
TIBCO Spotfire S+® Resample Library User's Manual

TIBCO Software Inc.

**Proprietary
Notice**

TIBCO Software Inc. owns both this software program and its documentation. Both the program and documentation are copyrighted with all rights reserved by TIBCO Software Inc.

The correct bibliographical reference for this document is as follows:

Spotfire S+ Resample Library User's Manual, TIBCO Software Inc.
Printed in the United States.

Copyright Notice Copyright © 1987-2008, TIBCO Software Inc. All rights reserved.

ACKNOWLEDGMENTS

The S+ Resample library has been supported by

- NIH SBIR 2R44CA67734--02 "Statistical Software for Resampling Methods"
- NSF SBIR DMI-0078706 "Bootstrap Tilting Inference and Large Data Sets"

Contributors include: Ashley Clipson, Steve Ellis, Rachel Epstein, Chris Fraley, Tim Hesterberg, Shan Jin, Charles Roosen, Jim Schimert, Robert Thurman.

We also appreciate advice from Brad Efron, Frank Harrell, Art Owen, Luigi Salmaso, Terry Therneau, and Rob Tibshirani.

	1
Chapter 1 Spotfire S+ Resample Library	1
Introduction	2
Overview of capabilities	3
Graphical Interface	5
Basic Resampling Techniques	7
The Bootstrap	7
The Jackknife	15
Permutation Tests	17
Confidence Limits	23
Empirical Percentiles	23
BCa Percentiles	23
Tilting Percentiles	24
Bootstrap-t intervals	24
More Examples	25
Resampling the Correlation Coefficient	25
Jackknife after bootstrap	27
Resampling Regression Coefficients	30
Influence and Linear Approximations	37
The influence function	38
The limits.abc function	43
The resampGetL function	44
Parametric and Smoothed Bootstrapping	47
The parametric bootstrap	47
The smoothed bootstrap	50
Parametric Bootstrap Testing	53
Another example	55
Estimating Prediction Error	59
Cross-validation: the crossVal function	60
Bootstrap prediction: the bootPred function	63
Methods for Resample Objects	67
The print Method	67
The summary Method	67
The plot Method	67
Normal Quantile-Quantile Plots	68

GUI Instructions (Windows only)	69
The Menu Hierarchy	69
References	83

Spotfire S+ RESAMPLE LIBRARY

1

INTRODUCTION

In statistical analysis, we are usually interested in obtaining not only a point estimate of a statistic but also an estimate of the variation in this point estimate, and a confidence interval for the true value of the parameter. For example, we may calculate not only a sample mean, but also the standard error of the mean and a confidence interval for the mean.

Traditionally, researchers have relied on the central limit theorem and normal approximations to obtain standard errors and confidence intervals. These techniques are often based on assumptions that cannot be justified on physical grounds. The resulting inferences may be valid asymptotically, but in practice are inaccurate for finite sample sizes. In other cases inferences are not even valid asymptotically, such as when the residual variance is not constant in regression or ANOVA.

A major motivation for the traditional reliance on normal-theory methods has been computational tractability. Now, with the availability of modern computing power, we need no longer rely on asymptotic theory to estimate the distribution of a statistic. Instead, we may use resampling methods which return inferential results for either normal or nonnormal distributions.

Resampling techniques such as the bootstrap and jackknife provide estimates of the standard error, confidence intervals, and sampling distributions for any statistic. In the bootstrap, for example, B new samples, each of the same size as the observed data, are drawn with replacement from the observed data. The statistic is calculated for each new set of data, yielding a bootstrap distribution for the statistic. The fundamental assumption of bootstrapping is that the observed data are representative of the underlying population. By resampling observations from the observed data, the process of sampling observations from the population is mimicked. For more detailed descriptions of bootstrapping, see Efron and Tibshirani (1993), Davison and Hinkley (1997), and Shao and Tu (1995).

Overview of capabilities

Spotfire S+ includes a suite of functions for resampling, including these capabilities:

- Given a vector, matrix, or data frame, create bootstrap or jackknife resamples of observations and use these to calculate resampling replicates of a specified statistic.
- Perform permutation tests for hypothesis testing.
- Use many different sampling mechanisms, including parametric bootstrapping, sampling by group or subject, finite population sampling, and block bootstrapping for time series.
- Calculate bootstrap tilting, BCa, and bootstrap-t confidence limits for a bootstrap object, and empirical percentiles for a jackknife, permutation, and bootstrap objects.
- Test the fundamental bootstrap assumption using bootstrap tilting and jackknife-after-bootstrap diagnostics.
- Calculate empirical influence values, which measure the influence of observations on a given statistic, using jackknife, finite difference, or bootstrap regression methods. Use those for linear approximations to bootstrap replicates and to estimate confidence limits.
- Use jackknife-after-bootstrap to examine the influence of observations on bootstrap distributions, and to estimate the standard error of a functional of the bootstrap distribution for a statistic.
- Use cross-validation or bootstrap estimates to measure prediction error; that is, how well a model predicts future observations.

A list of resampling functions is presented in Table 1.1. Not all of the functions in the table are described in this manual. See the help files for individual functions for more information.

Table 1.1: *S-PLUS bootstrapping and jackknifing functions.*

Function	Description
bootstrap	Main bootstrap function
jackknife	Main jackknife function
bootstrap.lm bootstrap.glm bootstrap.censorReg jackknife.lm jackknife.glm jackknife.censorReg	Modeling function methods for bootstrap and jackknife
addSamples	Add more replicates to a bootstrap object
permutationTest permutationTestMeans	Perform permutation tests
limits.emp limits.bca limits.abc limits.tilt bootstrapT	Calculate empirical percentiles, or a variety of confidence limits
jack.after.bootstrap tilt.after.bootstrap reweight	Perform diagnostics, examine influence of observations or parameters on bootstrap distributions.
influence resampGetL	Empirical influence, linear approximations
pbootstrap	Perform parametric bootstrap
pbootTest	Perform parametric bootstrap hypothesis testing

Table 1.1: *S-PLUS bootstrapping and jackknifing functions. (Continued)*

Function	Description
sbootstrap	Perform smoothed bootstrap
crossVal bootPred	Estimate prediction error
samp.MonteCarlo samp.boot.bal samp.permute samp.bootknife samp.finite samp.boot.block blockBootstrap	Sampling methods
print.resamp plot.resamp print.jack.after.bootstrap plot.jack.after.bootstrap qqnorm.resamp summary.resamp update.bootstrap	Methods for resamp objects. This list is not complete; other classes have their own methods for functions listed here.

Graphical Interface

The bootstrap library also features a graphical interface, on Windows only. This is described in the section "GUI Instructions (Windows only)" on page 69.

BASIC RESAMPLING TECHNIQUES

Spotfire S+ supports a number of resampling techniques. Here we consider three: the bootstrap, the jackknife, and permutation tests. These are implemented by functions `bootstrap`, `jackknife` and `permutationTest`. A specialized version of the latter, `permutationTestMeans`, is used for the difference of two sample means. These functions produce bootstrap objects, jackknife objects, and permutationTest objects, respectively, all of which inherit from the generic `resamp` object class. These various techniques are described in the following sections. Cross-validation, another important resampling technique, implemented by function `crossVal`, is described in a separate section.

The Bootstrap

In bootstrap resampling, B new samples, each the same size as the observed data, are drawn with replacement from the observed data. The statistic is first calculated using the observed data and then recalculated using each of the new samples, yielding a bootstrap distribution. The distributions are used to calculate the bootstrap estimates of bias, mean, and standard error for the statistic, and ultimately confidence intervals or other quantities.

Example: Resampling the Variance

The `swiss.x` matrix contains socioeconomic indicators for the provinces of Switzerland in 1888. We are interested in the variance of the `Education` variable, which contains the percent of the population whose education is beyond primary school.

```
> swiss.x[,"Education"]
 [1] 12  9  5  7 15  7  7  8  7 13  6 12  7 12  5  2  8 28 20
[20]  9 10  3 12  6  1  8  3 10 19  8  2  6  2  6  3  9  3 13
[39] 12 11 13 32  7  7 53 29 29
```

The `bootstrap` function is used to draw resamples and construct a bootstrap object.

```
> Edu <- swiss.x[,"Education"]
> boot.obj1 <- bootstrap(data = Edu, statistic = var,
+   B = 1000, seed = 0)
Forming replications 1 to 100
Forming replications 101 to 200
Forming replications 201 to 300
Forming replications 301 to 400
```

```

Forming replications 401 to 500
Forming replications 501 to 600
Forming replications 601 to 700
Forming replications 701 to 800
Forming replications 801 to 900
Forming replications 901 to 1000

```

The main arguments to `bootstrap` are `data` (a vector, matrix, or data frame) and `statistic`, which returns a scalar, vector, or matrix. The `statistic` may be an S-PLUS function (like `var`) or an expression (like `mean(data, trim = .2)`, for example). The number of resamples to draw is `B`. The default is 1000, which is the recommended minimum for estimating percentiles (however, newer research indicates that more are needed for accurate BCa limits). Although a smaller `B` may be specified, 250 is recommended as a minimum for estimating standard errors. The optional `seed` argument is used here to set the random number seed. This is important for reproducibility, since bootstrap resamples are random. To prevent the printed messages from being displayed, set `trace=F`. There are a number of optional arguments for controlling how the resamples are generated, how the statistic is calculated, the output components, and various other calculations. These are described in the `bootstrap` and `bootstrap.args` help files. A few are described below.

Printing the bootstrap object displays the call used to construct it, the number of replications used, and summary statistics for the parameter. The summary statistics are the observed value of the parameter, the mean of the parameter estimate replicates, and bootstrap estimates of bias and standard error.

```

> boot.obj1
Call:
bootstrap(data = Edu, statistic = var, B = 1000, seed = 0)

Number of Replications: 1000

Summary Statistics:
      Observed Mean   Bias   SE
var      92.46 89.09 -3.362 38.67

```

A more complete summary of the bootstrap object, obtained via the `summary` function, includes empirical and BCa percentiles for the statistic. These are described in detail in a later section.

```
> summary(boot.obj1)
Call:
bootstrap(data = Edu, statistic = var, B = 1000, seed = 0)

Number of Replications: 1000

Summary Statistics:
      Observed Mean   Bias   SE
var    92.46 89.09 -3.362 38.67

Empirical Percentiles:
      2.5%  5.0% 95.0% 97.5%
var 32.82 36.15 164.4 177.9

BCa Confidence Limits:
      2.5%   5%  95% 97.5%
var 45.31 51.43 215.2 223.5
```

The BCa percentiles, for example, show that the 95% confidence interval for the Education variance has endpoints 51.43 and 215.2.

You can plot the bootstrap object using `plot`.

```
> plot(boot.obj1)
```

See Figure 1.1. The `plot` method for resamp objects like `bootstrap` provides a histogram of the replicated variances along with a smooth density estimate. The solid vertical line indicates the observed parameter value, and the dotted line indicates the mean of the replicates. The difference between these two values is the bootstrap estimate of bias.

The histogram shows that the distribution of replicated variances is highly skewed. A normal quantile-quantile plot can be used to further assess deviation from the normal distribution.

```
> qqnorm(boot.obj1)
```

See Figure 1.2. In a qq-plot, if the points fall on a straight line the empirical distribution of the replicates is similar to that of a normal random variate. Figure 1.2 suggests that both tails of the empirical distribution deviate from the normal distribution. Thus there is evidence that bootstrapping is a better approach than normal-based methods.

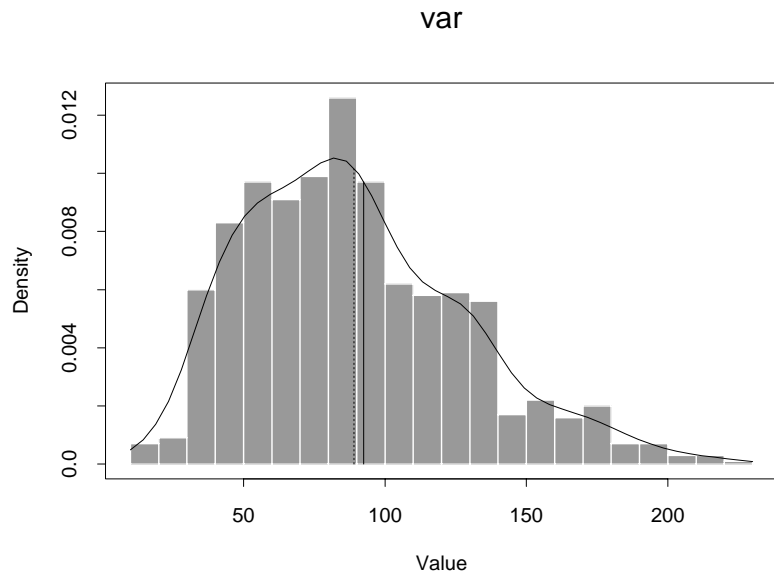


Figure 1.1: *Histogram of bootstrapped variances.*

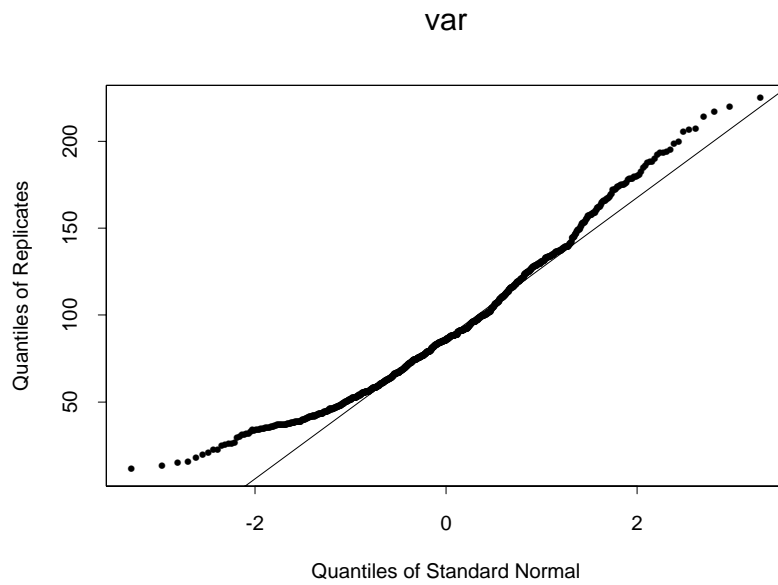


Figure 1.2: *Normal qq-plot of replicated variances.*

The group argument

The optional bootstrap argument `group` supports both stratified sampling, and two-sample or multiple-sample applications; resampling is performed independently within each stratum or sample.

For example, let's explore the degree to which education affects infant mortality in the `swiss.x` data set. We can bootstrap the difference in the mean infant mortality rates between those provinces having more than 20% of their citizens educated past primary school and those having less than 20%. First create a new data frame containing just the data we need.

```
> swiss.new <- data.frame(Edu=swiss.x["Education"],
+   IM=swiss.x["Infant Mortality"])
> boot.obj <- bootstrap(data = swiss.new,
+   statistic = mean(IM[Edu>20]) - mean(IM[Edu<=20]),
+   group = Edu>20, seed = 0, trace = F)
```

In this example the `group` argument is a logical vector calculated from the original `Edu` column – there are 5 T's and 42 F's. Each resample thus consists of 5 samples drawn with replacement from the higher-educated provinces and 42 drawn from the lower-educated provinces. The statistic is the difference of the means of those two groups. Let's look at the summary of the results.

```
> summary(boot.obj)
Call:
bootstrap(data = swiss.new, statistic = mean(IM[Edu > 20])
- mean(IM[Edu <= 20])
), group = Edu > 20, seed = 0, trace = F)
```

```
Number of Replications: 1000
```

```
Summary Statistics:
```

	Observed	Mean	Bias	SE
Param	-0.2267	-0.2213	0.005349	0.929

```
Empirical Percentiles:
```

	2.5%	5.0%	95.0%	97.5%
Param	-1.878	-1.654	1.366	1.632

```
BCa Confidence Limits:
```

	2.5%	5%	95%	97.5%
Param	-1.775	-1.595	1.434	1.75

The mean and observed values both suggest infant mortality is higher in the less educated provinces. But both of the 5% confidence intervals contain 0, so it's difficult to argue that there is a statistically significant difference between the two groups.

The subject argument

In sampling by subject, observations are grouped together to form subjects, and it is the subjects which are resampled, not the observations. Unique values of the `subject` argument determine the subjects.

For example, consider the `solder` data frame, which contains 900 observations which are the results of an experiment varying 5 factors relevant to the wave-soldering procedure for mounting components on printed circuit boards. The response variable, `skips`, is a count of how many solder skips appeared to a visual inspection. The `Opening` variable is an ordered factor indicating the amount of clearance around the mounting pad (**S** < **M** < **L**). There are 300 observations for each value of `Opening`. First we bootstrap the average value of `skips` for the observations for which `Opening` is "M".

```
> bootstrap(solder[solder$Opening == "M",], mean(skips),
+          seed = 0, trace = F)
Call:
bootstrap(data = solder[solder$Opening == "M", ],
  statistic = mean(
    skips), seed = 0, trace = F)
```

```
Number of Replications: 1000
```

```
Summary Statistics:
```

	Observed	Mean	Bias	SE
Param	3.57	3.57	-0.0003467	0.2714

The `PadType` factor indicates which of 10 mounting pads were used. There are 30 observations per pad type when `Opening` equals "M". If we use `subject = PadType`, each 30-observation subject is treated as a unit while resampling. This effectively reduces the sample size from 300 to 10.

```
> bootstrap(solder[solder$Opening == "M",], mean(skips),
+          subject = PadType, seed = 0, trace = F)
Call:
bootstrap(data = solder[solder$Opening == "M", ],
  statistic = mean(
```

```
skips), subject = PadType, seed = 0, trace = F)
```

```
Number of Replications: 1000
```

```
Summary Statistics:
```

	Observed	Mean	Bias	SE
Param	3.57	3.569	-0.0006633	0.5328

Note that the standard error is about twice that of the non-hierarchical case.

The `group` and `subject` arguments may be used simultaneously. In this case, `subject` must be nested within `group`; that is, observations for a given subject may appear in only one group.

Other sampling methods - the `sampler` argument

For bootstrapping to be accurate, the bootstrap sampling mechanism must match the sampling mechanism that produced the data. By default, `bootstrap` uses simple random sampling, drawing n numbers with replacement from the set $1:n$, where n is the sample size. Various alternatives to simple random sampling are available, using the `sampler` argument. No matter which sampler you choose, if `group` is supplied then samples are generated independently for each group, and if `subject` is supplied then the sampler selects indices for subjects.

- `samp.MonteCarlo` (formerly `samp.boot.mc`) is the default, performing simple random sampling.
- `samp.bootknife` and `samp.boot.bal` are minor variations on bootstrap sampling, intended to avoid the downward bias of bootstrap variances and standard errors, and to give more accurate estimates of bootstrap bias, respectively.
- `samp.finite` performs finite population sampling.
- `samp.permute` generates random permutations
- `samp.blockBootstrap` does block bootstrap sampling, for time series.
- `samp.combinations` is useful for complete enumeration in two-sample permutation tests.

The samples have various arguments, either required (e.g. you must supply the population size N to `samp.finite`) or optional (e.g. for `samp.MonteCarlo`, to draw observations with unequal probabilities, or with `size` that differs from n , where n is the sample size). These can be passed using either the `sampler.args` or `sampler.args.group` arguments to `bootstrap`.

All samplers and their arguments are described in one help file, e.g. `help(samp.MonteCarlo)`.

Other arguments to bootstrap

There are a large number of optional arguments to `bootstrap`. Here is a short list of some that are often used. For a complete list, see the help files for `bootstrap` and `bootstrap.args`.

- `trace`: logical flag indicating whether to print a message indicating which set of replicates is currently being drawn.
- `seed`: the random number seed; set this if you want to be able to reproduce results later.
- `assign.frame1`: logical flag, should bootstrap data sets be assigned to frame 1 before calling the statistic? This is sometimes necessary to ensure that the correct copy of the data is used.
- `save.indices`: logical flag indicating whether to save the matrix of resampling indices. By default, the value of the random number seed used is saved, and the sampler used is specified in the call, which is enough information to reproduce the resampling indices in later analyses. The matrix of resampling indices may be saved as part of the object by setting `save.indices=T`. This matrix has dimension $n \times B$.

Components of the object

All `bootstrap` objects have components `call`, `observed`, `replicates`, `estimate`, `B`, `n`, `dim.obs`, `seed.start`, `seed.end`, and `parent.frame`. Components `B` and `n` are the number of replicates and the sample size, respectively. The `observed` component contains the observed parameter values calculated using the original data. The `estimate` component is a data frame containing bootstrap estimates of bias, mean, and standard error. The `replicates` are the B bootstrap replicates of the parameters. The `call` component stores the call used to create the object, which is used by downstream functions like `addSamples` or `jack.after.bootstrap` for modifying or computing

statistics on the bootstrap object. Component `parent.frame`, which contains the frame of the caller of `bootstrap`, is also used for this purpose. The starting random number seed `seed.start` and ending random number seed `seed.end` are also stored for future reference. If `statistic` returns a matrix, then its dimension is stored as `dim.obs` for use in the layout of plots. Other components which may be returned are `group`, `B.missing`, `indices`, `weights`, `L`, `Lstar`, `n.groups`, `original`, `statistic`, and `actual.calls`. See the help file for `bootstrap.object` for more details.

The Jackknife

In jackknife resampling, a statistic is calculated for the n possible samples of size $n-1$, each with one observation left out. The jackknife predates the bootstrap. It is faster than the bootstrap (if n is small), but ignores interactions and is suitable only for smooth statistics. The jackknife is useful for computing empirical influence values, which are used to obtain linear approximations to a statistic. See section **Influence and Linear Approximations**.

The jackknife is implemented by function `jackknife`.

```
> jack.obj <- jackknife(data = swiss.x["Education"],
+   statistic = var)
> jack.obj
Call:
jackknife(data = swiss.x[, "Education"], statistic = var)
```

```
Number of Replications: 47
```

```
Summary Statistics:
```

	Observed	Mean	Bias	SE
var	92.46	92.46	-1.961e-012	40.73

Jackknife estimates of bias, mean, and standard error are calculated differently than the equivalent bootstrap statistics.

The summary method for jackknife objects prints the same information as for bootstrap objects, except that only empirical percentiles are computed.

```
> summary(jack.obj)
Call:
jackknife(data = swiss.x[, "Education"], statistic = var)
```

Number of Replications: 47

Summary Statistics:

	Observed	Mean	Bias	SE
var	92.46	92.46	-1.961e-012	40.73

Empirical Percentiles:

	2.5%	5.0%	95.0%	97.5%
var	60.43	85.54	94.49	94.51

The plot method produces the same histogram and density estimate plots as for bootstrap objects (see Figure 1.3).

```
> plot(jack.obj)
```

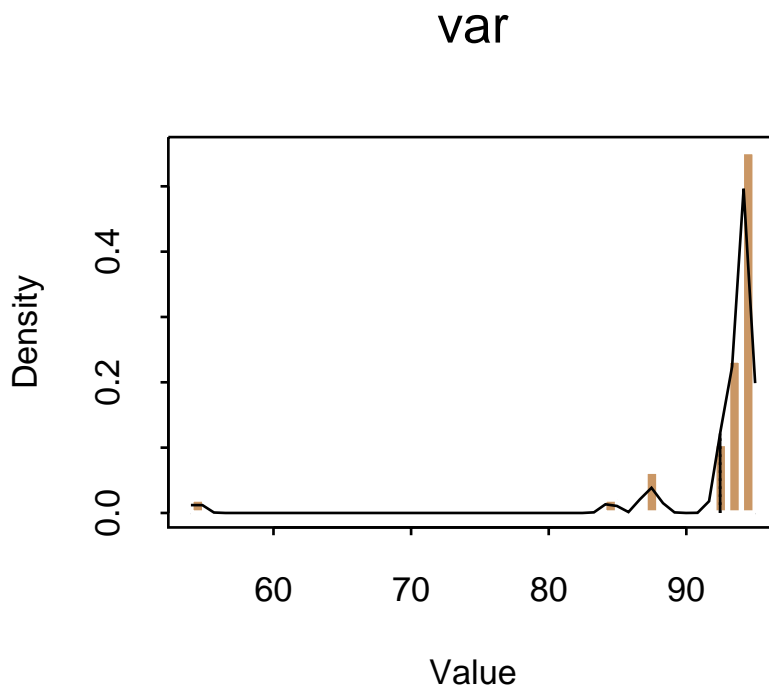


Figure 1.3: *Histogram of jackknifed variances.*

Note the effect of the outlier. That observation is in every jackknife sample except one, whose variance is therefore much smaller than for other samples.

Results are not random in the default “delete-1” case (unless there is randomization done by the statistic). More than one observation may be removed at a time using the `group.size` argument (see below), in which case random sampling is used.

Arguments

The `jackknife` function takes the arguments `data`, `statistic`, `args.stat`, and `assign.frame1`, which have the same meanings as for `bootstrap`. It also takes the `group` and `subject` arguments. The `subject` argument has the same meaning as for `bootstrap`: subjects (as defined by unique values of the subject vector), not observations, are deleted to form jackknife samples.

The `group` argument does not effect sampling, but does effect the calculation of standard error. Use `jackknife` with caution when sampling by group; deleting observations causes the proportions of observations in each group to vary across jackknife samples, which may make standard error and bias estimates invalid if this is inconsistent with how the data were created.

Other arguments are described in the `jackknife` help file.

Permutation Tests

Permutation tests, introduced by Fisher (1935), are used in hypothesis testing situations where a special symmetry exists under the null hypothesis so that it is possible to create a *permutation distribution* of the test statistic. For example, a classical two-sample problem has m observations from F and n from G . Under the null hypothesis that $F=G$, every one of the $\text{choose}(m+n, n)$ partitions of all observations into two groups is equally likely. The permutation test evaluates the statistic T for every partition (or a randomly-selected subset of them if $\text{choose}(m+n, n)$ is large) and obtains a p-value from the fraction of these T s that exceed the observed T .

Permutation tests are implemented by functions `permutationTest` and `permutationTestMeans`. The latter is specific to the case where the statistic is the difference of two sample means.

Example

A previous bootstrap example considered the difference in the average infant mortality rates between two populations based on education level, using the `swiss.x` dataset. Let’s treat this problem from a slightly different perspective using `permutationTestMeans`. Here the null hypothesis is that the the two populations have the same

distribution. That is, education has no effect on infant mortality. A permutation test is performed to test alternatives to the null hypothesis.

```
> IM <- swiss.x[,"Infant Mortality"]
> pt.obj0 <- permutationTestMeans(data = IM,
+   treatment = Edu > 20, alternative = "greater",
+   B = 1000, seed = 9)
> pt.obj0
Call:
permutationTestMeans(data = IM, treatment = Edu > 20, B =
1000,
  alternative = "greater", seed = 9)
```

Number of Replications: 1000

Summary Statistics:

	Observed	alternative	p.value
X1	838.6	greater	0.4336

The treatment vector, in this case a vector of T's and F's, defines the two samples, with T corresponding to the higher educated group. The alternative argument describes the alternative hypothesis to test: "less", "greater", or "two-sided". The "greater" option tests whether the difference $m1 - m2$ of the means (where $m1$ corresponds to the lowest level of treatment) is greater than zero. The "less" option tests for $m1 - m2 < 0$, and "two-sided" tests for $m1 - m2 \neq 0$. Since the lowest level of `Edu > 20` is F, we are testing the alternative hypothesis that the mean infant mortality rate is higher for the lower educated group.

The output `p.value` indicates that approximately 43.36% of replicates show a difference in means greater than that for the observed (this value is computed as $(k+1)/(B+1)$, where k is the number of replicates with difference greater than originally observed. If all infant mortality rates for the lower educated group had actually been higher than all of those for the higher educated group, the p-value would have been $1/(B+1)$, or nearly 0. A typical cut-off for p-values is .05, so we do not have strong evidence to accept the alternative hypothesis in this case.

A note about the `Observed` output component; this is the sum of the first group, rather than difference in means. Similarly for replicates. This is done for computational efficiency, and does not affect the p-value.

The relationship between the one-sided (`alternative = "less"` or "greater") and two-sided p-values is as follows. The p-value for "less" is 1 minus the p-value for "greater", and the p-value for "two-sided" is defined as twice the minimum of the one-sided p-values.

The `permutationTestMeans` function supports the `group` argument for stratified sampling. You might use this, for example, if you had male and female data for each state, and data should be permuted separately in each state. There are also arguments `combine`, `combineGroup`, and `combinationFunction` for combining p-values across groups and variables (data can be multivariate, in which case the differences of means and p-values are computed for each variable).

The `permutationTest` function provides more flexibility than `permutationTestMeans`. At heart it is a front-end to `bootstrap`, using `sampler = samp.permute` to generate the permutations. Here, for example, is one way to rework the above test using `permutationTest`.

```
> pt.obj <- permutationTest(data = IM,
+   statistic = mean(IM[Edu <= 20]) - mean(IM[Edu > 20]),
+   B = 1000, seed = 9, trace = F,
+   alternative = "greater")
> pt.obj
Call:
permutationTest(data = IM, statistic = mean(IM[Edu <= 20])
- mean(
  IM[Edu > 20]), B = 1000, alternative = "greater", seed =
9,
  trace = F)
```

```
Number of Replications: 1000
```

```
Summary Statistics:
```

	Observed	Mean	SE	alternative	p-value
Param	0.2267	0.03759	1.369	greater	0.4356

The difference in the p-values is small and due to the different mechanisms used by the two routines to do the permutations.

The arguments to `permutationTest` are `data`, `statistic`, `B`, `alternative`, `combine` and `combinationFunction`. In addition, any extra arguments (like `seed` and `trace`, above) are passed along to `bootstrap`.

For an example that doesn't use the difference of means, let's use correlation to further explore the relationship between the education and infant mortality variables. A scatter plot of the data (Figure 1.4) suggests a slight negative correlation, which we would like to test.

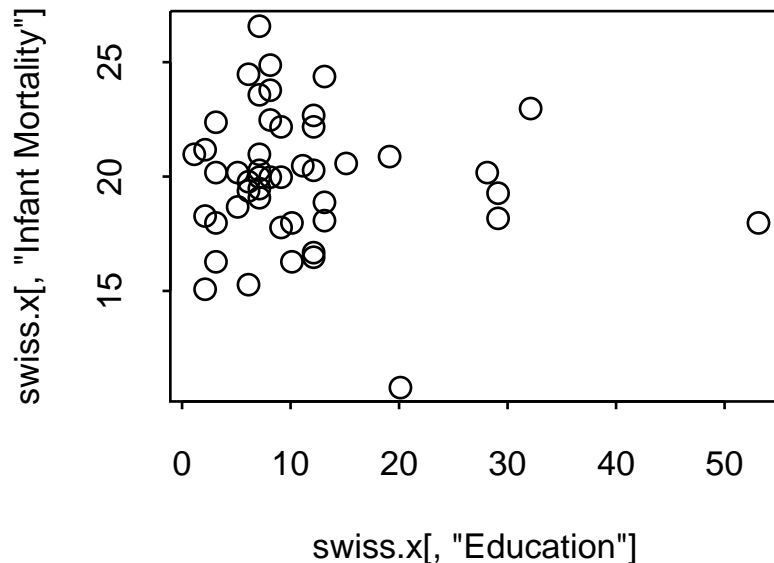


Figure 1.4: Scatter plot of infant mortality vs. education in the `swiss.x` dataset.

We use `alternative = "less"` to test the alternative hypothesis that the correlation is indeed negative.

```
> pt.obj1 <- permutationTest(data = swiss.x["Education"],
+   statistic = cor(data, swiss.x["Infant Mortality"]),
+   alternative = "less", seed = 0, trace = F)
> pt.obj1
Call:
permutationTest(data = swiss.x[, "Education"], statistic =
cor(data, swiss.x[, "Infant Mortality"]), alternative =
"less", seed = 0, trace = F)
```

```
Number of Replications: 999
```

```
Summary Statistics:
      Observed      Mean      SE alternative p-value
Param -0.09932 -0.002312 0.1522          less  0.259
```

The observed value of the correlation is -.099. The p-value of .259 indicates that approximately 25.9% of replicates had lower correlation. The summary method computes empirical percentiles for the replicates, just as it does for bootstrap.

```
> summary(pt.obj1)
Call:
permutationTest(data = swiss.x[, "Education"], statistic =
cor(data, swiss.x[, "Infant Mortality"]), alternative =
"less", seed = 0, trace = F)
```

```
Number of Replications: 999
```

```
Summary Statistics:
      Observed      Mean      SE alternative p-value
Param -0.09932 -0.002312 0.1522          less  0.259
```

```
Empirical Percentiles:
      2.5%      5%      95% 97.5%
Param -0.2969 -0.2629 0.2319 0.2913
```

These empirical percentiles should not be interpreted as confidence intervals.

The plot method for permutationTest and permutationTestMeans objects is the same as for other resamp objects (see Figure 1.5).

```
> plot(pt.obj)
```

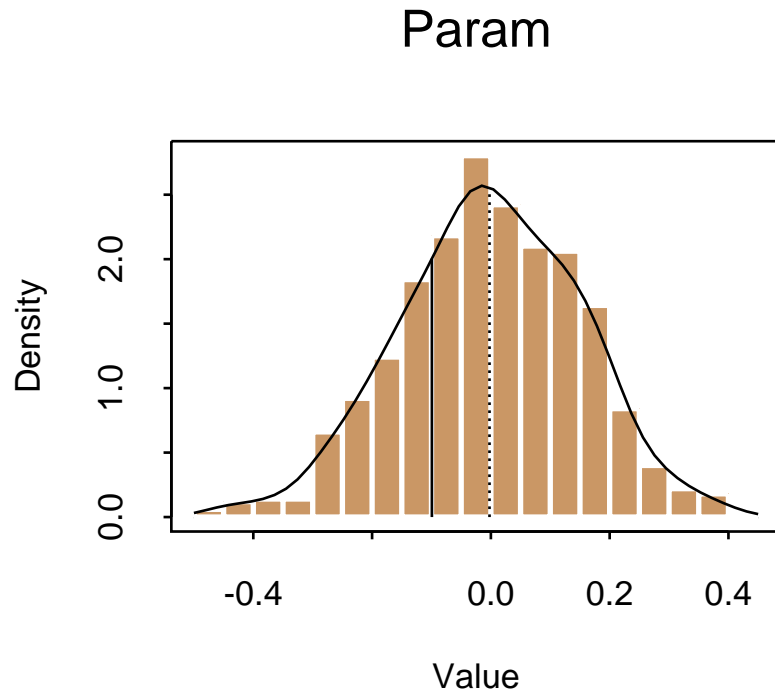


Figure 1.5: Histogram of permutation test replicates for correlation. Solid line is the observed value, dotted line is the mean of the replicates. 25.9% of the replicates lie to the left of the observed value.

CONFIDENCE LIMITS

After creating a bootstrap object, a number of confidence intervals may be calculated. Quick-and-dirty “bootstrap percentile” limits are calculated by `limits.emp`; this calculates percentiles of the bootstrap distribution. The errors in coverage for this procedure are typically $O(1/\sqrt{n})$, which decrease only slowly with the sample size n .

Bootstrap bias-corrected, accelerated (BCa) limits are calculated by `limits.bca`. Bootstrap tilting limits are calculated by `limits.tilt`. Bootstrap-t intervals are calculated by `bootstrapT`, provided that your bootstrap statistic calculated both estimates and standard errors. The errors for these procedures all typically decrease at the rate $O(1/n)$.

To reduce simulation error to reasonable levels, the number of bootstrap samples should be at least 1000 for `limits.emp` and `bootstrapT`, 2000 for `limits.emp` if z_0 is not supplied (see below) or 1000 if it is, and 60-100 for `limits.tilt`. These values may be reduced using variance-reduction procedures, control variates or importance sampling.

The `probs` argument to all functions indicates which confidence levels are computed. Linear interpolation is used if necessary to obtain the specified percentiles.

Another confidence interval procedure that requires no bootstrap sampling, approximate bootstrap confidence (ABC) intervals, is supported by function `limits.abc`. It is discussed in the Influence and Linear Approximations section.

Empirical Percentiles

The empirical percentiles are simply percentiles of the empirical bootstrap distribution; these are known as bootstrap percentile confidence intervals.

BCa Percentiles

The BCa method transforms the specified `prob` values based on bias and skewness, returning percentiles of the bootstrap distribution corresponding to these adjusted values.

To calculate BCa percentiles requires bias correction (denoted z_0) and acceleration parameters. If these values are not specified (and they usually are not), the bias correction is estimated from the replicates and acceleration is estimated using jackknifing. Note that

rather than doing a complete delete-1 jackknife, the data are broken into groups of size `group.size` and the groups are jackknifed. If `group.size` is not specified, it is calculated as `floor(n/20)`, which yields roughly 20 jackknife replicates depending on the magnitude of `n`.

Tilting Percentiles

Tilting intervals are those values for which a hypothesis test would not reject the null hypothesis, at the given confidence level. The hypothesis test, in turn, is based on bootstrap sampling from the null distribution -- finding a weighted empirical distribution (using maximum-likelihood or similar criteria) which satisfies the null hypothesis, then performing bootstrap sampling with unequal probabilities from that weighted distribution.

In practice, the bootstrap sampling is usually performed with equal probabilities, and results that would have been obtained for weighted sampling are estimated using importance sampling identities. This is computationally efficient, typically requiring 1/17 as many bootstrap samples for comparable simulation accuracy as for 95% bootstrap percentile intervals, or 1/37 as many as for BCa intervals if z_0 is estimated from the bootstrap distribution.

To perform the maximum likelihood or similar calculations requires a linear approximation to the statistic, which may be obtained using regression methods (if the number of bootstrap replications is sufficiently large) or jackknife or influence calculations (which require calculating n function evaluations). In either case, this may negate the computation advantage of these intervals. Linear approximations are discussed in the Influence and Linear Approximations section.

Bootstrap-t intervals

Bootstrap-t confidence limits are calculated using the `bootstrapT` function. We do not generally recommend these limits, as they often are excessively long due to uncertainty in standard error estimates, and also require that standard error estimates be calculated for each bootstrap sample, which may be expensive computationally.

MORE EXAMPLES

This section describes two examples. The first example resamples a correlation coefficient using bootstrap and jackknife, and introduces the jackknife after bootstrap. The second example shows how to test linear regression coefficients using the bootstrap and jackknife after bootstrap.

Resampling the Correlation Coefficient

This example uses the law school data from Efron and Tibshirani (1993, p. 9). Starting with 82 American law schools participating in a study of admission practices, they constructed a random sample of 15 schools. They then examined (p. 49) the correlation between LSAT score and GPA for the 1973 entering classes at these schools.

Traditionally, Fisher's transformation would be used to transform the correlation coefficient into a normally distributed variable on which normal-based inference would be used. This example uses resampling to obtain inferential quantities instead of employing Fisher's transformation.

First, the data are entered into Spotfire S+ and stored as a data frame.

```
> lsat <- c(576, 635, 558, 578, 666, 580, 555, 661, 651,  
+         605, 653, 575, 545, 572, 594)  
> gpa <- c(3.39, 3.30, 2.81, 3.03, 3.44, 3.07, 3.00, 3.43,  
+         3.36, 3.13, 3.12, 2.74, 2.76, 2.88, 2.96)  
> law.data <- data.frame(School = school, LSAT = lsat,  
+         GPA = gpa)
```

Next, we bootstrap the correlation and summarize the results.

```
> boot.obj2 <- bootstrap(law.data, cor(LSAT, GPA),  
+         B = 1000, seed = 0, trace = F)  
> summary(boot.obj2)  
Call:  
bootstrap(data = law.data, statistic = cor(LSAT, GPA),  
B = 1000, seed = 0, trace = F)
```

Number of Replications: 1000

Summary Statistics:

	Observed	Mean	Bias	SE
Param	0.7764	0.7676	-0.008768	0.1322

```

Empirical Percentiles:
      2.5%   5%   95%  97.5%
Param 0.4662 0.5222 0.9433 0.9601
BCa Percentiles:
      2.5%   5%   95%  97.5%
Param 0.3419 0.4476 0.9257 0.9389

```

The plot of the bootstrap object (Figure 1.6) reveals a distribution that is clearly skewed.

```
> plot(boot.obj2)
```

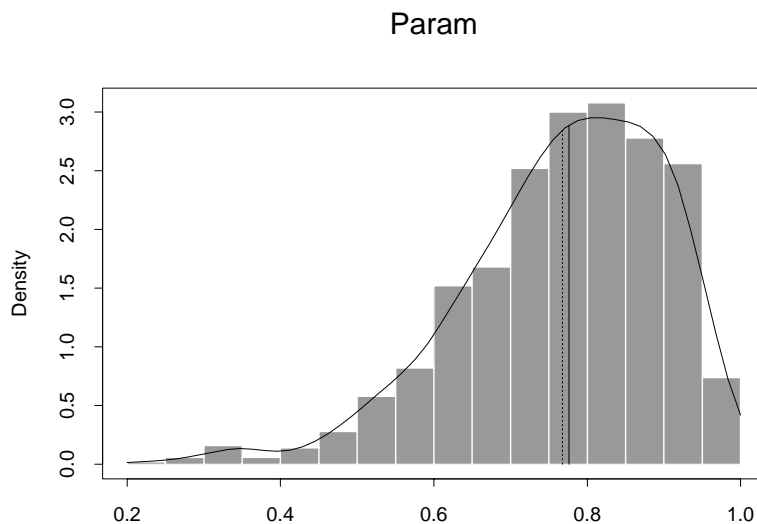


Figure 1.6: *Histogram of replicated correlations.*

Jackknife summary statistics for the correlation may be obtained also.

```

> jackknife(law.data, cor(LSAT, GPA))
Call:
jackknife(data = law.data, statistic = cor(LSAT, GPA))

```

```
Number of Replications: 15
```

```

Summary Statistics:
      Observed   Mean      Bias      SE
Param  0.7764 0.7759 -0.006474 0.1425

```

Jackknife after bootstrap

Another tool available for exploring the bootstrap object is the jackknife after bootstrap (Efron and Tibshirani, 1993, pp. 275-280). This technique provides standard error estimates for functionals of the bootstrap distribution. You may, for instance, want to know how good the bootstrap estimate -0.008768 of bias is from `boot.obj2`. The idea of obtaining such an estimate is to compute a separate bootstrap on the original data with each observation removed. There would be n such bootstraps, but jackknife after bootstrap uses a trick to simulate the same procedure using only the original bootstrap replicates already computed. In the process, information on the influence of each observation on the functionals is also computed.

Jackknife after bootstrap is implemented by function `jack.after.bootstrap`. By default, the functional is the mean of the distribution, although it is more commonly used for standard error or bias. We'll use `functional = bias` in this case, so that `jack.after.bootstrap` computes the standard error of the bootstrap bias estimates. The influence indicates the influence of each observation on the bias.

```
> jab.obj2 <- jack.after.bootstrap(boot.obj2,
+   functional = "bias")
> jab.obj2
Call:
jack.after.bootstrap(boot.obj = boot.obj2, functional =
"bias")
Functional Under Consideration:
[1] "bias"
Functional of Bootstrap Distribution of Parameters:
      Func SE.Func
Param -0.008768  0.0177
Observations with Large Influence on Functional:
$Param:
  Param
12 2.268
```

The `jack.after.bootstrap` object does not inherit from `resamp`. It contains components `call`, `functional`, `rel.influence`, `large.rel.influence`, `values.functional`, `dim.obs`, `threshold`, and `jabB`.

The `functional` component contains sub-components `Func` and `SE.Func`, which contain the observed bootstrap functional (which will match the functional value in the bootstrap object), and the jackknife after bootstrap estimate of standard error for that functional, respectively. In this case, the standard error estimate of .0177 is quite small, leading us to believe that our original bootstrap estimate of bias is pretty trustworthy.

The `values.functional` component contains the values of the functional for each of the n jackknife after bootstrap replicates. The `jabB` component lists the number of observations used from the original bootstrap replicates in each jackknife after bootstrap replicate.

Plotting the `jack.after.bootstrap` object displays the relative influence values (component `rel.influence`) using a plot similar to a Cook's distance plot (Figure 1.7). Influence values are discussed in detail in a later section. In general, however, observations with absolute relative influence greater than 2 are considered particularly influential.

```
> plot(jab.obj2)
```

The jackknife after bootstrap identifies observation 12 as being particularly influential (component `large.rel.influence`). A plot of GPA versus LSAT with this observation plotted as a triangle shows that this point is indeed an outlying observation (Figure 1.8). It is interesting to note that another observation (`law.data[1,]`), which appears to be even more of an outlier, does not have great influence on bias.

```
> plot(lsat[-12], gpa[-12], xlab = "LSAT", ylab = "GPA")  
> points(lsat[12], gpa[12], pch = 2)
```

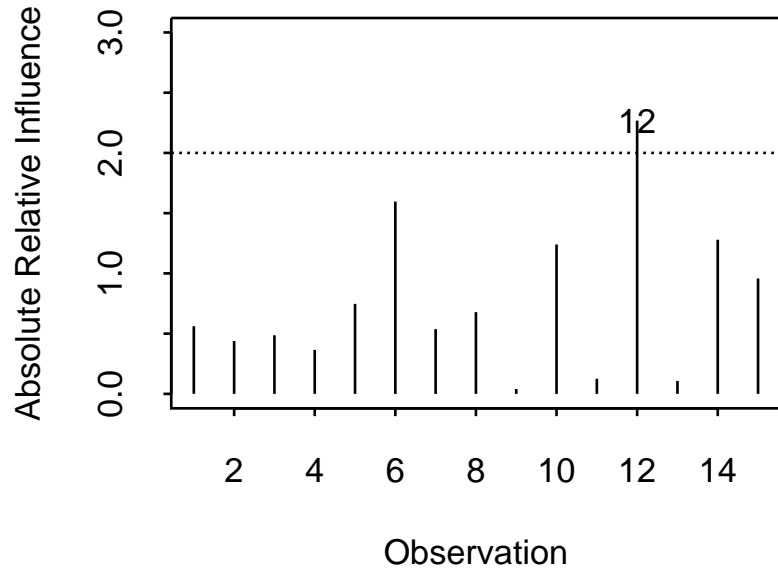


Figure 1.7: Influence plot for correlation.

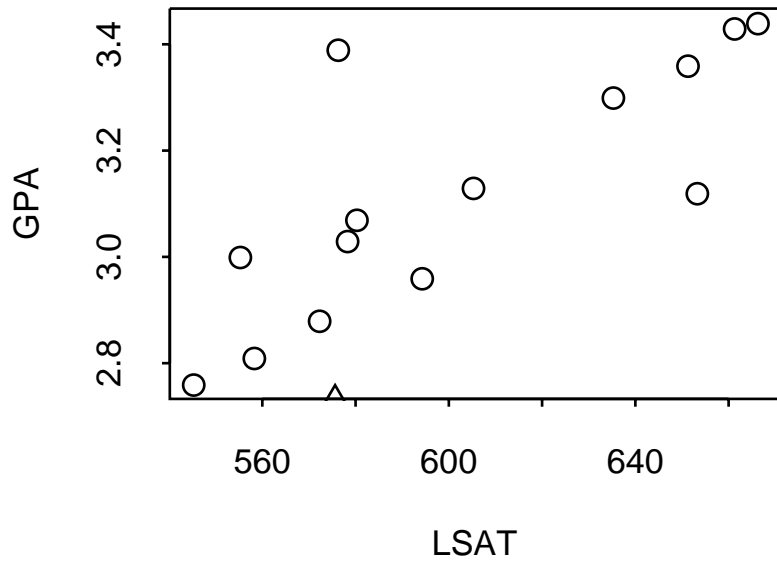


Figure 1.8: GPA versus LSAT.

Resampling Regression Coefficients

The last example shows how to test linear regression coefficients. The data are from operation of a plant for the oxidation of ammonia to nitric acid, measured on 21 consecutive days. See the Spotfire S+help file for `stack` for details.

First, the `stack.loss` vector and `stack.x` matrix are combined into a data frame.

```
> stack <- data.frame(stack.loss, stack.x)
> names(stack)

[1] "stack.loss" "Air.Flow"   "Water.Temp" "Acid.Conc."
```

We wish to bootstrap the vector of linear regression coefficients from the model of `stack.loss` regressed on `Air.Flow`, `Water.Temp`, and `Acid.Conc.`. There are two ways to perform this bootstrapping. The first uses typical bootstrap syntax, with the data frame as the `data` argument.

```
> boot.obj3 <- bootstrap(stack,
+   coef(lm(stack.loss ~ Air.Flow + Water.Temp +
+   Acid.Conc., stack)), B = 1000, seed = 0, trace = F)
```

The second way uses the fitted `lm` object as the `data` argument.

```
> boot.obj3 <- bootstrap(lm(stack.loss ~
+   Air.Flow + Water.Temp + Acid.Conc., stack),
+   coef, B = 1000, seed = 0, trace = F)
```

This method is much faster because it invokes a special `lm` method for `bootstrap`, `bootstrap.lm`, which performs only once the overhead that is required to create an `lm` object, as opposed to recalculating it for every resample, as in the first example. The results are equivalent.

```
> boot.obj3
Call:
bootstrap(data = lm(stack.loss ~ Air.Flow + Water.Temp +
Acid.Conc.,
  stack), statistic = coef, B = 1000, seed = 0, trace = F)
```

```
Number of Replications: 1000
```

```
Summary Statistics:
```

	Observed	Mean	Bias	SE
(Intercept)	-39.9197	-39.0905	0.829215	8.8239

```

Air.Flow    0.7156    0.7205    0.004886  0.1749
Water.Temp  1.2953    1.2639   -0.031415  0.4753
Acid.Conc.  -0.1521   -0.1573   -0.005164  0.1180

```

This is the first example we have seen of a vector statistic. The summary includes the correlation matrix for the replicate values and empirical and BCa percentiles.

```

> summary(boot.obj3)
Call:
bootstrap(data = lm(stack.loss ~ Air.Flow + Water.Temp +
Acid.Conc.,
  stack), statistic = coef, B = 1000, seed = 0, trace = F)

```

Number of Replications: 1000

Summary Statistics:

	Observed	Mean	Bias	SE
(Intercept)	-39.9197	-39.0905	0.829215	8.8239
Air.Flow	0.7156	0.7205	0.004886	0.1749
Water.Temp	1.2953	1.2639	-0.031415	0.4753
Acid.Conc.	-0.1521	-0.1573	-0.005164	0.1180

Empirical Percentiles:

	2.5%	5.0%	95.0%	97.5%
(Intercept)	-55.8067	-52.7793	-23.46083	-17.69383
Air.Flow	0.3842	0.4449	1.01537	1.05471
Water.Temp	0.3880	0.4765	2.05772	2.24383
Acid.Conc.	-0.4187	-0.3612	0.02224	0.06119

BCa Confidence Limits:

	2.5%	5%	95%	97.5%
(Intercept)	-58.9652	-54.5118	-25.372052	-21.06050
Air.Flow	0.3183	0.3859	0.987308	1.01966
Water.Temp	0.4965	0.5798	2.303938	2.48762
Acid.Conc.	-0.4283	-0.3759	0.009603	0.04748

Correlation of Replicates:

	(Intercept)	Air.Flow	Water.Temp	Acid.Conc.
(Intercept)	1.00000	-0.1376	0.03551	-0.7848
Air.Flow	-0.13760	1.0000	-0.79387	-0.1096
Water.Temp	0.03551	-0.7939	1.00000	-0.2007
Acid.Conc.	-0.78483	-0.1096	-0.20067	1.0000

Based on the 95% confidence limits, using either the empirical or the BCa percentiles, all coefficients except the `Acid.Conc.` coefficient are significantly different from zero.

The plot function provides histograms of the replicated regression coefficients (Figure 1.9). Skewness is particularly evident in the `Acid.Conc.` coefficients.

```
> plot(boot.obj3)
```

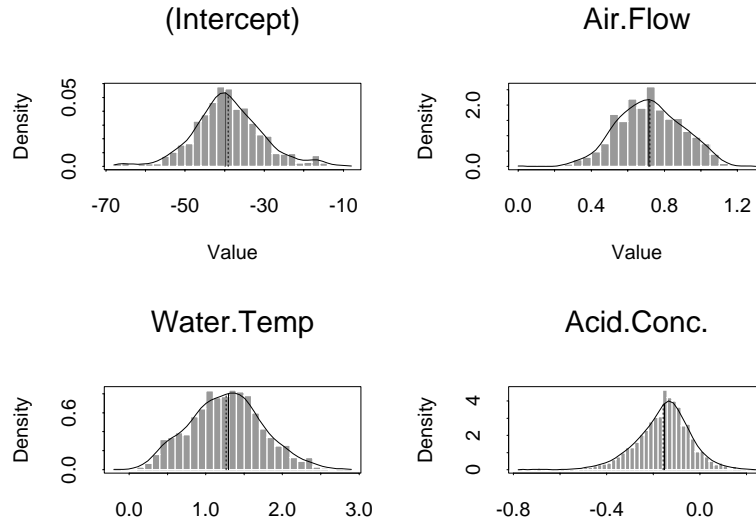


Figure 1.9: Histograms of replicated regression coefficients.

Next, the jackknife after bootstrap is used to assess the accuracy of the standard error estimates, and the influence of each observation on these estimates.

```
> jab.obj3 <- jack.after.bootstrap(boot.obj3, "SE")
> jab.obj3
Call:
jack.after.bootstrap(boot.obj = boot.obj3, functional =
"SE")
```

```
Functional Under Consideration:
[1] "SE"
```

```
Functional of Bootstrap Distribution of Parameters:
```

```
          Func SE.Func
(Intercept) 8.8239 3.67775
      Air.Flow 0.1749 0.06149
      Water.Temp 0.4753 0.17850
      Acid.Conc. 0.1180 0.05395
```

Observations with Large Influence on Functional:

```
$(Intercept)":
      (Intercept)
21          2.863
```

```
$(Air.Flow):
      Air.Flow
21      3.672
```

```
$(Water.Temp):
      Water.Temp
21          3.214
```

```
$(Acid.Conc.):
      Acid.Conc.
14          -2.184
21          2.589
```

The jackknife after bootstrap and the corresponding influence plot (Figure 1.10) suggest that points 14 and 21 are particularly influential.

```
> plot(jab.obj3)
```

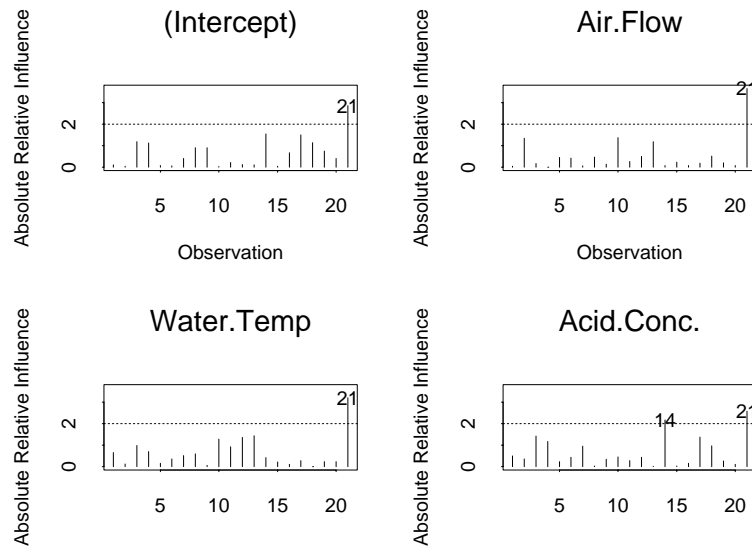


Figure 1.10: Influence plots for regression coefficients.

Other modeling function methods for bootstrap

The bootstrap function is generic, and other methods in addition to `bootstrap.lm` have been written to cut down on overhead when the statistic involves a modeling function like `lm`. Currently there are methods for `lm`, `glm` and `ensorReg`. In addition, bootstrap supports the syntax

```
> bootstrap(fitted.object, statistic, ...)
```

for any “model fit object”, defined as any object with a call component. Thus

```
> bootstrap(survReg(Surv(futime, fustat) ~ ecog.ps + rx,
+ data=ovarian, dist='lognormal'), coef, B = 250,
+ seed = 0)
```

is acceptable syntax, though it saves no execution time over the equivalent

```
> bootstrap(ovarian, coef(survReg(Surv(futime, fustat) ~
+ ecog.ps + rx, data=ovarian, dist='lognormal')),
+ B = 250, seed = 0)
```

A word of warning: the use of resample functions with modeling functions can sometimes lead to “scoping problems”, whereby the correct data cannot be found in the course of resampling. These problems are often signalled by a warning such as

```
Warning: bootstrap replicates are identical
```

or other unexplained results. A number of instances of scoping problems, along with their workarounds, have been summarized in the `resamp.problems` help file.

INFLUENCE AND LINEAR APPROXIMATIONS

Linear approximations for a statistic are required for many bootstrap computations, including variance reduction techniques and confidence interval procedures. They can also be used to approximate bootstrap replicates, using far fewer statistic evaluations.

Given a set of data (with n observations) and a statistic, the empirical influence is defined as the gradient of the statistic with respect to the probabilities assigned to the observations, with components

$$L_j = \lim_{\varepsilon \rightarrow 0} \frac{T(\mathbf{w}_0 + \varepsilon(e_j - \mathbf{w}_0)) - T(\mathbf{w}_0)}{\varepsilon} \quad (1.1)$$

where $\mathbf{w}_0 = \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ is the vector of equal probabilities, $T(\mathbf{w})$ is the value of the statistic calculated for the given data and the vector of probabilities $\mathbf{w} = (w_1, \dots, w_n)$, and e_j is the vector with 1 in position j and zero elsewhere. This can be used for a linear approximation to a statistic,

$$T(\mathbf{w}) \cong T(\mathbf{w}_0) + \sum_{j=1}^n w_j L_j \quad (1.2)$$

In the case of bootstrapping, if observation j is included in a bootstrap sample k times then

$$w_j = \frac{k}{n} \quad (1.3)$$

and the last term in (1.3) can be written as a mean of a bootstrap

sample of values of L_j , i.e. $\sum_{j=1}^n w_j L_j = \frac{1}{n} \sum_{j=1}^n L_{j^*}$, where j^* indicates

which original observation contributed the j th observation in a bootstrap sample.

The influence function

The influence function calculates approximations to the L_j using finite values of ε , which is specified by the `epsilon` argument. It takes many of the same arguments as `bootstrap`, including `data`, `statistic`, `args.stat`, `group`, and `subject`. The statistic must be a function, or an expression involving functions, that accept a `weights` argument. As an example, let's compute the influence values for the variance of the `Education` column of the `swiss.x` dataset.

```
> infl.obj <- influence(data = swiss.x[, "Education"],
+   statistic = var)
```

The default value for `epsilon` is `min(0.001, 1/n)`, where `n` is the number of observations in the data. The influence object inherits from `resamp`, and printing the object reveals the similarities.

```
> infl.obj
Call:
influence(data = swiss.x[, "Education"], statistic = var)
```

```
Number of Replications: 47
```

```
Summary Statistics:
```

	Observed	Mean	Bias	SE	acceleration	z0	cq
var	90.49	90.49	-1.925	38.54	0.1332	0.151	-0.0321

The seven summary statistics together form the estimate component of `infl.obj`. Here, `Mean` is the average of the replicates, which are the 47 values of the statistic evaluated at distance ε in each “direction”

(that is, the values of $T(\mathbf{w}_0 + \varepsilon(e_j - \mathbf{w}_0))$ for each j). `Bias`, `SE`, and `Observed` have the same meaning as for bootstrap objects.

Components `acceleration`, `z0`, and `cq` are used by other bootstrap procedures (see `limits.bca` for instance). The replicates themselves are stored in the `replicates` component. As for `resamp` objects, a plot of an influence object gives a histogram of the replicates along with a fitted density curve (Figure 1.11).

```
> plot(infl.obj)
```

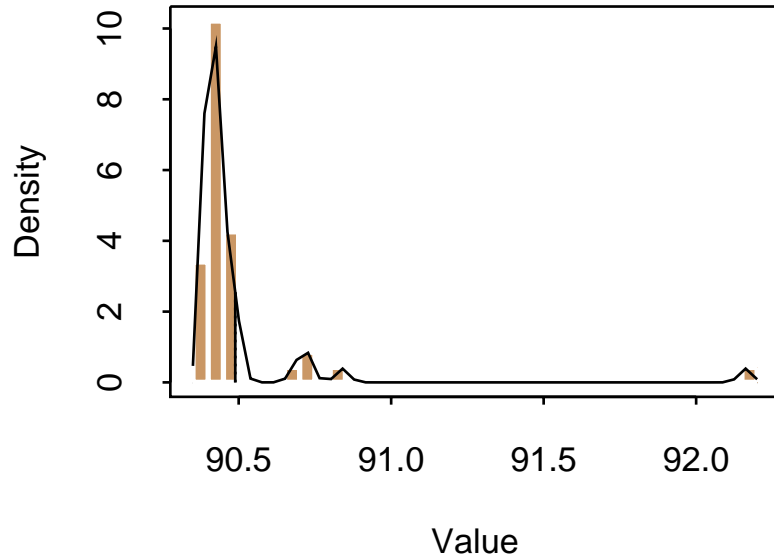


Figure 1.11: Histogram and density of empirical influence values for the variance of the Education column of the swiss.x dataset.

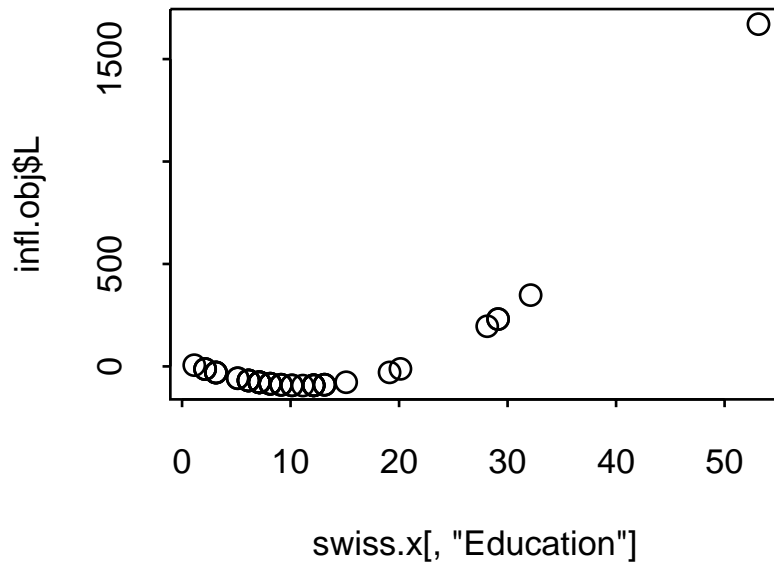


Figure 1.12: Empirical influence values for variance.

The influence values themselves occupy component `L`. A plot of the data versus the influence values reveals that, as one would expect, outliers have a particularly high influence on variance. See Figure 1.12.

```
> plot(swiss.x[, "Education"], infl.obj$L)
```

Special values of ε are notable; $\varepsilon = -\frac{1}{n-1}$ corresponds to the usual

(negative) jackknife, $\varepsilon = \frac{1}{n+1}$ corresponds to the positive jackknife

(Efron (1992)), and $\varepsilon = \frac{1}{\sqrt{n}}$ to the butcher knife (Hesterberg (1995)).

The finite- ε approximations to (1.1) are actually more useful than the derivatives would be. They may provide more accurate linear approximations, particularly using large ε for non-smooth statistics; the butcher knife, in particular, evaluates the statistic at a distance ε that matches the average distance for bootstrap samples (Hesterberg (1994)). Second, for smooth statistics the sum of the derivatives L_j is zero, but the sum of the finite-sample approximation values is nonzero, and provides an estimate of the trace of the Hessian of the statistic, which is used by `influence` for estimating the bias of the statistic. `influence` then normalizes the approximations for `L` to sum to zero.

For example, the following produce the same influence values, corresponding to the usual jackknife.

```
> jfit <- jackknife(swiss.x[, "Education"], var,
+   args.stat = list(unbiased = F))
> jack.L <- subtractMeans(-(jfit$n - 1) *
+   (jfit$replicates - jfit$observed))
> infl.L <- influence(swiss.x[, "Education"], var,
+   epsilon = -1/(jfit$n - 1))$L
> all.equal(jack.L, infl.L)
[1] T
```

We use `unbiased = F` as an argument to `var` in the call to `jackknife` in order to match the results from `influence`. This is because `influence` adds a `weights` argument to the statistic and for `var`, equal weights is equivalent to no weights and `unbiased = F`.

In addition to approximating the first derivatives (1.1), `influence` estimates the directional second-derivative of the statistic in the gradient direction. Subtracting that quantity from the bias estimate based on the trace of the Hessian yields an estimate of the bias of the statistic after a linearizing transformation is performed; this is a key quantity for computing bootstrap BCa and nonparametric ABC confidence limits (see below). Also needed for those intervals is “acceleration”; `influence` estimates this from the skewness of the L_j values.

The `influence` function can also calculate the influence from a vector of weights other than w_0 , and when sampling is by group or subject. This turns out to be surprisingly tricky, because of different ways that statistics may or may not normalize weights for each group or subject. Because of this uncertainty, additional quantities like directional second derivatives and acceleration are not calculated if sampling is by group or subject or if weights are other than w_0 .

Note that `influence` allows `statistic` to be either a function (which must accept a `weights` argument by that name) or an expression. In the latter case `influence` creates a modified version of the expression, adding a `weights` argument to every function used in the expression that accepts one. This symbolic programming is convenient for the user, but is not foolproof, because some functions used in the expression may accept weights under a different name. The output includes the modified expression in component `modifiedStatistic`, which you may inspect for correct behavior. `influence` also allows you to pass an appropriate modified expression instead of having it be constructed.

Approximating bootstrap replicates

Equation (1.2) can be used to approximate bootstrap replicates. These may be used for variance-reduction techniques such as control variates and concomitants, or even to replace bootstrap replicates if the statistic is expensive to compute. The weights given by equation (1.3) for bootstrap samples are based on the sampling indices, which we can generate directly using one of the sampler functions. For instance, the 500 sampling indices generated by `bootstrap` with `seed = 0` can be obtained by

```
> set.seed(0)
> inds <- samp.MonteCarlo(47, B = 500)
```

Then

```
> approxT <- infl.obj$observed + bootstrapMeans(infl.obj$L,
+       inds)
```

calculates (1.2) for all 1000 samples. For comparison, the actual bootstrap replicates can be generated using

```
> boot.obj <- bootstrap(swiss.x[, "Education"],
+       statistic = var, seed = 0, B = 500,
+       args.stat = list(unbiased = F))
```

We use `unbiased = F` for the same reason we did with `jackknife` above. Figure 1.13 plots the accuracy of the results, using the following.

```
> plot(approxT, boot.obj$replicates)
> abline(0,1, col = 2)
> points(boot.obj$observed, pch = "O")
```

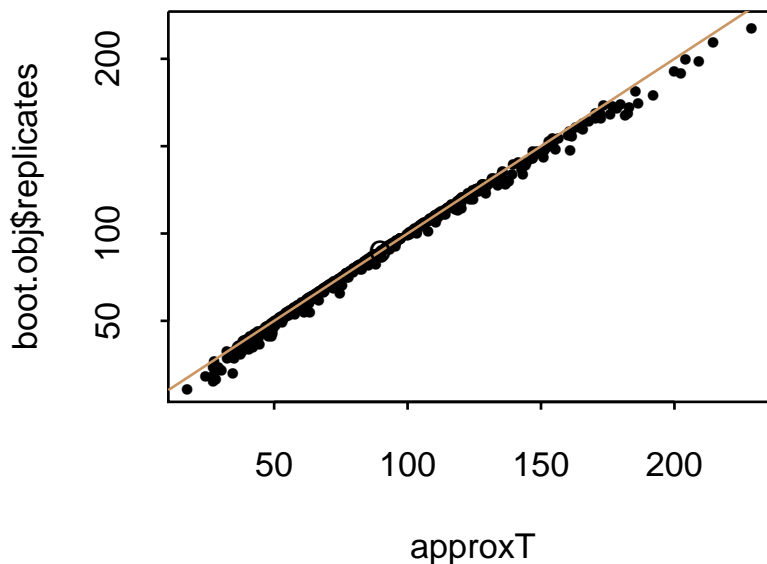


Figure 1.13: *Bootstrap replicates versus their linear approximations by influence values. The solid line is $y=x$. The observed value is marked with an 'O'.*

The results look quite good, and this is verified by the high value of correlation.

```
> cor(approxT, boot.obj$replicates)
```

```
var
[1,] 0.9977607
```

The `limits.abc` function

Another use for influence values is to calculate confidence intervals, without bootstrap sampling. `limits.abc` calculates approximate bootstrap confidence (ABC) intervals using influence values, a bias term obtained while calculating the influence values, and one additional bias term.

```
> limits.abc(data = swiss.x["Education"],
+   statistic = var)
Warning messages:
  Negative weights encountered, changed to zero, interval
  may be too
  narrow in: limits.abc(data = swiss.x["Education"],
  statistic
  = var)
Call:
limits.abc(data = swiss.x["Education"], statistic = var)

Number of Replications: 47

Summary Statistics:
      Observed Mean   Bias   SE acceleration   z0      cq
var    90.49 90.49 -1.925 38.57      0.1332 0.151 -0.03208

ABC confidence limits:
      2.5%      5%      95%      97.5%
var 46.78858 49.06474 198.2585 227.4245

ABC/exponential confidence limits:
      2.5%      5%      95%      97.5%
var 43.06846 48.82804 199.6258 230.1081
```

There are two sets of limits, ABC and ABC/exponential. The warning message indicates that during the computation of influence values, negative weights were created and subsequently set to zero. The ABC/exponential limits are calculated using “exponential tilting”, a technique that always results in positive weights, and therefore may give better results. This function requires a total of $n+5$ (52 in this case) function evaluations to calculate a single confidence interval: 1 for the observed statistic, n to estimate the gradient and the trace of

the Hessian (the first bias term), 2 to estimate the directional second derivative (a second bias term), and 2 to finally obtain the endpoints of the confidence interval. This is in contrast with the 1000 replicates recommended for bootstrap confidence intervals.

The resampGetL function

We used influence values above to estimate bootstrap replicates. Conversely, bootstrap replicates can be used to approximate influence values. This is one use of the `resampGetL` function, which calculates influence values from bootstrap, jackknife, influence, or other `resamp` objects using a variety of methods. For example,

```
> boot.L <- resampGetL(boot.obj)
```

is another estimate for the L_j , based on a linear regression of the replicates in `boot.obj`. Figure 1.14 compares these with the results from `influence`.

```
> plot(swiss.x[, "Education"], boot.L, pch = "+")
> points(swiss.x[, "Education"], infl.obj$L, pch = "o",
        col = 2)
```

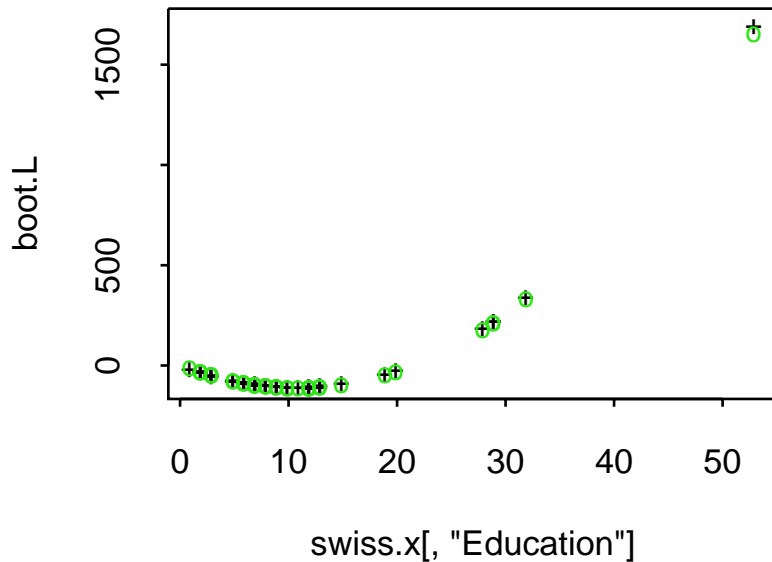


Figure 1.14: Influence values approximated by `resampGetL` using bootstrap replicates ('+'s) and by `influence` ('o's).

The method used by `resampGetL` is specified by the `method` argument, whose default value is determined by the first argument, `x`. In particular, if `L` is already present in `x` (as it would be if `x` were an `influence` object) it is merely extracted, not recomputed.

Possible values for `method` are “influence”, “jackknife”, “ace”, and “regression”. Methods “jackknife” and “influence” are based on calls to those functions, while “regression” and “ace” perform linear regressions with bootstrap replicates as the response variable (see Efron (1990), Hesterberg (1994), Hesterberg and Ellis (1999)), optionally after transforming the response (see Breiman and Friedman (1985)). The “ace” method, used in the above example, is the default for bootstrap objects when $n > 30$ and $B > 2n+100$.

The function returns a vector of `L` values. For the regression methods the vector has a “correlation” attribute giving the multiple correlation between (transformed) bootstrap replicates and the linear approximation.

```
> attr(boot.L, "correlation")
      [,1]
var 0.9985422
```


PARAMETRIC AND SMOOTHED BOOTSTRAPPING

This section discusses the related procedures of parametric and smoothed bootstrapping. In bootstrap algorithms, resampling is used to gain information about the underlying distribution from which a data set was sampled. (Specifically we assume that the data is comprised of n independent samples drawn from an unknown probability distribution F) But you may have *a priori* knowledge about the form of F , or wish to test bootstrap results against a parametric model for F . In this case, parametric bootstrapping can be used, in which resampling is done not from the original data, but from a parametric model of your choice. This is implemented by function `pbootstrap`.

In certain problems, a non-parametric framework may be desirable, except that a *smoothed* estimate of the distribution F may be more appropriate. Such problems include cases in which the effects of having a discrete distribution are large, such as situations involving sample quantiles of the distribution. One method of implementing such “smoothed” sampling is to sample the data with replacement as usual and add to it an independently generated smoothing variate (default: normal); this is equivalent to kernel smoothing. See Davison & Hinkley (1997) for further details. This is the implementation of `sbootstrap`, in which the user can specify both the smoothing function and the nonparametric sampling mechanism.

The parametric bootstrap

Here’s an example illustrating parametric bootstrapping. A histogram of the education data (Figure 1.15) suggests that it may follow a gamma distribution. We can estimate the shape and rate parameters defining the gamma distribution from the original data using a maximum likelihood estimate, for example. Here we use a simple method of moments estimate.

```
> Edu <- swiss.x[,"Education"]
> rate <- mean(Edu)/var(Edu)
> shape <- mean(Edu)*rate
```

The resulting density is superimposed on the histogram (normalized to look like a probability distribution) using the following.

```
> hist(Edu, xlim = c(0,70), ylim = c(0,.07), prob = T)
> x <- qgamma(ppoints(100), shape = shape, rate = rate)
```

```
> lines(x, dgamma(x, shape = shape, rate = rate))
```

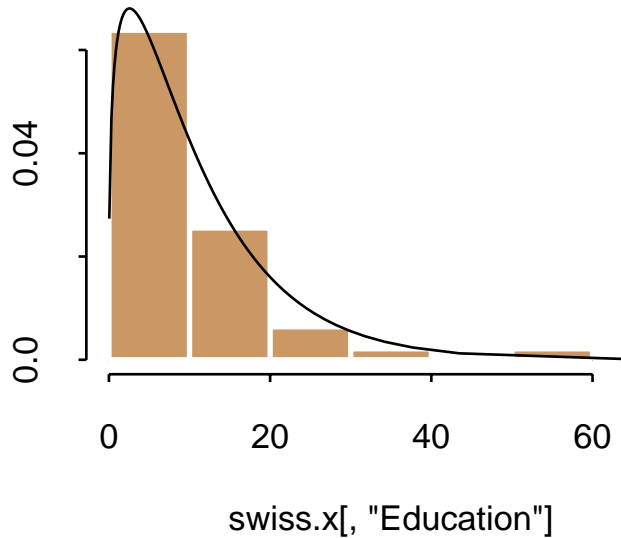


Figure 1.15: Histogram of the education data with a gamma distribution model superimposed.

Here's how we might use `pbootstrap` to bootstrap variances as we did in the Basic Resampling Techniques section, this time sampling from the gamma distribution above.

```
> pboot.obj <- pbootstrap(Edu, var, rsampler = rgamma,
+   args.rsampler = list(shape = shape, rate = rate),
+   seed = 0, B = 1000)
```

The `rsampler` argument specifies a function to return random values from the parametric model. Here we have used the built-in function `rgamma`, but you can also write your own. The first argument to this sampler function should be the number of values to return (it must be named `n`). The `args.sampler` argument is used to pass other arguments to the sampler. Note that the `data` argument is only used once – to calculate the observed value of the statistic, against which the statistic for the bootstrap replicates can be compared. Most other arguments to `pbootstrap` are the same as those to `bootstrap`. One notable exception: `pbootstrap` does not accept `group` and `subject` arguments.

The results from `pbootstrap` can be treated like any other resamp object.

```
> summary(pboot.obj)
Call:
pbootstrap(data = Edu, statistic = var, rsampler = rgamma,
  B = 1000,
  args.rsampler = list(shape = shape, rate = rate), seed =
  0)
```

```
Number of Replications: 1000
```

```
Summary Statistics:
```

	Observed	Bias	Mean	SE
var	92.46	-0.4088	92.05	34.43

```
Empirical Percentiles:
```

	2.5%	5.0%	95.0%	97.5%
var	42.51	46.32	153.5	172.9

Note that, in contrast with bootstrap objects, the `summary` function for `pbootstrap` objects computes only empirical percentiles. The `plot` method generates a histogram of the replicates (Figure 1.16).

```
> plot(pboot.obj)
```

As with bootstrap objects, the observed and mean values of the statistic are plotted with vertical solid and dotted lines, respectively, although here they are so close as to appear indistinguishable. Note the favorable comparison with the straight bootstrap results (see Figure 1.1 and the preceding summary of the Basic Resampling Techniques section).

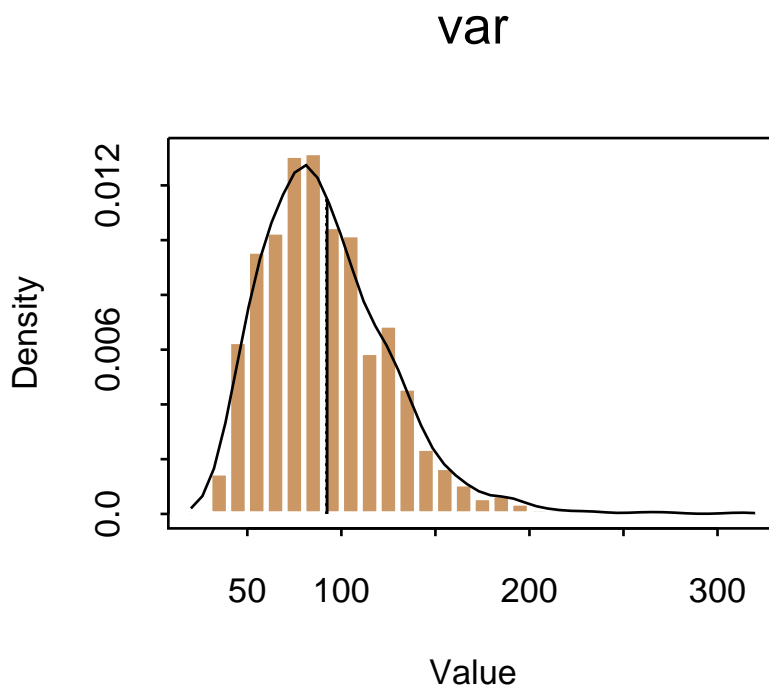


Figure 1.16: *Replicates from parametric bootstrapping using `rsampler = rgamma`.*

The smoothed bootstrap

Now let's use `sbootstrap` to try smoothed bootstrapping on the same example.

```
> sboot.obj <- sbootstrap(Edu, statistic = var, seed = 0)
> summary(sboot.obj)
Call:
sbootstrap(data = Edu, statistic = var, seed = 0)
```

Number of Replications: 1000

Summary Statistics:

	Observed	Bias	Mean	SE
var	92.46	-2.103	90.35	35.98

Empirical Percentiles:

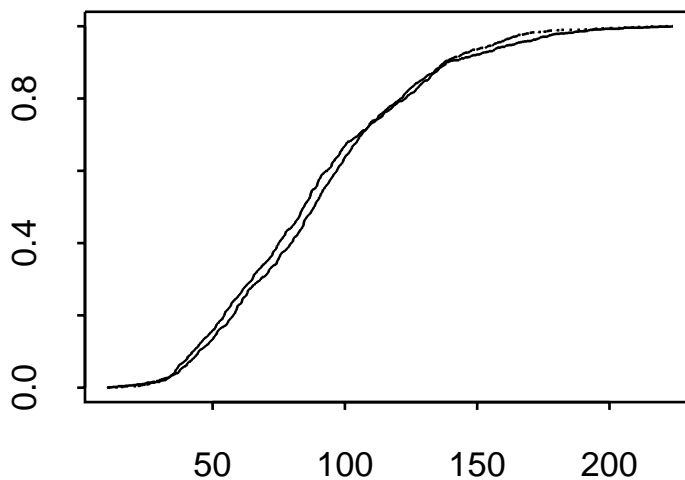
	2.5%	5.0%	95.0%	97.5%
var	31.59	37.91	156.4	165.4

Again, only empirical limits are computed by `summary` for `sbootstrap` objects. The function `sbootstrap` actually calls `pbootstrap`. The sampling function to be passed to `pbootstrap` is constructed by `sbootstrap` as follows. First, you must specify both a smoothing function (argument `smoother`) and a (nonparametric) sampling mechanism (argument `sampler`). (The defaults are `rmvnorm` and `samp.MonteCarlo`, respectively.) The "smoothed" sampling is a combination of these two elements: the nonparametric sampling mechanism chooses one of the n observations, and the smoother adds smoothed noise to it. The function that performs this "smoothed" sampling is passed to `pbootstrap` as its `rsampler` argument. Thus the smoothed bootstrap takes from the nonparametric bootstrap the idea of using the original data to approximate F , and adds smoothing, in order to sample from a continuous distribution, rather than a discrete one.

After the algorithm draws B bootstrap samples of size n (as stated above), it evaluates the statistic of interest (argument `statistic`) on each of the bootstrap samples.

We can compare the cumulative distribution function of smoothed bootstrap replicates to that of the simple nonparametric bootstrap replicates (see Figure 1.17).

```
> plotCDF(boot.obj$replicates)
> plotCDF(sboot.obj$replicates, new=F, col=2)
```



dotted line is cdf of sboot.reps

Figure 1.17: *Empirical distribution functions of replicates from bootstrap and sbootstrap.*

One might expect the curve representing the CDF of the smoothed bootstrap replicates to be smoother than the CDF of the replicates from the (non-parametric) bootstrap. However, in practice even nonparametric bootstrap distributions are practically continuous, unless the data are discrete or the sample size n is very small.

PARAMETRIC BOOTSTRAP TESTING

Hypothesis testing was introduced earlier using permutation tests (functions `permutationTest` and `permutationTestMeans`). Another type of hypothesis testing which uses resampling is *parametric bootstrap testing*. Given n data points, the algorithm generates samples from a parametric model, with at least one parameter constrained by the null hypothesis. Specifically, the algorithm repeatedly generates samples of size n and calculates the user-specified test statistic on the sample. The p-value is computed as the fraction of bootstrap samples in which the bootstrap sample test statistics are as extreme or more extreme than the test statistic calculated on the originally observed sample. See Davison & Hinkley (1997) for further details.

Parametric bootstrap testing is implemented by function `pbootTest`, which calls `pbootstrap` to do the parametric sampling.

To illustrate, recall the example from the previous section in which we used the mean and the variance of the `swiss.x` Education data to determine the shape and rate parameters for a gamma distribution model. We'll use parametric bootstrap testing to further gauge the appropriateness of this model. A gamma distribution's rate and shape parameters can be defined in terms of any two of the mean, variance and skewness of the distribution, from the relationships

$$\begin{aligned}\text{mean} &= \frac{\text{shape}}{\text{rate}} \\ \text{variance} &= \frac{\text{shape}}{\text{rate}^2} \\ \text{skewness} &= \frac{2}{\sqrt{\text{shape}}}\end{aligned}\tag{1.4}$$

In the following, `pbootTest` tests how often we might expect the skewness of randomly chosen observations from the parametric model we chose to match the skewness of the Education data.

```
> pbt.obj <- pbootTest(Edu, statistic = skewness,
+   rsampler = rgamma,
+   args.rsampler = list(rate = mean(Edu)/var(Edu),
+   shape = mean(Edu)^2/var(Edu)),
```

```

+     null.value="skewness = 2.4", alternative = "greater",
+     trace = F, seed = 0)
> pbt.obj
Call:
pbootTest(data = Edu, statistic = skewness, rsampler =
rgamma,
  args.rsampler = list(rate = mean(Edu)/var(Edu), shape =
mean(
  Edu)^2/var(Edu)), null.value = "skewness = 2.4",
alternative =
  "greater", trace = F, seed = 0)

```

Number of Replications: 999

In the following table, each row contains the summary statistics

for the given test statistic:

	Observed	null.value	alternative	p.value
skewness	2.421	skewness = 2.4	greater	0.078

Only 7.8% of randomly chosen samples had skewness greater than the skewness of the Education data. This might lead us to question, or at least not rely too much on, our choice for the model.

The arguments to `pbootTest` are `data`, `statistic`, `rsampler`, `B`, `args.stat`, `args.rsampler`, `null.value`, and `alternative`, most of which match, and have the same meaning as, those arguments to `pbootstrap`. In fact, any additional arguments to `pbootstrap` may be passed on to that function via the “...” argument. The parameters of the distribution must be specified and passed via `args.rsampler`. Typically, at least one parameter is determined by the null hypothesis, and other parameters may be estimated from the data. For example, in the normal case with the null hypothesis that the mean is zero, `args.rsampler` could be `list(mean = 0, sd = stdev(mydata))`; the mean is set by the null hypothesis while the standard deviation is estimated from the data. In the above example, however, the null hypothesis concerns a parameter (skewness) that is not an argument to the sampling function.

The optional argument `null.value` is available for the user to specify the null hypothesis, but is only currently used in the `print` method. `null.value` is a vector of parameter names and the corresponding null values. `null.value` is optional because the null hypothesis is entirely specified by the combination of `rsampler` and `args.rsampler`.

A sampling issue is that simulation is performed with all null hypothesis constraints enforced simultaneously, so it is up to the user to avoid asking for impossible simulations. For example, it is not possible to simulate under the conditions `mean = 0` and `mean = 1` at the same time, so make two separate calls to `pbootTest` if you wish to test both of these hypotheses. However, testing the composite hypothesis for a normal distribution that the `mean = 0` and `variance = 4` is allowed, as the simulation may be performed under both constraints.

The function `pbootTest` returns a `pbootTest` object, which inherits from `resamp`. The `pbootTest` object has the same components as a `pbootstrap` object, except that the `estimate` component (which is a data frame with columns `Bias`, `Mean`, and `SE` in a `pbootstrap` object) is a data frame with columns `alternative`, `p.value` and an optional column `null.value`. The columns `alternative` and `null.value` are as specified through the argument list, while `p.value` is calculated as described above.

Another example

Here is an artificial example illustrating the use of `pbootTest` with a vector statistic.

```
> set.seed(13) # to duplicate results of calling rmvnorm
> NN <- rmvnorm(125, d = 4)
> pbt.obj2 <- pbootTest(NN, colMeans, rmvnorm,
+   args.rsampler = list(mean = rep(0.2, 4),
+   sd = colStdevs(NN)), null.value = c("mean1 = 0.2",
+   mean2 = 0.2", "mean3 = 0.2", "mean4 = 0.2"),
+   trace = F)
> pbt.obj2
Call:
pbootTest(data = NN, statistic = colMeans, rsampler =
rmvnorm,
  args.rsampler = list(mean = rep(0.2, 4), sd =
colStdevs(NN)),
  null.value = c("mean1 = 0.2", "mean2 = 0.2", "mean3 =
0.2",
"mean4 = 0.2"), trace = F)
```

```
Number of Replications: 999
```

In the following table, each row contains the summary statistics

for the given test statistic:

```
Observed null.value alternative p.value
```

```
colMeans1  0.15546 mean1 = 0.2    two.sided  0.554
colMeans2 -0.01352 mean2 = 0.2    two.sided  0.012
colMeans3 -0.02964 mean3 = 0.2    two.sided  0.004
colMeans4 -0.08387 mean4 = 0.2    two.sided  0.004
```

The `null.value` can also be entered as a numeric vector, with or without names. Here is the result without names. (Note that the p-values are the same as we set the seed to the same initial value as above.)

```
> pbt.obj3 <- pbootTest(NN, colMeans, rmvnorm,
+   args.rsampler = list(mean = rep(0.2, 4),
+   sd = colStdevs(NN)), null.value = rep(0.2, 4),
+   trace = F, seed = pbt.obj2$seed.start)
> pbt.obj3
Call:
pbootTest(data = NN, statistic = colMeans,
  rsampler = rmvnorm, args.rsampler = list(mean = rep(0.2,
  4), sd = colStdevs(NN)), null.value = rep(0.2, 4),
  trace = F, seed = pbt.obj2$seed.start)
```

Number of Replications: 999

In the following table, each row contains the summary statistics for the given test statistic:

	Observed	null.value	alternative	p.value
colMeans1	0.15546	0.2	two.sided	0.554
colMeans2	-0.01352	0.2	two.sided	0.012
colMeans3	-0.02964	0.2	two.sided	0.004
colMeans4	-0.08387	0.2	two.sided	0.004

```
plot(pbt.obj3)
```

Note that no correction for multiple comparisons is made by the algorithm; these corrections, if necessary, may be done by the user.

The plot method for `pbootTest` objects displays a histogram of the replicates for each column of the statistic (see Figure 1.18), with observed values marked by dark vertical lines (difficult to see for `colMeans3` and `colMeans4`, due to their very small p-values).

```
> plot(pbt.obj3)
```

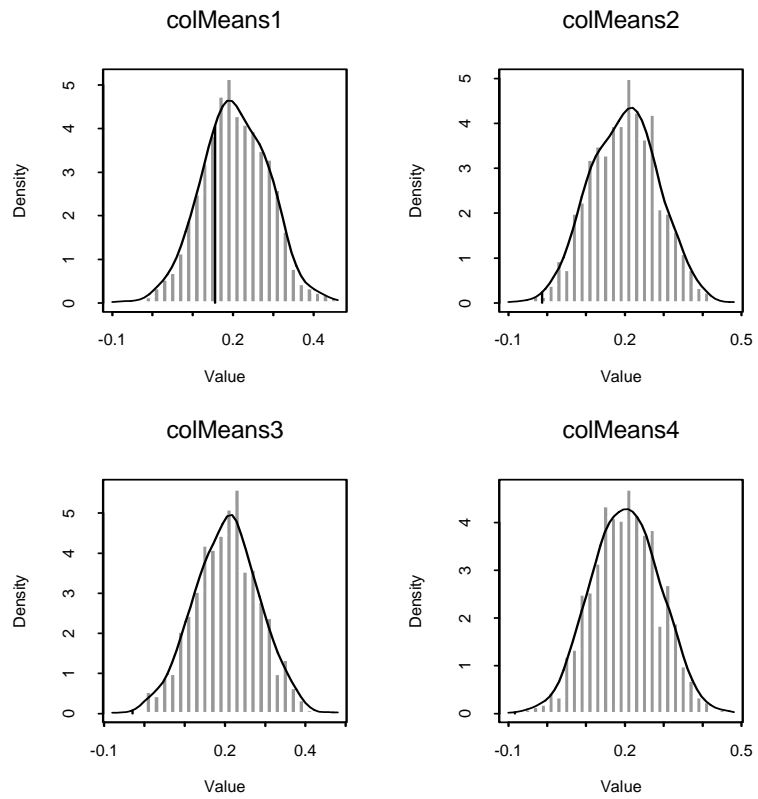


Figure 1.18: Plot of `pbootTest` object `pbt.obj3`.

ESTIMATING PREDICTION ERROR

Prediction error measures how accurately a model predicts the response of future data. In practice, prediction error is often used in model selection. When several related models are under consideration (different sets of variables in a linear model, or different levels of smoothing, for example), the model with the lowest prediction error is often chosen as the final model. Getting a good estimate of prediction error is not trivial. The naive procedure – simply considering the sum of squared residuals (the residual is the difference between the observed response and the predicted value) – leads to over-optimistic estimates of prediction error, because the same data is used to test the model as was used to fit the model.

This section considers two methods of estimating prediction error: K-fold cross-validation and bootstrap. K-fold cross-validation, implemented by function `crossVal`, leaves out a fraction of the observations, fits a model with the remaining observations, and forms predictions for the observations that were left out. Specifically, the data is split into K approximately equal subsets. For each of the K subsets, a model is fit using the other (K-1) subsets as training data to predict each of the points in the left-out subset. The predictions, in combination with an error function, are used to calculate an overall measure of prediction error.

There are actually several bootstrap estimates of prediction error. The function `bootPred` implements two of them – the `.632` and the `.632+` estimators. The algorithm does the following, B times: (1) draw a bootstrap sample from the data, (2) fit the model in question on this bootstrap sample, (3) use a prediction function to form predictions, both on the original sample and on the bootstrap sample. How these predictions are used to calculate prediction error depends on whether the `.632` or the `.632+` estimator is used. See Efron and Tibshirani (1997 and 1993) for further details. An important distinction is that while the cross-validation algorithm makes one prediction for each point in the original data set, this algorithm essentially makes B predictions for each point.

Both `crossVal` and `bootPred` require three functions as arguments: `modelFit` specifies how to fit the predictive model, `predFun` specifies how to predict, given a fitted model and a data set, and `errFun` specifies how to compute the error, based on the fitted values and the original data.

Both functions are generic functions, which dispatch according to whether the function given in `modelFit` has a formula argument. If so, the formula will be the first argument to `crossVal` or `bootPred`, and the generic implementation will dispatch on it, calling `crossVal.formula` or `bootPred.formula`, respectively. If there is no formula - the data is a matrix, for example - a default method is called. The default prediction function in both `crossVal` and `bootPred` is `predict`. There is random selection performed in both functions - in `crossVal` during the division of the data into K groups; and in `bootPred` during bootstrap sampling. Both functions therefore accept a `seed` argument for reproducibility.

Cross-validation: the `crossVal` function

The main arguments to `crossVal` are K , `modelFit`, `predFun`, and `errFun`. Argument K , the number of subsets, defaults to the sample size, which gives leave-one-out cross-validation. The default error function is mean squared error. In the following example we use `crossVal` to determine the “optimum” number of degrees of freedom to use with `smooth.spline` in fitting B-splines to ozone and temperature data from the `air` dataset, which has 111 observations that we divide into 10 roughly equal subsets.

```
> attach(air)
> tempErr <- rep(NA, 11)
> for(i in 1:11){
+   res <- crossVal(ozone, temperature, smooth.spline,
+   args.modelFit = list(df = i+1),
+   predFun = function(object, newdata)
+     predict(object, x = newdata)$y,
+   K = 10, seed = i)
+   tempErr[i] <- res$error
+ }
> tempErr
[1] 39.87933 39.12760 38.10327 36.84782 37.20555 36.86167
37.12167
[8] 37.57240 41.67375 38.53788 38.27774
```

```
> argminErr <- which(tempErr == min(tempErr))[1] + 1
> argminErr
[1] 5
```

Note that variability in the estimates is ignored: we simply choose the result with the smallest error. Five degrees of freedom is the optimal choice, although the values in `tempErr` reveal that it's a pretty close call between 5 through 8. In fact, changing the value of `seed` will sometimes change the optimum choice in this case.

The data and the five degree of freedom B-spline are plotted in Figure 1.19.

```
> plot(ozone,temperature)
> lines(smooth.spline(ozone,temperature, df = argminErr),
+       lty = 1)
```

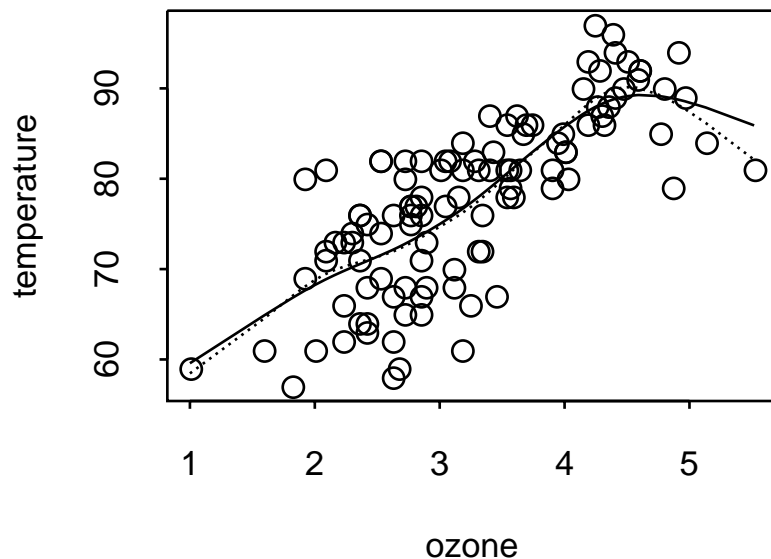


Figure 1.19: *B-spline fit of ozone and temperature data with $df = 5$ (solid curve) and $df = 7$ (dashed curve).*

The sampling itself is performed by the utility function `balancedSample`, which randomly samples in a way that ensures that no group has more than one more observation than any other group. Then, for the i -th subset, the model is fit using the function given in

`modelFit` on all of the subsets but the i -th. Using this model, `predFun` is then used to predict each of the cases in the i -th subset. After all K models have been fit, one prediction has been made for each of the data points, based on a model that the given point was not used to construct. This vector of predictions is compared to the actual response values using the error function `errFun`.

The output of the `crossVal` function is a `crossVal` object, which does not inherit from `resamp`. It has several components, of which `error` is the key quantity of interest:

- `call`: An image of the call that produced the object.
- `fitted`: A vector of fitted values produced by the cross-validation algorithm.
- `K`: The number of groups that were formed.
- `error`: The final estimate of prediction error.
- `seed.start`: The initial value of the random seed, which can be used to check results using the same random number sequence on future runs.
- `seed.end`: The final value of the random seed.
- `indices`: If indices are saved, a vector indicating to which group each observation was assigned.

A print method is available for the `crossVal` object. It displays only the `call`, `K`, and `error` components. Here is the printed result of the last iteration of the above loop.

```
> res
Call:
crossVal(x = ozone, y = temperature, modelFit =
smooth.spline, K = 10,
  args.modelFit = list(df = i + 1), predFun =
function(object,
  newdata)
predict(object, x = newdata)$y, seed = i)

Number of Groups: 10

Prediction Error: 38.28
```

Bootstrap prediction: the bootPred function

The main arguments to `bootPred` are `B`, `modelFit`, `predFun`, and `errFun`. `B` is the number of bootstrap samples formed. The sampling itself is performed by the function `samp.MonteCarlo`. For the b th bootstrap sample, the model is fit using the function given in `modelFit`, then predictions are done on both the original data and the bootstrap data using `predFun`. Next, errors are calculated using the error function `errFun`. The default error function is squared error. Two sets of errors are calculated--between the predictions done on the original data (using the bootstrap-fitted model) and the original data itself, and between the predictions done on the bootstrap data and the bootstrap data itself. After all B samples are done, *optimism* is calculated as the average difference between those two errors. Further calculations for the `.632` and `.632+` estimators of prediction error are based on the average error rates in bootstrap samples in which the i -th case does not appear. It is therefore important to choose B large enough that there is at least one bootstrap sample that omits every original observation. You can be approximately 99.9% sure of this if $(1 - e^{-1})^B < 0.001/n$. Beyond that, choosing a large value of B increases accuracy and computation time.

Here is the above example worked with `bootPred` using $B=30$ and the `.632+` estimator for each test.

```
> tempErr <- rep(NA, 11)
> for(i in 1:11){
+   res <- bootPred(ozone, temperature,
+   smooth.spline, args.modelFit = list(df = i+1),
+   predFun = function(object, newdata){ predict(object,
+   x = newdata)$y}, B = 30, seed = i)
+ tempErr[i] <- res$err632plus
+ }
> tempErr
[1] 40.80203 39.60939 38.43314 37.84439 38.18068 36.98436
39.54523
[8] 40.28747 39.18808 40.26209 39.85067

> argminErr <- which(tempErr == min(tempErr))[1] + 1
> argminErr
[1] 7
```

Again a close call, between 5 and 7 degrees of freedom, with 7 winning out.

```
> lines(smooth.spline(ozone,temperature, df = argminErr),
        lty = 2)
```

See Figure 1.19 for this fitted curve.

The output of the `bootPred` function is a `bootPred` object, which does not inherit from `resamp`. It has the following components, of which `err632` and `err632plus` are the key quantities of interest:

- `call`: An image of the call that produced the object.
- `B`: The number of bootstrap samples used.
- `apparent.error`: The prediction error calculated using the original model and the original data.
- `optimism`: The average decrease in error due to overfitting (bootstrap model on bootstrap data versus original data).
- `err632`: The prediction error estimate calculated by the .632 method.
- `err632plus`: The prediction error estimate calculated by the .632+ method.
- `seed.start`: The initial value of the random seed, which can be used to check results using the same random number sequence on future runs.
- `seed.end`: The final value of the random seed.
- `indices`: If indices are saved, a matrix indicating which observations were assigned to each bootstrap sample (the i -th column of the matrix represents the i -th bootstrap sample).

The print method for the `bootPred` object displays the `call`, `B`, `apparent.error`, `optimism`, `err632` and `err632plus` components.

```
> res

Call:
bootPred(x = ozone, y = temperature, modelFit =
smooth.spline, B = 30,
  args.modelFit = list(df = i + 1), predFun =
function(object,
  newdata)
{
  predict(object, x = newdata)$y
```

```
}  
, seed = i)  
Number of Replications: 30  
Apparent Error Rate: 31.53  
optimism: 8.155  
.632 Prediction Error: 39.52  
.632+ Prediction Error: 39.85
```


METHODS FOR RESAMPLE OBJECTS

The print Method

The `print` method for a resample object, `print.resamp`, prints out the call, the number of resamples used, and a table giving the values of the statistic for the original data and resampling estimates of bias, mean, and standard error for the statistic.

The summary Method

The `summary` method for a resample object prints out the same information as `print.resamp`, followed by the empirical percentiles of the replicates. The summary of a bootstrap object also calculates BCa percentiles. If the statistic is vector-valued, a correlation matrix for the components of the vector is also printed. The optional `probs` argument specifies probabilities at which the empirical quantiles are calculated.

Additional arguments useful in `limits.bca` may be specified with `summary.bootstrap`. These arguments include `z0`, `acceleration`, and `group.size`. By default, a `group.size` of `floor(n/20)` is used in `limits.bca` for reasons of speed. To do a full jackknifing when estimating acceleration, specify `group.size=1`.

The plot Method

The `plot` method for a resample object produces plots of the distributions of the statistics. For each component of the statistic, a histogram of the replicates is displayed with an overlaid smooth density estimate. A solid vertical line is plotted at the observed parameter value, and a dashed vertical line is plotted at the mean of the replicates. The distance between the dotted line and the solid line is the estimated bias. The shape of the distribution may be examined to assess issues such as skewness of the distribution of the statistic.

You may specify `plot` with a `bandwidth.func` argument to calculate the bandwidth of the density estimate. By default, the normal reference density estimate is used. In addition, you may specify `plot` with an `nclass.func` argument to calculate the number of classes in the histogram. By default, the Freedman and Diaconis rule is used. Arguments may also be passed to `histogram` through the ellipsis (...).

Plots are displayed in a grid (`grid=T`) by default. Use `nrow` to specify the number of rows in the grid. If the statistic is a matrix, then by default the plots are arranged in the same order as the terms appear in the matrix.

Normal Quantile- Quantile Plots

The `qqnorm` method for a `resample` object produces a plot with the same layout as in `plot.resamp`, but with each plot containing a normal quantile-quantile plot for the relevant statistic. If the argument `lines=T`, as is the default, then a `qqline` is also added to each plot.

This plot is used to assess the normality of the distribution of each statistic. If the points fall on a straight line, the empirical distribution of the replicates is similar to that of a normal random variate.

GUI INSTRUCTIONS (WINDOWS ONLY)

The Menu Hierarchy

When you load the Resample library, you enable three new GUI features. New Bootstrap and Jackknife menus are automatically added, allowing you to take advantage of the updated bootstrap and jackknife features discussed in the previous sections. In addition, a new bootstrap tab is added to the linear regression dialog.

The Bootstrap Dialog

To launch the new bootstrap menu, select **Statistics > Resample > BootstrapNEW** (see Figure 1.20). The original bootstrap menu is still available under **Statistics > Resample > Bootstrap**. To illustrate, let's follow the "Resampling the Variance" example beginning on page 7.

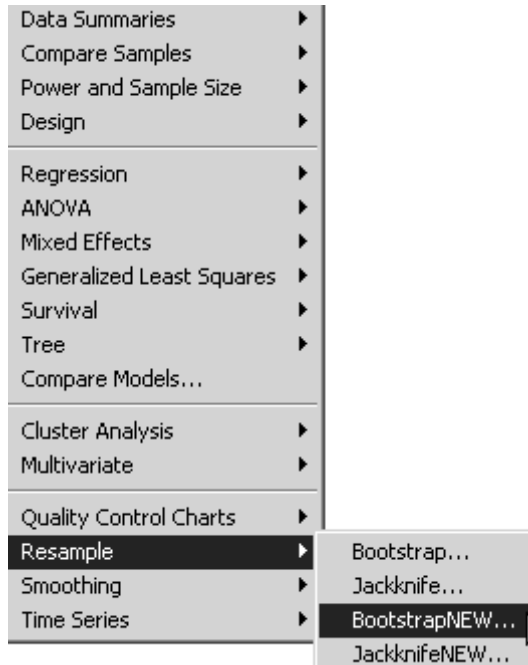


Figure 1.20: Location of Bootstrap Menu

The Bootstrap menu contains five tabs, discussed below. Note that additional help is available from every menu by clicking on the Help button at the lower right corner.

MODEL Tab:

The MODEL tab (Figure 1.21) allows you to specify the data set, the statistic to estimate, the sampler, options for saving, and it includes space for providing any additional arguments to `bootstrap` not covered by the rest of the tabs.

Following the example (page 7), we set `swiss.x[, "Education"]` as the Data Set, and the model is saved as `boot.obj1`. The statistic to estimate is `var` and the default sampler is accepted.

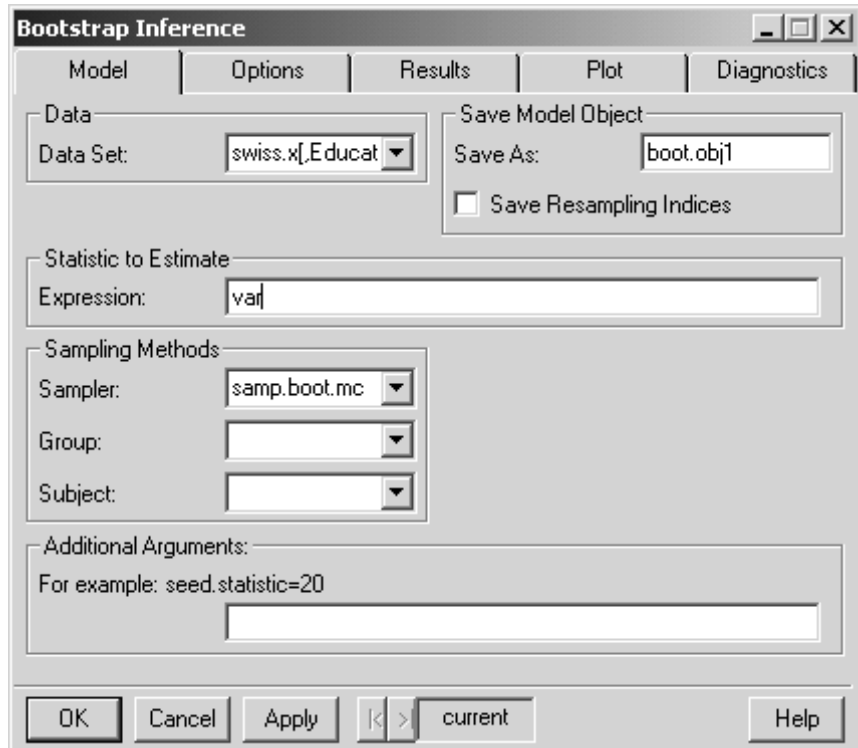


Figure 1.21: GUI for MODEL Tab of Bootstrap Inference Dialog

OPTIONS Tab:

The OPTIONS tab contains optional parameters such as the number of resamples and parameters for computing linear approximations.

In this example, the default settings on the OPTIONS tab (Figure 1.22) are accepted and the Random Seed (corresponding to bootstrap argument seed) is set to zero.

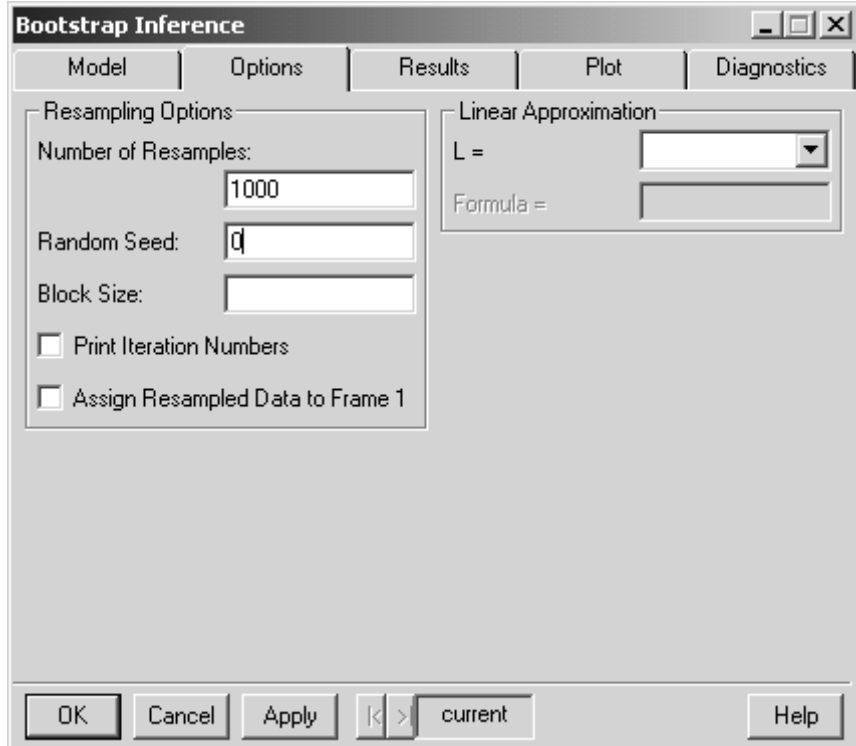


Figure 1.22: GUI for OPTIONS Tab in Bootstrap Inference Dialog

RESULTS Tab:

The RESULTS tab contains options for producing summary tables and confidence intervals.

In our example, in addition to the default settings on the RESULTS tab (Figure 1.23), the Empirical Percentiles option is selected to duplicate the summary output results on page 9.

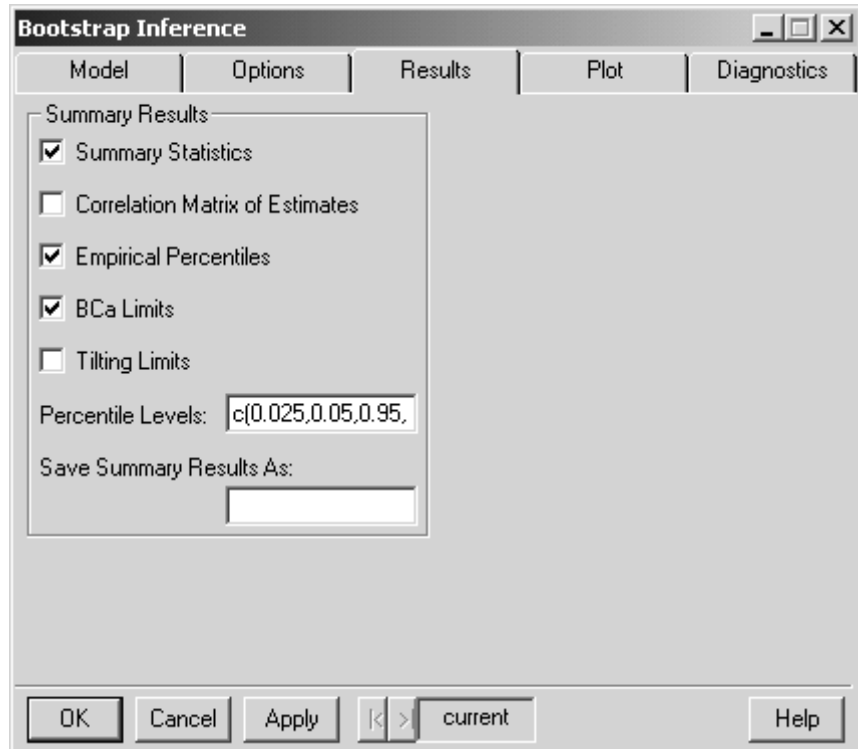


Figure 1.23: GUI for RESULTS Tab in Bootstrap Inference Dialog

PLOT Tab:

The PLOT tab contains options for producing plots of the results.

In our example, both plots are selected (Figure 1.24) corresponding to the two plots on page 10.

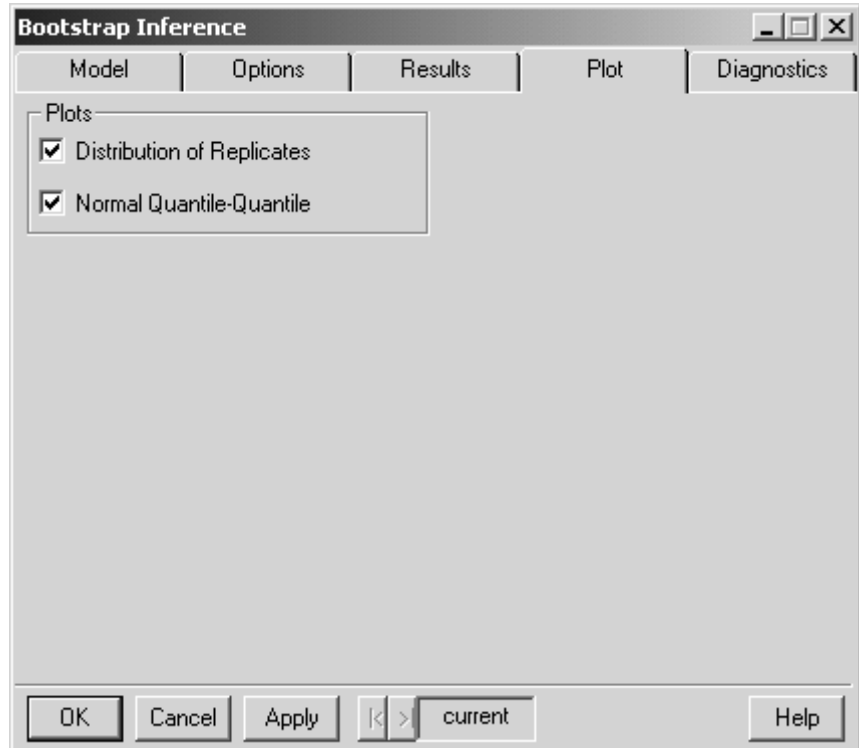


Figure 1.24: *GUI for PLOT Tab in Bootstrap Inference Dialog*

DIAGNOSTICS Tab:

The DIAGNOSTICS tab allows you to perform Jackknife After Bootstrap or Tilt After Bootstrap diagnostic and sensitivity analysis procedures.

We have chosen the default settings (Figure 1.25), which do not perform these procedures.

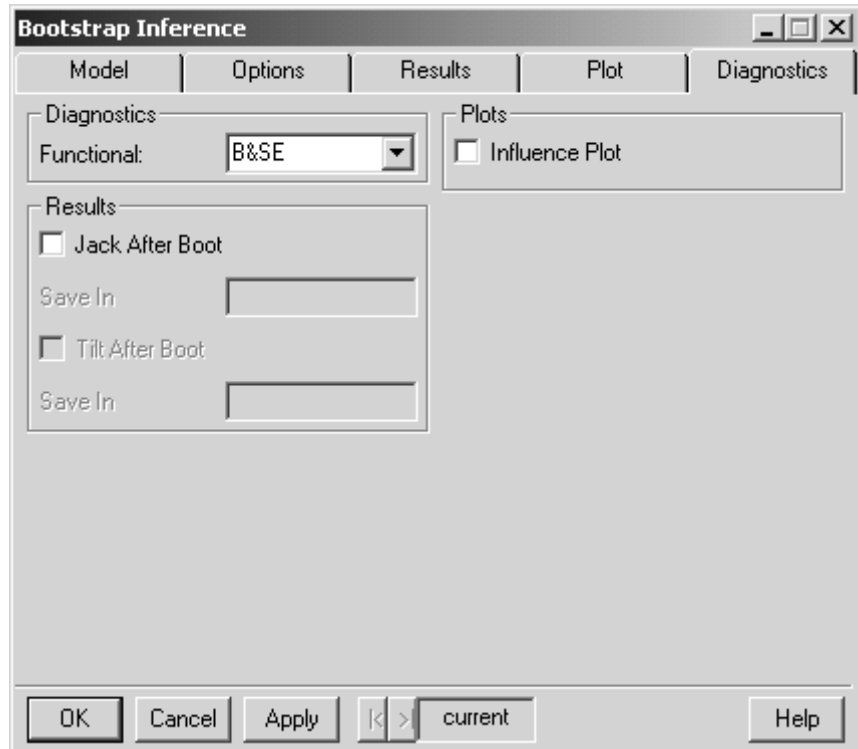


Figure 1.25: GUI for DIAGNOSTICS Tab in Bootstrap Inference Dialog

Hit the Apply or OK buttons (discussed in more detail on page 79) to carry out the bootstrap procedure and view the output.

The Jackknife Dialog

To launch the new jackknife menu, select **Statistics > Resample > JackknifeNEW** (Figure 1.26).

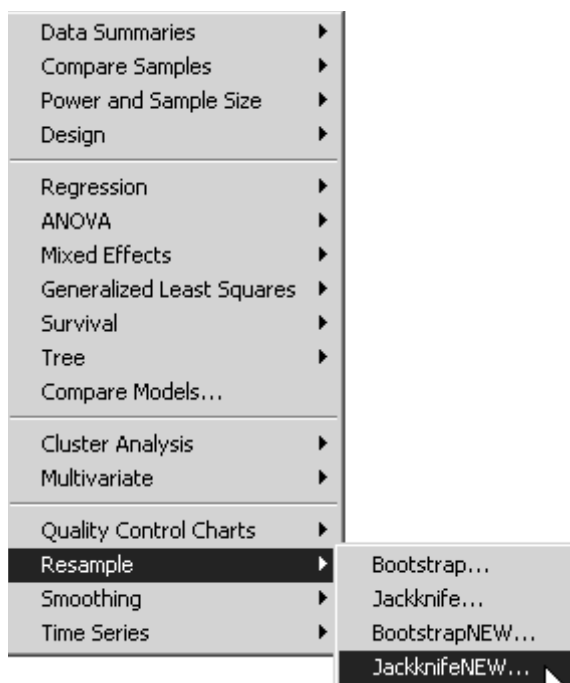


Figure 1.26: Location of Jackknife Menu

The Jackknife menu contains four tabs. To see options in a different tab in the dialog box, click on the page name or press CTRL+Tab to move from page to page.

The Jackknife tabs are similar to the MODEL, OPTIONS, RESULTS, and PLOT tabs in the Bootstrap menu.

Bootstrap Options on the Linear Regression Dialog

We discussed bootstrapping fitted objects such as those created by `lm` on page 30. The `LinearBootstrap` dialog allows you to bootstrap linear regression statistics directly from the linear regression dialog. To launch the `LinearBootstrap` menu, select **Statistics > Regression > LinearBootstrap** (see Figure 1.27). Notice that the original linear regression dialog is still available under **Statistics > Regression > Linear**.

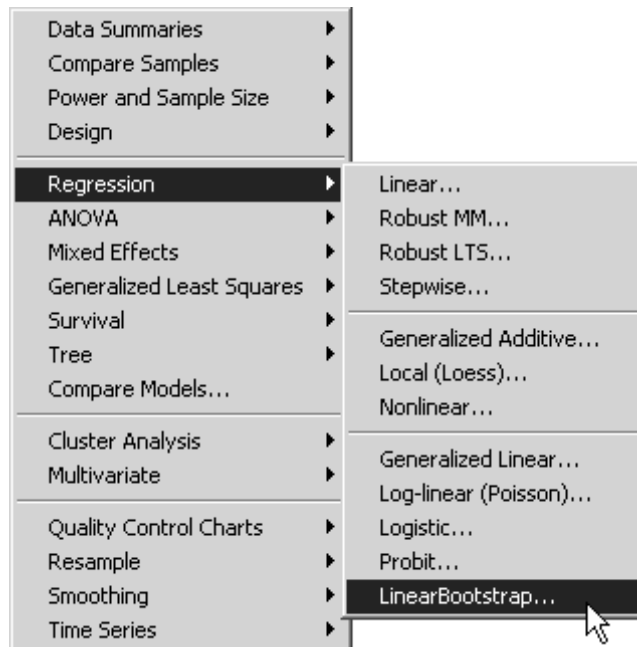


Figure 1.27: Location of the `LinearBootstrap` Menu

The Linear Bootstrap dialog has five tabs.

The tabs are MODEL, RESULTS, PLOT, PREDICT, and BOOTSTRAP (see Figure 1.28). Refer to the Help button in the lower right corner of the dialog for specific explanations of the contents of the first four tabs, which are the same as for the original linear regression dialog. The fifth tab, BOOTSTRAP has much of the content we've seen before on the Bootstrap Dialog, particularly the MODEL and the OPTIONS tabs. To illustrate, let's follow the "Resampling Regression Coefficients" example from page 30. Begin by defining `stack` in your commands window:

```
> stack <- data.frame(stack.loss, stack.x)
```

MODEL Tab:

The MODEL page for this example looks like Figure 1.28. The data set is `stack`. The formula is built automatically by selecting dependent and independent variables from the menus. The linear regression results (without bootstrapping) are saved as `lm.object`. Note that if you want to save the bootstrap results (we will do this from the BOOTSTRAP tab), you are required to save the regression results as well. A dialog will inform you of this if you try to save the bootstrap object without saving the regression model object.



Figure 1.28: *The MODEL Tab of the Linear Regression Dialog*

BOOTSTRAP Tab:

The BOOTSTRAP page for this example looks like Figure 1.29.

First, 'Perform Bootstrap' was selected. This activates all of the other fields. Then the default settings were accepted and Normal Quantile-Quantile, Correlation Matrix of Estimates, and Empirical Percentiles were selected. The bootstrap object is saved as `boot.obj3` and `seed = 0` and `trace = F` were added to the Additional Arguments section.

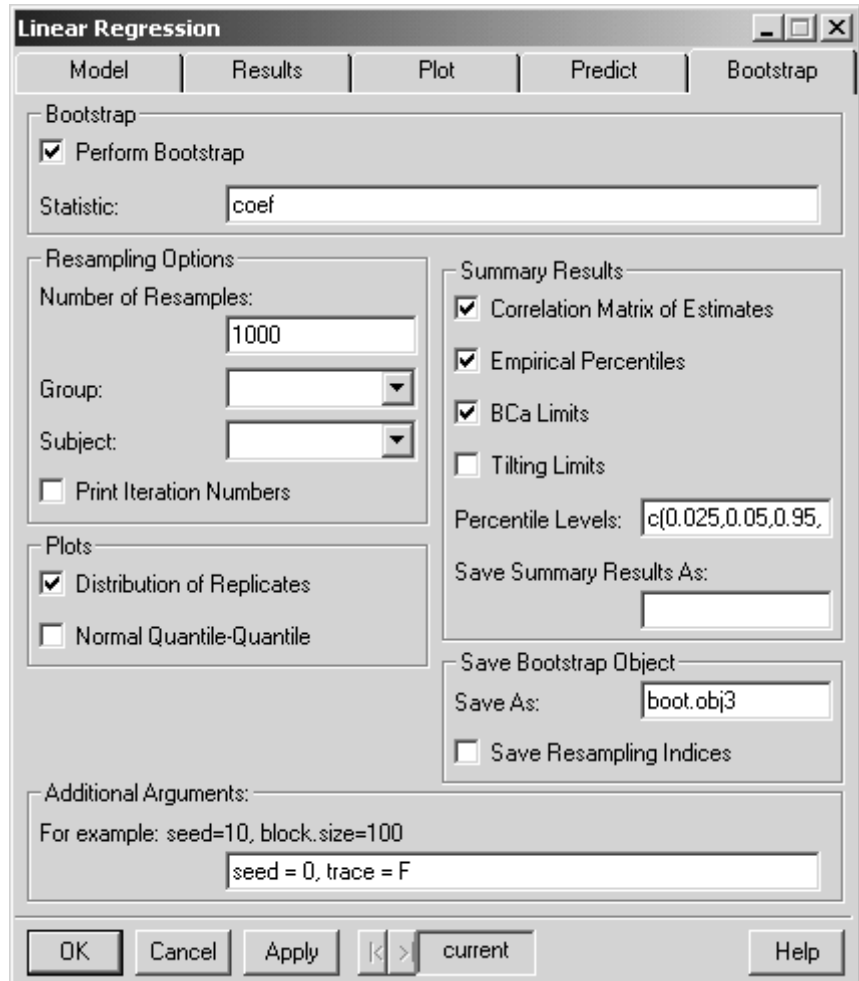


Figure 1.29: *The BOOTSTRAP Tab of the Linear Regression Dialog*

RESULTS, PLOT, and PREDICT Tabs:

The default settings on the RESULTS, PLOT, and PREDICT pages are used in this example.

Modeless Operation

Like all Spotfire S+ dialogs, the Bootstrap dialog is modeless. This means that it can be moved around on the screen and it stays open until you choose to close it. The benefit of this is that when you make changes by clicking APPLY, you can see the effects the changes have without closing the dialog.

The OK, Cancel, and Apply Buttons

When you are finished setting options in the Bootstrap or Jackknife Dialog, you can choose the OK, Cancel, or Apply button (see Figure 1.30).

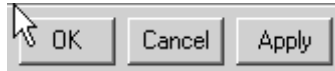


Figure 1.30: *The OK, Cancel, and Apply Buttons*

- Choose the OK button or press ENTER to close the dialog and carry out the resampling procedure. If you re-open the menu, many of your previous changes will be lost.
- Choose the Cancel button or press ESC to close the dialog and discard any changes you have made to the dialog pages. If you press cancel without having already clicked APPLY, no resampling is performed. Changes cannot be cancelled if you have already clicked APPLY.
- The APPLY button is similar to the OK button, but it does not close the dialog box. Instead, the actions you have chosen are implemented so you can view their effect, but the dialog box is still available to you for making further changes. To use this function, choose APPLY or press CTRL+ENTER. If no changes have been made, the APPLY button will be disabled.
- The Traceback button (see Figure 1.31) allows you to load data that was previously applied.



Figure 1.31: *The Traceback Button*

Typing and Editing

The following tasks can be performed in the Jackknife and Bootstrap Dialog using special keys.

Table 1.2: *Actions Using Special Keys*

Special Key	Action
Tab	Move to the next option in the dialog
SHIFT+Tab	Move to the previous option in the dialog
CTRL+Tab	Move between tabbed pages
ALT+underlined letter in the options name. Press again to move to additional options with the same underlined letter.	Move to a specific option and select it
ALT+DOWN direction key	Display a dropdown list
UP or DOWN direction keys to move, ALT+DOWN direction key to close the list	Select an item from a list
ALT+DOWN direction key	Close a list without selecting any of the items

The dialog page contains many text edit boxes. Text boxes allow you to type in information such as a file name or a graph title.

To replace text in a dialog:

Select the existing text with the mouse, or press ALT+underlined letter in the option name.

Type the new text.

Any highlighted text is immediately overwritten when you begin typing the new text.

To edit in a text box:

1. Position the insertion point in the text box. If text is highlighted, it will be replaced when you begin typing.
2. Edit the text.

REFERENCES

- Breiman, L. & Friedman, J. H. (1985). Estimating optimal transformations for multiple regression and correlation. (with discussion). *Journal of the American Statistical Association* 80, 580-619.
- Davison, A., & Hinkley, D. (1997). *Bootstrap Methods and their Applications*. Cambridge University Press.
- Efron, B. (1990). More efficient bootstrap computations. *Journal of the American Statistical Association*, 85(409):79-89.
- Efron, B. (1992). Jackknife-after-bootstrap standard errors and influence functions (with discussion). *Journal of the Royal Statistical Society, Series B*, 54:83-127.
- Efron, B. & Tibshirani, R.J. (1993). *An Introduction to the Bootstrap*. San Francisco: Chapman & Hall.
- Efron, B. & Tibshirani, R. (1997). Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548-560.
- Fisher, R. A. (1935). *The Design of Experiments*. Edinburgh: Oliver and Boyd.
- Hesterberg, T. C. (1994). Tail-specific linear approximations for efficient bootstrap simulations. In Sall, J. and Lehman, A., editors, *Proceedings of the Conference on the Interface between Computing Science and Statistics*, volume 26, pages 472-481. Interface Foundation.
- Hesterberg, T. C. (1995). Weighted average importance sampling and defensive mixture distributions. *Technometrics*, 37(2):185-194.
- Hesterberg, T. C. & Ellis, S. J. (1999). Linear approximations for functional statistics in large-sample applications. Research Department 86, Insightful Corp., 1700 Westlake Ave. N., Suite 500, Seattle, WA 98109.
- Shao, J. & Tu, D. (1995). *The Jackknife and Bootstrap*. New York: Springer-Verlag.

