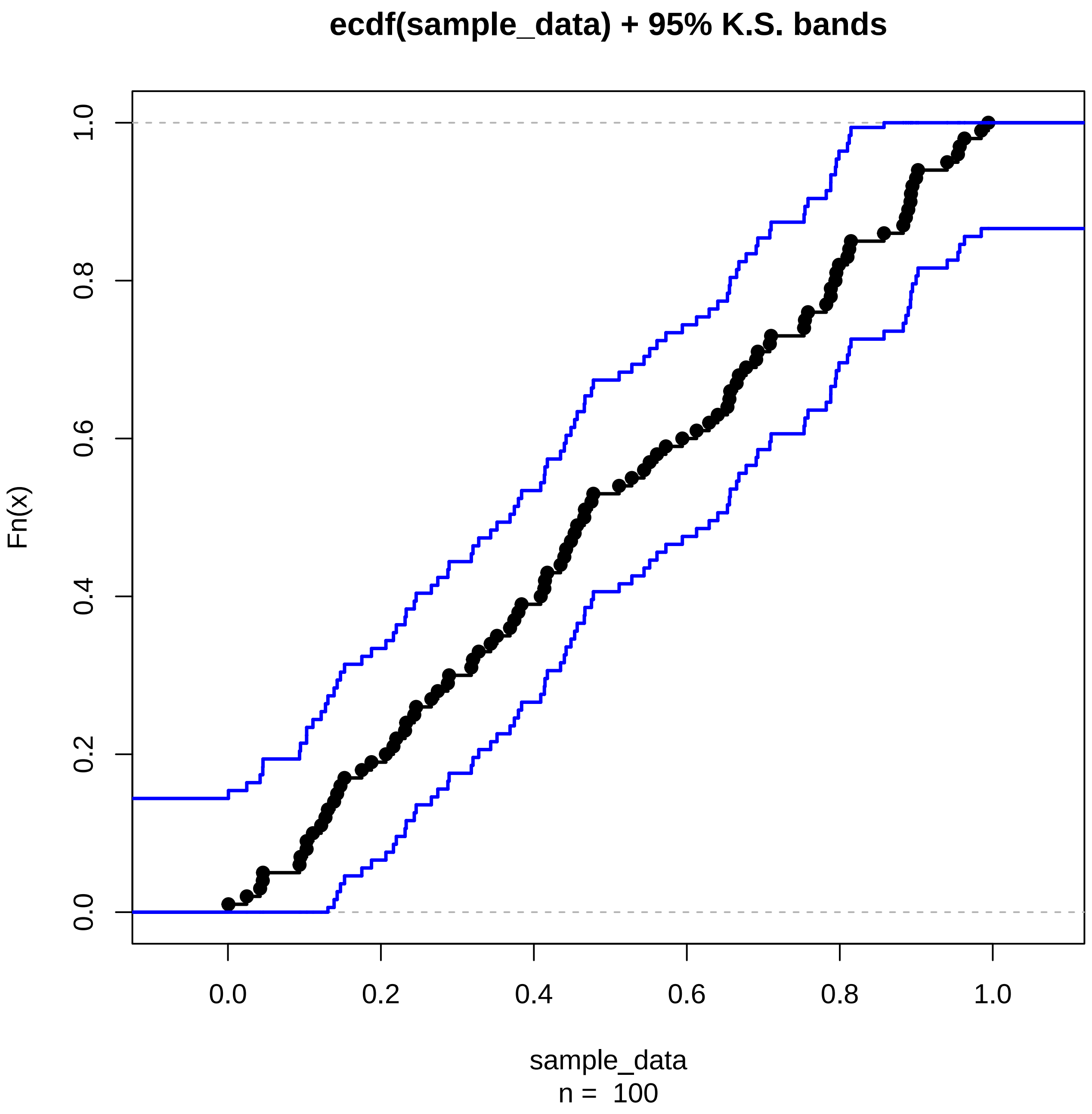
Confidence intervals based on the empirical distribution function

Example

```r
# Sample size and significance level
n = 100  # example sample size
alpha = 0.05  # significance level

# Simulating a sample
set.seed(123)  # for reproducibility
sample_data = runif(n)  # uniform distribution sample, for example

# Empirical distribution function
Fn = ecdf(sample_data)

# Epsilon calculation for confidence interval width
epsilon_n = sqrt((1/(2*n)) * log(2/alpha))

# Function to calculate confidence bounds
confidence_bounds = function(x) {
  F_hat = Fn(x)
  L = pmax(F_hat - epsilon_n, 0)
  U = pmin(F_hat + epsilon_n, 1)
  return(list(lower = L, upper = U))
}

# Example: Calculating confidence bounds for a value
x_val = 0.5  # example value
bounds = confidence_bounds(x_val)
cat("Confidence bounds at x =", x_val, "\nLower:", bounds$lower,
"\nUpper:", bounds$upper, "\n")
```

```
Confidence bounds at x = 0.5
Lower: 0.3941898
Upper: 0.6658102
```

```r
library(sfsmisc)

ecdf.ksCI(sample_data, ci.col="blue", lwd=2)
```



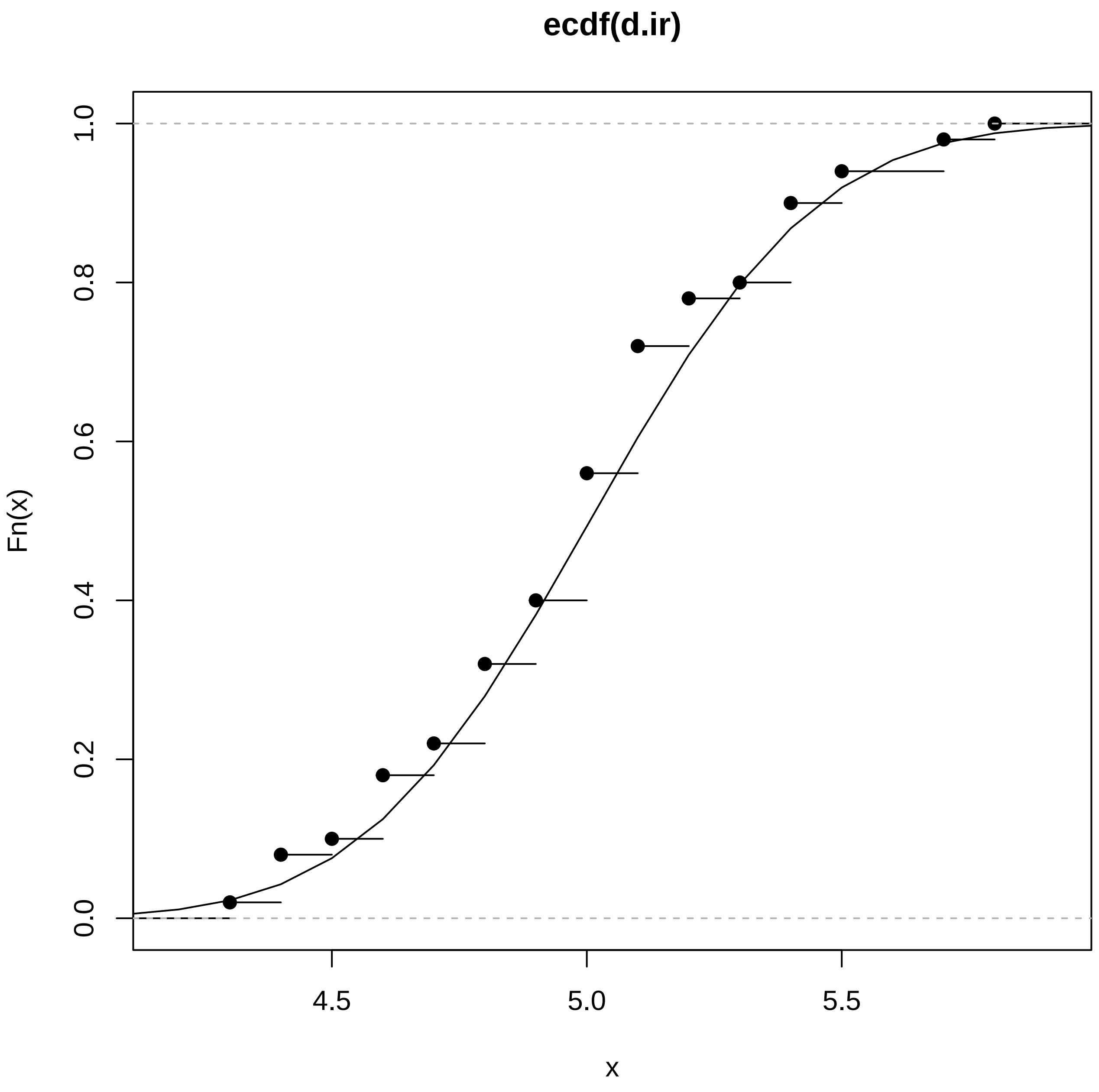
Alternatives to the empirical cdf

The following R code illustrates how to compare the empirical cdf with a parametric cdf, specifically the normal distribution, using the *iris* dataset.

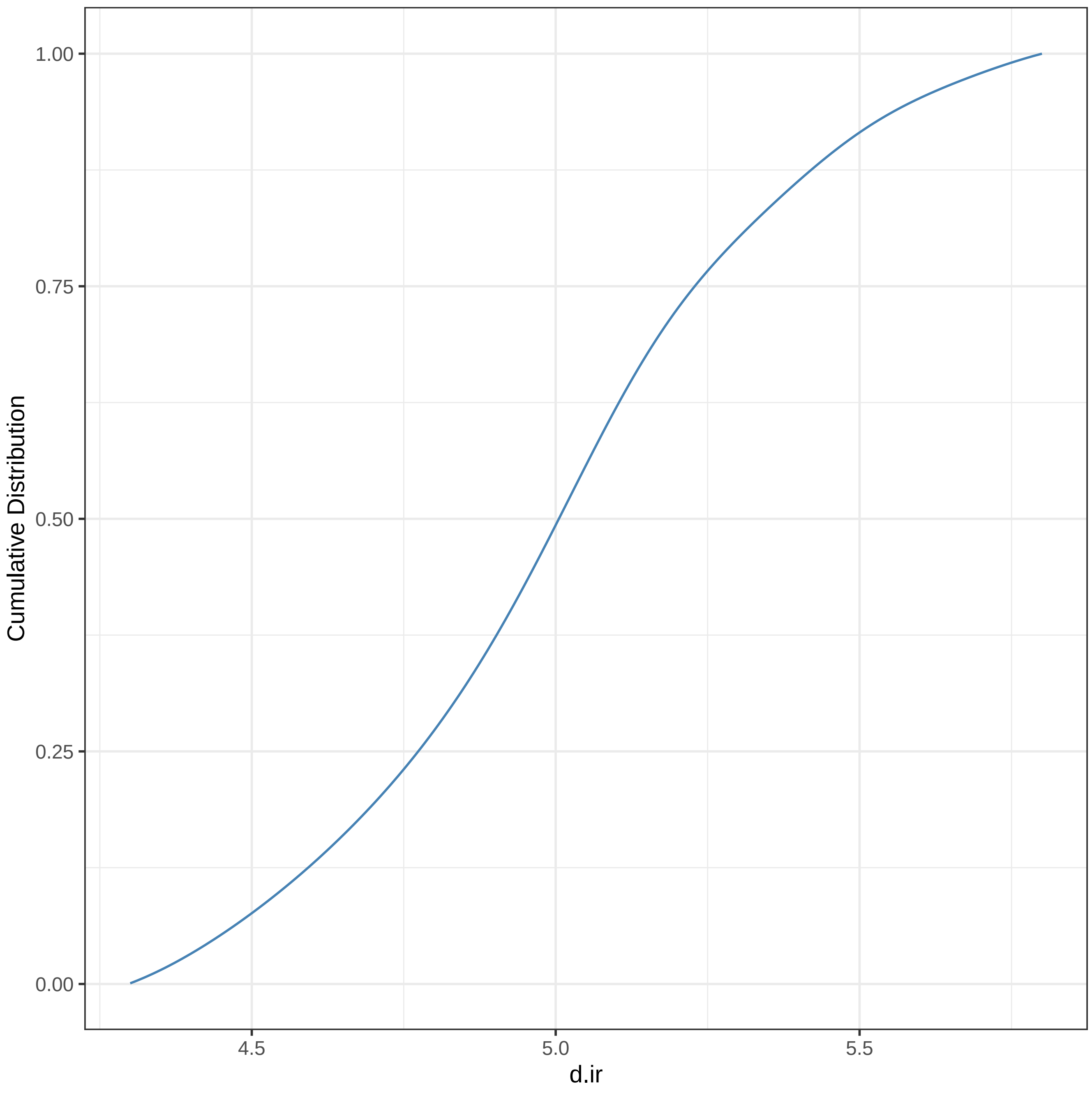
We extract the sepal lengths of the *setosa* species and then plot the empirical cdf of these sepal lengths.

Then, it overlays a line representing the cdf of the normal distribution, with the mean and standard deviation parameters estimated from the *setosa* sepal lengths.

```r
# Load the iris dataset
data("iris")

# Extract sepal lengths for the setosa species
d.ir = iris$Sepal.Length[iris$Species == "setosa"]

# Plot the empirical cdf of the setosa sepal lengths
plot(ecdf(d.ir))

# Generate a sequence of values covering the range of observed data
t = seq(min(d.ir) - 1, max(d.ir) + 1, by = 0.1)

# Overlay the normal cdf with parameters estimated from the data
points(t, pnorm(t, mean = mean(d.ir), sd = sd(d.ir)), type = "l")
```



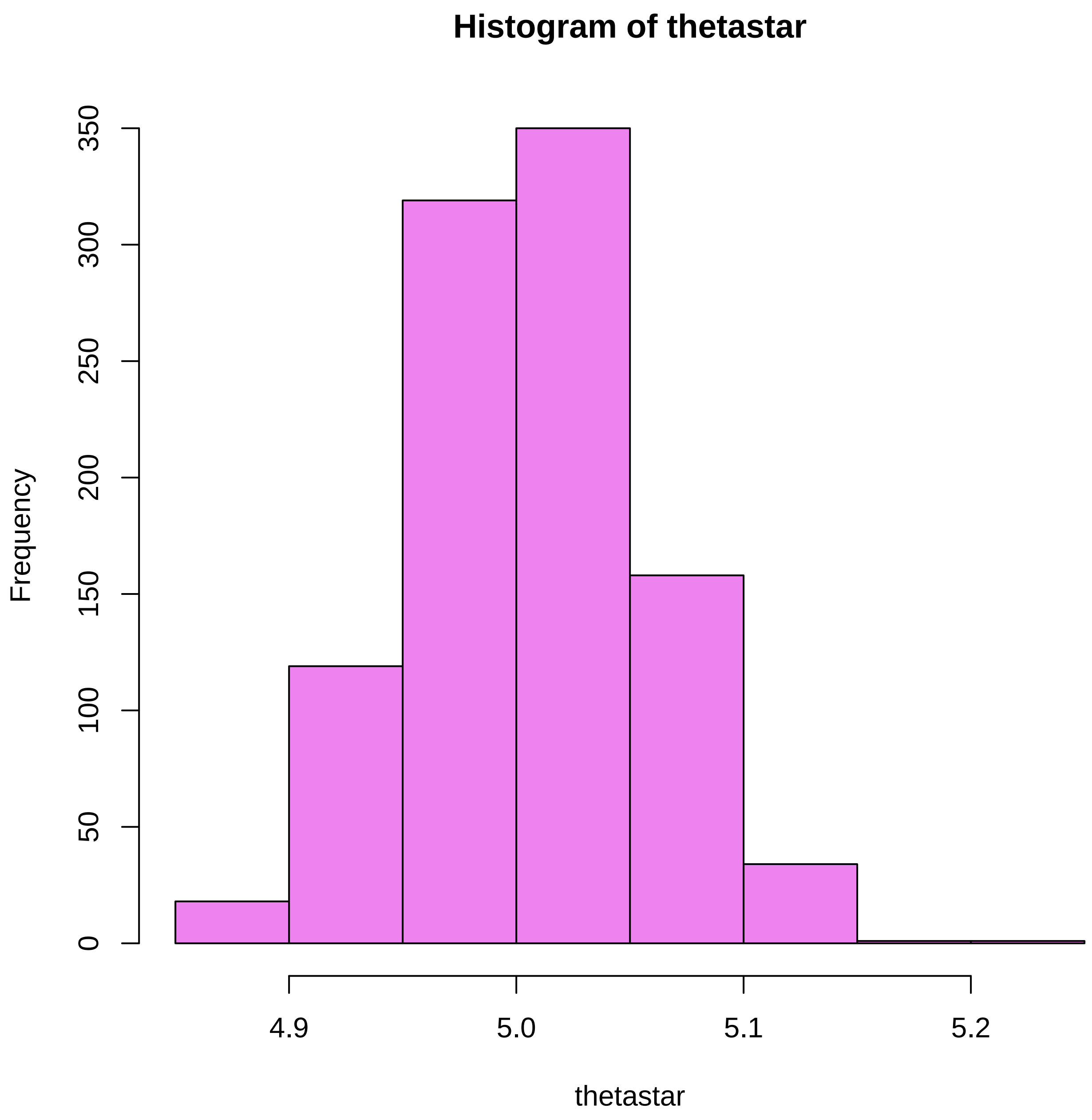
```
# install.packages("remotes")
# remotes::install_github("towananalytics/tatools")

library(tatools)

# Smoothed Empirical CDF plot
smtho = smooth_ecd(data = as.data.frame(d.ir), y = d.ir)
```



Estimating standard errors with bootstrap

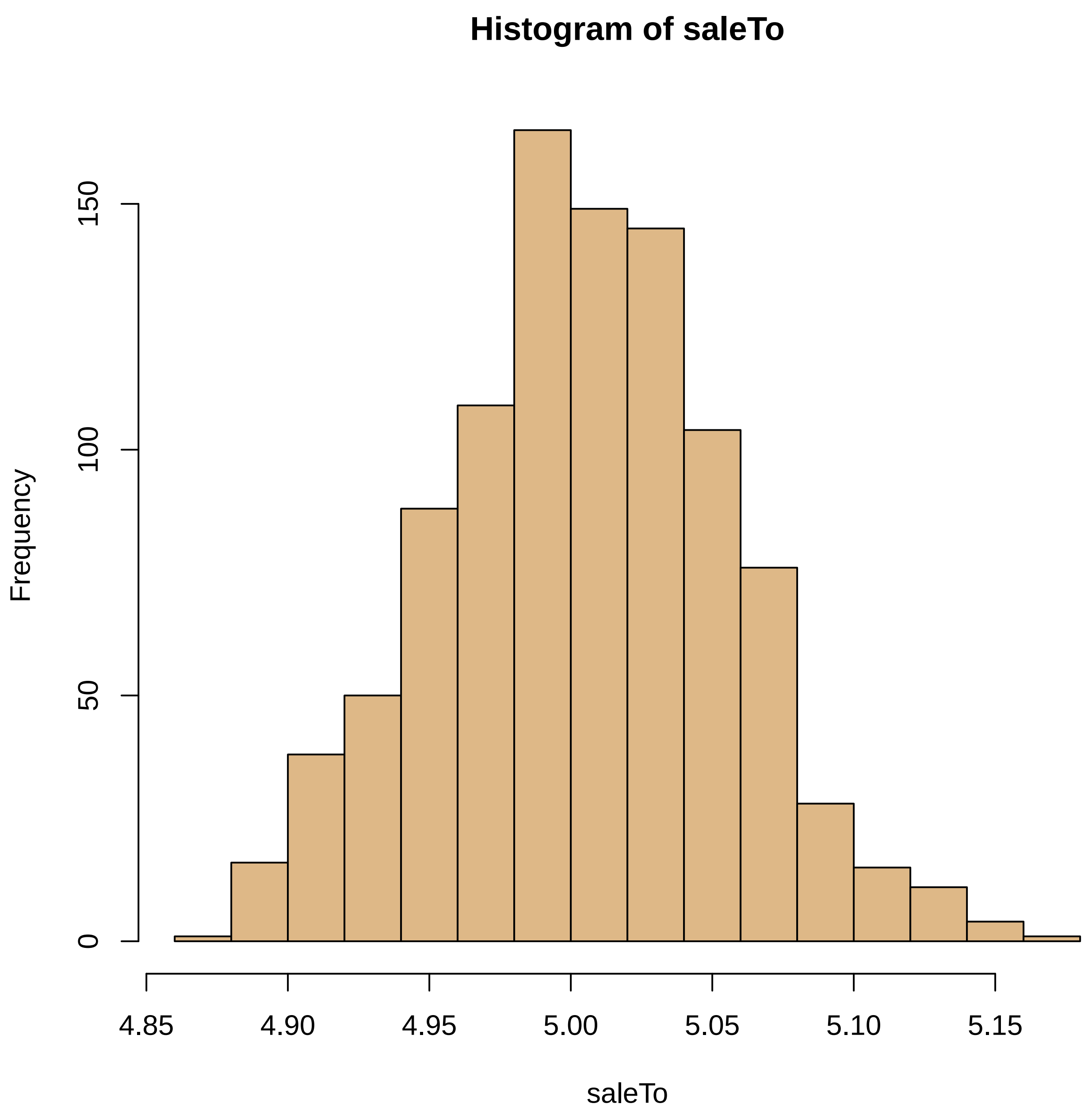
```r
data(iris)
# attach(iris)

# Using library bootstrap
```

```r
library(bootstrap)

B = 1000
x = iris$Sepal.Length[iris$Species=="setosa"]
n = length(x)
thetastar = bootstrap(x=x, nboot=B, theta=mean)$thetastar

hist(thetastar, col="violet")
```



```r
# Alternative by hand
bootstrapping = matrix(sample(size=B*n,x=x,replace=T), ncol=n,byrow=T)
```

```r
saleTo = apply(X=bootstrapping, MARGIN=1, FUN=mean)
hist(saleTo, col="burlywood")
```



```r
# standard error of the mean
sd(x)/sqrt(n)
```

```
[1] 0.04984957
```

```r
# bootstrap standard error
sd(bootstrap(x=x, nboot=B, theta=mean)$thetastar)
```

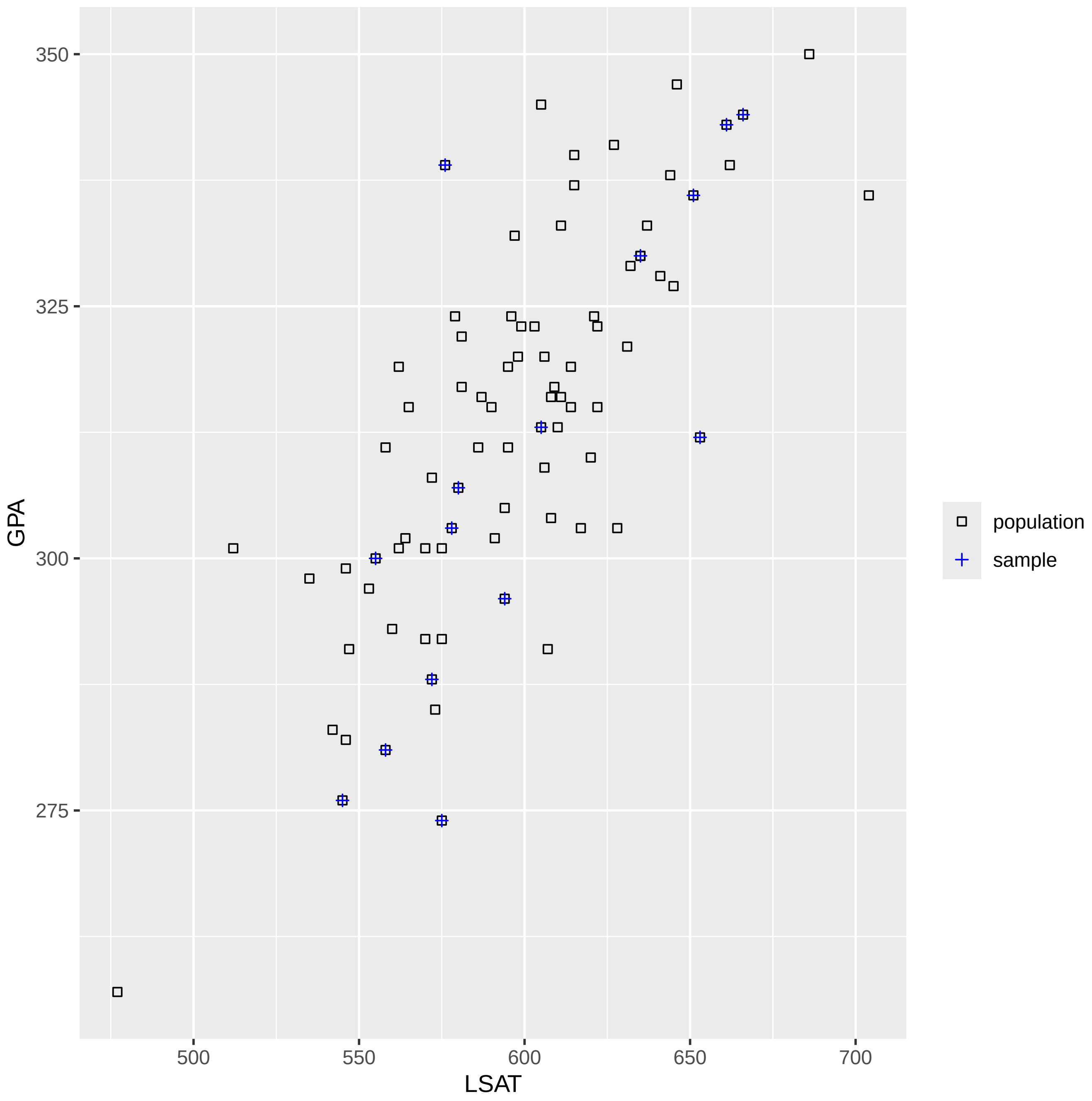
```
[1] 0.05101777
```

Parametric bootstrap

The example of universities with master's degrees in law is taken, which is included in the book by Efron and Tibshirani (1993).

This is a *population* of 82 universities that teach a master's degree in Law.

```r
library(bootstrap)
library(ggplot2)

df1 = data.frame(LSAT = law82$LSAT, GPA = 100*law82$GPA)
df2 = data.frame(LSAT = law$LSAT, GPA = 100*law$GPA)

ggplot() +
    geom_point(data = df1, aes(x = LSAT, y = GPA, color = "population"), shape = 0) +
    geom_point(data = df2, aes(x = LSAT, y = GPA, color = "sample"), shape = 3) +
    labs(x = "LSAT", y = "GPA") +
    scale_color_manual(name = "", values = c("population" = "black", "sample" = "blue"),
                       guide = guide_legend(override.aes = list(shape = c(0, 3))))
```



We calculate the correlation between GPA (the average score in undergraduate courses) and LSAT (admission score).

The population correlation is

```
with(law82, cor(GPA,LSAT))
```

```
[1] 0.7599979
```

The sample correlation (*plug-in* estimator) is

```
with(law, cor(GPA,LSAT))
```

```
[1] 0.7763745
```

In the law82 example, instead of estimating the distribution function F using the empirical distribution function, one can assume that the population is distributed as a bivariate normal.

For the mean and the covariance matrix of this distribution, the reasonable estimators would be respectively

$$(\bar{x}, \bar{y})$$

$$\frac{1}{14}\begin{pmatrix} \sum_i (x_i-\overline{x})^2 & \sum_i (x_i-\overline{x})(y_i-\overline{y}) \\ \sum_i (x_i-\overline{x})(y_i-\overline{y}) & \sum_i (y_i-\overline{y})^2 \end{pmatrix}$$

The bivariate normal population obtained with this mean and covariance matrix is denoted as $\widehat{F}_{par}$.

The parametric bootstrap estimator of the standard error of the parameter is called $se_{\hat{F}_{par}}(\widehat{\theta}^*)$.

Instead of sampling with replacement from the original data, B samples of size n are drawn from the parametric estimator of the population$\widehat{F}_{par}$

$$\widehat{F}_{par} \rightarrow (x_1^*, x_2^*, \ldots, x_n^*)$$

Subsequently, the same steps 2 and 3 of the general bootstrap *non-parametric* algorithm are followed: the corresponding statistic is calculated in each bootstrap sample and then the standard deviation of the B replicates is calculated.

In the example of the data from the master's in law study centers, if (x, y) are distributed as a bivariate normal, then observations of this vector can be generated, defining

$$\begin{aligned} x &= \mu_x + \sigma_x z_1 \\ y &= \mu_y + \sigma_y \frac{z_1 + c \cdot z_2}{\sqrt{1+c^2}} \end{aligned}$$

where $z_1, z_2 \sim N(0,1)$

$$c = \sqrt{\frac{\sigma_x^2 \sigma_y^2}{\sigma_{xy}^2} - 1}$$

```r
library(bootstrap)

paraBoot = function(datos) {
   ndatos = dim(datos)[1]
   sigma = cov(datos)
   mu = sapply(datos, mean)

   c = sqrt(prod(diag(sigma))/sigma[1,2]^2-1)
  z1 = rnorm(ndatos)
  z2 = rnorm(ndatos)
  x = mu[1] + sqrt(sigma[1,1])*z1
  y = mu[2] + sqrt(sigma[2,2])*(z1+c*z2)/sqrt(1+c^2)
cbind(x,y)
}
```

Alternatively, it can be programmed with a specific library that handles multivariate normal: `mvrnorm`

```r
paraBoot = function(datos) {
   ndatos = dim(datos)[1]
   sigma = cov(datos)
   mu = sapply(datos, mean)

   x = MASS::mvrnorm(ndatos, mu=mu, Sigma=sigma)
return(x)
}
```

Parametric bootstrap approach

```r
pBoot = replicate(5000, cor(paraBoot(law))[2,1])

sd(pBoot)
```

```
[1] 0.118582
```

With the library `boot`:

```r
library(boot)

simulaBoot = function(datos, ...) {
  ndatos = dim(datos)[1]
  x = MASS::mvrnorm(ndatos, mu=mu0, Sigma=sigma0)
  return(x)
}
```

```
sigma0 = Rfast::mvnorm.mle(as.matrix(law))$sigma
mu0 = Rfast::mvnorm.mle(as.matrix(law))$mu

estadistico = function(x,i){ cor(x)[2,1] }


saleB = boot(data=law, sim="parametric", ran.gen=simulaBoot, mle=list(mu0, sigma0),
statistic = estadistico, R=1000)

saleB
```

```
PARAMETRIC BOOTSTRAP


Call:
boot(data = law, statistic = estadistico, R = 1000, sim = "parametric",
    ran.gen = simulaBoot, mle = list(mu0, sigma0))


Bootstrap Statistics :
     original       bias    std. error
t1* 0.7763745 -0.007524265   0.1182801
```

```
boot.ci(saleB,type = "perc")
```

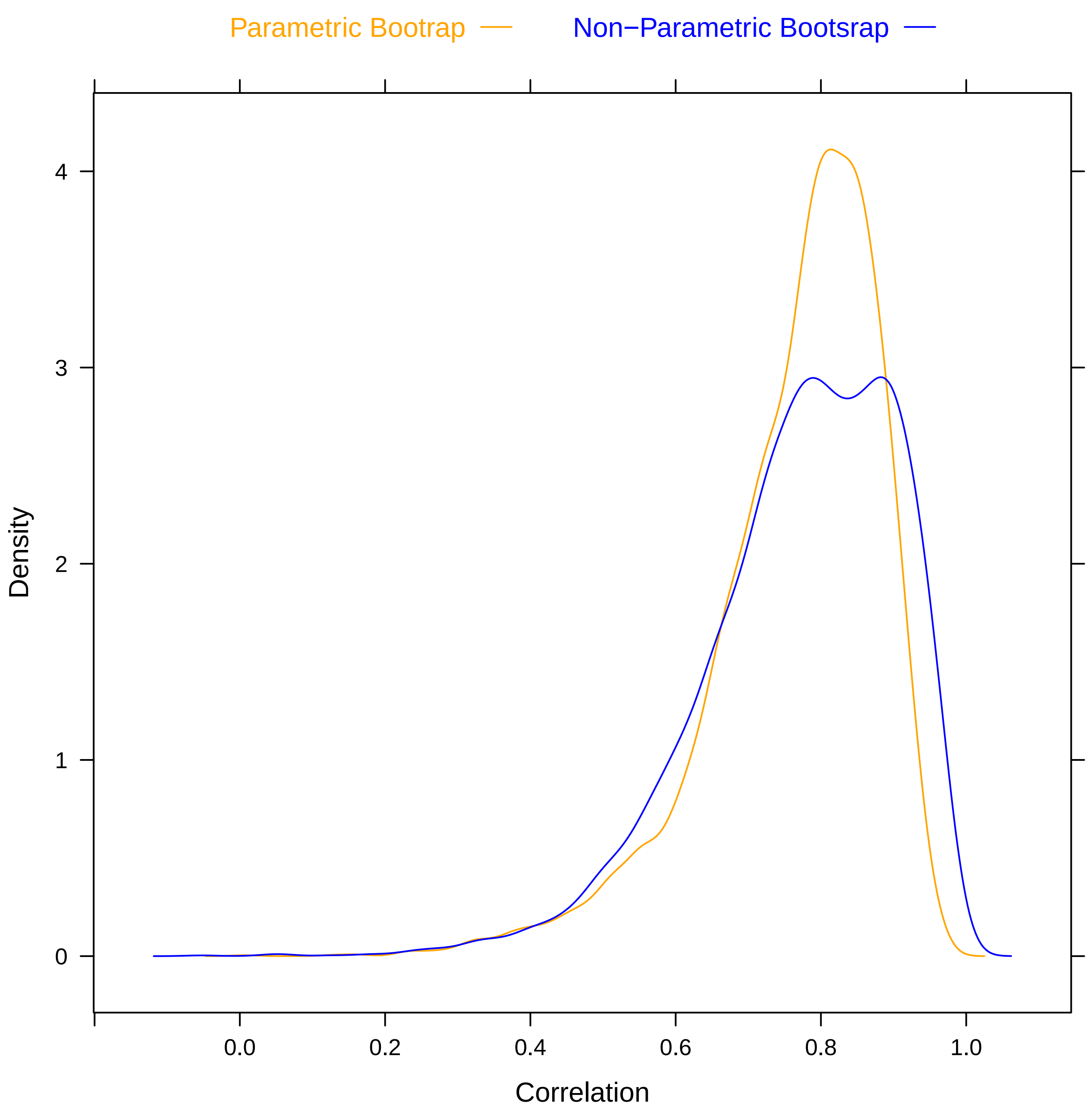
```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = saleB, type = "perc")

Intervals :
Level     Percentile
95%   ( 0.4826,  0.9286 )
Calculations and Intervals on Original Scale
```

Asymptotic approximation

```
samplesize = dim(law)[1]

(1-cor(law)[2,1]^2)/sqrt(samplesize-3)
```

```
[1] 0.1146741
```

```
library(latticeExtra)

ind1 = 1:dim(law)[1]
law.boot = replicate(5000, {indB = sample(ind1,
size=dim(law)[1], replace=TRUE);
with(law[indB,], cor(GPA,LSAT))})

densityplot(~pBoot + law.boot, plot.points=FALSE,
auto.key=list(columns=2, size=2, between=1, col=c("orange","blue"),
text=c("Parametric Bootrap", "Non-Parametric Bootsrap")), xlab="Correlation",
par.settings=list(superpose.line=list(col=c("orange","blue"))))
```



When bootstrap can fail

In general the bootstrap does not work when applied to parameters that are on the border of the parametric space.

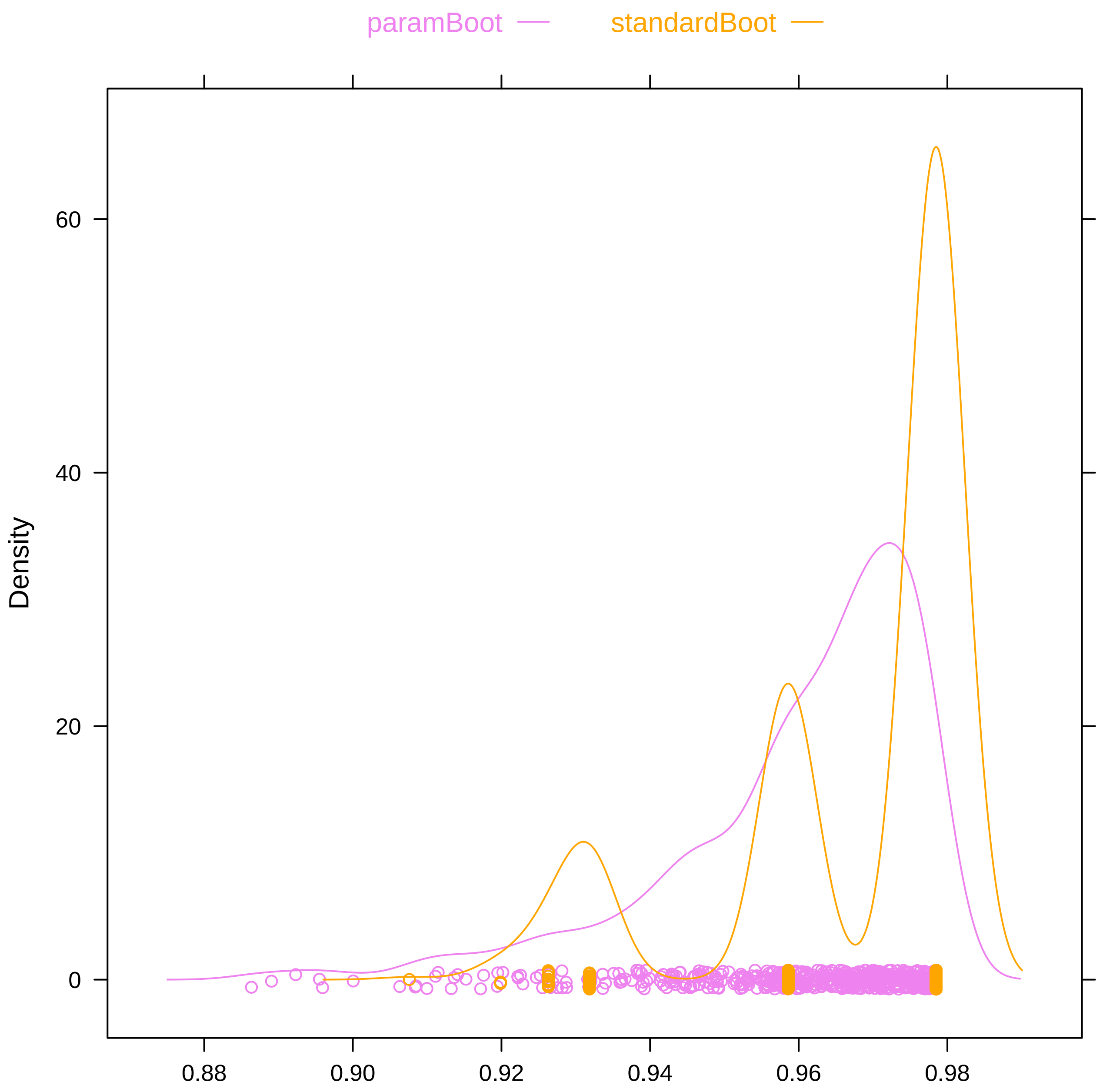
We consider the following example:

- X is distributed as a uniform distribution on $(0, \theta)$.
- The maximum likelihood estimator for θ is the $\max(X_i)$
- We have a sample of 50 observations.
- We compare the non-parametric bootstrap estimator of θ with respect to its parametric estimator.

```
N = 50
X = runif(N)
(thetaHat = max(X))
```

```
[1] 0.9784884
```

```
standardBoot = replicate(500,
max(sample(X, N, replace=TRUE)))

paramBoot = replicate(500,
max(runif(N, min=0, max=thetaHat)))

library(latticeExtra)

densityplot(~paramBoot + standardBoot, xlab="",
auto.key=list(columns=2, size=2, between=1,
col=c("violet", "orange")),
par.settings=list(superpose.line=
list(col=c("violet", "orange"))),
col=c("violet", "orange"))
```


paramBoot
standardBoot
Density
60
40
20
0
0.88
0.90
0.92
0.94
0.96
0.98

Jackknife

There is a series of data on the effect of hormonal patches on 8 people. These patches diffuse a medication into the blood.

The level of the hormone that appears after using three different patches is measured: a *placebo* patch (without hormone), an *old* patch, and a *new* patch. It is about studying its bioequivalence.

The criterion used by the USA National Drug Agency FDA is that

$$\frac{|E(new) - E(old)|}{E(old) - E(placebo)} \leq 0.20$$

That is, the FDA requires that the new type of patch adjust to the amount of hormone released by the old one (compared to the placebo) by no more than 20%.

It is called as

- **z**: (old patch measurement - placebo measurement)
- **y**: (new patch measurement - old patch measurement)

It is assumed that the values $\mathbf{x}_i = (z_i, y_i)$ are obtained from a bivariate distribution $F \to \mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_8)$.

The parameter in this case is then

$$\theta = t(F) = \frac{E_F(\mathbf{y})}{E_F(\mathbf{z})} = \frac{E(new) - E(old)}{E(old) - E(placebo)}$$

In this case $t(\cdot)$ is a function that has as its object the pairs of $\mathbf{X}$ and results in the expectation ratio.

The plug-in estimator of θ is

$$\hat{\theta} = t(\hat{F}) = \frac{\overline{y}}{\overline{z}} = \frac{\sum_i y_i/8}{\sum_i z_i/8}$$

In the example,

$$\frac{E(new) - E(old)}{E(old) - E(placebo)} = \frac{E(\mathbf{y})}{E(\mathbf{z})} \approx \frac{\overline{y}}{\overline{z}} = \frac{-452.25}{6342.375} \approx -0.0713$$

The *patch* hormone data are loaded from the `bootstrap` library (which contains the data files from the book by Efron and Tibshirani (1993)).

```
data(patch, package="bootstrap")

n = nrow(patch)
y = patch$y
z = patch$z
theta.hat = mean(y) / mean(z)

y
```

```
[1] -1200  2601 -2705  1982 -1290   351  -638 -2719
```

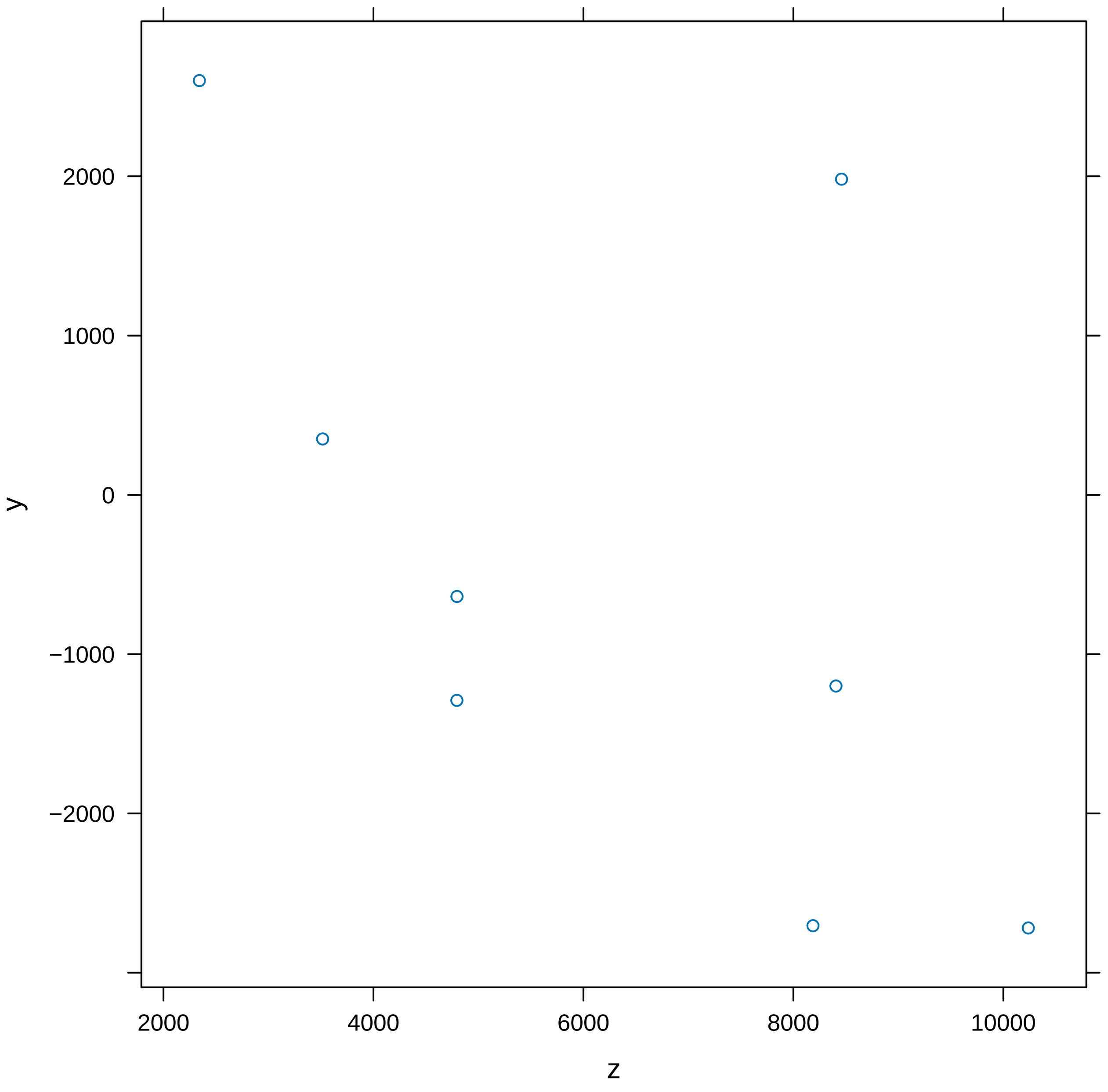
```
    z
```

```
[1]  8406  2342  8187  8459  4795  3516  4796 10238
```

```
    theta.hat
```

```
[1] -0.0713061
```

```r
library(lattice)
xyplot(y ~ z, data=patch)
```



```r
jacks = sapply(1:n, function(i) with(patch[-i,], mean(y)/mean(z)))
jacks
```

```
[1] -0.05711856 -0.12849970 -0.02145610 -0.13245033 -0.05067038 -0.08404803
[7] -0.06486298 -0.02219698
```

According to the theoretical formula:

```r
(biasJack = (n-1)*(mean(jacks) - theta.hat))
```

```
[1] 0.008002488
```

Alternatively

```r
theta.jack = numeric(n)

for (i in 1:n){
   theta.jack[i] = mean(y[-i]) / mean(z[-i])
}

(sesgo = (n-1)*(mean(theta.jack) - theta.hat))
```

```
[1] 0.008002488
```

According to the theoretical formula:

```r
se = sqrt((n-1) * mean((theta.jack - mean(theta.jack))^2))
se
```

```
[1] 0.1055278
```

You can use `Rcpp` which is more efficient when the sample size is high.

```r
library(Rcpp)

sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]

List jackknifeEstimas(NumericVector y, NumericVector z) {

    int n = y.size();
    NumericVector theta_jack(n);

    for (int i = 0; i < n; i++) {
        double mean_y = (sum(y) - y[i]) / (n - 1);
        double mean_z = (sum(z) - z[i]) / (n - 1);
        theta_jack[i] = mean_y / mean_z;
    }

    double theta_hat = mean(y) / mean(z);
    double bias = (n - 1) * (mean(theta_jack) - theta_hat);
    double se = sqrt((n-1) * mean(pow(theta_jack - mean(theta_jack), 2)));

    return List::create(Named("Theta Original") = theta_hat,
    Named("bias") = bias, Named("se") = se);
}
')

# Data
z = c(8406, 2342, 8187, 8459, 4795, 3516, 4796, 10238)
y = c(-1200,  2601, -2705,  1982, -1290, 351, -638, -2719)

resulta = jackknifeEstimas(y, z)
resulta
```

```
$`Theta Original`
[1] -0.0713061

$bias
[1] 0.008002488

$se
[1] 0.1055278
```

Alternatively, you can calculate the jackknife estimators using the bootstrap library:

```
library(bootstrap)

    n = nrow(patch)
    y = patch$y
    z = patch$z

datos = as.matrix(cbind(y, z))
```

```
sampbioeq = mean(y)/mean(z)
sampbioeq
```

```
[1] -0.0713061
```

```
# Define the bioequivalence function
bioeq = function(ind, datos) { mean(datos[ind,1])/mean(datos[ind,2]) }

# Apply jackknife
jackbioeq = jackknife(1:n, bioeq, datos)
jackbioeq
```

```
$jack.se
[1] 0.1055278

$jack.bias
[1] 0.008002488

$jack.values
[1] -0.05711856 -0.12849970 -0.02145610 -0.13245033 -0.05067038 -0.08404803
[7] -0.06486298 -0.02219698

$call
jackknife(x = 1:n, theta = bioeq, datos)
```

Estimation of bias

```
# Bias-corrected jackknife estimator
bioeqjack = sampbioeq - jackbioeq$jack.bias
bioeqjack
```

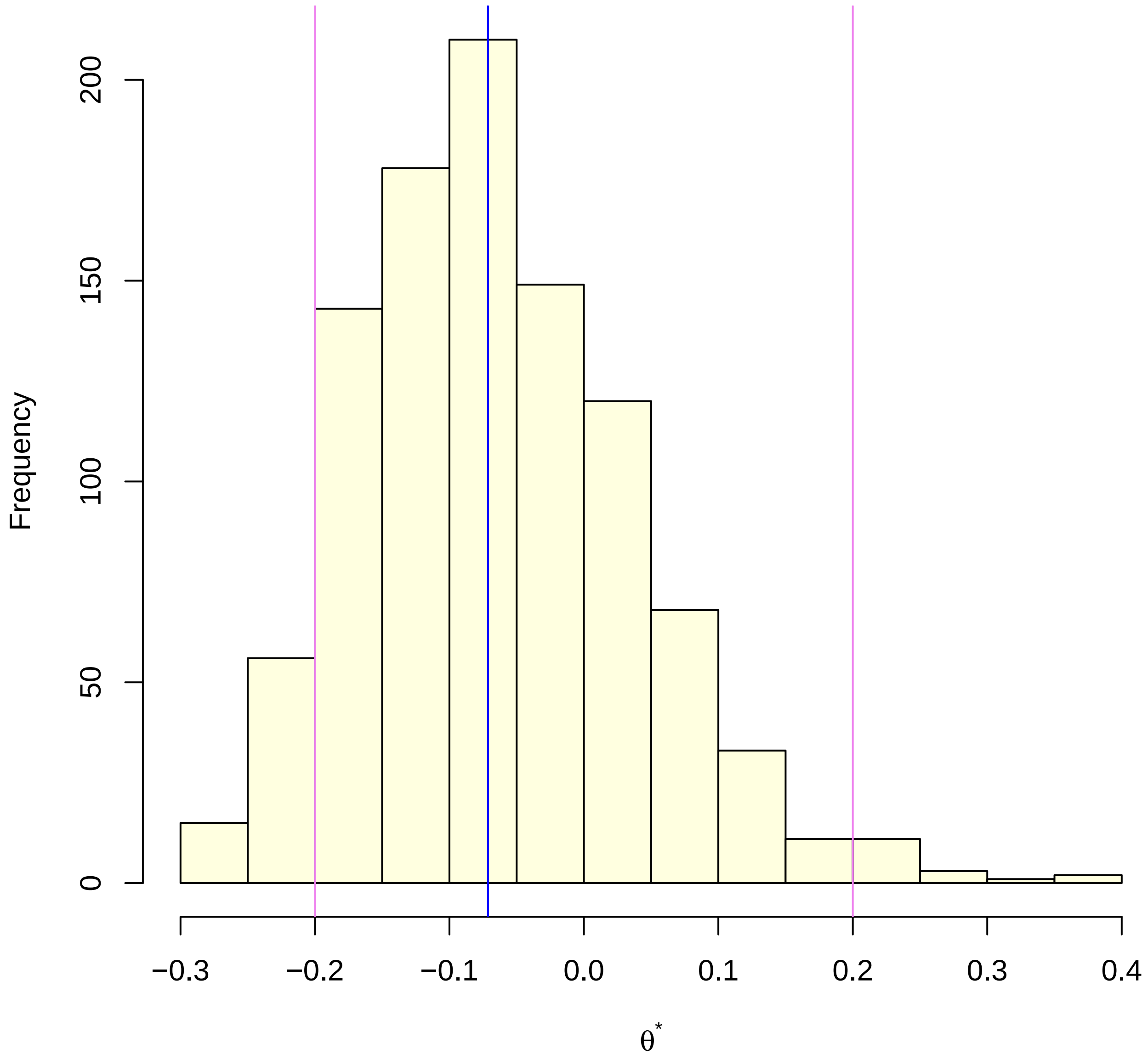
```
[1] -0.07930858
```

Applying the boot library, we obtain:

```
library(boot)

ratiofun = function(data, ind){
      Y = data[ind,6]
      Z = data[ind,5]
      return(mean(Y)/mean(Z)) }

patch.boot = boot(patch, statistic = ratiofun, R=1000)

(theta.hat = ratiofun(patch, 1:8))     # Original plug-in estimator
```

```
[1] -0.0713061
```

```
hist(patch.boot$t, xlab=expression(theta^"*"), col = "lightyellow",
main="")
abline(v=c(-0.2, 0.2), col="violet")
abline(v=theta.hat, col="blue")
```



The bootstrap standard error is obtained:

```
sd(patch.boot$t)
```

```
[1] 0.1019104
```

The bootstrap estimator of bias is also computed:

```
(mean(patch.boot$t) - theta.hat)
```

```
[1] 0.005869451
```

Or alternatively, it can be obtained from the output of the `boot` command:

```
patch.boot
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = patch, statistic = ratiofun, R = 1000)


Bootstrap Statistics :
      original      bias    std. error
t1* -0.0713061 0.005869451   0.1019104
```