Package 'svrep'

December 18, 2022

Type Package

Title Tools for Creating, Updating, and Analyzing Survey Replicate Weights

Version 0.4.1

Description Provides tools for creating and working with survey replicate weights, extending functionality of the 'survey' package from Lumley (2004) <doi:10.18637/jss.v009.i08>. Methods are provided for applying nonresponse adjustments to both full-sample and replicate weights as described by Rust and Rao (1996) <doi:10.1177/096228029600500305>. Implements methods for sample-based calibration described by Opsomer and Erciulescu (2021) <https://www150.statcan.gc.ca/n1/pub/12-001-x/2021002/article/00006-eng.htm>. Diagnostic functions are included to compare weights and weighted estimates from different sets of replicate weights.

License GPL (>= 3)

URL https://bschneidr.github.io/svrep/, https://github.com/bschneidr/svrep

BugReports https://github.com/bschneidr/svrep/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.2.0

Imports mvtnorm, stats, survey (>= 4.1), utils

Suggests knitr, covr, testthat (>= 3.0.0), rmarkdown, tidycensus, dplyr, srvyr, withr

Config/testthat/edition 3

Depends R (>= 2.10)

VignetteBuilder knitr

NeedsCompilation no

Author Ben Schneider [aut, cre] (<https://orcid.org/0000-0002-0406-8470>)

Maintainer Ben Schneider <benjamin.julius.schneider@gmail.com>

Repository CRAN

Date/Publication 2022-12-18 16:40:02 UTC

R topics documented:

`as_bootstrap_design` *Convert a survey design object to a bootstrap replicate design*

Description

Converts a survey design object to a replicate design object with replicate weights formed using a bootstrap method. Supports stratified, cluster samples with one or more stages of sampling. At each stage of sampling, either simple random sampling (with or without replacement) or unequal probability sampling (with or without replacement) may be used.

Usage

```
as_bootstrap_design(
  design,
  type = "Rao-Wu-Yue-Beaumont",
  replicates = 500,
  compress = TRUE,
  mse = getOption("survey.replicates.mse"),
  samp_method_by_stage = NULL
)
```

Arguments

`design` A survey design object created using the 'survey' (or 'srvyr') package, with class `'survey.design'` or `'svyimputationList'`.

`type` The type of bootstrap to use, which should be chosen based on its applicability to the sampling method used for the survey. The available types are the following:

- **"Rao-Wu-Yue-Beaumont"** (the default):
  The bootstrap method of Beaumont and Émond (2022), which is a generalization of the Rao-Wu-Yue bootstrap, and is applicable to a wide variety of designs, including single-stage and multistage stratified designs. The design may have different sampling methods used at different stages. Each stage of sampling may potentially be PPS (i.e., use unequal probabilities), with or without replacement, and may potentially use Poisson sampling.

  For a stratum with a fixed sample size of n sampling units, resampling in each replicate resamples $(n-1)$ sampling units with replacement.
- **"Rao-Wu"**:
  The basic Rao-Wu $(n-1)$ bootstrap method, which is only applicable to single-stage designs or multistage designs where the first-stage sampling fractions are small (and can thus be ignored). Accommodates stratified designs. All sampling within a stratum must be simple random sampling with or without replacement, although the first-stage sampling is effectively treated as sampling without replacement.
- **"Preston"**:
  Preston's multistage rescaled bootstrap, which is applicable to single-stage designs or multistage designs with arbitrary sampling fractions. Accommodates stratified designs. All sampling within a stratum must be simple random sampling with or without replacement.
- **"Canty-Davison"**:
  The Canty-Davison bootstrap, which is only applicable to single-stage designs, with arbitrary sampling fractions. Accommodates stratified designs. All sampling with a stratum must be simple random sampling with or without replacement.

`replicates` Number of bootstrap replicates (should be as large as possible, given computer memory/storage limitations). A commonly-recommended default is 500.

`compress` Use a compressed representation of the replicate weights matrix. This reduces the computer memory required to represent the replicate weights and has no impact on estimates.

`mse` If `TRUE`, compute variances from sums of squares around the point estimate from the full-sample weights, If `FALSE`, compute variances from sums of squares around the mean estimate from the replicate weights.

`samp_method_by_stage`
(Optional). By default, this function will automatically determine the sampling method used at each stage. However, this argument can be used to ensure the correct sampling method is identified for each stage.

Accepts a vector with length equal to the number of stages of sampling. Each element should be one of the following:

- "SRSWOR" - Simple random sampling, without replacement
- "SRSWR" - Simple random sampling, with replacement
- "PPSWOR" - Unequal probabilities of selection, without replacement
- "PPSWR" - Unequal probabilities of selection, with replacement
- "Poisson" - Poisson sampling: each sampling unit is selected into the sample at most once, with potentially different probabilities of inclusion for each sampling unit.

Value

A replicate design object, with class `svyrep.design`, which can be used with the usual functions, such as `svymean()` or `svyglm()`.

Use `weights(..., type = 'analysis')` to extract the matrix of replicate weights.
Use `as_data_frame_with_weights()` to convert the design object to a data frame with columns for the full-sample and replicate weights.

See Also

Use `estimate_boot_reps_for_target_cv` to help choose the number of bootstrap replicates.

For some complex designs, one can use `make_rwyb_bootstrap_weights` to create Rao-Wu-Yue-Beaumont bootstrap weights or adjustment factors given information for each stage of sampling (the type of sampling, strata IDs, cluster IDs, selection probabilities, etc.).

For systematic samples, one-PSU-per-stratum designs, or other especially complex sample designs, one can use the generalized survey bootstrap method. See `as_gen_boot_design` or `make_gen_boot_factors`.

Examples

```
library(survey)
# Example 1: A multistage sample with two stages of SRSWOR

  ## Load an example dataset from a multistage sample, with two stages of SRSWOR
  data("mu284", package = 'survey')
  multistage_srswor_design <- svydesign(data = mu284,
                                        ids = ~ id1 + id2,
                                        fpc = ~ n1 + n2)

  ## Convert the survey design object to a bootstrap design
  set.seed(2022)
  bootstrap_rep_design <- as_bootstrap_design(multistage_srswor_design,
                                              replicates = 500)

  ## Compare std. error estimates from bootstrap versus linearization
  data.frame(
    'Statistic' = c('total', 'mean', 'median'),
```

```
    'SE (bootstrap)' = c(SE(svytotal(x = ~ y1, design = bootstrap_rep_design)),
                         SE(svymean(x = ~ y1, design = bootstrap_rep_design)),
                         SE(svyquantile(x = ~ y1, quantile = 0.5,
                                        design = bootstrap_rep_design))),
    'SE (linearization)' = c(SE(svytotal(x = ~ y1, design = multistage_srswor_design)),
                             SE(svymean(x = ~ y1, design = multistage_srswor_design)),
                             SE(svyquantile(x = ~ y1, quantile = 0.5,
                                            design = multistage_srswor_design))),
    check.names = FALSE
  )

# Example 2: A multistage-sample,
# first stage selected with unequal probabilities without replacement
# second stage selected with simple random sampling without replacement

  data("library_multistage_sample", package = "svrep")

  multistage_pps <- svydesign(data = library_multistage_sample,
                              ids = ~ PSU_ID + SSU_ID,
                              fpc = ~ PSU_SAMPLING_PROB + SSU_SAMPLING_PROB,
                              pps = "brewer")

  bootstrap_rep_design <- as_bootstrap_design(
    multistage_pps, replicates = 500,
    samp_method_by_stage = c("PPSWOR", "SRSWOR")
  )

  ## Compare std. error estimates from bootstrap versus linearization
  data.frame(
      'Statistic' = c('total', 'mean'),
      'SE (bootstrap)' = c(
          SE(svytotal(x = ~ TOTCIR, na.rm = TRUE,
                      design = bootstrap_rep_design)),
          SE(svymean(x = ~ TOTCIR, na.rm = TRUE,
                     design = bootstrap_rep_design))),
      'SE (linearization)' = c(
          SE(svytotal(x = ~ TOTCIR, na.rm = TRUE,
                      design = multistage_pps)),
          SE(svymean(x = ~ TOTCIR, na.rm = TRUE,
                     design = multistage_pps))),
      check.names = FALSE
  )
```

`as_data_frame_with_weights` *Convert a survey design object to a data frame with weights stored as columns*

Description

Convert a survey design object to a data frame with weights stored as columns

Usage

```
as_data_frame_with_weights(
  design,
  full_wgt_name = "FULL_SAMPLE_WGT",
  rep_wgt_prefix = "REP_WGT_"
)
```

Arguments

design	A survey design object, created with either the survey or srvyr packages.
full_wgt_name	The column name to use for the full-sample weights
rep_wgt_prefix	For replicate design objects, a prefix to use for the column names of the replicate weights. The column names will be created by appending the replicate number after the prefix.

Value

A data frame, with new columns containing the weights from the survey design object

Examples

```
data("lou_vax_survey", package = 'svrep')

library(survey)
# Create a survey design object
survey_design <- svydesign(data = lou_vax_survey,
                           weights = ~ SAMPLING_WEIGHT,
                           ids = ~ 1)

rep_survey_design <- as.svrepdesign(survey_design,
                                    type = "boot",
                                    replicates = 10)

# Adjust the weights for nonresponse
nr_adjusted_design <- redistribute_weights(
  design = rep_survey_design,
  reduce_if = RESPONSE_STATUS == "Nonrespondent",
  increase_if = RESPONSE_STATUS == "Respondent",
  by = c("RACE_ETHNICITY", "EDUC_ATTAINMENT")
)

# Save the survey design object as a data frame
nr_adjusted_data <- as_data_frame_with_weights(
  nr_adjusted_design,
  full_wgt_name = "NR_ADJUSTED_WGT",
  rep_wgt_prefix = "NR_ADJUSTED_REP_WGT_"
)
head(nr_adjusted_data)
```

```
# Check the column names of the result
colnames(nr_adjusted_data)
```

`as_gen_boot_design` *Convert a survey design object to a generalized boostrap replicate design*

Description

Converts a survey design object to a replicate design object with replicate weights formed using the generalized bootstrap method. The generalized survey bootstrap is a method for forming bootstrap replicate weights from a textbook variance estimator, provided that the variance estimator can be represented as a quadratic form whose matrix is positive semi-definite (this covers a large class of variance estimators).

Usage

```
as_gen_boot_design(
  design,
  variance_estimator = NULL,
  replicates = 500,
  tau = "auto",
  mse = getOption("survey.replicates.mse"),
  compress = TRUE
)
```

Arguments

`design` A survey design object created using the 'survey' (or 'srvyr') package, with class `'survey.design'` or `'svyimputationList'`.

`variance_estimator`
The name of the variance estimator whose quadratic form matrix should be created. See the section "Variance Estimators" below. Options include:

- **"Yates-Grundy"**: The Yates-Grundy variance estimator based on first-order and second-order inclusion probabilities.
- **"Horvitz-Thompson"**: The Horvitz-Thompson variance estimator based on first-order and second-order inclusion probabilities.
- **"Stratified Multistage SRS"**: The usual stratified multistage variance estimator based on estimating the variance of cluster totals within strata at each stage.
- **"Ultimate Cluster"**: The usual variance estimator based on estimating the variance of first-stage cluster totals within first-stage strata.
- **"SD1"**: The non-circular successive-differences variance estimator described by Ash (2014), sometimes used for variance estimation for systematic sampling.

- **"SD2"**: The circular successive-differences variance estimator described by Ash (2014). This estimator is the basis of the "successive-differences replication" estimator commonly used for variance estimation for systematic sampling.

`replicates` Number of bootstrap replicates (should be as large as possible, given computer memory/storage limitations). A commonly-recommended default is 500.

`tau` Either `"auto"`, or a single number. This is the rescaling constant used to avoid negative weights through the transformation $\frac{w+\tau-1}{\tau}$, where w is the original weight and τ is the rescaling constant `tau`.
If `tau="auto"`, the rescaling factor is determined automatically as follows: if all of the adjustment factors are nonnegative, then `tau` is set equal to 1; otherwise, `tau` is set to the smallest value needed to rescale the adjustment factors such that they are all at least `0.01`.

`mse` If `TRUE`, compute variances from sums of squares around the point estimate from the full-sample weights, If `FALSE`, compute variances from sums of squares around the mean estimate from the replicate weights.

`compress` This reduces the computer memory required to represent the replicate weights and has no impact on estimates.

Value

A replicate design object, with class `svyrep.design`, which can be used with the usual functions, such as `svymean()` or `svyglm()`.

Use `weights(..., type = 'analysis')` to extract the matrix of replicate weights.
Use `as_data_frame_with_weights()` to convert the design object to a data frame with columns for the full-sample and replicate weights.

Statistical Details

Let $v(\hat{T}_y)$ be the textbook variance estimator for an estimated population total $\hat{T}_y$ of some variable y. The base weight for case i in our sample is w_i, and we let $\breve{y}_i$ denote the weighted value $w_i y_i$. Suppose we can represent our textbook variance estimator as a quadratic form: $v(\hat{T}_y) = \breve{y}\Sigma\breve{y}^T$, for some $n \times n$ matrix Σ. The only constraint on Σ is that, for our sample, it must be symmetric and positive semi-definite.

The bootstrapping process creates B sets of replicate weights, where the b-th set of replicate weights is a vector of length n denoted $\mathbf{a}^{(b)}$, whose k-th value is denoted $a_k^{(b)}$. This yields B replicate estimates of the population total, $\hat{T}_y^{*(b)} = \sum_{k \in s} a_k^{(b)} \breve{y}_k$, for $b = 1, \ldots B$, which can be used to estimate sampling variance.

$$v_B\left(\hat{T}_y\right) = \frac{\sum_{b=1}^{B}\left(\hat{T}_y^{*(b)} - \hat{T}_y\right)^2}{B}$$

This bootstrap variance estimator can be written as a quadratic form:

$$v_B\left(\hat{T}_y\right) = \breve{\mathbf{y}}'\Sigma_B\breve{\mathbf{y}}$$

where

$$\mathbf{\Sigma}_B = \frac{\sum_{b=1}^{B} \left(\mathbf{a}^{(b)} - \mathbf{1}_n\right)\left(\mathbf{a}^{(b)} - \mathbf{1}_n\right)'}{B}$$

Note that if the vector of adjustment factors $\mathbf{a}^{(b)}$ has expectation $\mathbf{1}_n$ and variance-covariance matrix $\mathbf{\Sigma}$, then we have the bootstrap expectation $E_*\left(\mathbf{\Sigma}_B\right) = \mathbf{\Sigma}$. Since the bootstrap process takes the sample values $\breve{y}$ as fixed, the bootstrap expectation of the variance estimator is $E_*\left(\breve{\mathbf{y}}'\Sigma_B\breve{\mathbf{y}}\right) = \breve{\mathbf{y}}'\Sigma\breve{\mathbf{y}}$. Thus, we can produce a bootstrap variance estimator with the same expectation as the textbook variance estimator simply by randomly generating $\mathbf{a}^{(b)}$ from a distribution with the following two conditions:
Condition 1: $\mathbf{E}_*(\mathbf{a}) = \mathbf{1}_n$
Condition 2: $\mathbf{E}_*\left(\mathbf{a} - \mathbf{1}_n\right)\left(\mathbf{a} - \mathbf{1}_n\right)' = \mathbf{\Sigma}$

While there are multiple ways to generate adjustment factors satisfying these conditions, the simplest general method is to simulate from a multivariate normal distribution: $\mathbf{a} \sim MVN(\mathbf{1}_n, \mathbf{\Sigma})$. This is the method used by this function.

Details on Rescaling to Avoid Negative Adjustment Factors

Let $\mathbf{A} = \left[\mathbf{a}^{(1)} \cdots \mathbf{a}^{(b)} \cdots \mathbf{a}^{(B)}\right]$ denote the $(n \times B)$ matrix of bootstrap adjustment factors. To eliminate negative adjustment factors, Beaumont and Patak (2012) propose forming a rescaled matrix of nonnegative replicate factors $\mathbf{A}^S$ by rescaling each adjustment factor $a_k^{(b)}$ as follows:

$$a_k^{S,(b)} = \frac{a_k^{(b)} + \tau - 1}{\tau}$$

where $\tau \geq 1 - a_k^{(b)} \geq 1$ for all k in $\{1, \ldots, n\}$ and all b in $\{1, \ldots, B\}$.

The value of τ can be set based on the realized adjustment factor matrix $\mathbf{A}$ or by choosing τ prior to generating the adjustment factor matrix $\mathbf{A}$ so that τ is likely to be large enough to prevent negative bootstrap weights.

If the adjustment factors are rescaled in this manner, it is important to adjust the scale factor used in estimating the variance with the bootstrap replicates, which becomes $\frac{\tau^2}{B}$ instead of $\frac{1}{B}$.

$$\textbf{Prior to rescaling: } v_B\left(\hat{T}_y\right) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{T}_y^{*(b)} - \hat{T}_y\right)^2$$

$$\textbf{After rescaling: } v_B\left(\hat{T}_y\right) = \frac{\tau^2}{B}\sum_{b=1}^{B}\left(\hat{T}_y^{S*(b)} - \hat{T}_y\right)^2$$

When sharing a dataset that uses rescaled weights from a generalized survey bootstrap, the documentation for the dataset should instruct the user to use replication scale factor $\frac{\tau^2}{B}$ rather than $\frac{1}{B}$ when estimating sampling variances.

Variance Estimators

The **Horvitz-Thompson** variance estimator:

$$v(\hat{Y}) = \sum_{i \in s}\sum_{j \in s}(1 - \frac{\pi_i\pi_j}{\pi_{ij}})\frac{y_i}{\pi_i}\frac{y_j}{\pi_j}$$

The **Yates-Grundy** variance estimator:

$$v(\hat{Y}) = -\frac{1}{2}\sum_{i\in s}\sum_{j\in s}(1-\frac{\pi_i\pi_j}{\pi_{ij}})(\frac{y_i}{\pi_i}-\frac{y_j}{\pi_j})^2$$

The **Stratified Multistage SRS** variance estimator is the recursive variance estimator proposed by Bellhouse (1985) and used in the 'survey' package's function svyrecvar. The estimator can be used for any number of sampling stages. For illustration, we describe its use for two sampling stages.

$$v(\hat{Y}) = \hat{V}_1 + \hat{V}_2$$

where

$$\hat{V}_1 = \sum_{h=1}^{H}(1-\frac{n_h}{N_h})\frac{n_h}{n_h-1}\sum_{i=1}^{n_h}(y_{hi.}-\bar{y}_{hi.})^2$$

and

$$\hat{V}_2 = \sum_{h=1}^{H}\frac{n_h}{N_h}\sum_{i=1}^{n_h}v_{hi}(y_{hi.})$$

where n_h is the number of sampled clusters in stratum h, N_h is the number of population clusters in stratum h, $y_{hi.}$ is the weighted cluster total in cluster i of stratum h, $\bar{y}_{hi.}$ is the mean weighted cluster total of stratum h, ($\bar{y}_{hi.} = \frac{1}{n_h}\sum_{i=1}^{n_h} y_{hi.}$), and $v_{hi}(y_{hi.})$ is the estimated sampling variance of $y_{hi.}$.

The **Ultimate Cluster** variance estimator is simply the stratified multistage SRS variance estimator, but ignoring variances from later stages of sampling.

$$v(\hat{Y}) = \hat{V}_1$$

This is the variance estimator used in the 'survey' package when the user specifies `option(survey.ultimate.cluster = TRUE)` or uses `svyrecvar(..., one.stage = TRUE)`. When the first-stage sampling fractions are small, analysts often omit the finite population corrections $(1-\frac{n_h}{N_h})$ when using the ultimate cluster estimator.

The **SD1** and **SD2** variance estimators are "successive difference" estimators sometimes used for systematic sampling designs. Ash (2014) describes each estimator as follows:

$$\hat{v}_{SD1}(\hat{Y}) = \left(1-\frac{n}{N}\right)\frac{n}{2(n-1)}\sum_{k=2}^{n}\left(\breve{y}_k-\breve{y}_{k-1}\right)^2$$

$$\hat{v}_{SD2}(\hat{Y}) = \left(1-\frac{n}{N}\right)\frac{1}{2}\left[\sum_{k=2}^{n}\left(\breve{y}_k-\breve{y}_{k-1}\right)^2+\left(\breve{y}_n-\breve{y}_1\right)^2\right]$$

where $\breve{y}_k = y_k/\pi_k$ is the weighted value of unit k with selection probability π_k. The SD1 estimator is recommended by Wolter (1984). The SD2 estimator is the basis of the successive difference replication estimator commonly used for systematic sampling designs. See Ash (2014) for details.

For multistage samples, SD1 and SD2 are applied to the clusters at each stage, separately by stratum. For later stages of sampling, the variance estimate from a stratum is multiplied by the product of sampling fractions from earlier stages of sampling. For example, at a third stage of sampling, the variance estimate from a third-stage stratum is multiplied by $\frac{n_1}{N_1}\frac{n_2}{N_2}$, which is the product of sampling fractions from the first-stage stratum and second-stage stratum.

References

The generalized survey bootstrap was first proposed by Bertail and Combris (1997). See Beaumont and Patak (2012) for a clear overview of the generalized survey bootstrap. The generalized survey bootstrap represents one strategy for forming replication variance estimators in the general framework proposed by Fay (1984) and Dippo, Fay, and Morganstein (1984).

- Ash, S. (2014). "*Using successive difference replication for estimating variances*." **Survey Methodology**, Statistics Canada, 40(1), 47–59.

Bellhouse, D.R. (1985). "*Computing Methods for Variance Estimation in Complex Surveys*." **Journal of Official Statistics**, Vol.1, No.3.

- Beaumont, Jean-François, and Zdenek Patak. 2012. "On the Generalized Bootstrap for Sample Surveys with Special Attention to Poisson Sampling: Generalized Bootstrap for Sample Surveys." International Statistical Review 80 (1): 127–48. https://doi.org/10.1111/j.1751-5823.2011.00166.x.

- Bertail, and Combris. 1997. "Bootstrap Généralisé d'un Sondage." Annales d'Économie Et de Statistique, no. 46: 49. https://doi.org/10.2307/20076068.

- Dippo, Cathryn, Robert Fay, and David Morganstein. 1984. "Computing Variances from Complex Samples with Replicate Weights." In, 489–94. Alexandria, VA: American Statistical Association. http://www.asasrms.org/Proceedings/papers/1984_094.pdf.

- Fay, Robert. 1984. "Some Properties of Estimates of Variance Based on Replication Methods." In, 495–500. Alexandria, VA: American Statistical Association. http://www.asasrms.org/Proceedings/papers/1984_095.pdf.

See Also

Use `estimate_boot_reps_for_target_cv` to help choose the number of bootstrap replicates.
For greater customization of the method, `make_quad_form_matrix` can be used to represent several common variance estimators as a quadratic form's matrix, which can then be used as an input to `make_gen_boot_factors`.

Examples

```
## Not run:
library(survey)

# Example 1: Bootstrap based on the Yates-Grundy estimator ----
   set.seed(2014)

   data('election', package = 'survey')

   ## Create survey design object
   pps_design_yg <- svydesign(
     data = election_pps,
     id = ~1, fpc = ~p,
     pps = ppsmat(election_jointprob),
```

```
  variance = "YG"
)

## Convert to generalized bootstrap replicate design
gen_boot_design_yg <- pps_design_yg |>
  as_gen_boot_design(variance_estimator = "Yates-Grundy",
                     replicates = 1000, tau = "auto")

svytotal(x = ~ Bush + Kerry, design = pps_design_yg)
svytotal(x = ~ Bush + Kerry, design = gen_boot_design_yg)

# Example 2: Bootstrap based on the successive-difference estimator ----

data('library_stsys_sample', package = 'svrep')

## First, ensure data are sorted in same order as was used in sampling
library_stsys_sample <- library_stsys_sample[
  order(library_stsys_sample$SAMPLING_SORT_ORDER),
]

## Create a survey design object
design_obj <- svydesign(
  data = library_stsys_sample,
  strata = ~ SAMPLING_STRATUM,
  ids = ~ 1,
  fpc = ~ STRATUM_POP_SIZE
)

## Convert to generalized bootstrap replicate design
gen_boot_design_sd2 <- as_gen_boot_design(
  design = design_obj,
  variance_estimator = "SD2",
  replicates = 2000
)

## Estimate sampling variances
svytotal(x = ~ TOTSTAFF, na.rm = TRUE, design = gen_boot_design_sd2)
svytotal(x = ~ TOTSTAFF, na.rm = TRUE, design = design_obj)

## End(Not run)
```

`calibrate_to_estimate` *Calibrate weights from a primary survey to estimated totals from a control survey, with replicate-weight adjustments that account for variance of the control totals*

Description

Calibrate the weights of a primary survey to match estimated totals from a control survey, using adjustments to the replicate weights to account for the variance of the estimated control totals.

The adjustments to replicate weights are conducted using the method proposed by Fuller (1998). This method can be used to implement general calibration as well as post-stratification or raking specifically (see the details for the `calfun` parameter).

Usage

```
calibrate_to_estimate(
  rep_design,
  estimate,
  vcov_estimate,
  cal_formula,
  calfun = survey::cal.linear,
  bounds = list(lower = -Inf, upper = Inf),
  verbose = FALSE,
  maxit = 50,
  epsilon = 1e-07,
  variance = NULL,
  col_selection = NULL
)
```

Arguments

`rep_design`	A replicate design object for the primary survey, created with either the `survey` or `srvyr` packages.
`estimate`	A vector of estimated control totals. The names of entries must match the names from calling `svytotal(x = cal_formula, design = rep_design)`.
`vcov_estimate`	A variance-covariance matrix for the estimated control totals. The column names and row names must match the names of `estimate`.
`cal_formula`	A formula listing the variables to use for calibration. All of these variables must be included in `rep_design`.
`calfun`	A calibration function from the `survey` package, such as cal.linear, cal.raking, or cal.logit. Use `cal.linear` for ordinary post-stratification, and `cal.raking` for raking. See calibrate for additional details.
`bounds`	Parameter passed to grake for calibration. See calibrate for details.
`verbose`	Parameter passed to grake for calibration. See calibrate for details.
`maxit`	Parameter passed to grake for calibration. See calibrate for details.
`epsilon`	Parameter passed to grake for calibration. After calibration, the absolute difference between each calibration target and the calibrated estimate will be no larger than `epsilon` times (1 plus the absolute value of the target). See calibrate for details.
`variance`	Parameter passed to grake for calibration. See calibrate for details.
`col_selection`	Optional parameter to determine which replicate columns will have their control totals perturbed. If supplied, `col_selection` must be an integer vector with length equal to the length of `estimate`.

Details

With the Fuller method, each of k randomly-selected replicate columns from the primary survey are calibrated to control totals formed by perturbing the k-dimensional vector of estimated control totals using a spectral decomposition of the variance-covariance matrix of the estimated control totals. Other replicate columns are simply calibrated to the unperturbed control totals.

Because the set of replicate columns whose control totals are perturbed should be random, there are multiple ways to ensure that this matching is reproducible. The user can either call set.seed before using the function, or supply a vector of randomly-selected column indices to the argument `col_selection`.

Value

A replicate design object, with full-sample weights calibrated to totals from `estimate`, and replicate weights adjusted to account for variance of the control totals. The element `col_selection` indicates, for each replicate column of the calibrated primary survey, which column of replicate weights it was matched to from the control survey.

References

Fuller, W.A. (1998). "Replication variance estimation for two-phase samples." **Statistica Sinica**, *8*: 1153-1164.

Opsomer, J.D. and A. Erciulescu (2021). "Replication variance estimation after sample-based calibration." **Survey Methodology**, *47*: 265-277.

Examples

```
## Not run:

# Load example data for primary survey ----

  suppressPackageStartupMessages(library(survey))
  data(api)

  primary_survey <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc) |>
    as.svrepdesign(type = "JK1")

# Load example data for control survey ----

  control_survey <- svydesign(id = ~ 1, fpc = ~fpc, data = apisrs) |>
    as.svrepdesign(type = "JK1")

# Estimate control totals ----

  estimated_controls <- svytotal(x = ~ stype + enroll,
                                 design = control_survey)
  control_point_estimates <- coef(estimated_controls)
  control_vcov_estimate <- vcov(estimated_controls)

# Calibrate totals for one categorical variable and one numeric ----
```

```
  calibrated_rep_design <- calibrate_to_estimate(
    rep_design = primary_survey,
    estimate = control_point_estimates,
    vcov_estimate = control_vcov_estimate,
    cal_formula = ~ stype + enroll
  )

# Inspect estimates before and after calibration ----

  ##_ For the calibration variables, estimates and standard errors
  ##_ from calibrated design will match those of the control survey

    svytotal(x = ~ stype + enroll, design = primary_survey)
    svytotal(x = ~ stype + enroll, design = control_survey)
    svytotal(x = ~ stype + enroll, design = calibrated_rep_design)

  ##_ Estimates from other variables will be changed as well

    svymean(x = ~ api00 + api99, design = primary_survey)
    svymean(x = ~ api00 + api99, design = control_survey)
    svymean(x = ~ api00 + api99, design = calibrated_rep_design)

# Inspect weights before and after calibration ----

  summarize_rep_weights(primary_survey, type = 'overall')
  summarize_rep_weights(calibrated_rep_design, type = 'overall')

# For reproducibility, specify which columns are randomly selected for Fuller method ----

  column_selection <- calibrated_rep_design$col_selection
  print(column_selection)

  calibrated_rep_design <- calibrate_to_estimate(
    rep_design = primary_survey,
    estimate = control_point_estimates,
    vcov_estimate = control_vcov_estimate,
    cal_formula = ~ stype + enroll,
    col_selection = column_selection
  )

## End(Not run)
```

`calibrate_to_sample` *Calibrate weights from a primary survey to estimated totals from a control survey, with replicate-weight adjustments that account for variance of the control totals*

Description

Calibrate the weights of a primary survey to match estimated totals from a control survey, using adjustments to the replicate weights to account for the variance of the estimated control totals. The adjustments to replicate weights are conducted using the method proposed by Opsomer and Erciulescu (2021). This method can be used to implement general calibration as well as post-stratification or raking specifically (see the details for the `calfun` parameter).

Usage

```
calibrate_to_sample(
  primary_rep_design,
  control_rep_design,
  cal_formula,
  calfun = survey::cal.linear,
  bounds = list(lower = -Inf, upper = Inf),
  verbose = FALSE,
  maxit = 50,
  epsilon = 1e-07,
  variance = NULL,
  control_col_matches = NULL
)
```

Arguments

`primary_rep_design`
: A replicate design object for the primary survey, created with either the `survey` or `srvyr` packages.

`control_rep_design`
: A replicate design object for the control survey.

`cal_formula`
: A formula listing the variables to use for calibration. All of these variables must be included in both `primary_rep_design` and `control_rep_design`.

`calfun`
: A calibration function from the `survey` package, such as cal.linear, cal.raking, or cal.logit. Use `cal.linear` for ordinary post-stratification, and `cal.raking` for raking. See calibrate for additional details.

`bounds`
: Parameter passed to grake for calibration. See calibrate for details.

`verbose`
: Parameter passed to grake for calibration. See calibrate for details.

`maxit`
: Parameter passed to grake for calibration. See calibrate for details.

`epsilon`
: Parameter passed to grake for calibration.
After calibration, the absolute difference between each calibration target and the calibrated estimate will be no larger than `epsilon` times (1 plus the absolute value of the target). See calibrate for details.

`variance`
: Parameter passed to grake for calibration. See calibrate for details.

`control_col_matches`
: Optional parameter to control which control survey replicate is matched to each primary survey replicate. Entries of `NA` denote a primary survey replicate not matched to any control survey replicate. If this parameter is not used, matching is done at random.

Details

With the Opsomer-Erciulescu method, each column of replicate weights from the control survey is randomly matched to a column of replicate weights from the primary survey, and then the column from the primary survey is calibrated to control totals estimated by perturbing the control sample's full-sample estimates using the estimates from the matched column of replicate weights from the control survey.

If there are fewer columns of replicate weights in the control survey than in the primary survey, then not all primary replicate columns will be matched to a replicate column from the control survey.

If there are more columns of replicate weights in the control survey than in the primary survey, then the columns of replicate weights in the primary survey will be duplicated `k` times, where `k` is the smallest positive integer such that the resulting number of columns of replicate weights for the primary survey is greater than or equal to the number of columns of replicate weights in the control survey.

Because replicate columns of the control survey are matched *at random* to primary survey replicate columns, there are multiple ways to ensure that this matching is reproducible. The user can either call set.seed before using the function, or supply a mapping to the argument `control_col_matches`.

Value

A replicate design object, with full-sample weights calibrated to totals from `control_rep_design`, and replicate weights adjusted to account for variance of the control totals. If `primary_rep_design` had fewer columns of replicate weights than `control_rep_design`, then the number of replicate columns and the length of `rscales` will be increased by a multiple `k`, and the `scale` will be updated by dividing by `k`.

The element `control_column_matches` indicates, for each replicate column of the calibrated primary survey, which column of replicate weights it was matched to from the control survey. Columns which were not matched to control survey replicate column are indicated by `NA`.

The element `degf` will be set to match that of the primary survey to ensure that the degrees of freedom are not erroneously inflated by potential increases in the number of columns of replicate weights.

References

Opsomer, J.D. and A. Erciulescu (2021). "Replication variance estimation after sample-based calibration." **Survey Methodology**, *47*: 265-277.

Examples

```
## Not run:

# Load example data for primary survey ----
```

```
  suppressPackageStartupMessages(library(survey))
  data(api)

  primary_survey <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc) |>
    as.svrepdesign(type = "JK1")

# Load example data for control survey ----

  control_survey <- svydesign(id = ~ 1, fpc = ~fpc, data = apisrs) |>
    as.svrepdesign(type = "JK1")

# Calibrate totals for one categorical variable and one numeric ----

  calibrated_rep_design <- calibrate_to_sample(
    primary_rep_design = primary_survey,
    control_rep_design = control_survey,
    cal_formula = ~ stype + enroll,
  )

# Inspect estimates before and after calibration ----

  ##_ For the calibration variables, estimates and standard errors
  ##_ from calibrated design will match those of the control survey

    svytotal(x = ~ stype + enroll, design = primary_survey)
    svytotal(x = ~ stype + enroll, design = control_survey)
    svytotal(x = ~ stype + enroll, design = calibrated_rep_design)

  ##_ Estimates from other variables will be changed as well

    svymean(x = ~ api00 + api99, design = primary_survey)
    svymean(x = ~ api00 + api99, design = control_survey)
    svymean(x = ~ api00 + api99, design = calibrated_rep_design)

# Inspect weights before and after calibration ----

  summarize_rep_weights(primary_survey, type = 'overall')
  summarize_rep_weights(calibrated_rep_design, type = 'overall')

# For reproducibility, specify how to match replicates between surveys ----

  column_matching <- calibrated_rep_design$control_col_matches
  print(column_matching)

  calibrated_rep_design <- calibrate_to_sample(
    primary_rep_design = primary_survey,
    control_rep_design = control_survey,
    cal_formula = ~ stype + enroll,
    control_col_matches = column_matching
  )

## End(Not run)
```

`estimate_boot_reps_for_target_cv` *Estimate the number of bootstrap replicates needed to reduce the bootstrap simulation error to a target level*

Description

This function estimates the number of bootstrap replicates needed to reduce the simulation error of a bootstrap variance estimator to a target level, where "simulation error" is defined as error caused by using only a finite number of bootstrap replicates and this simulation error is measured as a simulation coefficient of variation ("simulation CV").

Usage

```
estimate_boot_reps_for_target_cv(svrepstat, target_cv = 0.05)
```

Arguments

`svrepstat`	An estimate obtained from a bootstrap replicate survey design object, with a function such as `svymean(..., return.replicates = TRUE)` or `withReplicates(..., return.replicates = TRUE)`.
`target_cv`	A numeric value (or vector of numeric values) between 0 and 1. This is the target simulation CV for the bootstrap variance estimator.

Value

A data frame with one row for each value of `target_cv`. The column `TARGET_CV` gives the target coefficient of variation. The column `MAX_REPS` gives the maximum number of replicates needed for all of the statistics included in `svrepstat`. The remaining columns give the number of replicates needed for each statistic.

Suggested Usage

- **Step 1**: Determine the largest acceptable level of simulation error for key survey estimates, where the level of simulation error is measured in terms of the simulation CV. We refer to this as the "target CV." A conventional value for the target CV is 5%.

- **Step 2**: Estimate key statistics of interest using a large number of bootstrap replicates (such as 5,000) and save the estimates from each bootstrap replicate. This can be conveniently done using a function from the survey package such as `svymean(..., return.replicates = TRUE)` or `withReplicates(..., return.replicates = TRUE)`.

- **Step 3**: Use the function `estimate_boot_reps_for_target_cv()` to estimate the minimum number of bootstrap replicates needed to attain the target CV.

Statistical Details

Unlike other replication methods such as the jackknife or balanced repeated replication, the bootstrap variance estimator's precision can always be improved by using a larger number of replicates, as the use of only a finite number of bootstrap replicates introduces simulation error to the variance estimation process. Simulation error can be measured as a "simulation coefficient of variation" (CV), which is the ratio of the standard error of a bootstrap estimator to the expectation of that bootstrap estimator, where the expectation and standard error are evaluated with respect to the bootstrapping process given the selected sample.

For a statistic $\hat{\theta}$, the simulation CV of the bootstrap variance estimator $v_B(\hat{\theta})$ based on B replicate estimates $\hat{\theta}^\star_1, \ldots, \hat{\theta}^\star_B$ is defined as follows:

$$CV_\star(v_B(\hat{\theta})) = \frac{\sqrt{var_\star(v_B(\hat{\theta}))}}{E_\star(v_B(\hat{\theta}))} = \frac{CV_\star(E_2)}{\sqrt{B}}$$

where

$$E_2 = (\hat{\theta}^\star - \hat{\theta})^2$$

$$CV_\star(E_2) = \frac{\sqrt{var_\star(E_2)}}{E_\star(E_2)}$$

and $var_\star$ and $E_\star$ are evaluated with respect to the bootstrapping process, given the selected sample.

The simulation CV, denoted $CV_\star(v_B(\hat{\theta}))$, is estimated for a given number of replicates B by estimating $CV_\star(E_2)$ using observed values and dividing this by $\sqrt{B}$. If the bootstrap errors are assumed to be normally distributed, then $CV_\star(E_2) = \sqrt{2}$ and so $CV_\star(v_B(\hat{\theta}))$ would not need to be estimated. Using observed replicate estimates to estimate the simulation CV instead of assuming normality allows simulation CV to be used for a a wide array of bootstrap methods.

References

See Section 3.3 and Section 8 of Beaumont and Patak (2012) for details and an example where the simulation CV is used to determine the number of bootstrap replicates needed for various alternative bootstrap methods in an empirical illustration.

Beaumont, J.-F. and Z. Patak. (2012), "On the Generalized Bootstrap for Sample Surveys with Special Attention to Poisson Sampling." **International Statistical Review**, *80*: 127-148. doi:10.1111/j.17515823.2011.00166.x.

See Also

Use `estimate_boot_sim_cv` to estimate the simulation CV for the number of bootstrap replicates actually used.

Examples

```
## Not run:
set.seed(2022)

# Create an example bootstrap survey design object ----
library(survey)
```

```
data('api', package = 'survey')

boot_design <- svydesign(id=~1,strata=~stype, weights=~pw,
                         data=apistrat, fpc=~fpc) |>
 svrep::as_bootstrap_design(replicates = 5000)

# Calculate estimates of interest and retain estimates from each replicate ----

estimated_means_and_proportions <- svymean(x = ~ api00 + api99 + stype, design = boot_design,
                                           return.replicates = TRUE)
custom_statistic <- withReplicates(design = boot_design,
                                   return.replicates = TRUE,
                                   theta = function(wts, data) {
                                      numerator <- sum(data$api00 * wts)
                                      denominator <- sum(data$api99 * wts)
                                      statistic <- numerator/denominator
                                      return(statistic)
                                   })
# Determine minimum number of bootstrap replicates needed to obtain given simulation CVs ----

  estimate_boot_reps_for_target_cv(
    svrepstat = estimated_means_and_proportions,
    target_cv = c(0.01, 0.05, 0.10)
  )

  estimate_boot_reps_for_target_cv(
    svrepstat = custom_statistic,
    target_cv = c(0.01, 0.05, 0.10)
  )

## End(Not run)
```

`estimate_boot_sim_cv` *Estimate the bootstrap simulation error*

Description

Estimates the bootstrap simulation error, expressed as a "simulation coefficient of variation" (CV).

Usage

```
estimate_boot_sim_cv(svrepstat)
```

Arguments

`svrepstat`	An estimate obtained from a bootstrap replicate survey design object, with a function such as `svymean(..., return.replicates = TRUE)` or `withReplicates(..., return.replicates = TRUE)`.

Value

A data frame with one row for each statistic. The column `STATISTIC` gives the name of the statistic. The column `SIMULATION_CV` gives the estimated simulation CV of the statistic. The column `N_REPLICATES` gives the number of bootstrap replicates.

Statistical Details

Unlike other replication methods such as the jackknife or balanced repeated replication, the bootstrap variance estimator's precision can always be improved by using a larger number of replicates, as the use of only a finite number of bootstrap replicates introduces simulation error to the variance estimation process. Simulation error can be measured as a "simulation coefficient of variation" (CV), which is the ratio of the standard error of a bootstrap estimator to the expectation of that bootstrap estimator, where the expectation and standard error are evaluated with respect to the bootstrapping process given the selected sample.

For a statistic $\hat{\theta}$, the simulation CV of the bootstrap variance estimator $v_B(\hat{\theta})$ based on B replicate estimates $\hat{\theta}^\star_1, \ldots, \hat{\theta}^\star_B$ is defined as follows:

$$CV_\star(v_B(\hat{\theta})) = \frac{\sqrt{var_\star(v_B(\hat{\theta}))}}{E_\star(v_B(\hat{\theta}))} = \frac{CV_\star(E_2)}{\sqrt{B}}$$

where

$$E_2 = (\hat{\theta}^\star - \hat{\theta})^2$$

$$CV_\star(E_2) = \frac{\sqrt{var_\star(E_2)}}{E_\star(E_2)}$$

and $var_\star$ and $E_\star$ are evaluated with respect to the bootstrapping process, given the selected sample.

The simulation CV, denoted $CV_\star(v_B(\hat{\theta}))$, is estimated for a given number of replicates B by estimating $CV_\star(E_2)$ using observed values and dividing this by $\sqrt{B}$. If the bootstrap errors are assumed to be normally distributed, then $CV_\star(E_2) = \sqrt{2}$ and so $CV_\star(v_B(\hat{\theta}))$ would not need to be estimated. Using observed replicate estimates to estimate the simulation CV instead of assuming normality allows simulation CV to be used for a a wide array of bootstrap methods.

References

See Section 3.3 and Section 8 of Beaumont and Patak (2012) for details and an example where the simulation CV is used to determine the number of bootstrap replicates needed for various alternative bootstrap methods in an empirical illustration.

Beaumont, J.-F. and Z. Patak. (2012), "On the Generalized Bootstrap for Sample Surveys with Special Attention to Poisson Sampling." **International Statistical Review**, *80*: 127-148. doi:10.1111/j.17515823.2011.00166.x.

See Also

Use `estimate_boot_reps_for_target_cv` to help choose the number of bootstrap replicates.

Examples

```
## Not run:
set.seed(2022)

# Create an example bootstrap survey design object ----
library(survey)
data('api', package = 'survey')

boot_design <- svydesign(id=~1,strata=~stype, weights=~pw,
                         data=apistrat, fpc=~fpc) |>
 svrep::as_bootstrap_design(replicates = 5000)

# Calculate estimates of interest and retain estimates from each replicate ----

estimated_means_and_proportions <- svymean(x = ~ api00 + api99 + stype, design = boot_design,
                                           return.replicates = TRUE)
custom_statistic <- withReplicates(design = boot_design,
                                   return.replicates = TRUE,
                                   theta = function(wts, data) {
                                      numerator <- sum(data$api00 * wts)
                                      denominator <- sum(data$api99 * wts)
                                      statistic <- numerator/denominator
                                      return(statistic)
                                   })
# Estimate simulation CV of bootstrap estimates ----

  estimate_boot_sim_cv(
    svrepstat = estimated_means_and_proportions
  )

  estimate_boot_sim_cv(
    svrepstat = custom_statistic
  )

## End(Not run)
```

`libraries` *Public Libraries Survey (PLS): A Census of U.S. Public Libraries in FY2020*

Description

Data taken from a complete census of public libraries in the United States in FY2020 (April 2020 to March 2021). The Public Libraries Survey (PLS) is an annual census of public libraries in the U.S., including all public libraries identified by state library administrative agencies in the 50 states, the District of Columbia, and the outlying territories of American Samoa, Guam, the Northern Mariana Islands, and the U.S. Virgin Islands (Puerto Rico did not participate in FY2020).

The primary dataset, `library_census`, represents the full microdata from the census. The datasets

`library_multistage_sample` and `library_stsys_sample` are samples drawn from `library_census` using different sampling methods.

Usage

```
data(library_census)

data(library_multistage_sample)

data(library_stsys_sample)
```

Format

Library Census (`library_census`*):*

The dataset includes 9,245 records (one per library) and 23 variables. Each column has a variable label, accessible using the function `var_label()` from the 'labelled' package or simply by calling `attr(x, 'label')` to a given column. These data include a subset of the variables included in the public-use data published by PLS, specifically from the Public Library System Data File. Particularly relevant variables include:

Identifier variables and survey response status:

- FSCSKEY: A unique identifier for libraries.
- LIBNAME: The name of the library
- RESPONSE_STATUS: Response status for the Public Library Survey: indicates whether the library was a respondent, nonrespondent, or was closed.

Numeric summaries:

- TOTCIR: Total circulation
- VISITS: Total visitors
- REGBOR: Total number of registered users
- TOTSTAFF: Total staff (measured in full-time equivalent staff)
- LIBRARIA: Total librarians (measured in full-time equivalent staff)
- TOTOPEXP: Total operating expenses
- TOTINCM: Total income
- BRANLIB: Number of library branches
- CENTLIB: Number of central library locations

Location:

- LONGITUD: Geocoded longitude (in WGS84 CRS)
- LATITUD: Geocoded latitude (in WGS84 CRS)
- STABR: Two-letter state abbreviation
- CBSA: Five-digit identifer for a core-based statistical area (CBSA)

- MICROF: Flag for a metropolitan or micropolitan statistical area

Library Multistage Sample (`library_multistage_sample`*):*

These data represent a two-stage sample (PSUs and SSUs), where the first stage sample is selected using unequal probability sampling without replacement (PPSWOR) and the second stage sample is selected using simple random sampling without replacement (SRSWOR).

Includes the same variables as `library_census`, but with additional design variables.

- PSU_ID: A unique identifier for primary sampling units
- SSU_ID: A unique identifer for secondary sampling units
- SAMPLING_PROB: Overall inclusion probability
- PSU_SAMPLING_PROB: Inclusion probability for the PSU
- SSU_SAMPLING_PROB: Inclusion probability for the SSU
- PSU_POP_SIZE: The number of PSUs in the population
- SSU_POP_SIZE: The number of population SSUs within the PSU

Library Stratified Systematic Sample (`library_stsys_sample`*):*

These data represent a stratified systematic sample.

Includes the same variables as `library_census`, but with additional design variables.

- SAMPLING_STRATUM: Unique identifier for sampling strata
- STRATUM_POP_SIZE: The population size in the stratum
- SAMPLING_SORT_ORDER: The sort order used before selecting a random systematic sample
- SAMPLING_PROB: Overall inclusion probability

References

Pelczar, M., Soffronoff, J., Nielsen, E., Li, J., & Mabile, S. (2022). Data File Documentation: Public Libraries in the United States Fiscal Year 2020. Institute of Museum and Library Services: Washington, D.C.

`lou_pums_microdata` *ACS PUMS Data for Louisville*

Description

Person-level microdata from the American Community Survey (ACS) 2015-2019 public-use microdata sample (PUMS) data for Louisville, KY. This microdata sample represents all adults (persons aged 18 or over) in Louisville, KY.

These data include replicate weights to use for variance estimation.

Usage

```
data(lou_pums_microdata)
```

Format

A data frame with 80 rows and 85 variables

- UNIQUE_ID: Unique identifier for records
- AGE: Age in years (copied from the AGEP variable in the ACS microdata)
- RACE_ETHNICITY: Race and Hispanic/Latino ethnicity derived from RAC1P and HISP variables of ACS microdata and collapsed to a smaller number of categories.
- SEX: Male or Female
- EDUC_ATTAINMENT: Highest level of education attained ('Less than high school' or 'High school or beyond') derived from SCHL variable in ACS microdata and collapsed to a smaller number of categories.
- PWGTP: Weights for the full-sample
- PWGTP1-PWGTP80: 80 columns of replicate weights created using the Successive Differences Replication (SDR) method.

Examples

```
## Not run:
data(lou_pums_microdata)

# Prepare the data for analysis with the survey package
  library(survey)

  lou_pums_rep_design <- survey::svrepdesign(
    data = lou_pums_microdata,
    variables = ~ UNIQUE_ID + AGE + SEX + RACE_ETHNICITY + EDUC_ATTAINMENT,
    weights = ~ PWGTP, repweights = "PWGTP\\d{1,2}",
    type = "successive-difference",
    mse = TRUE
  )

# Estimate population proportions
  svymean(~ SEX, design = lou_pums_rep_design)

## End(Not run)
```

`lou_vax_survey` *Louisville Vaccination Survey*

Description

A survey measuring Covid-19 vaccination status and a handful of demographic variables, based on a simple random sample of 1,000 residents of Louisville, Kentucky with an approximately 50% response rate.

These data were created using simulation.

Usage

```
data(lou_vax_survey)
```

Format

A data frame with 1,000 rows and 6 variables

RESPONSE_STATUS Response status to the survey ('Respondent' or 'Nonrespondent')

RACE_ETHNICITY Race and Hispanic/Latino ethnicity derived from RAC1P and HISP variables of ACS microdata and collapsed to a smaller number of categories.

SEX Male or Female

EDUC_ATTAINMENT Highest level of education attained ('Less than high school' or 'High school or beyond') derived from SCHL variable in ACS microdata and collapsed to a smaller number of categories.

VAX_STATUS Covid-19 vaccination status ('Vaccinated' or 'Unvaccinated')

SAMPLING_WEIGHT Sampling weight: equal for all cases since data come from a simple random sample

---

lou_vax_survey_control_totals

Control totals for the Louisville Vaccination Survey

Description

Control totals to use for raking or post-stratification for the Louisville Vaccination Survey data. Control totals are population size estimates from the ACS 2015-2019 5-year Public Use Microdata Sample (PUMS) for specific demographic categories among adults in Jefferson County, KY.

These data were created using simulation.

Usage

```
data(lou_vax_survey_control_totals)
```

Format

A nested list object with two lists, `poststratification` and `raking`, each of which contains two elements: `estimates` and `variance-covariance`.

poststratification Control totals for the combination of `RACE_ETHNICITY`, `SEX`, and `EDUC_ATTAINMENT`.

- estimates: A numeric vector of estimated population totals.
- variance-covariance: A variance-covariance matrix for the estimated population totals.

raking Separate control totals for each of `RACE_ETHNICITY`, `SEX`, and `EDUC_ATTAINMENT`.

- estimates: A numeric vector of estimated population totals.
- variance-covariance: A variance-covariance matrix for the estimated population totals.

`make_gen_boot_factors` *Creates replicate factors for the generalized survey bootstrap*

Description

Creates replicate factors for the generalized survey bootstrap method. The generalized survey bootstrap is a method for forming bootstrap replicate weights from a textbook variance estimator, provided that the variance estimator can be represented as a quadratic form whose matrix is positive semi-definite (this covers a large class of variance estimators).

Usage

```
make_gen_boot_factors(Sigma, num_replicates, tau = "auto")
```

Arguments

`Sigma`	The matrix of the quadratic form used to represent the variance estimator. Must be positive semi-definite.
`num_replicates`	The number of bootstrap replicates to create.
`tau`	Either `"auto"`, or a single number. This is the rescaling constant used to avoid negative weights through the transformation $\frac{w+\tau-1}{\tau}$, where w is the original weight and τ is the rescaling constant `tau`. If `tau="auto"`, the rescaling factor is determined automatically as follows: if all of the adjustment factors are nonnegative, then `tau` is set equal to 1; otherwise, `tau` is set to the smallest value needed to rescale the adjustment factors such that they are all at least `0.01`.

Value

A matrix with the same number of rows as `Sigma`, and the number of columns equal to `num_replicates`. The object has an attribute named `tau` which can be retrieved by calling `attr(which = 'tau')` on the object. The value `tau` is a rescaling factor which was used to avoid negative weights.

In addition, the object has attributes named `scale` and `rscales` which can be passed directly to svrepdesign. Note that the value of `scale` is τ^2/B, while the value of `rscales` is vector of length B, with every entry equal to 1.

Statistical Details

Let $v(\hat{T}_y)$ be the textbook variance estimator for an estimated population total $\hat{T}_y$ of some variable y. The base weight for case i in our sample is w_i, and we let $\breve{y}_i$ denote the weighted value $w_i y_i$. Suppose we can represent our textbook variance estimator as a quadratic form: $v(\hat{T}_y) = \breve{y}\Sigma\breve{y}^T$, for some $n \times n$ matrix Σ. The only constraint on Σ is that, for our sample, it must be symmetric and positive semi-definite.

The bootstrapping process creates B sets of replicate weights, where the b-th set of replicate weights is a vector of length n denoted $\mathbf{a}^{(b)}$, whose k-th value is denoted $a_k^{(b)}$. This yields B replicate estimates of the population total, $\hat{T}_y^{*(b)} = \sum_{k \in s} a_k^{(b)} \breve{y}_k$, for $b = 1, \ldots B$, which can be used to estimate sampling variance.

$$v_B\left(\hat{T}_y\right) = \frac{\sum_{b=1}^{B}\left(\hat{T}_y^{*(b)} - \hat{T}_y\right)^2}{B}$$

This bootstrap variance estimator can be written as a quadratic form:

$$v_B\left(\hat{T}_y\right) = \breve{\mathbf{y}}'\Sigma_B\breve{\mathbf{y}}$$

where

$$\mathbf{\Sigma}_B = \frac{\sum_{b=1}^{B}\left(\mathbf{a}^{(b)} - \mathbf{1}_n\right)\left(\mathbf{a}^{(b)} - \mathbf{1}_n\right)'}{B}$$

Note that if the vector of adjustment factors $\mathbf{a}^{(b)}$ has expectation $\mathbf{1}_n$ and variance-covariance matrix $\mathbf{\Sigma}$, then we have the bootstrap expectation $E_*\left(\mathbf{\Sigma}_B\right) = \mathbf{\Sigma}$. Since the bootstrap process takes the sample values $\breve{y}$ as fixed, the bootstrap expectation of the variance estimator is $E_*\left(\breve{\mathbf{y}}'\Sigma_B\breve{\mathbf{y}}\right) = \breve{\mathbf{y}}'\Sigma\breve{\mathbf{y}}$. Thus, we can produce a bootstrap variance estimator with the same expectation as the textbook variance estimator simply by randomly generating $\mathbf{a}^{(b)}$ from a distribution with the following two conditions:

Condition 1: $\mathbf{E}_*(\mathbf{a}) = \mathbf{1}_n$
Condition 2: $\mathbf{E}_*\left(\mathbf{a} - \mathbf{1}_n\right)\left(\mathbf{a} - \mathbf{1}_n\right)' = \mathbf{\Sigma}$

While there are multiple ways to generate adjustment factors satisfying these conditions, the simplest general method is to simulate from a multivariate normal distribution: $\mathbf{a} \sim MVN(\mathbf{1}_n, \mathbf{\Sigma})$. This is the method used by this function.

Details on Rescaling to Avoid Negative Adjustment Factors

Let $\mathbf{A} = \left[\mathbf{a}^{(1)} \cdots \mathbf{a}^{(b)} \cdots \mathbf{a}^{(B)}\right]$ denote the $(n \times B)$ matrix of bootstrap adjustment factors. To eliminate negative adjustment factors, Beaumont and Patak (2012) propose forming a rescaled matrix of nonnegative replicate factors $\mathbf{A}^S$ by rescaling each adjustment factor $a_k^{(b)}$ as follows:

$$a_k^{S,(b)} = \frac{a_k^{(b)} + \tau - 1}{\tau}$$

where $\tau \geq 1 - a_k^{(b)} \geq 1$ for all k in $\{1, \ldots, n\}$ and all b in $\{1, \ldots, B\}$.

The value of τ can be set based on the realized adjustment factor matrix $\mathbf{A}$ or by choosing τ prior to generating the adjustment factor matrix $\mathbf{A}$ so that τ is likely to be large enough to prevent negative bootstrap weights.

If the adjustment factors are rescaled in this manner, it is important to adjust the scale factor used in estimating the variance with the bootstrap replicates, which becomes $\frac{\tau^2}{B}$ instead of $\frac{1}{B}$.

$$\textbf{Prior to rescaling: } v_B\left(\hat{T}_y\right) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{T}_y^{*(b)} - \hat{T}_y\right)^2$$

$$\textbf{After rescaling: } v_B\left(\hat{T}_y\right) = \frac{\tau^2}{B}\sum_{b=1}^{B}\left(\hat{T}_y^{S*(b)} - \hat{T}_y\right)^2$$

When sharing a dataset that uses rescaled weights from a generalized survey bootstrap, the documentation for the dataset should instruct the user to use replication scale factor $\frac{\tau^2}{B}$ rather than $\frac{1}{B}$ when estimating sampling variances.

References

The generalized survey bootstrap was first proposed by Bertail and Combris (1997). See Beaumont and Patak (2012) for a clear overview of the generalized survey bootstrap. The generalized survey bootstrap represents one strategy for forming replication variance estimators in the general framework proposed by Fay (1984) and Dippo, Fay, and Morganstein (1984).

- Beaumont, Jean-François, and Zdenek Patak. 2012. "On the Generalized Bootstrap for Sample Surveys with Special Attention to Poisson Sampling: Generalized Bootstrap for Sample Surveys." International Statistical Review 80 (1): 127–48. https://doi.org/10.1111/j.1751-5823.2011.00166.x.

- Bertail, and Combris. 1997. "Bootstrap Généralisé d'un Sondage." Annales d'Économie Et de Statistique, no. 46: 49. https://doi.org/10.2307/20076068.

- Dippo, Cathryn, Robert Fay, and David Morganstein. 1984. "Computing Variances from Complex Samples with Replicate Weights." In, 489–94. Alexandria, VA: American Statistical Association. http://www.asasrms.org/Proceedings/papers/1984_094.pdf.

- Fay, Robert. 1984. "Some Properties of Estimates of Variance Based on Replication Methods." In, 495–500. Alexandria, VA: American Statistical Association. http://www.asasrms.org/Proceedings/papers/1984_095.pdf.

See Also

The function `make_quad_form_matrix` can be used to represent several common variance estimators as a quadratic form's matrix, which can then be used as an input to `make_gen_boot_factors()`.

Examples

```
## Not run:
  library(survey)

# Load an example dataset that uses unequal probability sampling ----
```

```
  data('election', package = 'survey')

# Create matrix to represent the Horvitz-Thompson estimator as a quadratic form ----
  n <- nrow(election_pps)
  pi <- election_jointprob
  horvitz_thompson_matrix <- matrix(nrow = n, ncol = n)
  for (i in seq_len(n)) {
    for (j in seq_len(n)) {
      horvitz_thompson_matrix[i,j] <- 1 - (pi[i,i] * pi[j,j])/pi[i,j]
    }
  }

  ## Equivalently:

  horvitz_thompson_matrix <- make_quad_form_matrix(
    variance_estimator = "Horvitz-Thompson",
    joint_probs = election_jointprob
  )

# Make generalized bootstrap adjustment factors ----

  bootstrap_adjustment_factors <- make_gen_boot_factors(
    Sigma = horvitz_thompson_matrix,
    num_replicates = 80,
    tau = 'auto'
  )

# Determine replication scale factor for variance estimation ----
  tau <- attr(bootstrap_adjustment_factors, 'tau')
  B <- ncol(bootstrap_adjustment_factors)
  replication_scaling_constant <- tau^2 / B

# Create a replicate design object ----
  election_pps_bootstrap_design <- svrepdesign(
    data = election_pps,
    weights = 1 / diag(election_jointprob),
    repweights = bootstrap_adjustment_factors,
    combined.weights = FALSE,
    type = "other",
    scale = attr(bootstrap_adjustment_factors, 'scale'),
    rscales = attr(bootstrap_adjustment_factors, 'rscales')
  )

# Compare estimates to Horvitz-Thompson estimator ----

  election_pps_ht_design <- svydesign(
    id = ~1,
    fpc = ~p,
    data = election_pps,
    pps = ppsmat(election_jointprob),
    variance = "HT"
  )
```

```
svytotal(x = ~ Bush + Kerry,
         design = election_pps_bootstrap_design)
svytotal(x = ~ Bush + Kerry,
         design = election_pps_ht_design)

## End(Not run)
```

`make_quad_form_matrix` *Represent a variance estimator as a quadratic form*

Description

Common variance estimators for estimated population totals can be represented as a quadratic form. Given a choice of variance estimator and information about the sample design, this function constructs the matrix of the quadratic form.

In notation, let $v(\hat{Y}) = \breve{\mathbf{y}}'\mathbf{\Sigma}\breve{\mathbf{y}}$, where $\breve{y}$ is the vector of weighted values, $y_i/\pi_i, i = 1, \ldots, n$. This function constructs the $n \times n$ matrix of the quadratic form, $\mathbf{\Sigma}$.

Usage

```
make_quad_form_matrix(
  variance_estimator = "Yates-Grundy",
  joint_probs = NULL,
  cluster_ids = NULL,
  strata_ids = NULL,
  strata_pop_sizes = NULL,
  sort_order = NULL
)
```

Arguments

`variance_estimator`

The name of the variance estimator whose quadratic form matrix should be created. See the section "Variance Estimators" below. Options include:

- **"Yates-Grundy"**: The Yates-Grundy variance estimator based on first-order and second-order inclusion probabilities. If this is used, the argument `joint_probs` must also be used.
- **"Horvitz-Thompson"**: The Horvitz-Thompson variance estimator based on first-order and second-order inclusion probabilities. If this is used, the argument `joint_probs` must also be used.
- **"Stratified Multistage SRS"**: The usual stratified multistage variance estimator based on estimating the variance of cluster totals within strata at each stage. If this option is used, then it is necessary to also use the arguments `strata_ids`, `cluster_ids`, `strata_samp_sizes`, and `strata_pop_sizes`.

- **"Ultimate Cluster"**: The usual variance estimator based on estimating the variance of first-stage cluster totals within first-stage strata. If this option is used, then it is necessary to also use the arguments `strata_ids`, `cluster_ids`, `strata_samp_sizes`. Optionally, to use finite population correction factors, one can also use the argument `strata_pop_sizes`.
- **"SD1"**: The non-circular successive-differences variance estimator described by Ash (2014), sometimes used for variance estimation for systematic sampling.
- **"SD2"**: The circular successive-differences variance estimator described by Ash (2014). This estimator is the basis of the "successive-differences replication" estimator commonly used for variance estimation for systematic sampling.

`joint_probs` Only used if `variance_estimator = "Horvitz-Thompson"` or `variance_estimator = "Yates-Grundy"`. This should be a matrix of joint inclusion probabilities. Element `[i,i]` of the matrix is the first-order inclusion probability of unit `i`, while element `[i,j]` is the joint inclusion probability of units `i` and `j`.

`cluster_ids` Required unless `variance_estimator` equals `"Horvitz-Thompson"` or `"Yates-Grundy"`. This should be a matrix or data frame of cluster IDs. If there are multiple stages of sampling, then `cluster_ids` can have multiple columns, with one column for each level of sampling to be accounted for by the variance estimator.

`strata_ids` Required if `variance_estimator` equals `"Stratified Multistage SRS"` or `"Ultimate Cluster"`. This should be a matrix or data frame of strata IDs. If there are multiple stages of sampling, then `strata_ids` can have multiple columns, with one column for each level of sampling to be accounted for by the variance estimator.

`strata_pop_sizes` Required if `variance_estimator` equals `"Stratified Multistage SRS"`, but can optionally be used if `variance_estimator` equals `"Ultimate Cluster"`, `"SD1"`, or `"SD2"`. If there are multiple stages of sampling, then `strata_pop_sizes` can have multiple columns, with one column for each level of sampling to be accounted for by the variance estimator.

`sort_order` Required if `variance_estimator` equals `"SD1"` or `"SD2"`. This should be a vector that orders the rows of data into the order used for sampling.

Value

The matrix of the quadratic form representing the variance estimator.

Arguments required for each variance estimator

Below are the arguments that are required or optional for each variance estimator.

variance_estimator	joint_probs	cluster_ids	strata_ids	strata_pop_sizes	sort_order
Yates-Grundy	Required				
Horvitz-Thompson	Required				
Stratified Multistage SRS		Required	Required	Required	
Ultimate Cluster		Required	Required	Optional	
SD1		Required	Optional	Optional	Required

SD2	Required	Optional	Optional	Required

Variance Estimators

The **Horvitz-Thompson** variance estimator:

$$v(\hat{Y}) = \sum_{i \in s} \sum_{j \in s} (1 - \frac{\pi_i \pi_j}{\pi_{ij}}) \frac{y_i}{\pi_i} \frac{y_j}{\pi_j}$$

The **Yates-Grundy** variance estimator:

$$v(\hat{Y}) = -\frac{1}{2} \sum_{i \in s} \sum_{j \in s} (1 - \frac{\pi_i \pi_j}{\pi_{ij}}) (\frac{y_i}{\pi_i} - \frac{y_j}{\pi_j})^2$$

The **Stratified Multistage SRS** variance estimator is the recursive variance estimator proposed by Bellhouse (1985) and used in the 'survey' package's function svyrecvar. The estimator can be used for any number of sampling stages. For illustration, we describe its use for two sampling stages.

$$v(\hat{Y}) = \hat{V}_1 + \hat{V}_2$$

where

$$\hat{V}_1 = \sum_{h=1}^{H} (1 - \frac{n_h}{N_h}) \frac{n_h}{n_h - 1} \sum_{i=1}^{n_h} (y_{hi.} - \bar{y}_{hi.})^2$$

and

$$\hat{V}_2 = \sum_{h=1}^{H} \frac{n_h}{N_h} \sum_{i=1}^{n_h} v_{hi}(y_{hi.})$$

where n_h is the number of sampled clusters in stratum h, N_h is the number of population clusters in stratum h, $y_{hi.}$ is the weighted cluster total in cluster i of stratum h, $\bar{y}_{hi.}$ is the mean weighted cluster total of stratum h, ($\bar{y}_{hi.} = \frac{1}{n_h} \sum_{i=1}^{n_h} y_{hi.}$), and $v_{hi}(y_{hi.})$ is the estimated sampling variance of $y_{hi.}$.

The **Ultimate Cluster** variance estimator is simply the stratified multistage SRS variance estimator, but ignoring variances from later stages of sampling.

$$v(\hat{Y}) = \hat{V}_1$$

This is the variance estimator used in the 'survey' package when the user specifies `option(survey.ultimate.cluster = TRUE)` or uses `svyrecvar(..., one.stage = TRUE)`. When the first-stage sampling fractions are small, analysts often omit the finite population corrections $(1 - \frac{n_h}{N_h})$ when using the ultimate cluster estimator.

The **SD1** and **SD2** variance estimators are "successive difference" estimators sometimes used for systematic sampling designs. Ash (2014) describes each estimator as follows:

$$\hat{v}_{SD1}(\hat{Y}) = \left(1 - \frac{n}{N}\right) \frac{n}{2(n-1)} \sum_{k=2}^{n} (\breve{y}_k - \breve{y}_{k-1})^2$$

$$\hat{v}_{SD2}(\hat{Y}) = \left(1 - \frac{n}{N}\right) \frac{1}{2} \left[\sum_{k=2}^{n} \left(\breve{y}_k - \breve{y}_{k-1}\right)^2 + \left(\breve{y}_n - \breve{y}_1\right)^2\right]$$

where $\breve{y}_k = y_k/\pi_k$ is the weighted value of unit k with selection probability π_k. The SD1 estimator is recommended by Wolter (1984). The SD2 estimator is the basis of the successive difference replication estimator commonly used for systematic sampling designs. See Ash (2014) for details.

For multistage samples, SD1 and SD2 are applied to the clusters at each stage, separately by stratum. For later stages of sampling, the variance estimate from a stratum is multiplied by the product of sampling fractions from earlier stages of sampling. For example, at a third stage of sampling, the variance estimate from a third-stage stratum is multiplied by $\frac{n_1}{N_1}\frac{n_2}{N_2}$, which is the product of sampling fractions from the first-stage stratum and second-stage stratum.

References

Ash, S. (2014). "*Using successive difference replication for estimating variances*." **Survey Methodology**, Statistics Canada, 40(1), 47–59.

Bellhouse, D.R. (1985). "*Computing Methods for Variance Estimation in Complex Surveys*." **Journal of Official Statistics**, Vol.1, No.3.

Examples

```
## Not run:
# Example 1: The Horvitz-Thompson Estimator
  library(survey)
  data("election", package = "survey")

  ht_quad_form_matrix <- make_quad_form_matrix(variance_estimator = "Horvitz-Thompson",
                                               joint_probs = election_jointprob)
  ##_ Produce variance estimate
  wtd_y <- as.matrix(election_pps$wt * election_pps$Bush)
  t(wtd_y) %*% ht_quad_form_matrix %*% wtd_y

  ##_ Compare against result from 'survey' package
  svytotal(x = ~ Bush,
           design = svydesign(data=election_pps,
                              variance = "HT",
                              pps = ppsmat(election_jointprob),
                              ids = ~ 1, fpc = ~ p)) |> vcov()

# Example 2: Stratified multistage Sample ----

  data("mu284", package = 'survey')
  multistage_srswor_design <- svydesign(data = mu284,
                                        ids = ~ id1 + id2,
                                        fpc = ~ n1 + n2)

  multistage_srs_quad_form <- make_quad_form_matrix(
    variance_estimator = "Stratified Multistage SRS",
    cluster_ids = mu284[,c('id1', 'id2')],
    strata_ids = matrix(1, nrow = nrow(mu284), ncol = 2),
```

```
    strata_pop_sizes = mu284[,c('n1', 'n2')]
  )

  wtd_y <- as.matrix(weights(multistage_srswor_design) * mu284$y1)
  t(wtd_y) %*% multistage_srs_quad_form %*% wtd_y

  svytotal(x = ~ y1, design = multistage_srswor_design) |> vcov()

# Example 3: Successive-differences estimator ----

  data('library_stsys_sample', package = 'svrep')

  sd1_quad_form <- make_quad_form_matrix(
    variance_estimator = 'SD1',
    cluster_ids = library_stsys_sample[,'FSCSKEY',drop=FALSE],
    strata_ids = library_stsys_sample[,'SAMPLING_STRATUM',drop=FALSE],
    strata_pop_sizes = library_stsys_sample[,'STRATUM_POP_SIZE',drop=FALSE],
    sort_order = library_stsys_sample[['SAMPLING_SORT_ORDER']]
  )

  wtd_y <- as.matrix(library_stsys_sample[['TOTCIR']] /
                     library_stsys_sample$SAMPLING_PROB)
  wtd_y[is.na(wtd_y)] <- 0

t(wtd_y) %*% sd1_quad_form %*% wtd_y

## End(Not run)
```

`make_rwyb_bootstrap_weights` *Create bootstrap replicate weights for a general survey design, using the Rao-Wu-Yue-Beaumont bootstrap method*

Description

Creates bootstrap replicate weights for a multistage stratified sample design using the method of Beaumont and Émond (2022), which is a generalization of the Rao-Wu-Yue bootstrap.

The design may have different sampling methods used at different stages. Each stage of sampling may potentially use unequal probabilities (with or without replacement) and may potentially use Poisson sampling.

Usage

```
make_rwyb_bootstrap_weights(
  num_replicates = 100,
  samp_unit_ids,
  strata_ids,
  samp_unit_sel_probs,
```

```
  samp_method_by_stage = rep("PPSWOR", times = ncol(samp_unit_ids)),
  allow_final_stage_singletons = TRUE,
  output = "weights"
)
```

Arguments

num_replicates	Positive integer giving the number of bootstrap replicates to create
samp_unit_ids	Matrix or data frame of sampling unit IDs for each stage of sampling
strata_ids	Matrix or data frame of strata IDs for each sampling unit at each stage of sampling
samp_unit_sel_probs	Matrix or data frame of selection probabilities for each sampling unit at each stage of sampling.
samp_method_by_stage	A vector with length equal to the number of stages of sampling, corresponding to the number of columns in samp_unit_ids. This describes the method of sampling used at each stage. Each element should be one of the following: • "SRSWOR" - Simple random sampling, without replacement • "SRSWR" - Simple random sampling, with replacement • "PPSWOR" - Unequal probabilities of selection, without replacement • "PPSWR" - Unequal probabilities of selection, with replacement • "Poisson" - Poisson sampling: each sampling unit is selected into the sample at most once, with potentially different probabilities of inclusion for each sampling unit.
allow_final_stage_singletons	Logical value indicating whether to allow non-certainty singleton strata at the final sampling stage (rather than throw an error message). If TRUE, the sampling unit in a non-certainty singleton stratum will have its final-stage adjustment factor calculated as if it was selected with certainty at the final stage (i.e., its adjustment factor will be 1), and then its final bootstrap weight will be calculated by combining this adjustment factor with its final-stage selection probability.
output	Either "weights" (the default) or "factors". Specifying output = "factors" returns a matrix of replicate adjustment factors which can later be multiplied by the full-sample weights to produce a matrix of replicate weights. Specifying output = "weights" returns the matrix of replicate weights, where the full-sample weights are inferred using samp_unit_sel_probs.

Details

Beaumont and Émond (2022) describe a general algorithm for forming bootstrap replicate weights for multistage stratified samples, based on the method of Rao-Wu-Yue, with extensions to sampling without replacement and use of unequal probabilities of selection (i.e., sampling with probability proportional to size) as well as Poisson sampling. These methods are guaranteed to produce non-negative replicate weights and provide design-unbiased and design-consistent variance estimates for totals, for designs where sampling uses one or more of the following methods:

- "SRSWOR" - Simple random sampling, without replacement
- "SRSWR" - Simple random sampling, with replacement
- "PPSWR" - Unequal probabilities of selection, with replacement
- "Poisson" - Poisson sampling: each sampling unit is selected into the sample at most once, with potentially different probabilities of inclusion for each sampling unit.

For designs where at least one stage's strata have sampling without replacement with unequal probabilities of selection ("PPSWOR"), the bootstrap method of Beaumont and Émond (2022) is guaranteed to produce nonnegative weights, but is not design-unbiased, since the method only approximates the joint selection probabilities which would be needed for unbiased estimation.

Unless any stages use simple random sampling without replacement, the resulting bootstrap replicate weights are guaranteed to all be strictly positive, which may be useful for calibration or analyses of domains with small sample sizes. If any stages use simple random sampling without replacement, it is possible for some replicate weights to be zero.

If there is survey nonresponse, it may be useful to represent the response/nonresponse as an additional stage of sampling, where sampling is conducted with Poisson sampling where each unit's "selection probability" at that stage is its response propensity (which typically has to be estimated).

Value

A matrix of with the same number of rows as `samp_unit_ids` and the number of columns equal to the value of the argument `num_replicates`. Specifying `output = "factors"` returns a matrix of replicate adjustment factors which can later be multiplied by the full-sample weights to produce a matrix of replicate weights. Specifying `output = "weights"` returns the matrix of replicate weights, where the full-sample weights are inferred using `samp_unit_sel_probs`.

References

Beaumont, J.-F.; Émond, N. (2022). "A Bootstrap Variance Estimation Method for Multistage Sampling and Two-Phase Sampling When Poisson Sampling Is Used at the Second Phase." **Stats**, *5*: 339–357. https://doi.org/10.3390/stats5020019

Rao, J.N.K.; Wu, C.F.J.; Yue, K. (1992). "Some recent work on resampling methods for complex surveys." **Surv. Methodol.**, *18*: 209–217.

See Also

If the survey design can be accurately represented using `svydesign`, then it is easier to simply use `as_bootstrap_design` with argument `type = "Rao-Wu-Yue-Beaumont"`.

Use `estimate_boot_reps_for_target_cv` to help choose the number of bootstrap replicates.

Examples

```
## Not run:
 library(survey)

 # Example 1: A multistage sample with two stages of SRSWOR
```

```
## Load an example dataset from a multistage sample, with two stages of SRSWOR
data("mu284", package = 'survey')
multistage_srswor_design <- svydesign(data = mu284,
                                      ids = ~ id1 + id2,
                                      fpc = ~ n1 + n2)

## Create bootstrap replicate weights
set.seed(2022)
bootstrap_replicate_weights <- make_rwyb_bootstrap_weights(
  num_replicates = 5000,
  samp_unit_ids = multistage_srswor_design$cluster,
  strata_ids = multistage_srswor_design$strata,
  samp_unit_sel_probs = multistage_srswor_design$fpc$sampsize /
                        multistage_srswor_design$fpc$popsize,
  samp_method_by_stage = c("SRSWOR", "SRSWOR")
)

## Create a replicate design object with the survey package
bootstrap_rep_design <- svrepdesign(
  data = multistage_srswor_design$variables,
  repweights = bootstrap_replicate_weights,
  weights = weights(multistage_srswor_design, type = "sampling"),
  type = "bootstrap"
)

## Compare std. error estimates from bootstrap versus linearization
data.frame(
  'Statistic' = c('total', 'mean', 'median'),
  'SE (bootstrap)' = c(SE(svytotal(x = ~ y1, design = bootstrap_rep_design)),
                       SE(svymean(x = ~ y1, design = bootstrap_rep_design)),
                       SE(svyquantile(x = ~ y1, quantile = 0.5,
                                      design = bootstrap_rep_design))),
 'SE (linearization)' = c(SE(svytotal(x = ~ y1, design = multistage_srswor_design)),
                          SE(svymean(x = ~ y1, design = multistage_srswor_design)),
                           SE(svyquantile(x = ~ y1, quantile = 0.5,
                                          design = multistage_srswor_design))),
  check.names = FALSE
)

# Example 2: A single-stage sample selected with unequal probabilities, without replacement

## Load an example dataset of U.S. counties states with 2004 Presidential vote counts
data("election", package = 'survey')
pps_wor_design <- svydesign(data = election_pps,
                            pps = "overton",
                            fpc = ~ p, # Inclusion probabilities
                            ids = ~ 1)

## Create bootstrap replicate weights
set.seed(2022)
bootstrap_replicate_weights <- make_rwyb_bootstrap_weights(
  num_replicates = 5000,
  samp_unit_ids = pps_wor_design$cluster,
```

```
  strata_ids = pps_wor_design$strata,
  samp_unit_sel_probs = pps_wor_design$prob,
  samp_method_by_stage = c("PPSWOR")
)

## Create a replicate design object with the survey package
bootstrap_rep_design <- svrepdesign(
  data = pps_wor_design$variables,
  repweights = bootstrap_replicate_weights,
  weights = weights(pps_wor_design, type = "sampling"),
  type = "bootstrap"
)

## Compare std. error estimates from bootstrap versus linearization
data.frame(
  'Statistic' = c('total', 'mean'),
  'SE (bootstrap)' = c(SE(svytotal(x = ~ Bush, design = bootstrap_rep_design)),
                       SE(svymean(x = ~ I(Bush/votes),
                                  design = bootstrap_rep_design))),
  'SE (Overton\'s PPS approximation)' = c(SE(svytotal(x = ~ Bush,
                                                      design = pps_wor_design)),
                                          SE(svymean(x = ~ I(Bush/votes),
                                                     design = pps_wor_design))),
  check.names = FALSE
)

## End(Not run)
```

`redistribute_weights` *Redistribute weight from one group to another*

Description

Redistributes weight from one group to another: for example, from non-respondents to respondents. Redistribution is conducted for the full-sample weights as well as each set of replicate weights. This can be done separately for each combination of a set of grouping variables, for example to implement a nonresponse weighting class adjustment.

Usage

```
redistribute_weights(design, reduce_if, increase_if, by)
```

Arguments

`design`	A survey design object, created with either the `survey` or `srvyr` packages.
`reduce_if`	An expression indicating which cases should have their weights set to zero. Must evaluate to a logical vector with only values of TRUE or FALSE.
`increase_if`	An expression indicating which cases should have their weights increased. Must evaluate to a logical vector with only values of TRUE or FALSE.

by (Optional) A character vector with the names of variables used to group the redistribution of weights. For example, if the data include variables named "stratum" and "wt_class", one could specify by = c("stratum", "wt_class").

Value

The survey design object, but with updated full-sample weights and updated replicate weights. The resulting survey design object always has its value of combined.weights set to TRUE.

References

See Chapter 2 of Heeringa, West, and Berglund (2017) or Chapter 13 of Valliant, Dever, and Kreuter (2018) for an overview of nonresponse adjustment methods based on redistributing weights.

- Heeringa, S., West, B., Berglund, P. (2017). Applied Survey Data Analysis, 2nd edition. Boca Raton, FL: CRC Press. "Applied Survey Data Analysis, 2nd edition." Boca Raton, FL: CRC Press.

- Valliant, R., Dever, J., Kreuter, F. (2018). "Practical Tools for Designing and Weighting Survey Samples, 2nd edition." New York: Springer.

Examples

```
# Load example data
suppressPackageStartupMessages(library(survey))
data(api)

dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
dclus1$variables$response_status <- sample(x = c("Respondent", "Nonrespondent",
                                                 "Ineligible", "Unknown eligibility"),
                                           size = nrow(dclus1),
                                           replace = TRUE)
rep_design <- as.svrepdesign(dclus1)

# Adjust weights for cases with unknown eligibility
ue_adjusted_design <- redistribute_weights(
    design = rep_design,
    reduce_if = response_status %in% c("Unknown eligibility"),
    increase_if = !response_status %in% c("Unknown eligibility"),
    by = c("stype")
)

# Adjust weights for nonresponse
nr_adjusted_design <- redistribute_weights(
    design = ue_adjusted_design,
    reduce_if = response_status %in% c("Nonrespondent"),
    increase_if = response_status == "Respondent",
    by = c("stype")
)
```

stack_replicate_designs *Stack replicate designs, combining data and weights into a single object*

Description

Stack replicate designs: combine rows of data, rows of replicate weights, and the respective full-sample weights. This can be useful when comparing estimates before and after a set of adjustments made to the weights. Another more delicate application is when combining sets of replicate weights from multiple years of data for a survey, although this must be done carefully based on guidance from a data provider.

Usage

```
stack_replicate_designs(..., .id = "Design_Name")
```

Arguments

`...`	Replicate-weights survey design objects to combine. These can be supplied in one of two ways. • Option 1 - A series of design objects, for example `'adjusted' = adjusted_design, 'orig' = orig_design`. • Option 2 - A list object containing design objects, for example `list('nr' = nr_adjusted_design, 'ue' = ue_adjusted_design)`. All objects must have the same specifications for `type`, `rho`, `mse`, `scales`, and `rscales`.
`.id`	A single character value, which becomes the name of a new column of identifiers created in the output data to link each row to the design from which it was taken. The labels used for the identifiers are taken from named arguments.

Value

A replicate-weights survey design object, with class `svyrep.design` and `svyrep.stacked`. The resulting survey design object always has its value of `combined.weights` set to `TRUE`.

Examples

```
# Load example data, creating a replicate design object
suppressPackageStartupMessages(library(survey))
data(api)

dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
dclus1$variables$response_status <- sample(x = c("Respondent", "Nonrespondent",
                                                 "Ineligible", "Unknown eligibility"),
                                           size = nrow(dclus1),
```

```
                                            replace = TRUE)
orig_rep_design <- as.svrepdesign(dclus1)

# Adjust weights for cases with unknown eligibility
ue_adjusted_design <- redistribute_weights(
    design = orig_rep_design,
    reduce_if = response_status %in% c("Unknown eligibility"),
    increase_if = !response_status %in% c("Unknown eligibility"),
    by = c("stype")
)

# Adjust weights for nonresponse
nr_adjusted_design <- redistribute_weights(
    design = ue_adjusted_design,
    reduce_if = response_status %in% c("Nonrespondent"),
    increase_if = response_status == "Respondent",
    by = c("stype")
)

# Stack the three designs, using any of the following syntax options
stacked_design <- stack_replicate_designs(orig_rep_design, ue_adjusted_design, nr_adjusted_design,
                                          .id = "which_design")
stacked_design <- stack_replicate_designs('original' = orig_rep_design,
                                    'unknown eligibility adjusted' = ue_adjusted_design,
                                          'nonresponse adjusted' = nr_adjusted_design,
                                          .id = "which_design")
list_of_designs <- list('original' = orig_rep_design,
                        'unknown eligibility adjusted' = ue_adjusted_design,
                        'nonresponse adjusted' = nr_adjusted_design)
stacked_design <- stack_replicate_designs(list_of_designs, .id = "which_design")
```

`summarize_rep_weights` *Summarize the replicate weights*

Description

Summarize the replicate weights of a design

Usage

```
summarize_rep_weights(rep_design, type = "both", by)
```

Arguments

`rep_design`	A replicate design object, created with either the `survey` or `srvyr` packages.
`type`	Default is `"both"`. Use `type = "overall"`, for an overall summary of the replicate weights. Use `type = "specific"` for a summary of each column of replicate weights, with each column of replicate weights summarized in a given row of the summary.

Use type = "both" for a list containing both summaries, with the list containing the names "overall" and "both".

by (Optional) A character vector with the names of variables used to group the summaries.

Value

If type = "both" (the default), the result is a list of data frames with names "overall" and "specific". If type = "overall", the result is a data frame providing an overall summary of the replicate weights.

The contents of the "overall" summary are the following:

- "nrows": Number of rows for the weights
- "ncols": Number of columns of replicate weights
- "degf_svy_pkg": The degrees of freedom according to the survey package in R
- "rank": The matrix rank as determined by a QR decomposition
- "avg_wgt_sum": The average column sum
- "sd_wgt_sums": The standard deviation of the column sums
- "min_rep_wgt": The minimum value of any replicate weight
- "max_rep_wgt": The maximum value of any replicate weight

If type = "specific", the result is a data frame providing a summary of each column of replicate weights, with each column of replicate weights described in a given row of the data frame. The contents of the "specific" summary are the following:

- "Rep_Column": The name of a given column of replicate weights. If columns are unnamed, the column number is used instead
- "N": The number of entries
- "N_NONZERO": The number of nonzero entries
- "SUM": The sum of the weights
- "MEAN": The average of the weights
- "CV": The coefficient of variation of the weights (standard deviation divided by mean)
- "MIN": The minimum weight
- "MAX": The maximum weight

Examples

```
# Load example data
suppressPackageStartupMessages(library(survey))
data(api)

dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
dclus1$variables$response_status <- sample(x = c("Respondent", "Nonrespondent",
```

```
                                                "Ineligible", "Unknown eligibility"),
                                          size = nrow(dclus1),
                                          replace = TRUE)
rep_design <- as.svrepdesign(dclus1)

# Adjust weights for cases with unknown eligibility
ue_adjusted_design <- redistribute_weights(
    design = rep_design,
    reduce_if = response_status %in% c("Unknown eligibility"),
    increase_if = !response_status %in% c("Unknown eligibility"),
    by = c("stype")
)

# Summarize replicate weights

summarize_rep_weights(rep_design, type = "both")

# Summarize replicate weights by grouping variables

summarize_rep_weights(ue_adjusted_design, type = 'overall',
                      by = c("response_status"))

summarize_rep_weights(ue_adjusted_design, type = 'overall',
                      by = c("stype", "response_status"))

# Compare replicate weights

rep_wt_summaries <- lapply(list('original' = rep_design,
                                'adjusted' = ue_adjusted_design),
                           summarize_rep_weights,
                           type = "overall")
print(rep_wt_summaries)
```

`svyby_repwts` *Compare survey statistics calculated separately from different sets of replicate weights*

Description

A modified version of the `svyby()` function from the `survey` package. Whereas `svyby()` calculates statistics separately for each subset formed by a specified grouping variable, `svyby_repwts()` calculates statistics separately for each replicate design, in addition to any additional user-specified grouping variables.

Usage

```
svyby_repwts(
  rep_designs,
  formula,
```

```
  by,
  FUN,
  ...,
  deff = FALSE,
  keep.var = TRUE,
  keep.names = TRUE,
  verbose = FALSE,
  vartype = c("se", "ci", "ci", "cv", "cvpct", "var"),
  drop.empty.groups = TRUE,
  return.replicates = FALSE,
  na.rm.by = FALSE,
  na.rm.all = FALSE,
  multicore = getOption("survey.multicore")
)
```

Arguments

rep_designs	The replicate-weights survey designs to be compared. Supplied either as: • A named list of replicate-weights survey design objects, for example `list('nr' = nr_adjusted_design, 'ue' = ue_adjusted_design)`. • A 'stacked' replicate-weights survey design object created by `stack_replicate_designs()`. The designs must all have the same number of columns of replicate weights, of the same type (bootstrap, JKn, etc.)
formula	A formula specifying the variables to pass to FUN
by	A formula specifying factors that define subsets
FUN	A function taking a formula and survey design object as its first two arguments. Usually a function from the survey package, such as svytotal or svymean.
...	Other arguments to FUN
deff	A value of TRUE or FALSE, indicating whether design effects should be estimated if possible.
keep.var	A value of TRUE or FALSE. If FUN returns a svystat object, indicates whether to extract standard errors from it.
keep.names	Define row names based on the subsets
verbose	If TRUE, print a label for each subset as it is processed.
vartype	Report variability as one or more of standard error, confidence interval, coefficient of variation, percent coefficient of variation, or variance
drop.empty.groups	If FALSE, report NA for empty groups, if TRUE drop them from the output
return.replicates	If TRUE, return all the replicates as the "replicates" attribute of the result. This can be useful if you want to produce custom summaries of the estimates from each replicate.
na.rm.by	If true, omit groups defined by NA values of the by variables
na.rm.all	If true, check for groups with no non-missing observations for variables defined by formula and treat these groups as empty
multicore	Use multicore package to distribute subsets over multiple processors?

Value

An object of class "svyby": a data frame showing the grouping factors and results of FUN for each combination of the grouping factors. The first grouping factor always consists of indicators for which replicate design was used for an estimate.

Examples

```
## Not run:
suppressPackageStartupMessages(library(survey))
data(api)

dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
dclus1$variables$response_status <- sample(x = c("Respondent", "Nonrespondent",
                                                 "Ineligible", "Unknown eligibility"),
                                           size = nrow(dclus1),
                                           replace = TRUE)
orig_rep_design <- as.svrepdesign(dclus1)

# Adjust weights for cases with unknown eligibility
ue_adjusted_design <- redistribute_weights(
    design = orig_rep_design,
    reduce_if = response_status %in% c("Unknown eligibility"),
    increase_if = !response_status %in% c("Unknown eligibility"),
    by = c("stype")
)

# Adjust weights for nonresponse
nr_adjusted_design <- redistribute_weights(
    design = ue_adjusted_design,
    reduce_if = response_status %in% c("Nonrespondent"),
    increase_if = response_status == "Respondent",
    by = c("stype")
)

# Compare estimates from the three sets of replicate weights

  list_of_designs <- list('original' = orig_rep_design,
                          'unknown eligibility adjusted' = ue_adjusted_design,
                          'nonresponse adjusted' = nr_adjusted_design)

  ##_ First compare overall means for two variables
  means_by_design <- svyby_repwts(formula = ~ api00 + api99,
                                  FUN = svymean,
                                  rep_design = list_of_designs)

  print(means_by_design)

  ##_ Next compare domain means for two variables
  domain_means_by_design <- svyby_repwts(formula = ~ api00 + api99,
                                         by = ~ stype,
                                         FUN = svymean,
                                         rep_design = list_of_designs)
```

```
  print(domain_means_by_design)

# Calculate confidence interval for difference between estimates

ests_by_design <- svyby_repwts(rep_designs = list('NR-adjusted' = nr_adjusted_design,
                                                  'Original' = orig_rep_design),
                               FUN = svymean, formula = ~ api00 + api99)

differences_in_estimates <- svycontrast(stat = ests_by_design, contrasts = list(
  'Mean of api00: NR-adjusted vs. Original' = c(1,-1,0,0),
  'Mean of api99: NR-adjusted vs. Original' = c(0,0,1,-1)
))

print(differences_in_estimates)

confint(differences_in_estimates, level = 0.95)

## End(Not run)
```

Index