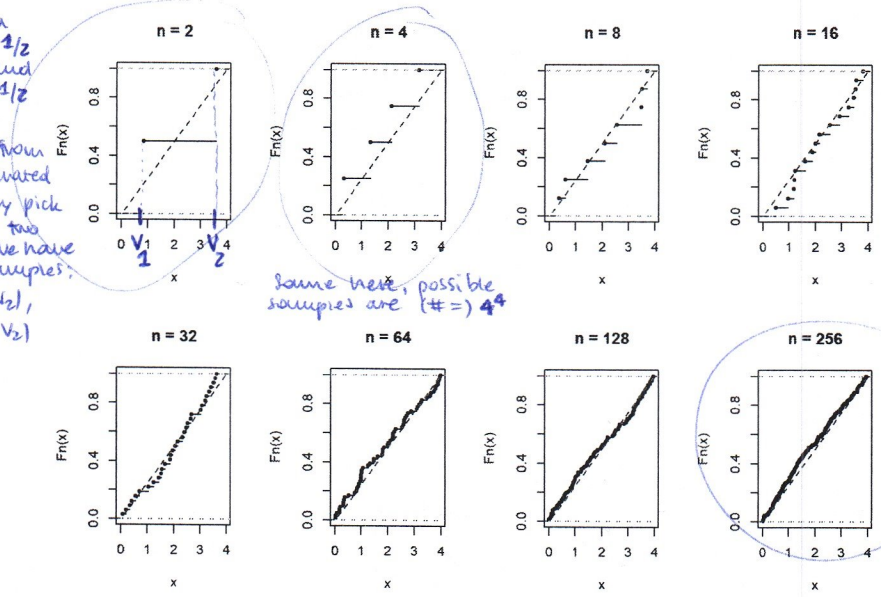
```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### BOOTSTRAP ESTIMATION - Real world vs. Boostrap world
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### Simulation: Law of Large Numbers
### ---------------------------------------------------------------------------
set.seed(24021979)
n.grid  <- 2^(1:8)
par(mfrow = c(2,4))
for(n in n.grid){
  x <- runif(n,0,4)
  plot(ecdf(x), main=paste('n =', n), xlim = c(0,4))
  lines(seq(-1, 5, by=0.1), punif(seq(-1, 5, by=0.1),0,4), type='l', col='red', lty=2)
}
```

We generate data from $\mathcal{U}([0,4])$

Here with probability $1/2$ we get v_1 and with prob. $1/2$ we get v_2. The sample from here is generated by randomly pick one of these two numbers. We have 4 possible samples: $(v_1,v_1), (v_1,v_2), (v_2,v_1), (v_2,v_2)$



Same here, possible samples are $(\#=)\ 4^4$

The true cumulative distribution function is the dashed (---) line. The black are the empirical distr. as the sample size changes. (The black lines are the $\hat{F}$, the distributions that we assume are the actual distributions in the Bootstrap world: in the Bootstrap world we sample from the black lines instead of the dashed lines)

Here we can see an empirical proof of the LLN: the black distribution converges to the true distribution (dashed line)

```
# Real distribution and Bootstrap distribution of the Sample Mean
set.seed(24021979)
n.grid  <- 2^(1:8)
M       <- 10000
B       <- 1000
x.obs.long <- runif(max(n.grid),0,4)

for(n in n.grid){
  T.real <- numeric(M)
  for(m in 1:M){
    x         <- runif(n,0,4)
    T.real[m] <- mean(x)
  }
  x.obs  <- x.obs.long[1:n]
  T.boot <- numeric(B)
  for(b in 1:B){
    x.b       <- sample(x.obs, replace = T)
    T.boot[b] <- mean(x.b)
  }
  plot(ecdf(T.real), main=paste('Sample mean: n =', n), col='red')
  lines(ecdf(T.boot), col='black')
}
```

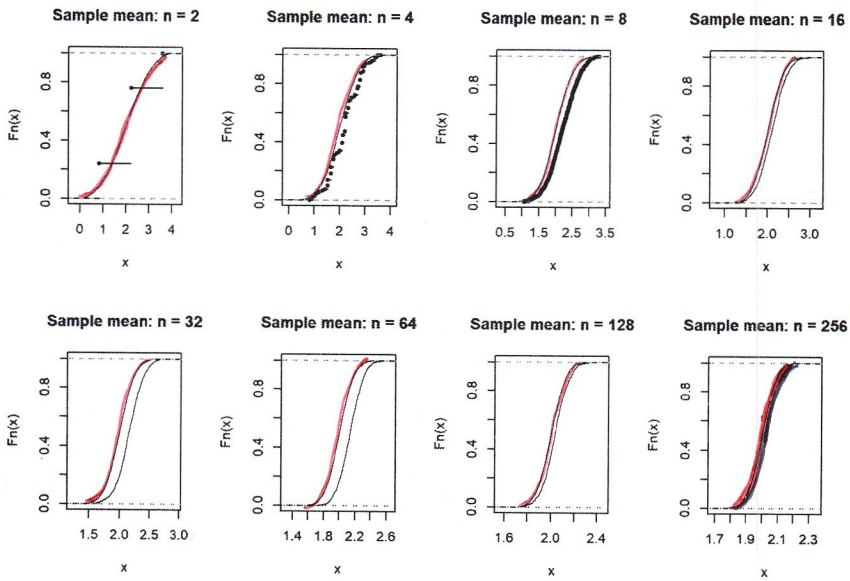
M used for the real distr. estimation

we won't ever use this because we'll never be able to know the true distribution !

→ The goal is not to obtain a distribution (approximate) of the data, the goal is to obtain an approximation of the distribution of the **estimator**!

→ We try to estimate the sample mean distribution using the same (↑) data

Scheme:
we go through different sample sizes and for each sample size we consider the sample that generates the black curve and we randomly pick B (= 1000) times and we generate B bootstrap samples (just randomly picking with replace from the original sample). Then, for each sample we compute the sample mean
⇒ we obtain a vector of B (= 1.000) realizations of the sample mean

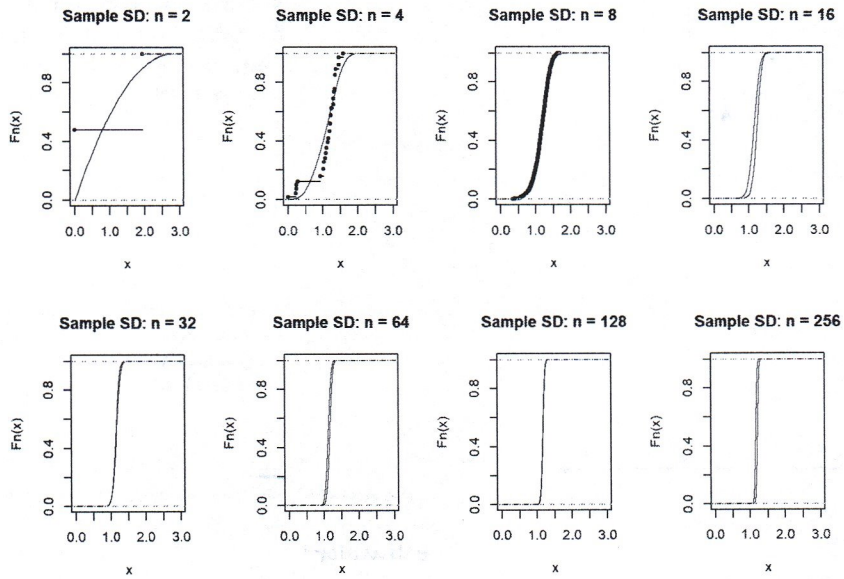


— = true distribution of the sample mean (obtained by picking data from the $\mathcal{U}([0,4])$, something that can't be done in the practice)

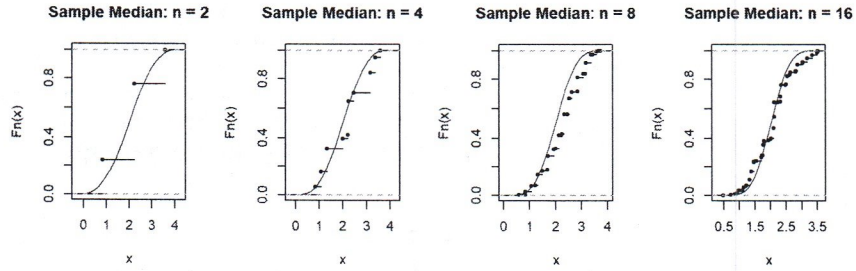
⇒ The bootstrap distribution tends to converge to the true distribution of the sample mean

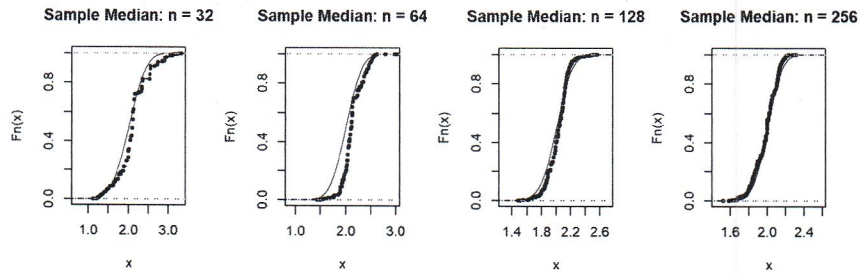
Notice: LLN does not say anything about the convergence rate (there is no way to understand that)

```r
# Real distribution and Bootstrap distribution of the Sample Standard Deviation
set.seed(24021979)
n.grid  <- 2^(1:8)
M       <- 10000
B       <- 1000
x.obs.long <- runif(max(n.grid),0,4)

par(mfrow = c(2,4))
for(n in n.grid){
  T.real <- numeric(M)
  for(m in 1:M){
    x         <- runif(n,0,4)
    T.real[m] <- sd(x)
  }
  x.obs  <- x.obs.long[1:n]
  T.boot <- numeric(B)
  for(b in 1:B){
    x.b       <- sample(x.obs, replace = T)
    T.boot[b] <- sd(x.b)
  }
  plot(ecdf(T.real), main=paste('Sample SD: n =', n), xlim = c(0,3), col='red')
  lines(ecdf(T.boot), col='black')
}
```

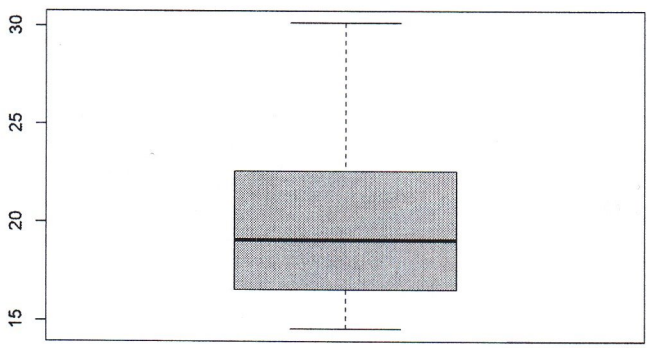



```r
# Real distribution and Bootstrap distribution of the Sample Median
set.seed(24021979)
n.grid  <- 2^(1:8)
M       <- 10000
B       <- 1000
x.obs.long <- runif(max(n.grid),0,4)

par(mfrow = c(2,4))
for(n in n.grid){
  T.real <- numeric(M)
  for(m in 1:M){
    x         <- runif(n,0,4)
    T.real[m] <- median(x)
  }
  x.obs  <- x.obs.long[1:n]
  T.boot <- numeric(B)
  for(b in 1:B){
    x.b       <- sample(x.obs, replace = T)
    T.boot[b] <- median(x.b)
  }
  plot(ecdf(T.real), main=paste('Sample Median: n =', n), col='red')
  lines(ecdf(T.boot), col='black')
}
```






```
### ---------------------------------------------------------------------------
### Bootstrap Estimation of Bias, Variance, and MSE
### ---------------------------------------------------------------------------
# Import data
grades    <- read.table('parziali.txt', header=T)
grades.PI <- grades[,'PI']
x11()
boxplot(grades.PI)
```

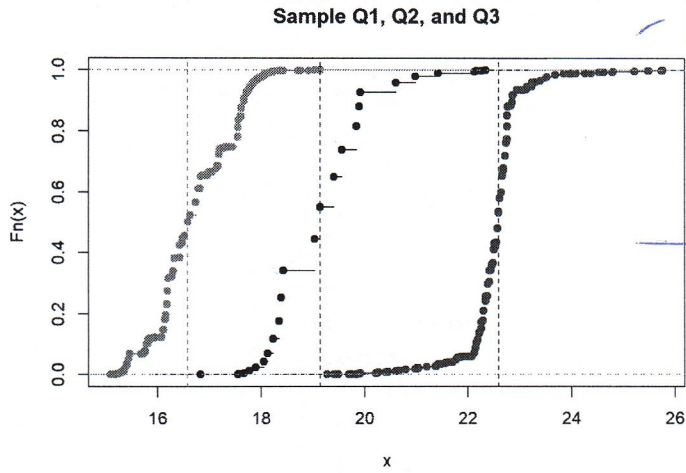


we want to compute: MSE, variance and bias for 3 estimators:

- sample first quantile (Q1)
- sample median (Q2)
- sample third quantile (Q3)

```
# Assessing the "statistical quality" of three sample quartiles
Q1 <- quantile(grades.PI, 0.25)
Q2 <- quantile(grades.PI, 0.50)
Q3 <- quantile(grades.PI, 0.75)

# Computing the bootstrap distribution of three sample quartiles
set.seed(24021979)
B         <- 10000
x.obs     <- grades.PI
T.boot.Q1 <- numeric(B)
T.boot.Q2 <- numeric(B)
T.boot.Q3 <- numeric(B)

for(b in 1:B){
  x.b          <- sample(x.obs, replace = T)
  T.boot.Q1[b] <- quantile(x.b, 0.25)
  T.boot.Q2[b] <- quantile(x.b, 0.50)
  T.boot.Q3[b] <- quantile(x.b, 0.75)
}

plot(ecdf(T.boot.Q1), main='Sample Q1, Q2, and Q3', col='green',
     xlim=range(c(T.boot.Q1, T.boot.Q2, T.boot.Q3)))
lines(ecdf(T.boot.Q2), col='black')
lines(ecdf(T.boot.Q3), col='red')
abline(v=c(Q1,Q2,Q3), col=c('green','black','red'), lty=2)
```

we pick a new bootstrap sample and we compute the first, second and third sample quantiles of this bootstrap sample (and we store them). We repeat this B (= 10.000) times.



bootstrap distributions of the first, second and third Sample quantiles

The vertical dashed lines are the first, second and third quantiles of the original sample (not bootstrap)

These distributions are the distributions that we're gonna use to estimate standard deviation, bias, MSE of the estimators

```
# Bootstrap estimation of the SD of three sample quartiles
sd(T.boot.Q1)
```

```
## [1] 0.7175128
```

```
sd(T.boot.Q2)
```

```
## [1] 0.8211977
```

```
sd(T.boot.Q3)
```

```
## [1] 0.5213033
```

```
# Bootstrap estimation of the Bias of three sample quartiles
mean(T.boot.Q1) - Q1
```

```
##      25%
## 0.128342
```

```
mean(T.boot.Q2) - Q2
```

quantiles of the original samples

```
##      50%
## 0.047755
```

```
mean(T.boot.Q3) - Q3
```

```
##         75%
## -0.0488215
```

```
# Bootstrap estimation of the square root of the MSE of three sample quartiles
sqrt( sd(T.boot.Q1)^2 + (mean(T.boot.Q1) - Q1)^2 )
```

```
##      25%
## 0.7289008
```

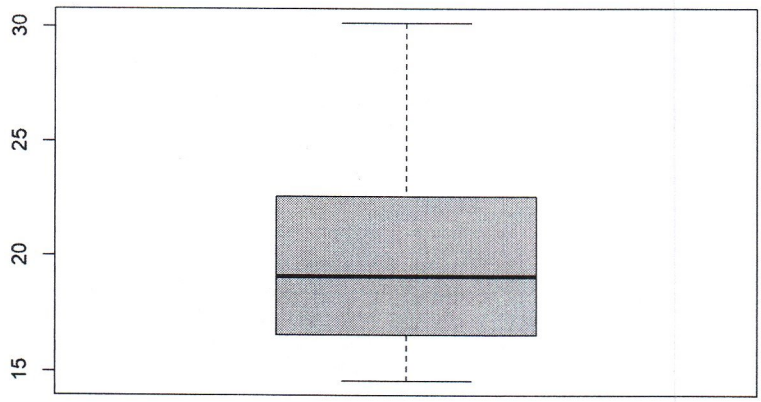
```
sqrt( sd(T.boot.Q2)^2 + (mean(T.boot.Q2) - Q2)^2 )
```

```
##      50%
## 0.8225851
```

```
sqrt( sd(T.boot.Q3)^2 + (mean(T.boot.Q3) - Q3)^2 )
```

```
##      75%
## 0.5235845
```

```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### BOOTSTRAP CONFIDENCE INTERVALS
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
# Import data
grades    <- read.table('parziali.txt', header=T)
grades.PI <- grades[,'PI']
boxplot(grades.PI)
```

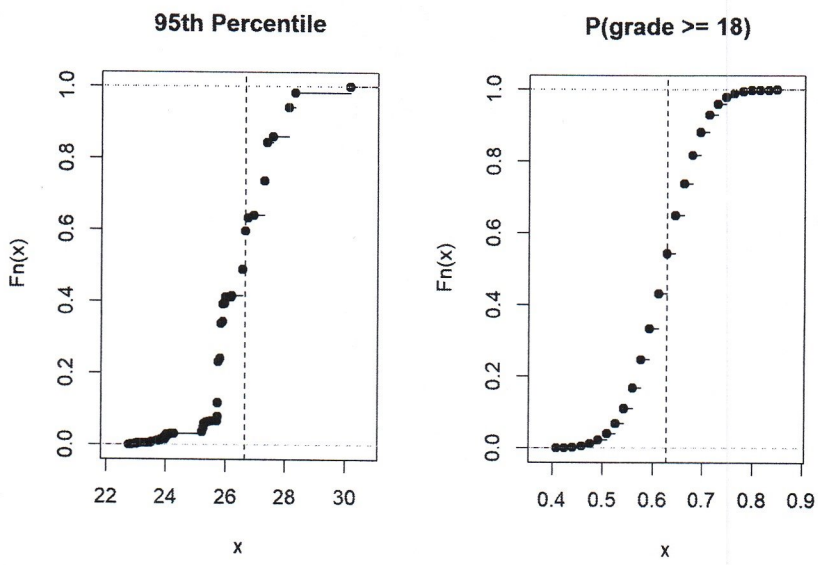


We want to build a confidence interval for:
- the 95-th percentile
- the probability that a student will pass the exam

```
Q95 <- quantile(grades.PI, 0.95)
P18 <- sum(grades.PI >= 18)/length(grades.PI)

# Computing the bootstrap distribution of the estimators
set.seed(24021979)
B          <- 10000
x.obs      <- grades.PI
T.boot.Q95 <- numeric(B)
T.boot.P18 <- numeric(B)

for(b in 1:B){
  x.b           <- sample(x.obs, replace = T)
  T.boot.Q95[b] <- quantile(x.b, 0.95)
  T.boot.P18[b] <- sum(x.b >= 18)/length(x.b)
}

par(mfrow=c(1,2))
plot(ecdf(T.boot.Q95), main='95th Percentile')
abline(v=Q95, lty=2)
plot(ecdf(T.boot.P18), main='P(grade >= 18)')
abline(v=P18, lty=2)
```

Q95 and P18 are the original-sample estimates.

We randomly pick data and for each bootstrap sample we compute (and store) the 95-th percentile and the probability of passing the exam (evaluated as a frequency)



Again, the two dashed vertical lines are the values of the estimators on the original sample

```
### ---------------------------------------------------------------------------
### Confidence Interval: Q95
### ---------------------------------------------------------------------------
alpha  <- 0.05
T.boot <- T.boot.Q95
T.obs  <- Q95

par(mfrow=c(1,1))
plot(ecdf(T.boot))
abline(v=T.obs, lty=2)

# Reverse Percentile Intervals
right.quantile <- quantile(T.boot, 1 - alpha/2)
left.quantile  <- quantile(T.boot, alpha/2)

T.obs
```

```
##    95%
## 26.642
```

```
right.quantile - T.obs
```

```
## 97.5%
## 1.672
```

```
left.quantile  - T.obs
```

```
##   2.5%
## -2.603
```

```
CI.RP <- c(T.obs - (right.quantile - T.obs), T.obs - (left.quantile - T.obs))
CI.RP
```

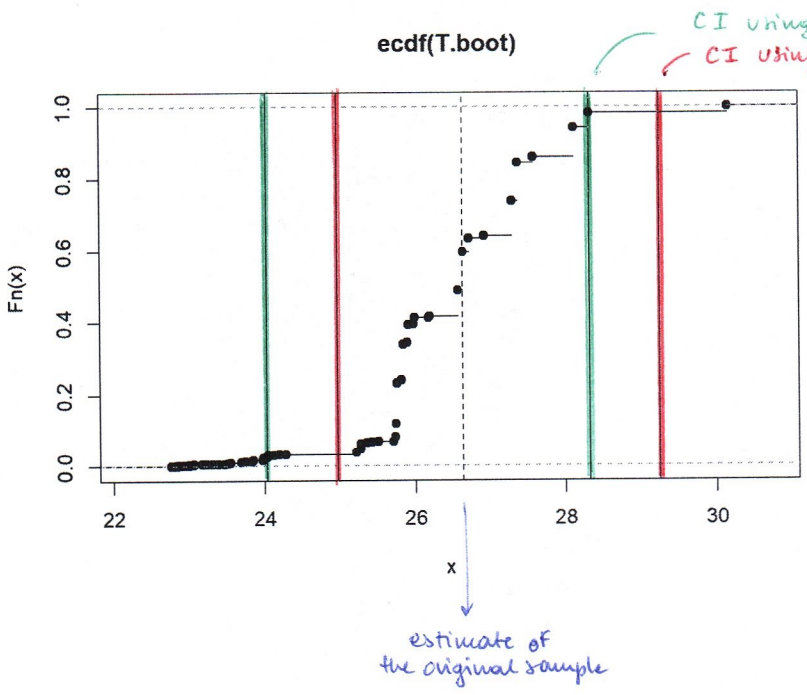
```
##    95%    95%
## 24.970 29.245
```

```
abline(v = CI.RP, col='red')

# Percentile Intervals
right.quantile <- quantile(T.boot, 1 - alpha/2)
left.quantile  <- quantile(T.boot, alpha/2)

CI.P <- c(left.quantile, right.quantile)
CI.P
```

WRONG (the distribution doesn't seem simmetric!)

```
##   2.5%  97.5%
## 24.039 28.314
```

```
abline(v = CI.P)
```



CI using the percentile intervals (WRONG)

CI using the reverse percentiles interval

estimate of the original sample

```r
# Bootstrap t-intervals
# (An estimator of the SD of the estimator of the parameter is needed)
# We cannot implement the bootstrap t-intervals for the 95-percentile because
# we don't have a "nice formula" to estimate the standard deviation of the
# sampled 95- percentile

### ---------------------------------------------------------------------------
### Confidence Interval: P18
### ---------------------------------------------------------------------------
alpha  <- 0.05
T.boot <- T.boot.P18
T.obs  <- P18

plot(ecdf(T.boot))
abline(v=T.obs, lty=2)

# Reverse Percentile Intervals
right.quantile <- quantile(T.boot, 1 - alpha/2)
left.quantile  <- quantile(T.boot, alpha/2)

T.obs
```

```
## [1] 0.6271186
```

```r
right.quantile - T.obs
```

```
##     97.5%
## 0.1186441
```

```r
left.quantile  - T.obs
```

```
##       2.5%
## -0.1186441
```

```r
CI.RP <- c(T.obs - (right.quantile - T.obs), T.obs - (left.quantile - T.obs))
CI.RP
```

```
##     97.5%      2.5%
## 0.5084746 0.7457627
```

```r
abline(v = CI.RP, col='red')

# Percentile Intervals
right.quantile <- quantile(T.boot, 1 - alpha/2)
left.quantile  <- quantile(T.boot, alpha/2)

CI.P <- c(left.quantile, right.quantile)
CI.P
```

```
##      2.5%     97.5%
## 0.5084746 0.7457627
```

```r
abline(v = CI.P)

# Bootstrap t-intervals
# (An estimator of the SD of the estimator of the parameter is needed)
set.seed(24021979)
B     <- 10000
x.obs <- grades.PI
T.boot.stud.P18 <- numeric(B)

P18 <- sum(grades.PI >= 18)/length(grades.PI)

for(b in 1:B){
  x.b       <- sample(x.obs, replace = T)
  P18.boot <- sum(x.b >= 18)/length(x.b)
  T.boot.stud.P18[b] <- ( P18.boot - P18 ) / sqrt(P18.boot * (1 - P18.boot) / length(x.obs))
}

right.t.quantile <- quantile(T.boot.stud.P18, 1 - alpha/2)
left.t.quantile  <- quantile(T.boot.stud.P18, alpha/2)

right.t.quantile
```

here we have a formula to estimate the standard deviation of the estimator

we've bootstrapping this!

```
##    97.5%
## 2.092917
```

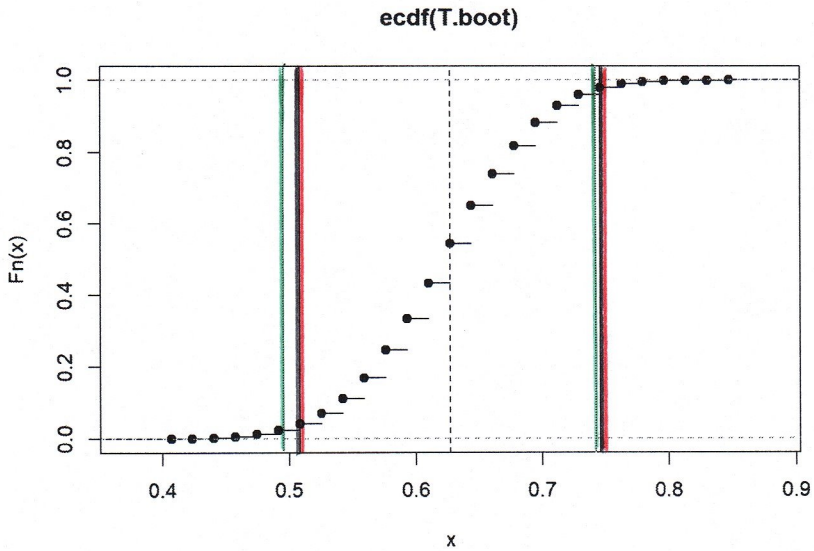
```r
left.t.quantile
```

```
##      2.5%
## -1.822907
```

```
CI.t <- c(P18 - right.t.quantile * sqrt(P18 * (1 - P18) / length(x.obs)),
          P18 - left.t.quantile  * sqrt(P18 * (1 - P18) / length(x.obs)))
CI.t
```

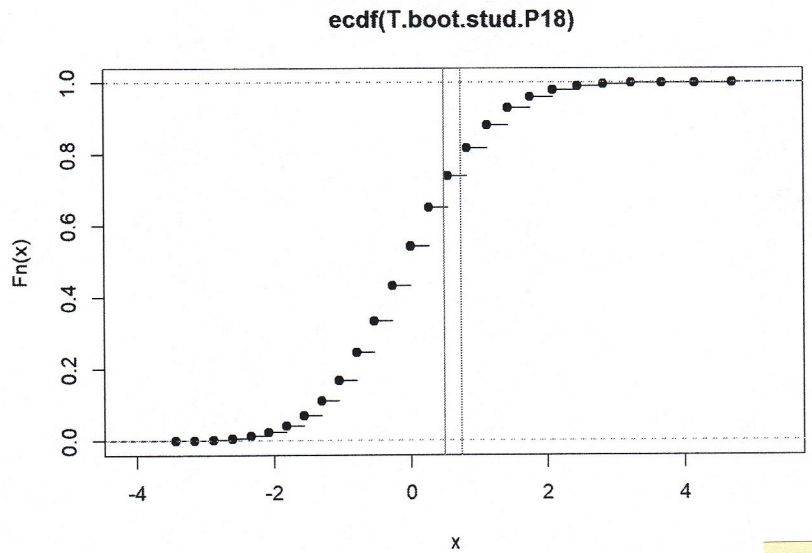
```
##     97.5%      2.5%
## 0.4953578 0.7418808
```

```
abline(v = CI.t, col='green')
```



— reverse percentiles interval
— percentiles interval
(they overlap since the distribution is simmetric)
— t-interval

```
# Plot of the distribution of the standardized variable (with the t-interval)
plot(ecdf(T.boot.stud.P18))
abline(v = CI.t, col='green')
```



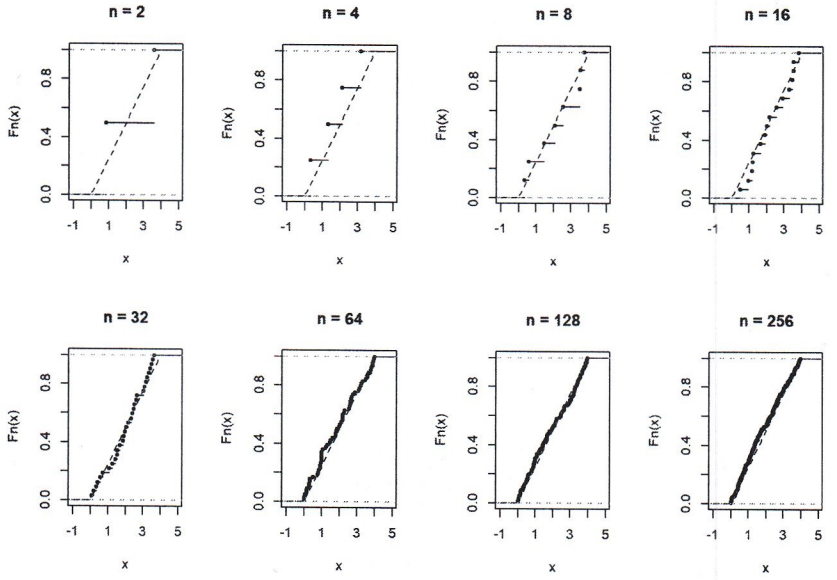
NOTE: we won't treat **bootstrap test**. This is because for a test is sufficient the confidence interval. Setting the level of the confidence interval is equal to setting the level of the test ($IC_{1-\alpha}$ leads to a test of level α). Also the p-value is computed according to the IC: the p-value is the right / left edge of the confidence interval.

In the permutational framework we compute CIs from tests, here we compute tests from CIs.

ATTENZIONE: if we use the reverse percentile interval then we have to use reversed percentiles!

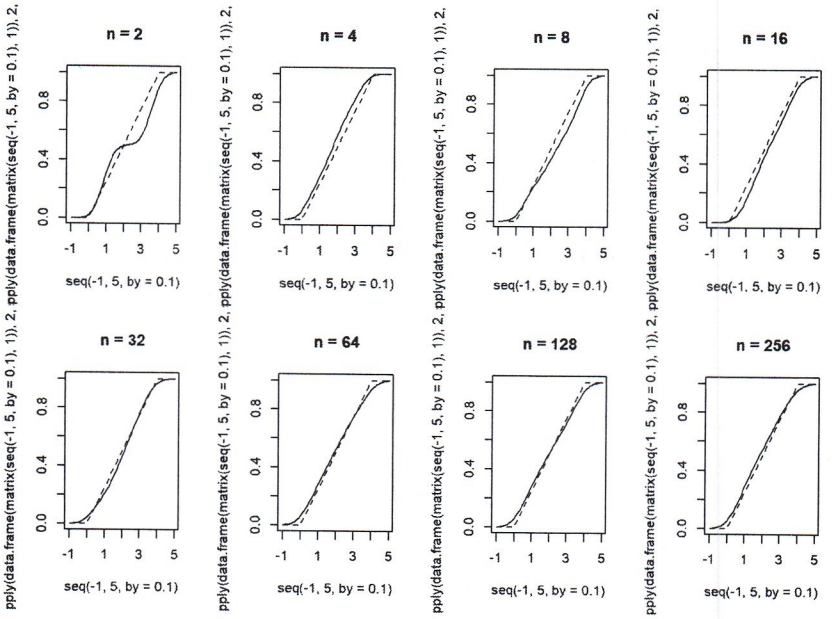
$H_0: \theta = \bar{\theta}$
$H_1: \theta > \bar{\theta}$

```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### BOOTSTRAP METHODS
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
# Simulation: Unif(0,4)

### ---------------------------------------------------------------------------
### Naive Bootstrap
### ---------------------------------------------------------------------------
set.seed(24021979)
n.grid  <- 2^(1:8)

par(mfrow = c(2,4))
for(n in n.grid){
  x <- runif(n,0,4)
  plot(ecdf(x), main=paste('n =', n), xlim = c(-1,5))
  lines(seq(-1, 5, by=0.1), punif(seq(-1, 5, by=0.1),0,4), type='l', col='red', lty=2)
}
```




```
### ---------------------------------------------------------------------------
### Smooth Bootstrap
### ---------------------------------------------------------------------------
set.seed(24021979)
n.grid  <- 2^(1:8)

par(mfrow = c(2,4))
for(n in n.grid){
  x           <- runif(n,0,4)
  smooth.ecdf <- function(x.0){sum(pnorm(x.0, x, 0.5))}
  plot(seq(-1,5,by=0.1), as.numeric(apply(data.frame(matrix(seq(-1,5,by=0.1),1)), 2, smooth.ecdf))/n, main=paste('n =', n),
 xlim = c(-1,5), type='l', ylim=c(0,1))
  lines(seq(-1, 5, by=0.1), punif(seq(-1, 5, by=0.1),0,4), type='l', col='red', lty=2)
}
```

→ sampling from a kernel-density estimate is like sampling from the data and add a small gaussian error



for large values they're equivalent to the naive bootstrap, but for small values it really can improve a lot!
(e.g. n = 2, 4, 8 are sooo different, the smoothing is much more efficient)

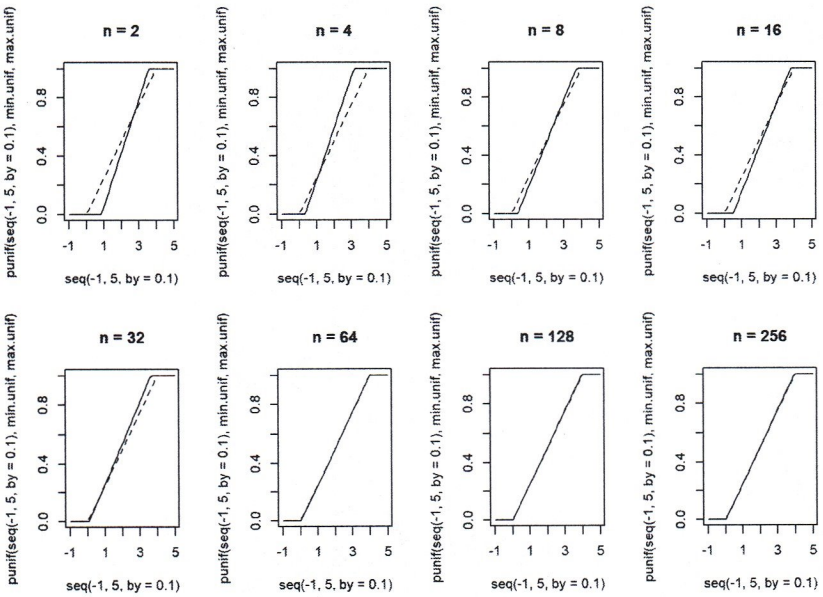
```
### ---------------------------------------------------------------------------
### Parametric Bootstrap
### ---------------------------------------------------------------------------
set.seed(24021979)
n.grid  <- 2^(1:8)

par(mfrow = c(2,4))
for(n in n.grid){
  x          <- runif(n,0,4)
  smooth.ecdf <- function(x.0){sum(pnorm(x.0, x, 0.5))}
  min.unif    <- min(x)
  max.unif    <- max(x)
  plot(seq(-1, 5, by=0.1), punif(seq(-1, 5, by=0.1),min.unif,max.unif), main=paste('n =', n),
       xlim = c(-1,5), type='l', ylim=c(0,1))
  lines(seq(-1, 5, by=0.1), punif(seq(-1, 5, by=0.1),0,4), type='l', col='red', lty=2)
}
```

→ we assume the data to come from $\mathcal{U}([a,b])$ (but we don't know a and b)

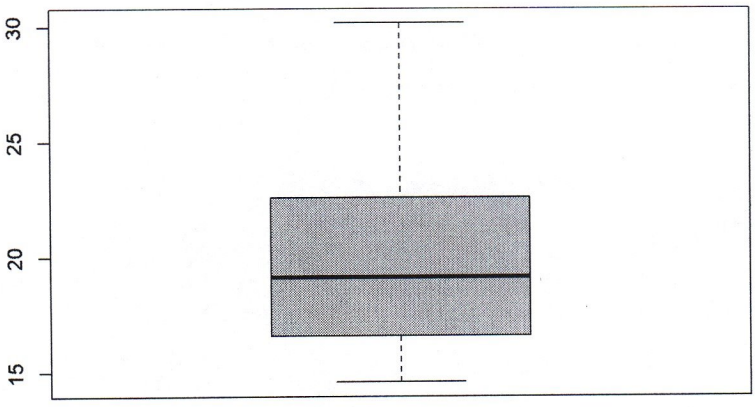
we estimate $\hat{a}_{MLE}$, $\hat{b}_{MLE}$:

$\hat{a}_{MLE}$ = min (values)

$\hat{b}_{MLE}$ = max (values)




```
### ---------------------------------------------------------------------------
### Naive, Smooth, and Parametric Bootstrap Estimation of quartiles
### ---------------------------------------------------------------------------
# Import data
grades    <- read.table('parziali.txt', header=T)
grades.PI <- grades[,'PI']
boxplot(grades.PI)
```

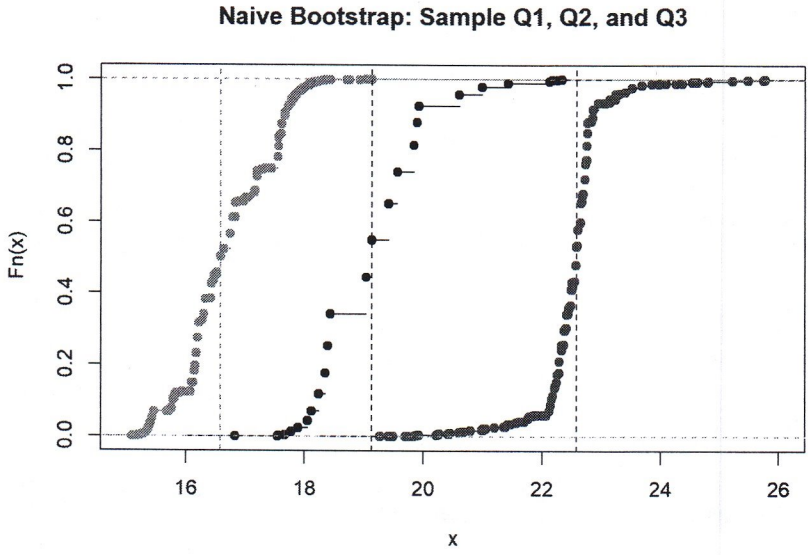


```r
# Assessing the "statistical quality" of three sample quartiles
Q1 <- quantile(grades.PI, 0.25)
Q2 <- quantile(grades.PI, 0.50)
Q3 <- quantile(grades.PI, 0.75)

# Naive Bootstrap
set.seed(24021979)
B         <- 10000
x.obs     <- grades.PI
T.boot.Q1 <- numeric(B)
T.boot.Q2 <- numeric(B)
T.boot.Q3 <- numeric(B)

for(b in 1:B){
  x.b          <- sample(x.obs, replace = T)
  T.boot.Q1[b] <- quantile(x.b, 0.25)
  T.boot.Q2[b] <- quantile(x.b, 0.50)
  T.boot.Q3[b] <- quantile(x.b, 0.75)
}

plot(ecdf(T.boot.Q1), main='Naive Bootstrap: Sample Q1, Q2, and Q3', col='green', xlim=c(15,26))
lines(ecdf(T.boot.Q2), col='black')
lines(ecdf(T.boot.Q3), col='red')
abline(v=c(Q1,Q2,Q3), col=c('green','black','red'), lty=2)
```

original estimates

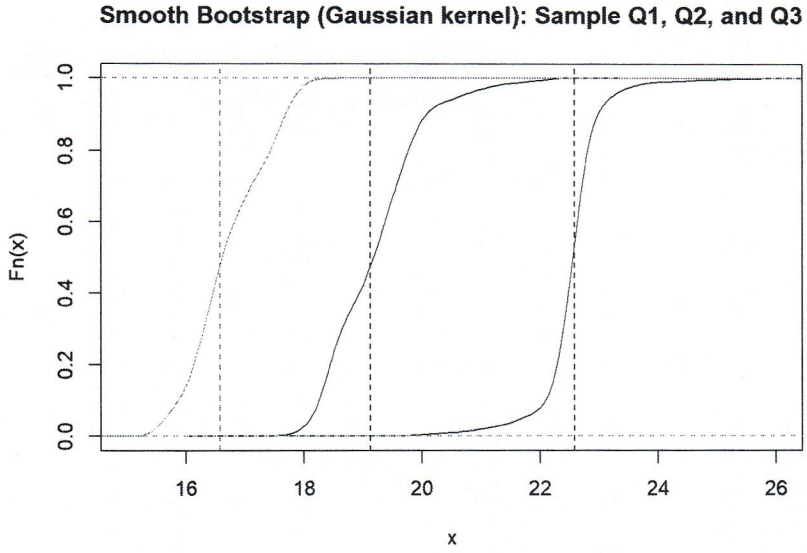
we want to estimate the bootstrap distribution of the first, second and third sample quantiles using: naive, smooth and parametric bootstraps.

→ we re-sample from the dataset

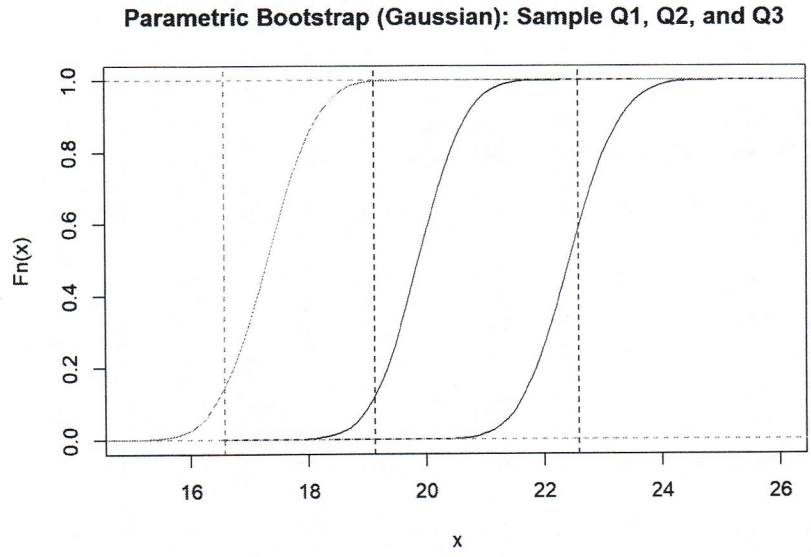



```r
# Smooth Bootstrap
set.seed(24021979)
B         <- 10000
x.obs     <- grades.PI
T.boot.Q1 <- numeric(B)
T.boot.Q2 <- numeric(B)
T.boot.Q3 <- numeric(B)

for(b in 1:B){
  x.b          <- sample(x.obs, replace = T) + rnorm(length(x.obs),0, 0.25)
  T.boot.Q1[b] <- quantile(x.b, 0.25)
  T.boot.Q2[b] <- quantile(x.b, 0.50)
  T.boot.Q3[b] <- quantile(x.b, 0.75)
}

plot(ecdf(T.boot.Q1), main='Smooth Bootstrap (Gaussian kernel): Sample Q1, Q2, and Q3', col='green',  xlim=c(15,26))
lines(ecdf(T.boot.Q2), col='black')
lines(ecdf(T.boot.Q3), col='red')
abline(v=c(Q1,Q2,Q3), col=c('green','black','red'), lty=2)
```

→ we randomly pick one of the datum and we add a small (normal) noise




```r
# Parametric Bootstrap (assuming Gaussian data)
set.seed(24021979)
B         <- 10000
x.obs     <- grades.PI
T.boot.Q1 <- numeric(B)
T.boot.Q2 <- numeric(B)
T.boot.Q3 <- numeric(B)

for(b in 1:B){
  x.b          <- rnorm(length(x.obs), mean(x.obs), sd(x.obs))
  T.boot.Q1[b] <- quantile(x.b, 0.25)
  T.boot.Q2[b] <- quantile(x.b, 0.50)
  T.boot.Q3[b] <- quantile(x.b, 0.75)
}

plot(ecdf(T.boot.Q1), main='Parametric Bootstrap (Gaussian): Sample Q1, Q2, and Q3', col='green',  xlim=c(15,26))
lines(ecdf(T.boot.Q2), col='black')
lines(ecdf(T.boot.Q3), col='red')
abline(v=c(Q1,Q2,Q3), col=c('green','black','red'), lty=2)
```

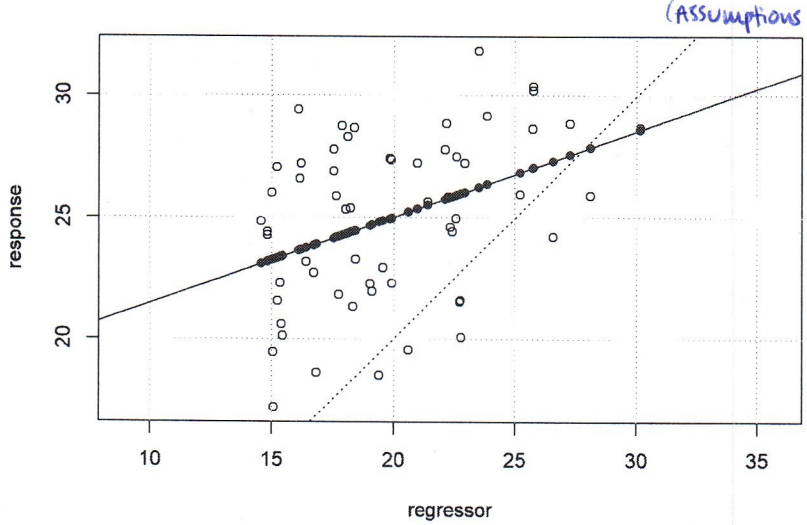


Here there is a little bias.
For instance: the 3 curves are equally shifted (even if in the other 2 methods (naive and smooth) they don't seem equally shifted).
Moreover we see that the distribution of Q2 is a little right-shifted w.r.t. its original sample estimate

```
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
### BOOTSTRAP REGRESSION
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
# Import data
grades    <- read.table('parziali.txt', header=T)
response  <- grades[,'PII']
regressor <- grades[,'PI']

# Plot data
plot(regressor, response, asp=1)
grid()
abline(0,1, lty=3)

# Fit a linear model
fm <- lm(response ~ regressor)

# plot the regression line and the fitted vaues
abline(coefficients(fm), col='red')
points(regressor, fitted(fm), col='red', pch=16)
```

Goal: predict the grades of the 2nd exams starting from the grades of the 1st

(ASSUMPTIONS : errors are gaussian distributed)




```
### ------------------------------------------------------------------------
### Classical parametric inference
### ------------------------------------------------------------------------
summary(fm)
```

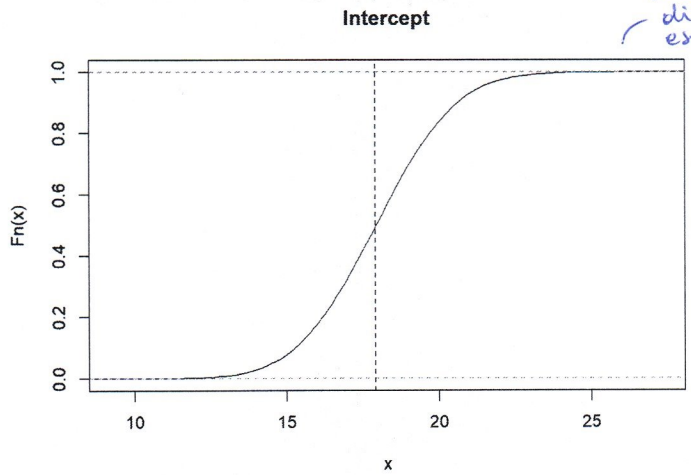
```
##
## Call:
## lm(formula = response ~ regressor)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.2837 -2.3859  0.1442  2.6246  5.8089
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  17.9174     2.1535   8.320 2.03e-11 ***
## regressor     0.3543     0.1064   3.329  0.00153 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.161 on 57 degrees of freedom
## Multiple R-squared:  0.1628, Adjusted R-squared:  0.1481
## F-statistic: 11.08 on 1 and 57 DF,  p-value: 0.001532
```

we want to estimate the standard deviations of the estimators and the confidence intervals of the intercept and the slope (we're still assuming gaussianity of the errors)

```
### ---------------------------------------------------------------------
### Bootstrap Inference
### ---------------------------------------------------------------------
# Compute residuals and fitted values
fitted.obs <- fitted(fm)
res.obs    <- residuals(fm)

b0.obs <- coefficients(fm)[1]
b1.obs <- coefficients(fm)[2]

set.seed(24021979)
B         <- 10000
T.boot.b0 <- numeric(B)
T.boot.b1 <- numeric(B)

for(b in 1:B){
  response.b   <- fitted.obs + sample(res.obs, replace = T)
  fm.b         <- lm(response.b ~ regressor)
  T.boot.b0[b] <- coefficients(fm.b)[1]
  T.boot.b1[b] <- coefficients(fm.b)[2]
}

plot(ecdf(T.boot.b0), main='Intercept')
abline(v=b0.obs, lty=2)
```

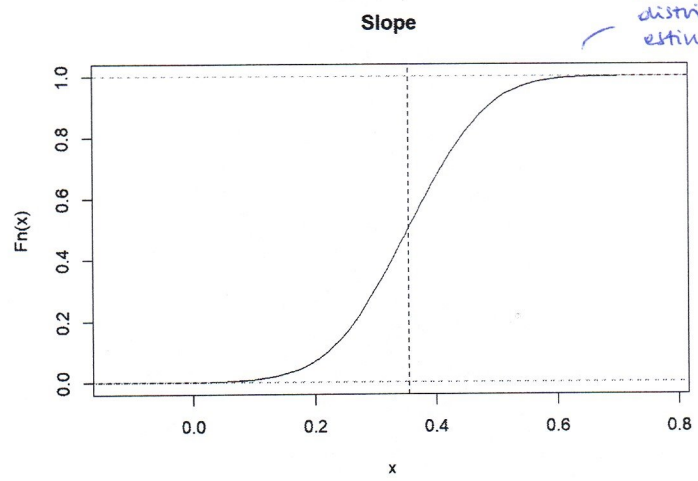
we generate new samples simply by sampling from the residuals (we use a naive bootstrap approach). We generated the bootstrap sample, we obtained new values for the response; we re-fit the model and we obtain the bootstrap estimate associated to that bootstrapped sample of β_0 and β_1

(bootstrap estimated) distribution of the OLS estimator of the intercept




```
plot(ecdf(T.boot.b1), main='Slope', col='red')
abline(v=b1.obs, lty=2, col='red')
```

(bootstrap estimated) distribution of the OLS estimator of the slope




```
# Bootstrap estimates of standard deviations of the OLS estimates (NB: without any analytical computation)
sd(T.boot.b0)
```

```
## [1] 2.097613
```

* vs. 2.1535

```
sd(T.boot.b1)
```

```
## [1] 0.1039132
```

* vs 0.1064

```
cov(T.boot.b0, T.boot.b1)
```

```
## [1] -0.2139072
```

```
# Reverse Percentile Interval of intercept
alpha             <- 0.05
right.quantile.b0 <- quantile(T.boot.b0, 1 - alpha/2)
left.quantile.b0  <- quantile(T.boot.b0, alpha/2)

b0.obs
```

```
## (Intercept)
##    17.91737
```

```
right.quantile.b0 - b0.obs
```

```
##    97.5%
## 4.185143
```

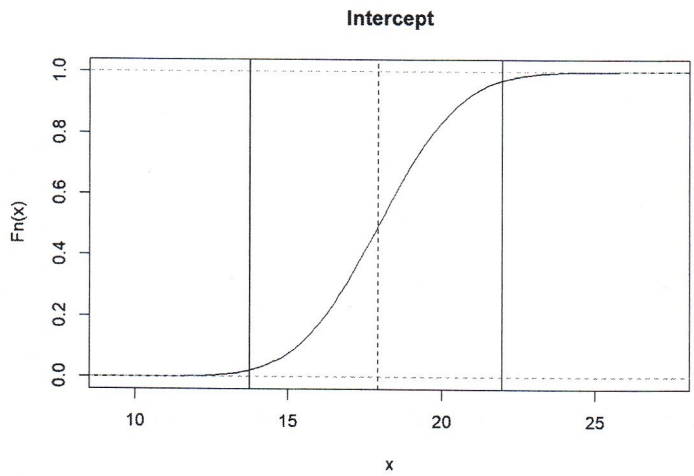
```
left.quantile.b0  - b0.obs
```

```
##     2.5%
## -4.04383
```

```
CI.RP.b0 <- c(b0.obs - (right.quantile.b0 - b0.obs), b0.obs - (left.quantile.b0 - b0.obs))
CI.RP.b0
```

```
## (Intercept) (Intercept)
##    13.73222    21.96120
```

```
plot(ecdf(T.boot.b0), main='Intercept')
abline(v = b0.obs, lty=2)
abline(v = CI.RP.b0)
```




```
# Reverse Percentile Interval of slope
alpha             <- 0.05
right.quantile.b1 <- quantile(T.boot.b1, 1 - alpha/2)
left.quantile.b1  <- quantile(T.boot.b1, alpha/2)

b1.obs
```

```
## regressor
## 0.3542663
```

```
right.quantile.b1 - b1.obs
```

```
##     97.5%
## 0.2011228
```

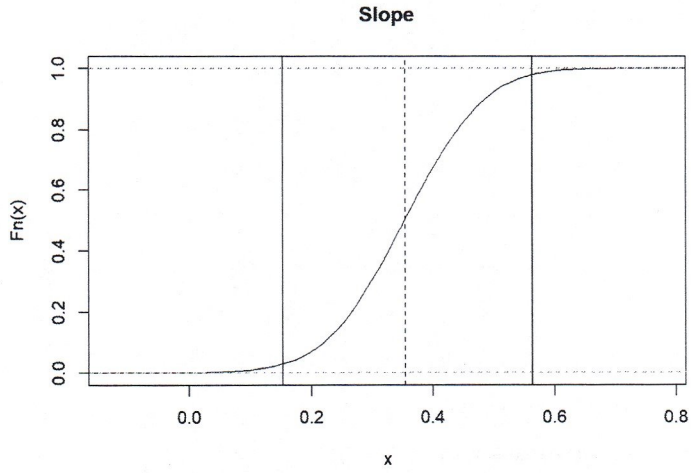
```
left.quantile.b1  - b1.obs
```

```
##       2.5%
## -0.2089994
```

```
CI.RP.b1 <- c(b1.obs - (right.quantile.b1 - b1.obs), b1.obs - (left.quantile.b1 - b1.obs))
CI.RP.b1
```

```
## regressor regressor
## 0.1531436 0.5632657
```

```
plot(ecdf(T.boot.b1), main='Slope', col='red')
abline(v = b1.obs, lty=2, col='red')
abline(v = CI.RP.b1, col='red')
```




```
# Reverse Percentile Interval of conditional mean at x0 = 24
alpha                  <- 0.05
x0                     <- 24
mean.x0.obs            <- b0.obs + b1.obs*x0
T.boot.mean.x0         <- T.boot.b0 + T.boot.b1*x0
right.quantile.mean.x0 <- quantile(T.boot.mean.x0, 1 - alpha/2)
left.quantile.mean.x0  <- quantile(T.boot.mean.x0, alpha/2)

mean.x0.obs
```

```
## (Intercept)
##    26.41976
```

```
right.quantile.mean.x0 - mean.x0.obs
```

```
##    97.5%
## 1.137323
```

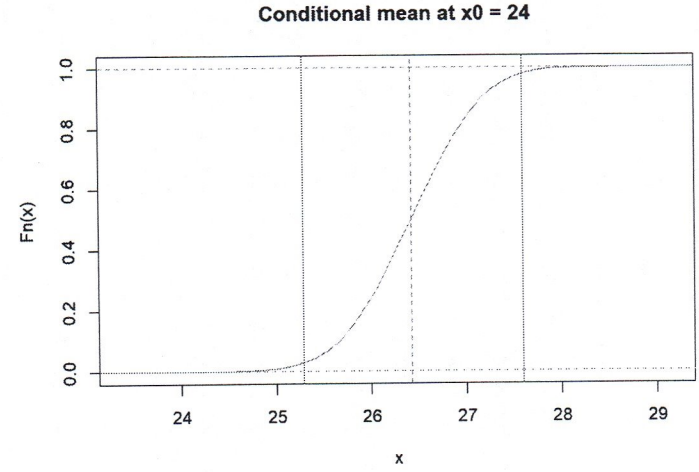
```
left.quantile.mean.x0  - mean.x0.obs
```

```
##      2.5%
## -1.177357
```

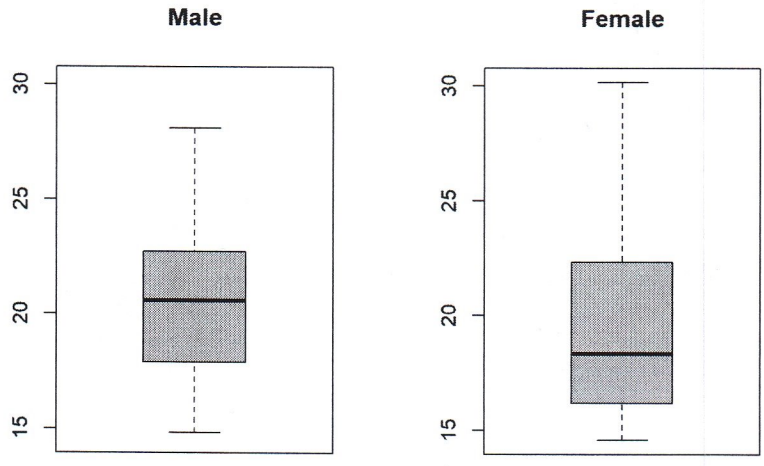
```
CI.RP.mean.x0 <- c(mean.x0.obs - (right.quantile.mean.x0 - mean.x0.obs), mean.x0.obs - (left.quantile.mean.x0 - mean.x0.ob
s))
CI.RP.mean.x0
```

```
## (Intercept) (Intercept)
##    25.28244    27.59712
```

```
plot(ecdf(T.boot.mean.x0), main='Conditional mean at x0 = 24', col='green')
abline(v = mean.x0.obs, lty=2, col='green')
abline(v = CI.RP.mean.x0, col='green')
```



```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### BOOTSTRAP FOR INDEPENDENT SAMPLES
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
# Import data
grades <- read.table('parziali.txt', header=T)
gender <- read.table('sesso.txt', header=T)
x1     <- grades[gender == 1,'PI']
x2     <- grades[gender == 0,'PI']

# Plot data
par(mfrow=c(1,2))
boxplot(x1, ylim=range(c(x1,x2)), main = 'Male')
boxplot(x2, ylim=range(c(x1,x2)), main = 'Female')
```




```
# Computing the bootstrap distribution of the difference of the two sample medians
set.seed(24021979)
B              <- 10000
x1.obs         <- x1
x2.obs         <- x2
diff.Q2.obs    <- quantile(x1, 0.50) - quantile(x2, 0.50)
T.boot.diff.Q2 <- numeric(B)

for(b in 1:B){
  x1.b              <- sample(x1.obs, replace = T)
  x2.b              <- sample(x2.obs, replace = T)
  T.boot.diff.Q2[b] <- quantile(x1.b, 0.50) - quantile(x2.b, 0.50)
}

par(mfrow=c(1,1))
plot(ecdf(T.boot.diff.Q2), main='Sample Median Male - Sample Median Female')
abline(v = diff.Q2.obs, lty=2)

# RP intervals
alpha          <- 0.05
right.quantile <- quantile(T.boot.diff.Q2, 1 - alpha/2)
left.quantile  <- quantile(T.boot.diff.Q2, alpha/2)

diff.Q2.obs
```

← original sample

```
##  50%
## 2.24
```

```
right.quantile - diff.Q2.obs
```

```
## 97.5%
## 2.665
```

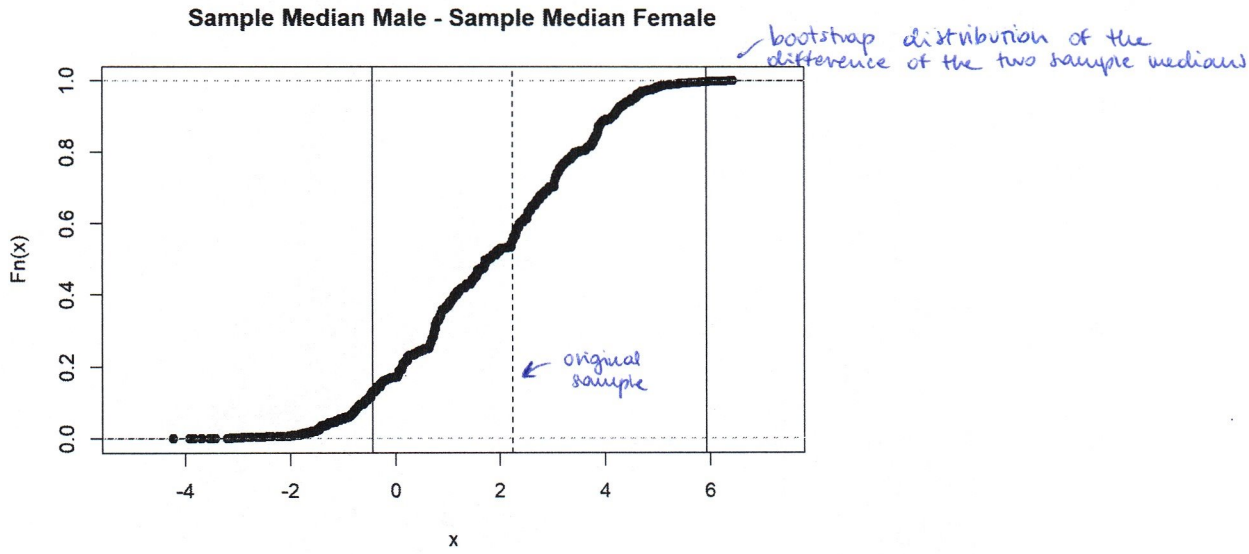
```
left.quantile  - diff.Q2.obs
```

```
##  2.5%
## -3.69
```

```
CI.RP <- c(diff.Q2.obs - (right.quantile - diff.Q2.obs), diff.Q2.obs - (left.quantile - diff.Q2.obs))
CI.RP
```

```
##     50%     50%
## -0.425   5.930
```

```
abline(v = CI.RP)
```



```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### BOOTSRAP NONLINEAR REGRESSION
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
rm(list=ls())
library(progress)
# Import data
data("Orange")
head(Orange)
```

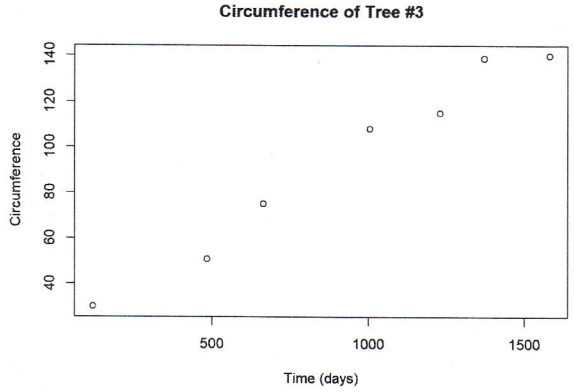
(data about the growth of orange trees)
(we have 5 trees but we select only one of them)

```
##   Tree  age circumference
## 1    1  118            30
## 2    1  484            58
## 3    1  664            87
## 4    1 1004           115
## 5    1 1231           120
## 6    1 1372           142
```

in days

(all trees)

```
O_3 = subset(Orange,Orange$Tree==3)
attach(O_3)

# Let's have a look at the data
plot(age, circumference, main='Circumference of Tree #3', xlab='Time (days)', ylab='Circumference')
```



Taking a closer look, the data does not seem linear. It seems to be possible to model them with a logistic curve.
(We assume that the trees grow with a logistic curve)

```
# The curve looks like a Logistic one... Let's try to estimate it

logistic = function(t,L,k,midpoint){L/(1+exp((midpoint-t)/k))}

# t is the independent variable, L is the asymptote parameter, k is the growth rate
# and midpoint is where the derivative is at it's maximum level
# (the peak, if you want an epidemic analogy.)
# We will need nls to fit this
model = nls(circumference ~ logistic(age,L,k,midpoint),
            start=list(L=150, k=500, midpoint=700))

summary(model)
```

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

(in the code we write t instead of x and midpoint instead of x_0)

we want to minimize the sum of square errors (residuals) in a non linear fashion ⟹ **nls**

NONLINEAR LEAST SQUARES
(it's a gradient descent algorithm and so we need starting values)

we try to estimate (un po' ad occhio, come facevamo con stat. geografica) some values that seems to be good

```
##
## Formula: circumference ~ logistic(age, L, k, midpoint)
##
## Parameters:
##          Estimate Std. Error t value Pr(>|t|)
## L          158.83      15.09  10.529  0.00046 ***
## k          400.95      74.33   5.394  0.00571 **
## midpoint   734.84     102.59   7.163  0.00201 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.258 on 4 degrees of freedom
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 6.516e-06
```

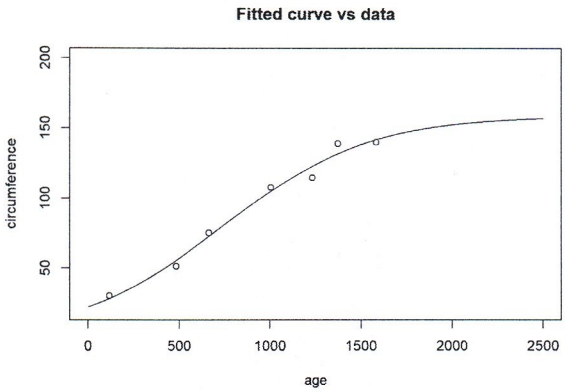
attempt on testing on the parameters (we have no idea about the parameters distributions)

we reach convergence, good!

we shouldn't really use it because we know nothing about the assumptions

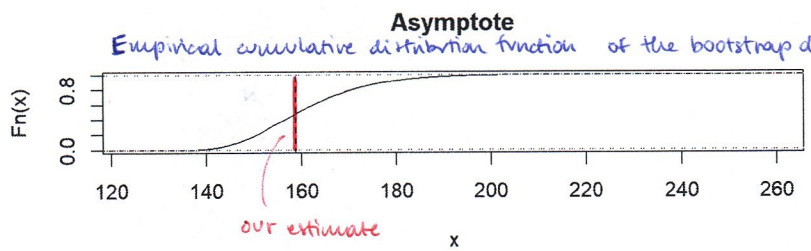
```
# Let's see the estimated curve
age_fine            = seq(0,2500)
circumference_fine = predict(model,list(age=age_fine))

plot(age,circumference,main='Fitted curve vs data',xlim=c(0,2500),ylim=c(20,200))
lines(age_fine,circumference_fine,col='red')
```

we predict the value of the response on a finite grid and we extend it in time



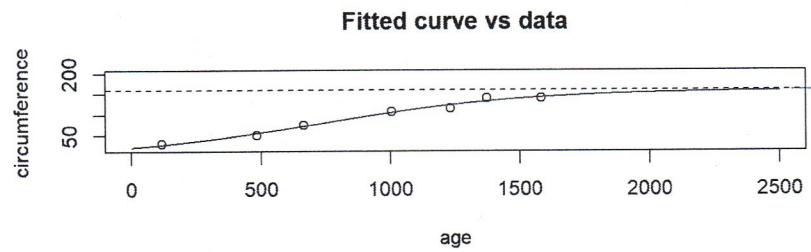
We can see that the logistic was a good choice: bias and variance seem both good.

```
# How can we make inference about these parameters?
# We act in the very same way as before Bootstrap Inference

# Compute residuals
fitted.obs = predict(model)
res.obs    = circumference-fitted.obs

# Let's build confidence intervals for the asymptote parameters (L)
L.obs = summary(model)$parameters[1,1]
set.seed(27081991)
B         = 10000
T.boot.L  = numeric(B)
formula.b = response.b ~ logistic(age,L,k,midpoint)

pb <- progress_bar$new(
  format = "  processing [:bar] :percent eta: :eta",
  total  = B, clear = FALSE)

for(b in 1:B) {
  response.b  <- fitted.obs + sample(res.obs, replace = T)
  fm.b        <- nls(formula.b ,start=list(L=150, k=500, midpoint=700))
  sm.b        = summary(fm.b)
  T.boot.L[b] = sm.b$parameters[1,1]
  pb$tick()
}

par(mfrow=c(2,1))
plot(ecdf(T.boot.L), main='Asymptote')
abline(v=L.obs, lty=2)

plot(age,circumference,main='Fitted curve vs data',xlim=c(0,2500),ylim=c(20,200))
lines(age_fine,circumference_fine,col='red')
abline(h=L.obs, lty=2)
```



Empirical cumulative distribution function of the bootstrap distribution:

Our estimate is $\sim \frac{1}{2}$, it's something good



← estimate of the asymptote (it seems more than reasonable)

Having the bootstrap. distr. we can compute the standard deviation and the bias of the estimator:

```
# Bootstrap estimates of standard deviations of the NLS estimates
# (NB: without any analytical computation)
sd(T.boot.L)
```

```
## [1] 12.69034
```

```
# Bootstrap estimation of bias
mean(T.boot.L)-L.obs
```

```
## [1] 2.34003
```

Moreover, having the distribution we can compute the confidence intervals (through percentiles)
method: reverse percentile interval

```
# Reverse Percentile Interval of peak
alpha <- 0.05

right.quantile.L <- quantile(T.boot.L, 1 - alpha/2)
left.quantile.L  <- quantile(T.boot.L, alpha/2)

L.obs
```

```
## [1] 158.8293
```

```
right.quantile.L - L.obs
```

```
##    97.5%
## 32.15798
```

```
left.quantile.L  - L.obs
```

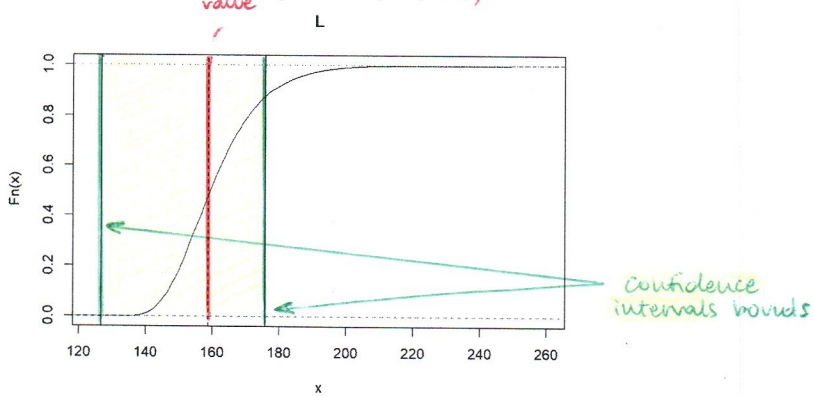
```
##      2.5%
## -17.00435
```

```
CI.RP.L <- c(L.obs - (right.quantile.L - L.obs), L.obs - (left.quantile.L - L.obs))
CI.RP.L
```

```
##    97.5%     2.5%
## 126.6713 175.8337
```

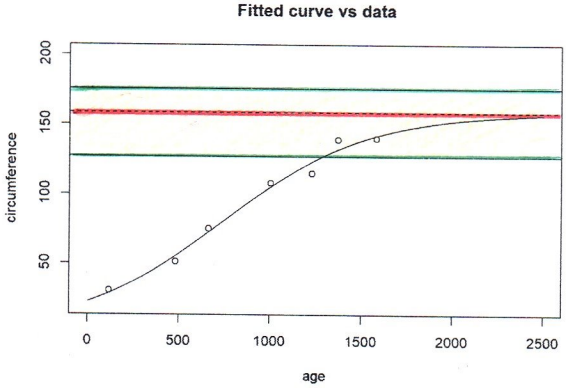
```
plot(ecdf(T.boot.L), main='L')
abline(v = L.obs, lty=2)
abline(v = CI.RP.L)
```

our value (estimated value)



confidence intervals bounds

```
plot(age,circumference,main='Fitted curve vs data',xlim=c(0,2500),ylim=c(20,200))
lines(age_fine,circumference_fine,col='red')
abline(h=L.obs, lty=2)
abline(h = CI.RP.L)
```



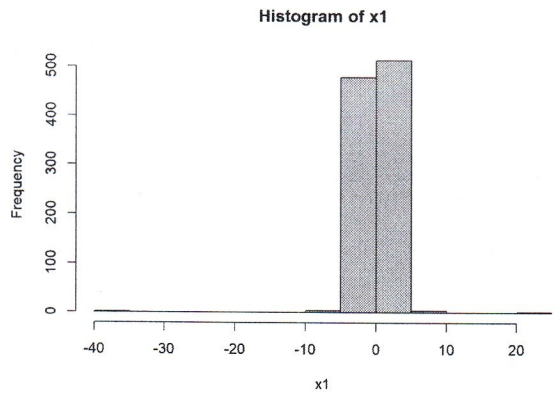

```
# How you would've done the same thing in a permutational framework?
```

Usually we prefer bootstrap for **CI** and permutational inference for testing (and to, **p-values**)

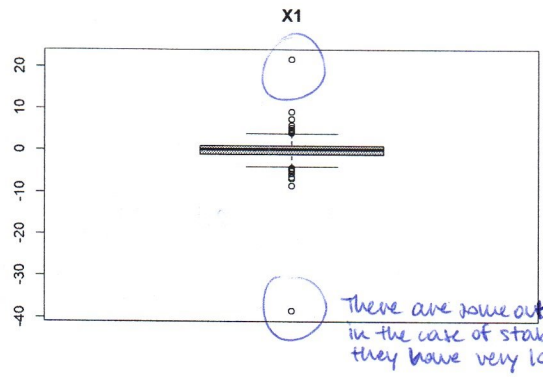
However, it's possible to obtain both CI and p-values from both bootstrap and permutational inference.

```
### -----------------------------------------------------------------------------
### -----------------------------------------------------------------------------
### BOOTSTRAP TESTS AND P-VALUES
### -----------------------------------------------------------------------------
### -----------------------------------------------------------------------------
# Let's generate some data:
set.seed(2781991)
x1 = stabledist::rstable(1000,1.8,0)

# Plot data
hist(x1)
```

(from a stable distribution)




```
boxplot(x1, main = 'X1')
```



There are some outliers, this is expected in the case of stable distributions because they have very long tails

```
# We want to test H_0: median(x1)=0 vs median(x1) \neq 0

# Computing the bootstrap distribution of the median of this distribution:
B      <- 1e5
T.obs  <- median(x1)
T.obs
```

(median because it's more robust than the mean)

we don't know the distribution of the median so we run permutation or bootstrap tests

```
## [1] 0.08967783
```

```
T.boot <- numeric(B)

library(pbapply)
library(parallel)
cl      = makeCluster(2)
wrapper = function(dummy){T.boot <- median(sample(x1, replace = T))}
clusterExport(cl=cl,list('x1'))
T.boot  = pbsapply(T.boot,wrapper,cl=cl)

plot(ecdf(T.boot), main='Sample median')
abline(v = T.obs, lty=2)

# We know very well how to compute confidence intervals
alpha <- 0.1

right.quantile <- quantile(T.boot, 1 - alpha/2)
left.quantile  <- quantile(T.boot, alpha/2)

T.obs
```

useless but we need to have it to avoid errors

This code is to estimate in a fast parallel fashion the bootstrap distribution of the median (we're running 10^5 iterations of the bootstrap algorithm in these lines)

(This code is just for computer in a faster way)

(reverse percentile intervals)

```
## [1] 0.08967783
```

```
right.quantile - T.obs
```

```
##       95%
## 0.0885416
```

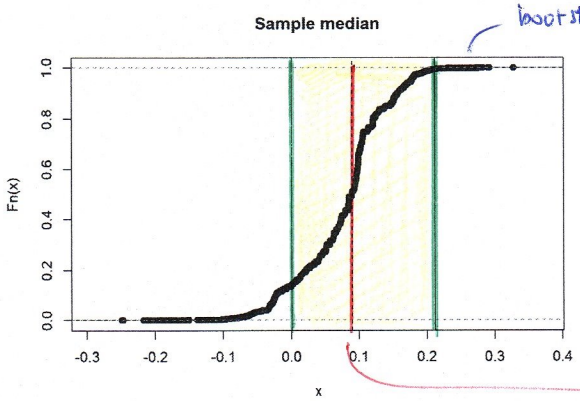
```
left.quantile  - T.obs
```

```
##         5%
## -0.1218363
```

```
CI.RP <- c(T.obs - (right.quantile - T.obs), T.obs - (left.quantile - T.obs))
CI.RP
```

```
##          95%           5%
## 0.001136235 0.211514144
```

```
abline(v = CI.RP)
```



bootstrap distribution of the sample median

median of our (original) sample

```
# How to find p-values? the smallest alpha level for which I reject H0.
# How to find it?
#Grid search:
alpha_grid = seq(0.001,0.5,by=0.001)
alpha_grid
```

→ smallest α for which CI_α does **not** contain the median of the sample. For example: the previous CI with $\alpha = 0.05$ contains the median of the original sample ($\Rightarrow$ we do **not** reject at level $\alpha = 0.05$ H_0: median = 0)

```
##   [1] 0.001 0.002 0.003 0.004 0.005 0.006 0.007 0.008 0.009 0.010 0.011 0.012
##  [13] 0.013 0.014 0.015 0.016 0.017 0.018 0.019 0.020 0.021 0.022 0.023 0.024
##  [25] 0.025 0.026 0.027 0.028 0.029 0.030 0.031 0.032 0.033 0.034 0.035 0.036
##  [37] 0.037 0.038 0.039 0.040 0.041 0.042 0.043 0.044 0.045 0.046 0.047 0.048
##    ..
## [457] 0.457 0.458 0.459 0.460 0.461 0.462 0.463 0.464 0.465 0.466 0.467 0.468
## [469] 0.469 0.470 0.471 0.472 0.473 0.474 0.475 0.476 0.477 0.478 0.479 0.480
## [481] 0.481 0.482 0.483 0.484 0.485 0.486 0.487 0.488 0.489 0.490 0.491 0.492
## [493] 0.493 0.494 0.495 0.496 0.497 0.498 0.499 0.500
```

```
CI_calc = function(alpha_level){
  right.quantile <- quantile(T.boot, 1 - alpha_level/2)
  left.quantile  <- quantile(T.boot, alpha_level/2)
  out            = c(T.obs - (right.quantile - T.obs), T.obs - (left.quantile - T.obs))
  names(out) = c('lwr','upr')
  return(out)
}

CI_list = pblapply(alpha_grid,CI_calc)
CI_mat  = dplyr::bind_rows(CI_list)
check   = CI_mat[,1]>0 | CI_mat[,2]<0
(alpha_grid[check])[1]
```

← here we're specifically checking H_0: median = 0 (we're checking if 0 is contained in the CI_α)

```
## [1] 0.082
```

p-value

alpha_grid[check] returns all the alphas for which $0 \notin CI_\alpha$
(alpha_grid[check])[1] returns the first alpha for which $0 \notin CI_\alpha$
and, because of how we created the grid, the first alpha is also the smallest alpha ($\Rightarrow$ p-value)

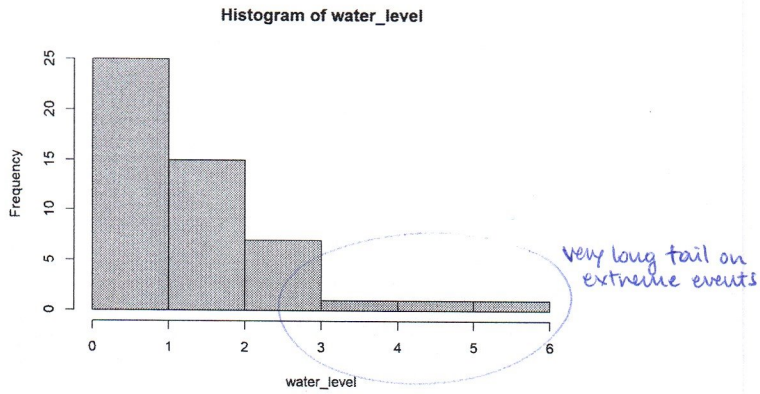
Exam-like Exercices

```
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
### Problem 1
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
### The local administration of Torgnon, a small municipality in Aosta Valley,
### has given you the task to assess the probability of a flood coming from the small river
### that crosses the village. To do so, they given you data about the maximum water level
### of the past 50 years. The major is only willing to accept a 5% probability of flood.
### So your wall should be high at least as the 95th percentile of the maximums distribution.
###   1. Assess the statistical quality of the sample 95th percentile,
###      and compute a 95 percent confidence interval for it
###   2. After some study, you've discovered that the maximum value of level of water
###      is usually distributed as a lognormal: How can you use this information?
###      Has your estimate of the sample 95th percentile improved?

### ------------------------------------------------------------------------------
### 1.
### ------------------------------------------------------------------------------
# Load the data
load('water_level.rda')
hist(water_level)
```

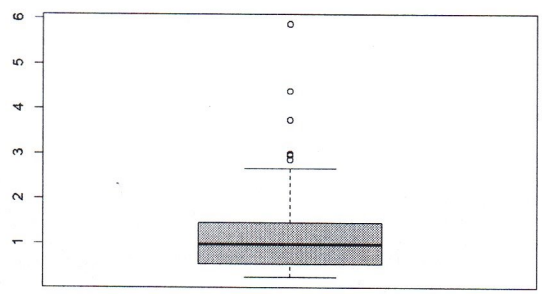
Hint: asking for a **CI** and not mentioning a test means "use **bootstrap**" !

This is asking a 95% CI for the 95-th percentile of the distribution of the max.

"Statistical quality" = compute some metrics about that quantity

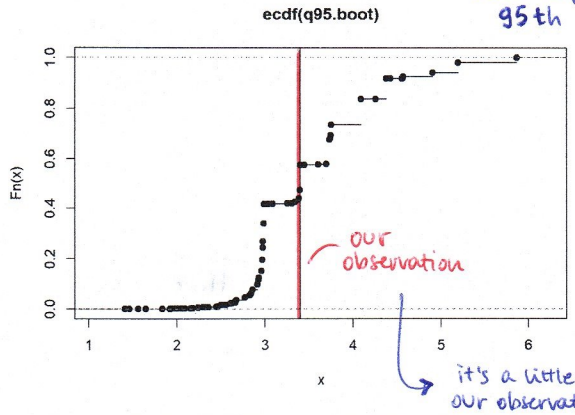



```
boxplot(water_level) # right skewed distribution... as expected.
```




```
# To assess the quality of the estimator and to compute a CI the way to go is by bootstrapping.
B          = 10000
water.obs = water_level

q95.obs  = quantile(water.obs,0.95)
q95.boot = numeric(B)
for(b in 1:B){
  water.b      = sample(water.obs, replace = T)
  q95.boot[b] = quantile(water.b,0.95)
}

plot(ecdf(q95.boot))
abline(v = q95.obs) # we can observe that it's quite skewed
```

bootstrap distribution of the 95th percentile



(empirical cumulative distribution function)

(It has a very bad shape and this is due to the fact that we have only 50 observations)

it's a little bit skewed (distorta): to not be skewed our observation should be around 0.5, here we do not have it, so it's skewed (biased)

statistical quality of the estimator

```
# Let's compute the variance
var  = var(q95.boot)
bias = mean(q95.boot)-q95.obs
RMSE = sqrt(var+bias^2)
var
```

```
## [1] 0.5163227
```

```
bias
```

```
##       95%
## 0.1506802
```

```
RMSE
```

```
##       95%
## 0.7341847
```

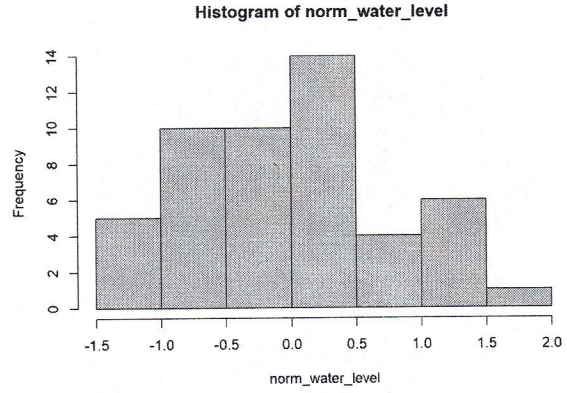
```
# For a CI, given that we do not have any good way to estimate the "variability",
# let's use classic reverse percentile intervals.
alpha <- 0.01

right.quantile <- quantile(q95.boot, 1 - alpha/2)
left.quantile  <- quantile(q95.boot, alpha/2)

CI.RP <- c(q95.obs - (right.quantile - q95.obs), q95.obs, q95.obs - (left.quantile - q95.obs))
CI.RP
```

credo che $\alpha = 0.1$ per avere degli intervalli di 95%.

```
##       95%       95%       95%
## 0.9310779 3.3995828 4.5544873
```

```
### ------------------------------------------------------------------------------
### 2.
### ------------------------------------------------------------------------------
# To solve the second part, we understand that we can use a parametric bootstrap.
# We need to fit a lognormal to our data, and then run bootstrap simulations out of it...
# but how to estimate the parameters of a lognormal? do I need to do it? No.
norm_water_level = log(water_level)
hist(norm_water_level)
```

because we pass through the gaussian distr.




```
shapiro.test(norm_water_level)
```

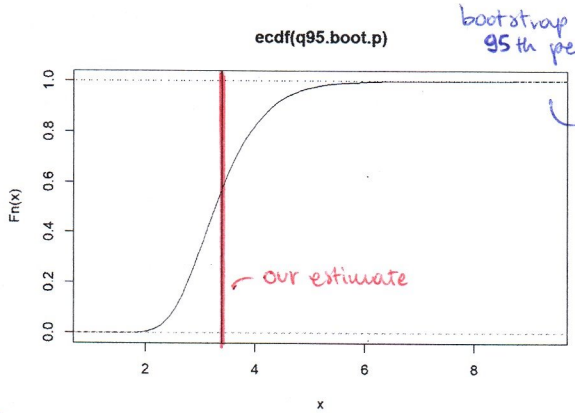
```
##
##  Shapiro-Wilk normality test
##
## data:  norm_water_level
## W = 0.97866, p-value = 0.4969
```

✓

Now that we have a concrete distribution: ↓

```
mean = mean(norm_water_level)
sd   = sqrt(var(norm_water_level))
n    = length(norm_water_level)

q95.boot.p = numeric(B)
for(b in 1:B){
  water.b       = exp(rnorm(n,mean,sd))
  q95.boot.p[b] = quantile(water.b,0.95)
}

plot(ecdf(q95.boot.p))
abline(v = q95.obs)
```

we can do this instead of resampling from the data



bootstrap distribution of the 95th percentile (obtained in this second way)

(empirical cumulative distribution function (prettier than before))

To answer the question: "how can you use this information?" (That data are distributed as lognormal)

→ instead of running a full nonparametric bootstrap we can run a parametric bootstrap (which is supposed to improve the estimation of the bootstrap distribution) !

To see how better we're doing :

```
var.p  = var(q95.boot.p)
bias.p = mean(q95.boot.p)-q95.obs
RMSE.p = sqrt(var.p+bias.p^2)

var  = var(q95.boot)
bias = mean(q95.boot)-q95.obs
RMSE = sqrt(var+bias^2)

data.frame("Non Parametric"=c(var,bias,RMSE),"Parametric"=c(var.p,bias.p,RMSE.p))
```

```
##   Non.Parametric  Parametric
## 1      0.5163227  0.52644150
## 2      0.1506802 -0.02767975
## 3      0.7341847  0.72609067
```

RMSE is a bit lower under the parametric settings ⟹ improvement

```
alpha <- 0.05

right.quantile <- quantile(q95.boot.p, 1 - alpha/2)
left.quantile  <- quantile(q95.boot.p, alpha/2)

CI.RP.p <- c(q95.obs - (right.quantile - q95.obs), q95.obs, q95.obs - (left.quantile - q95.obs))
CI.RP.p
```

```
##      95%      95%      95%
## 1.724061 3.399583 4.562174
```

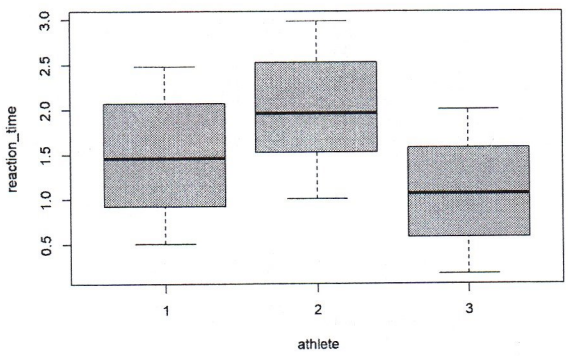
```
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
### Problem 2
### ------------------------------------------------------------------------------
### ------------------------------------------------------------------------------
### The chief coach of the Aosta ski club has tasked you to select who, among its
### three top-class athletes in alpine skiing, can successfully compete also in ski-cross races,
### which have been recently "promoted" to an alpine discipline from its former "freestyle" status.
### One of the key areas in ski-cross is the moment when the athlete "jumps" in the track:
### this, differently from alpine skiing, happens after the blow of a whistle.
### For this reason, fast reaction times can make the difference between losing and winning.
### You're so given the data about 100 "start" trials for the three athletes, stored in
### "parallel_gate.rda". The chief coach is asking if:
###   1. Are there any differences among the athletes?
###      (see if you can use a parametric approach, if not, use a permutational one)
###   2. From a preliminary visual analysis, athlete 3 seems the best:
###      how can you assess this, knowing what you discovered in 1.?
###   3. The coach is also asking you an idea of the "consistency" of athlete 3 out of the gate:
###      provide him with a confidence interval for the mean of his reaction time.

### ------------------------------------------------------------------------------
### 1.
### ------------------------------------------------------------------------------
# Load the data
load('parallel_gate.rda')
head(chrono)
```

```
##   reaction_time athlete
## 1     2.3109997       1
## 2     2.2880683       1
## 3     1.6137042       1
## 4     0.8816244       1
## 5     1.9886475       1
## 6     1.5348088       1
```

```
# Convert athlete to factor
chrono$athlete=factor(chrono$athlete)
summary(chrono)
```

```
##  reaction_time    athlete
##  Min.   :0.1691   1:100
##  1st Qu.:0.9751   2:100
##  Median :1.5398   3:100
##  Mean   :1.5237
##  3rd Qu.:2.0319
##  Max.   :2.9766
```

```
attach(chrono)
boxplot(reaction_time ~ athlete)
```




```
# Times seem different. Let's assess if I can use a classic ANOVA
model_norm = aov(reaction_time ~ athlete)
summary(model_norm)
```

either:
- check normality assumptions in each group and equality of variance among groups
- check the normality on the residuals

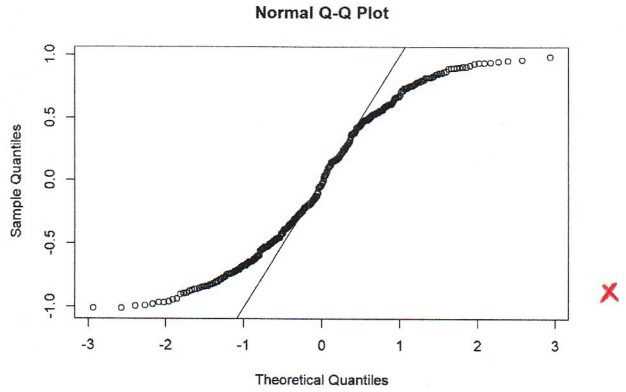
```
##              Df Sum Sq Mean Sq F value Pr(>F)
## athlete       2   45.9   22.95   67.47 <2e-16 ***
## Residuals   297  101.0    0.34
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
shapiro.test(model_norm$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  model_norm$residuals
## W = 0.94391, p-value = 2.888e-09
```

✗

```
qqnorm(model_norm$residuals)
abline(a=0,b=1,col='red')
```

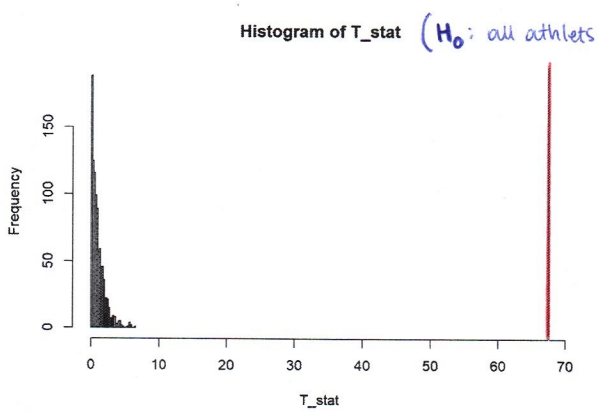


✗

```
# Normality is not verified, let's use permutational anova.
T0 <- summary(model_norm)[[1]][1,4]
T0
```

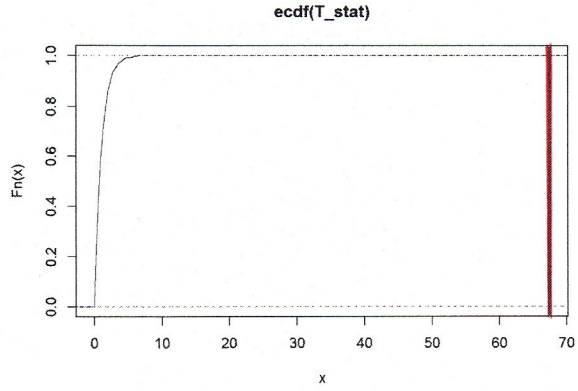
```
## [1] 67.46514
```

```
B      <- 1000 # Number of permutations
T_stat <- numeric(B)
n      <- nrow(chrono)

for(perm in 1:B){
  # Permutation:
  permutation   <- sample(1:n)
  reaction_perm <- reaction_time[permutation]
  model_perm    <- aov(reaction_perm ~ athlete)

  # Test statistic:
  T_stat[perm]  <- summary(model_perm)[[1]][1,4]
}

hist(T_stat,xlim=range(c(T_stat,T0)),breaks=30)
abline(v=T0,col=3,lwd=2)
```



(H_0: all athlets are the same)

```
plot(ecdf(T_stat),xlim=range(c(T_stat,T0)))
abline(v=T0,col=3,lwd=4)
```




```
# p-value
p_val <- sum(T_stat>=T0)/B
p_val
```

```
## [1] 0
```

```
### ------------------------------------------------------------------------
### 2.
### ------------------------------------------------------------------------
# To answer the next point, you need to perform a proper post hoc test.
# Let's go with two permutational t-tests, using as a reference level 3 (the best candidate)
# to be the fastest out of the gate.
# 3 vs 1
pooled1 = chrono[athlete!=2,]
n1      = nrow(pooled1)
ath_3   = pooled1$athlete==3
T0      = abs(mean(pooled1$reaction_time[ath_3])-mean(pooled1$reaction_time[!ath_3]))
T0
```

(Standard) parametric settings

```
## [1] 0.4339291
```

3 vs. 1

```
T_stat=numeric(B)
for(perm in 1:B){
  # permutation:
  permutation <- sample(1:n1)
  time_perm   <- pooled1$reaction_time[permutation]
  ref_perm    <- time_perm[ath_3]
  other_perm  <- time_perm[!ath_3]

  # test statistic:
  T_stat[perm] <- abs(mean(other_perm) - mean(ref_perm))
}

p_val <- sum(T_stat>=T0)/B
p_val
```

```
## [1] 0
```

- **Notice**: here we're calculating p-values, if we would had to test with a level α we would had to do a Bonferroni correction: we want all the test to be significant at the same time $\rightarrow \frac{\alpha}{k}$, $k = \#$ tests

new value α for each test

```
pooled2 = chrono[athlete!=1,]
n2      = nrow(pooled2)
ath_3   = pooled2$athlete==3
T0      = abs(mean(pooled2$reaction_time[ath_3])-mean(pooled2$reaction_time[!ath_3]))
T0
```

```
## [1] 0.9567053
```

3 vs. 2

```
T_stat = numeric(B)
for(perm in 1:B){
  # permutation:
  permutation <- sample(1:n1)
  time_perm   <- pooled1$reaction_time[permutation]
  ref_perm    <- time_perm[ath_3]
  other_perm  <- time_perm[!ath_3]

  # test statistic:
  T_stat[perm] <- abs(mean(other_perm) - mean(ref_perm))
}

p_val <- sum(T_stat>=T0)/B
p_val
```

```
## [1] 0
```

```
### ------------------------------------------------------------------------------
### 3.
### ------------------------------------------------------------------------------
# To answer the last point we are suggested to use the permutational approach.
uni_t_perm = function(data,mu0,B=1000){
  data_trans = data-mu0
  T0         = abs(mean(data_trans))
  T_perm     = numeric(B)
  n          = length(data)

  for(perm in 1:B){
    refl         = rbinom(n, 1, 0.5)*2 - 1
    T_perm[perm] = abs(mean(data_trans*refl))
  }
  return(sum(T_perm>=T0)/B)
}

library(pbapply)
library(parallel)

grid  = seq(0,2,by=0.001)
cl    = makeCluster(2)
ath_3 = reaction_time[athlete==3]

clusterExport(cl,varlist=list("ath_3","uni_t_perm"))

mean(ath_3)
```

(Method 1.)

```
## [1] 1.0602
```

```
perm_wrapper  = function(grid_point){uni_t_perm(ath_3,grid_point)}
pval_function = pbsapply(grid,perm_wrapper,cl=cl)
pval_function
```

```
##    [1] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
##   [13] 0.000 0.000 0.000 ..
```

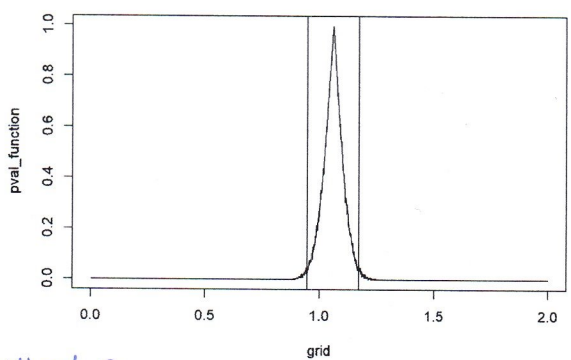
Permutational CI

```
plot(grid,pval_function,type='l')

range(grid[pval_function>0.05])
```

We want a 95% CI so we want to select all those points in the grid where the p-value function is above 0.05

```
## [1] 0.947 1.173
```

```
abline(v=range(grid[pval_function>0.05]))
```



Method 2:

```
# The part could've been equivalently solved by using a bootstrap approach:

T0     = mean(ath_3)
B      = 1000
T.boot = numeric(B)

for(b in 1:B){
  x.b       <- sample(ath_3, replace = T)
  T.boot[b] <- mean(x.b)
}

# RP intervals
alpha <- 0.05

right.quantile <- quantile(T.boot, 1 - alpha/2)
left.quantile  <- quantile(T.boot, alpha/2)

T0
```

```
## [1] 1.0602
```

```
right.quantile - T0
```

```
##     97.5%
## 0.1096465
```

```
left.quantile  - T0
```

```
##        2.5%
## -0.09991792
```

```
CI.RP <- c(T0 - (right.quantile - T0), T0 - (left.quantile - T0))
CI.RP
```

```
##     97.5%      2.5%
## 0.9505533 1.1601178
```

```
range(grid[pval_function>0.05])
```

```
## [1] 0.947 1.173
```