Package 'fwb'

June 12, 2025

Type Package

Title Fractional Weighted Bootstrap

Version 0.4.0

Description
An implementation of the fractional weighted bootstrap to be used as a drop-in for functions in the 'boot' package. The fractional weighted bootstrap (also known as the Bayesian bootstrap) involves drawing
weights randomly that are applied to the data rather than resampling units from the data. See Xu et al. (2020)
<doi:10.1080/00031305.2020.1731599> for details.

Depends R (>= 3.6.0)

Imports rlang (>= 1.1.6), chk (>= 0.10.0), pbapply (>= 1.7-2),
graphics, stats, utils

Suggests survival, cobalt, boot (>= 1.3-31), sandwich (>= 2.4-0),
ggdist (>= 3.3.3), lmtest, nnet, parallel, future,
future.apply, testthat (>= 3.2.3), waldo (>= 0.6.1), knitr,
rmarkdown

License GPL (>= 2)

Encoding UTF-8

URL https://ngreifer.github.io/fwb/, https://github.com/ngreifer/fwb,

BugReports https://github.com/ngreifer/fwb/issues

RoxygenNote 7.3.2

LazyData true

Config/testthat/edition 3

Config/testthat/parallel false

VignetteBuilder knitr

NeedsCompilation no

Author Noah Greifer [aut, cre] (ORCID:
<https://orcid.org/0000-0003-3067-7154>)

Maintainer Noah Greifer <noah.greifer@gmail.com>

Repository CRAN
Date/Publication 2025-06-12 00:10:02 UTC

Contents

`bearingcage` *Bearing Cage field failure data*

Description

The data consist of 1703 aircraft engines put into service over time. There were 6 failures and 1697 right-censored observations. These data were originally given in Abernethy et al. (1983) and were reanalyzed in Meeker and Escobar (1998, chap.8). The dataset used here specifically comes from Xu et al. (2020) and are used in a Weibull analysis of failure times.

Usage

```
data("bearingcage")
```

Format

A data frame with 1703 rows and 2 variables:

`hours` integer; the number of hours until failure or censoring

`failure` logical; whether a failure occurred

References

Abernethy, R. B., Breneman, J. E., Medlin, C. H., and Reinman, G. L. (1983), "Weibull Analysis Handbook," Technical Report, Air Force Wright Aeronautical Laboratories, available at https://apps.dtic.mil/sti/citations/ADA143100.

Meeker, W. Q., and Escobar, L. A. (1998), *Statistical Methods for Reliability Data*, New York: Wiley.

Xu, L., Gotwalt, C., Hong, Y., King, C. B., & Meeker, W. Q. (2020). Applications of the Fractional-Random-Weight Bootstrap. *The American Statistician*, 74(4), 345–358. doi:10.1080/00031305.2020.1731599

fwb *Fractional Weighted Bootstrap*

Description

fwb() implements the fractional (random) weighted bootstrap, also known as the Bayesian bootstrap. Rather than resampling units to include in bootstrap samples, weights are drawn to be applied to a weighted estimator.

Usage

```
fwb(
  data,
  statistic,
  R = 999,
  cluster = NULL,
  simple = NULL,
  wtype = getOption("fwb_wtype", "exp"),
  strata = NULL,
  drop0 = FALSE,
  verbose = TRUE,
  cl = NULL,
  ...
)

## S3 method for class 'fwb'
print(x, digits = getOption("digits"), index = 1L:ncol(x$t), ...)
```

Arguments

data	the dataset used to compute the statistic
statistic	a function, which, when applied to data, returns a vector containing the statistic(s) of interest. The function should take at least two arguments; the first argument should correspond to the dataset and the second argument should correspond to a vector of weights. Any further arguments can be passed to statistic through the ... argument.
R	the number of bootstrap replicates. Default is 999 but more is always better. For the percentile bootstrap confidence interval to be exact, it can be beneficial to use one less than a multiple of 100.
cluster	optional; a vector containing cluster membership. If supplied, will run the cluster bootstrap. See Details. Evaluated first in data and then in the global environment.
simple	logical; if TRUE, weights will be generated on-the-fly in each bootstrap replication; if FALSE, all weights will be generated at once and then supplied to statistic. Cannot be TRUE when wtype = "multinom". The default (NULL) sets to FALSE if wtype = "multinom" and to TRUE otherwise.

wtype	string; the type of weights to use. Allowable options include "exp" (the default), "pois", "multinom", and "mammen". See Details. See set_fwb_wtype() to set a global default.
strata	optional; a vector containing stratum membership for stratified bootstrapping. If supplied, will essentially perform a separate bootstrap within each level of strata. This does not affect results when wtype = "poisson".
drop0	logical; when wtype is "multinom" or "poisson", whether to drop units that are given weights of 0 from the dataset and weights supplied to statistic in each iteration. Ignored for other wtypes because they don't produce 0 weights. Default is FALSE.
verbose	logical; whether to display a progress bar.
cl	a cluster object created by parallel::makeCluster(), an integer to indicate the number of child-processes (integer values are ignored on Windows) for parallel evaluations, or the string "future" to use a future backend. See the cl argument of pbapply::pblapply() for details. If NULL, no parallelization will take place. See vignette("fwb-rep") for details.
...	other arguments passed to statistic.
x	an fwb object; the output of a call to fwb().
digits	the number of significant digits to print
index	the index or indices of the position of the quantity of interest in x$t0 if more than one was specified in fwb(). Default is to print all quantities.

Details

fwb() implements the fractional weighted bootstrap and is meant to function as a drop-in for boot::boot(., stype = "f") (i.e., the usual bootstrap but with frequency weights representing the number of times each unit is drawn). In each bootstrap replication, when wtype = "exp" (the default), the weights are sampled from independent exponential distributions with rate parameter 1 and then normalized to have a mean of 1, equivalent to drawing the weights from a Dirichlet distribution. Other weights are allowed as determined by the wtype argument (see below for details). The function supplied to statistic must incorporate the weights to compute a weighted statistic. For example, if the output is a regression coefficient, the weights supplied to the w argument of statistic should be supplied to the weights argument of lm(). These weights should be used any time frequency weights would be, since they are meant to function like frequency weights (which, in the case of the traditional bootstrap, would be integers). Unfortunately, there is no way for fwb() to know whether you are using the weights correctly, so care should be taken to ensure weights are correctly incorporated into the estimator.

When fitting binomial regression models (e.g., logistic) using glm(), it may be useful to change the family to a "quasi" variety (e.g., quasibinomial()) to avoid a spurious warning about "non-integer #successes".

The cluster bootstrap can be requested by supplying a vector of cluster membership to cluster. Rather than generating a weight for each unit, a weight is generated for each cluster and then applied to all units in that cluster.

Bootstrapping can be performed within strata by supplying a vector of stratum membership to strata. This essentially rescales the weights within each stratum to have a mean of 1, ensuring

that the sum of weights in each stratum is equal to the stratum size. For multinomial weights, using strata is equivalent to drawing samples with replacement from each stratum. Strata do not affect bootstrapping when using Poisson weights.

Ideally, `statistic` should not involve a random element, or else it will not be straightforward to replicate the bootstrap results using the `seed` included in the output object. Setting a seed using `set.seed()` is always advised. See `vignette("fwb-rep")` for details.

The `print()` method displays the value of the statistics, the bias (the difference between the statistic and the mean of its bootstrap distribution), and the standard error (the standard deviation of the bootstrap distribution).

Weight types:

Different types of weights can be supplied to the `wtype` argument. A global default can be set using `set_fwb_wtype()`. The allowable weight types are described below.

`"exp"` Draws weights from an exponential distribution with rate parameter 1 using `rexp()`. These weights are the usual "Bayesian bootstrap" weights described in Xu et al. (2020). They are equivalent to drawing weights from a uniform Dirichlet distribution, which is what gives these weights the interpretation of a Bayesian prior. The weights are scaled to have a mean of 1 within each stratum (or in the full sample if `strata` is not supplied).

`"multinom"` Draws integer weights using `sample()`, which samples unit indices with replacement and uses the tabulation of the indices as frequency weights. This is equivalent to drawing weights from a multinomial distribution. Using `wtype = "multinom"` is the same as using `boot::boot(., stype = "f")` instead of `fwb()` (i.e., the resulting estimates will be identical). When `strata` is supplied, unit indices are drawn with replacement within each stratum so that the sum of the weights in each stratum is equal to the stratum size.

`"poisson"` Draws integer weights from a Poisson distribution with 1 degree of freedom using `rpois()`. This is an alternative to the multinomial weights that yields similar estimates (especially as the sample size grows) but can be faster. Note `strata` is ignored when using `"poisson"`.

`"mammen"` Draws weights from a modification of the distribution described by Mammen (1983) for use in the wild bootstrap. These positive weights have a mean, variance, and skewness of 1, making them second-order accurate (in contrast to the usual exponential weights, which are only first-order accurate). The weights w are drawn such that $P(w = (3 + \sqrt{5})/2) = (\sqrt{5} - 1)/2\sqrt{5}$ and $P(w = (3 - \sqrt{5})/2) = (\sqrt{5} + 1)/2\sqrt{5}$. The weights are scaled to have a mean of 1 within each stratum (or in the full sample if `strata` is not supplied).

`"exp"` is the default due to it being the formulation described in Xu et al. (2020) and in the most formulations of the Bayesian bootstrap; it should be used if one wants to remain in line with these guidelines or to maintain a Bayesian flavor to the analysis, whereas `"mammen"` might be preferred for its frequentist operating characteristics, though its performance has not been studied in this context. `"multinom"` and `"poisson"` should only be used for comparison purposes.

Value

An `fwb` object, which also inherits from `boot`, with the following components:

`t0`	The observed value of `statistic` applied to `data` with uniform weights.
`t`	A matrix with `R` rows, each of which is a bootstrap replicate of the result of calling `statistic`.

R	The value of R as passed to fwb().
data	The data as passed to fwb().
seed	The value of .Random.seed just prior to generating the weights (after the first call to statistic with uniform weights).
statistic	The function statistic as passed to fwb().
call	The original call to fwb().
cluster	The vector passed to cluster, if any.
strata	The vector passed to strata, if any.
wtype	The type of weights used as determined by the wtype argument.

fwb objects have coef() and vcov() methods, which extract the t0 component and covariance of the t components, respectively.

Methods (by generic)

- print(fwb): Print an fwb object

References

Mammen, E. (1993). Bootstrap and Wild Bootstrap for High Dimensional Linear Models. *The Annals of Statistics*, 21(1). doi:10.1214/aos/1176349025

Rubin, D. B. (1981). The Bayesian Bootstrap. *The Annals of Statistics*, 9(1), 130–134. doi:10.1214/aos/1176345338

Xu, L., Gotwalt, C., Hong, Y., King, C. B., & Meeker, W. Q. (2020). Applications of the Fractional-Random-Weight Bootstrap. *The American Statistician*, 74(4), 345–358. doi:10.1080/00031305.2020.1731599

The use of the "mammen" formulation of the bootstrap weights was suggested by Lihua Lei here.

See Also

fwb.ci() for calculating confidence intervals; summary.fwb() for displaying output in a clean way; plot.fwb() for plotting the bootstrap distributions; vcovFWB() for estimating the covariance matrix of estimates using the FWB; set_fwb_wtype() for an example of using weights other than the default exponential weights; boot::boot() for the traditional bootstrap.

See vignette("fwb-rep") for information on reproducibility.

Examples

```
# Performing a Weibull analysis of the Bearing Cage
# failure data as done in Xu et al. (2020)
set.seed(123, "L'Ecuyer-CMRG")
data("bearingcage")

weibull_est <- function(data, w) {
  fit <- survival::survreg(survival::Surv(hours, failure) ~ 1,
                           data = data, weights = w,
                           dist = "weibull")
```

```
  c(eta = unname(exp(coef(fit))), beta = 1/fit$scale)
}

boot_est <- fwb(bearingcage, statistic = weibull_est,
                R = 199, verbose = FALSE)
boot_est

#Get standard errors and CIs; uses bias-corrected
#percentile CI by default
summary(boot_est, ci.type = "bc")

#Plot statistic distributions
plot(boot_est, index = "beta", type = "hist")
```

`fwb.ci` *Fractional Weighted Bootstrap Confidence Intervals*

Description

`fwb.ci()` generates several types of equi-tailed two-sided nonparametric confidence intervals. These include the normal approximation, the basic bootstrap interval, the percentile bootstrap interval, the bias-corrected percentile bootstrap interval, and the bias-correct and accelerated (BCa) bootstrap interval.

Usage

```
fwb.ci(
  fwb.out,
  conf = 0.95,
  type = "bc",
  index = 1L,
  h = base::identity,
  hinv = base::identity,
  ...
)

## S3 method for class 'fwbci'
print(x, hinv = NULL, ...)
```

Arguments

`fwb.out`	an `fwb` object; the output of a call to `fwb()`.
`conf`	the desired confidence level. Default is .95 for 95% confidence intervals.
`type`	the type of confidence interval desired. Allowable options include `"norm"` (normal approximation), `"basic"` (basic interval), `"perc"` (percentile interval), `"bc"` (bias-correct percentile interval), and `"bca"` (BCa interval). More than one is allowed. Can also be `"all"` to request all of them. BCa intervals require that the number of bootstrap replications is larger than the sample size.

index	the index of the position of the quantity of interest in fwb.out$t0 if more than one was specified in fwb(). Only one value is allowed at a time. By default the first statistic is used.
h	a function defining a transformation. The intervals are calculated on the scale of h(t) and the inverse function hinv applied to the resulting intervals. It must be a function of one variable only and for a vector argument, it must return a vector of the same length. Default is the identity function.
hinv	a function, like h, which returns the inverse of h. It is used to transform the intervals calculated on the scale of h(t) back to the original scale. The default is the identity function. If h is supplied but hinv is not, then the intervals returned will be on the transformed scale.
...	ignored
x	an fwbci object; the output of a call to fwb.ci().

Details

fwb.ci() functions similarly to boot::boot.ci() in that it takes in a bootstrapped object and computes confidence intervals. This interface is a bit old-fashioned, but was designed to mimic that of boot.ci(). For a more modern interface, see summary.fwb().

The bootstrap intervals are defined as follows, with $\alpha = 1$ - conf, t_0 the estimate in the original sample, $\hat{t}$ the average of the bootstrap estimates, s_t the standard deviation of the bootstrap estimates, $t^{(i)}$ the set of ordered estimates with i corresponding to their quantile, and $z_{\frac{\alpha}{2}}$ and $z_{1-\frac{\alpha}{2}}$ the upper and lower critical z scores.

- "norm" (normal approximation): $[2t_0 - \hat{t} + s_t z_{\frac{\alpha}{2}}, 2t_0 - \hat{t} + s_t z_{1-\frac{\alpha}{2}}]$

This involves subtracting the "bias" $(\hat{t} - t_0)$ from the estimate t_0 and using a standard Wald-type confidence interval. This method is valid when the statistic is normally distributed.

- "basic": $[2t_0 - t^{(1-\frac{\alpha}{2})}, 2t_0 - t^{(\frac{\alpha}{2})}]$
- "perc" (percentile confidence interval): $[t^{(\frac{\alpha}{2})}, t^{(1-\frac{\alpha}{2})}]$
- "bc" (bias-corrected percentile confidence interval): $[t^{(l)}, t^{(u)}]$

$l = \Phi\left(2z_0 + z_{\frac{\alpha}{2}}\right)$, $u = \Phi\left(2z_0 + z_{1-\frac{\alpha}{2}}\right)$, where $\Phi(.)$ is the normal cumulative density function (i.e., pnorm()) and $z_0 = \Phi^{-1}(q)$ where q is the proportion of bootstrap estimates less than the original estimate t_0. This is similar to the percentile confidence interval but changes the specific quantiles of the bootstrap estimates to use, correcting for bias in the original estimate. It is described in Xu et al. (2020). When t^0 is the median of the bootstrap distribution, the "perc" and "bc" intervals coincide.

- "bca" (bias-corrected and accelerated confidence interval): $[t^{(l)}, t^{(u)}]$

$l = \Phi\left(z_0 + \frac{z_0 + z_{\frac{\alpha}{2}}}{1 - a(z_0 + z_{\frac{\alpha}{2}})}\right)$, $u = \Phi\left(z_0 + \frac{z_0 + z_{1-\frac{\alpha}{2}}}{1 - a(z_0 + z_{1-\frac{\alpha}{2}})}\right)$, using the same definitions as above, but with the additional acceleration parameter a, where $a = \frac{1}{6}\frac{\sum L^3}{(\sum L^2)^{3/2}}$. L is the empirical influence value of each unit, which is computed using the regression method described in boot::empinf(). When $a = 0$, the "bca" and "bc" intervals coincide. The acceleration parameter corrects for bias

and skewness in the statistic. It can only be used when clusters are absent and the number of bootstrap replications is larger than the sample size. Note that BCa intervals cannot be requested when `simple = TRUE` and there is randomness in thew `statistic` supplied to `fwb()`.

Interpolation on the normal quantile scale is used when a non-integer order statistic is required, as in `boot::boot.ci()`. Note that unlike with `boot::boot.ci()`, studentized confidence intervals (`type = "stud"`) are not allowed.

Value

An `fwbci` object, which inherits from `bootci` and has the following components:

`R`	the number of bootstrap replications in the original call to `fwb()`.
`t0`	the observed value of the statistic on the same scale as the intervals (i.e., after applying `h` and then `hinv`.
`call`	the call to `fwb.ci()`.

There will be additional components named after each confidence interval type requested. For `"norm"`, this is a matrix with one row containing the confidence level and the two confidence interval limits. For the others, this is a matrix with one row containing the confidence level, the indices of the two order statistics used in the calculations, and the confidence interval limits.

Functions

- `print(fwbci)`: Print a bootstrap confidence interval

See Also

`fwb()` for performing the fractional weighted bootstrap; `get_ci()` for extracting confidence intervals from an `fwbci` object; `summary.fwb()` for producing clean output from `fwb()` that includes confidence intervals calculated by `fwb.ci()`; `boot::boot.ci()` for computing confidence intervals from the traditional bootstrap; `vcovFWB()` for computing parameter estimate covariance matrices using the fractional weighted bootstrap

Examples

```
set.seed(123, "L'Ecuyer-CMRG")
data("infert")

fit_fun <- function(data, w) {
  fit <- glm(case ~ spontaneous + induced, data = data,
             family = "quasibinomial", weights = w)
  coef(fit)
}

fwb_out <- fwb(infert, fit_fun, R = 199,
               verbose = FALSE)

# Bias corrected percentile interval
bcci <- fwb.ci(fwb_out, index = "spontaneous",
               type = "bc")
```

```
bcci

# Using `get_ci()` to extract confidence limits

get_ci(bcci)

# Interval calculated on original (log odds) scale,
# then transformed by exponentiation to be on OR
fwb.ci(fwb_out, index = "induced", type = "norm",
       hinv = exp)
```

---

get_ci *Extract Confidence Intervals from a* bootci *Object*

Description

get_ci() extracts the confidence intervals from the output of a call to boot::boot.ci() or fwb.ci() in a clean way. Normally the confidence intervals can be a bit challenging to extract because of the unusual structure of the object.

Usage

```
get_ci(x, type = "all")
```

Arguments

x	a bootci object; the output of a call to boot::boot.ci() or fwb.ci().
type	the type of confidence intervals to extract. Only those available in x are allowed. Should be a given as a subset of the types passed to type in boot.ci() or fwb.ci(). The default, "all", extracts all confidence intervals in x.

Value

A list with an entry for each confidence interval type; each entry is a numeric vector of length 2 with names "L" and "U" for the lower and upper interval bounds, respectively. The "conf" attribute contains the confidence level.

See Also

fwb.ci(), confint.fwb(), boot::boot.ci()

Examples

```
#See example at help("fwb.ci")
```

plot.fwb *Plots of the Output of a Fractional Weighted Bootstrap*

Description

plot.fwb() takes an fwb object and produces plots for the bootstrap replicates of the statistic of interest.

Usage

```
## S3 method for class 'fwb'
plot(
  x,
  index = 1L,
  qdist = "norm",
  nclass = NULL,
  df,
  type = c("hist", "qq"),
  ...
)
```

Arguments

x	an fwb object; the output of a call to fwb().
index	the index of the position of the quantity of interest in x$t0 if more than one was specified in fwb(). Only one value is allowed at a time. By default the first statistic is used.
qdist	character; when a Q-Q plot is requested (as it is by default; see type argument below), the distribution against which the Q-Q plot should be drawn. Allowable options include "norm" (normal distribution - the default) and "chisq" (chi-squared distribution).
nclass	when a histogram is requested (as it is by default; see type argument below), the number of classes to be used. The default is the integer between 10 and 100 closest to ceiling(length(R)/25) where R is the number of bootstrap replicates.
df	if qdist is "chisq", the degrees of freedom for the chi-squared distribution to be used. If not supplied, the degrees of freedom will be estimated using maximum likelihood.
type	the type of plot to display. Allowable options include "hist" for a histogram of the bootstrap estimates and "qq" for a Q-Q plot of the estimates against the distribution supplied to qdist.
...	ignored.

Details

This function can produces two side-by-side plots: a histogram of the bootstrap replicates and a Q-Q plot of the bootstrap replicates against theoretical quantiles of a supplied distribution (normal or chi-squared). For the histogram, a vertical dotted line indicates the position of the estimate computed in the original sample. For the Q-Q plot, the expected line is plotted.

Value

x is returned invisibly.

See Also

fwb(), summary.fwb(), boot::plot.boot(), hist(), qqplot()

Examples

```
# See examples at help("fwb")
```

---

set_fwb_wtype *Set weights type*

Description

Set the default for the type of weights used in the weighted bootstrap computed by fwb() and vcovFWB().

Usage

```
set_fwb_wtype(wtype = getOption("fwb_wtype", "exp"))

get_fwb_wtype(fwb)
```

Arguments

wtype	string; the type of weights to use. Allowable options include "exp" (the default), "pois", "multinom", and "mammen". Abbreviations allowed. See fwb() for what these mean.
fwb	optional; an fwb object, the output of a call to fwb(). If left empty, will extract the weights type from options().

Details

set_fwb_wtype(x) is equivalent to calling options(fwb_wtype = x). get_fwb_wtype() is equivalent to calling getOption("fwb_wtype") when no argument is supplied and to extracting the wtype component of an fwb object when supplied.

Value

`set_fwb_wtype()` returns a call to `options()` with the options set to those prior to `set_fwb_wtype()` being called. This makes it so that calling `options(op)`, where `op` is the output of `set_fwb_wtype()`, resets the `fwb_wtype` to its original value. `get_fwb_wtype()` returns a string containing the `fwb_wtype` value set globally (if no argument is supplied) or used in the supplied `fwb` object.

See Also

`fwb()` for a definition of each types of weights; `vcovFWB()`; `options()`; `boot::boot()` for the traditional bootstrap.

Examples

```
# Performing a Weibull analysis of the Bearing Cage
# failure data as done in Xu et al. (2020)
set.seed(123, "L'Ecuyer-CMRG")
data("bearingcage")

#Set fwb type to "mammen"
op <- set_fwb_wtype("mammen")

weibull_est <- function(data, w) {
  fit <- survival::survreg(survival::Surv(hours, failure) ~ 1,
                           data = data, weights = w,
                           dist = "weibull")

  c(eta = unname(exp(coef(fit))), beta = 1/fit$scale)
}

boot_est <- fwb(bearingcage, statistic = weibull_est,
                R = 199, verbose = FALSE)
boot_est

#Get the fwb type used in the bootstrap
get_fwb_wtype(boot_est)
get_fwb_wtype()

#Restore original options
options(op)

get_fwb_wtype()
```

`summary.fwb` *Summarize* `fwb` *Output*

Description

summary() creates a regression summary-like table that displays the bootstrap estimates, their empirical standard errors, and their confidence intervals. confint() produces just the confidence intervals, which are computed using fwb.ci(), and is called internally by summary().

Usage

```
## S3 method for class 'fwb'
summary(
  object,
  conf = 0.95,
  ci.type = "bc",
  p.value = FALSE,
  index = 1L:ncol(object$t),
  ...
)

## S3 method for class 'fwb'
confint(object, parm, level = 0.95, ci.type = "bc", ...)
```

Arguments

object	an fwb object; the output of a call to fwb().
conf, level	the desired confidence level. Default is .95 for 95% confidence intervals.
ci.type	the type of confidence interval desired. Allowable options include "norm" (normal approximation), "basic" (basic interval), "perc" (percentile interval), "bc" (bias-correct percentile interval), and "bca" (bias-corrected and accelerated [BCa] interval). Only one is allowed. BCa intervals require that the number of bootstrap replications is larger than the sample size. See fwb.ci() for details. The default is "bc".
p.value	logical; whether to display p-values for the test that each parameter is equal to 0. The p-value is computed using a Z-test with the test statistic computed as the ratio of the estimate to its bootstrap standard error. This test is only valid when the bootstrap distribution is normally distributed around 0 and is not guaranteed to agree with any of the confidence intervals. Default is FALSE.
index, parm	the index or indices of the position of the quantity of interest in x$t0 if more than one was specified in fwb(). Default is to display all quantities.
...	ignored.

Value

For summary(), a summary.fwb object, which is a matrix with the following columns:

- Estimate: the statistic estimated in the original sample
- Std. Error: the standard deviation of the bootstrap estimates
- CI {L}% and CI {U}%, the upper and lower confidence interval bounds computed using the argument to ci.type.

When p.value = TRUE, two additional columns, z value and Pr(>|z|), are included containing the z-statistic and p-value for each computed statistic, respectively.

For confint(), a matrix with a row for each statistic and a column for the upper and lower confidence interval limits.

See Also

fwb() for performing the fractional weighted bootstrap; fwb.ci() for computing multiple confidence intervals for a single bootstrapped quantity

Examples

```
set.seed(123, "L'Ecuyer-CMRG")
data("infert")

fit_fun <- function(data, w) {
  fit <- glm(case ~ spontaneous + induced, data = data,
             family = "quasibinomial", weights = w)
  coef(fit)
}

fwb_out <- fwb(infert, fit_fun, R = 199,
               verbose = FALSE)

# Basic confidence interval for both estimates
summary(fwb_out, ci.type = "basic")

# Just for "induced" coefficient; p-values requested
summary(fwb_out, index = "induced", p.value = TRUE)
```

---

vcovFWB *Fractional Weighted Bootstrap Covariance Matrix Estimation*

Description

vcovFWB() estimates the covariance matrix of model coefficient estimates using the fractional weighted bootstrap. It serves as a drop-in for stats::vcov() or sandwich::vcovBS(). Clustered covariances are can be requested.

Usage

```
vcovFWB(
  x,
  cluster = NULL,
  R = 1000,
  start = FALSE,
  wtype = getOption("fwb_wtype", "exp"),
```

```
  ...,
  fix = FALSE,
  use = "pairwise.complete.obs",
  .coef = stats::coef,
  verbose = FALSE,
  cl = NULL
)
```

Arguments

x	a fitted model object, such as the output of a call to `lm()` or `glm()`. The model object must result from a function that can be updated using `update()` and has a `weights` argument to input non-integer case weights.
cluster	a variable indicating the clustering of observations, a `list` (or `data.frame`) thereof, or a formula specifying which variables from the fitted model should be used (see examples). By default (`cluster = NULL`), either `attr(x, "cluster")` is used (if any) or otherwise every observation is assumed to be its own cluster.
R	the number of bootstrap replications.
start	`logical`; should `.coef(x)` be passed as `start` to the `update(x, weights = ...)` call? In case the model `x` is computed by some numeric iteration, this may speed up the bootstrapping.
wtype	string; the type of weights to use. Allowable options include `"exp"` (the default), `"pois"`, `"multinom"`, and `"mammen"`. See `fwb()` for details. See `set_fwb_wtype()` to set a global default.
...	ignored.
fix	`logical`; if `TRUE`, the covariance matrix is fixed to be positive semi-definite in case it is not.
use	`character`; specification passed to `stats::cov()` for handling missing coefficients/parameters.
.coef	a function used to extract the coefficients from each fitted model. Must return a numeric vector. By default, `stats::coef` is used, but `marginaleffects::get_coef` can be a more reliable choice for some models that have a non-standard `coef()` method, like that for `nnet::multinom()` models.
verbose	`logical`; whether to display a progress bar.
cl	a cluster object created by `parallel::makeCluster()`, an integer to indicate the number of child-processes (integer values are ignored on Windows) for parallel evaluations, or the string `"future"` to use a `future` backend. See the `cl` argument of `pbapply::pblapply()` for details. If `NULL`, no parallelization will take place. See `vignette("fwb-rep")` for details.

Details

`vcovFWB()` functions like other `vcov()`-like functions, such as those in the **sandwich** package, in particular, `sandwich::vcovBS()`, which implements the traditional bootstrap (and a few other bootstrap varieties for linear models). Sets of weights are generated as described in the documentation

for fwb(), and the supplied model is re-fit using those weights. When the fitted model already has weights, these are multiplied by the bootstrap weights.

For lm objects, the model is re-fit using .lm.fit() for speed, and, similarly, glm objects are re-fit using glm.fit() (or whichever fitting method was originally used). For other objects, update() is used to populate the weights and re-fit the model (this assumes the fitting function accepts non-integer case weights through a weights argument). If a model accepts weights in some other way, fwb() should be used instead; vcovFWB() is inherently limited in its ability to handle all possible models. It is important that the original model was not fit using frequency weights (i.e., weights that allow one row of data to represent multiple full, identical, individual units) unless clustering is used.

See sandwich::vcovBS() and sandwich::vcovCL() for more information on clustering covariance matrices, and see fwb() for more information on how clusters work with the fractional weighted bootstrap. When clusters are specified, each cluster is given a bootstrap weight, and all members of the cluster are given that weight; estimation then proceeds as normal. By default, when cluster is unspecified, each unit is considered its own cluster.

Value

A matrix containing the covariance matrix estimate.

See Also

fwb() for performing the fractional weighted bootstrap on an arbitrary quantity; fwb.ci() for computing nonparametric confidence intervals for fwb objects; summary.fwb() for producing standard errors and confidence intervals for fwb objects; sandwich::vcovBS() for computing covariance matrices using the traditional bootstrap (the fractional weighted bootstrap is also available but with limited options).

Examples

```
set.seed(123, "L'Ecuyer-CMRG")
data("infert")
fit <- glm(case ~ spontaneous + induced, data = infert,
             family = "binomial")
lmtest::coeftest(fit, vcov. = vcovFWB, R = 200)


# Example from help("vcovBS", package = "sandwich")
data("PetersenCL", package = "sandwich")
m <- lm(y ~ x, data = PetersenCL)

# Note: this is not to compare performance, just to
# demonstrate the syntax
cbind(
  "BS" = sqrt(diag(sandwich::vcovBS(m))),
  "FWB" = sqrt(diag(vcovFWB(m))),
  "BS-cluster" = sqrt(diag(sandwich::vcovBS(m, cluster = ~firm))),
  "FWB-cluster" = sqrt(diag(vcovFWB(m, cluster = ~firm)))
)
```

```
# Using `wtype = "multinom"` exactly reproduces
# `sandwich::vcovBS()`
set.seed(11)
s <- sandwich::vcovBS(m, R = 200)
set.seed(11)
f <- vcovFWB(m, R = 200, wtype = "multinom")

all.equal(s, f)


# Using a custom argument to `.coef`
set.seed(123)
data("infert")

fit <- nnet::multinom(education ~ age, data = infert,
                      trace = FALSE)

# vcovFWB(fit, R = 200) ## error
coef(fit) # coef() returns a matrix

# Write a custom function to extract vector of
# coefficients (can also use marginaleffects::get_coef())
coef_multinom <- function(x) {
  p <- t(coef(x))

  setNames(as.vector(p),
           paste(colnames(p)[col(p)],
                 rownames(p)[row(p)],
                 sep = ":"))
}
coef_multinom(fit) # returns a vector

vcovFWB(fit, R = 200, .coef = coef_multinom)
```

---

w_mean *Calculate weighted statistics*

Description

These functions are designed to compute weighted statistics (mean, variance, standard deviation, covariance, correlation, median and quantiles) to perform weighted transformation (scaling, centering, and standardization) using bootstrap weights. These automatically extract bootstrap weights when called from within `fwb()` to facilitate computation of bootstrap statistics without needing to think to hard about how to correctly incorporate weights.

Usage

```
w_mean(x, w = NULL, na.rm = FALSE)
```

```
w_var(x, w = NULL, na.rm = FALSE)

w_sd(x, w = NULL, na.rm = FALSE)

w_cov(x, w = NULL, na.rm = FALSE)

w_cor(x, w = NULL)

w_quantile(
  x,
  w = NULL,
  probs = seq(0, 1, by = 0.25),
  na.rm = FALSE,
  names = TRUE,
  type = 7L,
  digits = 7L
)

w_median(x, w = NULL, na.rm = FALSE)

w_std(x, w = NULL, na.rm = TRUE, scale = TRUE, center = TRUE)

w_scale(x, w = NULL, na.rm = TRUE)

w_center(x, w = NULL, na.rm = TRUE)
```

Arguments

x	a numeric variable; for w_cov() and w_cor(), a numeric matrix.
w	optional; a numeric vector of weights with length equal to the length of x (or number of rows if x is a matrix). If unspecified, will use bootstrapped weights when called from within fwb() or vcovFWB() and unit weights (i.e., for unweighted estimates) otherwise.
na.rm	logical; whether to exclude NA values in the weights and x when computing statistics. Default is FALSE for the weighted statistics (like with their unweighted counterparts) and TRUE for weighted transformations.
probs, names, type, digits	see quantile(). Only type = 7 is allowed.
scale, center	logical; whether to scale or center the variable.

Details

These function automatically incorporate bootstrap weights when used inside fwb() or vcovFWB(). This works because fwb() and vcovFWB() temporarily set an option with the environment of the function that calls the estimation function with the sampled weights, and the w_*() functions access the bootstrap weights from that environment, if any. So, using, e.g., w_mean() inside the function supplied to the statistic argument of fwb(), computes the weighted mean using the bootstrap

weights. Using these functions outside `fwb()` works just like other functions that compute weighted statistics: when `w` is supplied, the statistics are weighted, and otherwise they are unweighted.

See below for how each statistic is computed.

Weighted statistics:

For all weighted statistics, the weights are first rescaled to sum to 1. w in the formulas below refers to these weights.

`w_mean()` Calculates the weighted mean as

$$\bar{x}_w = \sum_i w_i x_i$$

This is the same as `weighted.mean()`.

`w_var()` Calculates the weighted variance as

$$s^2_{x,w} = \frac{\sum_i w_i (x_i - \bar{x}_w)^2}{1 - \sum_i w_i^2}$$

`w_sd()` Calculates the weighted standard deviation as

$$s_{x,w} = \sqrt{s^2_{x,w}}$$

`w_cov()` Calculates the weighted covariances as

$$s_{x,y,w} = \frac{\sum_i w_i (x_i - \bar{x}_w)(y_i - \bar{y}_w)}{1 - \sum_i w_i^2}$$

This is the same as `cov.wt()`.

`w_cor()` Calculates the weighted correlation as

$$r_{x,y,w} = \frac{s_{x,y,w}}{s_{x,w} s_{y,w}}$$

This is the same as `cov.wt()`.

`w_quantile()` Calculates the weighted quantiles using linear interpolation of the weighted cumulative density function.

`w_median()` Calculates the weighted median as `w_quantile(., probs = .5)`.

Weighted transformations:

Weighted transformations use the weighted mean and/or standard deviation to re-center or re-scale the given variable. In the formulas below, x is the original variable and w are the weights.

`w_center()` Centers the variable at its (weighted) mean.

$$x_{c,w} = x - \bar{x}_w$$

`w_scale()` Scales the variable by its (weighted) standard deviation.

$$x_{s,w} = x / s_{x,w}$$

`w_std()` Centers and scales the variable by its (weighted) mean and standard deviation.

$$x_{z,w} = (x - \bar{x}_w) / s_{x,w}$$

`w_scale()` and `w_center()` are efficient wrappers to `w_std()` with `center = FALSE` and `scale = FALSE`, respectively.

Value

`w_mean()`, `w_var()`, `w_sd()`, and `w_median()` return a numeric scalar. `w_cov()` and `w_cor()` return numeric matrices. `w_quantile()` returns a numeric vector of length equal to `probs`. `w_std()`, `w_scale()`, and `w_center()` return numeric vectors of length equal to the length of `x`.

See Also

- `mean()` and `weighted.mean()` for computing the unweighted and weighted mean
- `var()` and `sd()` for computing the unweighted variance and standard deviation
- `median()` and `quantile()` for computing the unweighted median and quantiles
- `cov()`, `cor()`, and `cov.wt()` for unweighted and weighted covariance and correlation matrices
- `scale()` for standardizing variables using arbitrary (by default, unweighted) centering and scaling factors

Examples

```
# G-computation of average treatment effects using lalonde
# dataset

data("lalonde", package = "cobalt")

ate_est <- function(data, w) {
  fit <- lm(re78 ~ treat * (age + educ + married + race +
                              nodegree + re74 + re75),
            data = data, weights = w)

  p0 <- predict(fit, newdata = transform(data, treat = 0))
  p1 <- predict(fit, newdata = transform(data, treat = 1))

  # Weighted means using bootstrap weights
  m0 <- w_mean(p0)
  m1 <- w_mean(p1)

  c(m0 = m0, m1 = m1, ATE = m1 - m0)
}

set.seed(123, "L'Ecuyer-CMRG")
boot_est <- fwb(lalonde, statistic = ate_est,
                R = 199, verbose = FALSE)
summary(boot_est)

# Using `w_*()` data transformations inside a model
# supplied to vcovFWB():
fit <- lm(re78 ~ treat * w_center(age),
          data = lalonde)

lmtest::coeftest(fit, vcov = vcovFWB, R = 500)
```

Index