Package 'distillery'

April 17, 2020

Version 1.0-7

Date 2020-04-12

Title Method Functions for Confidence Intervals and to Distill Information from an Object

Author Eric Gilleland

Maintainer Eric Gilleland <ericg@ucar.edu>

Depends R (>= 2.10.0)

Description Some very simple method functions for confidence interval calculation, bootstrap resampling aimed at atmospheric science applications, and to distill pertinent information from a potentially complex object; primarily used in common with packages extRemes and SpatialVx.

License GPL (>= 2)

URL http://www.ral.ucar.edu/staff/ericg

NeedsCompilation no

Repository CRAN

Date/Publication 2020-04-17 04:40:02 UTC

R topics documented:

distillery-package *distillery: Methods to Distill Information from R Objects*

Description

distillery contains primarily method functions to distill out pertinent information from R objects, as well as to compute confidence intervals. It now also contains new fairly general bootstrap functions.

Details

Primary functions include:

`distill`: Typically, to distill pertinent information from a complicated (usually a list) object and return a named vector.

`ci`: Calculate confidence intervals. This is a method function for calculating confidence intervals. Includes methods for numeric vectors and matrices, whereby the mean is taken (column-wise for matrices) and normal approximation confidence intervals for the mean are calculated and returned.

`booter`, `pbooter` and `tibber`: Functions to perform bootstrap resampling that work with `ci` (`booter` and `pbooter`). Allows for m < n bootstrap resampling, circular block bootstrapping, parametric bootstrap resampling (`pbooter`), and the test-inversion bootstrap approach (`tibber`).

Author(s)

Eric Gilleland

Examples

```
## See help files for above named functions and datasets
## for specific examples.
```

booter *Bootstrap Resampling*

Description

Generate B bootstrap replicates of size rsize and apply a statistic to them. Can do IID or Circular Block Bootstrap (CBB) methods.

Usage

```
booter(x, statistic, B, rsize, block.length = 1, v.terms, shuffle = NULL,
    replace = TRUE, ...)
```

Arguments

x	Original data series. May be a vector, matrix or data frame object.
statistic	Function that minimally takes arguments: data and .... The argument data must be the input data for which resamples are taken. Must return a vector of all desired statistics.
B	number of bootstrap resamples to make.
rsize	Number giving the resample size for each bootstrap sample. Must be between 1 and the length of x, if x is a vector, else the number of rows of x. Default is to use the size of the original data.
block.length	Number giving the desired block lengths. Default (block.length = 1) is to do IID resamples. Should be longer than the length of dependence in the data, but much shorter than the size of the data.
replace	logical, should the resamples be taken with replacement?
v.terms	If statistic returns variance estimates for other parameters, then use this argument to specify the indices returned that give the variance estimates. There must be a component for every other parameter returned, and they must be in the same order as the other parameters (see examples below). If an estimate does not exist, an NA should be returned for that spot.
shuffle	rsize by B matrix giving the indices for each bootstrap replication. If provided, B may be missing.
...	Optional arguments passed to statistic.

Details

Similar functionality to boot from package **boot**, but allows for easier implementation of certain other approaches. For example, m-out-of-n bootstrap resampling (appropriate for heavy-tail distributed data) can be performed via the rsize argument. The ci function is used to obtain subsequent confidence limits. For parameteric bootstrap resampling, see pbooter.

For more complicated bootstrap resampling, e.g., Bayesian bootstrap sampling, the shuffle argument may prove useful. That is, no weighting is allowed with this function through the standard mechanism, but the same result may be obtained by supplying your own indices through the shuffle argument. For parametric bootstrap resampling, see the pbooter function, but for certain types of parametric resampling, the shuffle argument could prove useful.

If the block length is > 1, then rsize overlapping blocks of this length are sampled from the data. In order to minimize over or under sampling of the end points, the blocks are circular (cf. Lahiri 2003).

Many good books and other materials are available about bootstrap resampling. One good text on IID bootstrap resampling is Efron and Tibshirani (1998) and for the block bootstrap, Lahiri (2003).

Value

A list object of class "booted" is returned with components:

call	the function call
data	original data series

`statistic`	statistic argument passed in
`statistic.args`	all other arguments passed by ...
`B`	Number of bootstrap replicate samples
`block.length`	The block length used
`replace`	logical stating whether the samples are taken with replacement or not.
`v.terms`	if variance terms are returned by statistic, the argument is repeated in the returned object.
`rsize`	the size of the bootstrap resamples.
`indices`	rsize by B matrix giving the resample indices used (rows) for each bootstrap resample (columns).
`v`	B length vector or B column matrix (if statistic returns a vector) giving the estimated parameter variances for each bootstrap replicate.
`orig.v`	vector giving the parameter variances (i.e. se^2) of statistic when applied to the original data.
`original.est`	vector giving the estimated parameter values when statistic is applied to the original data.
`results`	B length vector or B column matrix giving the parameter estimates for each bootstrap resample.
`type`	character stating whether the resample method is iid or cbb.

Author(s)

Eric Gilleland

References

Efron, B. and Tibshirani, R. J. (1998) *An Introduction to the Bootstrap*. Chapman \& Hall, Boca Raton, Florida, 436 pp.

Lahiri, S. N. (2003) *Resampling Methods for Dependent Data*. Springer-Verlag, New York, New York, 374 pp.

See Also

`pbooter`, `ci.booted` `tibber`

Examples

```
z <- rnorm( 100 )

zfun <- function( data, ... ) {

    return( c( mean( data ), var( data ), mean( data^2 ), var( data^2 ) ) )

} # end of 'zfun' function.
```

```
res <- booter( x = z, statistic = zfun, B = 500, v.terms = c(2, 4) )

print( res )

ci( res )
```

---

ci *Find Confidence Intervals*

Description

Method function for finding confidence intervals.

Usage

```
ci(x, alpha = 0.05, ...)

## S3 method for class 'matrix'
ci(x, alpha = 0.05, ...)

## S3 method for class 'numeric'
ci(x, alpha = 0.05, ...)

## S3 method for class 'ci'
print(x, ...)
```

Arguments

x	ci: an R object that has a ci method function for it. print: output from ci.
alpha	number between zero and one giving the 1 - alpha confidence level.
...	Optional arguments depending on the specific method function. In the case of those for ci.matrix and ci.numeric, these are any optional arguments to mean and var. Not used by print method function.

Details

ci.numeric: Calculates the mean and normal approximation CIs for the mean.

ci.matrix: Does the same as ci.numeric, but applies to each column of x.

Value

ci.numeric: a numeric vector giving the CI bounds and mean value.

ci.matrix: a matrix giving the mean and CI bounds for each column of x.

Author(s)

Eric Gilleland

Examples

```
ci(rnorm(100, mean=10, sd=2))

ci(matrix(rnorm(10000, mean=40, sd=10), 100, 100))
```

---

ci.booted *Bootstrap Confidence Intervals*

Description

Calculate confidence intervals for objects output from the booter and pbooter functions.

Usage

```
## S3 method for class 'booted'
ci(x, alpha = 0.05, ..., type = c("perc", "basic", "stud", "bca", "norm"))
```

Arguments

x	object of class "booted" as returned by the booter or pbooter function.
alpha	Significance level for which the (1 - alpha) * 100 percent confidence intervals are determined.
...	Not used.
type	character stating which intervals are to be reutrned. Default will do them all.

Details

Many methods exist for sampling parameters associated with a data set, and many methods for calculating confidence intervals from those resamples are also available. Some points to consider when using these methods are the accuracy of the intervals, and whether or not they are range-preserving and/or transformation-respecting. An interval that is range-preserving means that if a parameter can only take on values within a specified range, then the end points of the interval will also fall within this range. Transformation-respecting means that if a parameter, say phi, is transformed by a monotone function, say m(phi), then the (1 - alpha) * 100 percent confidence interval for m(phi) can be derived by applying m() to the limits of the (1 - alpha) * 100 percent interval for phi. That is [L(phi), U(phi)] = [m(L(phi)), m(U(phi))].

For accuracy, a (1 - 2 * alpha) * 100 percent confidence interval, (L, U), is presumed to have probability alpha of not covering the true value of the parameter from above or below. That is, if theta is the true value of the parameter, then Pr(theta < L) = alpha, and Pr(theta > U) = alpha. A second-order accurate interval means that the error in these probabilities tends to zero at a rate that is inversely proportional to the sample size. On the other hand, first-order accuracy means that

the error tends to zero more slowly, at a rate inversely proportional to the square root of the sample size.

the types of intervals available, here, are described below along with some considerations for their use.

Percentile intervals (`type` = "perc") are 1st order accurate, range-preserving, and transformation-respecting. However, they may have poor coverage in some situations. They are given by (L, U) where L and U are the 1 - alpha / 2 and alpha / 2 quantiles of the non-parametric distribution obtained through bootstrap resampling.

The basic interval (`type` = "basic") is the originally proposed interval and is given by (2 * theta - U, 2 * theta - L), where U and L are as for the percentile interval. This interval is 1st order accurate, but is not range-preserving or transformation-respecting.

Studentized (or Bootstrap-t) intervals (`type` = "stud") are 2nd order accurate, but not range-preserving or transformation-respecting, and they can be erratic for small samples, as well as sensitive to outliers. They are obtained by the basic bootstrap, but where U and L are taken from the studentized version of the resampled parameter estimates. That is, T' is taken for each bootstrap replicate, b, to be:

T'(b) = (theta'(b) - theta) / (se'(b)), where theta'(b) and se'(b) are the estimated value of the parameter and its estimated standard error, resp., for bootstrap replicate b, and theta is the estimated parameter value using the original data.

The bias-corrected and accelerated (BCa, `type` = "bca") method applies a bias correction and adjustment to the percentile intervals. The intervals are 2nd order accurate, range-preserving and transformation-respecting. However, the estimation performed, here (Eq 14.15 in Efron and Tibshirani 1998), requires a further jacknife resampling estimation, so the computational burden can be more expensive. The estimates for the bias-correction and acceleration adjustment can be found in Efron and Tibshirani (1998) p. 178 to 201. The bias-correction factor includes an adjustment for ties.

Finally, the normal approximation interval (`type` = "norm") uses the average of the estimated parameters from the bootstrap replicates, call it m, and their standard deviation, call is s, to make the usual normal approximation interval. An assumption of normality for the parameter estimates is assumed, which means that they will be symmetric. This method yields 1st order accurate intervals that are not range-preserving or transformation-respecting.

Value

A list object of class "ci.booted" is returned with components depending on which types of intervals are calculated.

`booted.object` The object passed through the x argument.

`perc, basic, stud, bca, norm`
: vectors of length 3 or 3-column matrices giving the intervals and original parameter estimates for each CI method.

`bias.correction, accelerated`
: If type includes "bca", then the estiamted bias correction factor and acceleration are given in these components.

Author(s)

Eric Gilleland

References

Efron, B. and Tibshirani, R. J. (1998) *An Introduction to the Bootstrap*. Chapman \& Hall, Boca Raton, Florida, 436 pp.

See Also

booter, pbooter

Examples

```
##
## See the help file for booter and/or pbooter for examples.
##
```

`datagrabber` *Get Original Data from an R Object*

Description

Get the original data set used to obtain the resulting R object for which a method function exists.

Usage

```
datagrabber(x, ...)
```

Arguments

`x`	An R object that has a method function for `datagrabber`.
`...`	Not used.

Details

Often when applying functions to data, it is handy to be able to grab the original data for subsequent routines (e.g., plotting, etc.). In some cases, information about where to obtain the original data might be available (more difficult) and in other cases, the data may simply be contained within a fitted object. This method function is generic, but some packages (e.g., **extRemes** >= 2.0, **SpatialVx** >= 1.0) have datagrabber functions specific to particular object types.

Value

The original pertinent data in whatever form it takes.

Author(s)

Eric Gilleland

Examples

```
## Not run:
## From the extRemes (>= 2.0) package.
y <- rnorm(100, mean=40, sd=20)
y <- apply(cbind(y[1:99], y[2:100]), 1, max)
bl <- rep(1:3, each=33)

ydc <- decluster(y, quantile(y, probs=c(0.95)), r=1, blocks=bl)

yorig <- datagrabber(ydc)
all(y - yorig == 0)


## End(Not run)
```

`distill` *Distill An Object*

Description

Distill a complex object to something easier to manage, like a numeric vector.

Usage

```
distill(x, ...)

## S3 method for class 'list'
distill(x, ...)

## S3 method for class 'matrix'
distill(x, ...)

## S3 method for class 'data.frame'
distill(x, ...)
```

Arguments

`x`	A list, vector, matrix or data frame, or other object that has a `distill` method, e.g., `fevd` objects.
`...`	Not used.

Details

Perhaps a fine line exists between functions such as `c`, `print`, `str`, `summary`, etc. The idea behind the `distill` method is to have a function that "distills" out the most pertinent information from a more complex object. For example, when fitting a model to a number of spatial locations, it can be

useful to pull out only certain information into a vector for ease of analysis. With many models, it might not be feasible to store (or analyze) large complicated data objects. In such a case, it may be useful to keep only a vector with the most pertinent information (e.g., parameter estimates, their standard errors, the likelihood value, AIC, BIC, etc.). For example, this is used within **extRemes** >= 2.0 on the "fevd" class objects with the aim at fitting models to numerous locations within an `apply` call so that something easily handled is returned, but with enough information as to be useful.

The data frame and matrix methods attempt to name each component of the vector. The list method simply does `c(unlist(x))`.

Value

numeric vector, possibly named.

Author(s)

Eric Gilleland

See Also

`c`, `unlist`, `print`, `summary`, `str`, `args`

Examples

```
x <- cbind(1:3, 4:6, 7:9)
distill(x)

x <- data.frame(x=1:3, y=4:6, z=7:9)
distill(x)
```

---

is.even *Identify Even or Odd Numbers*

Description

Simple functions to test for or return the even or odd numbers.

Usage

```
is.even(x)
is.odd(x)
even(x)
odd(x)
```

Arguments

x	any numeric, but maybe makes the most sense with integers.

Details

Return a logical vector/matrix of the same dimension as the argument `x` telling whether each component is odd (`is.odd`) or even (`is.even`), or return just the even (`even`) or odd (`odd`) numbers from the vector/matrix. Uses `%%`.

Value

Returns a logical vector/matrix/array of the same dimension as x in the case of is.even and is.odd, and returns a vector of length less than or equal to x in the case of even and odd; or if no even/odd values, returns integer(0).

Author(s)

Eric Gilleland

See Also

`%%`

Examples

```
is.even( 1:7 )
is.odd( 1:7 )
even( 1:7 )
odd( 1:7 )
```

`is.formula` *Is the R Object a Formula*

Description

Tests to see if an object is a formula or not.

Usage

```
is.formula(x)
```

Arguments

`x`	An R object.

Details

This function is a very simple one that simplifies checking whether or not the class of an object is a formula or not.

Value

single logical

Author(s)

Eric Gilleland

Examples

```
is.formula(~1)
is.formula(1:3)
```

`pbooter` *Parametric Bootstrap Resampling*

Description

Creates sample statistics for several replicated samples derived by sampling from a parametric distribution.

Usage

```
pbooter(x, statistic, B, rmodel, rsize, v.terms, verbose = FALSE, ...)
```

Arguments

`x`	Original data set. If it is a vector, then it is assumed to be univariate. If it is a matrix, it is assumed to be multivariate where each column is a variate.
`statistic`	Function that minimally takes arguments: `data` and `....` The argument `data` must be the input data for which resamples are taken. Must return a vector of all desired statistics.
`B`	number of bootstrap resamples to make.
`rmodel`	Function that generates the data to be applied to statistic. Must have arguments `size`, giving the size of the data to be returned, and `....`
`rsize`	Number giving the resample size for each bootstrap sample. If missing and `x` is a vector, it will be the length of `x`, and if it is a matrix, it will be the number of rows of `x`.
`v.terms`	If `statistic` returns variance estimates for other parameters, then use this argument to specify the indices returned that give the variance estimates. There must be a component for every other parameter returned, and they must be in the same order as the other parameters (see examples below). If an estimate does not exist, an NA should be returned for that spot.
`verbose`	logical, should progress information be printed to the screen?
`...`	Optional arguments to `statistic` or `rmodel`.

Details

Similar functionality to `boot` from **boot** when `sim` = "parametric". In this case, the function is a little simpler, and is intended for use with `ci.booted`, or just `ci`. It is similar to `booter`, but uses parametric sampling instead of resampling from the original data.

Value

A list object of class "booted" is returned with components:

`call`	the function call
`data`	original data series
`statistic`	statistic argument passed in
`statistic.args`	all other arguments passed by ...
`B`	Number of bootstrap replicate samples
`v.terms`	if variance terms are returned by statistic, the argument is repeated in the returned object.
`rsize`	the size of the bootstrap resamples.
`rdata`	rsize by B matrix giving the rmodel generated data.
`v`	B length vector or B column matrix (if statistic returns a vector) giving the estimated parameter variances for each bootstrap replicate.
`orig.v`	vector giving the parameter variances (i.e. se^2) of statistic when applied to the original data.
`original.est`	vector giving the estimated parameter values when statistic is applied to the original data.
`results`	B length vector or B column matrix giving the parameter estimates for each bootstrap resample.
`type`	character stating whether the resample method is iid or cbb.

Author(s)

Eric Gilleland

References

Efron, B. and Tibshirani, R. J. (1998) *An Introduction to the Bootstrap*. Chapman \& Hall, Boca Raton, Florida, 436 pp.

See Also

`booter`, `ci.booted` `tibber`

Examples

```
z <- rnorm( 100 )

zfun <- function( data, ... ) {

    return( c( mean( data ), var( data ), mean( data^2 ), var( data^2 ) ) )

} # end of 'zfun' function.

rfun <- function( size, ... ) rnorm( size, ... )
```

```
res <- pbooter( x = z, statistic = zfun, rmodel = rfun, B = 500,
    rsize = 100, v.terms = c(2, 4) )

print( res )

ci( res )
```

---

tibber *Test-Inversion Bootstrap*

Description

Calculate (1 - alpha) * 100 percent confidence intervals for an estimated parameter using the test-inversion bootstrap method.

Usage

```
tibber(x, statistic, B, rmodel, test.pars, rsize, block.length = 1, v.terms,
    shuffle = NULL, replace = TRUE, alpha = 0.05, verbose = FALSE, ...)

tibberRM(x, statistic, B, rmodel, startval, rsize, block.length = 1,
    v.terms, shuffle = NULL, replace = TRUE, alpha = 0.05, step.size,
    tol = 1e-04, max.iter = 1000, keep.iters = TRUE, verbose = FALSE,
    ...)
```

Arguments

x	numeric vector or data frame giving the original data series.
statistic	function giving the estimated parameter value. Must minimally contain arguments data and ....
B	number of replicated bootstrap samples to use.
rmodel	function that simulates data based on the nuisance parameter provided by test.pars. Must minimally take arguments: data, par, n, and .... The first, data, is the data series (it need not be used by the function, but it must have this argument, and the original data are passed to it via this argument), par is the nuisance parameter, n is the sample size, and ... are any additional arguments that might be needed.
test.pars	single number or vector giving the nuisance parameter value. If a vector of length greater than one, then the interpolation method will be applied to estimate the confidence bounds.
startval	one or two numbers giving the starting value for the nuisance parameter in the Robbins-Monro algorithm. If two numbers are given, the first is used as the starting value for the lower bound, and the second for the upper.

rsize	(optional) numeric less than the length of the series given by x, used if an m-out-of-n bootstrap sampling procedure should be used.
block.length	(optional) length of blocks to use if the circular block bootstrap resampling scheme is to be used (default is iid sampling).
v.terms	(optional) gives the positions of the variance estimate in the output from statistic. If supplied, then Studentized intervals are returned instead of (tibberRM) of in addition to (tibber) the regular intervals. Generally, such intervals are not ideal for the test-inversion method.
shuffle	n (or rsize) by B matrix giving the indices for the resampling procedure (obviates arguments block.length and B).
replace	logical stating whether or not to sample with replacement.
alpha	significance level for the test.
step.size	Step size for the Robbins-Monro algorithm.
tol	tolerance giving the value for how close the estimated p-value needs to be to alpha before stopping the Robbins-Monro algorithm.
max.iter	Maximum number of iterations to perform before stopping the Robbins-Monro algorithm.
keep.iters	logical, should information from each iteration of the Robbins-Monro algorithm be saved?
verbose	logical should progress information be printed to the screen.
...	Optional arguments to booter, statistic and rmodel.

Details

The test-inversion bootstrap (Carpenter 1999; Carpenter and Bithell 2000; Kabaila 1993) is a parametric bootstrap procedure that attempts to take advantage of the duality between confidence intervals and hypothesis tests in order to create bootstrap confidence intervals. Let X = X_1,...,X_n be a series of random variables, T, is a parameter of interest, and R(X) is an estimator for T. Further, let x = x_1,...,x_n be an observed realization of X, and r(x) an estimate for R(X), and let x* be a bootstrap resample of x, etc. Suppose that X is distributed according to a distribution, F, with parameter T and nuisance parameter V.

The procedure is carried out by estimating the p-value, say p*, from r*_1, ..., r*_B estimated from a simulated sample from rmodel assuming a specific value of V by way of finding the sum of r*_i < r(x) (with an additional correction for the ties r*_i = r(x)). The procedure is repeated for each of k values of V to form a sample of p-values, p*_1, ..., p*_k. Finally, some form of root-finding algorithm must be employed to find the values r*_L and r*_U that estimate the lower and upper values, resp., for R(X) associated with (1 - alpha) * 100 percent confidence limits. For tibber, the routine can be executed one time if test.pars is of length one, which will enable a user to employ their own root-finding algorithm. If test.pars is a vector, then an interpolation estimate is found for the confidence end points. tibberRM makes successive calls to tibber and uses the Robbins-Monro algorithm (Robbins and Monro 1951) to try to find the appropriate bounds, as suggested by Garthwaite and Buckland (1992).

Value

For tibber, if test.pars is of length one, then a 3 by 1 matrix is returned (or, if `v.terms` is supplied, then a 4 by 1 matrix) where the first two rows give estimates for R(X) based on the original simulated series and the median from the bootstrap samples, respectively. the last row gives the estimated p-value. If `v.terms` is supplied, then the fourth row gives the p-value associated with the Studentized p-value.

If test.pars is a vector with length k > 1, then a list object of class "tibbed" is returned, which has components:

`results` 3 by k matrix (or 4 by k, if `v.terms` is not missing) giving two estimates for R(X) (one from the simulated series and one of the median of the bootstrap resamples, resp.) and the third row giving the estimated p-value for each value of V.

`TIB.interpolated, STIB.interpolated` numeric vector of length 3 giving the lower bound estimate, the estimate from the original data (i.e., r(x)), and the estimated upper bound as obtained from interpolating over the vector of possible values for V given by test.pars. The Studentized TIB interval, `STIB.interpolated`, is only returned if `v.terms` is provided.

`Plow, Pup, PstudLow, PstudUp` Estimated p-values used for interpolation of p-value.

`call` the original function call.

`data` the original data passed by the x argument.

`statistic, B, rmodel, test.pars, rsize, block.length, alpha, replace` arguments passed into the orignal function call.

`n` original sample size.

`total.time` Total time it took for the function to run.

For tibberRM, a list of class "tibRMed" is returned with components:

`call` the original function call.

`x, statistic, B, rmodel, rsize, block.length, alpha, replace` arguments passed into the orignal function call.

`result` vector of length 3 giving the estimated confidence interval with the original parameter estimate in the second component.

`lower.p.value, upper.p.value` Estimated achieved p-values for the lower and upper bounds.

`lower.nuisance.par, upper.nuisance.par` nuisance parameter values associated with the lower and upper bounds.

`lower.iterations, upper.iterations` number of iterations of the Robbins-Monro algorithm it took to find the lower and upper bounds.

`total.time` Total time it took for the function to run.

Author(s)

Eric Gilleland

References

Carpenter, James (1999) Test inversion bootstrap confidence intervals. *J. R. Statist. Soc. B*, **61** (1), 159–172.

Carpenter, James and Bithell, John (2000) Bootstrap confidence intervals: when, which, what? A practical guide for medical statisticians. *Statist. Med.*, **19**, 1141–1164.

Garthwaite, P. H. and Buckland, S. T. (1992) Generating Monte Carlo confidence intervals by the Robbins-Monro process. *Appl. Statist.*, **41**, 159–171.

Kabaila, Paul (1993) Some properties of profile bootstrap confidence intervals. *Austral. J. Statist.*, **35** (2), 205–214.

Robbins, Herbert and Monro, Sutton (1951) A stochastic approximation method. *Ann. Math Statist.*, **22** (3), 400–407.

See Also

booter, pbooter

Examples

```
# The following example follows the example provided at:
#
# http://influentialpoints.com/Training/bootstrap_confidence_intervals.htm
#
# which is provided with a creative commons license:
#
# https://creativecommons.org/licenses/by/3.0/
#
y <- c( 7, 7, 6, 9, 8, 7, 8, 7, 7, 7, 6, 6, 6, 8, 7, 7, 7, 7, 6, 7,
        8, 7, 7, 6, 8, 7, 8, 7, 8, 7, 7, 7, 5, 7, 7, 7, 6, 7, 8, 7, 7,
        8, 6, 9, 7, 14, 12, 10, 13, 15 )

trm <- function( data, ... ) {

    res <- try( mean( data, trim = 0.1, ... ) )
    if( class( res ) == "try-error" ) return( NA )
    else return( res )

} # end of 'trm' function.

genf <- function( data, par, n, ... ) {

    y <- data * par
    h <- 1.06 * sd( y ) / ( n^( 1 / 5 ) )
    y <- y + rnorm( rnorm( n, 0, h ) )
    y <- round( y * ( y > 0 ) )

    return( y )

} # end of 'genf' function.

look <- tibber( x = y, statistic = trm, B = 500, rmodel = genf,
```

```
    test.pars = seq( 0.85, 1.15, length.out = 100 ) )

look

plot( look )
# outer vertical blue lines should cross horizontal blue lines
# near where an estimated p-value is located.

tibber( x = y, statistic = trm, B = 500, rmodel = genf, test.pars = 1 )


## Not run:
look2 <- tibberRM(x = y, statistic = trm, B = 500, rmodel = genf, startval = 1,
    step.size = 0.03, verbose = TRUE )

look2
# lower achieved est. p-value should be close to 0.025
# upper should be close to 0.975.

plot( look2 )

trm2 <- function( data, par, n, ... ) {

    a <- list( ... )
    res <- try( mean( data, trim = a$trim ) )
    if( class( res ) == "try-error" ) return( NA )
    else return( res )

} # end of 'trm2' function.

tibber( x = y, statistic = trm2, B = 500, rmodel = genf,
    test.pars = seq( 0.85, 1.15, length.out = 100 ), trim = 0.1 )

# Try getting the STIB interval.  v.terms = 2 below because mfun
# returns the variance of the estimated parameter in the 2nd position.
#
# Note: the STIB interval can be a bit unstable.

mfun <- function( data, ... ) return( c( mean( data ), var( data ) ) )

gennorm <- function( data, par, n, ... ) {

    return( rnorm( n = n, mean = mean( data ), sd = sqrt( par ) ) )

} # end of 'gennorm' function.

set.seed( 1544 )
z <- rnorm( 50 )
mean( z )
var( z )

# Trial-and-error is necessary to get a good result with interpolation method.
res <- tibber( x = z, statistic = mfun, B = 500, rmodel = gennorm,
```

```
    test.pars = seq( 0.95, 1.10, length.out = 100 ), v.terms = 2 )

res

plot( res )

# Much trial-and-error is necessary to get a good result with RM method.
# If it fails to converge, try increasing the tolerance.
res2 <- tibberRM( x = z, statistic = mfun, B = 500, rmodel = gennorm,
    startval = c( 0.95, 1.1 ), step.size = 0.003, tol = 0.001, v.terms = 2,
    verbose = TRUE )
# Note that it only gives the STIB interval.

res2

plot( res2 )


## End(Not run)
```

Index