APPLIED RESAMPLING METHODS

Michael A. Martin, PhD, AStat
Steven E. Stern, PhD, AStat

Michael.Martin@anu.edu.au
Steven.Stern@anu.edu.au

3 MAJOR GOALS OF STATISTICS

1. **What data should I collect?**
   How do I collect my data (sampling)?
   What assumptions am I making about my data?

2. **How do I organise and analyse my data?**
   Graphically?
   Numerically?

3. **What do my data analyses and summaries mean?**
   How "accurate" are my estimators?
   How "precise" are my estimators?
   What do "accurate" and "precise" even mean?

Statistical Inference promises answers to 3.

A simple example: Does aspirin prevent heart attacks?

Experimental situation: **population consists of healthy, middle-aged men**

Type of study:

Controlled – half of the subjects were given aspirin, the other half a placebo (control group)

Randomised – subjects were randomly assigned to aspirin and placebo groups

Double-blind – neither patients nor doctors knew to which group patients belonged

The Data:

	Subjects	heart attacks
Aspirin Group	11037	104
Placebo Group	11034	189

How might we analyse this data?

One possible approach (there are others…)

Rate of heart attacks in aspirin group:

$$\frac{104}{11037} = 0.0094.$$

Rate of heart attacks in placebo group:

$$\frac{189}{11034} = 0.0171.$$

A reasonable statistic of interest is

$$\hat{\theta} = \frac{\text{Rate in Aspirin Group}}{\text{Rate in Placebo Group}} = 0.55.$$

This is a ratio estimator. We could also take logs to produce a location-type estimator based on differences.

Now what? How did we do?

Data Collection:

- **the use of a control group allows meaningful comparison…… ✓**

- **randomisation reduces the risk of systematic bias……✓**

- **double-blind study removes potential for "psychological" effects or the tendency for doctors to treat higher-risk patients……✓**

Analysing data:

- **The statistic of interest makes sense – it obviously compares the rate of heart attacks between the two groups. In this study, we can see that aspirin patients suffer heart attacks at about half the rate of non-aspirin patients**

- **Is that it? Time to buy shares in Herron (or Parke-Davis?)**

Hold on… 55% is not the true rate, but rather an *estimated* rate. In statistical terms, we might say that $\hat{\pi}$ estimates a true, unknown rate π. We don't yet know if $\hat{\pi}$ is a good estimator of π, or even what we mean by "good"…

"Good" might include features such as low (or no) bias, and low variance

These qualities can be assessed theoretically using limit theorems as sample size *n* grows, and in this case we're probably OK for large *n*.

How large is n in this case?

$\hat{\pi}$ is based on 22071 patients (large)

$\hat{\pi}$ is based on 293 heart attacks (large?)

If the experiment were conducted again, would the resultant value of $\hat{\pi}$ be very different from what happened this time??

STANDARD TOOLS OF INFERENCE

- **POINT ESTIMATION**
  Bias (location)
  Variance (spread)

- **INTERVAL ESTIMATION**
  Find two (data-based) values between which the true value probably lies.
  The "standard" 95% interval is
  $\hat{\theta} \pm 1.96 \times$ **standard error**$(\hat{\theta})$**, and is based on the idea that the quantity**

$$\frac{\hat{\theta} - \theta}{\text{standard error}(\hat{\theta})}$$

  has an asymptotic standard Normal distribution

- **HYPOTHESIS TESTING**

SAMPLING DISTRIBUTIONS AND REPEATED SAMPLING…

Imagine repeating this experiment a large number, say 1000, of times (for our example, this proposal would mean finding 22,000,000 healthy, middle-aged men!!!), each time calculating a value of $\hat{\pi}$.

The histogram of $\hat{\pi}$ values so obtained is an approximation to the distribution of $\hat{\pi}$ under *repeated sampling*. From this distribution, the mean, variance and other properties of $\hat{\pi}$ can be estimated.

PRACTICAL PROBLEM: When <u>you</u> conduct an experiment, you get only <u>one</u> value of $\hat{\pi}$, so how can you say what will happen to $\hat{\pi}$ under repeated sampling?

CLASSICAL APPROACH: Use some theoretical distribution to approximate the (usually asymptotic) sampling distribution of $\hat{\theta}$ (or a quantity based on $\hat{\theta}$), and use that result even in small- to moderate-sample situations

If the statistic is based on means or proportions, a Central Limit Theorem will often help in defining an appropriate limiting distribution, without the need to actually perform repeated sampling

CLT's apply in other cases too, but do practitioners *really* want to be dealing with all that theory in every single case they consider??

SOME PROBLEMS WITH THE "CLASSICAL" APPROACH

The sampling distribution of $\hat{\theta}$ often depends on unknown parameters (such as θ itself). "Pivotal" quantities, whose distributions do *not* depend on unknown parameters, should be used in inference

A SIMPLE EXAMPLE

Suppose $X_1,\ldots,X_n$ are i.i.d. $N(\mu,\sigma^2)$. Then,

So, even if you do use a CLT to form an approximately pivotal quantity on which to base inference, you are immediately subject to the error involved in the fixed-*n* behaviour of the quantity unless you've been lucky enough to develop an exact pivot

The "standard interval"

$$\hat{\theta} \pm 1.96 \text{ s.e.}(\hat{\theta})$$

makes LOTS of assumptions, based on ideas such as sufficiency and the CLT. In essence, it reduces the inference to involving only $\hat{\theta}$, its standard error, and the (asymptotic) normality of the usual "pivot"

BUT does the data contain more information that the standard interval doesn't use, for instance about tail behaviour of the pivot or about the distribution's shape for small *n*?

A SIMPLE EXAMPLE: The correlation coefficient

Asthma data (Rice): height and respiratory resistance in children

Standard interval: $\hat{\rho} \pm 1.96$ s.e.$(\hat{\rho})$ **where**

$$\text{s.e.}(\hat{\rho}) = \sqrt{\frac{\rho^2}{n}\left[\frac{\mu_{22}}{\mu_{11}^2} + \frac{1}{4}\left[\frac{\mu_{40}}{\mu_{20}^2} + \frac{\mu_{04}}{\mu_{02}^2} + \frac{2\mu_{22}}{\mu_{20}\mu_{02}}\right] - \left[\frac{\mu_{31}}{\mu_{20}\mu_{11}} + \frac{\mu_{13}}{\mu_{11}\mu_{02}}\right]\right]}$$

and

$\mu_{ij} = \frac{1}{n}\sum_{k=1}^{n}(X_k - \bar{X})^i (Y_k - \bar{Y})^j$**, and** $\hat{\rho}$ **is the sample correlation coefficient**

The expression for s.e.$(\hat{\rho})$ **is a mess – and is unstable in small samples**

In this case, particularly for small samples and when the sampling distribution of $\hat{\rho}$ **is very skewed, the resultant interval can extend beyond** $[-1, 1]$

```
> # A SIMPLE EXAMPLE

> varrho <- function(x, y){
      n <- length(x)
      rho <- cor(x, y)
      x1 <- x - mean(x)
      y1 <- y - mean(y)
      mu11 <- mean(x1 * y1)
      mu20 <- mean(x1 * x1)
      mu02 <- mean(y1 * y1)
      mu22 <- mean(x1 * x1 * y1 * y1)
      mu31 <- mean(x1 * x1 * x1 * y1)
      mu13 <- mean(x1 * y1 * y1 * y1)
      mu40 <- mean(x1 * x1 * x1 * x1)
      mu04 <- mean(y1 * y1 * y1 * y1)
      varrho <- (rho * rho)/n *
         (mu22/(mu11 * mu11) + 0.25 *
         (mu40/(mu20 * mu20) + mu04/
         (mu02 * mu02) + (2 * mu22)/
         (mu02 * mu20)) - (mu31/(mu20 *
         mu11) + mu13/(mu02 * mu11)))
   varrho
   }

> x1_rnorm(10,0,1)
> x2_x1+rnorm(10,0,0.2)
> # True value of correlation is 0.9805807
> cor(x1,x2)-1.96*sqrt(varrho(x1,x2))
[1] 0.959696
> cor(x1,x2)+1.96*sqrt(varrho(x1,x2))
[1] 1.002654
```

What went wrong? Of course, the problem is that when *n* is only 10, we can't rely on the CLT… but what have we got in its place?

A frequentist interpretation of the CLT is that under "repeated sampling", the histogram of our "pivots" should be standard normal. Small samples cause us trouble because the quality of the standard normal approximation can be poor. But even so, what does "repeated sampling" mean? In theory, we are thinking of same-size samples of data drawn from the same population and under the same set of "sampling rules" as the original sample.

What are those sampling rules? How can we characterise the underlying population? Can we simulate "repeated sampling" in a meaningful way?

One way to think of resampling methods is as a way of simulating "repeated sampling". Under classical methods based on a CLT, the resultant sampling distribution is a theoretical limiting distribution – can our simulated repeated sampling capture the behaviour of the sampling distribution of a quantity for small samples, using information available in the data rather than relying on theoretical limiting distributions?

FOUR MAJOR RESAMPLING METHODS

1. **JACKKNIFE – used mainly for bias correction and calculation of variance**

2. **CROSS-VALIDATION – used mainly for model selection and estimating error rates**

3. **PERMUTATION TESTS – used mainly to compare distributions**

4. **BOOTSTRAP – broadly applicable, includes Monte Carlo simulation as a special case**

THE JACKKNIFE

Quenouille (1949) introduced the idea to correct for bias in time series estimation

Tukey (1958) popularised the idea and coined the name in perhaps Statistics' most famous abstract (there was no paper)

Given a sample $\mathbf{X} = (X_1, \ldots, X_n)$ and an estimator $\hat{\theta} = \hat{\theta}(\mathbf{X}) = \hat{\theta}(X_1, \ldots, X_n)$ which is symmetric in the data

Define the Jackknife (leave-one-out) samples:

$$\begin{aligned}
\mathbf{X}_1 &= (X_2, \ldots, X_n) \\
\mathbf{X}_2 &= (X_1, X_3, \ldots, X_n) \\
&\vdots \\
\mathbf{X}_i &= (X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n) \\
&\vdots \\
\mathbf{X}_n &= (X_1, \ldots, X_{n-1})
\end{aligned}$$

new samples

Define the jackknife replicates:

$$\hat{\theta}_{(i)} = \hat{\theta}(\mathbf{X}_i) = \hat{\theta}(X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n),$$

the value of $\hat{\theta}$ computed using the *i*'th jackknife sample. Define

$$\hat{\theta}_{(\bullet)} = \frac{1}{n}\sum_{i=1}^{n} \hat{\theta}_{(i)}$$

then the jackknife estimate of the bias of $\hat{\theta}$ in estimating θ is

$$\hat{b}_{Jack}(\hat{\theta}) = (n-1)[\hat{\theta}_{(\bullet)} - \hat{\theta}]$$

and so a bias-corrected estimate of θ is

$$\hat{\theta}_{corrected} = \hat{\theta} - \hat{b}_{Jack}(\hat{\theta}) = n\hat{\theta} - (n-1)\hat{\theta}_{(\bullet)}$$

How does this work in some simple cases?

The mean: $\hat{\theta} = \bar{X}$, and $\hat{\theta}_{(\bullet)} = \bar{X}$, and so the estimated bias is 0, which is correct as the sample mean is unbiased for the true mean

The variance: Consider the maximum likelihood variance estimator, $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$**, as an estimator of the true variance** σ^2**. This estimator is biased for** σ^2**. Then, after some rather tedious algebra,**

$$\hat{\sigma}^2_{(\bullet)} = \frac{n-2}{(n-1)^2}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

and so,

$$\hat{\sigma}^2_{corrected} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

which is the usual unbiased variance estimator.

Basically, the jackknife removes the first-order (n^{-1}) **term in the bias expansion of the original estimator. In the case of the variance discussed above it removes *all* of the bias.**

ANOTHER WAY OF LOOKING AT THE JACKKNIFE

Define the "pseudo-values",

$$\tilde{\theta}_i = n\hat{\theta} - (n-1)\hat{\theta}_{(i)},$$

and use as the jackknife estimate of θ the average of the pseudo-values:

,

which is, of course, the jackknife bias-corrected estimate.

The pseudo-values were constructed so that in the case $\hat{\theta} = \bar{X}$, the pseudo-values were just the original data points and the jackknife estimator was simply the sample mean

Heuristically, the *i*'th pseudo-value is a measure of the "influence" the *i*'th data point has on the estimator $\hat{\theta}$, and so the pseudo-values act as "data points" on the same scale as the parameter estimate.

JACKKNIFE VARIANCE ESTIMATION

Tukey reasoned that if one thinks of the pseudo-values as one thinks of data points, their use may be generalised to include variance estimation:

$$\hat{V}_{Jack}(\hat{\theta}) = \frac{n-1}{n}\sum_{i=1}^{n}(\hat{\theta}_{(i)} - \hat{\theta}_{(\bullet)})^2,$$

or, in terms of pseudo-values

$$\hat{V}_{Jack}(\hat{\theta}) = \frac{1}{n(n-1)}\sum_{i=1}^{n}(\tilde{\theta}_i - \hat{\theta}_{corrected})^2,$$

the latter formula reminiscent of the simple mean case:

$$\hat{V}(\bar{X}) = \frac{1}{n(n-1)}\sum_{i=1}^{n}(X_i - \bar{X})^2 = \frac{s^2}{n}.$$

The jackknife estimate is based on the deviations $(\hat{\theta}_{(i)} - \hat{\theta}_{(\bullet)})$ **or** $(\tilde{\theta}_i - \hat{\theta}_{corrected})$**.**

Similarly, the bias estimate is based on the deviations $(\hat{\theta}_{(i)} - \hat{\theta})$ or $(\tilde{\theta}_i - \hat{\theta})$:

$\hat{b}_{Jack}(\hat{\theta}) = (n-1)(\hat{\theta}_{(\bullet)} - \hat{\theta})$ $= \frac{n-1}{n}\sum_{i=1}^{n}(\hat{\theta}_{(i)} - \hat{\theta})$	$\hat{b}_{Jack}(\hat{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\tilde{\theta}_i - \hat{\theta}$ $= \frac{1}{n}\sum_{i=1}^{n}(\tilde{\theta}_i - \hat{\theta})$

the left-hand expression involving jackknife replicates, the right-hand side the pseudo-values

The factor $\frac{n-1}{n}$ arises so that the jackknife estimate of variance gives the usual answer in the case of the mean. It also reflects (in the denominator) that the jackknife replicates are "too close together" if we consider the jackknife samples as mimicking repeated sampling.

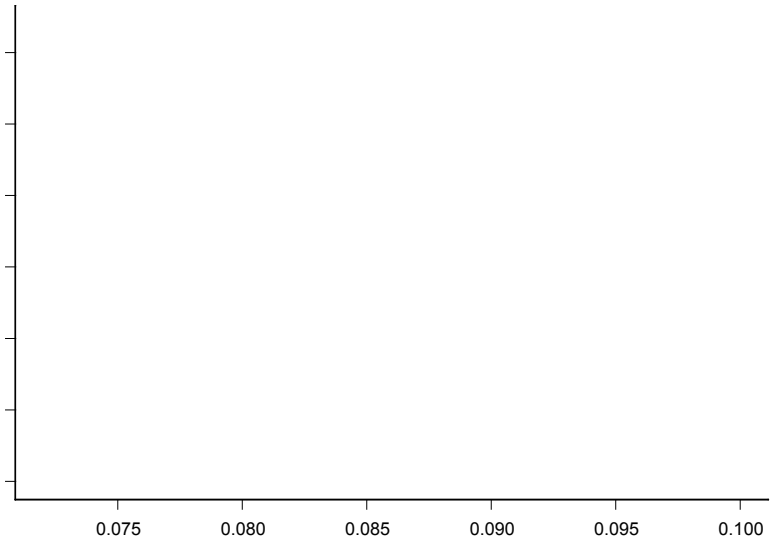
The pseudo-values act like data points – and, in fact, are data points in the case of the sample mean.

Some S-Plus functions for the jackknife…

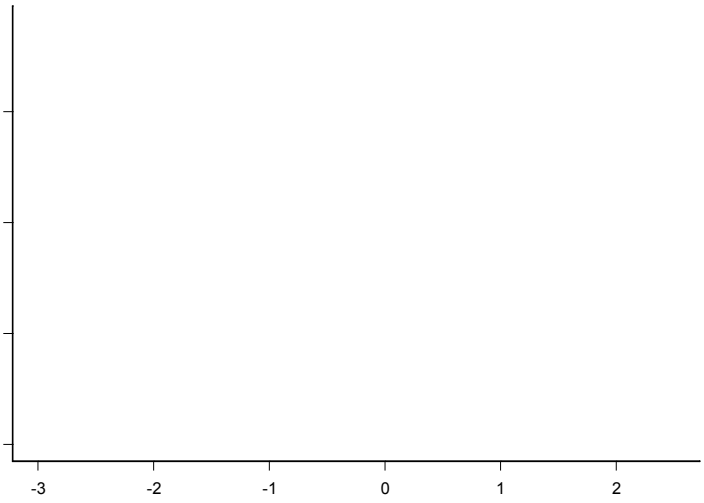
```
#  The function jackknife.bias calculates the jackknife
#  bias estimate for an estimator theta.  The input
#  parameters are:
#   1) data = The data values in either vector or matrix
#      form
#   NOTE: If matrix data is given, the observations
#   must correspond to the rows of the matrix,
#   i.e., the deletions must take place by row.
#   2) theta = The functional form of the estimator being
#      used

jackknife.bias <- function(data,theta) {
   theta.hat <- theta(data)
   theta.tld <- 0
   if (is.matrix(data)) {
     for(i in 1:length(data[,1])) {
         theta.tld[i] <- theta(data[-i,])
     }
bias <- (length(data[,1]) -1)*(mean(theta.tld) - theta.hat)
   }
   else if (is.vector(data)) {
for(i in 1:length(data)) {
          theta.tld[i] <- theta(data[-i])
       }
bias <- (length(data) - 1)*(mean(theta.tld) - theta.hat)
   }
   else print("Invalid data structure")
print("The jackknife estimate of bias is:",quote=F)
print(bias)
bias

# The following function calculates the jackknife variance
# estimate for a given statistic theta.  The inputs are:
#
#  1. Data - the input data structure, which can be either
#      a vector, or a matrix in which the rows correspond
#      to distinct realizations the random (vector-valued)
#      variable.
#  2. Theta - the desired statistic, which must be real-
```

```
#      valued and must operate on the appropriate structure
#      (i.e. a vector if data is a vector, etc.


jackknife.var <- function(data,theta) {
  theta.i <- 0
    if(is.vector(data)) {
    n <- length(data)
    for(i in 1:n) {
      theta.i[i] <- theta(data[-i])}
             }
  else if(is.matrix(data)) {
    n <- length(data[,1])
    for(i in 1:n) {
      theta.i[i] <- theta(data[-i,])
    }
  }
  else print("Invalid Data Structure",quote=F)
  ((n-1)/n)*sum((theta.i-mean(theta.i))^2)
}

# The following function produces the jackknife replicates
# theta.hat(i) for the leave-one-out samples
jackrep <- function(data, theta){
    theta.tld <- 0
    if(is.matrix(data)) {
        for(i in 1:length(data[, 1])) {
             theta.tld[i] <- theta(data[-i,])
                }
        }
    else if(is.vector(data)) {
        for(i in 1:length(data)) {
             theta.tld[i] <- theta(data[-i])
                }
        }
    else print("Invalid data structure")
    theta.tld
}

# The following function produces the pseudovalues
pseudovalues <- function(data, theta){
    theta.tld <- 0
    if(is.matrix(data)) {
       n <- length(data[, 1])
```

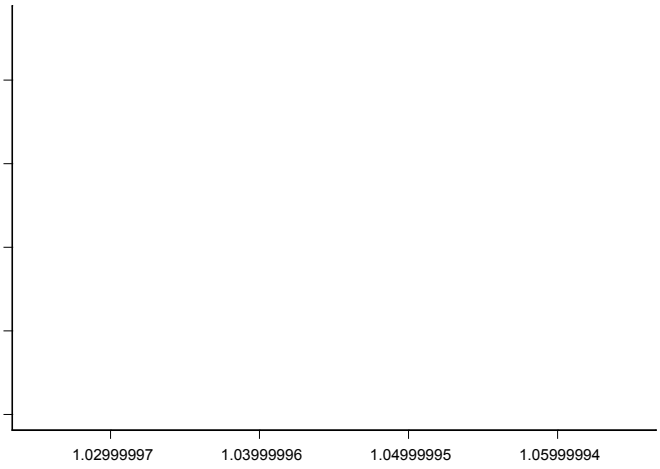
```
       for(i in 1:n) {
           theta.tld[i] <- theta(data[-i,])
                }
        }
    else if(is.vector(data)) {
         n <- length(data)
         for(i in 1:n) {
             theta.tld[i] <- theta(data[-i])
                }
        }
    else print("Invalid data structure")
pseudo <- n * theta(data) - (n - 1) * theta.tld
pseudo
}


# Now let's try it out on some data

x <- rnorm(200,0,1)
# This is simulated normal(0,1) data
theta.tld<-jackrep(x,mean)
pseudo.x<-pseudovalues(x,mean)
hist(theta.tld,xlab="Jackknife Replications",
    main="Histogram of Jackknife Replications - Normal
    Mean")
# Fig 1
abline(v=mean(x),lty=2)
```



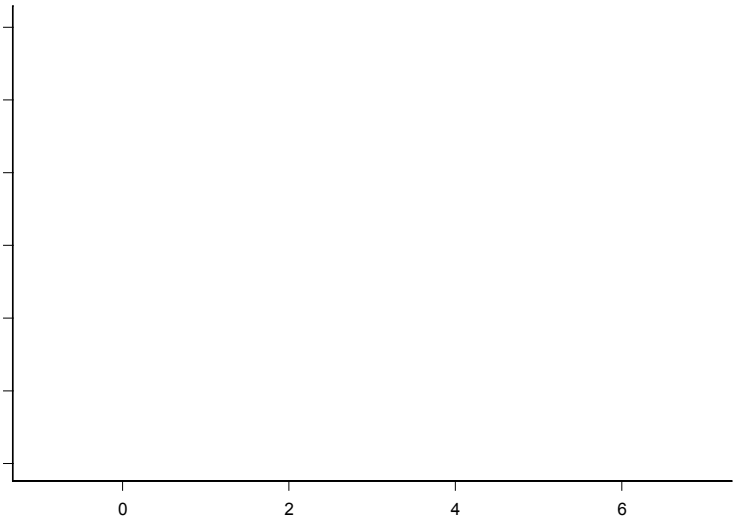
```
hist(pseudo.x,xlab="Pseudovalues",
    main="Histogram of Pseudovalues - Normal Mean")
# Fig 2
abline(v=mean(x),lty=2)
```




```
theta.tld<-jackrep(x,var)
pseudo.x<-pseudovalues(x,var)
hist(theta.tld,xlab="Jackknife Replications",
    main="Histogram of Jackknife Replications - Normal
    Variance")
# Fig 3
abline(v=var(x),lty=2)
```



```
hist(pseudo.x,xlab="Pseudovalues",
    main="Histogram of Pseudovalues - Normal Variance")
# Fig 4
abline(v=var(x),lty=2)
```




```
wrongvar<-function(data){
    wrongvar<-var(data)*(length(data)-1)/length(data)
    wrongvar
    }

# The jackknife estimate of the bias of the mean of x
should be zero
jackknife.bias(x,mean)
[1] The jackknife estimate of bias is:
[1] 5.52336e-015

# The jackknife estimate of the bias of the variance of x
# should also be zero
xvarbias<-jackknife.bias(x,var)
[1] The jackknife estimate of bias is:
[1] 1.325606e-013
```

```
# Now, if we look at the wrong variance function (the one
# that uses n instead of (n-1), we get a non-zero bias
#
xwrongvarbias<-jackknife.bias(x,wrongvar)
[1] The jackknife estimate of bias is:
[1] -0.00527471
# And if we bias-correct, we should get back the right
# variance
wrongvar(x)-xwrongvarbias
[1] 1.054942
var(x)
[1] 1.054942
# Now, let's use the jackknife to estimate the variance of
# the estimator mean(x). It should equal s^2/n
xmeanvar<-jackknife.var(x,mean)
var(x)/length(x)
[1] 0.00527471
xmeanvar
[1] 0.00527471
```

WHEN THE JACKKNIFE WORKS AND WHEN THE JACKKNIFE FAILS

The primary use of the jackknife is in bias correction and variance estimation, but it performs poorly if one uses the pseudo-values to construct confidence intervals or hypothesis tests – the intervals tend to have low coverage accuracy.

The jackknife bias estimate is itself unbiased for the true bias when the estimator in question is a *quadratic functional* (i.e. it can be expressed in a form using terms that involve the data points at most two at a time). This fact explains why the jackknife works so well for means and variances.

But the jackknife can fail spectacularly! For example, the jackknife estimate of the variance of a sample quantile $\hat{\theta}_q$ is inconsistent – in fact,

$$\frac{\hat{V}_{Jack}(\hat{\theta}_q)}{V(\hat{\theta}_q)} \xrightarrow{D} \text{Weibull}(1, \tfrac{1}{2}),$$

so, in the case of the median, the jackknife variance is unbiased for *twice* the true variance. The jackknife requires a smooth functional of F.

CROSS-VALIDATION

Basic Idea: Divide the data into two parts, at random: a training set; and a validation set. Use the training set to e.g. build a model or estimate a parameter, and then check if the inferences drawn from the training set are consistent with the validation set.

For example, take a regression data set and split it into two equal parts, randomly selecting points to be in the training set. Using the training set, build a model. For example,

$$\log(y) = 3.6 + 2.5x_1 + x_2 - 1.6x_1x_2$$

This model was fit using only half the data. For the other half, find the *predicted* response:

$$\log(\hat{y}_i) = 3.6 + 2.5x_{1i} + x_{2i} - 1.6x_{1i}x_{2i}, \; i \in \text{Validation Set}$$

Now calculate a measure of how well the model fit on the first half of the data predicted the second half, a PRESS (PREdiction Sum of Squares) statistic:

$$\sum_{i \in \text{Validation Set}} \{\log(\hat{y}_i) - \log(y_i)\}^2 .$$

K-FOLD CROSS-VALIDATION

General version of cross-validation partitions the data at random into K (usually equal-size) parts.

For *i* ranging from 1 to K,

- **Fit a model based on each of the $K - 1$ subsets of the data other than the *i*'th**
- **Predict for the data in the *i*'th subset**
- **Compute a PRESS for the *i*'th subset**

Now, combine the K estimates of prediction error, perhaps by averaging

Cross-validation differs from the other resampling methods described here as it is more a "sample re-use" rather than "resampling" method, and it doesn't really mimic the "repeated sampling" theme of the other methods – as such we will not discuss it further here.

PERMUTATION TESTS

The oldest resampling method – introduced in the 1930's by R A Fisher, but essentially shelved until recent times when computing power became enough to allow their use.

Setting: Suppose we have sets of data

$$X_1, X_2, ..., X_n$$

and

$$Y_1, Y_2, ..., Y_m,$$

***m* not necessarily equal to *n*. We wish to test whether X and Y arise from the same distribution (so if F is the distribution function of X, and G is the distribution of Y, we want to test $F = G$.)**

Under the hypothesis that *F* and *G* are, in fact, the same, then we can think of $X_1, X_2, ..., X_n$ and $Y_1, Y_2, ..., Y_m$ as a single sample of size $n + m$ from the common distribution.

IDEA: Sample from all $n + m$ data points at random without replacement and allocating the first n points to an X sample, and the remaining m points to a Y sample. This process simulates repeated sampling under the assumption that the parent distributions are the same.

For example, suppose the X sample is {1,2,3} and the Y sample is {4,5}, then possible *permutations* include X:{1,2,4}, Y:{3,5} and X:{2,4,5}, Y:{1,3}, and so on. Even for small n and m, the number of permutations can be quite large.

Now for each permutation, compute the value of some test statistic which measures disparity between the distributions of X and Y. There are many possible choices! For example,

$$\overline{X} - \overline{Y}, \text{ (e.g. close to 0 if } F = G)$$

$$\overline{X} / \overline{Y}, \text{ (e.g. close to 1 if } F = G)$$

$$median(X) - median(Y)$$

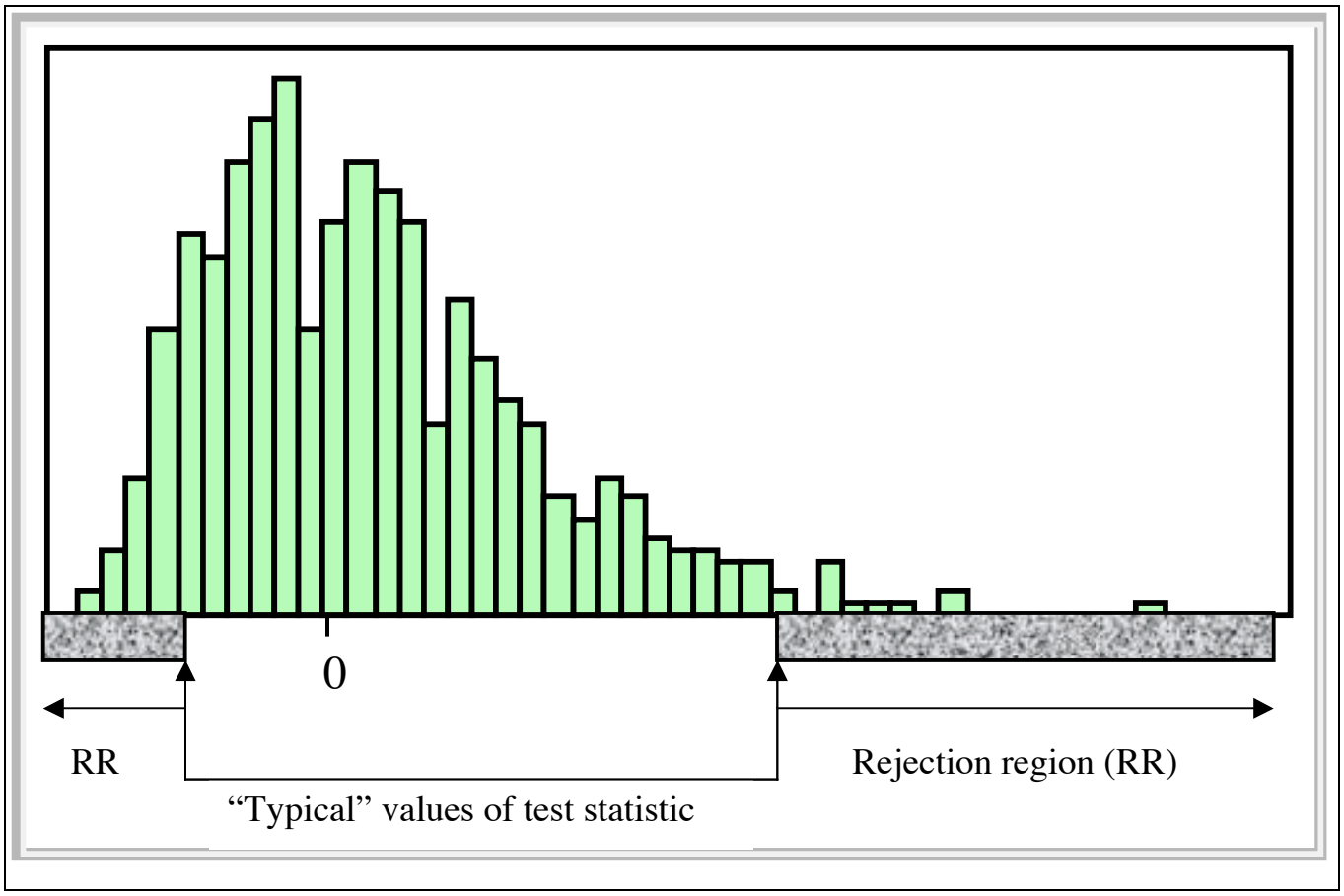
$$s_X - s_Y, s_X^2 - s_Y^2, s_X / s_Y, s_X^2 / s_Y^2$$

"Unusual" values of the test statistic support the hypothesis that the two distributions differ, and "unusual" is defined in terms of the null distribution of the test statistic. Unlike traditional tests, where null distributions are often simply tabulated, here we generate the null distribution using our simulated "repeated sampling" – our permutations.

For each permutation, calculate the value of the test statistic, and construct a histogram of the resultant values over a large number of permutations.

There are, totally, $\binom{n+m}{n}$ **possible permutations (assuming distinct data points), so usually a subset of these is chosen randomly.**

This histogram is an approximation to the sampling distribution of the test statistic under the null hypothesis. Comparing the test statistic observed using the original data with the quantiles of the histogram just computed yields a permutation test.



Permutation Distribution of a test statistic under *F=G*

If the observed test statistic lies in the shaded region, then the data is inconsistent with the assumption that *F=G* and the null hypothesis is rejected.

The shaded region is found by locating appropriate quantiles on the histogram – define

$$\text{ASL} = \frac{\#(\text{Permutation values} \geq \text{Observed Test Statistic})}{\text{Total number of permutation values}}$$

Large values of ASL (near 1) mean that the observed test statistic is in the far right tail of the permutation distribution, while small values (near 0) mean the observed statistic is in the far left tail of the distribution.

Estimate the *p*-value of the test using

$$2 \cdot \min\{\text{ASL}, (1 - \text{ASL})\}.$$

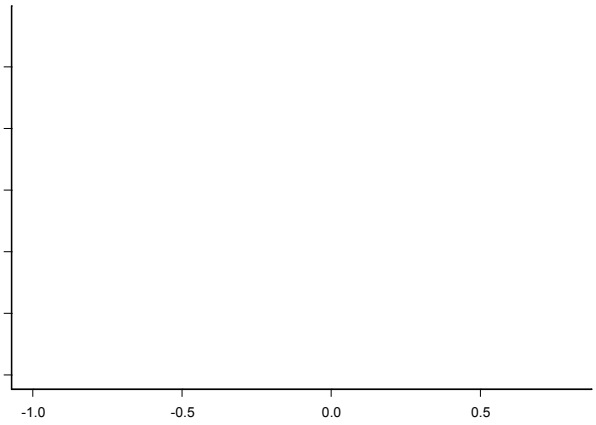
Alternatively, the test can be carried out with level α by rejecting the null hypothesis if

$$\min\{\text{ASL}, (1 - \text{ASL})\} < \frac{\alpha}{2}.$$

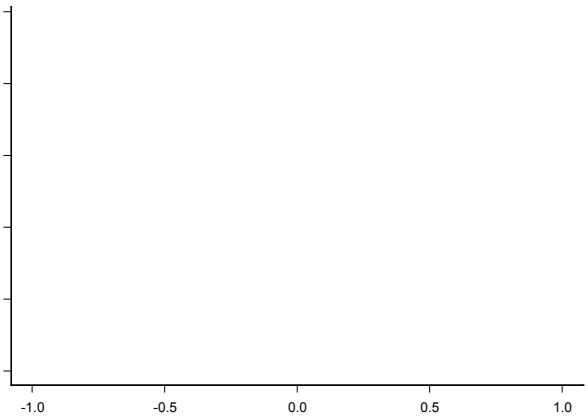
S-Plus routines for constructing a permutation test are given on the following slide. The first function returns the set of values of a test statistic (denoted `teststat`) for a particular number of permutations (`nreps`). The second function returns the associated value of ASL when the result of a call to the preceding function is given as input.

```
# This function computes the values of the
# permutation distribution of a test statistic for X
# data called data1 and Y data called data2, nreps
# is the desired number of permutations
permreps <- function(data1, data2, teststat, nreps){
 alldata <- c(data1, data2)
 perm <- 0
 for(i in 1:nreps) {
   permute <- sample(alldata)
   permdata1 <- permute[1:length(data1)]
   permdata2 <- permute[(length(data1) +
                          1):length(alldata)]
   perm[i] <- teststat(permdata1, permdata2)
   }
 perm
}


# Here is an example of a test statistic

teststat1 <- function(data1, data2)
{
        teststat1 <- mean(data1) - mean(data2)
        teststat1
}


# Here is a function to calculate ASL (actual
# significance level) and associated p-value

asl <- function(data1, data2, teststat, permdata){
  observed <- teststat(data1, data2)
  asl <- sum(observed <= permdata)/length(permdata)
  pval <- min(asl,1-asl)*2
 c(asl,pval)
}


# Now for some examples:
```

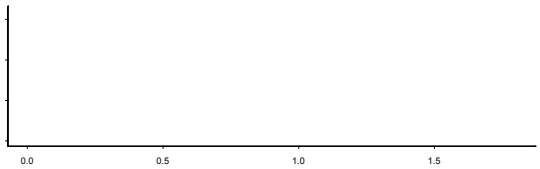
```
# First, a simple question where x and y are normal
# with identical means

x<-rnorm(30,0,1)
y<-rnorm(25,0,1)

reps<-permreps(x,y,teststat1,1000)
hist(reps,xlab="Values of Test Statistic for
     Permutations", main="Permutation distribution for
     Normal Data")
# Fig 5
abline(v=teststat1(x,y),lty=2)
```




```
teststat1(x,y)
[1] 0.314872
asl(x,y,teststat1,reps)
[1] 0.115 0.230

# Now an example where X and Y are normal with means
# further apart

x<-rnorm(30,0,1)
y<-rnorm(25,0.5,1)
reps<-permreps(x,y,teststat1,1000)
hist(reps,xlab="Values of Test Statistic for
     Permutations", main="Permutation distribution for
     Normal Data, different means")
# Figure 6
abline(v=teststat1(x,y),lty=2)
```

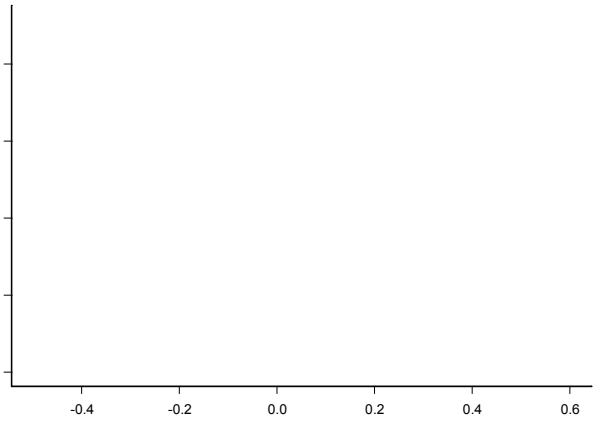



```
teststat1(x,y)
[1] -0.7232461
asl(x,y,teststat1,reps)
[1] 0.998 0.004

# What about an example where X and Y have different
# distributions, but the same mean?

x<-rexp(40,2)
y<-rnorm(50,0.5,1)
par(mfrow=c(2,1))
hist(x,xlab="Exponential Data")
hist(y,xlab="Normal Data")
# Fig 7
```

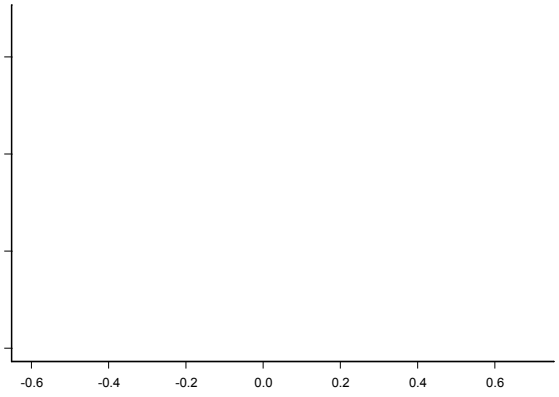



```
par(mfrow=c(1,1))
reps<-permreps(x,y,teststat1,1000)
hist(reps,xlab="Values of Test Statistic for
     Permutations", main="Permutation distribution for
     Normal/exponential data, same mean")
```

```
# Fig 8
abline(v=teststat1(x,y),lty=2)
```




```
teststat1(x,y)
[1] 0.1345616
asl(x,y,teststat1,reps)
[1] 0.214 0.428

# Different distributions, different means
x<-rexp(40,2)
y<-rnorm(20,0,1)
reps<-permreps(x,y,teststat1,1000)
hist(reps,xlab="Values of Test Statistic for
     Permutations", main="Permutation distribution for
     Normal/exponential data, diff mean")
# Fig 9
abline(v=teststat1(x,y),lty=2)
```




```
teststat1(x,y)
[1] 0.6361796
```

```
asl(x,y,teststat1,reps)
[1] 0 0

# Could try different test statistics too
# for example:

teststat1<-function(data1,data2){
teststat1<-mean(data1)/mean(data2)
teststat1
}
# OR
teststat1<-function(data1,data2){
teststat1<-var(data1)/var(data2)
teststat1
}
# OR OTHERS STILL!
```

THE BOOTSTRAP

Origins: "To pull oneself up by one's bootstraps"

Attributed to Baron Munchausen…

"I have ever confined myself to facts," declared Baron Munchausen, according to Rudolf Raspe, author of *The Travels and Surprising Adventures of Baron Munchausen*. And surprising indeed were the Baron's adventures, including a voyage to an island composed entirely of cheese (and, incidentally, judged by the Baron to be larger than Europe), two voyages to the Moon (one admittedly accidental), and salvation from death by drowning in quicksand by elevating himself from the quicksand using only his bootstraps. The real Baron Munchausen took little solace in the telling of his legendary feats. He was regarded widely as a celebrated liar, and was besieged by tourists who clamoured to hear the "Euclid of liars" in action. He died in 1797, killed, legend has it, by the misery caused by Raspe, his unauthorized biographer.

– from my review of Efron and Tibshirani in *Chance*

In fact, a careful reading of *The Travels and Surprising Adventures of Baron Munchausen* by Raspe, available from

ftp://mirror.aarnet.edu.au/pub/gutenberg/etext02/baron10.txt **, reveals no tale of rescue by his own bootstraps, but it is certainly consistent with his other feats!**

(The word also has (considerable!) currency in the computer world, to boot(strap) up meaning a computer self-executing code to start operating… but it's only fair statistics borrow "their" word, as computer science obtained the words "byte" and "software" from a statistician – John Tukey.)

BASIC PREMISE: Simulate repeated sampling either by using the data plus knowledge of the underlying distribution (parametric bootstrap – Monte Carlo simulation) or just the data itself (nonparametric bootstrap)

THE PARAMETRIC BOOTSTRAP

If you knew your data arose from a *N*(0,1) distribution, "repeated sampling" would simply involve (literally!) repeatedly drawing same-size samples from the known *N*(0,1) distribution. The distribution of some quantity, say $\sqrt{n}\overline{X}$, can then be approximated using values of the quantity computed for each of the repeated samples. Now in this case, we know that $\sqrt{n}\overline{X} \sim N(0,1)$.

How?

- **Prove it mathematically (easy using mgf's!)**
- **"Verify" it empirically – look at the histogram of values of $\sqrt{n}\overline{X}$ computed under the repeated sampling.**

Of course, in the real world, you can't assume data derives from a *N*(0,1) distribution, so you might assume a $N(\mu,\sigma^2)$, with parameters μ and σ unknown. Repeated sampling now involves sampling from this distribution – but of course the parameters are unknown, so we use the obvious plug-in estimates, and sample instead from a $N(\bar{X},s^2)$ distribution, adopting a plug-in principle popularised by von Mises in the 1930's. Now, for each sample, we could construct the value of some quantity, say $\sqrt{n}\bar{X}$ again, and use the values so produced to estimate its sampling distribution. This process is called Monte Carlo simulation, and its use raises some critical questions…

- **Does it matter what you plug in? Here we plugged in $\bar{X}$ for μ and s for σ, but we might just have easily plugged in the sample median for μ or the MLE $\frac{1}{n}\sum_{i=1}^{n}(X_i-\bar{X})^2$ for σ^2. What differences might we expect under different choices, and are these difference important?**

- **We inevitably incur sampling error which we can control by varying the number of samples we draw, but if *n* is small our plug-in values may simply be poor no matter what we do**

- **What if our data wasn't Normal to start with? What if it were Gamma? Well, we could adopt a similar plug-in principle with a Gamma distribution.**

- **The Central Limit Theorem can help in terms of what we might expect to find under repeated sampling, but sample size is an important consideration, as is model specification**

Monte Carlo simulation is an old technique, now rebadged *the parametric bootstrap*, but as with all parametric methods, belief in the underlying probability model is critical.

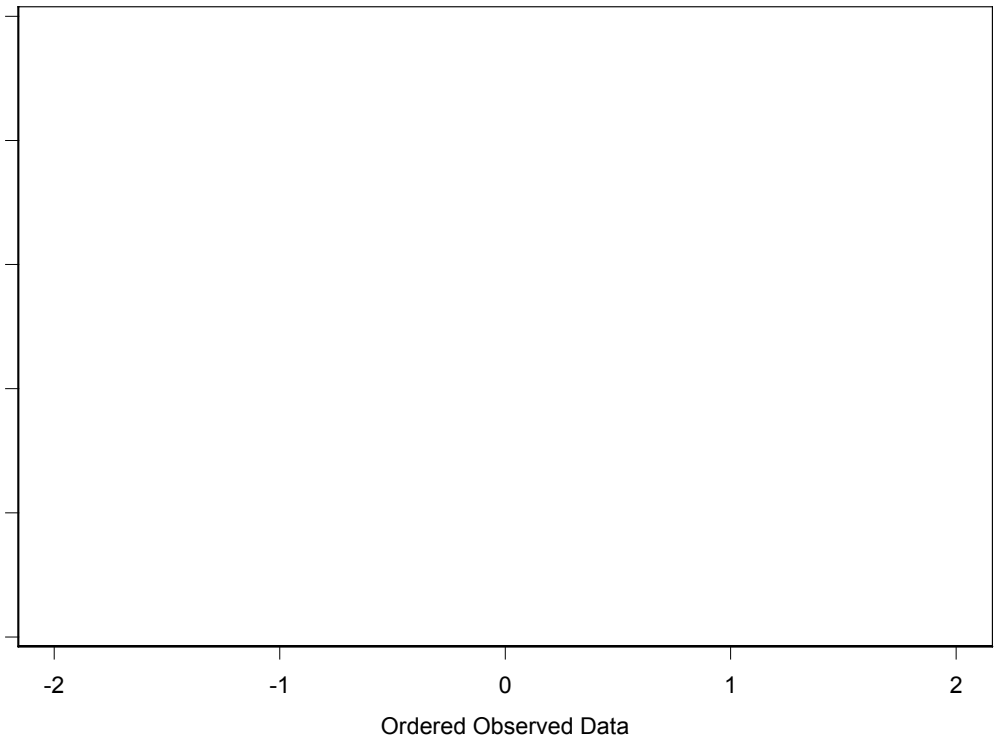
THE NONPARAMETRIC BOOTSTRAP – OR JUST "THE BOOTSTRAP"

What if you don't know or aren't willing to guess what the distribution from which the data arose was? What other alternatives are there?

A very simple alternative is the *empirical distribution*

$$\hat{F}(x) = \frac{\#(X_i \leq x)}{n},$$

where n is sample size, which represents the proportion of the observed data not exceeding x.



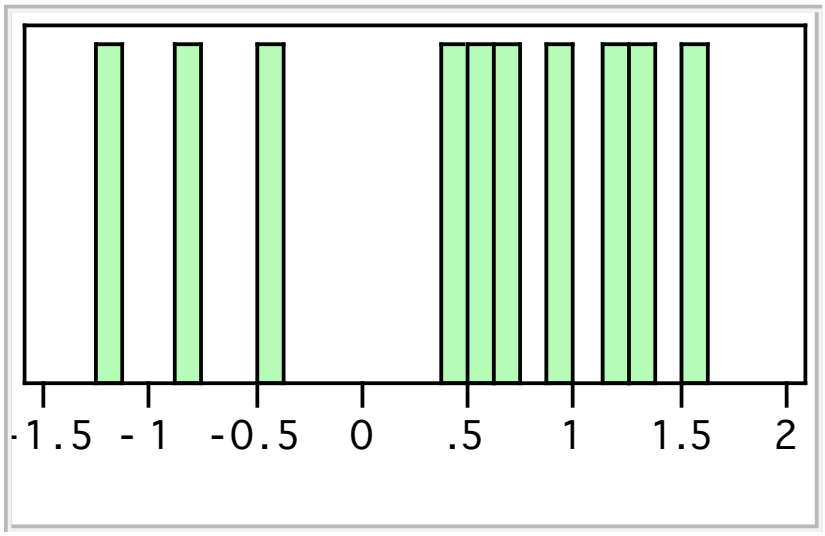
For i.i.d. data and under mild conditions,

$$\left|\hat{F}(x) - F(x)\right| = O_p(n^{-\frac{1}{2}}),$$

and $\hat{F}(x)$ estimates the true underlying distribution F well (in "normal deviations" regions) for even small values of n.

The bootstrap paradigm involves sampling from $\hat{F}(x)$ in order to simulate repeated sampling from F.

How do we sample from $\hat{F}(x)$? It helps to think of $\hat{F}(x)$ a different way: as the distribution of a random variable whose probability mass function places weight $\frac{1}{n}$ at each of the observed data points:



Sampling from $\hat{F}(x)$:

Resample: $X_1^*,....,X_n^*$ **from** $X_1,...,X_n$ **using the rule**

$$P(X_i^* = X_j) = \frac{1}{n}, \text{ for all } i, j.$$

If $n = 5$**, and** $\{X_1,...,X_n\} = \{1.1, 2.3, 4, 3.5, 2.1\}$**, possible (unordered) resamples include**

$$\{1.1, 1.1, 2.3, 4, 2.1\}, \{2.3, 1.1, 1.1, 2.3, 3.5\},$$

$$\{3.5, 3.5, 4, 1.1, 1.1\}, \{1.1, 1.1, 1.1, 1.1, 1.1\},$$

$$\{3.5, 3.5, 3.5, 4, 3.5\}, \{1.1, 2.3, 4, 3.5, 2.1\}$$

with $\{1.1, 1.1, 1.1, 1.1, 1.1\}$ **the least likely, and** $\{1.1, 2.3, 4, 3.5, 2.1\}$ **the most likely.**

Bootstrap resampling involves sampling *with* replacement from n **equally-likely data points. By contrast, permutation tests used without-replacement sampling.**

The nature of bootstrap resampling makes clear the importance of two assumptions: clearly, in resampling, the points sampled can appear *in any order* and *in roughly equal proportions*. The *in any order* statement suggests that independence (or at least exchangeability) is an important assumption; the *in roughly equal*

proportions **part suggests that the data should arise from the same distribution. As a result, the basic assumption is that the data is i.i.d.**

Both of these assumptions can be relaxed, but doing so requires significant modification to the bootstrap algorithm to allow for resampling independent "units" with like structure.

HOW MANY RESAMPLES?

In principle, "repeated sampling" in the traditional sense can be an *ad infinitum* thought experiment. But the finite nature of the data means that bootstrapping cannot be conducted this way, even in thought (and certainly not in deed).

Luckily, even for small n the number of potential resamples is large: exponential in n at least. The total number of possible distinct resamples is $\binom{2n-1}{n}$ (assuming distinct data values).

(A proof of this fact is, like many combinatorial proofs, quite delightful…place $2n-1$ points in a row and select, at random, $n-1$ of them to act as dividers. Then interpret the number of points between dividers as the number of times a particular data point appears in a particular resample. Clearly, the number of such arrangements must be the same as the total number of distinct resamples, and so the number of resamples is simply $\binom{2n-1}{n-1} = \binom{2n-1}{n}$.)

This number grows *fast* as n grows…

n	Bootstraps	n	Bootstraps
1	1	11	352,716
2	3	12	1,352,078
3	10	13	5,200,300
4	35	14	20,058,300
5	126	15	77,558,760
6	462	16	300,540,195
7	1,716	17	1,166,803,110
8	6,435	18	4,537,567,650
9	24,310	19	17,672,631,900
10	92,378	20	68,923,264,410

In practice, a random sample of resamples is drawn, and the number of resamples required for reasonable performance varies depending on

the application. On the order of 25 to 50 resamples seems sufficient for point estimation (bias, variance), while 1000 to 2000 resamples are necessary for interval estimation where tail information is needed.

Exhaustive (all possible resample) methods are useful only up to about n = 10, but these days might even be feasible up to n = 20.

Once the resamples are drawn, the value of some statistic is calculated for each resample, and the "bootstrap distribution" of the quantity is the resultant histogram, and it estimates the sampling distribution of the quantity.

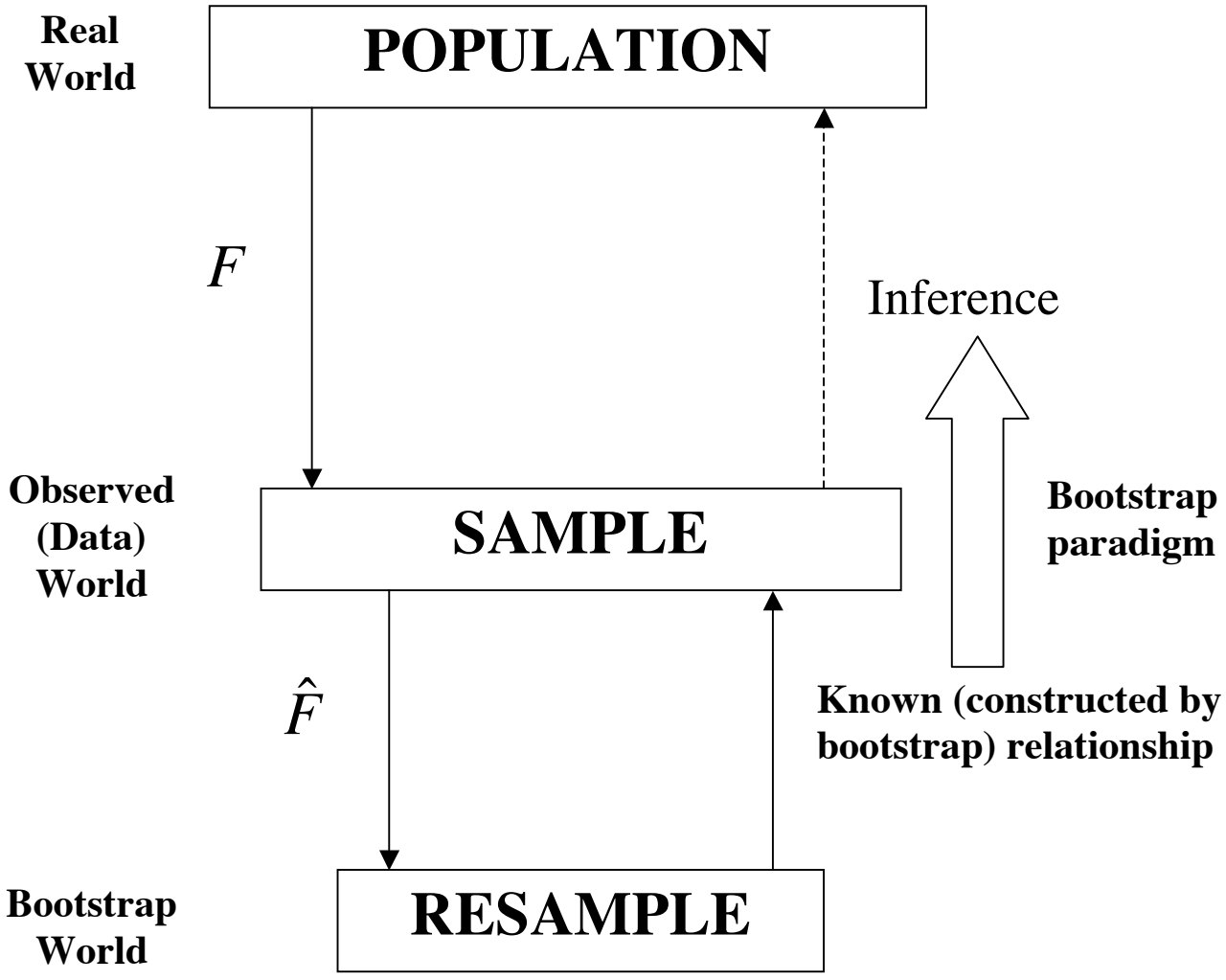
Potential problem: The true sampling distribution may be continuous, but the bootstrap distribution is discrete – with a potentially large number of atoms. Don't worry! The atoms are small: the largest, corresponding to the original data set itself, is of size only $\frac{n!}{n^n} \approx e^{-n}$, while the smallest is of size $\frac{1}{n^n}$.

SO, WHY (HOW?) DOES THE BOOTSTRAP WORK?

Many statistics of interest are functionals of the underlying distribution function: $\theta = \theta(F)$. For example, the mean, $\theta = \int x\, dF$. A natural "plug-in" estimator of θ replaces F by $\hat{F}$, the empirical cdf, so the "bootstrap estimator" of θ is $\hat{\theta} = \theta(\hat{F})$, the same functional of $\hat{F}$. Informally, because $\hat{F} \approx F$, provided θ is a reasonably smooth functional, then $\hat{\theta} \approx \theta$.

Each resample, conditional on the data, has distribution function $\hat{F}$, so "repeated sampling" versions of $\hat{\theta} = \theta(\hat{F}) = \theta(X_1, \ldots, X_n)$ are simply equal to $\hat{\theta}^* = \theta(X_1^*, \ldots, X_n^*)$, and the bootstrap distribution of $\hat{\theta}$ (represented as the histogram of $\hat{\theta}^*$ values, and conditional on the original data) estimates the sampling distribution of $\hat{\theta}$.

A SCHEMATIC DIAGRAM FOR THE BOOTSTRAP



BOOTSTRAP BIAS ESTIMATION

The bias of an estimator $\hat{\theta}$,

$$b(\hat{\theta}) = E_F(\hat{\theta}) - \theta$$
$$= E_F\{\theta(\hat{F})\} - \theta(F)$$

is a functional of the underlying distribution and also of the empirical cdf. Using the "plug-in principle" here involves replacing population quantities by sample quantities and sample quantities by resample quantities:

$$\hat{b}_{Boot}(\hat{\theta}) = E_{\hat{F}}(\hat{\theta}^*) - \hat{\theta}$$
$$= E_{\hat{F}}\{\theta(\hat{F}^*)\} - \theta(\hat{F})$$
$$= E_{\hat{F}}\{\theta(X_1^*, ..., X_n^*)\} - \theta(X_1, ..., X_n)$$

where $\hat{F}^*$ is the empirical cdf of a resample $(X_1^*, ..., X_n^*)$. Hence, as in the schematic diagram above, the bootstrap paradigm involves shifting emphasis from a "population – sample" setting to a "sample – resample" setting, and then using the information constructed in that setting as if it were relevant in the original setting.

The bootstrap bias formula involves the quantity $E_{\hat{F}}(\hat{\theta}^*)$, which could be calculated exactly if we had access to all possible resamples. In practical terms, we resort to a Monte Carlo algorithm to approximate it:

Step 1: Given a sample $\{X_1,...,X_n\}$ compute $\hat{\theta}$;

Step 2: Resample $\{X_1^*,...,X_n^*\}$ from $\{X_1,...,X_n\}$ and compute $\hat{\theta}^* = \hat{\theta}\{X_1^*,...,X_n^*\}$;

Step 3: Repeat Step 2 a large number, say B, times, hence obtaining B values $\hat{\theta}_1^*,\hat{\theta}_2^*,...,\hat{\theta}_B^*$;

Step 4: The approximation for $E_{\hat{F}}(\hat{\theta}^*)$ is the average of the $\hat{\theta}^*$ values : $\hat{E}_{\hat{F}}(\hat{\theta}^*) = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}_b^*$;

Step 5: The bootstrap bias estimate is therefore

$$\hat{b}_{Boot}(\hat{\theta}) = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}_b^* - \hat{\theta},$$

and the bootstrap bias-corrected estimate is

$$\hat{\theta}_{Boot,corrected} = \hat{\theta} - \hat{b}_{Boot}(\hat{\theta}) = 2\hat{\theta} - \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}_b^*,$$

which typically has bias an order of magnitude lower than that of $\hat{\theta}$.

BOOTSTRAP ESTIMATION OF VARIANCE

The variance of $\hat{\theta}$ is defined by

$$V(\hat{\theta}) = E_F[\{\hat{\theta} - E_F(\hat{\theta})\}^2] = E_F(\hat{\theta}^2) - \{E_F(\hat{\theta})\}^2.$$

The bootstrap estimator of $V(\hat{\theta})$, using the plug-in principle adopted earlier is:

$$\hat{V}_{Boot}(\hat{\theta}) = E_{\hat{F}}\{(\hat{\theta}^*)^2\} - \{E_{\hat{F}}(\hat{\theta}^*)\}^2,$$

where once again expectation with respect to $\hat{F}$ can be calculated exactly if all resamples are enumerated. A Monte Carlo approximation can be employed using the "bootstrap values" $\hat{\theta}_1^*, \hat{\theta}_2^*, \ldots, \hat{\theta}_B^*$:

Let $\bar{\theta}^* = \frac{1}{B}\sum_{b=1}^{B} \hat{\theta}_b^*$ represent the average of the $\hat{\theta}_1^*, \hat{\theta}_2^*, \ldots, \hat{\theta}_B^*$ values, and using the obvious substitutions (replacing means by averages) we arrive at

$$\tilde{V}_{Boot}(\hat{\theta}) = \frac{1}{B}\sum_{b=1}^{B} \{\hat{\theta}_b^*\}^2 - \left\{\frac{1}{B}\sum_{b=1}^{B} \hat{\theta}_b^*\right\}^2$$

$$= \frac{1}{B}\sum_{b=1}^{B} (\hat{\theta}_b^* - \bar{\theta}^*)^2,$$

which, of course, is biased for the true variance, since it uses a divisor of B rather than $B-1$. The more common, unbiased, bootstrap variance estimator is

$$\hat{V}_{Boot}(\hat{\theta}) = \frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}_b^* - \bar{\theta}^*)^2,$$

which is simply the sample variance of the bootstrap quantities $\hat{\theta}_1^*, \hat{\theta}_2^*, \ldots, \hat{\theta}_B^*$.

Important to remember: there are now two types of sampling error to consider, n-error and B-error. As users we have complete control over the B-error, and can make it as small as we like by choosing B appropriately large. We're stuck with the n-error.

Typically for bias and variance estimation, around 25-50 bootstrap replications are usually sufficient (though as usual, the more the better, and this prescription depends on the type of estimator, of course).

S-Plus routines for bootstrap bias and variance estimation follow, inputs including the data, the estimator, and the number of resamples, B.

```
boot.bias <- function(data, theta, B){
 theta.star <- 0
 if(is.vector(data)) {
 for(i in 1:B) {
   theta.star[i] <- theta(sample(data,replace=T))
   }
 }
 else if(is.matrix(data)) {
 for(i in 1:B) {
 theta.star[i] <-
  theta(data[sample(1:length(data[,1]),replace=T),])
 }
}
else print("Invalid Data Structure", quote = F)
mean(theta.star - theta(data))
}



boot.var <- function(data, theta, B){
 theta.star <- 0
 if(is.vector(data)) {
   for(i in 1:B) {
    theta.star[i] <- theta(sample(data,replace=T))
 }
}
else if(is.matrix(data)) {
 for(i in 1:B) {
 theta.star[i] <-
 theta(data[sample(1:length(data[,1]),replace=T),])
}
}
 else print("Invalid Data Structure", quote = F)
var(theta.star)
}
```

```
# Now for some examples.

x_rexp(20,2)   # True values, mean=0.5, var=0.25
mean(x)
[1] 0.4283532
var(x)
[1] 0.1007585

# The mean is unbiased, so the bootstrap bias should
# be close to zero

boot.bias(x,mean,100)
[1] -0.001455913

# Does the number of resamples make much difference?
boot.bias(x,mean,1000)
[1] 0.00299887

# wrongvar is the wrong variance estimator (using n
# not n-1). It is biased.
wrongvar(x)
[1] 0.09572057
boot.bias(x,wrongvar,1000)
[1] -0.004959735

# Following is a bias-corrected version of wrongvar
wrongvar(x)-boot.bias(x,wrongvar,1000)
[1] 0.1006314

# Is it close to the right answer (given by var)
var(x)
[1] 0.1007585

# Now, we'll use the bootstrap to estimate the
variance of xbar.


# It should be sigma^2/n
```

```
boot.var(x,mean,100)
[1] 0.005419886
boot.var(x,mean,1000)
[1] 0.004940765
var(x)/length(x)
[1] 0.005037925

# We can try the same thing on normal data
x_rnorm(20,0,1)
mean(x)
[1] 0.2906218
var(x)
[1] 0.8985458
# The mean is unbiased, so the bootstrap bias should
# be close to zero
boot.bias(x,mean,100)
[1] -0.01038963

# Does the number of resamples make much difference?
boot.bias(x,mean,1000)
[1] 0.002884271

# wrongvar is the wrong variance estimator (using n
# not n-1). It is biased.
wrongvar(x)
[1] 0.8536185
boot.bias(x,wrongvar,1000)
[1] -0.04603256
# Following is a bias-corrected version of wrongvar
wrongvar(x)-boot.bias(x,wrongvar,1000)
[1] 0.8960173
# Is it close to the right answer (given by var)
var(x)
[1] 0.8985458
# Now, we'll use the bootstrap to estimate the
variance of xbar.
# It should be sigma^2/n
boot.var(x,mean,100)
[1] 0.04423756
```

```
boot.var(x,mean,1000)
[1] 0.04286407
var(x)/length(x)
[1] 0.04492729

# We can also try to estimate bias and variance for
# other statistics, eg the correlation coefficient.

correl_function(data){
correl_cor(data)[1,2]
correl
}
data_cysfib
correl(data)
[1] -0.2713439
boot.bias(data,correl,1000)
[1] 0.005327988
boot.var(data,correl,1000)
[1] 0.04766051
varrho(data[,1], data[,2])
[1] 0.04264581
```

BOOTSTRAP CONFIDENCE INTERVALS

The "standard $100(1-\alpha)\%$ **confidence interval"**

$$\hat{\theta} \pm z_{1-\frac{\alpha}{2}} \text{ s.e.}(\hat{\theta})$$

makes the assumption that the standardised quantity $\dfrac{\hat{\theta}-\theta}{\text{s.e.}(\hat{\theta})}$ **has a Standard Normal distribution.**

[Note for later: The interval might be better written as $[\hat{\theta} - z_{1-\frac{\alpha}{2}} \text{ s.e.}(\hat{\theta}), \hat{\theta} - z_{\frac{\alpha}{2}} \text{ s.e.}(\hat{\theta})]$ **(even though for a Standard Normal distribution** $z_{\frac{\alpha}{2}} = -z_{1-\frac{\alpha}{2}}$**) to stress the fact that for intervals constructed using location-scale pivots, it is the upper critical point of the distribution of the pivot that appears in the formula for the lower confidence limit, and vice-versa.]**

Our discussion of bootstrap confidence intervals is based on the concept of pivots, quantities whose (asymptotic) distributions do not depend on unknown parameters.

The various bootstrap methods replace the Standard Normal assumption by bootstrap estimates of the sampling distributions of various "pivots".

We will discuss several kinds of bootstrap confidence intervals, each based on different approximate "pivots":

- **Percentile method – simplest method, called "backwards" by Hall (1988)**
- **Another Percentile method – variation on the percentile method, called "hybrid" by Hall**
- **Percentile-*t* method – also bootstrap-*t***
- **BC_a method – based on a refined Normal approximation**
- **ABC method – an approximation to the BC_a interval with low computational requirements**

THE PERCENTILE METHOD

Premise: The bootstrap distribution of $\hat{\theta}^*$ (that is, the distribution of $\hat{\theta}^*$ under repeated sampling from $\hat{F}$) is close to the distribution of $\hat{\theta}$ under F.

Quantiles of the bootstrap distribution of $\hat{\theta}^*$ are then used as interval endpoints; mathematically, if $\hat{G}$ represents the cdf of the bootstrap distribution of $\hat{\theta}^*$, then the interval $[\hat{G}^{-1}(\frac{\alpha}{2}), \hat{G}^{-1}(1-\frac{\alpha}{2})]$ is a $100(1-\alpha)\%$ percentile method interval for θ.

But how does the interval perform? In principle, the percentile interval does no better than the standard interval, at least in terms of coverage accuracy. It is an asymmetric interval, which means its shape may be more appropriate than that of the standard interval. Hall (1988), however, argues that it is "backwards" insofar as it uses the lower critical point of the distribution of its implied pivot in constructing the lower confidence limit, the opposite of the situation in classical location-scale inference.

- **An example of how the percentile interval "gets it wrong"…**

Suppose X **arises from a** $N(\mu, \sigma^2)$ **distribution, with** μ **and** σ **unknown. Then, under *parametric* bootstrapping, the distribution of** $\hat{\theta}^* = \bar{X}^*$ **is** $N(\bar{X}, \frac{s^2}{n})$**, while the distribution of** $\hat{\theta} = \bar{X}$ **is** $N(\mu, \frac{\sigma^2}{n})$**. The percentile method assumes these distributions are the same, but there are two clear departures: the use of** $\bar{X}$ **to approximate** μ **and the use of** s **to approximate** σ**.**

- **Cases in which the percentile method "gets it right"…**

If there exists a monotone transformation g **of** $\hat{\theta}$ **for which** $\hat{\phi} = g(\hat{\theta}) \sim N(\phi, \sigma^2)$**, where** $\phi = g(\theta)$ **and** σ **is a known constant, then the percentile method works well. In this case, we can think of the percentile method as being based on the pivot** $\frac{\hat{\phi} - \phi}{\sigma}$**. Moreover, the percentile method is transformation-respecting, so the practitioner need not know what the right transformation is,**

just that there is one! This is a major improvement on the standard interval. An excellent example of this phenomenon is the correlation coefficient, $\hat{\zeta}$, where Fisher's $\tanh^{-1}$ function effectively normalises $\hat{\zeta}$ to a constant variance scale.

PROS AND CONS OF PERCENTILE METHOD INTERVALS

- ✔ **Easy to calculate**
- ✔ **Easy to understand**
- ✔ **Stable endpoints**
- ✔ **Transformation-respecting**
- ✔ **Range-respecting**

- ✘ **Poor coverage accuracy**
- ✘ **Sometimes wrong "shape" (right/left)***
- ✘ **Often no better than standard interval – for a lot of extra computation!**

*** For example, if the distribution of the estimator is skewed left, this indicates that the observed value of the estimator is likely too low. In this case, an interval with a longer *upper* side is desirable, yet the percentile method produces an interval with a longer *lower* side.**

ALGORITHM FOR CONSTRUCTING PERCENTILE METHOD INTERVALS

Given data $X_1,...,X_n$ and an estimator $\hat{\theta}$ of θ.

Step 1: Resample $\{X_1^*,...,X_n^*\}$ from $\{X_1,...,X_n\}$ and compute $\hat{\theta}^* = \hat{\theta}\{X_1^*,...,X_n^*\}$;

Step 2: Repeat Step 1 a large number (B) of times to obtain B values

$$\hat{\theta}_1^*, \hat{\theta}_2^*, \ldots, \hat{\theta}_B^*.$$

Step 3: Order the $\hat{\theta}_b^*$ values to obtain

$$\hat{\theta}_{[1]}^* \le \hat{\theta}_{[2]}^* \le \cdots \le \hat{\theta}_{[B]}^*.$$

Step 4: Define the $100(1-\alpha)\%$ confidence interval for θ to be

$$\left[\hat{\theta}_{\left[\frac{\alpha B}{2}+1\right]}^*, \hat{\theta}_{\left[\frac{(1-\alpha)B}{2}+1\right]}^*\right].$$

So for a 95% percentile method interval

$$\left[\hat{\theta}_{[0.025B+1]}^*, \hat{\theta}_{[0.975B+1]}^*\right],$$

or the 2.5% and 97.5% quantiles on the bootstrap histogram.

THE OTHER PERCENTILE METHOD (HYBRID)

Premise: The bootstrap distribution of $\hat{\theta}^* - \hat{\theta}$ (that is, the distribution of $\hat{\theta}^* - \hat{\theta}$ under repeated sampling from $\hat{F}$) is close to the distribution of $\hat{\theta} - \theta$ under F.

Quantiles of the bootstrap distribution of $\hat{\theta}^* - \hat{\theta}$ are then used to construct interval endpoints; mathematically, if $\hat{H}$ represents the cdf of the bootstrap distribution of $\hat{\theta}^* - \hat{\theta}$, then the interval $[\hat{\theta} - \hat{H}^{-1}(1 - \frac{\alpha}{2}), \hat{\theta} - \hat{H}^{-1}(\frac{\alpha}{2})]$ is a $100(1-\alpha)\%$ (hybrid) percentile method interval for θ. Note that this interval can also be expressed in terms of the bootstrap cdf $\hat{G}$ of $\hat{\theta}^*$, as

$$[2\hat{\theta} - \hat{G}^{-1}(1 - \tfrac{\alpha}{2}), 2\hat{\theta} - \hat{G}^{-1}(\tfrac{\alpha}{2})]$$

But how does the interval perform? Again, no better than the standard interval, at least in terms of coverage accuracy. It is an asymmetric interval – in fact, it sometimes gets the shape more "right" than the original percentile interval, and it follows the classical "location-scale" pivot approach insofar as the lower

critical point of the bootstrap distribution is used in the construction of the upper confidence limit.

- **An example of how the new percentile interval "gets it wrong" (but arguably more "right" than the original percentile method)**

Suppose X **arises from a** $N(\mu,\sigma^2)$ **distribution, with** μ **and** σ **unknown. Then, under *parametric* bootstrapping, the distribution of** $\hat{\theta}^* - \hat{\theta} = \bar{X}^* - \bar{X}$ **is** $N(0,\frac{s^2}{n})$**, while the distribution of** $\hat{\mu} - \mu = \bar{X} - \mu$ **is** $N(0,\frac{\sigma^2}{n})$**. The new percentile method assumes these distributions are the same, but there is one clear departure: the use of** s **to approximate** σ**.**

- **The new percentile method interval "gets it right" in some of the same ways the original percentile interval does – it works best when there is a monotone transformation of the approximate pivot to a constant variance, zero-mean Normal model.**

PROS AND CONS

Same as for the original percentile interval except it is neither range-respecting nor transformation-respecting

WHICH PERCENTILE INTERVAL TO USE?

The distinction between the two types of percentile intervals lies almost exclusively in their shape (right/left). The new percentile interval works better when a traditional "location-scale" model is appropriate, while the original percentile method works better in other cases such as the correlation coefficient and the median (where the percentile method reproduces a sign-test interval). Problem is: in most practical cases, it is hard to tell *a priori* which one will work better…

ALGORITHM FOR CONSTRUCTING (HYBRID) PERCENTILE INTERVALS

Given data $X_1,...,X_n$ and an estimator $\hat{\theta}$ of θ.

Step 1: Resample $\{X_1^*,...,X_n^*\}$ from $\{X_1,...,X_n\}$ and compute

$$\hat{\theta}^* = \hat{\theta}^* - \hat{\theta} = \hat{\theta}\{X_1^*,...,X_n^*\} - \hat{\theta}\{X_1,...,X_n\};$$

Step 2: Repeat Step 1 a large number (B) of times to obtain B values

$$\hat{\theta}_1^*, \hat{\theta}_2^*, \ldots, \hat{\theta}_B^*.$$

Step 3: Order the $\hat{\theta}_b^*$ values to obtain

$$\hat{\theta}_{[1]}^* \le \hat{\theta}_{[2]}^* \le \cdots \le \hat{\theta}_{[B]}^*.$$

Step 4: Define the $100(1-\alpha)\%$ confidence interval for θ to be

$$\left[\hat{\theta} - \hat{\theta}^*_{\left[\frac{(1-\alpha)B}{2}+1\right]}, \hat{\theta} - \hat{\theta}^*_{\left[\frac{\alpha B}{2}+1\right]}\right] = \left[2\hat{\theta} - \hat{\theta}^*_{\left[\frac{(1-\alpha)B}{2}+1\right]}, 2\hat{\theta} - \hat{\theta}^*_{\left[\frac{\alpha B}{2}+1\right]}\right].$$

So for a 95% hybrid percentile interval

$$\left[\hat{\theta} - \hat{\theta}^*_{[0.975B+1]}, \hat{\theta} - \hat{\theta}^*_{[0.025B+1]}\right] = \left[2\hat{\theta} - \hat{\theta}^*_{[0.975B+1]}, 2\hat{\theta} - \hat{\theta}^*_{[0.025B+1]}\right].$$

PERCENTILE-*t* INTERVAL

Premise: The bootstrap distribution of $\frac{\hat{\theta}^* - \hat{\theta}}{s^*}$

(that is, the distribution of $\frac{\hat{\theta}^* - \hat{\theta}}{s^*}$ **under repeated sampling from** $\hat{F}$**) is close to the distribution of** $\frac{\hat{\theta} - \theta}{s}$ **under** F**, where** $s = s(X_1, \ldots, X_n)$ **is an estimate of the standard error of** $\hat{\theta}$ **and** $s^* = s(X_1^*, \ldots, X_n^*)$ **is its bootstrap version.**

Quantiles of the bootstrap distribution of $\frac{\hat{\theta}^* - \hat{\theta}}{s^*}$ **are then used to construct interval endpoints; mathematically, if** $\hat{K}$ **represents the cdf of the bootstrap distribution of** $\frac{\hat{\theta}^* - \hat{\theta}}{s^*}$**, then the interval** $[\hat{\theta} - s\hat{K}^{-1}(1-\frac{\alpha}{2}), \hat{\theta} - s\hat{K}^{-1}(\frac{\alpha}{2})]$ **is a** $100(1-\alpha)\%$ **percentile-*t* method interval for** θ**.**

The percentile-*t* interval performs well *provided an adequate estimate of the standard error of $\hat{\theta}$ is available*. "Adequate" means more than just calculable! It means it must perform well in small samples and be fairly easy to calculate. Nevertheless, the interval endpoints produced by the percentile-*t* method are *second-order correct* meaning that they agree with theoretically "correct" endpoints to terms of order $O(n^{-1})$. In this sense, it gets the endpoints more "correct" than either of the percentile method intervals (which are each only first-order correct, like those of the classical "standard interval").

- **An example of how the percentile-*t* interval "gets it right"…**

Suppose X arises from a $N(\mu,\sigma^2)$ distribution, with μ and σ unknown. Then, under *parametric* bootstrapping, the distribution of $\dfrac{\hat{\theta}^* - \hat{\theta}}{s^*} = \dfrac{\bar{X}^* - \bar{X}}{s^*/\sqrt{n}}$ where s on the right-hand side is the usual unbiased variance estimator, is $t_{(n-1)}$, while the

distribution of $\frac{\hat{\theta}-\theta}{s} = \frac{\bar{X}-\mu}{s/\sqrt{n}}$ is also $t_{(n-1)}$. The percentile-*t* method assumes these distributions are the same – which they are, exactly, in this case.

• But the percentile-*t* method can get it wrong too…

Coverage isn't everything! If the scale estimate is not stable enough, especially in small samples, the percentile-*t* method can go awry. For example, in the case of the correlation coefficient, percentile-*t* intervals can go well outside the $[-1,1]$ range of the correlation coefficient. Unfortunately, like the standard interval, the percentile-*t* interval is not range-respecting. Neither is it transformation-respecting, so it does not share the advantage of the percentile interval of automatically benefiting from the mere existence of such a normalising transformation.

PROS AND CONS OF PERCENTILE-*t* METHOD INTERVALS

- ✔ **Good coverage accuracy**
- ✔ **Correct "shape" – second-order correct interval endpoints**
- ✔ **Usually better than the standard interval, particularly if a good standard error estimate is available**

- ✘ **Not transformation-respecting**
- ✘ **Not range-respecting**
- ✘ **What if there is no good standard error estimate available?**
- ✘ **Hard to understand without statistical training (is this necessarily a bad thing?)**

ALGORITHM FOR CONSTRUCTING PERCENTILE-t METHOD INTERVALS

Given data $X_1,\ldots,X_n$, an estimator $\hat{\theta}$ of θ, and an estimator s of the standard error of $\hat{\theta}$.

Step 1: Resample $\{X_1^*,\ldots,X_n^*\}$ from $\{X_1,\ldots,X_n\}$ and compute

$$\tilde{\theta}^* = \frac{\{\hat{\theta}(X_1^*,\ldots,X_n^*) - \hat{\theta}(X_1,\ldots,X_n)\}}{s(X_1^*,\ldots,X_n^*)};$$

Step 2: Repeat Step 1 a large number (B) of times to obtain B values

$$\tilde{\theta}_1^*, \tilde{\theta}_2^*, \ldots, \tilde{\theta}_B^*.$$

Step 3: Order the $\tilde{\theta}_b^*$ values to obtain

$$\tilde{\theta}_{[1]}^* \le \tilde{\theta}_{[2]}^* \le \cdots \le \tilde{\theta}_{[B]}^*.$$

Step 4: Define the $100(1-\alpha)\%$ confidence interval for θ to be

$$\left[\hat{\theta} - s\tilde{\theta}^*_{\left[\frac{(1-\alpha)B}{2}+1\right]}, \hat{\theta} - s\tilde{\theta}^*_{\left[\frac{\alpha B}{2}+1\right]}\right].$$

So for a 95% percentile-t method interval

$$\left[\hat{\theta} - s\tilde{\theta}^*_{[0.975B+1]}, \hat{\theta} - s\tilde{\theta}^*_{[0.025B+1]}\right].$$

ACCELERATED BIAS-CORRECTED (BC_a) INTERVALS

Premise: The percentile methods tacitly assume that there exists a monotone transformation g of $\hat{\theta}$ for which $\hat{\phi} = g(\hat{\theta}) \sim N(\phi, \sigma^2)$, where $\phi = g(\theta)$ and σ is a known constant. The BC_a interval assumes that a Normalising transformation g exists, but that the resultant model is potentially biased and has non-constant variance. Instead, it allows that

$$\frac{\hat{\phi} - \phi}{\sigma(1 + a\phi)} \sim N(-z_0, 1)$$

[or $\frac{\hat{\phi} - \phi}{\sigma} \sim N(-z_0(1 + a\phi), (1 + a\phi)^2)$ **] for constants z_0 (a biasing constant) and a (the acceleration constant – so-called as it measures the rate of change of the standard error with respect to the normalised parameter).**

The BC_a method adjusts the quantiles of the distribution of $\hat{\theta}^*$ to account for the bias and accelerated variance of the Normal approximation above. The adjustment is non-trivial to derive but is based on the assumed

Standard Normal distribution of the quantity $\frac{\hat{\theta}-\theta}{\sigma(1+a\theta)}+z_0$**. The resultant interval is** $\left[\hat{G}^{-1}(\alpha_1),\hat{G}^{-1}(\alpha_2)\right]$ **where**

$$\alpha_1=\Phi\left(\hat{z}_0+\frac{\hat{z}_0+z_{\frac{\alpha}{2}}}{1-\hat{a}\left(\hat{z}_0+z_{\frac{\alpha}{2}}\right)}\right)$$

$$\alpha_2=\Phi\left(\hat{z}_0+\frac{\hat{z}_0+z_{1-\frac{\alpha}{2}}}{1-\hat{a}\left(\hat{z}_0+z_{1-\frac{\alpha}{2}}\right)}\right)$$

Efron (1987) developed the BC_a **interval as an adjustment to the percentile interval to allow for a more flexible model for the distribution of** $\frac{\hat{\theta}-\theta}{\sigma}$**. Thought of in this way, the** BC_a **method is an attempt to approximate the distribution of this "pivot" better than the percentile method does by allowing the limiting distribution to incorporate appropriate bias and variance terms.**

Hall (1988) explained the BC_a approach differently, as an attempt to correct for the fact that the endpoints of the percentile interval are not second-order correct. The end result, of course, is the same.

WHAT ARE $\hat{z}_0$ and $\hat{a}$?

A key part of implementing the BC_a approach is the estimation of the biasing and acceleration constants. The biasing constant is easy to estimate, since

$$G(\theta) = P(\hat{\theta} < \theta) = P(Z < z_0) = \Phi(z_0),$$

so, using the usual "plug-in" approach,

$$\hat{G}(\hat{\theta}) = P_{\hat{F}}(\hat{\theta}^* < \hat{\theta}) = \Phi(\hat{z}_0),$$

and $\hat{z}_0 = \Phi^{-1}\{\hat{G}(\hat{\theta})\}$.

The acceleration constant is trickier! A simple expression for it, based on the jackknife applied to $\hat{\theta}$ is

$$\hat{a} = \frac{\sum_{i=1}^{n} \left(\hat{\theta}_{(\bullet)} - \hat{\theta}_{(i)}\right)^3}{6\left\{\sum_{i=1}^{n} \left(\hat{\theta}_{(\bullet)} - \hat{\theta}_{(i)}\right)^2\right\}^{\frac{3}{2}}}$$

PROS AND CONS OF BC_a METHOD INTERVALS

- ✔ **Good coverage accuracy**
- ✔ **Correct "shape" – second-order correct interval endpoints**
- ✔ **Transformation-respecting**
- ✔ **Range-respecting**

- ✘ **Hard to understand, particularly for people with little statistical training**
- ✘ **Relatively complicated to compute, particularly the acceleration constant**
- ✘ **The acceleration constant essentially reflects estimation of skewness in the underlying population, and its estimation generally works better in large samples.**

ABC METHOD INTERVALS

Premise: An analytical approximation to BC_a intervals designed to avoid the need for significant computation

The algorithm replaces bootstrap quantities in the BC_a algorithm by analytical Taylor series approximations

The ABC method shares the advantages of the BC_a method, and can be computed in a small fraction of the time that BC_a intervals can be constructed.

COMPUTATIONAL NOTES:

In order to use the S-Plus routines for the ABC method, the statistic of interest must be put in "resampling form". This involves finding the estimated proportions

$$p_i^* = \frac{\#(X_j^* = X_i)}{n}, \quad i = 1,\ldots,n,$$

and expressing the statistic of interest in terms of these proportions. For example, for a mean,

$$\bar{X}^* = \sum_{i=1}^{n} p_i^* X_i,$$

while for a correlation coefficient, we must use the (rather cumbersome) expression:

$$\rho^* = \frac{\sum_{i=1}^{n} p_i^* (X_i - \bar{X}^*)(Y_i - \bar{Y}^*)}{\sqrt{\sum_{i=1}^{n} p_i^* (X_i - \bar{X}^*)^2 \sum_{i=1}^{n} p_i^* (Y_i - \bar{Y}^*)^2}},$$

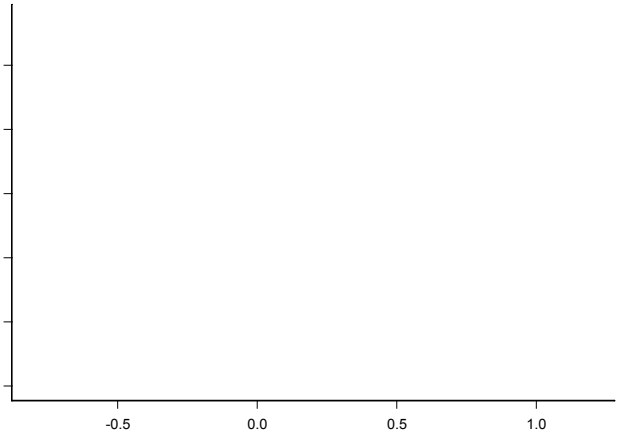
where here the p_i^*'s refer to the resampling proportions for pairs $(X_i, Y_i), i = 1,\ldots,n$.

```
# Function that returns the bootstrap replications
# of theta
bootreps <- function(data, theta, B){
 theta.star <- 0
 if(is.vector(data)) {
   for(i in 1:B) {
    theta.star[i]<-theta(sample(data, replace=T))
    }
 }
 else if(is.matrix(data)) {
   for(i in 1:B) {
    theta.star[i] <-
theta(data[sample(1:length(data[,1]), replace=T),])
   }
}
else print("Invalid Data Structure", quote = F)
theta.star
}

# Function that returns a percentile interval of
# level (1-alpha) from B resamples, given data and
# theta
percentile <- function(data, theta, alpha, B){
 theta.star <- 0
 if(is.vector(data)) {
   for(i in 1:B) {
    theta.star[i]<-theta(sample(data,replace = T))
   }
 }
 else if(is.matrix(data)) {
  for(i in 1:B) {
    theta.star[i] <-
theta(data[sample(1:length(data[,1]), replace=T),])
    }
}
else print("Invalid Data Structure", quote = F)
sorted <- sort(theta.star)
```

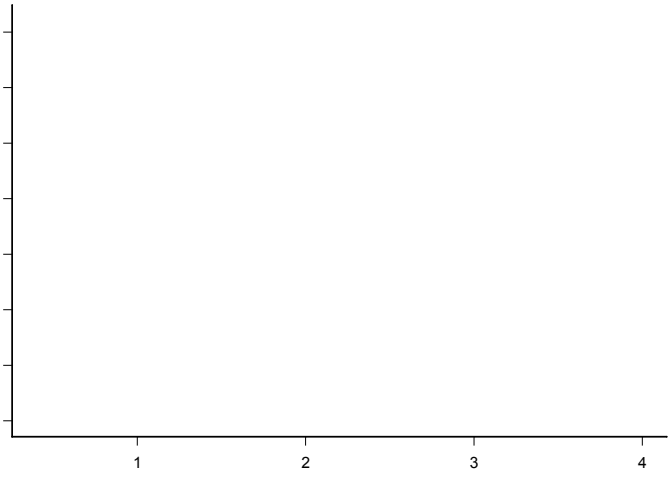
```
interval <- c(sorted[trunc((alpha*B)/2)+1],
sorted[trunc((1 - alpha/2)*B)+1])
interval
}

# Percentile (type 2) interval of level (1-alpha),
# from B resamples
percentile2 <- function(data, theta, alpha, B){
 thetahat <- theta(data)
 phi.star <- 0
   if(is.vector(data)) {
    for(i in 1:B) {
     phi.star[i]<- theta(sample(data, replace = T))
                       -thetahat
    }
 }
else if(is.matrix(data)) {
   for(i in 1:B) {
     phi.star[i] <-
theta(data[sample(1:length(data[,1]), replace=T),])
                     -theta(data)
   }
}
else print("Invalid Data Structure", quote = F)
sorted <- sort(phi.star)
interval<-c(thetahat-sorted[trunc((1-alpha/2)*B)+1],
       thetahat-sorted[trunc((alpha*B)/2)+1])
interval
}

# Percentile-t interval of level (1-alpha), B
# resamples
percentile.t <- function(data,theta,se,alpha,B){
thetahat <- theta(data)
sehat <- se(data, theta, B)
t.star <- 0
if(is.vector(data)) {
  for(i in 1:B) {
    resample <- sample(data, replace = T)
```

```
    t.star[i] <- (theta(resample)-thetahat)/se(
          resample, theta, B)
    }
  }
  else if(is.matrix(data)) {
    for(i in 1:B) {
    resample <- data[sample(1:length(data[, 1]),
           replace = T),]
    t.star[i] <- (theta(resample) - theta(data))/se(
        resample, theta, B)
    }
  }
  else print("Invalid Data Structure", quote = F)
  sorted <- sort(t.star)
  interval <- c(thetahat - sehat * sorted[trunc((1-
        alpha/2) * B) + 1], thetahat - sehat *
        sorted[trunc((alpha * B)/2) + 1])
  interval
}

# Examples of se functions for the correlation
# coefficient and the mean
secorr <- function(data){
        secorr <- sqrt(varrho(data[, 1], data[, 2]))
        secorr
}
semean <- function(data){
        semean <- sqrt(var(data)/length(data))
        semean
}

# Remember the correlation coefficient variance
# estimator, varrho
# Routines for Bca and ABC methods obtained from
# Statlib — see Efron and Tibshirani for discussion

# First, some examples of bootstrap histograms
x <- rnorm(20,0,1)
thetastar_bootreps(x,mean,1000)
```

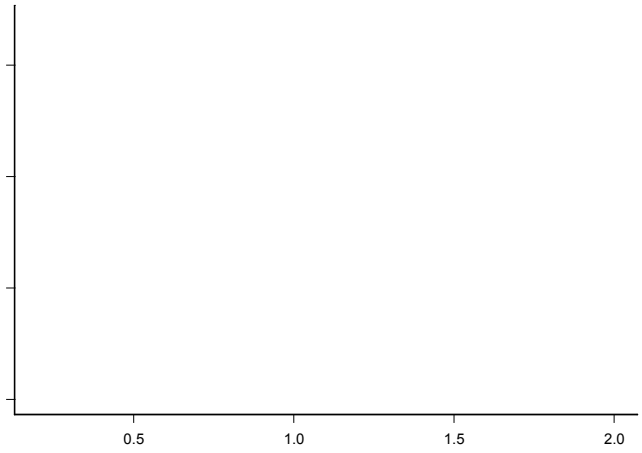
```
hist(thetastar,xlab="Bootstrap means",
    main="Histogram of Bootstrap Means - Normal
    Data")
# Fig 10
abline(v=mean(x),lty=2)
```



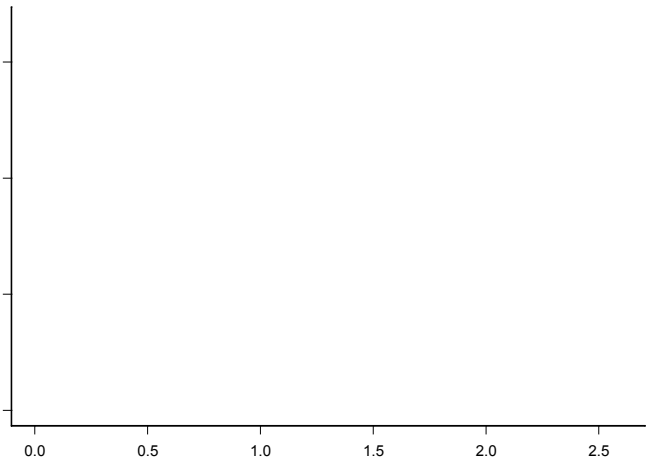

```
thetastar_bootreps(x,var,1000)
hist(thetastar,xlab="Bootstrap variances",
  main="Histogram of Bootstrap Variances - Normal
  Data")
# Fig 11
abline(v=var(x),lty=2)
```



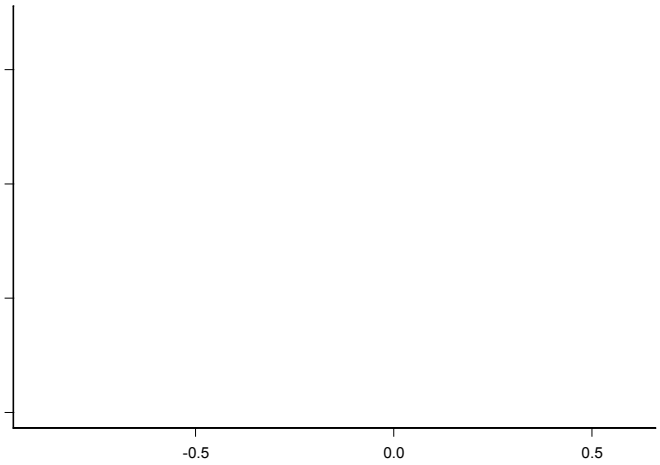
```
x_rexp(20,1)
thetastar_bootreps(x,mean,1000)
hist(thetastar,xlab="Bootstrap means",
  main="Histogram of Bootstrap Means - Exponential
  Data")
# Fig 12
abline(v=mean(x),lty=2)
```




```
thetastar_bootreps(x,var,1000)
hist(thetastar,xlab="Bootstrap variances",
    main="Histogram of Bootstrap Variances -
    Exponential Data")
# Fig 13
abline(v=var(x),lty=2)
```



```
data_cysfib
thetastar_bootreps(data,correl,2000)
hist(thetastar,xlab="Bootstrap correlation",
    main="Histogram of Bootstrap Correlations -
    Cystic Fibrosis Data")
# Fig 14
abline(v=correl(data),lty=2)
```




```
# Confidence interval for the mean, normal data
x <- rnorm(20,0,1)
standard_c(mean(x)-1.96*sqrt(var(x)/length(x)),
    mean(x)+1.96*sqrt(var(x)/length(x)))
standard
[1] -0.4014690  0.6238077
standard.t_c(mean(x)-qt(0.975,19)* sqrt(var(x)/
    length(x)),mean(x)+qt(0.975,19)*sqrt(var(x)/
    length(x)))
standard.t
[1] -0.4362615  0.6586001

percentile(x,mean,0.05,1000)
[1] -0.3995051  0.6368763
percentile2(x,mean,0.05,1000)
[1] -0.3612026  0.6045193
```

```
percentile.t(x,mean,semean,0.05,1000)
[1] -0.4226867  0.6639950
bcanon(x,1000,mean,alpha=c(0.025,0.975))$confpoints[,2]
[1] -0.3894215  0.5937195
meanp <- function(p,x){sum(p*x)/sum(p)}
abcnon(x,meanp,alpha=c(0.025,0.975))$limits[,2]
[1] -0.3727008  0.6274266

x <- rexp(20,1)
standard_c(mean(x)-1.96*sqrt(var(x)/length(x)),
    mean(x)+1.96*sqrt(var(x)/length(x)))
standard
[1] 0.3932507 1.0027099
standard.t_c(mean(x)-qt(0.975,19)* sqrt(var(x)/
    length(x)),mean(x)+qt(0.975,19)*sqrt(var(x)/
    length(x)))
standard.t
[1] 0.3725688 1.0233918
percentile(x,mean,0.05,1000)
[1] 0.4213207 1.0219574
percentile2(x,mean,0.05,1000)
[1] 0.3191787 0.9504203
percentile.t(x,mean,semean,0.05,1000)
[1] 0.4231254 1.1892949
bcanon(x,1000,mean,alpha=c(0.025,0.975))$confpoints[,2]
[1] 0.4785055 1.0519839
meanp <- function(p,x){sum(p*x)/sum(p)}
abcnon(x,meanp,alpha=c(0.025,0.975))$limits[,2]
[1] 0.4660487 1.0945041

# Now back to our real cystic fibrosis data

standard_c(correl(data)-
    1.96*sqrt(varrho(data[,1],data[,2])),
    correl(data)+1.96*
    sqrt(varrho(data[,1],data[,2])))
standard
[1] -0.6761008  0.1334129
percentile(data,correl,0.05,2000)
```

```
[1] -0.6696097  0.2042439
percentile2(data,correl,0.05,2000)
[1] -0.7490091  0.1211898
percentile.t(data,correl,secorr,0.05,2000)
[1] -0.7100600  0.4237678
correl1<-function(x,xdata){
    cor(xdata[x,1],xdata[x,2])}
bcanon(1:length(data[,1]),1000,correl1,data,
    alpha=c(0.025,0.975))$confpoints[,2]
[1] -0.6292060  0.2365949
correlp <- function(p,x){
    x1m <-sum(p*x[,1])/sum(p)
    x2m <-sum(p*x[,2])/sum(p)
    num <-sum(p*(x[,1]-x1m)*(x[,2]-x2m))
    den <- sqrt(sum(p*(x[,1]-x1m)^2)*
        sum(p*(x[,2]-x2m)^2))
    return(num/den)}
abcnon(data,correlp,alpha=c(0.025,0.975))$limits[,2]
[1] -0.6145012  0.2370864


# Do the intervals always work?
xx_rnorm(20,0,1)
yy_2*xx+rnorm(20,0,0.25)
data_cbind(xx,yy)
standard_c(correl(data)-
    1.96*sqrt(varrho(data[,1],data[,2])),
    correl(data)+
    1.96*sqrt(varrho(data[,1],data[,2])))
standard
[1] 0.9786347 0.9966983
percentile(data,correl,0.05,2000)
[1] 0.9706586 0.9945641
percentile2(data,correl,0.05,2000)
[1] 0.9808264 1.0028662
percentile.t(data,correl,secorr,0.05,2000)
[1] 0.9709581 0.9953369
```

```
# Simulated highly correlated data

xx_rnorm(10,0,1)
yy_2*xx+rnorm(10,0,0.25)
cor(xx,yy)
[1] 0.993517
standard_c(correl(data)-
 1.96*sqrt(varrho(data[,1],data[,2])),correl(data)+
 1.96*sqrt(varrho(data[,1],data[,2])))
standard
[1] 0.9786347 0.9966983
percentile(data,correl,0.05,1000)
[1] 0.9724515 0.9945202
percentile2(data,correl,0.05,1000)
[1] 0.9809758 1.0050530
percentile.t(data,correl,secorr,0.05,1000)
[1] 0.9687005 0.9949720

# Correlated log normal data
xx_exp(xx)
yy_exp(yy)
cor(xx,yy)
[1] 0.9391768
data_cbind(xx,yy)
standard_c(correl(data)-
    1.96*sqrt(varrho(data[,1],data[,2])),
    correl(data)+
    1.96*sqrt(varrho(data[,1],data[,2])))
standard
[1] 0.8683067 1.0100469
percentile(data,correl,0.05,2000)
[1] 0.8652852 0.9977460
percentile2(data,correl,0.05,2000)
[1] 0.8818033 1.0102349
percentile.t(data,correl,secorr,0.05,2000)
[1] -1.3440657  0.9838303
```

BOOTSTRAP HYPOTHESIS TESTS

Approach 1: Construct a bootstrap confidence interval and define the rejection region as the complement of the confidence interval.

This approach uses the duality between confidence intervals and hypothesis testing, but it does not allow for resampling to be carried out *under the null hypothesis* and resampling schemes that allow this are likely to be better.

Approach 2: Set up a formal testing framework, develop a test statistic, and use the bootstrap to develop its distribution under the null hypothesis. This approach requires the user to think about what properties resamples should possess under the null.

A simple way to proceed is to discuss bootstrap hypothesis testing in the context of several examples…

A BOOTSTRAP HYPOTHESIS TEST FOR THE MEAN

Suppose we have a sample $X_1, \ldots, X_n$ and we wish to test whether or not the mean of the population from which the data arises is equal to $\mu_0 = 10$. A natural test statistic is

$$T_{obs} = \frac{\bar{X} - \mu_0}{s/\sqrt{n}} = \frac{\bar{X} - 10}{s/\sqrt{n}},$$

as for classical inference. Of course, under the assumption that the data arose from a Normal distribution, T would have a $t_{(n-1)}$ distribution under the null hypothesis, but here we make no normal assumption.

A bootstrap test would construct bootstrap versions of T, but it is important that the resampling is carried out *under the assumption that the true population mean is* 10. To assure this, let $Y_i = X_i - \bar{X} + 10$, and resample the Y_i's at random with replacement and construct for each resample

$$T^* = \frac{\bar{Y}^* - 10}{s_Y^*/\sqrt{n}}.$$

Resampling B times from the Y_i's yields B values $T_1,\ldots,T_B$, so the bootstrap p-value for the test is

$$2\min(\mathrm{ASL}_{Boot}, 1 - \mathrm{ASL}_{Boot}),$$

where

$$\mathrm{ASL}_{Boot} = \frac{\#(T_b^* \geq T_{obs})}{B}.$$

In this case, note that this procedure turns out to be equivalent to resampling from the original data and constructing a percentile-t interval and using this interval to form a decision rule.

Why? Because

$$T^* = \frac{\bar{Y}^* - 10}{s_Y^*/\sqrt{n}} = \frac{(\bar{X}^* - \bar{X} + 10) - 10}{s_X^*/\sqrt{n}} = \frac{\bar{X}^* - \bar{X}}{s_X^*/\sqrt{n}},$$

which is the same as the "pivot" used to construct a percentile-t interval.

For the parametric bootstrap, resampling under the null hypothesis is particularly easy: simply resample from the parametric family plugging in the hypothesised value for the parameter of interest and using bootstrap "plug-in" values for the other parameters.

```
bootmeanhyp <- function(data, mu0, B){
    nulldata <- data - mean(data) + mu0
    nulltest <- 0
    tobs <- (mean(data) - mu0)/semean(data)
    for(i in 1:B) {
        nullresample <- sample(nulldata, replace = T)
        nulltest[i] <- (mean(nullresample) - mu0)/semean(
            nullresample)
    }
    asl <- sum(nulltest <= tobs)/B
    pval1side <- min(asl, 1 - asl)
    pval2side <- 2 * pval1side
    cbind(tobs, pval1side, pval2side)
}

bootmeanhyp2 <- function(data, mu0, B){
    test <- 0
    tobs <- (mean(data) - mu0)/semean(data)
    for(i in 1:B) {
        resample <- sample(data, replace = T)
        test[i] <- (mean(resample) - mean(data))/semean(
            resample)
    }
    asl <- sum(test <= tobs)/B
    pval1side <- min(asl, 1 - asl)
    pval2side <- 2 * pval1side
    cbind(tobs, pval1side, pval2side)
}

# First, try with some Normal data
x <- rnorm(10,0,1)
bootmeanhyp(x,0,1000)
          tobs pval1side pval2side
[1,] -1.200594     0.195      0.39
bootmeanhyp2(x,0,1000)
          tobs pval1side pval2side
[1,] -1.200594     0.196     0.392

x <- rnorm(10,0,1)
bootmeanhyp(x,1,1000)
          tobs pval1side pval2side
[1,] -1.071226     0.166     0.332
bootmeanhyp2(x,1,1000)
          tobs pval1side pval2side
```

```
[1,] -1.071226     0.168     0.336
mean(x)
[1] 0.4401312
var(x)/length(x)
[1] 0.2731557

x <- rnorm(10,0,0.5)
bootmeanhyp(x,1,1000)
          tobs pval1side pval2side
[1,] -10.23893         0         0
bootmeanhyp2(x,1,1000)
          tobs pval1side pval2side
[1,] -10.23893         0         0

x <- rnorm(20,0,1)
bootmeanhyp(x,-0.2,1000)
           tobs pval1side pval2side
[1,] -0.9075589     0.206     0.412
bootmeanhyp2(x,-0.2,1000)
           tobs pval1side pval2side
[1,] -0.9075589     0.185      0.37

# Now some exponential data
x <- rexp(20,2)
bootmeanhyp(x,0,1000)
         tobs pval1side pval2side
[1,] 4.219671         0         0
bootmeanhyp(x,0.5,1000)
          tobs pval1side pval2side
[1,] 0.5455703     0.253     0.506
bootmeanhyp2(x,0.5,1000)
          tobs pval1side pval2side
[1,] 0.5455703     0.262     0.524


# Cauchy data-the bootstrap doesn't care
# even if the mean doesn't exist!
x <- rcauchy(20)
bootmeanhyp(x,0,1000)
          tobs pval1side pval2side
[1,] -1.144913       0.1       0.2
bootmeanhyp2(x,0,1000)
          tobs pval1side pval2side
[1,] -1.144913     0.107     0.214
```

BOOTSTRAP TEST FOR A TWO-SAMPLE PROBLEM

Here we revisit the problem of comparing two distributions previously discussed using permutation tests – the bootstrap can address this problem as well.

Under the null hypothesis, $F = G$, so when resampling, pool the X and Y data into a single group of size $n + m$ and resample (with replacement) from this group with resamples also of size $n + m$, assigning the first n elements of the resample to be X_i^*'s, and the rest to be Y_j^*'s. This is a very similar process to that carried out for permutation tests, but in this case sampling is *with* replacement.

Now, to test, for example, $\mu_X = \mu_Y$, set the test statistic to be $T_{obs} = \dfrac{\bar{X} - \bar{Y}}{s_{pooled}\sqrt{\frac{1}{n} + \frac{1}{m}}}$, calculated using the original data. Now, for the bootstrap, obtain B resamples as described above, and for each one

obtain the value of $T^* = \dfrac{\bar{X}^* - \bar{Y}^*}{s_{pooled}\sqrt{\frac{1}{n}+\frac{1}{m}}}$**, thus calculating** $T_1,\ldots,T_B$**. As before, the bootstrap** p**-value for the test is**

$$2\min(\mathrm{ASL}_{Boot}, 1-\mathrm{ASL}_{Boot}),$$

where

$$\mathrm{ASL}_{Boot} = \frac{\#(T_b^* - T_{obs})}{B}.$$

NOTES:

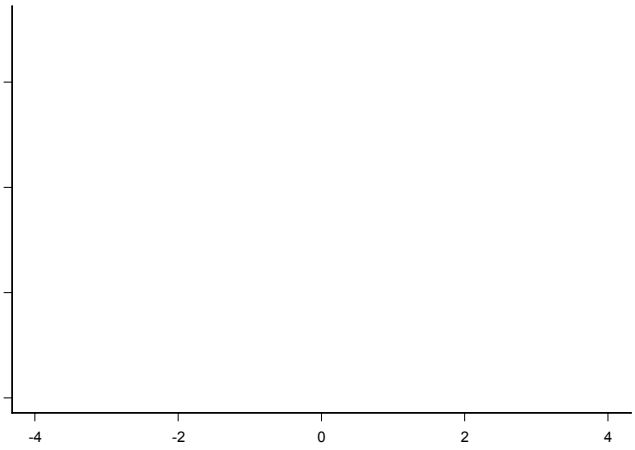
1. **Here, we have used a pooled variance estimator in the test statistic. This choice is not necessary in this case, as we don't require a** t**-distribution for the test statistic under the null. Nevertheless, a means comparison makes most sense when the variances are comparable, so we will make this assumption. If the variances are very different, should we be comparing means?**

2. **The "confidence interval" method of constructing a test would, in this case, yield a different procedure, as the resampling carried out under the null is not equivalent in this case to the resampling carried out for a percentile-**t **interval; see Hall and Martin (1988).**

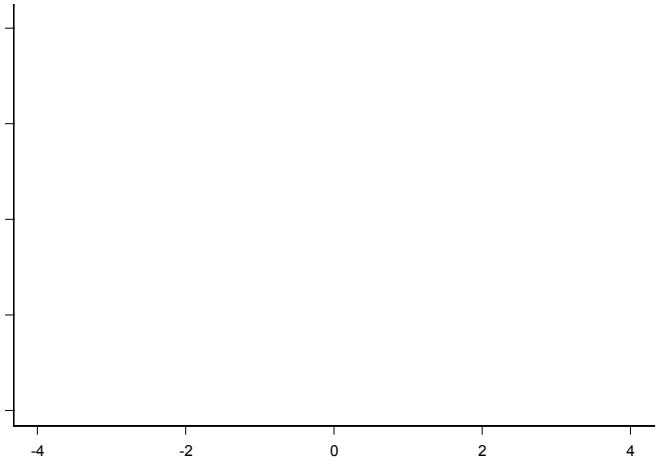
3. **Unlike the permutation test case, we were careful to use an approximately pivotal test statistic following our discussion of confidence intervals. Of course, we could have made a similar choice for permutation tests.**

```
boot2sampreps <- function(data1, data2, teststat, B){
# Outputs the bootstrap replications of the test statistic
# assuming resampling under the null hypothesis of equality
# of distributions for a two-sample bootstrap hypothesis
# test. Inputs include the two data sets, a
# function that calculates a test statistic, and the
# number of resamples, B. Output is a list, the first
# component of which, obs, is the observed test statistic,
# and the second of which is a vector (bootreps) containing
# the bootstrap replications.
    tobs <- teststat(data1, data2)
    alldata <- c(data1, data2)
    nulltest <- 0
    for(i in 1:B) {
        resample <- sample(alldata, replace = T)
        newdata1 <- resample[1:(length(data1))]
        newdata2 <- resample[(length(data1) +
                 1):length(alldata)]
        nulltest[i] <- teststat(newdata1, newdata2)
    }
    list(obs = tobs, bootreps = nulltest)
}

boot2samphyp <- function(data1, data2, teststat, B){
# Conducts a bootstrap hypothesis test of equality of
# distributions. Inputs include the two data sets, a
# function that calculates a test statistic, and the
# number of resamples, B. Outputs are the observed test
# statistic and one- and two-sided p-values.
    tobs <- teststat(data1, data2)
    alldata <- c(data1, data2)
    nulltest <- 0
    for(i in 1:B) {
        resample <- sample(alldata, replace = T)
        newdata1 <- resample[1:(length(data1))]
        newdata2 <- resample[(length(data1) +
                 1):length(alldata)]
        nulltest[i] <- teststat(newdata1, newdata2)
    }
    asl <- sum(nulltest <= tobs)/B
    pval1side <- min(asl, 1 - asl)
    pval2side <- 2 * pval1side
    cbind(tobs, pval1side, pval2side)
}
```

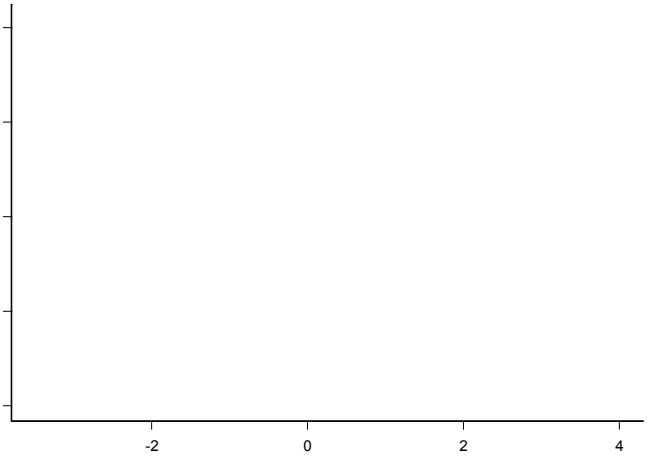
```
# First, some normal data, same means
x_rnorm(15,0,1)
y_rnorm(10,0,1)
breps <- boot2sampreps(x,y,tteststat,1000)$bootreps
hist(breps,xlab="Bootstrap replicates under Null",
    main="Histogram of Bootstrap replicates, Normal Mean
    difference")
abline(v=tteststat(x,y),lty=2)
# Fig 15
boot2samphyp(x,y,tteststat,1000)
          tobs pval1side pval2side
[1,] 0.1294882     0.458     0.916
```



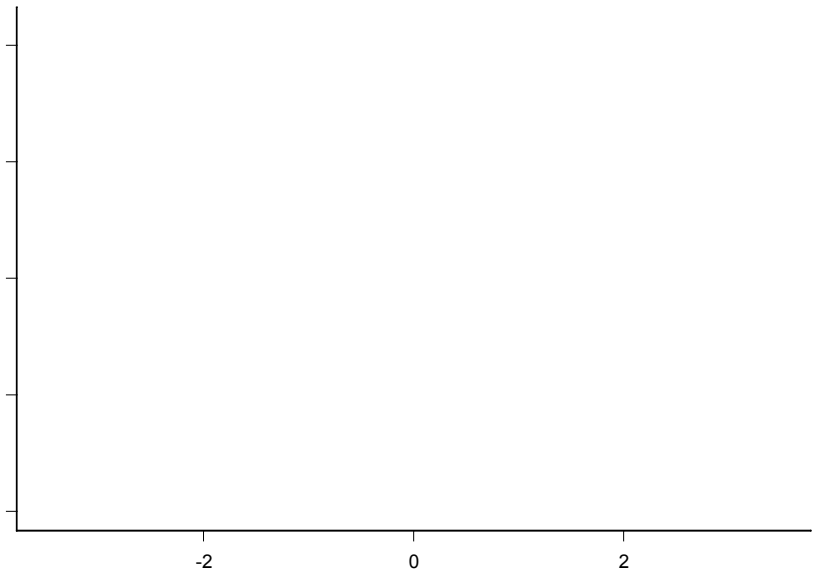

```
# Normal Data, different means
x_rnorm(15,1,1)
y_rnorm(10,0,1)
breps <- boot2sampreps(x,y,tteststat,1000)$bootreps
hist(breps,xlab="Bootstrap replicates under Null",
    main="Histogram of Bootstrap replicates, Normal Mean
    difference")
abline(v=tteststat(x,y),lty=2)
# Fig 16
boot2samphyp(x,y,tteststat,1000)
         tobs pval1side pval2side
[1,] 1.505929     0.069     0.138
```




```
# Different distributions, same means
x_rexp(20,2)
y_rnorm(15,0.5,1)
breps <- boot2sampreps(x,y,teststat,1000)$bootreps
hist(breps,xlab="Bootstrap replicates under Null",
    main="Histogram of Bootstrap replicates,
    normal/exponential equal means")
abline(v=teststat(x,y),lty=2)
# Fig 17
boot2samphyp(x,y,teststat,1000)
          tobs pval1side pval2side
[1,] -1.414212     0.096     0.192
```



```
# Different distributions, different means
x_rexp(20,2)
y_rnorm(15,0,1)
breps <- boot2sampreps(x,y,tteststat,1000)$bootreps
hist(breps,xlab="Bootstrap replicates under Null",
    main="Histogram of Bootstrap replicates,
    normal/exponential unequal means")
abline(v=tteststat(x,y),lty=2)
# Fig 18
boot2samphyp(x,y,tteststat,1000)
         tobs pval1side pval2side
[1,] 3.205276     0.001     0.002
```



BOOTSTRAP TEST FOR CORRELATION

Suppose we wish to test that the correlation between paired (X,Y) data is zero. A bootstrap procedure under the null hypothesis would resample under the assumption of no correlation: that is, that no pairing existed between the X and Y data. So, resampling under the null would resample each of the X and Y data *separately* and forming new "pairs" from the resamples. The bootstrap test would then compare the observed correlation from the original, paired data with the distribution of the bootstrap correlations under the assumption of no pairing.

```
bootcorrhyp <- function(data, B){
# Function to conduct a bootstrap test of no
# correlation between two variables
    nulltest <- 0
    n <- length(data[, 1])
    rhoobs <- correl(data)
    for(i in 1:B) {
        newx <- data[sample(1:n, replace = T), 1]
        newy <- data[sample(1:n, replace = T), 2]
        newdata <- cbind(newx, newy)
        nulltest[i] <- correl(newdata)
    }
    asl <- sum(nulltest <= rhoobs)/B
    pval1side <- min(asl, 1 - asl)
    pval2side <- 2 * pval1side
```

```
    cbind(rhoobs, pval1side, pval2side)
}

# The respiratory resistance data:

data_cysfib
bootcorrhyp(data,1000)
         rhoobs pval1side pval2side
[1,] -0.2713439       0.1       0.2
data_asthma
bootcorrhyp(data,1000)
         rhoobs pval1side pval2side
[1,] -0.3755076     0.006     0.012

# Some “uncorrelated” data
x_rnorm(10,0,1)
y_rnorm(10,0,1)
data_cbind(x,y)
bootcorrhyp(data,1000)
          rhoobs pval1side pval2side
[1,] -0.09396667     0.416     0.832

# Some correlated data (true correlation = 0.5)
x_rnorm(10,0,1)
y_rnorm(10,0,1)
z_rnorm(10,0,1)
xx_x+z
yy_y+z
data_cbind(xx,yy)
bootcorrhyp(data,1000)
        rhoobs pval1side pval2side
[1,] 0.3771476     0.147     0.294
```

BOOTSTRAP ANOVA F-TEST

Similar to the test for no correlation, a bootstrap ANOVA F-test can also be developed – more discussion of bootstrapping for regression models will be presented in a later section. For the moment, we will consider how resampling can be carried out under the null hypothesis.

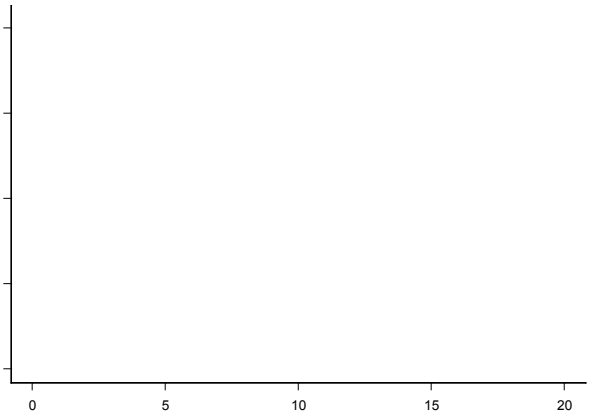
Under the null hypothesis of no regression, X and Y data are resampled separately and then re-joined to effectively break the relationship between X and Y for each resampled "data point" (since no such relationship exists under the null hypothesis). The observed F value (F_{obs}) can then be compared to the bootstrap distribution of the resampled F values. Specifically, for each of B resamples, a model is fit resulting in the bootstrap F statistics $F_b^*, b = 1, \ldots, B$. The bootstrap test of no regression then has estimated level

$$\text{ASL}_{Boot} = \frac{\#(F_b^* \geq F_{obs})}{B}.$$

```
bootregfhyp <- function(data, B, ythenx = T){
# Conducts a simple bootstrap ANOVA F-test for regression
# data. Function assumes data is in matrix form. If the
# response is in the first column ythenx is set as True
# (default). If the response is in the last column ythenx
# is set as False. Resampling is carried out under the null
# by resampling X's and Y's separately. Output is the
# observed F statistic and the bootstrap p-value
    data <- as.matrix(data)
    if(ythenx) {
       y <- data[, 1]
       x <- data[, -1]
    }
    else {
       m <- length(data[1,  ])
       y <- data[, m]
       x <- data[,  - m]
    }
    n <- length(y)
    nullf <- 0
    fobs <- summary(lm(y ~ x))$fstatistic[1]
    for(i in 1:B) {
        bootind1 <- sample(1:n, replace = T)
        bootind2 <- sample(1:n, replace = T)
        if(length(data[1,  ]) > 2) {
           newmod <- lm(y[bootind2] ~ x[bootind1,  ])
           }
           else {
           newmod <- lm(y[bootind2] ~ x[bootind1])
           }
        nullf[i] <- summary(newmod)$fstatistic[1]
        }
    asl <- sum(nullf >= fobs)/B
    pval <- asl
    cbind(fobs, pval)
}

bootreghypreps <- function(data, B, ythenx = T){
# Calculates bootstrap replicates of ANOVA F-statistic
# Function assumes data is in matrix form. If the response
# is in the first column ythenx is set as True (default).
# If the response is in the last column ythenx is set as
# False. Resampling is carried out under the null by
```

```
# resampling X's and Y's separately. Output is a list
# including the observed F-statistic (obs) and the
# bootstrap replications (dist)
   data <- as.matrix(data)
   if(ythenx) {
      y <- data[, 1]
      x <- data[, -1]
   }
   else {
      m <- length(data[1,  ])
      y <- data[, m]
      x <- data[,  - m]
   }
   n <- length(y)
   nullf <- 0
   fobs <- summary(lm(y ~ x))$fstatistic[1]
   for(i in 1:B) {
      bootind1 <- sample(1:n, replace = T)
      bootind2 <- sample(1:n, replace = T)
      if(length(data[1,  ]) > 2) {
         newmod <- lm(y[bootind2] ~ x[bootind1,  ])
      }
      else {
         newmod <- lm(y[bootind2] ~ x[bootind1])
         }
      nullf[i] <- summary(newmod)$fstatistic[1]
      }
   list(obs = fobs, dist = nullf)
}

# Delinquency data: DI is a delinquency index (0-50) and IQ
# is Intelligence Quotient, for 18 youths between 15 and 20
delinq
      DI  IQ
 1 26.20 110
 2 33.00  89
 3 17.50 102
 4 25.25  98
 5 20.30 110
 6 31.90  98
 7 21.10 122
 8 22.70 119
 9 10.70 120
10 22.10  92
```

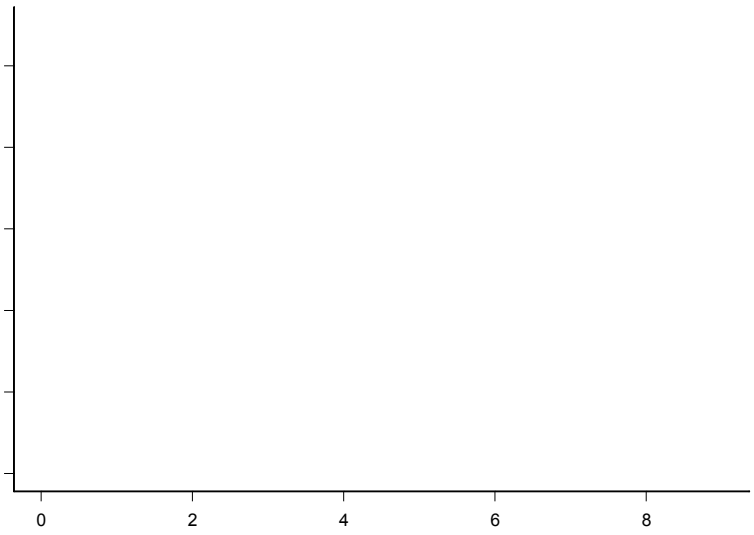
```
11 18.60 116
12 35.50  85
13 38.00  73
14 30.00  90
15 19.70 104
16 41.10  82
17 39.60 134
18 25.15 114

bootregfhyp(delinq,1000)
          fobs  pval
value 4.653871 0.045

breps <- bootreghypreps(delinq,1000)$dist
hist(breps,xlab="Bootstrap replications under Null",
    main="Histogram of bootstrap replications -
    regression")
abline(v=4.653871,lty=2)
# Fig 19
```




```
# A standard regression approach
Call: lm(formula = IQ ~ DI)
Coefficients:
                Value Std. Error   t value  Pr(>|t|)
(Intercept)  127.2799   11.6770    10.9001    0.0000
         DI   -0.9052    0.4196    -2.1573    0.0465

Residual standard error: 14.69 on 16 degrees of freedom
Multiple R-Squared: 0.2253
F-statistic: 4.654 on 1 and 16 degrees of freedom, the p-
value is 0.04653
```

```
# The anchovy data
anchovy
      Year Recruits      Eggs Parents  SST Tran Turb
 [1,] 1953   12.074  2.025403  18.361 18.8  192  179
 [2,] 1954   11.662  1.459475   6.410 16.6  236  229
 [3,] 1955   11.747  3.867642  17.064 17.3  256  251
 [4,] 1956   16.168  9.951466  24.950 17.8  261  259
 [5,] 1957   17.807  3.053031  40.369 20.0  277  291
 [6,] 1958   17.775  2.723855  15.875 18.9  239  221
 [7,] 1959   31.896  1.757811  10.710 18.4  233  212
 [8,] 1960   16.926  5.728070  12.087 17.6  220  199
 [9,] 1961   14.239  9.581805  32.445 17.7  219  195
[10,] 1962   14.331  1.914888  32.850 17.2  217  188
[11,] 1963   19.592  7.476276  26.766 17.9  412  185
[12,] 1964   31.580  4.551050  26.925 16.9  217  185
[13,] 1965   44.474 11.114772  54.107 19.0  214  180
[14,] 1966   40.631  2.425623  37.257 17.5  250  221
[15,] 1967   37.523 10.186960  24.487 16.8  248  218
[16,] 1968   42.983  5.156291  27.792 16.7  278  273
[17,] 1969   82.544  6.364590  41.028 18.6  245  225
[18,] 1970   51.551  9.111961  83.620 18.1  268  250
[19,] 1971    9.019  6.156392  41.331 18.2  233  206
[20,] 1972    5.304  2.364613  40.723 20.4  254  239
[21,] 1973    6.209  6.050940  22.424 18.1  258  243
[22,] 1974    9.925  0.404999  31.342 17.5  177  148
[23,] 1975   17.728  6.380809  25.105 17.3  250  252
[24,] 1976   10.921  2.402701  31.744 19.3   25  238
[25,] 1977   13.120  7.819894  15.776 18.5  216  190
[26,] 1978    9.305  3.338115   6.545 17.6  202  163
[27,] 1979    3.281  4.281967   8.066 18.0  207  178
[28,] 1980    5.423  4.599826   5.916 17.9  181  201
[29,] 1981    1.802  1.719678  18.791 17.5  136  138
[30,] 1982    0.105  4.558412  17.749 18.7   97   76
[31,] 1983    0.279  0.096052   0.224 21.9  222  203
[32,] 1984    5.632  0.514389   0.594 16.9  163  139
recruits_anchovy[,-1]

bootregfhyp(recruits,1000,ythenx=T)
          fobs pval
value 3.708863 0.02

breps <- bootreghypreps(recruits,1000,ythenx=T)$dist
hist(breps,xlab="Bootstrap replications under Null",
```

```
    main="Histogram of bootstrap replications -
    regression")
    abline(v=3.708863,lty=2)

# Fig 20
```




```
# And the standard analysis:
Call: lm(formula = Recruits ~ Eggs + Parents + SST + Tran +
Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept)  35.4883  46.1638     0.7687   0.4490
       Eggs   0.6082   1.0523     0.5780   0.5682
    Parents   0.4920   0.1880     2.6170   0.0146
        SST  -2.7431   2.4641    -1.1133   0.2758
       Tran   0.0343   0.0508     0.6757   0.5052
       Turb   0.0517   0.0722     0.7158   0.4805

Residual standard error: 14.99 on 26 degrees of freedom
Multiple R-Squared: 0.4163
F-statistic: 3.709 on 5 and 26 degrees of freedom, the p-
value is 0.01147
```

A BOOTSTRAP TEST FOR FIRST-ORDER AUTOCORRELATION

A more detailed look at resampling in a time series context will be developed later. For the moment, we consider a bootstrap hypothesis test for the presence of first-order autocorrelation.

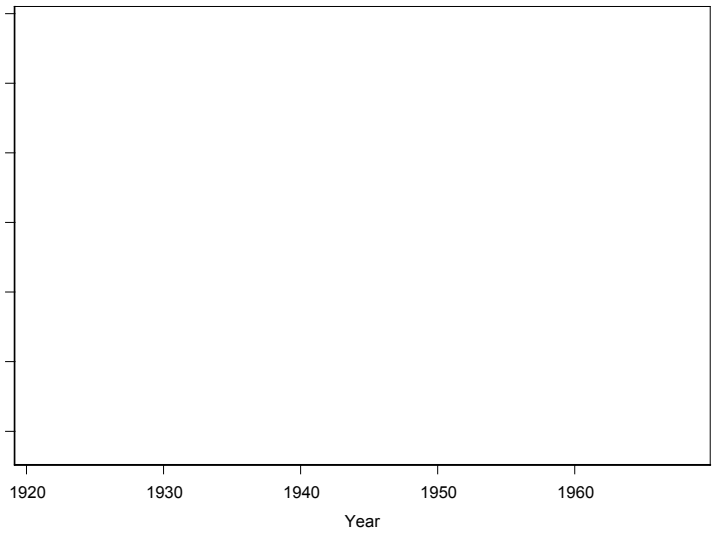
Under the null hypothesis of no first-order autocorrelation, resampling can be carried out assuming that the data are i.i.d.* – that is, standard uniform resampling. The resultant resamples can then each be used to approximate the sampling distribution of the first-order autocorrelation coefficient. The observed sample autocorrelation coefficient, $\text{acf}_{1,obs}$**, can then be compared to the bootstrap distribution constructed under the null hypothesis. If** $\text{acf}^*_{1,1},\ldots,\text{acf}^*_{1,B}$ **are the bootstrap replicates, the bootstrap** *p***-value is** $2\min(\text{ASL}_{Boot}, 1-\text{ASL}_{Boot})$**, where**

$$\text{ASL}_{Boot} = \frac{\#(\text{acf}^*_{1,b} \geq \text{acf}_{1,obs})}{B}.$$

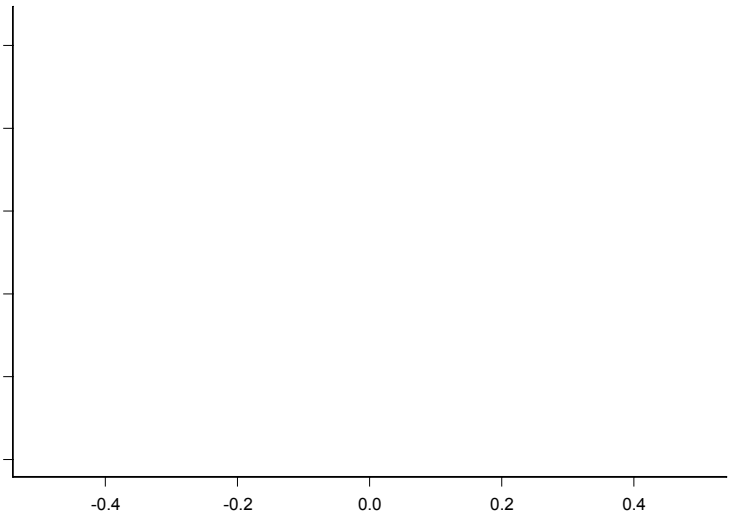
* **Note: Strictly speaking, the null hypothesis does not require i.i.d. structure, and so uniform resampling, although sufficient, is not necessarily "resampling under the null hypothesis". Such compromises are often required in constructing bootstrap hypothesis tests.**

```
boottimehyp <- function(data, B){
# Constructs a bootstrap test for first-order serial
# correlation for a time series. Bootstrap test statistic
# is the first autocorrelation coefficient. Input is the
# time series, and the number of resamples, B. Output
# includes the observed autocorrelation, and one- and
# two-sided bootstrap p-values. Resampling is carried
# out under the null by using i.i.d. uniform resampling
    obs <- acf(data, plot = F)$acf[2]
    n <- length(data)
    nullacf <- 0
    for(i in 1:B) {
        resample <- data[sample(1:n, replace = T)]
        nullacf[i] <- acf(resample, plot = F)$acf[2]
        }
    asl <- sum(nullacf <= obs)/B
    pval1side <- min(asl, 1 - asl)
    pval2side <- 2 * pval1side
    cbind(obs, pval1side, pval2side)
}

boottimereps <- function(data, B){
# Calculates bootstrap replications of the first-order
# serial correlation for a time series. Related to the
# bootstrap hypothesis test of no first-order serial
# correlation.
    obs <- acf(data, plot = F)$acf[2]
    n <- length(data)
    nullacf <- 0
    for(i in 1:B) {
        resample <- data[sample(1:n, replace = T)]
        nullacf[i] <- acf(resample, plot = F)$acf[2]
        }
    list(obs = obs, dist = nullacf)
}

# Bicoal.tons data. The data presents as non-stationary,
# but first differences (called bc) appear stationary.
# We can now test whether these first differences are
# plausibly uncorrelated.

tsplot(bc,xlab="Year",ylab="Differenced bicoal.tons",
    main="Time Plot of first-differenced bitumous coal
    data")
```

```
# Fig 21
```




```
bootacf <- boottimereps(bc,1000)
hist(bootacf$dist,xlab="Bootstrap replications of ACF(1)",
    main="Histogram of bootstrapped first-order
    autocorrelation")
abline(v=bootacf$obs,lty=2)
# Fig 22
```




```
boottimehyp(bc,1000)
             obs pval1side pval2side
[1,] -0.0682366     0.383     0.766
```

OTHER CASES

Clearly, bootstrap hypothesis testing cannot be made as "automatic" as other bootstrap procedures we have discussed. In particular, each individual case requires the user to think carefully about how *resampling under the null hypothesis* might be carried out (or even if it can be carried out!)

BOOTSTRAP POWER CALCULATIONS

The bootstrap can be used to estimate the significance level of tests by resampling under the null hypothesis. In a similar way, the bootstrap can also be used to estimate the power of tests by resampling under particular alternatives. For example, it would be easy in the case of a test for a mean to formulate a resampling scheme under a particular value of μ other than μ_0, and so estimate the power of a test under that alternative. Other cases would require significant thought as to how resampling *under the alternative* might work.

BOOTSTRAP FOR REGRESSION MODELS

Consider bivariate data $(X_i, Y_i),\ i = 1,\ldots,n$**, where** X **represents an explanatory variable and** Y **a response variable. Here, we will consider a simple linear model relating** Y **to** X**, although the technique is <u>much</u> more general in application. The model we will consider here is**

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i, \qquad i = 1,\ldots,n,$$

where X **can either be a fixed or random covariate and the errors** ε_i **are assumed to have zero location (usually mean) and constant variance, and are assumed independent. Here, we do *not* need to assume the errors are Normally distributed.**

The form of the model is unimportant here, although a simple model has been chosen to fix discussions. A more general approach might simply specify $Y_i = g(\beta, X_i) + \varepsilon_i$**, for some function** g **(possibly non-linear, possibly unknown).**

There are two major approaches to bootstrapping in regression models. Briefly, they are "resampling based on residuals" and "resampling based on data points".

BOOTSTRAPPING BASED ON RESIDUALS

Goal: Inference for regression parameters β_0, β_1.

Step 1: Fit the model $Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i,\ i = 1,\ldots,n$ using least-squares or another fitting method to obtain initial estimates $\hat{\beta}_0, \hat{\beta}_1$.

Step 2: Calculate the sample residuals from the model: $\hat{\varepsilon}_i = Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i,\ i = 1,\ldots,n$.

Step 3: Resample at random and with replacement from the sample residuals to obtain a resample $\hat{\varepsilon}_1^*, \hat{\varepsilon}_2^*, \ldots, \hat{\varepsilon}_n^*$. Using the resampled residuals, construct a "bootstrap data set" using the original X's:

.

Step 4: Re-fit the model to the bootstrap data, $\{(X_i, Y_i^*); i = 1,\ldots,n\}$ to obtain bootstrap versions of the regression coefficients $\hat{\beta}_0^*, \hat{\beta}_1^*$.

Step 5: Repeat Steps 3 and 4 a large number (B) times to obtain B "bootstrap coefficients"

$$(\hat{\beta}_0^*, \hat{\beta}_1^*)_1, (\hat{\beta}_0^*, \hat{\beta}_1^*)_2, \cdots, (\hat{\beta}_0^*, \hat{\beta}_1^*)_B.$$

RESAMPLING BASED ON DATA POINTS

Goal: Inference for regression parameters β_0, β_1.

Step 1: Resample at random and with replacement from the data pairs $(X_i, Y_i),\ i = 1,\ldots,n$ to obtain a resample data set $(X_i^*, Y_i^*),\ i = 1,\ldots,n$.

Step 2: Re-fit the model to the bootstrap data, $\{(X_i^*, Y_i^*); i = 1,\ldots,n\}$ to obtain bootstrap versions of the regression coefficients $\hat{\beta}_0^*, \hat{\beta}_1^*$.

Step 3: Repeat Steps 1 and 2 a large number (B) times to obtain B "bootstrap coefficients"

$$(\hat{\beta}_0^*, \hat{\beta}_1^*)_1, (\hat{\beta}_0^*, \hat{\beta}_1^*)_2, \cdots, (\hat{\beta}_0^*, \hat{\beta}_1^*)_B.$$

For each approach to resampling, the bootstrap replications

$$(\hat{\beta}_0^*, \hat{\beta}_1^*)_1, (\hat{\beta}_0^*, \hat{\beta}_1^*)_2, \cdots, (\hat{\beta}_0^*, \hat{\beta}_1^*)_B$$

are then used to construct inference (confidence intervals, hypothesis tests, etc.) for β_0, β_1.

WHAT IS THE DIFFERENCE?

Bootstrapping based on residuals assumes:

- **the sample residuals can be thought of as realisations of an i.i.d. process**
- **all (or most) of the random variation in the data is captured in the residual process**
- **the model is appropriate or correct (this approach is very sensitive to model misspecification)**
- **the X's are fixed, or conditioned on (since the original X's are used to construct the resampled Y's).**

Bootstrapping based on the data points assumes:

- **the data are i.i.d. realisations from a bivariate distribution having some distribution function F.**
- **more specifically, the X data is random (since the X's are sampled as well).**

WHICH METHOD TO USE?

The short answer is to use resampling based on residuals when X is fixed or when you really believe the model, and use the method based on data points when X is genuinely random or when model misspecification may be a real possibility (keeping in mind that serious model mis-specification will obviously affect either method)

INFERENCE FOR REGRESSION COEFFICIENTS

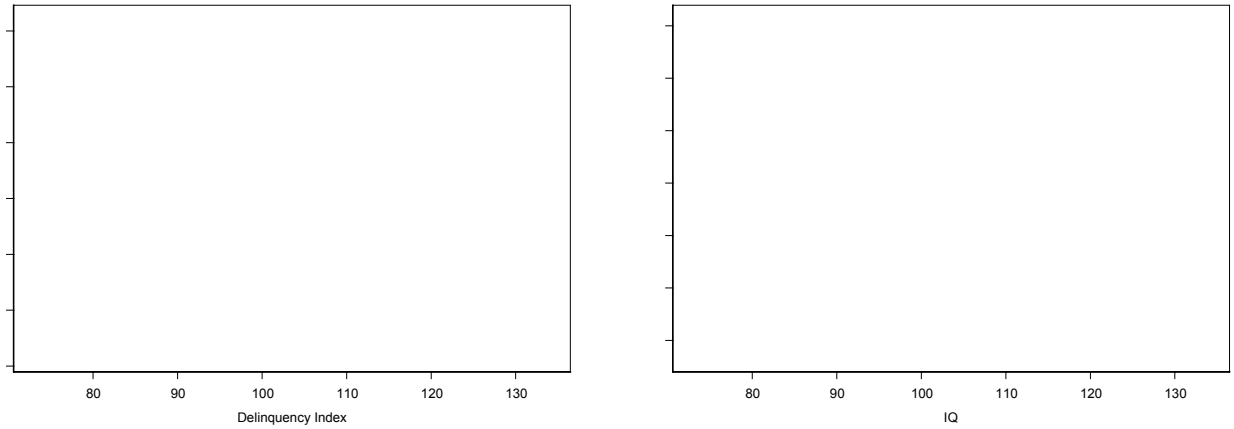
Once the bootstrap replicates of $\hat{\beta}_0^*, \hat{\beta}_1^*$ have been obtained, they can be used in the obvious way to estimate the sampling distributions of $\hat{\beta}_0$ and $\hat{\beta}_1$, to set the various types of bootstrap confidence intervals, and to test hypotheses. An important question is how one might resample under a particular null hypothesis. For example, if the null hypothesis specified $\beta_1 = 1$, then under residual-based resampling, resampled data points could be constructed using

$$Y_i^* = \hat{\beta}_0 + 1 \cdot X_i + \hat{\varepsilon}_i^*, \; i = 1, \ldots, n.$$

```
# Bootstrapping for regression

bootreg <- function(data, B, residual = T, ythenx = T){
# Calculates bootstrap regression coefficients and s.e.'s.
# Function assumes data is in matrix form. If the response
# is in the first column ythenx is set as True (default).
# If the response is in the last column ythenx is set as
# False. Resampling can either be via residuals (setting
# residual=T, default) or by data points (residual=F). B is
# number of resamples. The output is a list whose first
# component $bootcoef is a matrix whose i'th column
# contains B bootstrap replications of the (i-1)th
# coefficient (the first column is for the model
# intercept). The second component of the list is the
# corresponding standard errors
    data <- as.matrix(data)
    if(ythenx) {
      y <- data[, 1]
      x <- data[, -1]
     }
     else {
           m <- length(data[1,  ])
           y <- data[, m]
           x <- data[,  - m]
     }
     n <- length(y)
   bootcoef <- matrix(0, nrow=B, ncol=(length(data[1,])))
   bootse <- matrix(0, nrow=B, ncol=(length(data[1,])))
     if(residual) {
        model <- lsfit(x, y)
        for(i in 1:B) {
            bootres <- sample(model$residuals, replace = T)
            newy <- cbind(1, x) %*% model$coef + bootres
            newmod <- lsfit(x, newy)
            bootcoef[i,  ] <- newmod$coef
            bootse[i,  ] <- ls.diag(newmod)$std.err
            }
        }
        else {
            for(i in 1:B) {
            bootind <- sample(1:n, replace = T)
            if(length(data[1,  ]) > 2) {
            newmod <- lsfit(x[bootind,  ], y[bootind])
            }
```

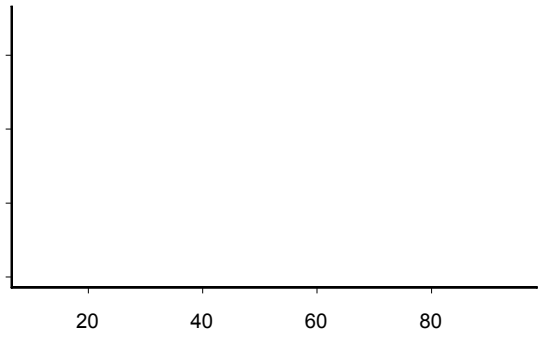
```
            else {
            newmod <- lsfit(x[bootind], y[bootind])
            }
            bootcoef[i,  ] <- newmod$coef
            bootse[i,  ] <- ls.diag(newmod)$std.err
            }
        }
    list(bootcoef = bootcoef, bootse = bootse)
}

percentilereg <- function(data, B, coefficient, residual=T,
ythenx=T, alpha){
# Takes the output list from a call to bootreg and produces
# a percentile interval for the beta coefficient required.
# e.g. if coefficient is set as 0 the interval is for the
# intercept and so on. The residual option dictates whether
# resampling is residual-based, and ythenx option dictates
# how the data is originally presented (y as first column
# versus y as last column)
   bootr <- bootreg(data, B, residual, ythenx)
   bootreps <- bootr$bootcoef[, (coefficient + 1)]
   sorted <- sort(bootreps)
   interval <- c(sorted[trunc((alpha * B)/2) + 1],
               sorted[trunc((1 - alpha/2) * B) + 1])
   interval
}

percentile.treg <- function(data,B,coefficient,residual=T,
    ythenx = T, alpha){
# Takes the output list from a call to bootreg and produces
# a percentile-t interval for the beta coefficient
# required. e.g. if coefficient is set as 0 the interval is
# for the intercept and so on. The residual option dictates
# whether resampling is residual based, and ythenx option
# dictates how the data is originally presented (y as first
# column versus y as last column)
    bootr <- bootreg(data, B, residual, ythenx)
    bootreps <- bootr$bootcoef[, (coefficient + 1)]
    bootses <- bootr$bootse[,(coefficient+1)]
    if(ythenx) {
        origmodel <- lsfit(data[,-1], data[,1])
    }
    else {
    origmodel <- lsfit(data[,-length(data[1,])],data[,
```

```
        length(data[1,])])
    }
    thetahat <- origmodel$coef[(coefficient+1)]
    sehat <- ls.diag(origmodel)$std.err[(coefficient+1)]
    t.star <- (bootreps - thetahat)/bootses
    sorted <- sort(t.star)
    interval <- c(thetahat-sehat*sorted[trunc((1-alpha/2)*
        B)+1],thetahat-sehat*sorted[trunc((alpha*B)/2)+1])
    interval
}

attach(delinq)
DI
    1  2    3     4    5    6    7    8    9   10   11
 26.2 33 17.5 25.25 20.3 31.9 21.1 22.7 10.7 22.1 18.6
   12 13 14   15   16   17    18
 35.5 38 30 19.7 41.1 39.6 25.15
IQ
   1  2   3  4   5  6   7   8   9 10  11 12 13 14  15 16
 110 89 102 98 110 98 122 119 120 92 116 85 73 90 104 82
  17  18
 134 114
```

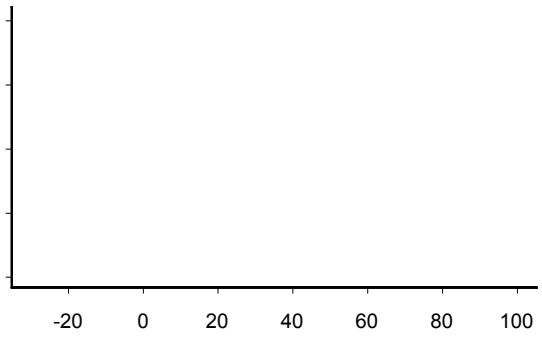



```
delinqboot_bootreg(delinq,1000,residual=T,ythenx=T)

names(delinqboot)
[1] "bootcoef" "bootse"
dim(delinqboot$bootcoef)
[1] 1000    2
dim(delinqboot$bootse)
[1] 1000    2
```

```
delinqls <- lsfit(IQ,DI)
par(mfrow=c(2,2), oma=c(0,0,3,0))
hist(delinqboot$bootcoef[,1],xlab="Replications of b0",
    main="Histogram of bootstrap intercepts")
abline(v=delinqls$coef[1],lty=2)
hist(delinqboot$bootcoef[,2],xlab="Replications of b1",
    main="Histogram of bootstrap slopes")
abline(v=delinqls$coef[2],lty=2)
hist(delinqboot$bootse[,1],xlab="Replications of se(b0)",
    main="Histogram of bootstrap intercept se")
abline(v=ls.diag(delinqls)$std.err[1],lty=2)
hist(delinqboot$bootse[,2],xlab="Replications of se(b1)",
    main="Histogram of bootstrap slope se")
abline(v=ls.diag(delinqls)$std.err[2],lty=2)
mtext("Residual-based bootstrap histograms – delinquency
    data", outer=T,cex=1.5,side=3)
```

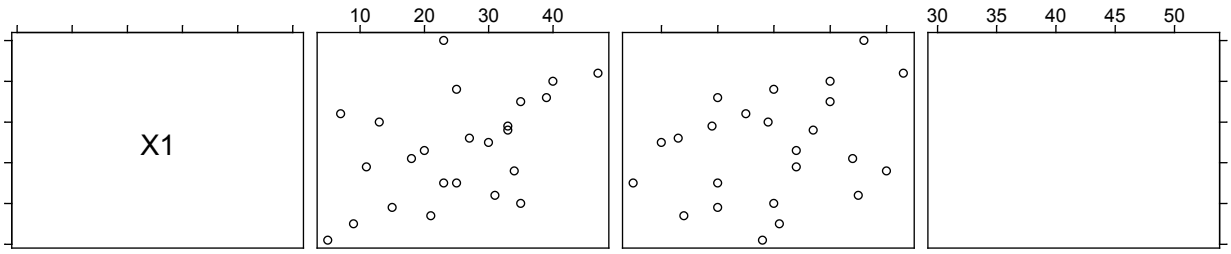


```
# Now, some confidence intervals for the coefficients
# standard interval for intercept
c(delinqls$coef[1]-qt(0.975,16)*
    ls.diag(delinqls)$std.err[1],delinqls$coef[1]+
    qt(0.975,16)*ls.diag(delinqls)$std.err[1])
[1]  26.73126  77.81462
# percentile interval
percentilereg(delinq,1000,0,residual=T,ythenx=T,0.05)
[1]  30.71032 76.56889
# percentile-t interval
percentile.treg(delinq,1000,0,residual=T,ythenx=T,0.05)
[1]  28.40967 78.86768

# standard interval for slope
c(delinqls$coef[2]-qt(0.975,16)*
    ls.diag(delinqls)$std.err[2],delinqls$coef[2]+
    qt(0.975,16)*ls.diag(delinqls)$std.err[2])
[1] -0.493548 -0.004313066
# percentile interval
percentilereg(delinq,1000,1,residual=T,ythenx=T,0.05)
[1] -0.46192186 -0.04662173
# percentile-t interval
percentile.treg(delinq,1000,1,residual=T,ythenx=T,0.05)
[1] -0.4874995 -0.004896509

# Is there much difference under point-based resampling?

delinqboot_bootreg(delinq,1000,residual=F,ythenx=T)

par(mfrow=c(2,2), oma=c(0,0,3,0))
hist(delinqboot$bootcoef[,1],xlab="Replications of b0",
    main="Histogram of bootstrap intercepts")
abline(v=delinqls$coef[1],lty=2)
hist(delinqboot$bootcoef[,2],xlab="Replications of b1",
    main="Histogram of bootstrap slopes")
abline(v=delinqls$coef[2],lty=2)
hist(delinqboot$bootse[,1],xlab="Replications of se(b0)",
    main="Histogram of bootstrap intercept se")
abline(v=ls.diag(delinqls)$std.err[1],lty=2)
hist(delinqboot$bootse[,2],xlab="Replications of se(b1)",
    main="Histogram of bootstrap slope se")
abline(v=ls.diag(delinqls)$std.err[2],lty=2)
mtext("Data-point-based bootstrap histograms – delinquency
    data", outer=T,cex=1.5,side=3)
```



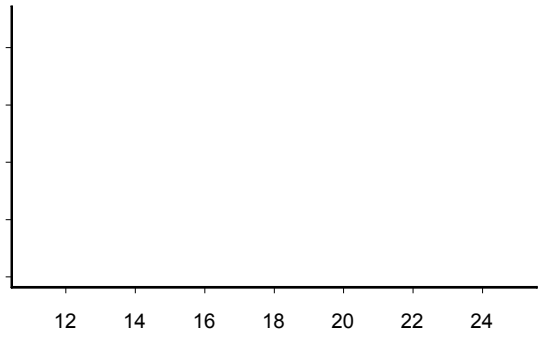

```
# Now, some confidence intervals for the coefficients
# standard interval for intercept
c(delinqls$coef[1]-qt(0.975,16)*
    ls.diag(delinqls)$std.err[1],delinqls$coef[1]+
    qt(0.975,16)*ls.diag(delinqls)$std.err[1])
[1]  26.73126  77.81462
# percentile interval
percentilereg(delinq,1000,0,residual=F,ythenx=T,0.05)
[1]  18.89316 83.20220
# percentile-t interval
percentile.treg(delinq,1000,0,residual=F,ythenx=T,0.05)
[1]  2.281797 81.40421

# A little wider

# standard interval for slope
c(delinqls$coef[2]-qt(0.975,16)*
    ls.diag(delinqls)$std.err[2],delinqls$coef[2]+
    qt(0.975,16)*ls.diag(delinqls)$std.err[2])
```

```
[1] -0.493548 -0.004313066
# percentile interval
percentilereg(delinq,1000,1,residual=F,ythenx=T,0.05)
[1] -0.54108596  0.08123153
# percentile-t interval
percentile.treg(delinq,1000,1,residual=F,ythenx=T,0.05)
[1] -0.5595176  0.2349784
# Again, wider

# Another data set; more regression coefficients
# Mathematician's salaries
# X1=publication quality, X2=years experience
# X3=success in grants, Y=Salary (thousands)
math
    X1 X2  X3    Y
 1 3.5  9 6.1 33.2
 2 5.3 20 6.4 40.3
 3 5.1 18 7.4 38.7
 4 5.8 33 6.7 46.8
 5 4.2 31 7.5 41.4
 6 6.0 13 5.9 37.5
 7 6.8 25 6.0 39.0
 8 5.5 30 4.0 40.7
 9 3.1  5 5.8 30.1
10 7.2 47 8.3 52.9
11 4.5 25 5.0 38.2
12 4.9 11 6.4 31.8
13 8.0 23 7.6 43.3
14 6.5 35 7.0 44.1
15 6.6 39 5.0 42.8
16 3.7 21 4.4 33.6
17 6.2  7 5.5 34.2
18 7.0 40 7.0 48.0
19 4.0 35 6.0 38.0
20 4.5 23 3.5 35.9
21 5.9 33 4.9 40.4
22 5.6 27 4.3 36.8
23 4.8 34 8.0 45.2
24 3.9 15 5.0 35.1

attach(math)
mathlm <- lm(Y~X1+X2+X3)
```




```
summary(mathlm)

Call: lm(formula = Y ~ X1 + X2 + X3)
Coefficients:
              Value Std. Error t value Pr(>|t|)
(Intercept) 17.8469  2.0019     8.9151  0.0000
         X1  1.1031  0.3296     3.3471  0.0032
         X2  0.3215  0.0371     8.6643  0.0000
         X3  1.2889  0.2985     4.3184  0.0003

Residual standard error: 1.753 on 20 degrees of freedom
Multiple R-Squared: 0.9109
F-statistic: 68.12 on 3 and 20 degrees of freedom, p-value
is 1.124e-010

mathboot <- bootreg(math,1000,residual=T, ythenx=F)

par(mfrow=c(2,2), oma=c(0,0,3,0))
hist(mathboot$bootcoef[,1],xlab="Replications of b0",
```

```
    main="Histogram of bootstrap intercepts")
abline(v=mathlm$coef[1],lty=2)
hist(mathboot$bootcoef[,2],xlab="Replications of b1",
    main="Histogram of bootstrap X1 slopes")
abline(v=mathlm$coef[2],lty=2)
hist(mathboot$bootcoef[,3],xlab="Replications of b2",
    main="Histogram of bootstrap X2 slopes")
abline(v=mathlm$coef[3],lty=2)
hist(mathboot$bootcoef[,4],xlab="Replications of b2",
    main="Histogram of bootstrap X3 slopes")
abline(v=mathlm$coef[4],lty=2)
mtext("Residual-based bootstrap histograms – math data",
outer=T,cex=1.5,side=3)
```

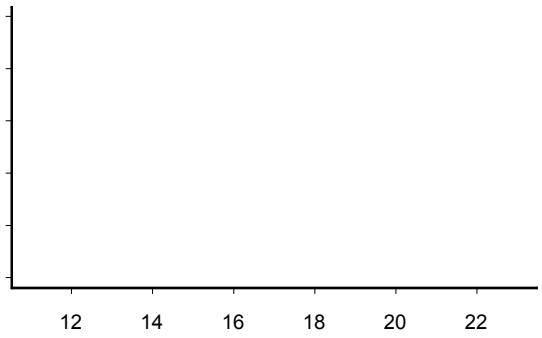



```
# Now, some confidence intervals for the coefficients
# standard interval for intercept
c(mathlm$coef[1]-qt(0.975,16)*
    summary(mathlm)$coefficients[1,2], mathlm$coef[1]+
    qt(0.975,16)* summary(mathlm)$coefficients[1,2])
```

[1] 13.60314 22.09072

```
# percentile interval
percentilereg(math,1000,0,residual=T,ythenx=F,0.05)
[1] 14.31942 21.22230
# percentile-t interval
percentile.treg(math,1000,0,residual=T,ythenx=F,0.05)
[1] 13.54452  21.80157

# standard interval for slope of X1
c(mathlm$coef[2]-qt(0.975,16)*
    summary(mathlm)$coefficients[2,2], mathlm$coef[2]+
    qt(0.975,16)* summary(mathlm)$coefficients[2,2])
[1] 0.4044662 1.801795
# percentile interval
percentilereg(math,1000,1,residual=T,ythenx=F,0.05)
[1] 0.526520 1.696543
# percentile-t interval
percentile.treg(math,1000,1,residual=T,ythenx=F,0.05)
[1] 0.4528368 1.801691

# standard interval for slope of X2
c(mathlm$coef[3]-qt(0.975,16)*
    summary(mathlm)$coefficients[3,2], mathlm$coef[3]+
    qt(0.975,16)* summary(mathlm)$coefficients[3,2])
[1] 0.2428529 0.4001865
# percentile interval
percentilereg(math,1000,2,residual=T,ythenx=F,0.05)
[1] 0.2600853 0.3885713
# percentile-t interval
percentile.treg(math,1000,2,residual=T,ythenx=F,0.05)
[1] 0.2440483 0.3968788

# standard interval for slope of X3
c(mathlm$coef[4]-qt(0.975,16)*
    summary(mathlm)$coefficients[4,2], mathlm$coef[4]+
    qt(0.975,16)* summary(mathlm)$coefficients[4,2])
[1] 0.6561934 1.921688
# percentile interval
percentilereg(math,1000,3,residual=T,ythenx=F,0.05)
[1] 0.7376538 1.8098408
# percentile-t interval
percentile.treg(math,1000,3,residual=T,ythenx=F,0.05)
[1] 0.6176107 1.888526
```

```
# Data-based resampling...

mathboot <- bootreg(math,1000,residual=F, ythenx=F)

par(mfrow=c(2,2), oma=c(0,0,3,0))
hist(mathboot$bootcoef[,1],xlab="Replications of b0",
    main="Histogram of bootstrap intercepts")
abline(v=mathlm$coef[1],lty=2)
hist(mathboot$bootcoef[,2],xlab="Replications of b1",
    main="Histogram of bootstrap X1 slopes")
abline(v=mathlm$coef[2],lty=2)
hist(mathboot$bootcoef[,3],xlab="Replications of b2",
    main="Histogram of bootstrap X2 slopes")
abline(v=mathlm$coef[3],lty=2)
hist(mathboot$bootcoef[,4],xlab="Replications of b2",
    main="Histogram of bootstrap X3 slopes")
abline(v=mathlm$coef[4],lty=2)
mtext("Data point-based bootstrap histograms – math data",
outer=T,cex=1.5,side=3)
```



```
# standard interval for intercept
c(mathlm$coef[1]-qt(0.975,16)*
    summary(mathlm)$coefficients[1,2], mathlm$coef[1]+
    qt(0.975,16)* summary(mathlm)$coefficients[1,2])
[1]  13.60314   22.09072
# percentile interval
percentilereg(math,1000,0,residual=F,ythenx=F,0.05)
[1] 14.49139 21.04696
# percentile-t interval
percentile.treg(math,1000,0,residual=F,ythenx=F,0.05)
[1] 13.62862  21.31928
# standard interval for slope of X1
c(mathlm$coef[2]-qt(0.975,16)*
    summary(mathlm)$coefficients[2,2], mathlm$coef[2]+
    qt(0.975,16)* summary(mathlm)$coefficients[2,2])
[1] 0.4044662 1.801795
# percentile interval
percentilereg(math,1000,1,residual=F,ythenx=F,0.05)
[1] 0.4275188 1.7742336
# percentile-t interval
percentile.treg(math,1000,1,residual=F,ythenx=F,0.05)
[1] 0.497675  1.829271
# standard interval for slope of X2
c(mathlm$coef[3]-qt(0.975,16)*
    summary(mathlm)$coefficients[3,2], mathlm$coef[3]+
    qt(0.975,16)* summary(mathlm)$coefficients[3,2])
[1] 0.2428529 0.4001865
# percentile interval
percentilereg(math,1000,2,residual=F,ythenx=F,0.05)
[1] 0.2452809 0.4028810
# percentile-t interval
percentile.treg(math,1000,2,residual=F,ythenx=F,0.05)
[1] 0.2402099 0.4039845
# standard interval for slope of X3
c(mathlm$coef[4]-qt(0.975,16)*
    summary(mathlm)$coefficients[4,2], mathlm$coef[4]+
    qt(0.975,16)* summary(mathlm)$coefficients[4,2])
[1] 0.6561934 1.921688
# percentile interval
percentilereg(math,1000,3,residual=F,ythenx=F,0.05)
[1] 0.7893637 1.8656090
# percentile-t interval
percentile.treg(math,1000,3,residual=F,ythenx=F,0.05)
[1] 0.6983314 1.806129
```

BOOTSTRAPPING DEPENDENT DATA

The bootstrap was designed for i.i.d. data – in fact, both parts of the i.i.d. assumption are embedded strongly in the rationale for how bootstrap resamples are defined.

Yet dependent data is a very common, important application, so several attempts have been made to adjust the bootstrap to cater for dependence among the data.

The main idea behind the most common approach to resampling for dependent data is that of attempting to identify independent "units" that can be resampled in more or less the usual way.

Two approaches are prevalent: resampling based on residuals, and "block"-based resampling where sequences, or blocks, of observations are sampled as complete units.

RESAMPLING BASED ON RESIDUALS

Here we assume a time-dependent sequence

$$\{X_t; t = 1, \ldots, N\}.$$

- **Similar approach to the regression setting**
- **Requires a model for the dependence, e.g.**

$$X_t = \phi_1 X_{t-1} + \ldots + \phi_p X_{t-p} + \varepsilon_t.$$

This model may be suggested by a standard Box-Jenkins identification based on the original data.

Step 1: Fit the model (in this case an $AR(p)$) to the original data, and form the residuals

$$\hat{\varepsilon}_t = X_t - \hat{\phi}_1 X_{t-1} - \cdots - \hat{\phi}_p X_{t-p},$$

for values of t for which values of X_{t-p} are available. Suppose the number of available sample residuals is T.

Step 2: Resample at random with replacement from among the T values of $\hat{\varepsilon}_t$ to obtain the resampled residuals $\hat{\varepsilon}_t^*$, from which a bootstrapped time series $\{X_t^*\}$ can be constructed (use $X_t^* = X_t$ for the p starting values of the series). Using this resampled data set, bootstrapped versions of the

model parameters $\hat{\theta}_1^*, \hat{\theta}_2^*, \ldots, \hat{\theta}_p^*$ can be estimated.

Step 3: Repeat Step 2 a large number (B) of times to obtain B bootstrap replications of the parameters: $\{\hat{\theta}_{i=1,\ldots,p}^*\}_1, \ldots, \{\hat{\theta}_{i=1,\ldots,p}^*\}_B$, from which inference for the true parameters can be conducted, as for our previous examples.

The key to this approach is the premise that the errors $\{\varepsilon_t\}$ can be assumed i.i.d., and can therefore be bootstrapped effectively.

The primary disadvantage of this approach is that it is clearly *very* dependent (no pun!) on the model for dependence being right. If the model is wrong, then the residuals may still contain considerable structure and dependence, in which case the bootstrap replications of the series are unlikely to behave at all like the original series.

Another approach that is not model dependent is the "block bootstrap", which although it assumes a particular "range" for dependence, is not tied to a particular model for that dependence.

THE BLOCK BOOTSTRAP

The premise of the block bootstrap is that while points within a short lag of one another in a time series may experience significant autocorrelation, such effects are often short-range, and that blocks of observations separated by larger lags may be effectively uncorrelated.

The original series is divided into k equal-length "blocks" of size ℓ such that $N = k\ell$, where N is the length of the original series. That is, the series is represented as the sequence of blocks $A_1, \ldots, A_k$:

$$\underbrace{X_1, \ldots, X_\ell}_{A_1}, \underbrace{X_{\ell+1}, \ldots, X_{2\ell}}_{A_2}, \ldots, \underbrace{X_{(k-1)\ell}, \ldots, X_N}_{A_k}$$

These blocks are then resampled at random with replacement, then the resampled blocks are reassembled into a resampled series:

$$P(A_j^* = A_i) = \frac{1}{k}, \quad i, j = 1, \ldots, k,$$

leading to a resampled series, for example:

$$\underbrace{A_1^*}_{X_{7\ell+1}, \ldots, X_{8\ell}}, \underbrace{A_2^*}_{X_{3\ell+1}, \ldots, X_{4\ell}}, \cdots, \underbrace{A_{k-1}^*}_{X_{7\ell+1}, \ldots, X_{8\ell}}, \underbrace{A_k^*}_{X_1, \ldots, X_\ell}$$

This process is then repeated B times to obtain B resampled series. These resampled series become the basis for the exploration of the sampling distributions of parameters from models fit to the series, in a similar way to that discussed earlier for regression and other time series models.

CHOICE OF BLOCK LENGTH, ℓ

This topic is a subject of ongoing research, so only indicative advice is given here. Basically, the choice of ℓ is subject to a judgement as to the extent of the dependence present in the original series. For short-range dependence, as in an $AR(p)$ model for small p, a small value of ℓ, say 3 to 5, can work quite well. For longer-range dependence, larger values of ℓ are indicated. The extent of dependence can be effectively measured by traditional means such as the sample autocorrelation function.

REFINEMENTS

- **Random start points for the blocks (cycling)**
- **Random block length**
- **Stationary block bootstrap**

WHICH METHOD TO USE?

- **Block bootstrapping is not model-dependent, and therefore offers greater flexibility than residual-based resampling**

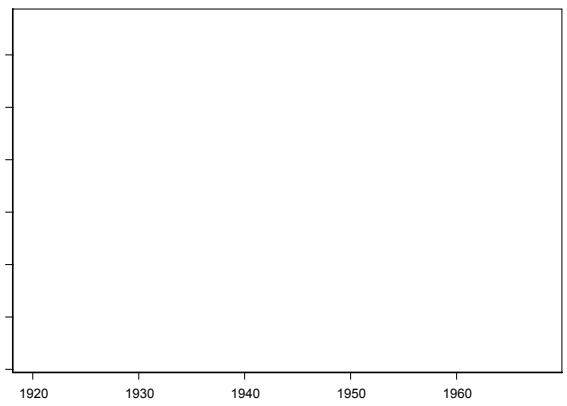
- **Block bootstrapping requires the choice of a block-length, which can be tricky in practice. If the length chosen is not appropriate, then the resamples will typically not carry the "right" dependence structure**

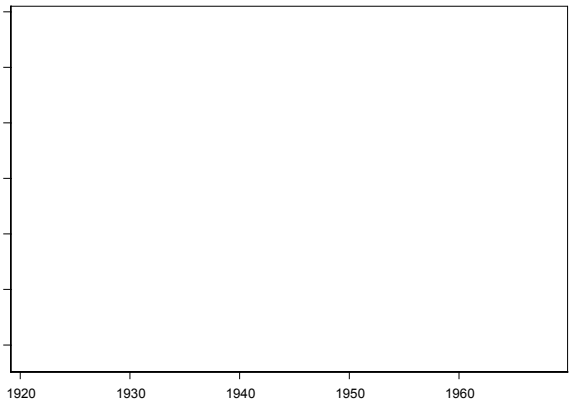
- **Residual-based resampling is simple in concept, provided the model is credible**

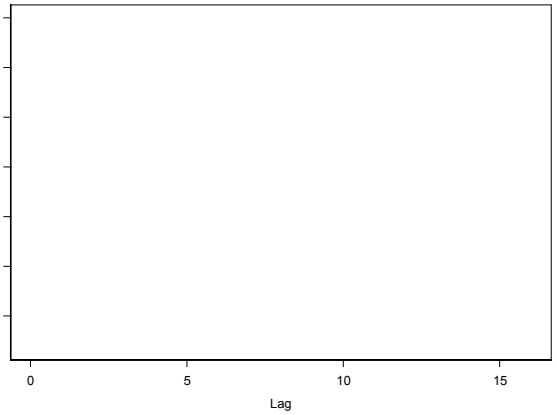
- **The underlying idea behind each of the methods is that once an independent element, whether it be residuals or blocks of data points, is identified, that is the target of bootstrap resampling**

```
timeboot <- function(data, B, residual = T, ...){
# Bootstrap replications of time series data. Input data is
# the original series. If residual=T, resampling is based
# on residuals of an autoregressive model of order p, where
# p is input as an optional parameter. If residual=F, block
# bootstrapping is carried out and the block length l is
# required as an optional parameter. Output is in the form
# of a matrix whose rows are the resampled time series.
   data <- ts(data)
   bootreps <- matrix(0, nrow = B, ncol = length(data))
   if(residual) {
      newdata <- data
      p <- c(...)
      model <- ar(data, aic = F, order.max = p)
      for(i in 1:B) {
    bootres <- c(rep(0, p),sample(model$resid[!is.na(model$
                     resid)], replace = T))
         for(j in (p + 1):length(data)) {
         newdata[j] <- t(newdata[seq(j-1,j-p,-1)])%*%
                           model$ar + bootres[j]
         }
         bootreps[i,  ] <- newdata
         }
      }
      else {
      blocklength <- c(...)
      if((trunc(length(data)/blocklength) == (length(data)/
               blocklength))) {
      blocks <- matrix(0,ncol=blocklength, nrow = (length(
               data)/blocklength))
      for(i in 1:(length(data)/blocklength)) {
          blocks[i,] <- data[seq(((i-1)*blocklength+1),
                                 i*blocklength, 1)]
          }
          for(i in 1:B) {
       bootind <- sample(1:trunc(length(data)/blocklength),
                            replace = T)
              bootreps[i,] <- c(t(blocks[bootind,  ]))
          }
      }
      else {
bootreps <- 1
print("Length of data not a multiple of block length")
      }
```

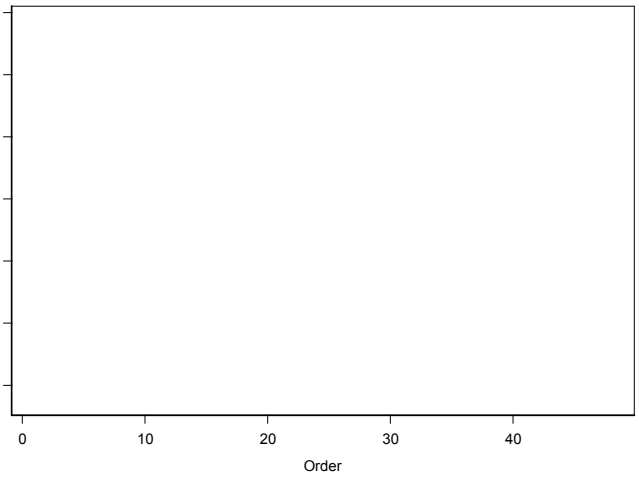
```
  }
  bootreps
}

percentile.time <- function(bootreps, alpha, order,
whichcoef){
# Calculates a percentile bootstrap confidence interval
# for an autoregressive coefficient given the bootstrap
# replications of a time series, output by the timeboot
# routine. Inputs include the bootstrap replications in a
# matrix (bootreps) whose rows represent the bootstrap
# replications, the level alpha (confidence=1-alpha),
# the order of the required autoregression, and the index
# of the required coefficient
   coef <- 0
   B <- length(bootreps[, 1])
   for(i in 1:B) {
       model <- ar(bootreps[i,], aic=F, order.max=order)
       coef[i] <- model$ar[whichcoef]
   }
   sorted <- sort(coef)
   interval <- c(sorted[trunc((alpha * B)/2) + 1],
                 sorted[trunc((1 - alpha/2) * B) + 1])
   interval
}

percentile.treps_function(bootreps,order,whichcoef){
# Calculates the bootstrap replicates of an autoregressive
# coefficient using resamples constructed by the timeboot
# routine
    coef <- 0
    B <- length(bootreps[, 1])
    for(i in 1:B) {
        model <- ar(bootreps[i,  ], aic = F,
            order.max = order)
        coef[i] <- model$ar[whichcoef]
    }
  coef
  }

# Bicoal.tons, bitumous coal production over 49 years
# The original series presents as non-stationary, so
# we will work on the first-differenced scale which appears
# to behave better
```



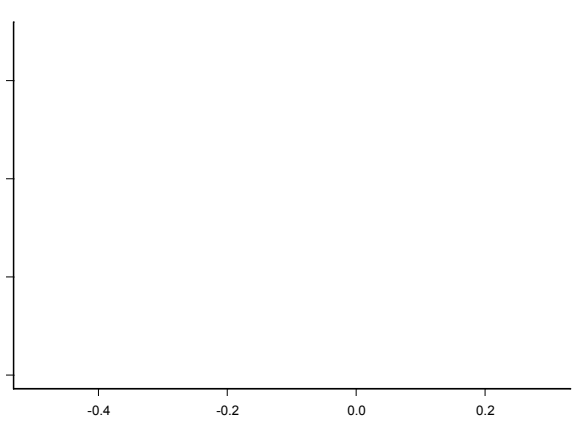


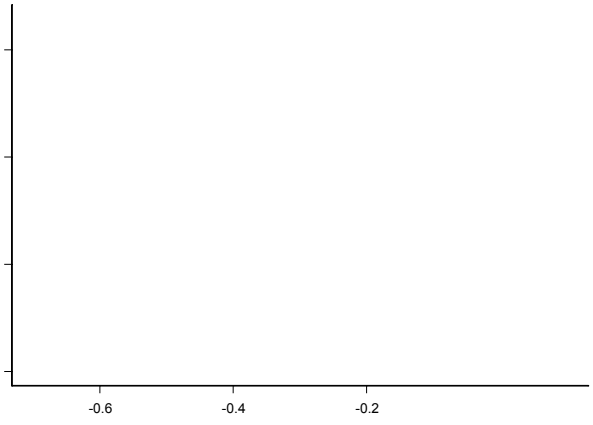



```
bcrepsres <- timeboot(bc,1000,residual=T,2)
bcreps <- bcrepsres
plot(1:48,bc,type='l',xlab="Order",
     ylab="Bootstrap series",
     main="Ten Residual-based bootstrap replications")
matlines(1:48,t(bcreps[1:10,]),type="l",col=2:11)
```

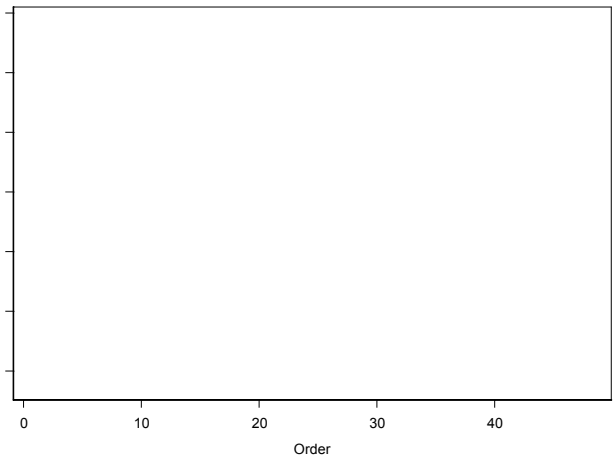


```
ar(bc,aic=F,order.max=2)$ar
             [,1]
[1,] -0.09076699
[2,] -0.33018032

bcpercentileres1_percentile.time(bcreps,0.05,2,1)
bcpercentileres1
[1] -0.3264169  0.1433261

bcpercentileres2 <- percentile.time(bcreps,0.05,2,2)
bcpercentileres2
[1] -0.54661512 -0.07591539

hist(percentile.treps(bcreps,2,1),
    xlab="Bootstrap replications",
    main="Histogram of Bootstrap first-order coefficient")
abline(v=ar(bc,aic=F,order.max=2)$ar[1],lty=2)
hist(percentile.treps(bcreps,2,2),
    xlab="Bootstrap replications",
    main="Histogram of Bootstrap second-order coefficient")
abline(v=ar(bc,aic=F,order.max=2)$ar[2],lty=2)
```

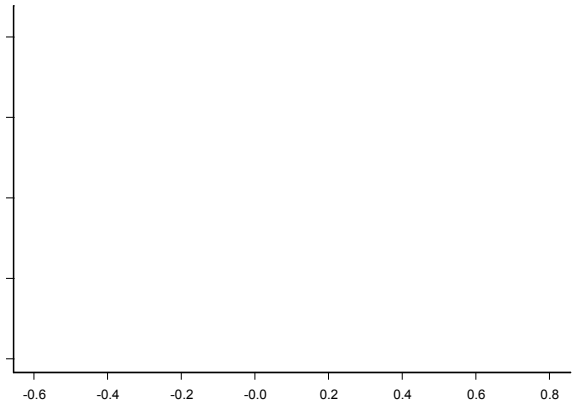


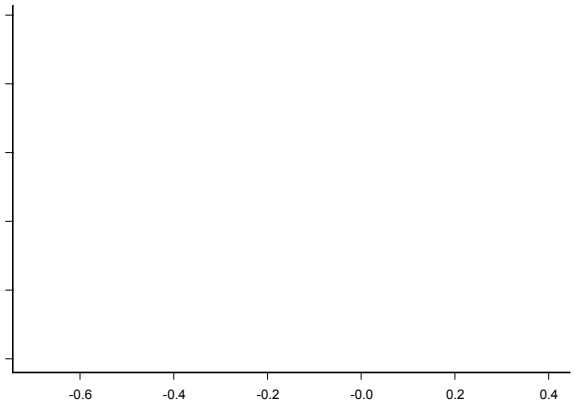



```
bcrepsblock6 <- timeboot(bc,1000,residual=F,6)
bcrepsblock3_timeboot(bc,1000,residual=F,3)
bcreps <- bcrepsblock6
plot(1:48,bc,type='l',xlab="Order",
    ylab="Bootstrap series",
    main="Ten block-based bootstrap replications, l=6")
matlines(1:48,t(bcreps[1:10,]),type="l",col=2:11)
```

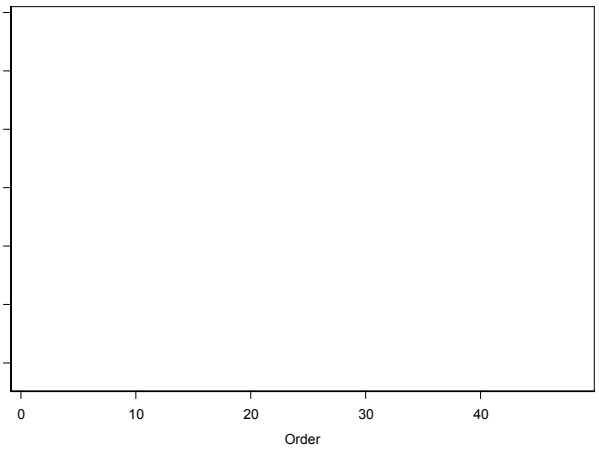



```
bcpercentileblock1_percentile.time(bcreps,0.05,2,1)
bcpercentileblock1
[1] -0.3537284  0.3476451

bcpercentileblock2 <- percentile.time(bcreps,0.05,2,2)
bcpercentileblock2
[1] -0.4623095  0.1912232

hist(percentile.treps(bcreps,2,1),
     xlab="Bootstrap replications",
     main="Histogram of Bootstrap first-order coefficient")
abline(v=ar(bc,aic=F,order.max=2)$ar[1],lty=2)
hist(percentile.treps(bcreps,2,2),
     xlab="Bootstrap replications",
     main="Histogram of Bootstrap second-order coefficient")
abline(v=ar(bc,aic=F,order.max=2)$ar[2],lty=2)
```

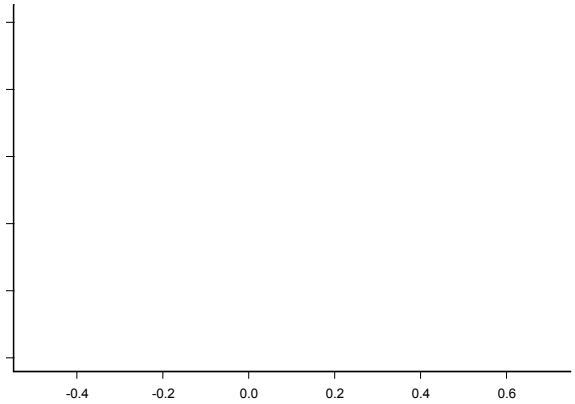


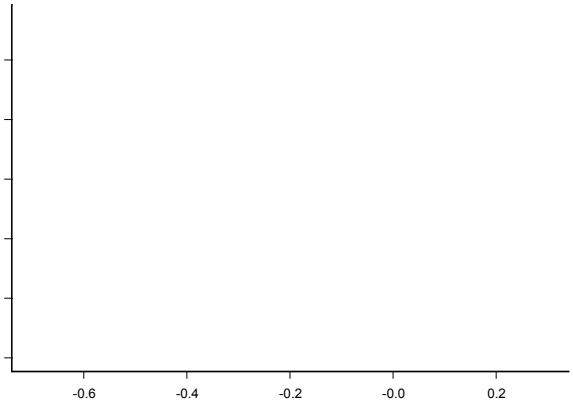


```
bcreps <- bcrepsblock3
plot(1:48,bc,type='l',xlab="Order",
    ylab="Bootstrap series",
    main="Ten block-based bootstrap replications, l=3")
matlines(1:48,t(bcreps[1:10,]),type="l",col=2:11)
```




```
bcpercentileblock31_percentile.time(bcreps,0.05,2,1)
bcpercentileblock31
[1] -0.2734901  0.3160574
bcpercentileblock32 <- percentile.time(bcreps,0.05,2,2)
bcpercentileblock32
[1] -0.46643659  0.05333611
hist(percentile.treps(bcreps,2,1),
    xlab="Bootstrap replications",
    main="Histogram of Bootstrap first-order coefficient")
abline(v=ar(bc,aic=F,order.max=2)$ar[1],lty=2)
hist(percentile.treps(bcreps,2,2),
    xlab="Bootstrap replications",
    main="Histogram of Bootstrap second-order coefficient")
abline(v=ar(bc,aic=F,order.max=2)$ar[2],lty=2)
```





BOOTSTRAPPING THE BOOTSTRAP – THE ITERATED BOOTSTRAP

Premise: Use the bootstrap to estimate and correct for errors in bootstrap procedures

Applications: Bias correction of bootstrap bias-corrected estimators; improving coverage accuracy in bootstrap confidence intervals.

Benefits: Order of magnitude improvement in bootstrap accuracy. For example, in the case of bias reduction, a typical example:

Estimator	Bias
Original estimator	$O(n^{-1})$
Bootstrap bias-corrected	$O(n^{-2})$
Iterated bootstrap bias-corrected	$O(n^{-3})$

Disadvantages: LOTS of computational effort – if the initial cost is B resamples, iterated bootstrap costs $O(B^2)$. Also, there is the issue of shrinking effective sample size for second-stage resamples.

BOOTSTRAP BIAS CORRECTION

Recall the bootstrap bias-corrected estimator:

$$\hat{\theta} = \hat{\theta}_{Boot,corrected} = 2\hat{\theta} - \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}_b^*,$$

an approximation to $\hat{\Delta}(F,\hat{F}) = 2\hat{\theta} - E_{\hat{F}}(\hat{\theta}^*)$**. In order to bias-correct** $\hat{\theta}$ **using the bootstrap, the bias of** $\hat{\theta}$ **is** $\hat{b}(\hat{\theta}) = E_F(\hat{\theta}) - \theta$**, which has bootstrap estimate** $\hat{b}_{Boot}(\hat{\theta}) = E_{\hat{F}}(\hat{\theta}^*) - \hat{\theta}$**. Since** $\hat{\theta}$ **itself involved first-level resamples from the data, then** $\hat{\theta}^*$ **must involve second-level resamples (i.e. resamples from the resamples drawn at the first stage). The bias-corrected estimate is then**

$$\begin{aligned}
\hat{\theta}_{Boot,corrected} &= \hat{\theta} - \hat{b}_{Boot}(\hat{\theta}) \\
&= \hat{\theta} + \hat{\theta} - E_{\hat{F}}(\hat{\theta}^*) \\
&= 2\hat{\theta} - E_{\hat{F}}(\hat{\theta}^*) + \hat{\theta} - \{2E_{\hat{F}}(\hat{\theta}^*) - E_{\hat{F},\hat{F}^*}(\hat{\theta}^{**})\} \\
&= 3\hat{\theta} - 3E_{\hat{F}}(\hat{\theta}^*) + E_{\hat{F},\hat{F}^*}(\hat{\theta}^{**})
\end{aligned}$$

The difficulty lies in approximating the second term $E_{\hat{F},\hat{F}^*}(\hat{\theta}^{**})$ **in the above expression. An algorithm for approximating** $E_{\hat{F},\hat{F}^*}(\hat{\theta}^{**})$ **using two nested levels of bootrapping is:**

Step 1: Draw a resample $X_1^*,\ldots,X_n^*$ from $X_1,\ldots,X_n$.

Step 2: From this resample, draw B_2 "re-resamples" according to the rule

$$P(X_j^{**} = X_i^*) = \tfrac{1}{n}, \quad i,j = 1,\ldots,n,$$

and compute the value $\frac{1}{B_2}\sum_{b=1}^{B_2}\hat{\theta}_b^{**}$ where $\hat{\theta}_b^{**}$ is the value of $\hat{\theta}$ computed using the b'th second-level resample ($b = 1,\ldots,B_2$).

Step 3: Now repeat Steps 1 and 2 a large number (B_1) of times, and compute the average

$$\frac{1}{B_1}\sum_{c=1}^{B_1}\frac{1}{B_2}\sum_{b=1}^{B_2}\hat{\theta}_{cb}^{**} = \frac{1}{B_1 B_2}\sum_{c=1}^{B_1}\sum_{b=1}^{B_2}\hat{\theta}_{cb}^{**},$$

where $\hat{\theta}_{cb}^{**}$ denotes the value of $\hat{\theta}$ computed using the b'th second-level resample arising from the c'th first-level resample from the original data ($c = 1,\ldots,B_1; b = 1,\ldots,B_2$). This average is the bootstrap approximation to $E_{\hat{F},\hat{F}^*}(\hat{\theta}^{**})$.

The iterated bootstrap bias-corrected estimate is

$$\hat{\theta}_{Boot,corrected} = 3\hat{\theta} - \frac{3}{B_1}\sum_{c=1}^{B_1}\hat{\theta}_c^* + \frac{1}{B_1 B_2}\sum_{c=1}^{B_1}\sum_{b=1}^{B_2}\hat{\theta}_{cb}^{**}.$$

This process can be repeated – with caution ☺!

```
doubleboot.biascor <- function(data, theta, B1, B2){
# Calculates the iterated bootstrap bias-corrected estimate
# of an unknown parameter theta. Input includes the data
# (vector or matrix), function theta, and inner and outer
# level numbers of resamples
     theta.star <- 0
     theta.starstar <- 0
     temp <- 0
     if(is.vector(data)) {
        for(i in 1:B1) {
            resample <- sample(data, replace = T)
            theta.star[i] <- theta(resample)
            for(j in 1:B2) {
                reresample <- sample(resample, replace = T)
                theta.starstar[j] <- theta(reresample)
                }
            temp[i] <- mean(theta.starstar)
            }
        level2 <- mean(temp)
        level1 <- mean(theta.star)
      }
      else if(is.matrix(data)) {
        for(i in 1:B1) {
   resample <- data[sample(1:length(data[,1]),replace=T),]
            theta.star[i] <- theta(resample)
            for(j in 1:B2) {
reresample<-resample[sample(1:length(data[,1]),replace=T),]
             theta.starstar[j] <- theta(reresample)
            }
            temp[i] <- mean(theta.starstar)
         }
         level2 <- mean(temp)
         level1 <- mean(theta.star)
      }
 else print("Invalid Data Structure", quote = F)
 dbbiascor <- 3 * mean(theta(data)) - 3 * level1 + level2
 dbbiascor
}
```

```
# Now for some examples.

x_rexp(20,2) # True values, mean=0.5, var=0.25
mean(x)
[1] 0.3262987
var(x)
[1] 0.1096148

# The mean is unbiased, so the bootstrap bias should
# be close to zero

boot.bias(x,mean,1000)
[1] -0.003328476

# wrongvar is the wrong variance estimator (using n
# not n-1). It is biased.
wrongvar(x)
[1] 0.1041341
boot.bias(x,wrongvar,1000)
[1] -0.004634924
wrongvar(x)-boot.bias(x,wrongvar,1000)
[1] 0.1066342
doubleboot.biascor(x,wrongvar,1000,1000)
[1] 0.1103325

# Is it close to the right answer (given by var)
var(x)
[1] 0.1096148

# We can try the same thing on normal data
x_rnorm(20,0,1)
mean(x)
[1] 0.243449
var(x)
[1] 0.6978224
# The mean is unbiased, so the bootstrap bias should
# be close to zero
boot.bias(x,mean,1000)
[1] -0.003378793
mean(x)-boot.bias(x,mean,1000)
[1] 0.2413985
doubleboot.biascor(x,mean,1000,1000)
[1] 0.2328735
```

```
# wrongvar is the wrong variance estimator (using n
# not n-1). It is biased.
wrongvar(x)
[1] 0.6629312
# Following is a bias-corrected version of wrongvar
wrongvar(x)-boot.bias(x,wrongvar,1000)
[1] 0.7014425
doubleboot.biascor(x,wrongvar,1000,1000)
[1] 0.7185529
# Is it close to the right answer (given by var)
var(x)
[1] 0.6978224

# We can also try to estimate bias for
# other statistics, eg the correlation coefficient.

data_cysfib
correl(data)
[1] -0.2713439
boot.bias(data,correl,1000)
[1] 0.006068503
correl(data)-boot.bias(data,correl,1000)
[1] -0.2694937
doubleboot.biascor(data,correl,1000,1000)
[1] -0.2754744
```

BOOTSTRAP ITERATION FOR CONFIDENCE INTERVALS

Let $I_1(\theta; X, X^*)$ be a nominal $1-\alpha$-level confidence interval for the unknown parameter θ. The interval I_1 could be one of the bootstrap intervals discussed earlier or a normal theory interval. The notation reflects the possible dependence of the interval on sample and resample information. Denote the coverage probability of $I_1(\theta; X, X^*)$ by

$$C(\theta) = P(\theta \in I_1(\theta; X, X^*)).$$

The true coverage $C(\theta)$ is often not exactly equal to the nominal coverage $1-\alpha$, so we could try to calibrate the confidence interval by finding the nominal coverage level, say β_α, so that the true coverage of the resultant interval is exactly $1-\alpha$. That is, if we let β_α be the solution of the equation

$$C(\beta_\alpha) = \alpha,$$

then the interval $I_1(\beta_\alpha; X, X^*)$ has coverage exactly $1-\alpha$. It is rarely possible to find β_α exactly, but it is possible to estimate it using the bootstrap. Let $I_1(\theta; X^*, X^{**})$ denote the version of

$I_1(\theta; X, X^*)$ **computed using the resample** $X_1^*, \ldots, X_n^*$ **instead of the sample** $X_1, \ldots, X_n$**. The bootstrap estimate of** $\gamma(\theta)$ **is**

$$\hat{\gamma}(\theta) = P_{\hat{F}}(\hat{\theta} \in I_1(\theta; X^*, X^{**})).$$

The bootstrap estimate, $\hat{\theta}_\pi$**, of** θ_π **is obtained as the solution of the equation**

$$\hat{\gamma}(\hat{\theta}_\pi) = \pi.$$

The bootstrap coverage-corrected interval is $I_2 = I_1(\hat{\theta}_\pi; X, X^*)$**. In practice, it is necessary to approximate** $\hat{\gamma}(\theta)$ **using a Monte Carlo approach. We describe such an algorithm for the construction of a coverage-corrected percentile-method interval.**

Step 1: Draw a resample $X_1^*, \ldots, X_n^*$ **from** $X_1, \ldots, X_n$**.**

Step 2: Draw B_2 **inner-level "re-resamples"** $X_1^{**}, \ldots, X_n^{**}$ **from** $X_1^*, \ldots, X_n^*$**, and form percentile intervals at several nominal levels (say** $\alpha_1, \alpha_2, \alpha_3$**) near** π **based on the** B_2 **resulting** $\hat{\gamma}^{**}$ **values.**

Step 3: Repeat Steps 1 and 2 B_1 **times, and compute the values** $\hat{\gamma}(\alpha_i), i = 1, 2, 3$ **as**

$$\hat{\gamma}(\alpha_i) = \frac{\#(\hat{\theta} \in I_b(\alpha_i; X^*, X^{**}); b = 1, \ldots, B_1)}{B_1}, \quad i = 1, 2, 3,$$

the proportion of times among the B_1 resamples for which the percentile interval constructed using the second-level resamples contained $\hat{\theta}$.

Step 4: Now, interpolate between the values of $(\alpha_1, G(\alpha_1)), (\alpha_2, G(\alpha_2)), (\alpha_3, G(\alpha_3))$ to find the value of $\hat{\alpha}_\pi$ for which $G(\hat{\alpha}_\pi) = \pi$.

Step 5: Finally, construct the percentile method interval of nominal level $\hat{\alpha}_\pi$ based on the original (outer-level) resamples.

WHY ITERATED BOOTSTRAP CONFIDENCE INTERVALS?

- **Better Coverage.**
  Typically, if the coverage error of an interval $I_1(\alpha; X, X^*)$ is of order $O(n^{-1})$, then the corresponding iterated bootstrap interval I_2 will have coverage error only $O(n^{-2})$.

- **Second-order correct endpoints.**
  Like BC_a and percentile-*t* intervals, iterated bootstrap intervals have second-order correct endpoints.

PROS AND CONS OF ITERATED BOOTSTRAP INTERVALS

- ✔ **Good coverage accuracy**
- ✔ **Correct "shape" – second-order correct interval endpoints**
- ✔ **Transformation-respecting (when the original interval is)**
- ✔ **Range-respecting (when the original interval is)**
- ✔ **Conceptually simple: no standard errors to estimate; no bias or acceleration constants**
- ✔ **Can perform *very* well under the parametric bootstrap**

- ✘ **<u>Much</u> more computation required than for ordinary bootstrap intervals**
- ✘ **Not good for very small samples (neither is percentile-*t* or BC_a), particularly as the support of "re-resamples" shrinks**

The iterated bootstrap can even be used to create "standard intervals" with good coverage accuracy. If I_1 is a standard, Normal theory interval, then the iterated bootstrap version I_2 essentially uses a single level of bootstraps to find the standard interval with smallest estimated coverage accuracy.

Despite this apparent good fortune, the iterated percentile interval is still probably a better choice, despite the increased computational cost. Why? Because the iterated standard interval remains symmetric, and achieves its new-found coverage accuracy essentially by increasing its length appropriately without adjusting the "shape" of the interval.

```
doubleboot <- function(data, theta, alpha, b1, b2){
#
# Calculates an iterated bootstrap confidence interval for
# a parameter theta. Input data is either a vector or a
# matrix whose rows represent data points. Other inputs
# include the function theta for the estimator, confidence
# alpha (usually 0.9 or 0.95), and inner (b2) and outer
# (b1) loop numbers of resamples. Output are the
# confidence interval endpoints. This code uses a simple
# quadratic interpolant between three coverages around the
# required nominal coverage.
#
    if(is.vector(data)) {
       n <- length(data)
    }
    else {
       n <- length(data[, 1])
    }
    thetahat <- theta(data)
    alpha1 <- alpha - 0.15
    gamma11 <- (1 - alpha1)/2
    gamma12 <- (1 + alpha1)/2
    alpha2 <- alpha
    gamma21 <- (1 - alpha2)/2
    gamma22 <- (1 + alpha2)/2
    alpha3 <- 0.999
    gamma31 <- (1 - alpha3)/2
    gamma32 <- (1 + alpha3)/2
    probest <- rep(0, b1)
    thetastar <- rep(0, b1)
    if(is.vector(data)) {
       for(i in 1:b1) {
           resample <- data[sample(1:n, replace = T)]
           thetastar[i] <- theta(resample)
           thetastarstar <- rep(0, b2)
           for(j in 1:b2) {
         reresample <- resample[sample(1:n, replace = T)]
              thetastarstar[j] <- theta(reresample)
              }
           probest[i] <- sum(thetastarstar <= thetahat)/b2
           }
       }
     if(is.matrix(data)) {
        for(i in 1:b1) {
```

```
            resample <- data[sample(1:n, replace = T),  ]
            thetastar[i] <- theta(resample)
            thetastarstar <- rep(0, b2)
            for(j in 1:b2) {
     reresample <- resample[sample(1:n, replace = T),]
                thetastarstar[j] <- theta(reresample)
                }
            probest[i] <- sum(thetastarstar <= thetahat)/b2
            }
         }
pi1 <- sum((probest < gamma12) & (probest > gamma11))/b1
pi2 <- sum((probest < gamma22) & (probest > gamma21))/b1
pi3 <- sum((probest < gamma32) & (probest > gamma31))/b1
aaa <- pi1/((alpha1-alpha2)*(alpha1 - alpha3))pi2/((alpha1-
     alpha2)*(alpha2-alpha3))+pi3/((alpha1-alpha3)*(alpha2-
     alpha3))
bbb <- (-pi1*(alpha2+alpha3))/((alpha1-alpha3)*(alpha1-
     alpha2))+(pi2*(alpha1+alpha3))/((alpha1-alpha2)*(
     alpha2-alpha3))-(pi3*(alpha1+alpha2))/((alpha1-alpha3)
     *(alpha2 - alpha3))
ccc <- (pi1*alpha2*alpha3)/((alpha1-alpha3)*(alpha1-
     alpha2))-(pi2*alpha1*alpha3)/((alpha1-alpha2)*(alpha2-
     alpha3))+(pi3*alpha1*alpha2)/((alpha1-alpha3)*(alpha2
     -alpha3))
delta1 <- (-bbb+sqrt(bbb*bbb-4*aaa*(ccc-alpha)))/(2*aaa)
delta2 <- (-bbb-sqrt(bbb*bbb-4*aaa*(ccc-alpha)))/(2*aaa)
delta <- delta2
if(abs(delta1 - alpha) < abs(delta2 - alpha))
   delta <- delta1
   if(delta > 1)
      delta <- 1
sortedthetas <- sort(thetastar)
interval <- c(sortedthetas[trunc((1-delta)/2*b1)+1],
    sortedthetas[trunc((1+delta)/2*b1)])
interval
}
```

```
# Confidence interval for the mean, normal data
x <- rnorm(20,0,1)
standard_c(mean(x)-1.96*sqrt(var(x)/length(x)),
    mean(x)+1.96*sqrt(var(x)/length(x)))
standard
[1] -0.2783248  0.5550864
standard.t_c(mean(x)-qt(0.975,19)* sqrt(var(x)/
    length(x)),mean(x)+qt(0.975,19)*sqrt(var(x)/
    length(x)))
standard.t
[1] -0.3066064  0.5833680
percentile(x,mean,0.05,1000)
[1] -0.3107396  0.5181968
percentile2(x,mean,0.05,1000)
[1] -0.2458086  0.5464973
percentile.t(x,mean,semean,0.05,1000)
[1] -0.4390714  0.5250340
bcanon(x,1000,mean,alpha=c(0.025,0.975))$confpoint[,2]
[1] -0.3498580  0.4864922
abcnon(x,meanp,alpha=c(0.025,0.975))$limits[,2]
[1] -0.3361171  0.4905425
doubleboot(x,mean,0.95,1000,1000)
[1] -0.3951379  0.6052981


# Confidence interval for the mean, exponential data
x <- rexp(20,1)
standard_c(mean(x)-1.96*sqrt(var(x)/length(x)),
    mean(x)+1.96*sqrt(var(x)/length(x)))
standard
[1] 0.4039575 0.9231385
standard.t_c(mean(x)-qt(0.975,19)* sqrt(var(x)/
    length(x)),mean(x)+qt(0.975,19)*sqrt(var(x)/
    length(x)))
standard.t
[1] 0.3863392 0.9407568
percentile(x,mean,0.05,1000)
[1] 0.4261539 0.9097930
percentile2(x,mean,0.05,1000)
[1] 0.4040791 0.9027109
percentile.t(x,mean,semean,0.05,1000)
[1] 0.4093287 0.9761450
bcanon(x,1000,mean,alpha=c(0.025,0.975))$confpoint[,2]
[1] 0.4375507 0.9497833
```

```
abcnon(x,meanp,alpha=c(0.025,0.975))$limits[,2]
[1] 0.4395390 0.9519871
doubleboot(x,mean,0.95,1000,1000)
[1] 0.3865784 0.9616234



# Now our cystic fibrosis data
data_cysfib
standard_c(correl(data)-
    1.96*sqrt(varrho(data[,1],data[,2])),
    correl(data)+1.96*
    sqrt(varrho(data[,1],data[,2])))
standard
[1] -0.6761008  0.1334129
percentile(data,correl,0.05,1000)
[1] -0.6792741  0.1722341
percentile2(data,correl,0.05,1000)
[1] -0.7509027  0.1149186
percentile.t(data,correl,secorr,0.05,1000)
[1] -0.6579483  0.3515261
theta1<-function(x,xdata){
    cor(xdata[x,1],xdata[x,2])}
bcanon(1:length(data[,1]),1000,theta1,data,
    alpha=c(0.025,0.975))$confpoints[,2]
[1] -0.6303270  0.2342018
abcnon(data,correlp,alpha=c(0.025,0.975))$limits[,2]
[1] -0.6145012  0.2370864
doubleboot(data,correl,0.95,1000,1000)
[1] -0.6887344  0.2295549
```

BALANCED RESAMPLING

Uniform resampling selects resample points at random, with replacement from the original data set, but over the complete set of resamples, the original data points may each appear a different number of times.

A more efficient resampling scheme, balanced resampling, ensures that over the complete set of resamples, each data point appears exactly the same number of times.

Both types of resampling produce bootstrap approximations that converge to the same ideal "infinite *B*" bootstrap estimates.

Balanced resampling offers modest efficiency gains over uniform resampling, reductions by a constant factor up to about 3 in the centre of the distribution and just over 1 in the tails, for estimating a distribution function. Hall (1992) gives the asymptotic efficiency as

$$\frac{\Phi(x)\{1-\Phi(x)\}}{\Phi(x)\{1-\Phi(x)\}-\phi(x)^2}$$

Implementing balanced resampling is straightforward:

Step 1: Create B copies of the data, and concatenate them into a single vector of length Bn.

Step 2: Create a random permutation of this vector, and divide it into B pieces, and regard each piece as a resample. Over all resamples, each data point will appear exactly B times.

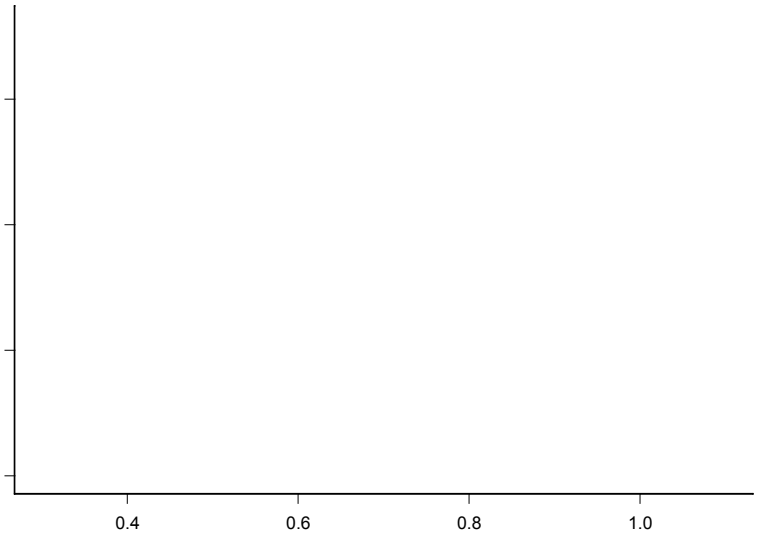
PROS AND CONS OF BALANCED RESAMPLING

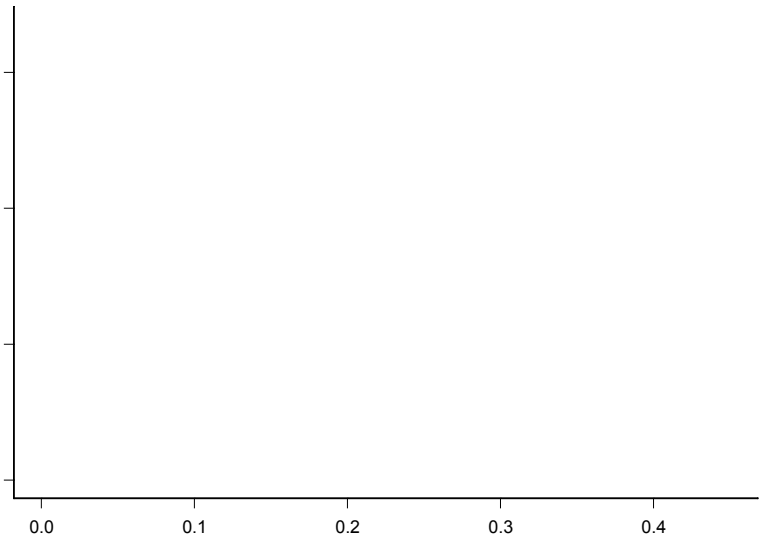
✔ **More efficient than uniform resampling, although gains are only by a constant factor and do not persist into the tails of the distribution (where we most often seek information)**

✘ **Increased storage requirements – all B resamples must be stored simultaneously, while for uniform resampling they need only be stored one at a time**

```
balanceboot <- function(data, B){
# Function takes a vector of data and produces B balanced
# bootstrap replicates of the data. The output is a B times
# n matrix whose rows contain the bootstrap replicates
   n <- length(data)
   boots <- matrix(0, ncol = n, nrow = B)
   bal <- rep(data, B)
   bal <- sample(bal)
   for(i in 1:B) {
      boots[i,] <- bal[seq((i - 1) * n + 1, i * n, 1)]
   }
  boots
}

balancedist <- function(boots, theta){
# Function calculates the bootstrap distribution
# of an estimator theta under balanced bootstrapping
# carried out by the balanceboot routine
   B <- length(boots[, 1])
   thetaboot <- 0
   for(i in 1:B) {
       thetaboot[i] <- theta(boots[i,  ])
   }
  thetaboot
}

z_rexp(20,2)
zbal_balanceboot(z,1000)
zbalmeans_balancedist(zbal,mean)
zbalvars_balancedist(zbal,var)
mean(zbalmeans)
[1] 0.65242
mean(z)
[1] 0.65242
# mean(zbalmeans) and mean(z) are equal because of the
# balanced design of the resamples
var(zbalmeans)
[1] 0.01154344
var(z)/length(z)
[1] 0.01161165
mean(zbalvars)
[1] 0.2200941
var(z)
[1] 0.232233
```

```
hist(zbalmeans,xlab="Balanced bootstrap replications",
    main="Balanced bootstrap means")
abline(v=mean(z),lty=2)
hist(zbalvars,xlab="Balanced bootstrap replications",
    main="Balanced bootstrap variances")
abline(v=var(z),lty=2)
```





EMPIRICAL LIKELIHOOD

Likelihood plays an important role in parametric inference, so it is natural to try to employ a similar approach for nonparametric circumstances.

Suppose we have an i.i.d. sample $X_1,\ldots,X_n$ from an unknown distribution F. The *nonparametric likelihood function* is defined as $L(F)=\prod_{i=1}^{n} F|_{X_i=x_i}$, where x_i denotes the realised value of X_i. The empirical cdf $\hat{F}$ maximises $L(F)$ and so $\hat{F}$ is the nonparametric maximum likelihood estimator of F. As a result, it follows that in estimating a parameter $\theta=\theta(F)$, the nonparametric maximum likelihood estimator for θ is $\hat{\theta}=\theta(\hat{F})$.

The empirical cdf is that of a multinomial distribution with probability vector $\left(\frac{1}{n},\frac{1}{n},\ldots,\frac{1}{n}\right)$. If we consider the more general multinomial model that places probability p_i at X_i, $i=1,\ldots,n$, then the resultant likelihood is $L(F_p)=\prod_{i=1}^{n} p_i$. Ideally,

for inference about θ, we would maximise $L(F_p)$ over all such distributions for which $\hat{\theta}(F_p) = \theta$, a process which corresponds to the *profile likelihood* for θ under the multinomial model having support on the observed data:

$$L_{EL}(\theta) = \sup_{p:\, \hat{\theta}(F_p)=\theta} \prod_{i=1}^{n} p_i .$$

The quantity $L_{EL}(\theta)$ is known as the empirical likelihood. In many cases of interest, the empirical likelihood ratio statistic

$$-2\log\left\{\frac{L_{EL}(\theta)}{L_{EL}(\hat{\theta})}\right\}$$

has an asymptotic χ^2_1 distribution, and inference can be based on this fact.

Unfortunately, the constrained maximisation required to compute the empirical likelihood is often very difficult, and so its use has not become common. Nevertheless, it is an interesting technique that extends our understanding of the links between parametric likelihood-based inference and nonparametric methods.

PROS AND CONS OF EMPIRICAL LIKELIHOOD

- ✔ **Correct "shape" based on the data**
- ✔ **Transformation-respecting**
- ✔ **Range-respecting**

- ✘ **Computationally very difficult to implement in all but the simplest cases**
- ✘ **Only first-order correct**

COMPUTING NOTES

S-Plus is our preferred computing language – it is object-oriented, modular and extensible, and has excellent graphics support. The S-Plus code used in this workshop is available in hard-copy as an Appendix to this work. It is also available on the web at

http://www.statsoc.org.au/~martin/resample.S

The bootstrap functions made available by Efron and Tibshirani as an add-on to their book is available from Statlib (http://lib.stat.cmu.edu) at

http://lib.stat.cmu.edu/S/bootstrap.funs

There is also a pre-compiled version for Windows implementations of S-Plus at

http://lib.stat.cmu.edu/DOS/S/bootstra.zip

Art Owen offers an S-Plus implementation of empirical likelihood at

http://www-stat.stanford.edu/reports/owen/el.S

Other popular statistical programming languages include SAS, SPSS, and Minitab, among others. These packages can be used to explore the world of resampling also, and the

implementation issues are very similar to those encountered using S-Plus.

A number of resources are available on the web to facilitate the implementation of bootstrap and resampling methods in these languages, a brief overview of which is given here.

SAS

The SAS Institute makes available basic jackknife and bootstrap routines from their support website. The routines, %jack and %boot are available from

http://ftp.sas.com/techsup/download/stat/jackboot.sas

Several people have also made their SAS code for resampling available on the internet. Eric Weiss gives an elementary introduction to bootstrapping in SAS at

http://www.winchendon.com/bootstrap.html

while a more detailed discussion, including jackknifing and the bootstrap is given at the University of Texas at

http://www.utexas.edu/cc/docs/stat56.html

Also, Gillian Raab at Napier University in the UK has made available SAS code for

constructing bootstrap inference for means and correlations

http://www.maths.napier.ac.uk/~gillianr/distinf/bootcor.sas
http://www.maths.napier.ac.uk/~gillianr/distinf/bootsdat.sas
http://www.maths.napier.ac.uk/~gillianr/distinf/bootsmean.sas

SPSS

No bootstrap routines are offered directly by SPSS Inc. Nevertheless, some advice on programming the bootstrap in SPSS is available from several people. Rich Herrington provides an elementary discussion of bootstrap confidence intervals at

http://www.unt.edu/UNT/departments/CC/Benchmarks/sprsum97/resamp.htm

while David Hitchin at the University of Sussex implemented Efron's BC_a confidence intervals at

http://pages.infinit.net/rlevesqu/Syntax/Bootstrap/bcnon.txt

Minitab

The venerable Minitab can also be used to conduct resampling. See, for example, Miguel A. Arcones page at

http://www.math.binghamton.edu/arcones/341/7.2.html

A web page listing all of these resources is available at

http://www.statsoc.org.au/~martin/resample.html

Fundamentally, any statistics package that can produce random samples can be used to implement resampling – even a $20 hand calculator with a RAND button could do it if used with enough diligence and patience. An abacus? Well, probably not…but then again, the vigour with which early simulators such as Pearson attacked their tasks suggest nothing is impossible (however unlikely).

With computer power doubling approximately every 18 months, computers today are several thousand times faster than they were when Efron introduced the bootstrap in 1979. Now, complex iterated bootstrap calculations don't have to happen overnight or on a weekend. And, what takes minutes today will take seconds in 2002. The sky is the limit, or so it seems…

REFERENCES

There are several very good books that provide information on the bootstrap and resampling in general. Here is a very brief critique…

Efron and Tibshirani, *An Introduction to the Bootstrap* (Chapman and Hall) – an excellent reference for both beginners and experts. The resampler's Bible.

Davison and Hinkley, *Bootstrap Methods and their Applications* (Cambridge University Press) – a modern and lively treatment, focussing on new applications as well as traditional approaches. Another must.

Chernick, *Bootstrap Methods, A Practitioner's Guide* (Wiley) – a conversational, friendly, but somewhat superficial treatment. An excellent, comprehensive bibliography running to 80 pages of references!

And, for the technically-minded:

Hall, *The Bootstrap and Edgeworth Expansion* (Springer-Verlag) – theory for the bootstrap based on Edgeworth expansions. Seminal development, but very technical.

Shao and Tu, *The Jackknife and Bootstrap* (Springer-Verlag) – Exhaustive, technical treatment of all things resampling. Comprehensive, but not for the faint-hearted as no detail is spared.

Other books (not yet reviewed…)

Good, *Resampling Methods*

Politis, Romano and Wolf, *Subsampling* (Springer-Verlag)

Appendix: S-Plus functions for Applied Resampling Methods

```
# Functions related to the Jackknife
jackknife.bias <- function(data, theta){
# Calculates the jackknife estimate of the bias of an estimator
# of an unknown parameter theta. Input data can be either a vector
# or a matrix whose rows represent data points and a function
# theta.
   theta.hat <- theta(data)
   theta.tld <- 0
   if(is.matrix(data)) {
      for(i in 1:length(data[, 1])) {
          theta.tld[i] <- theta(data[ - i,  ])
      }
      bias <- (length(data[, 1]) - 1) * (mean(theta.tld) - theta.hat)
      }
    else if(is.vector(data)) {
      for(i in 1:length(data)) {
          theta.tld[i] <- theta(data[ - i])
      }
     bias <- (length(data) - 1) * (mean(theta.tld) - theta.hat)
    }
   else print("Invalid data structure")
   print("The jackknife estimate of bias is:", quote = F)
   print(bias)
   bias
}

jackknife.var <- function(data, theta){
# Calculates the jackknife estimate of the variance of an estimator
# of an unknown parameter theta. Input data can be either a vector
# or a matrix whose rows represent data points and a function
# theta.
   theta.i <- 0
   if(is.vector(data)) {
      n <- length(data)
      for(i in 1:n) {
          theta.i[i] <- theta(data[ - i])
      }
    }
    else if(is.matrix(data)) {
      n <- length(data[, 1])
      for(i in 1:n) {
          theta.i[i] <- theta(data[ - i,  ])
      }
    }
    else print("Invalid Data Structure", quote = F)
 ((n - 1)/n) * sum((theta.i - mean(theta.i))^2)
}
```

```
jackrep <- function(data, theta){
# Calculates the jackknife replicates: the values of
# the estimator theta for each of the leave-one-out
# samples
   theta.tld <- 0
   if(is.matrix(data)) {
   for(i in 1:length(data[, 1])) {
       theta.tld[i] <- theta(data[ - i,  ])
       }
   }
   else if(is.vector(data)) {
   for(i in 1:length(data)) {
       theta.tld[i] <- theta(data[ - i])
       }
   }
   else print("Invalid data structure")
   theta.tld
}

pseudovalues <- function(data, theta){
# Calculates the jackknife pseudovalues for an estimator theta
   theta.tld <- 0
   if(is.matrix(data)) {
      n <- length(data[, 1])
      for(i in 1:n) {
          theta.tld[i] <- theta(data[ - i,  ])
      }
   }
   else if(is.vector(data)) {
   n <- length(data)
   for(i in 1:n) {
       theta.tld[i] <- theta(data[ - i])
      }
   }
   else print("Invalid data structure")
   pseudo <- n * theta(data) - (n - 1) * theta.tld
   pseudo
}

# Functions related to permutation tests

permreps <- function(data1, data2, teststat, nreps){
# Calculates the permutation replications for a permutation test
# of the difference between two populations. Inputs are the two
# data sets, a function calculating the test statistic, and the
# required number of permutations. The output vector contains
# the values of the test statistics for the permutations.
   alldata <- c(data1, data2)
   perm <- 0
   for(i in 1:nreps) {
       permute <- sample(alldata)
       permdata1 <- permute[1:length(data1)]
```

```
        permdata2 <- permute[(length(data1) + 1):length(alldata)]
        perm[i] <- teststat(permdata1, permdata2)
    }
    perm
}

asl <- function(data1, data2, teststat, permdata){
# Calculates the ASL and p-value for a permutation test
# for the difference between two distributions. Input
# parameters are the two data sets, a function that
# calculates the test statistics and the permutation
# replications from a call to permreps
    observed <- teststat(data1, data2)
    asl <- sum(observed > permdata)/length(permdata)
    pval <- min(asl, 1 - asl) * 2
    c(asl, pval)
}

# Functions related to bootstrap bias and variance estimation

boot.bias <- function(data, theta, B){
# Calculates the bootstrap estimate of the bias of an
# estimator for a parameter theta. Inputs include the
# data, either as a vector or a matrix whose rows
# represent observations, a function that calculates
# the estimator of theta, and the number of resamples, B.
    theta.star <- 0
    if(is.vector(data)) {
       for(i in 1:B) {
       theta.star[i] <- theta(sample(data, replace = T))
       }
     }
     else if(is.matrix(data)) {
       for(i in 1:B) {
       theta.star[i] <- theta(data[sample(1:length(data[, 1]),
                                replace = T),  ])
       }
     }
     else print("Invalid Data Structure", quote = F)
     mean(theta.star - theta(data))
}

boot.var <- function(data, theta, B){
# Calculates the bootstrap variance of an estimator of an
# unknown parameter theta. Inputs include the
# data, either as a vector or a matrix whose rows
# represent observations, a function that calculates
# the estimator of theta, and the number of resamples, B.
   theta.star <- 0
   if(is.vector(data)) {
     for(i in 1:B) {
         theta.star[i] <- theta(sample(data, replace = T))
```

```
        }
   }
   else if(is.matrix(data)) {
        for(i in 1:B) {
          theta.star[i] <- theta(data[sample(1:length(data[, 1]),
                        replace = T),  ])
        }
   }
   else print("Invalid Data Structure", quote = F)
   var(theta.star)
}

# Functions related to bootstrap confidence intervals

percentile <- function(data, theta, alpha, B){
# Calculates a bootstrap percentile method interval. Input
# data may be either a vector or a matrix whose rows
# represent data points. Also input are the function theta,
# the level alpha (confidence level = 1-alpha) and the
# number of resamples, B.
   theta.star <- 0
   if(is.vector(data)) {
      for(i in 1:B) {
          theta.star[i] <- theta(sample(data, replace = T))
      }
   }
   else if(is.matrix(data)) {
      for(i in 1:B) {
          theta.star[i] <- theta(data[sample(1:length(data[, 1]),
                                replace = T),  ])
          }
      }
   else print("Invalid Data Structure", quote = F)
   sorted <- sort(theta.star)
   interval <- c(sorted[trunc((alpha*B)/2)+1],sorted[trunc((1-
alpha/2)*
                 B) + 1])
   interval
}

percentile2 <- function(data, theta, alpha, B){
# Calculates a hybrid percentile method interval. Input
# data may be either a vector or a matrix whose rows
# represent data points. Also input are the function theta,
# the level alpha (confidence level = 1-alpha) and the
# number of resamples, B.
   thetahat <- theta(data)
   phi.star <- 0
   if(is.vector(data)) {
      for(i in 1:B) {
          phi.star[i] <- theta(sample(data, replace=T))-thetahat
          }
```

```
   }
   else if(is.matrix(data)) {
   for(i in 1:B) {
       phi.star[i] <- theta(data[sample(1:length(data[, 1]),
                       replace = T),  ]) - theta(data)
       }
   }
   else print("Invalid Data Structure", quote = F)
   sorted <- sort(phi.star)
   interval <- c(thetahat - sorted[trunc((1 - alpha/2) * B) + 1],
                 thetahat - sorted[trunc((alpha * B)/2) + 1])
   interval
}

percentile.t <- function(data, theta, se, alpha, B){
# Calculates a bootstrap percentile-t method interval. Input
# data may be either a vector or a matrix whose rows
# represent data points. Also input are the function theta,
# an associated function se which calculates the standard error
# of the estimator, the level alpha (confidence level = 1-alpha)
# and the number of resamples, B.
   thetahat <- theta(data)
   sehat <- se(data, theta, B)
   t.star <- 0
   if(is.vector(data)) {
      for(i in 1:B) {
        resample <- sample(data, replace = T)
        t.star[i]<-(theta(resample)-thetahat)/se(resample,theta,B)
        }
   }
   else if(is.matrix(data)) {
   for(i in 1:B) {
       resample <- data[sample(1:length(data[,1]),replace=T),]
       t.star[i] <- (theta(resample)-
theta(data))/se(resample,theta,B)
       }
   }
   else print("Invalid Data Structure", quote = F)
   sorted <- sort(t.star)
   interval <- c(thetahat-sehat * sorted[trunc((1 - alpha/2) * B) +
1],
                 thetahat - sehat * sorted[trunc((alpha * B)/2) + 1])
   interval
}

# Bootstrap hypothesis testing

bootmeanhyp <- function(data, mu0, B){
# Conducts a bootstrap hypothesis test for a univariate
# mean. Resampling is carried out under the null by recentering
# the data points at the hypothesised mean. Inputs include the
# data vector, the hypothesised value of the mean and the number
```

```
# of resamples, B. Output is the observed test statistic and one-
# and two-sided p-values.
    nulldata <- data - mean(data) + mu0
    nulltest <- 0
    tobs <- (mean(data) - mu0)/semean(data)
    for(i in 1:B) {
       nullresample <- sample(nulldata, replace = T)
       nulltest[i] <- (mean(nullresample)-mu0)/semean(nullresample)
       }
    asl <- sum(nulltest <= tobs)/B
    pval1side <- min(asl, 1 - asl)
    pval2side <- 2 * pval1side
    cbind(tobs, pval1side, pval2side)
}

bootmeanhyp2 <- function(data, mu0, B){
# Alternative function for bootstrap hypothesis test for
# a univariate mean. Resampling uses the standard t-statistic,
# which is equivalent to resampling under the null. Inputs include
the
# data vector, the hypothesised value of the mean and the number
# of resamples, B. Output is the observed test statistic and one-
# and two-sided p-values.
    test <- 0
    tobs <- (mean(data) - mu0)/semean(data)
    for(i in 1:B) {
        resample <- sample(data, replace = T)
        test[i] <- (mean(resample)-mean(data))/semean(resample)
    }
    asl <- sum(test <= tobs)/B
    pval1side <- min(asl, 1 - asl)
    pval2side <- 2 * pval1side
    cbind(tobs, pval1side, pval2side)
}

boot2samphyp <- function(data1, data2, teststat, B){
# Conducts a bootstrap hypothesis test of equality of
# distributions. Inputs include the two data sets, a
# function that calculates a test statistic, and the
# number of resamples, B. Outputs are the observed test
# statistic and one- and two-sided p-values.
   tobs <- teststat(data1, data2)
   alldata <- c(data1, data2)
   nulltest <- 0
     for(i in 1:B) {
        resample <- sample(alldata, replace = T)
        newdata1 <- resample[1:(length(data1))]
        newdata2 <- resample[(length(data1)+1):length(alldata)]
        nulltest[i] <- teststat(newdata1, newdata2)
   }
   asl <- sum(nulltest <= tobs)/B
   pval1side <- min(asl, 1 - asl)
```

```
   pval2side <- 2 * pval1side
   cbind(tobs, pval1side, pval2side)
}

boot2sampreps <- function(data1, data2, teststat, B){
# Outputs the bootstrap replications of the test statistic
# assuming resampling under the null hypothesis of equality
# of distributions for a two-sample bootstrap hypothesis
# test. Inputs include the two data sets, a
# function that calculates a test statistic, and the
# number of resamples, B. Output is a list, the first
# component of which, obs, is the observed test statistic,
# and the second of which is a vector (bootreps) containing
# the bootstrap replications.
    tobs <- teststat(data1, data2)
    alldata <- c(data1, data2)
    nulltest <- 0
    for(i in 1:B) {
       resample <- sample(alldata, replace = T)
       newdata1 <- resample[1:(length(data1))]
       newdata2 <- resample[(length(data1)+1):length(alldata)]
       nulltest[i] <- teststat(newdata1, newdata2)
    }
    list(obs = tobs, bootreps = nulltest)
}

bootcorrhyp <- function(data, B){
# Function to conduct a bootstrap test of no correlation
# between two variables. Inputs include the data as a
# matrix whose columns represent the two data sets, and
# the number of resamples, B. Outputs include the observed
# correlation coefficient and one- and two-sided bootstrap
# p-values
   nulltest <- 0
   n <- length(data[, 1])
   rhoobs <- correl(data)
     for(i in 1:B) {
        newx <- data[sample(1:n, replace = T), 1]
        newy <- data[sample(1:n, replace = T), 2]
        newdata <- cbind(newx, newy)
        nulltest[i] <- correl(newdata)
     }
    asl <- sum(nulltest <= rhoobs)/B
    pval1side <- min(asl, 1 - asl)
    pval2side <- 2 * pval1side
    cbind(rhoobs, pval1side, pval2side)
}


boottimehyp <- function(data, B){
# Constructs a bootstrap test for first-order serial
# correlation for a time series. Bootstrap test statistic
```

```
# is the first autocorrelation coefficient. Input is the
# time series, and the number of resamples, B. Output includes
# the observed autocorrelation, and one- and two-sided bootstrap
# p-values. Resampling is carried out under the null by using
# i.i.d. uniform resampling
    obs <- acf(data, plot = F)$acf[2]
    n <- length(data)
    nullacf <- 0
    for(i in 1:B) {
        resample <- data[sample(1:n, replace = T)]
        nullacf[i] <- acf(resample, plot = F)$acf[2]
        }
    asl <- sum(nullacf <= obs)/B
    pval1side <- min(asl, 1 - asl)
    pval2side <- 2 * pval1side
    cbind(obs, pval1side, pval2side)
}

boottimereps <- function(data, B){
# Calculates bootstrap replications of the first-order
# serial correlation for a time series. Related to the
# bootstrap hypothesis test of no first-order serial
# correlation.
    obs <- acf(data, plot = F)$acf[2]
    n <- length(data)
    nullacf <- 0
    for(i in 1:B) {
        resample <- data[sample(1:n, replace = T)]
        nullacf[i] <- acf(resample, plot = F)$acf[2]
        }
    list(obs = obs, dist = nullacf)
}

# Bootstrapping for regression

bootreg <- function(data, B, residual = T, ythenx = T){
# Calculates bootstrap regression coefficients and s.e.'s.
# Function assumes data is in matrix form. If the response is
# in the first column ythenx is set as True (default). If the
# response is in the last column ythenx is set as False.
# Resampling can either be via residuals (setting residual=T,
# default) or by data points (residual=F). B is number of resamples
# The output is a list whose first component
# $bootcoef is a matrix whose i'th column contains B bootstrap
# replications of the (i-1)th coefficient (the first column is
# for the model intercept). The second component of the list is
# the corresponding standard errors
    data <- as.matrix(data)
    if(ythenx) {
      y <- data[, 1]
      x <- data[, -1]
     }
```

```
     else {
           m <- length(data[1,  ])
           y <- data[, m]
           x <- data[,  - m]
     }
     n <- length(y)
     bootcoef <- matrix(0, nrow = B, ncol = (length(data[1,  ])))
     bootse <- matrix(0, nrow = B, ncol = (length(data[1,  ])))
     if(residual) {
        model <- lsfit(x, y)
        for(i in 1:B) {
            bootres <- sample(model$residuals, replace = T)
            newy <- cbind(1, x) %*% model$coef + bootres
            newmod <- lsfit(x, newy)
            bootcoef[i,  ] <- newmod$coef
            bootse[i,  ] <- ls.diag(newmod)$std.err
            }
        }
        else {
            for(i in 1:B) {
            bootind <- sample(1:n, replace = T)
            if(length(data[1,  ]) > 2) {
            newmod <- lsfit(x[bootind,  ], y[bootind])
            }
            else {
            newmod <- lsfit(x[bootind], y[bootind])
            }
            bootcoef[i,  ] <- newmod$coef
            bootse[i,  ] <- ls.diag(newmod)$std.err
            }
        }
    list(bootcoef = bootcoef, bootse = bootse)
}

percentilereg <- function(data, B, coefficient, residual=T, ythenx=T,
alpha){
# Takes the output list from a call to bootreg and produces a
# percentile interval for the beta coefficient required. e.g.
# if coefficient is set as 0 the interval is for the intercept
# and so on. The residual option dictates whether resampling is
# residual based, and ythenx option dictates how the data is
# originally presented (y as first column versus y as last column)
   bootr <- bootreg(data, B, residual, ythenx)
   bootreps <- bootr$bootcoef[, (coefficient + 1)]
   sorted <- sort(bootreps)
   interval <- c(sorted[trunc((alpha * B)/2) + 1],
               sorted[trunc((1 - alpha/2) * B) + 1])
   interval
}

percentile.treg <- function(data, B, coefficient, residual = T,
ythenx = T, alpha){
```

```
# Takes the output list from a call to bootreg and produces a
# percentile-t interval for the beta coefficient required. e.g.
# if coefficient is set as 0 the interval is for the intercept
# and so on. The residual option dictates whether resampling is
# residual based, and ythenx option dictates how the data is
# originally presented (y as first column versus y as last column)
     bootr <- bootreg(data, B, residual, ythenx)
     bootreps <- bootr$bootcoef[, (coefficient + 1)]
     bootses <- bootr$bootse[,(coefficient+1)]
     if(ythenx) {
          origmodel <- lsfit(data[, -1], data[, 1])
     }
     else {
          origmodel <- lsfit(data[,  - length(data[
               1,  ])], data[, length(data[1,  ]
               )])
     }
     thetahat <- origmodel$coef[(coefficient + 1)]
     sehat <- ls.diag(origmodel)$std.err[(coefficient +
          1)]
     t.star <- (bootreps - thetahat)/bootses
     sorted <- sort(t.star)
     interval <- c(thetahat - sehat * sorted[trunc((1 -
          alpha/2) * B) + 1], thetahat - sehat *
          sorted[trunc((alpha * B)/2) + 1])
     interval
}

# Bootstrapping time series

timeboot <- function(data, B, residual = T, ...){
# Bootstrap replications of time series data. Input data is
# the original series. If residual=T, resampling is based
# on residuals of an autoregressive model of order p, where
# p is input as an optional parameter. If residual=F, block
# bootstrapping is carried out and the block length l is
# required as an optional parameter. Output is in the form
# of a matrix whose rows are the resampled time series.
   data <- ts(data)
   bootreps <- matrix(0, nrow = B, ncol = length(data))
   if(residual) {
      newdata <- data
      p <- c(...)
      model <- ar(data, aic = F, order.max = p)
      for(i in 1:B) {
         bootres <- c(rep(0, p), sample(model$resid[!is.na(model$
                     resid)], replace = T))
         for(j in (p + 1):length(data)) {
            newdata[j] <- t(newdata[seq(j - 1, j - p, -1)]) %*%
                          model$ar + bootres[j]
            }
         bootreps[i,  ] <- newdata
```

```
        }
      }
      else {
      blocklength <- c(...)
      if((trunc(length(data)/blocklength) == (length(data)/
               blocklength))) {
      blocks <- matrix(0, ncol = blocklength, nrow = (length(
               data)/blocklength))
      for(i in 1:(length(data)/blocklength)) {
          blocks[i,  ] <- data[seq(((i - 1) * blocklength+1),
                                   i * blocklength, 1)]
          }
          for(i in 1:B) {
              bootind <- sample(1:trunc(length(data)/blocklength),
                                replace = T)
              bootreps[i,  ] <- c(t(blocks[bootind,  ]))
          }
      }
      else {
      bootreps <- 1
      print("Length of data not a multiple of block length")
      }
  }
  bootreps
}

percentile.time <- function(bootreps, alpha, order, whichcoef){
# Calculates a percentile bootstrap confidence interval for an
# autoregressive coefficient given the bootstrap replications of a
# time series, output by the timeboot routine. Inputs include the
# bootstrap replications in a matrix (bootreps) whose rows represent
# the bootstrap replications, the level alpha (confidence=1-alpha),
# the order of the required autoregression, and the index of the
# required coefficient
   coef <- 0
   B <- length(bootreps[, 1])
   for(i in 1:B) {
       model <- ar(bootreps[i,  ], aic = F, order.max = order)
       coef[i] <- model$ar[whichcoef]
   }
   sorted <- sort(coef)
   interval <- c(sorted[trunc((alpha * B)/2) + 1],
                 sorted[trunc((1 - alpha/2) * B) + 1])
   interval
}

percentile.treps_function(bootreps,order,whichcoef){
# Calculates the bootstrap replicates of an autoregressive
# coefficient using resamples constructed by the timeboot
# routine
     coef <- 0
     B <- length(bootreps[, 1])
```

```
     for(i in 1:B) {
          model <- ar(bootreps[i,  ], aic = F,
               order.max = order)
          coef[i] <- model$ar[whichcoef]
     }
  coef
  }

# Iterated bootstrap bias correction

doubleboot.biascor <- function(data, theta, B1, B2){
# Calculates the iterated bootstrap bias-corrected estimate of
# an unknown parameter theta. Input includes the data (vector or
# matrix), function theta, and inner and outer level numbers
# of resamples
     theta.star <- 0
     theta.starstar <- 0
     temp <- 0
     if(is.vector(data)) {
        for(i in 1:B1) {
            resample <- sample(data, replace = T)
            theta.star[i] <- theta(resample)
            for(j in 1:B2) {
                reresample <- sample(resample, replace = T)
                theta.starstar[j] <- theta(reresample)
                }
            temp[i] <- mean(theta.starstar)
            }
        level2 <- mean(temp)
        level1 <- mean(theta.star)
      }
      else if(is.matrix(data)) {
        for(i in 1:B1) {
            resample <- data[sample(1:length(data[,1]),replace=T),]
            theta.star[i] <- theta(resample)
            for(j in 1:B2) {
             reresample<-
resample[sample(1:length(data[,1]),replace=T),]
             theta.starstar[j] <- theta(reresample)
            }
            temp[i] <- mean(theta.starstar)
         }
         level2 <- mean(temp)
         level1 <- mean(theta.star)
      }
      else print("Invalid Data Structure", quote = F)
      dbbiascor <- 3 * mean(theta(data)) - 3 * level1 + level2
     dbbiascor
}


# Iterated bootstrap confidence intervals
```

```
doubleboot <- function(data, theta, alpha, b1, b2){
# Calculates an iterated bootstrap confidence interval for a
# parameter theta. Input data is either a vector or a matrix
# whose rows represent data points. Other inputs include the
# function theta for the estimator, confidence alpha (usually
# 0.9 or 0.95), and inner (b2) and outer (b1) loop
# numbers of resamples. Output are the confidence interval
# endpoints
    if(is.vector(data)) {
       n <- length(data)
    }
    else {
       n <- length(data[, 1])
    }
    thetahat <- theta(data)
    alpha1 <- alpha - 0.15
    gamma11 <- (1 - alpha1)/2
    gamma12 <- (1 + alpha1)/2
    alpha2 <- alpha
    gamma21 <- (1 - alpha2)/2
    gamma22 <- (1 + alpha2)/2
    alpha3 <- 0.999
    gamma31 <- (1 - alpha3)/2
    gamma32 <- (1 + alpha3)/2
    probest <- rep(0, b1)
    thetastar <- rep(0, b1)
    if(is.vector(data)) {
       for(i in 1:b1) {
           resample <- data[sample(1:n, replace = T)]
           thetastar[i] <- theta(resample)
           thetastarstar <- rep(0, b2)
           for(j in 1:b2) {
               reresample <- resample[sample(1:n, replace = T)]
               thetastarstar[j] <- theta(reresample)
               }
           probest[i] <- sum(thetastarstar <= thetahat)/b2
           }
       }
     if(is.matrix(data)) {
        for(i in 1:b1) {
            resample <- data[sample(1:n, replace = T),  ]
            thetastar[i] <- theta(resample)
            thetastarstar <- rep(0, b2)
            for(j in 1:b2) {
                reresample <- resample[sample(1:n, replace = T),]
                thetastarstar[j] <- theta(reresample)
                }
            probest[i] <- sum(thetastarstar <= thetahat)/b2
            }
        }
     pi1 <- sum((probest < gamma12) & (probest > gamma11))/b1
```

```
     pi2 <- sum((probest < gamma22) & (probest > gamma21))/b1
     pi3 <- sum((probest < gamma32) & (probest > gamma31))/b1
     aaa <- pi1/((alpha1 - alpha2)*(alpha1 - alpha3))-pi2/((alpha1 -
       alpha2) * (alpha2 - alpha3)) + pi3/((alpha1 - alpha3) *
(alpha2 -
       alpha3))
     bbb <- ( -pi1*(alpha2 + alpha3))/((alpha1 - alpha3) * (alpha1 -
       alpha2)) + (pi2 * (alpha1 + alpha3))/((alpha1 - alpha2) * (
       alpha2 - alpha3)) - (pi3 * (alpha1 + alpha2))/((alpha1 -
alpha3)
       *(alpha2 - alpha3))
     ccc <- (pi1*alpha2*alpha3)/((alpha1-alpha3)*(alpha1-alpha2))-(
       pi2*alpha1*alpha3)/((alpha1-alpha2)*(alpha2-
alpha3))+(pi3*alpha1
       *alpha2)/((alpha1 - alpha3) * (alpha2 - alpha3))
     delta1 <- (-bbb+sqrt(bbb*bbb-4*aaa*(ccc-alpha)))/(2*aaa)
     delta2 <- (-bbb-sqrt(bbb*bbb-4*aaa*(ccc-alpha)))/(2*aaa)
     delta <- delta2
     if(abs(delta1 - alpha) < abs(delta2 - alpha))
        delta <- delta1
        if(delta > 1)
           delta <- 1
     sortedthetas <- sort(thetastar)
     interval <- c(sortedthetas[trunc((1-delta)/2*b1)+1],
sortedthetas[
                 trunc((1+delta)/2*b1)])
     interval
}

# Balanced Bootstrapping

balanceboot <- function(data, B){
# Function takes a vector of data and produces B balanced
# bootstrap replicates of the data. The output is a B times
# n matrix whose rows contain the bootstrap replicates
        n <- length(data)
        boots <- matrix(0, ncol = n, nrow = B)
        bal <- rep(data, B)
        bal <- sample(bal)
        for(i in 1:B) {
                boots[i,  ] <- bal[seq((i - 1) * n + 1, i * n, 1)]
        }
        boots
}

balancedist <- function(boots, theta){
# Function calculates the bootstrap distribution
# of an estimator theta under balanced bootstrapping
# carried out by the balanceboot routine
        B <- length(boots[, 1])
        thetaboot <- 0
        for(i in 1:B) {
```

```
                thetaboot[i] <- theta(boots[i,  ])
        }
        thetaboot
}

# General auxiliary functions

correl <- function(data){
# Calculates the correlation coefficient between two
# data sets input as the columns of a matrix
    correl <- cor(data)[1, 2]
    correl
}

correlp <- function(p, x){
# The correlation coefficient in "resampling form"
   x1m <- sum(p * x[, 1])/sum(p)
   x2m <- sum(p * x[, 2])/sum(p)
   num <- sum(p * (x[, 1] - x1m) * (x[, 2] - x2m))
   den <- sqrt(sum(p * (x[, 1] - x1m)^2) * sum(p * (x[, 2] - x2m)^2))
   return(num/den)
}

meanp <- function(p, x){
# The sample mean in "resampling form"
        sum(p * x)/sum(p)}

seboot <- function(data, theta, B){
# Calculates the bootstrap standard error of an estimator
# theta. Based on the boot.var routine.
   seboot <- sqrt(boot.var(data, theta, B))
   seboot
}

secorr <- function(data, theta, B){
# Calculates the standard error of the sample correlation
# coefficient.
   secorr <- sqrt(varrho(data[, 1], data[, 2]))
   secorr
}

sejack <- function(data, theta, B){
# Calculates the jackknife estimate of the standard error of
# an estimator theta. Based on the jackknife.var routine.
   sejack <- sqrt(jackknife.var(data, theta))
   sejack
}

semean <- function(data, theta, B){
# Calculates the standard error of the mean
   semean <- sqrt(var(data)/length(data))
   semean}
```

```
teststat2 <- function(data1, data2){
# Sample test statistic for hypothesis test for
# difference between two populations
   teststat2 <- mean(data1) - mean(data2)
   teststat2
}

teststat3 <- function(data1, data2){
# Sample test statistic for hypothesis test for
# difference between two populations
   teststat3 <- sqrt(var(data1))/sqrt(var(data2))
   teststat3
}

tteststat <- function(data1, data2){
# Sample test statistic, here the usual pooled variance t-stat
   v1 <- var(data1)
   v2 <- var(data2)
   n1 <- length(data1)
   n2 <- length(data2)
   spooled <- sqrt(((n1 - 1) * v1 + (n2 - 1) * v2)/(n1 + n2 - 2))
   teststat <- (mean(data1)-mean(data2))/(spooled*sqrt(1/n1+1/n2))
   teststat
}

varrho <- function(x, y){
# Formula for the variance of the sample correlation coefficient
   n <- length(x)
   rho <- cor(x, y)
   x1 <- x - mean(x)
   y1 <- y - mean(y)
   mu11 <- mean(x1 * y1)
   mu20 <- mean(x1 * x1)
   mu02 <- mean(y1 * y1)
   mu22 <- mean(x1 * x1 * y1 * y1)
   mu31 <- mean(x1 * x1 * x1 * y1)
   mu13 <- mean(x1 * y1 * y1 * y1)
   mu40 <- mean(x1 * x1 * x1 * x1)
   mu04 <- mean(y1 * y1 * y1 * y1)
   varrho <- (rho * rho)/n * (mu22/(mu11 * mu11) + 0.25 * (mu40/(mu20
*
      mu20) + mu04/(mu02 * mu02) + (2 * mu22)/(mu02 * mu20)) -
(mu31/(
      mu20 * mu11) + mu13/(mu02 * mu11)))
   varrho
}

wrongvar <- function(data){
# Maximum likelihood variance estimator - uses divisor n instead of
# n-1 and is hence biased
   wrongvar <- (var(data) * (length(data) - 1))/length(data)
   wrongvar}
```