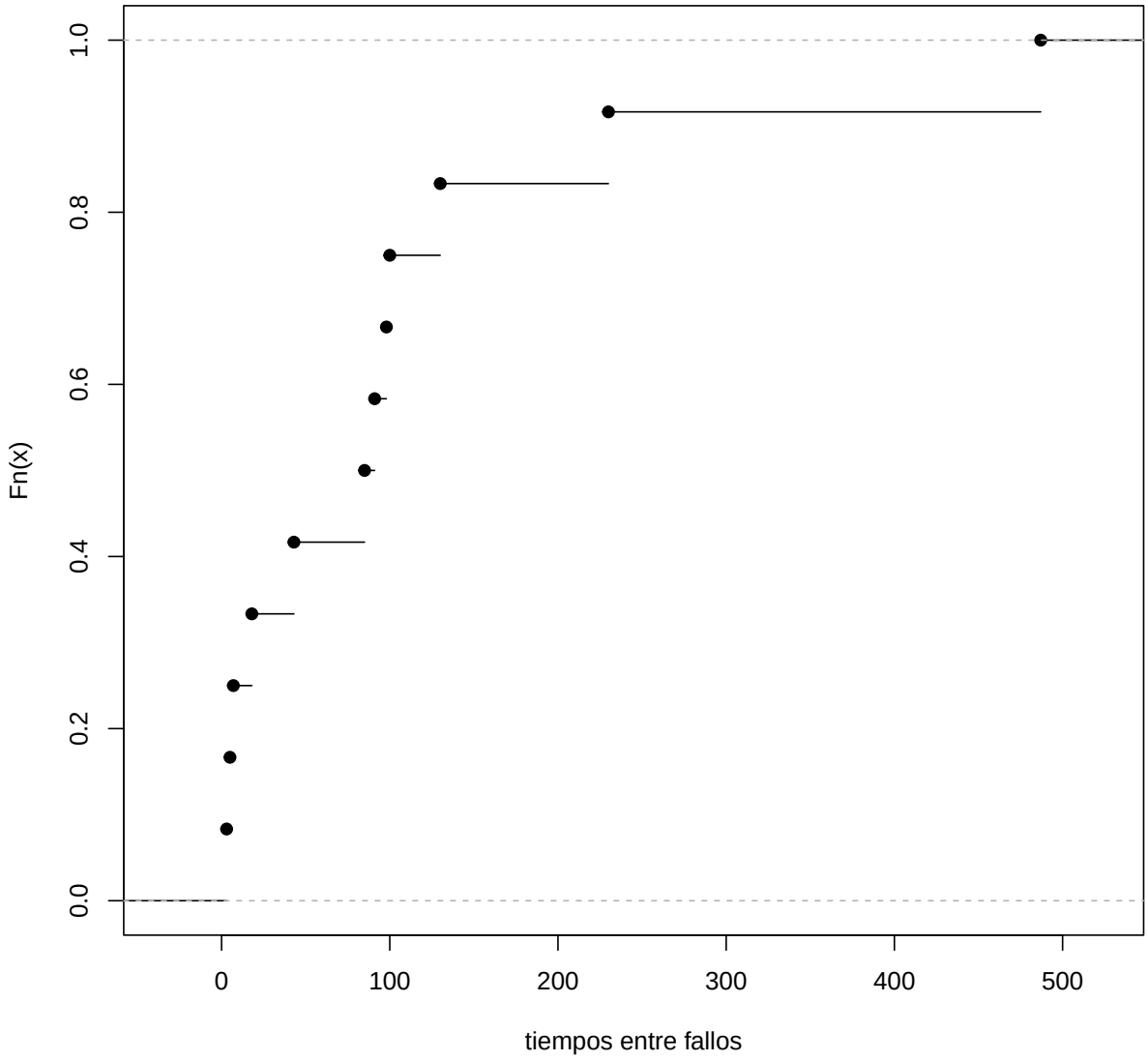
Tema 7. Intervalos de confianza y contrastes de hipótesis basados en remuestreos

Datos sobre la fiabilidad (tiempos de fallo) de aparatos de aire acondicionado.

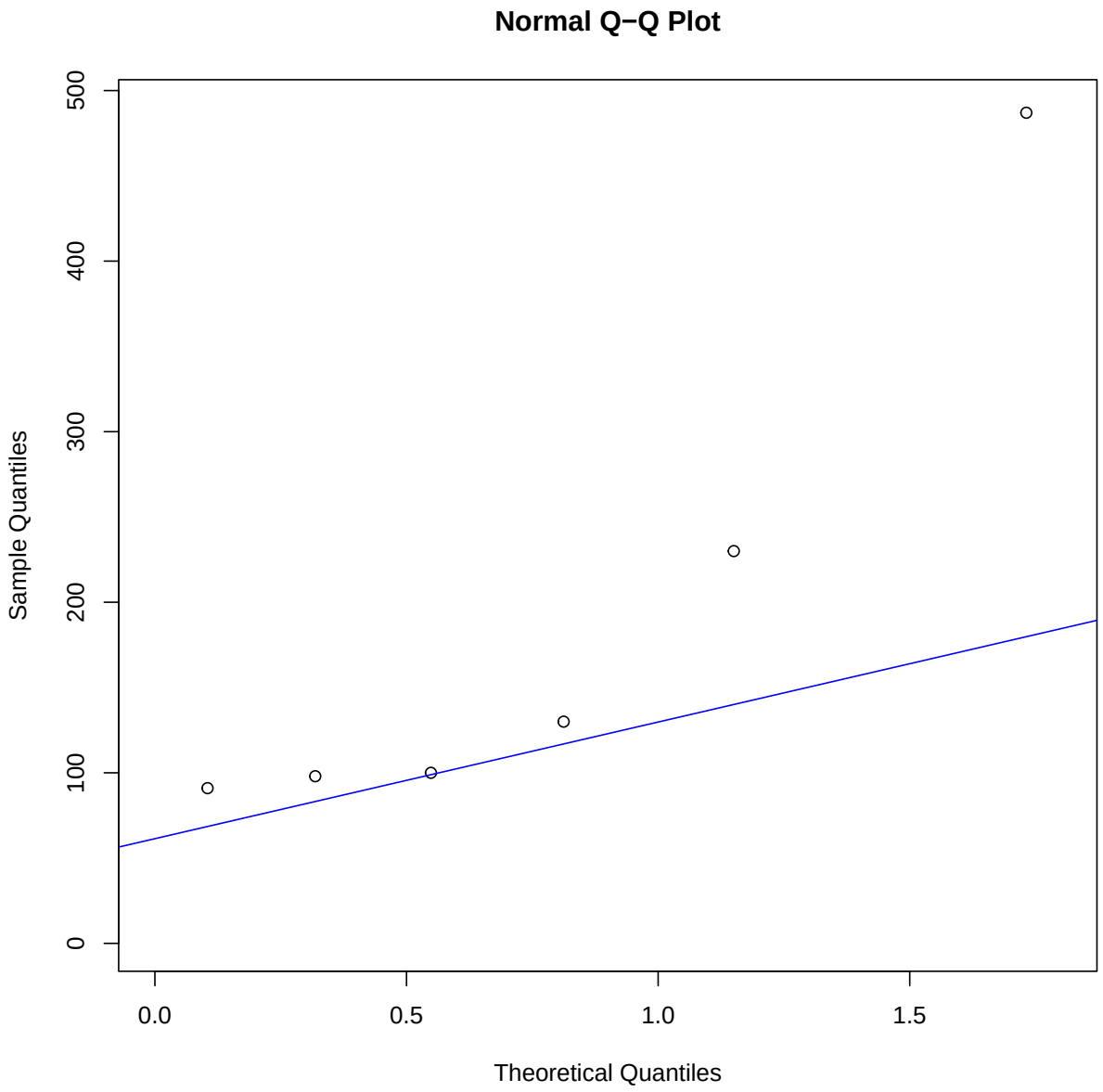
Los datos no parecen distribuirse como una normal.

```
library(boot)
data(aircondit)

plot(ecdf(aircondit$hours),main="",
xlab="tiempos entre fallos")
```



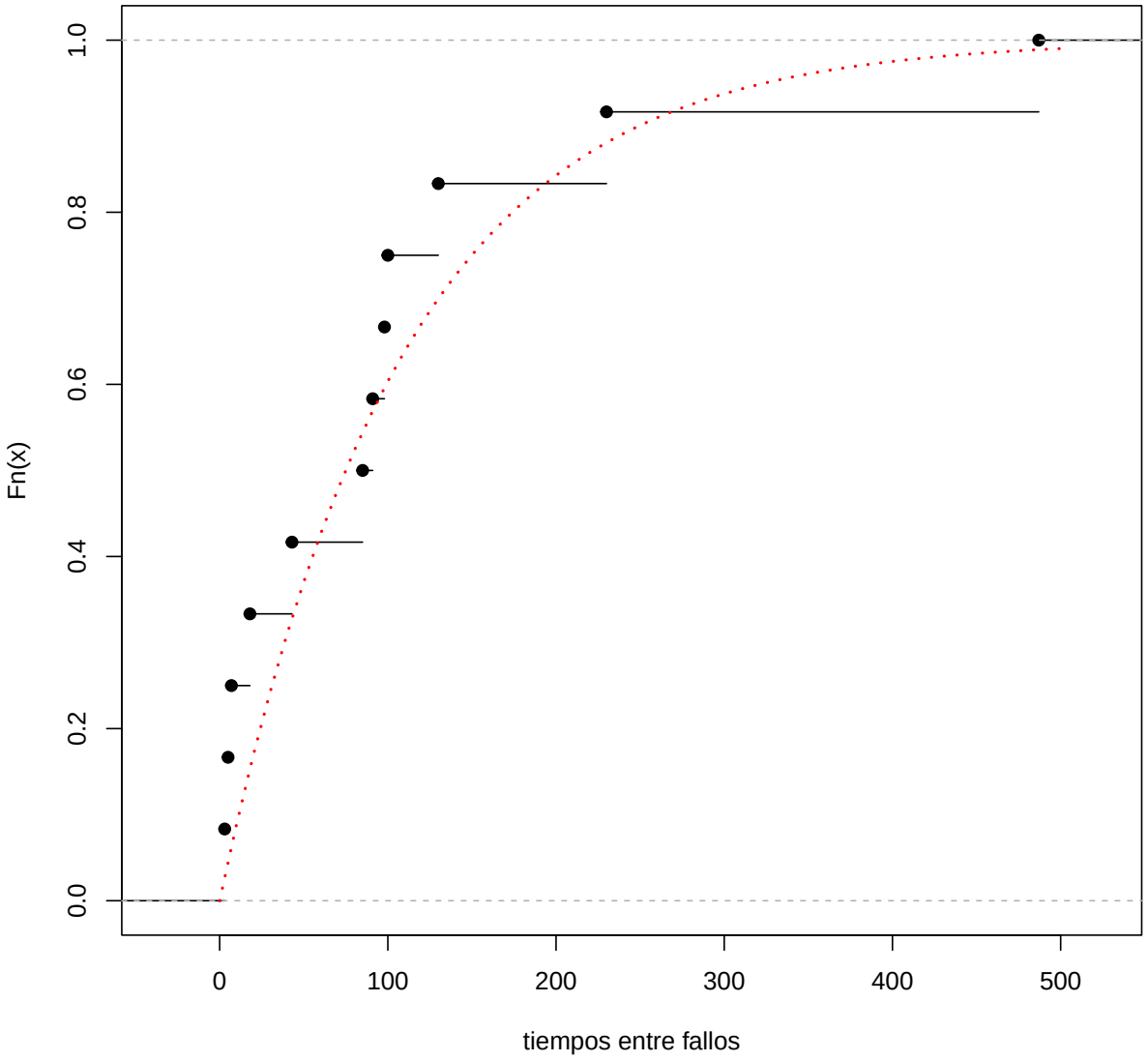

```
qqnorm(aircondit$hours, xlim=c(0,1.8))
qqline(aircondit$hours, col="blue")
```



Asumimos una distribución exponencial. Calculamos su función de distribución.

```
n = length(aircondit$hours)
lambda = length(aircondit$hours)/sum(aircondit$hours)

plot(ecdf(aircondit$hours),main="",
xlab="tiempos entre fallos")
with(list(x=seq(0,500,10)),
lines(1-exp(-lambda*x) ~ x, lty="dotted", lwd=2, col="red"))
```



Al asumir una distribución exponencial, se puede calcular el intervalo de confianza exacto para el parámetro (o para la media). Ver:

en.wikipedia.org/wiki/Exponential_distribution

$$IC_{\frac{1}{\lambda}} = \left[\frac{\bar{x} \cdot 2n}{\chi^2_{2n,1-\frac{\alpha}{2}}}; \frac{\bar{x} \cdot 2n}{\chi^2_{2n,\frac{\alpha}{2}}} \right]$$

De este modo

```r
n = length(aircondit$hours)
media = sum(aircondit$hours)/length(aircondit$hours)
alfa = 0.05

sapply(c(1-alfa/2, alfa/2),
function(alfa) (2*n*media)/qchisq(alfa,2*n))
```

```
[1]  65.89765 209.17415
```

La aproximación a la normal se puede hacer cuando la muestra es grande.

Así, la media muestral se distribuye asintóticamente como una normal aplicando el *TCL* (teorema Central del Límite).

Aproximación normal basada en estimadores paramétricos de σ:

```r
confint(lm(hours ~ 1, data=aircondit))
```

```
               2.5 %   97.5 %
(Intercept) 21.52561 194.6411
```

Aproximación normal basada en estimadores bootstrap de μ y σ.

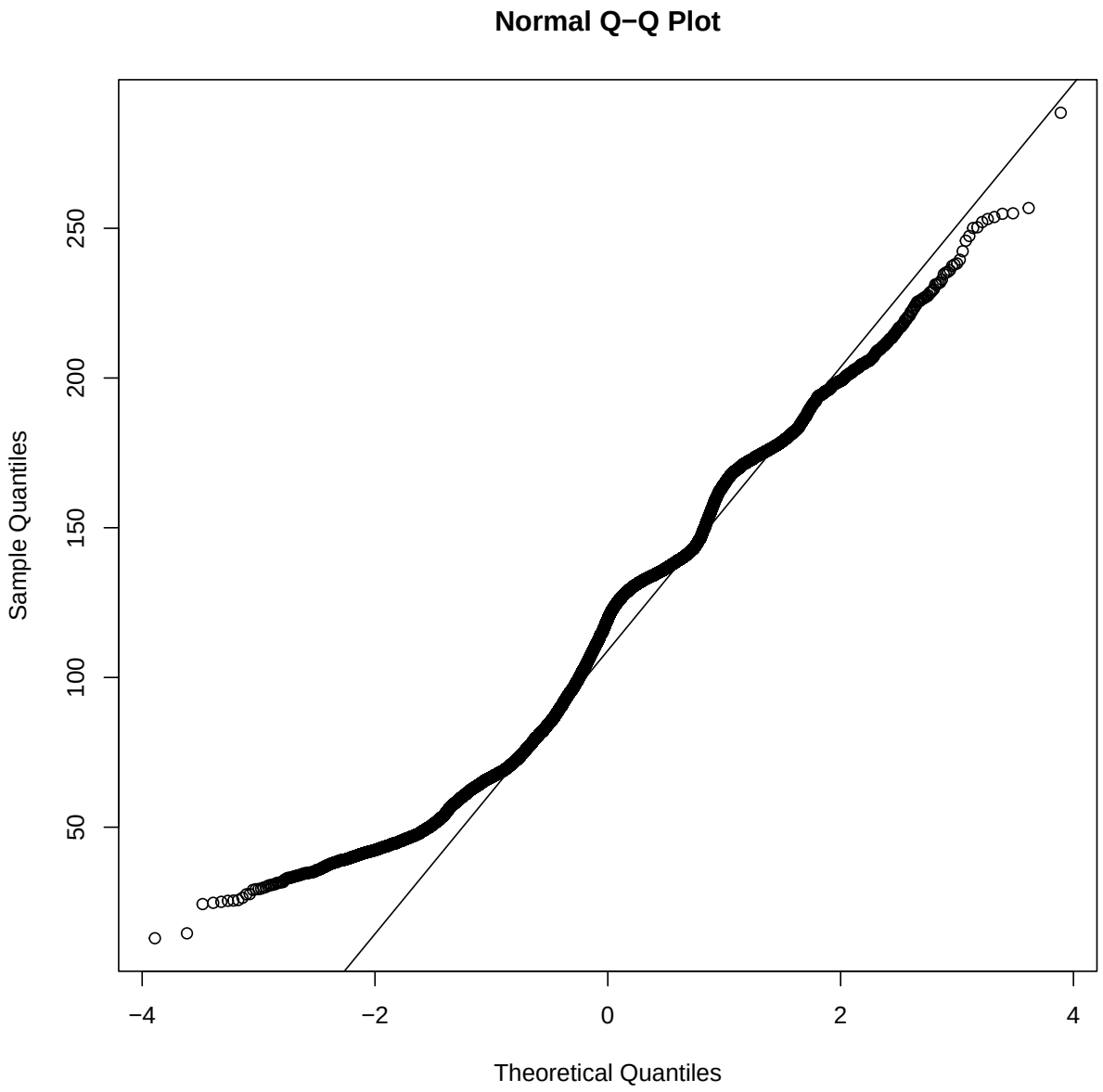
```r
mediasdboot = with(aircondit, replicate(5000,{
boo = sample(hours, replace=TRUE);
c(media=mean(boo), sdev=sd(boo))}))

mean(mediasdboot["media",]) +
c(1,-1)*qnorm(alfa/2)*sd(mediasdboot["media",])
```

```
[1]  34.56012 181.62812
```

Comprobamos el ajuste de la distribución de la media a la normal. Tampoco funciona bien.

```r
qqnorm(mediasdboot)
qqline(mediasdboot)
```



Intervalos bootstrap t

En el ejemplo del aire acondicionado:

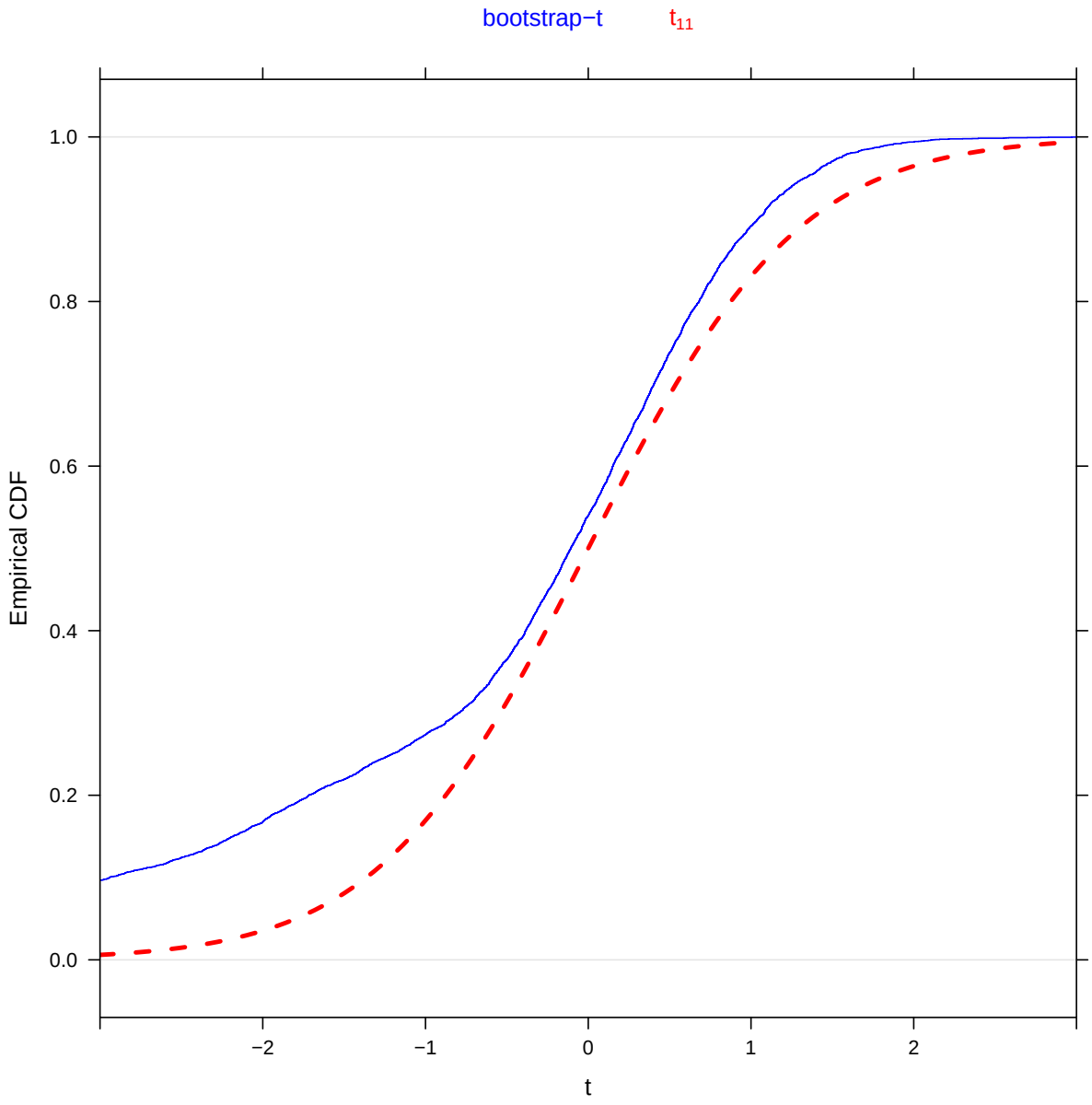
```
N = dim(aircondit)[1]
thetaHat = mean(aircondit[["hours"]])
```

```
mediasdboot = with(aircondit, replicate(5000,{
boo = sample(hours, replace=TRUE);
c(media=mean(boo), sdev=sd(boo))}))

bT = (mediasdboot["media",]-thetaHat)/
     (mediasdboot["sdev",]/sqrt(N))

library(latticeExtra)

ecdfplot(bT,xlim=c(-3,3),xlab="t", col="blue",
key=simpleKey(c("bootstrap-t",expression(t[11])), points=FALSE,lines=FALSE,columns=2,
↪  col=c("blue","red"))) +
layer(panel.curve(pt(x,N-1),col="red",lty=2,lwd=3))
```



Si comparamos ahora los cuantiles de la distribución original t de Student con los cuantiles bootstrap se obtienen resultados muy diferentes.

```
alfa = 0.05

(extremos = quantile(bT, c(1-alfa/2, alfa/2)))
```

```
    97.5%      2.5% 
 1.544491 -4.853443 
```

```
(t.standard = qt(c(1-alfa/2, alfa/2),N-1))
```

```
[1]  2.200985 -2.200985
```

Intervalo bootstrap-t

```
mean(mediasdboot["media",]) - extremos*sd(aircondit[["hours"]])/sqrt(N)
```

```
    97.5%      2.5% 
 47.13571 298.74603 
```

Si se usa la librería bootstrap:

```
library(bootstrap)

lamedia = function(x){mean(x)}

resulta = boott(aircondit[["hours"]], lamedia, VS=TRUE, perc=c(0.025, 0.975))

resulta$confpoints
```

```
        0.025    0.975
[1,] 59.67888 214.4821
```

Intervalos tipo percentil

```r
data(aircondit, package="boot")

alfa = 0.05
meansdboot = with(aircondit, replicate(5000,
c(mean=mean(sample(hours, replace=TRUE)),
sd=sd(sample(hours, replace=TRUE)))))

quantile(meansdboot["mean",], c(alfa/2, 1-alfa/2))
```

```
    2.5%     97.5%
47.91667 190.75625
```

Intervalo *BCa*

```r
boot.BCa =
   function(x, th0, th, stat, conf=0.95) {

       # bootstrap con intervalos de confianza BCa

       # th0 es el estadístico observado

       # th es el vector de replicas bootstrap

       # stat es la función que calcula el estadístico

       x = as.matrix(x)
       n = nrow(x)      # observaciones en filas
       N = 1:n
       alfa = (1 + c(-conf, conf))/2
       zalfa = qnorm(alfa)

       # Factor de corrección del sesgo
       z0 = qnorm(sum(th < th0) / length(th))

       # factor de aceleracion (est. jackknife)
       th.jack = numeric(n)

       for (i in 1:n) {
            J = N[1:(n-1)]
            th.jack[i] = stat(x[-i, ], J)
       }
       L = mean(th.jack) - th.jack
       a = sum(L^3)/(6 * sum(L^2)^1.5)

       # limites confianza BCa
       adj.alfa = pnorm(z0 + (z0+zalfa)/(1-a*(z0+zalfa)))
       limits = quantile(th, adj.alfa, type=6)
```

```
        return(list("est"=th0, "BCa"=limits))
   }
```

```
data(patch, package="bootstrap")
n = nrow(patch)
B = 2000
x = cbind(patch$y, patch$z)
theta.b = numeric(B)
theta.hat = mean(patch$y) / mean(patch$z)

for (b in 1:B) {
   i = sample(1:n, size=n, replace=TRUE)
   y = patch$y[i]
   z = patch$z[i]
   theta.b[b] = mean(y)/mean(z)
}

estadis = function(dat, index) {
      mean(dat[index, 1])/mean(dat[index, 2])}
```

Se obtiene como resultado

```
boot.BCa(x, th0=theta.hat, th=theta.b, stat=estadis)
```

```
$est
[1] -0.0713061

$BCa
 3.078787%  98.02795%
-0.2219138  0.1879028
```

Con la librería `bootstrap`

```
library(bootstrap)

xdata = matrix(rnorm(30),ncol=2)
n = 15

theta = function(ind, xdata){
cor(xdata[ind, 1],xdata[ind, 2])}
```

```
bcanon(1:n, 100, theta, xdata,
alpha=c(0.025, 0.975))$confpoints
```

```
     alpha  bca point
[1,] 0.025 -0.6468535
[2,] 0.975  0.4326426
```

Con la librería `boot`

```
x = c(10, 27, 30, 40, 46, 51, 52, 104, 146)

library(boot)

mean.boot = function(x,ind){
return(c(mean(x[ind]), var(x[ind])/length(ind)))
}

eso = boot(x, mean.boot, 1000)
```

Se obtienen como resultados

```
boot.ci(eso, index=1)
```

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

CALL :
boot.ci(boot.out = eso, index = 1)

Intervals :
Level      Normal              Basic
95%   (29.59, 82.59 )   (27.90, 79.77 )

Level     Percentile            BCa
95%   (32.68, 84.55 )   (35.33, 91.59 )
Calculations and Intervals on Original Scale
Some BCa intervals may be unstable
```

```
boot.ci(eso, index=2)
```

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

CALL :
boot.ci(boot.out = eso, index = 2)

Intervals :
Level      Normal              Basic
95%   ( 34.0, 407.7 )   ( 38.2, 388.9 )

Level     Percentile            BCa
95%   ( 12.1, 362.8 )   ( 31.2, 482.3 )
Calculations and Intervals on Original Scale
Some BCa intervals may be unstable
```

Con la librería `boot` y regresión:

```
d = data.frame(x=1:20, y=runif(20))
m1 = lm(y~x, data=d)

lmcoef = function(data, i){
      d = data[i, ]
      d.reg = lm(y~x, d)
      c(coef(d.reg)) }

m1
```

```
Call:
lm(formula = y ~ x, data = d)

Coefficients:
(Intercept)            x
   0.343252     0.009502
```

Se obtiene como resultado

```
lmboot = boot(d, lmcoef, R=1000)
boot.ci(lmboot, index=2)
```

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

CALL :
boot.ci(boot.out = lmboot, index = 2)
```

```
Intervals :
Level      Normal              Basic
95%   (-0.0109,  0.0302 )   (-0.0105,  0.0308 )

Level     Percentile            BCa
95%   (-0.0118,  0.0295 )   (-0.0121,  0.0291 )
Calculations and Intervals on Original Scale
```

Contrastes de hipótesis

Contrastes Bootstrap-t

Consideramos, por ejemplo, la H_0 de que las medias y varianzas son iguales en ambas poblaciones:

```
library(bootstrap)

thetaHat = mean(mouse.t) - mean(mouse.c)
xy = c(mouse.c,mouse.t)
n = length(mouse.t)
m = length(mouse.c)

tStat = replicate(5000, {
xx = sample(xy, n, replace=TRUE);
yy = sample(xy, m, replace=TRUE);
(mean(xx)-mean(yy))/(sd(c(xx,yy)*sqrt(1/n+1/m)))})

(ASLt = mean(tStat > thetaHat/(sd(xy)*sqrt(1/n+1/m))))
```

```
[1] 0.1352
```

Contrastes de hipótesis BCa

Se simulan dos columnas de datos

```
library(bootstrap)


x = rnorm(100)
y = rnorm(100)

# Se calcula la correlacion
(teta.hat = cor(x, y))
```

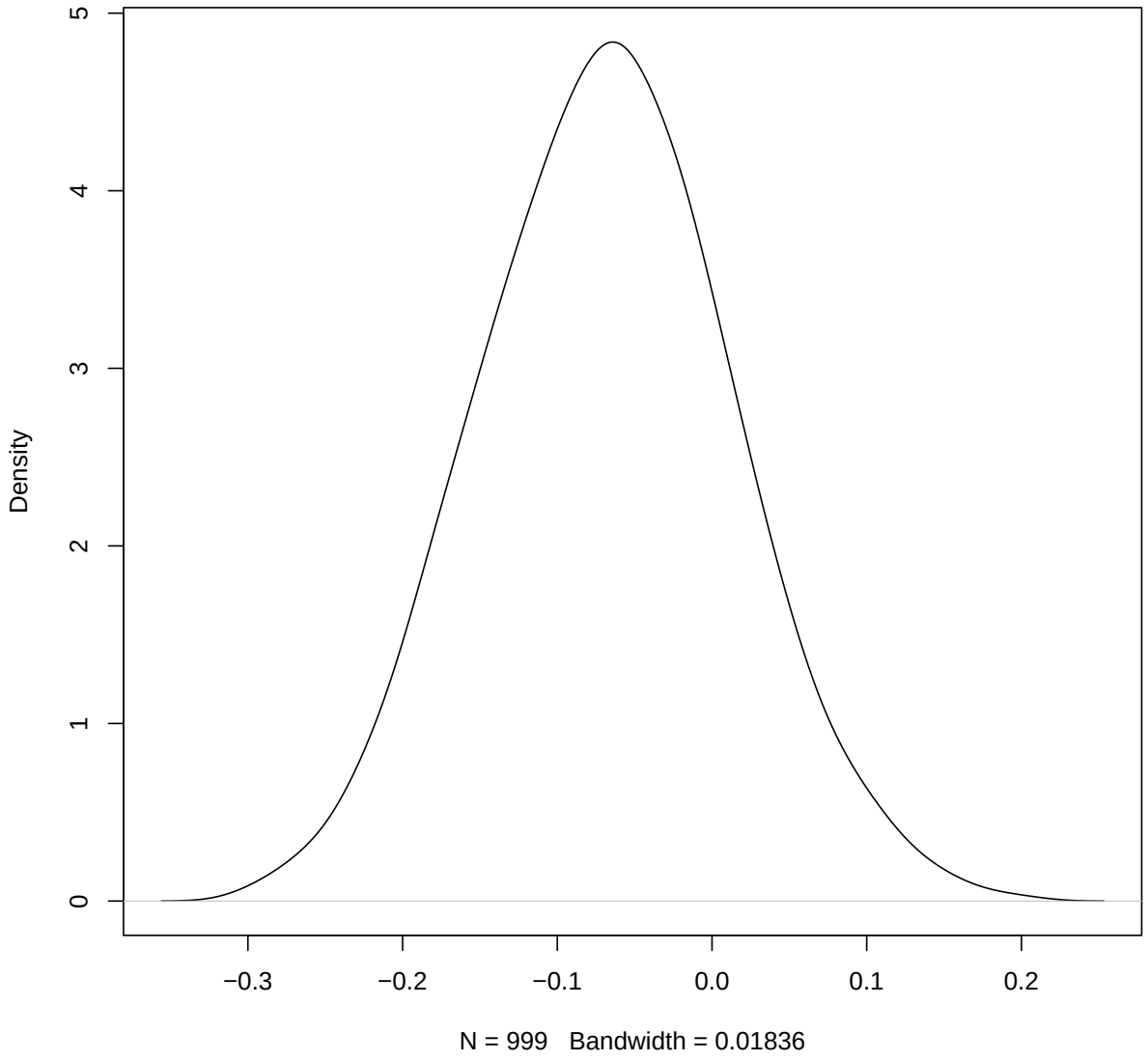
```
[1] -0.06688323
```

`bcanon` calcula los los límites de los intervalos de confianza de intervalos BCa

```r
corre = function(k){
  cor(x[k], y[k])
}

# Sacas una lista de posibles valores de alfa para encontrar el
# que más se ajuste al theta.hat observado
bca.out = bcanon(1:length(x), nboot = 10000, theta=corre, alpha=seq(0.001, 0.999, 0.001))
```

```r
plot(density(bca.out$confpoints[,2]), main="")
```



Test de una cola inferior: p-valor para para $\theta = \hat{\theta}$

```
# Buscas por interpolación el valor de teta.hat observado en la lista de puntos
ltpv = approx(bca.out$confpoints[,2], bca.out$confpoints[,1], xout=teta.hat)$y
```

Test de una cola superior para $\theta = \hat{\theta}$

```
1 - ltpv
```

```
[1] 0.4949835
```

Test de dos colas para $\theta = 0$

```
2 * min(ltpv, 1 - ltpv)
```

```
[1] 0.9899671
```

Alternativa

Ir a

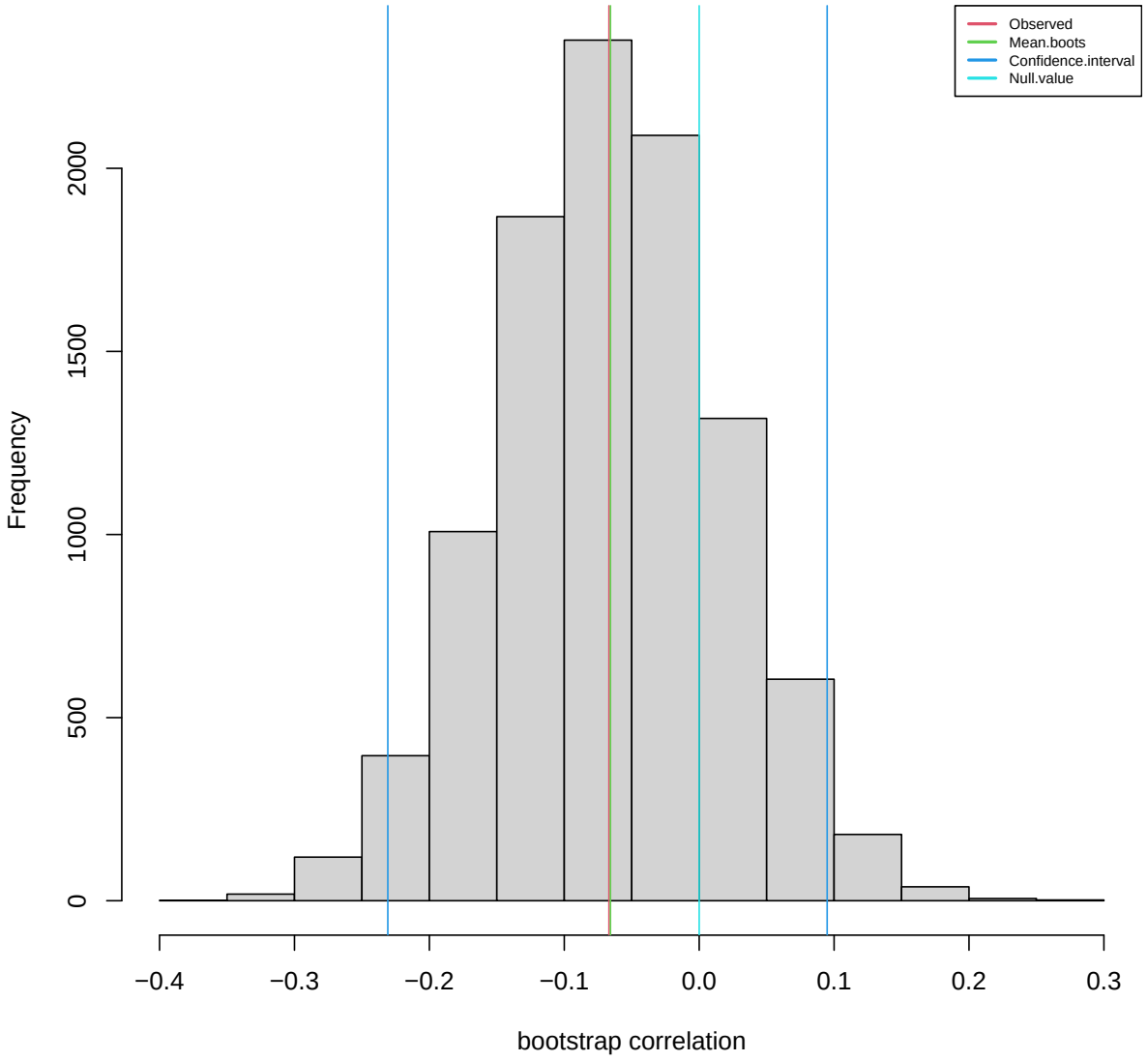
https://cran.r-project.org/src/contrib/Archive/wBoot/wBoot_1.0.3.tar.gz

Instalar el paquete en R usando

```
install.packages(pkgs="c:/Tus descargas.../wBoot_1.0.3.tar.gz", repos=NULL, type="source")
```

```
library(wBoot)

# boot.cor.bca(x, y, null.hyp = 0, alternative = "greater")
boot.cor.bca(x, y, null.hyp = 0)
```




```
RESULTS OF BCa BOOTSTRAP FOR CORRELATION

   SUMMARY Variable.1 Variable.2   n   Statistic    Observed
STATISTICS          x          y 100 correlation -0.06688323

BOOTSTRAP Replications        Mean        SE    Bias Percent.bias
  SUMMARY         9999 -0.06576612 0.0832129 0.00112         1.67

HYPOTHESIS Null Alternative P.value
      TEST    0   not-equal   0.419
```

```
 CONFIDENCE Level      Type Confidence.interval
   INTERVAL   95% two-sided  (-0.2308, 0.09485)
```

Tests unimuestrales

```
data(aircondit)

library(MKinfer)

# Test con Ho: mu = 100
boot.t.test(aircondit[["hours"]], mu=100)
```

```
    Bootstrap One Sample t-test

data:  aircondit[["hours"]]
bootstrap p-value = 0.7703
bootstrap mean of x (SE) = 107.8726 (34.9565)
95 percent bootstrap percentile confidence interval:
  46.58333 189.58333

Results without bootstrap:
t = 0.20554, df = 11, p-value = 0.8409
alternative hypothesis: true mean is not equal to 100
95 percent confidence interval:
  21.52561 194.64105
sample estimates:
mean of x
 108.0833
```

```
unaMuestraBootpvalor = function(x, mu0, R = 1000){

# x: datos observados

# mu0: Media bajo la hipótesis nula

# estadistico del test
tstat = function(d, i, mu0) {
sqrt(length(i)) * abs(mean(d[i]) - mu0) / sd(d[i])
}

# estadistico del test para datos observados x
t0 = tstat(x, 1:length(x), mu0)
```

```r
# R estadisticos de test remuestrados donde
# mu0 = mean(x) se pasa a la funcion  tstat

bt = boot::boot(x, tstat, R = R, mu0 = mean(x))$t[,1]

# Se obtiene el p-valor
c(pvalue = mean(bt > t0))
}
```

Se aplica la función a un conjunto de datos en los que la hipótesis nula es cierta:

```r
set.seed(666)

# H0 es correcta
x = rexp(100, rate = 1/17)
unaMuestraBootpvalor(x, 17)
```

```
pvalue
  0.11
```

Tests bimuestrales

```r
DosMuestrasBootpvalor = function(x, y, alternativa = c("bilateral", "menor_que",
↪  "mayor_que"), R = 2000){
# x: datos observados (primera muestra)
# y: datos observados (segunda muestra)
# alternativa - especifica la hipótesis alternativa

alternativa = match.arg(alternativa)
n1 = length(x)
n2 = length(y)

# estadistico del test

tstat = function(d, i){
boot.xy = d[i]
x = boot.xy[1:n1]
y = boot.xy[-(1:n1)]
s = sqrt( ((n1-1) * var(x) + (n2 - 1) * var(y)) / (n1 + n2 - 2) )
mu.x = mean(x)
mu.y = mean(y)
(mu.x - mu.y) / s / sqrt(1 / n1 + 1 / n2)
}

xy = c(x,y)
```

```r
# estadistico del test para los datos observados
t0 = tstat(xy, 1:(n1 + n2))

# Remuestreo del estadistico del test
bt = boot::boot(xy, tstat, R = R)$t[,1]

# p-valor
if(alternativa == "mayor_que") return(c(pvalue = mean(bt > t0)))
if(alternativa == "menor_que") return(c(pvalue = mean(bt < t0)))
c(pvalue = mean(abs(bt) > abs(t0)))
}
```

Se aplica la función a un conjunto de datos en los que la hipótesis nula es cierta:

```r
# H0 es correcta, mu_x es igual a mu_y
set.seed(666)
datos1 = rnorm(10, mean = 3, sd = 2)
datos2 = rnorm(20, mean = 3, sd = 2)

DosMuestrasBootpvalor(datos1, datos2, alternativa = "mayor_que")
```

```
pvalue
 0.313
```

```r
# library(MKinfer)

boot.t.test(datos1, datos2)
```

```
    Bootstrap Welch Two Sample t-test

data:  datos1 and datos2
bootstrap p-value = 0.6675
bootstrap difference of means (SE) = 0.4797342 (1.008988)
95 percent bootstrap percentile confidence interval:
 -1.537767  2.465471

Results without bootstrap:
t = 0.44627, df = 14.379, p-value = 0.662
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.804340  2.755463
sample estimates:
mean of x mean of y
 2.807849  2.332287
```

Se puede comparar con un test de permutaciones:

```r
# library(MKinfer)

perm.t.test(datos1, datos2)
```

```
    Permutation Welch Two Sample t-test

data:  datos1 and datos2
(Monte-Carlo) permutation p-value = 0.6545
permutation difference of means (SE) = 0.4712101 (0.9550028)
95 percent (Monte-Carlo) permutation percentile confidence interval:
 -1.349719  2.308100

Results without permutation:
t = 0.44627, df = 14.379, p-value = 0.662
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.804340  2.755463
sample estimates:
mean of x mean of y
 2.807849  2.332287
```

Varianza estimadores bootstrap

```r
# Datos de parches
data(patch, package = "bootstrap")
n = nrow(patch)

B = 1000
bootstat = numeric(B)
bootsam = matrix(nrow = B, ncol = n)

for (b in 1:B) {

  indices = sample(1:n, replace = TRUE)
   y = patch$y[indices]
   z = patch$z[indices]

   bootstat[b] = mean(y) / mean(z)
   bootsam[b, ] = indices
}
```

```r
# Jackknife-after-bootstrap
jreps = numeric(n)

for (i in 1:n) {

  # Identificas las muestras bootstrap que no contienen a la i-esima observacion
  jacksam = which(!apply(bootsam, 1, function(x) i %in% x))

  # Calculas la desviación estándar de los estadísticos para estas muestras bootstrap
  jreps[i] <- sd(bootstat[jacksam])

}

# Estimas el error (estimador jackknife)
(varjack = (n-1)/n * sum((jreps - mean(jreps))^2))
```

```
[1] 0.0006760233
```

```r
# Estimador bootstrap original del error
sd(bootstat)
```

```
[1] 0.09865365
```

Alternativa

```r
# library(boot)
# data(patch, package = "bootstrap")

n = nrow(patch)

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

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

library(stringr)
x = capture.output(patch.boot)

rbind(x[10],x[11])
```

```
     [,1]
[1,] "      original     bias    std. error"
[2,] "t1* -0.0713061  0.0104997   0.1056087"
```

```r
# Extraes los coeficientes bootstrap
patch.b = patch.boot$t[, 1]

# Jackknife-after-bootstrap
jreps = numeric(n)

for (i in 1:n) {
  # Identificas las muestras bootstrap que no contienen a la i-esima observacion
  excluyo_i = apply(patch.boot$t[, -1], 1, function(x) !i %in% x)

  # Calculas la desviación estándar de los estadísticos para estas muestras bootstrap
  jreps[i] = sd(patch.b[excluyo_i])
}

# Estimas el error (estimador jackknife)
(varjack = ((n - 1) / n) * sum((jreps - mean(jreps))^2))
```

```
[1] 0.0009085384
```

```r
# Estimador bootstrap original del error
sd(patch.boot$t[, 1])
```

```
[1] 0.1056087
```

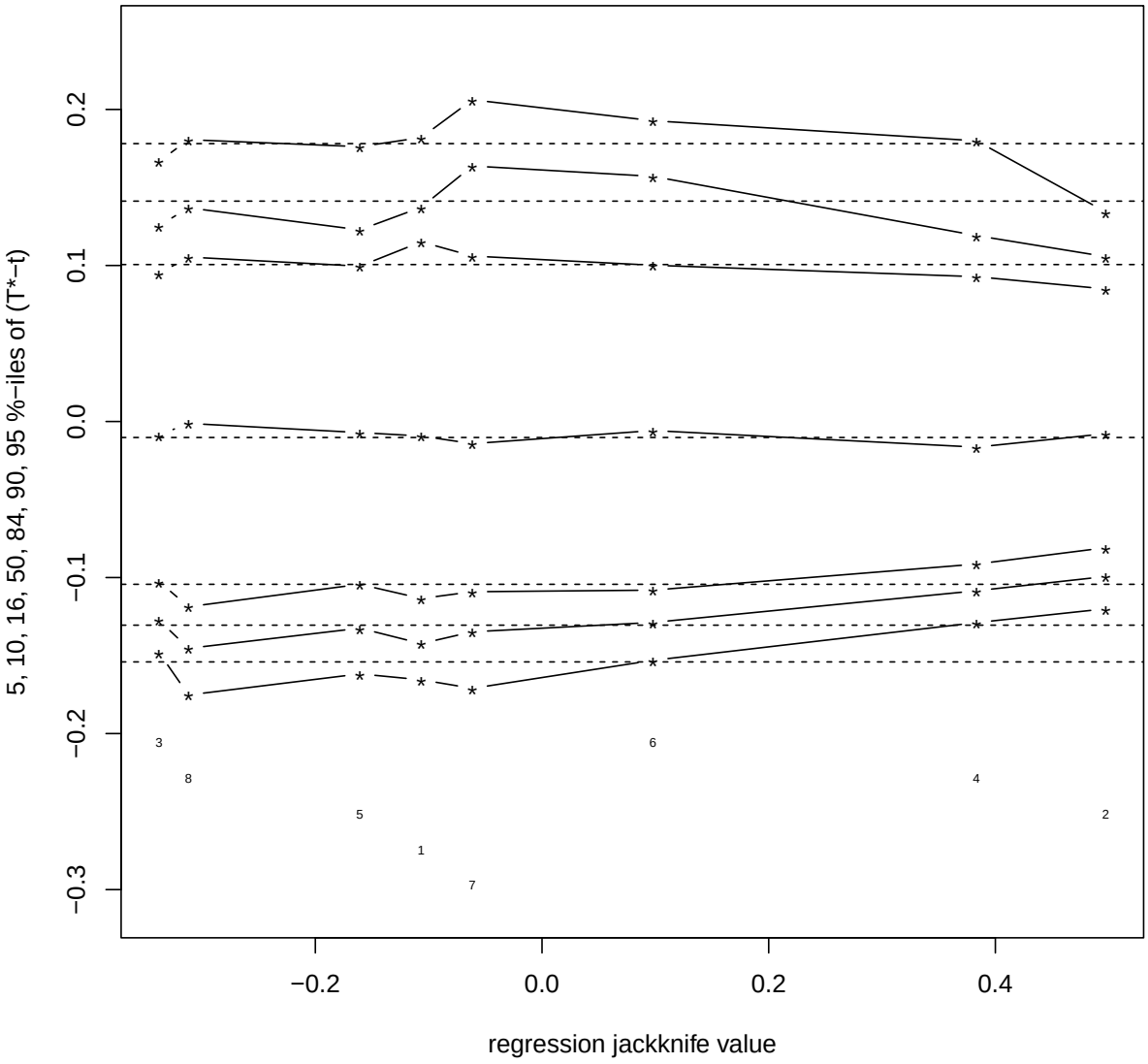
Los cuantiles jackknife centrados para cada observación, se estiman a partir de las muestras bootstrap en las que no apareció cada observación en particular.

El gráfico consta de una serie de líneas de puntos horizontales que corresponden a los cuantiles de la distribución bootstrap centrada. De arriba hacia abajo, las líneas de puntos horizontales muestran los percentiles 95, 90, 84, 50, 16, 10 y 5. Se trazan, para cada una de las observaciones, los cuantiles de la distribución bootstrap frente a los valores jackknife calculados omitiendo ese punto.

El número de la observación se muestra debajo de los gráficos. Para que sea más fácil ver el efecto de omitir puntos en los cuantiles, los cuantiles trazados se unen mediante líneas.

Estos gráficos proporcionan una herramienta de diagnóstico útil para establecer el efecto de las observaciones individuales en la distribución bootstrap.

```
jack.after.boot(patch.boot, useJ = FALSE, stinf = FALSE)
```


5, 10, 16, 50, 84, 90, 95 %−iles of (T*−t)
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.2
0.0
0.2
0.4
regression jackknife value
3
6
8
4
5
2
1
7