# Resampling Methods in R: The boot Package

*by Angelo J. Canty*

## Introduction

The bootstrap and related resampling methods are statistical techniques which can be used in place of standard approximations for statistical inference. The basic methods are very easily implemented but for the methods to gain widespread acceptance among users it is necessary that they be implemented in standard statistical packages. In this article I will describe the **boot** package which implements many variants of resampling methods in R. The package was originally written as an S-Plus library released in conjunction with the book by Davison and Hinkley (1997). Subsequently the library was ported to R by Brian Ripley. The **boot** package described here is distinct from the limited suite of bootstrap functions which are now included in S-Plus. It is also distinct from the **bootstrap** package originally written by R. Tibshirani for Efron and Tibshirani (1993).

In this paper I describe the main components of the **boot** package and their use. I will not go into any theoretical detail about the methods described. It is strongly recommended, however, that the user read Davison and Hinkley (1997) before using the package. Initially I will describe the basic function for bootstrapping i.i.d. data and analyzing the bootstrap output. Then I will describe functions for sampling randomly right-censored data and time series data. The availability of the bootstrap for such nonstandard cases is one of the major advantages of the **boot** package.

## The main bootstrap function

The most important function in the package is the `boot` function which implements resampling methods for i.i.d. data. The basic bootstrap in such cases works by fitting a distribution function $\hat{F}$ to the unknown population distribution $F$. The Monte Carlo bootstrap method then proceeds by taking $R$ samples, of the same size as the original sample, from $\hat{F}$. In the parametric bootstrap $F = F_\psi$ is a member of a class of distribution functions indexed by the parameter vector $\psi$ and so $\hat{F} = F_{\hat{\psi}}$ where $\hat{\psi}$ is some consistent estimate of $\psi$. It is then the responsibility of the user to supply (through the parameter `ran.gen`) a function which takes the original data and returns a sample from $\hat{F}$. In the nonparametric case the estimate of $F$ is the empirical distribution function. In this case a sample from $\hat{F}$ can be found by sampling with replacement from the original data, $X_1, \ldots, X_n$. In `boot` all $R$ samples are found by constructing an $R \times n$ matrix of random integers from `1:n`. Indexing the original data by each row of this matrix gives a bootstrap sample.

Let us now suppose that interest is in some functional $\theta = t(F)$. The *plug-in* estimate of this functional is then $t = t(\hat{F})$. Corresponding to a bootstrap sample is a distribution function $\hat{F}^*$. We can then use $t^* = t(\hat{F}^*)$ to estimate $t$. Under suitable regularity conditions we can then approximate the distribution of $t - \theta$ by the empirical distribution of $t^* - t$. Thus estimates of the bias and variance of the estimator $T$ are

$$b = \bar{t}^* - t, \quad v = \frac{1}{R-1}\sum_{r=1}^{R}(t_r^* - \bar{t}^*)^2 \qquad (1)$$

Since `boot` is designed to be a general function for the bootstrap, the user must supply a function (`statistic`) which calculates the required functional. In the parametric bootstrap this is simply a function of a dataset. In the non-parametric case, `statistic` must be a function of the original dataset and a second argument which is used to determine a bootstrap sample. The simplest form that this second argument can take is to be a vector of indices such as a row of the index matrix constructed by `boot`. An equivalent method for exchangeable data is to supply the vector of frequencies of the original data points in the bootstrap sample. Another alternative is to supply the probability weights corresponding to $\hat{F}^*$. For bootstrap samples such weights are simply the frequencies divided by the sample size. They have the advantage, however, that any set of $n$ weights which sum to 1 can be used. This allows for numerical differentiation of the functional $t(\cdot)$ at the datapoints which gives us the *empirical influence values*. Note that `boot` automatically normalizes weights to sum to 1 prior to calling `statistic`. The user must tell `boot` which second argument is being expected by `statistic`, this is achieved by specifying the `stype` parameter which can be `"i"`, `"f"` or `"w"`.

The following is an example of using `boot` at its most basic level. For many uses, this will be sufficient to run the bootstrap. The code performs a non-parametric bootstrap for the mean of the `aircondit` data.

```
> mean.w <- function(x, w) sum(x*w)
> air.boot <- boot(data=aircondit$hours,
+                 statistic=mean.w,
+                 R=999, stype="w")
```

The use of this second argument may seem confusing at first but it allows more flexibility in the types of data structures that we can bootstrap and how bootstrapping is applied to the data. For example, consider the case of bootstrapping for linear models. The data would generally be a matrix or dataframe. The two methods that could be used are to resample rows or resample residuals and then reconstruct a response vector. The following code shows how both of these can be achieved for the catsM data set.

```
> data(catsM)
> cats.lm <-lm(Hwt~Bwt, data=catsM)
> cats1 <- catsM
> cats1$fit <- fitted(cats.lm)
> cats1$res <- resid(cats.lm)
> cats.fit <- function(data) {
+    mod <- lm(data$Hwt~data$Bwt)
+    c(coef(mod),
+      summary(mod)$coef[,2]^2) }
> case.fun <- function(d,i)
+    cats.fit(d[i,])
> model.fun <- function(d,i) {
+    d$Hwt <- d$fit+d$res[i]
+    cats.fit(d) }
> cats.case <- boot(cats1, case.fun,
+                  R=999)
> cats.mod <- boot(cats1, model.fun,
+                 R=999)
```

One advantage of the **boot** package is that it implements many variants on the basic non-parametric bootstrap method. One obvious extension is to multi-sample problems. The user need only specify strata as a numeric vector or factor defining the groups. This fits different empirical distribution functions to each stratum and samples accordingly. Another possibility is that we may want to resample from the data with unequal weights. This arises in the context of bootstrap hypothesis testing and in using importance sampling with the bootstrap as suggested by Johns (1988) and Davison (1988). The sampling probabilities are passed using the weights argument to boot. Other types of resampling can also be done and are specified using the sim argument. Two of these are attempts at more efficient Monte Carlo sampling for the bootstrap; the balanced bootstrap (Davison et al., 1986) and the antithetic bootstrap (Hall, 1989). The final option is to resample without replacement as required for permutation tests.

## Analysis of bootstrap output

The result of calling the boot function is that an object having the class "boot" is returned. This object contains most of the inputs to the boot function or the default values of those not specified. It also contains three additional components. The first of these is t0 which is the result of evaluating the statistic on the original dataset. The second, t, is the matrix of bootstrap replicates. Each row of the matrix corresponds to the value of the statistic applied to a bootstrap dataset. Finally, there is a component seed which contains the value of .Random.seed used to start the Monte Carlo sampling. There are two main reasons why this component is useful. The first is the issue of reproducibility of the bootstrap. In research it is often useful to apply different statistics to the same set of bootstrap samples for comparisons. Without the saved random seed this would not be possible. The second reason is that there are situations in which it is important to be able to look at the bootstrap samples themselves. This can be done by constructing the matrix of bootstrap indices (or frequencies). In an early version of the package that matrix was stored but that required excessive storage. By storing the random seed we can recreate the matrix whenever required in very little time. The function boot.array does this.

The two main methods for boot objects are print and plot methods. If the user prints a boot object then a short summary of the bootstrap results are given. Here are the results of case resampling the catsM dataset.

```
> cats.case
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = cats1, statistic = case.fun,
     R = 999)


Bootstrap Statistics :
    original      bias    std. error
t1*  -1.1841  0.029573     1.14684
t2*   4.3127 -0.010809     0.40831
t3*   0.9966 -0.012316     0.16043
t4*   0.1155 -0.001534     0.01769
```

In this example, the statistic returns a vector of length 4, the first two components are the coefficient estimates and the second two are the estimated variances of the estimates from the usual linear model theory. We note that the bootstrap standard errors are 15–20% higher than the usual standard errors.

It is not safe to use the output of a bootstrap without first looking at a graphical plot of the bootstrap replicates. They are the first and most basic check that the bootstrap has produced sensible results. One common problem is that of discreteness of the bootstrap distribution. If this is a problem it should be fairly evident from the plots. The solution may be as simple as increasing the number of replicates or there may be some fundamental problem such as occurs in the case of the sample maximum. In either case the plots will alert the user that the results of this bootstrap cannot be used for inference. The plots may also show up bugs in the coding of the statistic which were not previously evident. Because of the nature of a bootstrap sample, it is possible to get datasets not normally seen in practice (such as having many ties) and the code which worked for the original sample

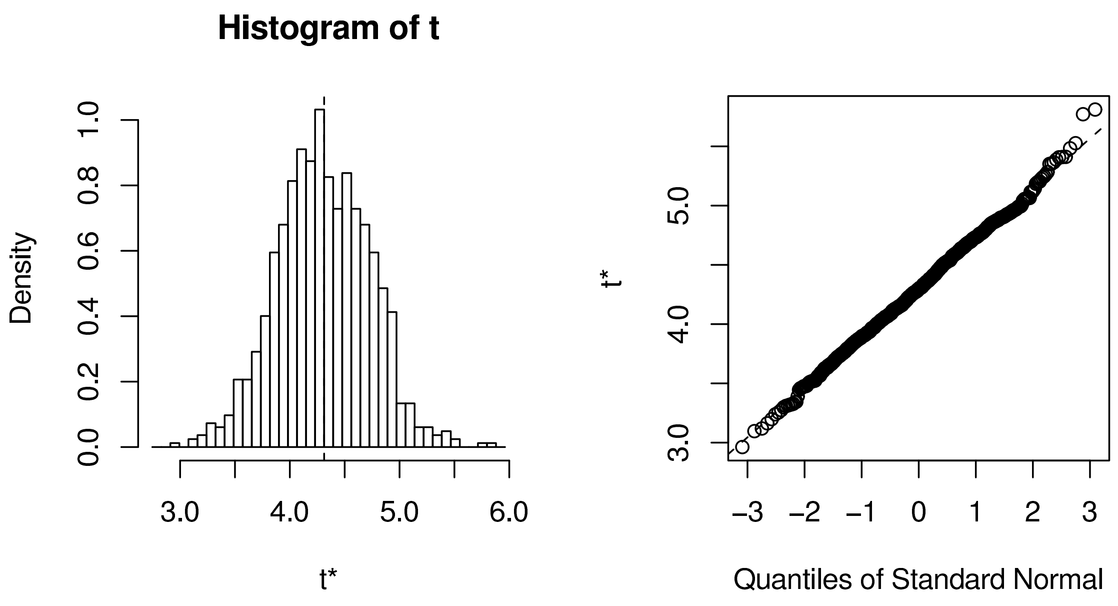

Figure 1: Plots of the bootstrap output for the slope of the catsM dataset using case resampling. The command used was plot(cats.case, index=2).

may not work for some bootstrap samples. Figure 1 shows the plots for the slope estimate of the catsM dataset using case resampling.

A more sophisticated diagnostic tool is the *Jackknife-after-bootstrap* plot suggested by Efron (1992). The plot method may call the function jack.after.boot to draw these. Users should note that the plots produced by jack.after.boot are very different from those produced by the S-Plus function plot.jack.after.bootstrap. As a diagnostic, I think that the version I have used are more informative.

Having examined the bootstrap output and determined that it seems to be sensible the user can proceed to use the output. The most common use is to producing confidence intervals. The function boot.ci can be used for this. This takes a bootstrap output object and returns one or more types of confidence interval for a scalar component of the functional being estimated. There is an index argument to specify which component should be analyzed. boot.ci can produce five different types of bootstrap confidence interval. The *bootstrap normal* interval assumes an asymptotic normal distribution an used the bootstrap estimates of bias and variance for the parameters of the distribution. The *basic bootstrap* and *bootstrap percentile* intervals have less restrictive assumptions. All three intervals are asymptotically equivalent but the latter two tend to have better small sample properties when the normal assumption is questionable. This can be checked using the normal quantile plot produced by the plot method. Two intervals that are asymptotically better than these are the *studentized* and *BCa* intervals. The price for this improvement is more calculation. For the studentized bootstrap one needs a consistent estimate of the variance of each bootstrap replicate. Such variances can be found using asymptotic considerations or through methods such as the jackknife or non-parametric delta method (infinitesimal jackknife). Another alternative is to use a nested bootstrap but this can rapidly become computationally prohibitive. Whichever method is chosen, it must be done for each bootstrap sample. The BCa interval requires calculation of the empirical influence values for the original sample only so the extra computation is minimal but the statistic should be in weighted form unless a closed form is readily available. See Canty et al. (1996) for a simulation study comparing the various intervals

## Resampling censored data

In many practical settings, data is censored and so the usual bootstrap is not applicable. The function censboot implements the bootstrap for random right-censored data. Such data is typically comprised of the bivariate observations $(Y_i, D_i)$ where

$$Y_i = \min(X_i, C_i) \qquad D_i = I(X_i \leq C_i)$$

where $X_i \sim F$ and $C_i \sim G$ independently and $I(A)$ is the indicator function of the event $A$. Non-parametric estimates of $F$ and $G$ are given by the Kaplan–Meier estimates $\hat{F}$ and $\hat{G}$, the latter being obtained by replacing $d_i$ by $1 - d_i$. We can then proceed by sampling $X_1^*, \ldots, X_n^*$ from $\hat{F}$ and independently sampling $C_1^*, \ldots, C_n^*$ from $\hat{G}$. $(Y_i^*, D_i^*)$ can then be found from $(X_i^*, C_i^*)$ in the same way as for the original data. Efron (1981) showed that this is identical to resampling with replacement from the original pairs.

An alternative approach is the conditional bootstrap. This approach conditions the resampling on the observed censoring pattern since this is, in effect, an ancillary statistic. We therefore sample $X_1^*, \ldots, X_n^*$ from $\hat{F}$ as before. If the $i^{\text{th}}$ observation is censored then we set $C_i^* = y_i$ and if it is not censored we sample an observation from the estimated conditional distribution of $C_i$ given that $C_i > y_i$. Having thus obtained $X_1^*, \ldots, X_n^*$ and $C_1^*, \ldots, C_n^*$ we proceed as before. There is one technicality which must be addressed for this method to proceed. Suppose that the maximum value of $y_1, \ldots, y_n$, $y_k$ say, is a censored observation. Then $X_k^* < y_k$ and $C_k^* = y_k$ so the bootstrap observation will always be uncensored. Alternatively, if the the maximum value is uncensored, the estimated conditional distribution does not exist. In order to overcome these problems we add one extra point to the dataset which has an observed value greater than $\max(y_1, \ldots, y_n)$ and has the opposite value of the censoring indicator to the maximum.

One other method for resampling from censored data is the weird bootstrap introduced by Andersen et al. (1993). This works by simulating from the Nelson-Aalen estimate of the cumulative hazard function.

The function `censboot` implements all of these resampling schemes. The user simply specifies the data (as a matrix or data.frame with at least 2 columns), the statistic (an R function taking the dataset as its input), the number of resamples and the Kaplan–Meier estimates of the survival and censoring distributions as found using the function `survfit` in the recommended package **survival**.

In practice there are usually covariates which affect the survival distribution. The most common assumption is that the survival distribution depends on the covariates, $x$, through a *Cox Proportional Hazards Model* (Cox, 1972).

$$1 - F(y; \beta, x) = \{1 - F_0(y)\}^{\exp(x^T\beta)}$$

where $\beta$ is a vector of unknown parameters and $1 - F_0(y)$ is a baseline survivor function. The function `coxph` in the **survival** package fits such models. For the bootstrap this does not cause a great complication. The only difference from before is the estimated distribution from which failure times are generated. Earlier we mentioned that, without covariates, sampling from the model was identical to sampling pairs, this is no longer true in the case with covariates. Both methods are possible with `censboot`, specifying `sim="ordinary"` will always sample cases whereas `sim="model"` will resample from the Cox model if appropriate and otherwise will resample cases.

The following example looks at the ulcerated cases in the `melanoma` dataset. The aim is to produce a confidence interval for the exponent of the coefficient of tumor thickness in the proportional hazards model. Note that it is necessary to ensure that the censoring indicator is of the form specified above.

```
> data(melanoma)
> library(survival)
> mel <- melanoma[melanoma$ulcer==1,]
> mel$cens <- 1*(mel1$status==1)
> mel.cox <- coxph(Surv(time, cens)
+                  ~thickness, data=mel)
> mel.surv <- survfit(mel1.cox)
> mel.cens <- survfit(Surv(time,1-cens)
+                     ~1, data=mel)
> mel.fun <- function(d) {
+    coxph(Surv(time, cens)~thickness,
+          data=d)$coefficients }
> mel.boot <- censboot(mel, mel.fun,
+                R=999, sim="cond",
+                F.surv=mel.surv,
+                G.surv=mel.cens,
+                cox=mel.cox,
                 index=c(1,8))
> mel.boot
CONDITIONAL BOOTSTRAP FOR CENSORED DATA
Call:
censboot(data=mel, statistic=mel.fun,
         R=999, F.surv=mel.surv,
         G.surv=mel.cens, sim="cond",
         cox=mel.cox, index=c(1,8))

Bootstrap Statistics :
      original      bias    std. error
t1* 0.09971374 0.03175672   0.0456912

>  boot.ci(mel.boot,
+          type=c("basic", "perc"),
+          h=exp)
BOOTSTRAP CONFIDENCE INTERVAL
CALCULATIONS
Based on 999 bootstrap replicates

CALL :
boot.ci(boot.out=mel.boot,
        type=c("basic", "perc"),  h=exp)

Intervals :
Level      Basic          Percentile
95%   (0.952, 1.158)  (1.052, 1.258)
Calculations and Intervals on
Transformed Scale
```

The usual interval from `summary(mel.cox)` is $(1.0205, 1.1962)$ which is narrower than the bootstrap intervals suggesting that the asymptotic interval may be undercovering.

## Resampling time series

In applying the bootstrap to time series data it is essential that the autocorrelations be properly accounted for. The most common way of doing this is to sample the observations in blocks rather than individually (Künsch, 1989). Thus if we choose a block length $l$ and we have $n = ml$ for some integer $m$ then the resampled time series is constructed by putting $m$ blocks together. When $m = n/l$ is not an integer the last block is shortened so that the resampled time series is of the appropriate length. Blocks are usually taken as overlapping so that there are $n - l + 1$ possible blocks. This can be increased to $n$ by allowing blocks to wrap around from the end to the start of the time series. One problem with the block bootstrap is that the resulting time series is not stationary. Politis and Romano (1994) proposed the *stationary bootstrap* to overcome this problem. In the stationary bootstrap the block length is randomly generated with a geometric distribution and the block start is selected randomly from the integers $\{1, \ldots, n\}$. Wrapping at the end of the time series is necessary for this method to work correctly. `tsboot` can do either of these methods by specifying `sim="fixed"` or `sim="geom"` respectively. A simple call to `tsboot` includes the time series, a function for the statistic (the first argument of this function being the time series itself), the number of bootstrap replicates, the simulation type and the (mean) block length .

```
> library(ts)
> data(lynx)
> lynx.fun <- function(tsb) {
+     fit <- ar(tsb, order.max=25)
+     c(fit$order, mean(tsb)) }
> tsboot(log(lynx), lynx.fun, R=999
+        sim="geom", l=20)
STATIONARY BOOTSTRAP FOR TIME SERIES
Average Block Length of 20

Call:
tsboot(tseries=log(lynx),
       statistic=lynx.fun,
       R=999, l=20, sim="geom")

Bootstrap Statistics :
     original        bias    std. error
t1* 11.000000 -6.593593594   2.5400596
t2*  6.685933 -0.001994561   0.1163937
```

An alternative to the block bootstrap is to use model based resampling. In this case a model is fitted to the time series so that the errors are i.i.d. The observed residuals are sampled as an i.i.d. series and then a bootstrap time series is reconstructed. In constructing the bootstrap time series from the residuals, it is recommended to generate a long time series and then discard the initial burn-in stage. Since the length of burn-in required is problem specific, `tsboot` does not actually do the resampling. Instead the user should give a function which will return the bootstrap time series. This function should take three arguments, the time series as supplied to `tsboot`, a value `n.sim` which is the length of the time series required and the third argument containing any other information needed by the random generation function such as coefficient estimates. When the random generation function is called it will be passed the arguments `data`, `n.sim` and `ran.args` passed to `tsboot` or their defaults.

One problem with the model-based bootstrap is that it is critically dependent on the correct model being fitted to the data. Davison and Hinkley (1997) suggest *post-blackening* as a compromise between the block bootstrap and the model-based bootstrap. In this method a simple model is fitted and the residuals are found. These residuals are passed as the dataset to `tsboot` and are resampled using the block (or stationary) bootstrap. To create the bootstrap time series the resampled residuals should be put back through the fitted model filter. The function `ran.gen` can be used to do this.

```
> lynx1 <- log(lynx)
> lynx.ar <- ar(lynx1)
> lynx.res <- lynx.ar$resid
> lynx.res <- [!is.na(lynx.res)]
> lynx.res <- lynx.res-mean(lynx.res)
> lynx.ord <- c(lynx.ar$order,0,0)
> lynx.mod <- list(order=lynx.ord,
+                  ar=lynx.ar$ar)
> lynx.args <- list(mean=mean(lynx1),
+                   model=lynx.mod)
> lynx.black <- function(res, n.sim,
+                        ran.args) {
+    m <- ran.args$mean
+    ts.mod <- ran.args$model
+    m+filter(res, ts.mod$ar,
+             method="recursive") }
> tsboot(lynx.res, lynx.fun, R=999,
+        l=20, sim="fixed", n.sim=114,
+        ran.gen=lynx.black,
+        ran.args=lynx.args)
POST-BLACKENED BLOCK BOOTSTRAP FOR
TIME SERIES
Fixed Block Length of 20

Call:
tsboot(tseries=lynx.res,
       statistic=lynx.fun, R=999,
       l=20, sim="fixed", n.sim=114,
       ran.gen=lynx.black,
       ran.args=lynx.args)

Bootstrap Statistics :
         original   bias    std. error
t1*  0.000000e+00 9.732733  3.48395113
t2* -4.244178e-18 6.685819  0.09757974
```

A final method which is available for bootstrapping of time series is *phase scrambling*. Unlike the other methods described above, phase scrambling works on the frequency domain. See Braun and Kulperger (1997) for a discussion of the properties of this method.

## Further comments

In this article I have attempted to describe concisely the main functions in the **boot** package for bootstrapping. The package also has functions which implement saddlepoint approximations to the bootstrap as described in Canty and Davison (1999). There are also functions which do exponential tilting of the resampling distribution and other forms of importance resampling in the bootstrap. These are quite specialized uses of the package and so the user is advised to read the relevant sections of Davison and Hinkley (1997) before using these functions.

## Acknowledgments

I would like to thank A. C. Davison and V. Ventura for their many helpful suggestions in the development of this library. Thanks are also due to B. D. Ripley for a great deal of help and for porting the code to R. Any bugs in the code, however, are my responsibility and should be reported to me in the first instance.

## Bibliography

Andersen, P. K., Borgan, Ø., Gill, R. D., and Keiding, N. (1993), *Statistical Models Based on Counting Processes*, New York: Springer. 5

Braun, W. J. and Kulperger, R. J. (1997), "Properties of a Fourier bootstrap method for time series," *Communications in Statistics — Theory and Methods*, **26**, 1329–1327. 6

Canty, A. J. and Davison, A. C. (1999), "Implementation of saddlepoint approximations in resampling problems," *Statistics and Computing*, **9**, 9–15. 6

Canty, A. J., Davison, A. C., and Hinkley, D. V. (1996), "Reliable confidence intervals. Discussion of "Bootstrap confidence intervals", by T. J. DiCiccio and B. Efron," *Statistical Science*, **11**, 214–219. 4

Cox, D. R. (1972), "Regression Models and Life Tables" (with discussion), *Journal of the Royal Statistical Society series B*, **34**, 187–220. 5

Davison, A. C. (1988), "Discussion of the Royal Statistical Society meeting on the bootstrap," *Journal of the Royal Statistical Society series B*, **50**, 356–357. 3

Davison, A. C. and Hinkley, D. V. (1997), *Bootstrap Methods and Their Application*, Cambridge: Cambridge University Press. 2, 6, 7

Davison, A. C., Hinkley, D. V., and Schechtman, E. (1986), "Efficient bootstrap simulation," *Biometrika*, **73**, 555–566. 3

Efron, B. (1981), "Censored data and the bootstrap," *Journal of the American Statistical Association*, **76**, 312–319. 4

—— (1992), "Jackknife-after-bootstrap standard errors and influence functions " (with discussion), *Journal of the Royal Statistical Society series B*, **54**, 83–127. 4

Efron, B. and Tibshirani, R. J. (1993), *An Introduction to the Bootstrap*, New York: Chapman & Hall. 2

Hall, P. (1989), "On efficient bootstrap simulation," *Biometrika*, **76**, 613–617. 3

Johns, M. V. (1988), "Importance sampling for bootstrap confidence intervals," *Journal of the American Statistical Association*, **83**, 709–714. 3

Künsch, H. R. (1989), "The jackknife and bootstrap for general stationary observations," *Annals of Statistics*, **17**, 1217–1241. 5

Politis, D. N. and Romano, J. P. (1994), "The stationary bootstrap," *Journal of the American Statistical Association*, **89**, 1303–1313. 6

*Angelo J. Canty*
*McMaster University, Hamilton, Ont, Canada*
cantya@mcmaster.ca

# Diagnostic Checking in Regression Relationships

*by Achim Zeileis and Torsten Hothorn*

## Introduction

The classical linear regression model

$$y_i \quad = \quad x_i^\top \beta + u_i \qquad (i = 1, \dots, n) \qquad (1)$$

is still one of the most popular tools for data analysis despite (or due to) its simple structure. Although it is appropriate in many situations, there are many pitfalls that might affect the quality of conclusions drawn from fitted models or might even lead to uninterpretable results. Some of these pitfalls that are considered especially important in applied econometrics are heteroskedasticity or serial correlation of the error terms, structural changes in the regression coefficients, nonlinearities, functional misspecification or omitted variables. Therefore, a rich variety of diagnostic tests for these situations have been developed in the econometrics community, a collection of which has been implemented in the packages **lmtest**