Package 'svrep'

May 19, 2025

Type Package

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

Version 0.8.0

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>. Implements bootstrap methods for complex surveys, including the generalized survey bootstrap as described by Beaumont and Patak (2012) <doi:10.1111/j.1751-5823.2011.00166.x>. 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 7.3.2

Imports DBI, Matrix, methods, mvtnorm, sampling, stats, survey (>= 4.1), utils

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

Config/testthat/edition 3

Depends R (>= 4.1.0)

VignetteBuilder knitr, rmarkdown

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2025-05-18 23:00:02 UTC

Contents

`add_inactive_replicates` *Add inactive replicates to a survey design object*

Description

Adds inactive replicates to a survey design object. An inactive replicate is a replicate that does not contribute to variance estimates but adds to the matrix of replicate weights so that the matrix has the desired number of columns. The new replicates' values are simply equal to the full-sample weights.

Usage

```
add_inactive_replicates(design, n_total, n_to_add, location = "last")
```

Arguments

`design`	A survey design object, created with either the `survey` or `srvyr` packages.
`n_total`	The total number of replicates that the result should contain. If the design already contains `n_total` replicates (or more), then no update is made.
`n_to_add`	The number of additional replicates to add. Can only use the `n_total` argument OR the `n_to_add` argument, not both.
`location`	Either `"first"`, `"last"` (the default), or `"random"`. Specifies where the columns of new replicates should be located in the matrix of replicate weights. Use `"first"` to place new replicates first (i.e., in the leftmost part of the matrix), `"last"` to place the new replicates last (i.e., in the rightmost part of the matrix). Use `"random"` to intersperse the new replicates in random column locations of the matrix; the original replicates will still be in their original order.

Value

An updated survey design object, where the number of columns of replicate weights has potentially increased. The increase only happens if the user specifies the `n_to_add` argument instead of `n_total`, of if the user specifies `n_total` and `n_total` is less than the number of columns of replicate weights that the design already had.

Statistical Details

Inactive replicates are also sometimes referred to as "dead replicates", for example in Ash (2014). The purpose of adding inactive replicates is to increase the number of columns of replicate weights without impacting variance estimates. This can be useful, for example, when combining data from a survey across multiple years, where different years use different number of replicates, but a consistent number of replicates is desired in the combined data file.

Suppose the initial replicate design has L replicates, with respective constants c_k for $k = 1, \ldots, L$ used to estimate variance with the formula

$$v_R = \sum_{k=1}^{L} c_k \left(\hat{T}_y^{(k)} - \hat{T}_y \right)^2$$

where $\hat{T}_y$ is the estimate produced using the full-sample weights and $\hat{T}_y^{(k)}$ is the estimate from replicate k.

Inactive replicates are simply replicates that are exactly equal to the full sample: that is, the replicate k is called "inactive" if its vector of replicate weights exactly equals the full-sample weights. In this case, when using the formula above to estimate variances, these replicates contribute nothing to the variance estimate.

If the analyst uses the variant of the formula above where the full-sample estimate $\hat{T}_y$ is replaced by the average replicate estimate (i.e., $L^{-1}\sum_{k=1}^{L}\hat{T}_y^{(k)}$), then variance estimates will differ before vs. after adding the inactive replicates. For this reason, it is strongly recommend to explicitly specify `mse=TRUE` when creating a replicate design object in R with functions such as `svrepdesign()`, `as_bootstrap_design()`, etc. If working with an already existing replicate design, you can update the `mse` option to `TRUE` simply by using code such as `my_design$mse <- TRUE`.

References

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

Examples

```
library(survey)
set.seed(2023)

# Create an example survey design object

  sample_data <- data.frame(
    PSU     = c(1,2,3)
  )

  survey_design <- svydesign(
    data = sample_data,
    ids = ~ PSU,
    weights = ~ 1
  )

  rep_design <- survey_design |>
    as.svrepdesign(type = "JK1", mse = TRUE)

# Inspect replicates before subsampling

  rep_design |> weights(type = "analysis")

# Inspect replicates after adding inactive replicates

  rep_design |>
    add_inactive_replicates(n_total = 5, location = "first") |>
    weights(type = "analysis")

  rep_design |>
    add_inactive_replicates(n_to_add = 2, location = "last") |>
    weights(type = "analysis")
```

```
rep_design |>
  add_inactive_replicates(n_to_add = 5, location = "random") |>
  weights(type = "analysis")
```

---

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.
- **"Antal-Tille"**:
  The doubled half bootstrap method proposed by Antal and Tillé (2014), which is only applicable to single-stage designs or multistage designs where the first-stage sampling fractions are small (and can thus be ignored). Accomodates stratified designs. Sampling within each stratum can be simple random sampling or unequal probability sampling with or 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.

References

Antal, E. and Tillé, Y. (2014). "A new resampling method for sampling designs without replacement: The doubled half bootstrap." **Computational Statistics**, *29*(5), 1345-1363. https://doi.org/10.1007/s00180-014-0495-0

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

Canty, A.J.; Davison, A.C. (1999). "Resampling-based variance estimation for labour force surveys." **The Statistician**, *48*: 379-391.

Preston, J. (2009). "Rescaled bootstrap for stratified multistage sampling." **Survey Methodology**, *35*(2): 227-234.

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

See Also

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

The underlying function for the Rao-Wu-Yue-Beaumont bootstrap is `make_rwyb_bootstrap_weights`, and the underlying function for the Antal-Tillé bootstrap is `make_doubled_half_bootstrap_weights`. Other bootstrap methods are implemented using functions from the 'survey' package, including: `bootweights` (Canty-Davison), `subbootweights` (Rao-Wu), and `mrbweights` (Preston).

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_",
  vars_to_keep = NULL
)
```

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.
vars_to_keep	By default, all variables in the data will be kept. To select only a subset of the non-weight variables, you can supply a character vector of variable names to keep.

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_fays_gen_rep_design` *Convert a survey design object to a replication design using Fay's generalized replication method*

Description

Converts a survey design object to a replicate design object with replicate weights formed using the generalized replication method of Fay (1989). The generalized replication method forms replicate weights from a textbook variance estimator, provided that the variance estimator can be represented as a quadratic form whose matrix is positive semidefinite (this covers a large class of variance estimators).

Usage

```
as_fays_gen_rep_design(
  design,
  variance_estimator = NULL,
  aux_var_names = NULL,
  max_replicates = Inf,
  balanced = TRUE,
  psd_option = "warn",
  mse = TRUE,
  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 variance-estimators for a detailed description of each variance estimator. 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.
- **"Poisson Horvitz-Thompson"**:
  The Horvitz-Thompson variance estimator based on assuming Poisson sampling, with first-order inclusion probabilities inferred from the sampling probabilities of the survey design object.
- **"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.
- **"Deville-1"**:
  A variance estimator for unequal-probability sampling without replacement, described in Matei and Tillé (2005) as "Deville 1".
- **"Deville-2"**:
  A variance estimator for unequal-probability sampling without replacement, described in Matei and Tillé (2005) as "Deville 2".
- **"Deville-Tille":**
  A variance estimator useful for balanced sampling designs, proposed by Deville and Tillé (2005).
- **"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.
- **"BOSB"**:
  The kernel-based variance estimator proposed by Breidt, Opsomer, and Sanchez-Borrego (2016) for use with systematic samples or other finely stratified designs. Uses the Epanechnikov kernel with the bandwidth automatically chosen to result in the smallest possible nonempty kernel window.
- **"Beaumont-Emond"**:
  The variance estimator of Beaumont and Emond (2022) for multistage unequal-probability sampling without replacement.

`aux_var_names` (Only used if `variance_estimator = "Deville-Tille"`). A vector of the names of auxiliary variables used in sampling.

`max_replicates` The maximum number of replicates to allow (should be as large as possible, given computer memory/storage limitations). A commonly-recommended default is 500. If the number of replicates needed for a balanced, fully-efficient estimator is less than `max_replicates`, then only the number of replicates needed will be created. If more replicates are needed than `max_replicates`, then the

full number of replicates needed will be created, but only a random subsample will be retained.

`balanced` If `balanced=TRUE`, the replicates will all contribute equally to variance estimates, but the number of replicates needed may slightly increase.

`psd_option` Either `"warn"` (the default) or `"error"`. This option specifies what will happen if the target variance estimator has a quadratic form matrix which is not positive semidefinite. This can occasionally happen, particularly for two-phase designs.
If `psd_option="error"`, then an error message will be displayed.
If `psd_option="warn"`, then a warning message will be displayed, and the quadratic form matrix will be approximated by the most similar positive semidefinite matrix. This approximation was suggested by Beaumont and Patak (2012), who note that this is conservative in the sense of producing overestimates of variance. Beaumont and Patak (2012) argue that this overestimation is expected to be small in magnitude. See `get_nearest_psd_matrix` for details of the approximation.

`mse` If `TRUE` (the default), 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. For Fay's generalized replication method, setting `mse = FALSE` can potentially lead to large underestimates of variance.

`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

See Fay (1989) for a full description of this replication method, or see the documentation in make_fays_gen_rep_factors for implementation details.

See variance-estimators for a description of each variance estimator available for use with this function.

Use `rescale_reps` to eliminate negative adjustment factors.

Two-Phase Designs

For a two-phase design, `variance_estimator` should be a list of variance estimators' names, with two elements, such as `list('Ultimate Cluster', 'Poisson Horvitz-Thompson')`. In two-phase designs, only the following estimators may be used for the second phase:

- "Ultimate Cluster"
- "Stratified Multistage SRS"

- "Poisson Horvitz-Thompson"

For statistical details on the handling of two-phase designs, see the documentation for make_twophase_quad_form.

References

The generalized replication method was first proposed in Fay (1984). Fay (1989) refined the generalized replication method to produce "balanced" replicates, in the sense that each replicate contributes equally to variance estimates. The advantage of balanced replicates is that one can still obtain a reasonable variance estimate by using only a random subset of the replicates.

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

- 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

- Breidt, F. J., Opsomer, J. D., & Sanchez-Borrego, I. (2016). "*Nonparametric Variance Estimation Under Fine Stratification: An Alternative to Collapsed Strata*." **Journal of the American Statistical Association**, 111(514), 822-833. https://doi.org/10.1080/01621459.2015.1058264

- Deville, J. C., and Tillé, Y. (2005). "*Variance approximation under balanced sampling.*" **Journal of Statistical Planning and Inference**, 128, 569-591.

- 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.

- Fay, Robert. 1989. "Theory And Application Of Replicate Weighting For Variance Calculations." In, 495-500. Alexandria, VA: American Statistical Association. http://www.asasrms.org/Proceedings/papers/1989_033.pdf

- Matei, Alina, and Yves Tillé. (2005). "*Evaluation of Variance Approximations and Estimators in Maximum Entropy Sampling with Unequal Probability and Fixed Sample Size.*" **Journal of Official Statistics**, 21(4):543-70.

See Also

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_fays_gen_rep_factors`.

Examples

```
if (FALSE) {

  library(survey)
```

```
  ## Load an example systematic sample ----
  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 |>
    sort_by(~ 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 replicate design

  gen_rep_design_sd2 <- as_fays_gen_rep_design(
    design = design_obj,
    variance_estimator = "SD2",
    max_replicates = 250,
    mse = TRUE
  )

  svytotal(x = ~ TOTSTAFF, na.rm = TRUE, design = gen_rep_design_sd2)
}
```

`as_gen_boot_design` *Convert a survey design object to a generalized bootstrap 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 semidefinite (this covers a large class of variance estimators).

Usage

```
as_gen_boot_design(
  design,
  variance_estimator = NULL,
  aux_var_names = NULL,
  replicates = 500,
  tau = 1,
  exact_vcov = FALSE,
```

```
  psd_option = "warn",
  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 variance-estimators for a detailed description of each variance estimator. 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.
- **"Poisson Horvitz-Thompson"**:
  The Horvitz-Thompson variance estimator based on assuming Poisson sampling, with first-order inclusion probabilities inferred from the sampling probabilities of the survey design object.
- **"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.
- **"Deville-1"**:
  A variance estimator for unequal-probability sampling without replacement, described in Matei and Tillé (2005) as "Deville 1".
- **"Deville-2"**:
  A variance estimator for unequal-probability sampling without replacement, described in Matei and Tillé (2005) as "Deville 2".
- **"Deville-Tille":**
  A variance estimator useful for balanced sampling designs, proposed by Deville and Tillé (2005).
- **"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.

- **"BOSB"**:
  The kernel-based variance estimator proposed by Breidt, Opsomer, and Sanchez-Borrego (2016) for use with systematic samples or other finely stratified designs. Uses the Epanechnikov kernel with the bandwidth automatically chosen to result in the smallest possible nonempty kernel window.
- **"Beaumont-Emond"**:
  The variance estimator of Beaumont and Emond (2022) for multistage unequal-probability sampling without replacement.

`aux_var_names` (Only used if `variance_estimator = "Deville-Tille"`). A vector of the names of auxiliary variables used in 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; the default value is 1. 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`. Instead of using `tau="auto"`, the user can instead use the function `rescale_reps()` to rescale the replicates later.

`exact_vcov` If `exact_vcov=TRUE`, the replicate factors will be generated such that variance estimates for totals exactly match the results from the target variance estimator. This requires that `num_replicates` exceeds the rank of `Sigma`. The replicate factors are generated by applying PCA-whitening to a collection of draws from a multivariate Normal distribution, then applying a coloring transformation to the whitened collection of draws.

`psd_option` Either `"warn"` (the default) or `"error"`. This option specifies what will happen if the target variance estimator has a quadratic form matrix which is not positive semidefinite. This can occasionally happen, particularly for two-phase designs.
If `psd_option="error"`, then an error message will be displayed.
If `psd_option="warn"`, then a warning message will be displayed, and the quadratic form matrix will be approximated by the most similar positive semidefinite matrix. This approximation was suggested by Beaumont and Patak (2012), who note that this is conservative in the sense of producing overestimates of variance. Beaumont and Patak (2012) argue that this overestimation is expected to be small in magnitude. See `get_nearest_psd_matrix` for details of the approximation.

`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 semidefinite.

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. This rescaling method does not affect variance estimates for linear statistics, but its impact on non-smooth statistics such as quantiles is unclear.

Two-Phase Designs

For a two-phase design, `variance_estimator` should be a list of variance estimators' names, with two elements, such as `list('Ultimate Cluster', 'Poisson Horvitz-Thompson')`. In two-phase designs, only the following estimators may be used for the second phase:

- "Ultimate Cluster"
- "Stratified Multistage SRS"
- "Poisson Horvitz-Thompson"

For statistical details on the handling of two-phase designs, see the documentation for make_twophase_quad_form.

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.

- Breidt, F. J., Opsomer, J. D., & Sanchez-Borrego, I. (2016). "*Nonparametric Variance Estimation Under Fine Stratification: An Alternative to Collapsed Strata*." **Journal of the American Statistical Association**, 111(514), 822-833. https://doi.org/10.1080/01621459.2015.1058264

- 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.

- Deville, J. C., and Tillé, Y. (2005). "*Variance approximation under balanced sampling.*" **Journal of Statistical Planning and Inference**, 128, 569-591.

- 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.

- Matei, Alina, and Yves Tillé. (2005). "*Evaluation of Variance Approximations and Estimators in Maximum Entropy Sampling with Unequal Probability and Fixed Sample Size.*" **Journal of Official Statistics**, 21(4):543-70.

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`. The function `rescale_reps` is used to implement the rescaling of the bootstrap adjustment factors.

See variance-estimators for a description of each variance estimator.

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 |>
    sort_by(~ 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)

# Example 3: Two-phase sample ----
# -- First stage is stratified systematic sampling,
# -- second stage is response/nonresponse modeled as Poisson sampling

  nonresponse_model <- glm(
    data = library_stsys_sample,
    family = quasibinomial('logit'),
    formula = I(RESPONSE_STATUS == "Survey Respondent") ~ 1,
    weights = 1/library_stsys_sample$SAMPLING_PROB
  )

  library_stsys_sample[['RESPONSE_PROPENSITY']] <- predict(
    nonresponse_model,
    newdata = library_stsys_sample,
    type = "response"
  )

  twophase_design <- twophase(
    data = library_stsys_sample,
    # Identify cases included in second phase sample
```

```
    subset = ~ I(RESPONSE_STATUS == "Survey Respondent"),
    strata = list(~ SAMPLING_STRATUM, NULL),
    id = list(~ 1, ~ 1),
    probs = list(NULL, ~ RESPONSE_PROPENSITY),
    fpc = list(~ STRATUM_POP_SIZE, NULL),
  )

  twophase_boot_design <- as_gen_boot_design(
    design = twophase_design,
    variance_estimator = list(
      "SD2", "Poisson Horvitz-Thompson"
    )
  )

  svytotal(x = ~ LIBRARIA, design = twophase_boot_design)


## End(Not run)
```

`as_random_group_jackknife_design` *Convert a survey design object to a random-groups jackknife design*

Description

Forms a specified number of jackknife replicates based on grouping primary sampling units (PSUs) into random, (approximately) equal-sized groups.

Usage

```
as_random_group_jackknife_design(
  design,
  replicates = 50,
  var_strat = NULL,
  var_strat_frac = NULL,
  sort_var = NULL,
  adj_method = "variance-stratum-psus",
  scale_method = "variance-stratum-psus",
  group_var_name = ".random_group",
  compress = TRUE,
  mse = getOption("survey.replicates.mse")
)
```

Arguments

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

`replicates` The number of replicates to create for each variance stratum. The total number of replicates created is the number of variance strata times `replicates`. Every design stratum must have at least as many primary sampling units (PSUs), as `replicates`.

`var_strat` Specifies the name of a variable in the data that defines variance strata to use for the grouped jackknife. If `var_strat = NULL`, then there is effectively only one variance stratum.

`var_strat_frac` Specifies the sampling fraction to use for finite population corrections in each value of `var_strat`. Can use either a single number or a variable in the data corresponding to `var_strat`.

`sort_var` (Optional) Specifies the name of a variable in the data which should be used to sort the data before assigning random groups. If a variable is specified for `var_strat`, the sorting will happen within values of that variable.

`adj_method` Specifies how to calculate the replicate weight adjustment factor. Available options for `adj_method` include:

- `"variance-stratum-psus"` (the default)
  The replicate weight adjustment for a unit is based on the number of PSUs in its variance stratum.
- `"variance-units"`
  The replicate weight adjustment for a unit is based on the number of variance units in its variance stratum.

See the section "Adjustment and Scale Methods" for details.

`scale_method` Specifies how to calculate the scale factor for each replicate. Available options for `scale_method` include:

- `"variance-stratum-psus"`
  The scale factor for a variance unit is based on its number of PSUs compared to the number of PSUs in its variance stratum.
- `"variance-units"`
  The scale factor for a variance unit is based on the number of variance units in its variance stratum.

See the section "Adjustment and Scale Methods" for details.

`group_var_name` (Optional) The name of a new variable created to save identifiers for which random group each PSU was grouped into for the purpose of forming replicates. Specify `group_var_name = NULL` to avoid creating the variable in the data.

`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.

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.

Formation of Random Groups

Within each value of `VAR_STRAT`, the data are sorted by first-stage sampling strata, and then the PSUs in each stratum are randomly arranged. Groups are then formed by serially placing PSUs into each group. The first PSU in the `VAR_STRAT` is placed into the first group, the second PSU into the second group, and so on. Once a PSU has been assigned to the last group, the process begins again by assigning the next PSU to the first group, the PSU after that to the second group, and so on.

The random group that each observation is assigned to can be saved as a variable in the data by using the function argument `group_var_name`.

Adjustment and Scale Methods

The jackknife replication variance estimator based on R replicates takes the following form:

$$v(\hat{\theta}) = \sum_{r=1}^{R}(1 - f_r) \times c_r \times \left(\hat{\theta}_r - \hat{\theta}\right)^2$$

where r indexes one of the R sets of replicate weights, c_r is a corresponding scale factor for the r-th replicate, and $1 - f_r$ is an optional finite population correction factor that can potentially differ across variance strata.

To form the replicate weights, the PSUs are divided into $\tilde{H}$ variance strata, and the $\tilde{h}$-th variance stratum contains $G_{\tilde{h}}$ random groups. The number of replicates R equals the total number of random groups across all variance strata: $R = \sum_{\tilde{h}}^{\tilde{H}} G_{\tilde{h}}$. In other words, each replicate corresponds to one of the random groups from one of the variance strata.

The weights for replicate r corresponding to random group g within variance stratum $\tilde{h}$ is defined as follows.

If case i is not in variance stratum $\tilde{h}$, then $w_i^{(r)} = w_i$.

If case i is in variance stratum $\tilde{h}$ and not in random group g, then $w_i^{(r)} = a_{\tilde{h}g} w_i$.

Otherwise, if case i is in random group g of variance stratum $\tilde{h}$, then $w_i^{(r)} = 0$.

The R function argument `adj_method` determines how the adjustment factor $a_{\tilde{h}g}$ is calculated. When `adj_method = "variance-units"`, then $a_{\tilde{h}g}$ is calculated based on $G_{\tilde{h}}$, which is the number of random groups in variance stratum $\tilde{h}$. When `adj_method = "variance-stratum-psus"`, then $a_{\tilde{h}g}$ is calculated based on $n_{\tilde{h}g}$, which is the number of PSUs in random group g in variance stratum $\tilde{h}$, as well as $n_{\tilde{h}}$, the total number of PSUs in variance stratum $\tilde{h}$.

If `adj_method = "variance-units"`, then:

$$a_{\tilde{h}g} = \frac{G_{\tilde{h}}}{G_{\tilde{h}} - 1}$$

If `adj_method = "variance-stratum-psus"`, then:

$$a_{\tilde{h}g} = \frac{n_{\tilde{h}}}{n_{\tilde{h}} - n_{\tilde{h}g}}$$

The scale factor c_r for replicate r corresponding to random group g within variance stratum $\tilde{h}$ is calculated according to the function argument `scale_method`.

If `scale_method = "variance-units"`, then:

$$c_r = \frac{G_{\tilde{h}} - 1}{G_{\tilde{h}}}$$

If `scale_method = "variance-stratum-psus"`, then:

$$c_r = \frac{n_{\tilde{h}} - n_{\tilde{h}g}}{n_{\tilde{h}}}$$

The sampling fraction f_r used for finite population correction $1 - f_r$ is by default assumed to equal 0. However, the user can supply a sampling fraction for each variance stratum using the argument `var_strat_frac`.

When variance units in a variance stratum have differing numbers of PSUs, the combination `adj_method = "variance-stratum-psus"` and `scale_method = "variance-units"` is recommended by Valliant, Brick, and Dever (2008), corresponding to their method `"GJ2"`.

The random-groups jackknife method often referred to as "DAGJK" corresponds to the options `var_strat = NULL`, `adj_method = "variance-units"`, and `scale_method = "variance-units"`. The DAGJK method will yield upwardly-biased variance estimates for totals if the total number of PSUs is not a multiple of the total number of replicates (Valliant, Brick, and Dever 2008).

References

See Section 15.5 of Valliant, Dever, and Kreuter (2018) for an introduction to the grouped jackknife and guidelines for creating the random groups.

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

See Valliant, Brick, and Dever (2008) for statistical details related to the `adj_method` and `scale_method` arguments.

- Valliant, Richard, Michael Brick, and Jill Dever. 2008. "Weight Adjustments for the Grouped Jackknife Variance Estimator." *Journal of Official Statistics*. 24: 469-88.

See Chapter 4 of Wolter (2007) for additional details of the jackknife, including the method based on random groups.

- Wolter, Kirk. 2007. "Introduction to Variance Estimation." New York, NY: Springer New York. https://doi.org/10.1007/978-0-387-35099-8.

Examples

```
library(survey)

# Load example data

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

 api_strat_design <- svydesign(
   data = apistrat,
```

```
  id = ~ 1,
  strata = ~stype,
  weights = ~pw
)

# Create a random-groups jackknife design

jk_design <- as_random_group_jackknife_design(
  api_strat_design,
  replicates = 15
)
print(jk_design)
```

`as_sdr_design` *Convert a survey design object to a successive differences replicate design*

Description

Converts a survey design object to a replicate design object with replicate weights formed using the successive differences replication (SDR) method. The SDR method is suitable for designs that use systematic sampling or finely-stratified sampling designs.

Usage

```
as_sdr_design(
  design,
  replicates,
  sort_variable = NULL,
  use_normal_hadamard = FALSE,
  compress = TRUE,
  mse = TRUE
)
```

Arguments

`design`	A survey design object created using the 'survey' (or 'srvyr') package, with class `'survey.design'` or `'svyimputationList'`.
`replicates`	The target number of replicates to create. This will determine the order of the Hadamard matrix to use when creating replicate factors. If `use_normal_hadamard = TRUE`, then the actual number of replicates will be greater than or equal to `replicates` and determined by identifying the smallest available Hadamard matrix available from the 'survey' package. If `use_normal_hadamard = FALSE`, then the actual number of replicates will be the smallest *power* of 4 that is greater or equal to the specified value of `replicates`.
`sort_variable`	A character string specifying the name of a sorting variable. This variable should give the sort order used in sampling. If the design includes strata, then the replicate factors will be assigned after first sorting by the first-stage strata identifier and then sorting by the value of `sort_variable` within each stratum.

`use_normal_hadamard`
: Whether to use a normal Hadamard matrix: that is, a matrix whose first row and first column only have entries equal to 1. This means that one of the replicates will be an "inactive" replicate. See the "Details" section for more information.

`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.

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 Overview

The successive difference replication method was proposed by Fay and Train (1995) as a replication method appropriate for samples selected using systematic sampling. It is designed to yield variance estimates for totals that are equivalent to successive difference variance estimators described in Fay and Train (1995). There are different methods for forming the replicate factors depending on whether the replicate variance estimator is meant to be equivalent to the SD2 variance estimator (i.e., the circular successive difference estimator) or the SD1 variance estimator (the non-circular successive difference estimator) described in Ash (2014). This function uses the approach based on the SD2 variance estimator. For multistage designs, this replication method only takes into account information about the first stage of sampling.

The scale factor to be used for variance estimation with the replicate weights is $4/R$, where R is the number of replicates. This scale factor will be used even when there are finite population corrections; see the subsection below.

As an alternative to the successive difference replication estimator, one can use a generalized replication method where the target variance estimator is the "SD1" or "SD2" estimator. See the functions as_gen_boot_design or as_fays_gen_rep_design for more details on generalized replication and see the help section variance-estimators for more details on the "SD1" and "SD2" variance estimators.

Details on Stratification and Finite Population Corrections

If the design includes strata, then the replicate factors will be assigned after first sorting by the first-stage strata identifier and then sorting by the value of `sort_variable` within each stratum.

If there are finite population correction factors, then these finite population correction factors will be applied to the replicate factors. This means that variance estimates with the finite population correction do not require any adjustment to the overall scale factor used in variance estimation. This is the approach used by the U.S. Census Bureau for the 5-year American Community Survey (ACS) replicate weights (U.S. Census Bureau, 2022, p. 12-8). This approach is used regardless of whether the

design has one overall finite population correction factor or has different finite population correction factors for different strata.

Details on Row Assignments for Creating Replicate Factors

The number of replicates must match the order of an available Hadamard matrix. A Hadamard matrix can either be normal or non-normal: a normal Hadamard matrix is one where the entries in the first row and in the first column are all equal to one. If the user specifies `use_normal_hadamard = TRUE`, then there are more choices of Hadamard matrix sizes available, and so greater flexibility in choosing the number of replicates to create. When a normal Hadamard matrix is used, this will result in the creation of an inactive replicate (sometimes referred to as a "dead" replicate), which is a replicate where all the replicate factors equal one. Inactive replicates are perfectly valid for variance estimation, though some users may find them confusing.

An important part of the process of creating replicate weights is the assignment of rows of the Hadamard matrix to primary sampling units. The method of Ash (2014) referred to as "RA1" is used for row assignments, which means that the replication-based variance estimates for totals will be equivalent to the SD2 variance estimator described by Ash (2014). The number of cycles used with the "RA1" method is the smallest integer greater than n/R, where n is the number of primary sample units and R is the number of replicates.

References

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

Fay, R.E. and Train, G.F. (1995). "*Aspects of Survey and Model-Based Postcensal Estimation of Income and Poverty Characteristics for States and Counties*." Joint Statistical Meetings, Proceedings of the Section on Government Statistics, 154-159.

U.S. Census Bureau. (2022). "*American Community Survey and Puerto Rico Community Survey Design and Methodology, Version 3.0.*"

Examples

```
library(survey)

# Load example stratified systematic sample
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 SDR replicate design
sdr_design <- as_sdr_design(
  design              = design_obj,
  replicates          = 180,
  sort_variable       = "SAMPLING_SORT_ORDER",
  use_normal_hadamard = TRUE
)

## Compare to generalized bootstrap
## based on the SD2 estimator that SDR approximates
gen_boot_design <- as_gen_boot_design(
  design             = design_obj,
  variance_estimator = "SD2",
  replicates         = 180,
  exact_vcov         = TRUE
)

## Estimate sampling variances
svytotal(x = ~ TOTSTAFF, na.rm = TRUE, design = sdr_design)
svytotal(x = ~ TOTSTAFF, na.rm = TRUE, design = gen_boot_design)
```

`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.

Syntax for Common Types of Calibration

For ratio estimation with an auxiliary variable `X`, use the following options:
- `cal_formula = ~ -1 + X`
- `variance = 1`,
- `cal.fun = survey::cal.linear`

For post-stratification, use the following option:

- `cal.fun = survey::cal.linear`

For raking, use the following option:

- `cal.fun = survey::cal.raking`

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 specify which control survey replicate is matched to each primary survey replicate. If the $i-th$ entry of control_col_matches equals k, then replicate i in primary_rep_design is matched to replicate k in control_rep_design. 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.

Syntax for Common Types of Calibration

For ratio estimation with an auxiliary variable `X`, use the following options:
- `cal_formula = ~ -1 + X`
- `variance = 1`,
- `cal.fun = survey::cal.linear`

For post-stratification, use the following option:

- `cal.fun = survey::cal.linear`

For raking, use the following option:

- `cal.fun = survey::cal.raking`

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)
```

`get_design_quad_form` *Determine the quadratic form matrix of a variance estimator for a survey design object*

Description

Determines the quadratic form matrix of a specified variance estimator, by parsing the information stored in a survey design object created using the 'survey' package.

Usage

```
get_design_quad_form(
  design,
```

```
  variance_estimator,
  ensure_psd = FALSE,
  aux_var_names = NULL
)
```

Arguments

`design` A survey design object created using the 'survey' (or 'srvyr') package, with class `'survey.design'` or `'svyimputationList'`. Also accepts two-phase design objects with class `'twophase2'`; see the section below titled "Two-Phase Designs" for more information about handling of two-phase designs.

`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.
- **"Poisson Horvitz-Thompson"**:
  The Horvitz-Thompson variance estimator based on assuming Poisson sampling, with first-order inclusion probabilities inferred from the sampling probabilities of the survey design object.
- **"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.
- **"Deville-1"**:
  A variance estimator for unequal-probability sampling without replacement, described in Matei and Tillé (2005) as "Deville 1".
- **"Deville-2"**:
  A variance estimator for unequal-probability sampling without replacement, described in Matei and Tillé (2005) as "Deville 2".
- **"Deville-Tille":**
  A variance estimator useful for balanced sampling designs, proposed by Deville and Tillé (2005).
- **"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.

- **"Beaumont-Emond"**:
  The variance estimator of Beaumont and Emond (2022) for multistage unequal-probability sampling without replacement.
- **"BOSB"**:
  The kernel-based variance estimator proposed by Breidt, Opsomer, and Sanchez-Borrego (2016) for use with systematic samples or other finely stratified designs. Uses the Epanechnikov kernel with the bandwidth automatically chosen to result in the smallest possible nonempty kernel window.

`ensure_psd` If `TRUE` (the default), ensures that the result is a positive semidefinite matrix. This is necessary if the quadratic form is used as an input for replication methods such as the generalized bootstrap. For mathematical details, please see the documentation for the function `get_nearest_psd_matrix()`. The approximation method is discussed by Beaumont and Patak (2012) in the context of forming replicate weights for two-phase samples. The authors argue that this approximation should lead to only a small overestimation of variance.

`aux_var_names` Only required if `variance_estimator = "Deville-Tille"` or if `variance_estimator = "BOSB"`. For the Deville-Tillé estimator, this should be a character vector of variable names for auxiliary variables to be used in the variance estimator. For the BOSB estimator, this should be a string giving a single variable name to use as an auxiliary variable in the kernel-based variance estimator of Breidt, Opsomer, and Sanchez-Borrego (2016).

Value

A matrix representing the quadratic form of a specified variance estimator, based on extracting information about clustering, stratification, and selection probabilities from the survey design object.

Variance Estimators

See variance-estimators for a description of each variance estimator.

Two-Phase Designs

For a two-phase design, `variance_estimator` should be a list of variance estimators' names, with two elements, such as `list('Ultimate Cluster', 'Poisson Horvitz-Thompson')`. In two-phase designs, only the following estimators may be used for the second phase:

- "Ultimate Cluster"
- "Stratified Multistage SRS"
- "Poisson Horvitz-Thompson"

For statistical details on the handling of two-phase designs, see the documentation for make_twophase_quad_form.

References

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

- 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.

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

- Breidt, F. J., Opsomer, J. D., & Sanchez-Borrego, I. (2016). "*Nonparametric Variance Estimation Under Fine Stratification: An Alternative to Collapsed Strata*." **Journal of the American Statistical Association**, 111(514), 822-833. https://doi.org/10.1080/01621459.2015.1058264

- Deville, J.C., and Tillé, Y. (2005). "*Variance approximation under balanced sampling.*" **Journal of Statistical Planning and Inference**, 128, 569-591.

- Särndal, C.E., Swensson, B., & Wretman, J. (1992). "*Model Assisted Survey Sampling*." Springer New York.

Examples

```
## Not run:
# Example 1: Quadratic form for successive-difference variance 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
   )

   ## Obtain quadratic form
   quad_form_matrix <- get_design_quad_form(
     design = design_obj,
     variance_estimator = "SD2"
   )

   ## Estimate variance of estimated population total
   y <- design_obj$variables$LIBRARIA
   wts <- weights(design_obj, type = 'sampling')
```

```
    y_wtd <- as.matrix(y) * wts
    y_wtd[is.na(y_wtd)] <- 0

    pop_total <- sum(y_wtd)

    var_est <- t(y_wtd) %*% quad_form_matrix %*% y_wtd
    std_error <- sqrt(var_est)

    print(pop_total); print(std_error)

    # Compare to estimate from assuming SRS
    svytotal(x = ~ LIBRARIA, na.rm = TRUE,
             design = design_obj)

# Example 2: Kernel-based variance estimator ----

    Q_BOSB <- get_design_quad_form(
      design             = design_obj,
      variance_estimator = "BOSB",
      aux_var_names      = "SAMPLING_SORT_ORDER"
    )

    var_est <- t(y_wtd) %*% Q_BOSB %*% y_wtd
    std_error <- sqrt(var_est)

    print(pop_total); print(std_error)

# Example 3: Two-phase design (second phase is nonresponse) ----

  ## Estimate response propensities, separately by stratum
  library_stsys_sample[['RESPONSE_PROB']] <- svyglm(
    design = design_obj,
    formula = I(RESPONSE_STATUS == "Survey Respondent") ~ SAMPLING_STRATUM,
    family = quasibinomial('logistic')
  ) |> predict(type = 'response')

  ## Create a survey design object,
  ## where nonresponse is treated as a second phase of sampling
  twophase_design <- twophase(
    data = library_stsys_sample,
    strata = list(~ SAMPLING_STRATUM, NULL),
    id = list(~ 1, ~ 1),
    fpc = list(~ STRATUM_POP_SIZE, NULL),
    probs = list(NULL, ~ RESPONSE_PROB),
    subset = ~ I(RESPONSE_STATUS == "Survey Respondent")
  )

  ## Obtain quadratic form for the two-phase variance estimator,
  ## where first phase variance contribution estimated
  ## using the successive differences estimator
  ## and second phase variance contribution estimated
  ## using the Horvitz-Thompson estimator
  ## (with joint probabilities based on assumption of Poisson sampling)
```

```
  get_design_quad_form(
    design = twophase_design,
    variance_estimator = list(
      "SD2",
      "Poisson Horvitz-Thompson"
    )
  )

## End(Not run)
```

`get_nearest_psd_matrix` *Approximates a symmetric, real matrix by the nearest positive semidefinite matrix.*

Description

Approximates a symmetric, real matrix by the nearest positive semidefinite matrix in the Frobenius norm, using the method of Higham (1988). For a real, symmetric matrix, this is equivalent to "zeroing out" negative eigenvalues. See the "Details" section for more information.

Usage

```
get_nearest_psd_matrix(X)
```

Arguments

`X`	A symmetric, real matrix with no missing values.

Details

Let A denote a symmetric, real matrix which is not positive semidefinite. Then we can form the spectral decomposition $A = \Gamma\Lambda\Gamma'$, where Λ is the diagonal matrix whose entries are eigenvalues of A. The method of Higham (1988) is to approximate A with $\tilde{A} = \Gamma\Lambda_+\Gamma'$, where the ii-th entry of Λ_+ is $\max(\Lambda_{ii}, 0)$.

Value

The nearest positive semidefinite matrix of the same dimension as `X`.

References

- Higham, N. J. (1988). "*Computing a nearest symmetric positive semidefinite matrix.*" Linear Algebra and Its Applications, 103, 103-118.

Examples

```
X <- matrix(
  c(2, 5, 5,
    5, 2, 5,
    5, 5, 2),
  nrow = 3, byrow = TRUE
)
get_nearest_psd_matrix(X)
```

`is_psd_matrix` *Check whether a matrix is positive semidefinite*

Description

Check whether a matrix is positive semidefinite, based on checking for symmetric and negative eigenvalues.

Usage

```
is_psd_matrix(X, tolerance = sqrt(.Machine$double.eps))
```

Arguments

`X`	A matrix with no missing or infinite values.
`tolerance`	Tolerance for controlling whether a tiny computed eigenvalue will actually be considered negative. Computed negative eigenvalues will be considered negative if they are less than which are less than `-abs(tolerance * max(eigen(X)$values))`. A small nonzero tolerance is recommended since eigenvalues are nearly always computed with some floating-point error.

Value

A logical value. `TRUE` if the matrix is deemed positive semidefinite. Negative otherwise (including if `X` is not symmetric).

See Also

The function `get_nearest_psd_matrix()` can be used to approximate a symmetric matrix which is not positive semidefinite, by a similar positive semidefinite matrix.

Examples

```
X <- matrix(
  c(2, 5, 5,
    5, 2, 5,
    5, 5, 2),
  nrow = 3, byrow = TRUE
)
```

```
is_psd_matrix(X)

eigen(X)$values
```

`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_doubled_half_bootstrap_weights

Create bootstrap replicate weights using the "doubled half bootstrap" method of Antal and Tillé (2014).

Description

Creates bootstrap replicate weights using the method of Antal and Tillé (2014). This method is applicable to single-stage sample designs, potentially with stratification and clustering. It can be used for designs that use simple random sampling without replacement or unequal probability sampling without replacement. One advantage of this method is that it yields integer replicate factors of 0, 1, 2, or 3.

Usage

```
make_doubled_half_bootstrap_weights(
  num_replicates = 100,
  samp_unit_ids,
  strata_ids,
  samp_unit_sel_probs,
  output = "weights"
)
```

Arguments

num_replicates	Positive integer giving the number of bootstrap replicates to create.
samp_unit_ids	Vector of sampling unit IDs.
strata_ids	Vector of strata IDs for each sampling unit at each stage of sampling.
samp_unit_sel_probs	Vector of selection probabilities for each sampling unit.
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

For stratified sampling, the replicate factors are generated independently in each stratum. For cluster sampling at a given stage, the replicate factors are generated at the cluster level and then the cluster's replicate factors are applied to all units in the cluster.

In the case of unequal probability sampling, this bootstrap method is only recommended for high entropy sampling methods (i.e., most methods other than systematic sampling).

See Section 7 of Antal and Tillé (2014) for a clear description of how the replicates are formed. The paper presents two options for the resampling probabilities used in replication: the R function uses the option referred to in the paper as "the π-bootstrap."

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

Antal, E. and Tillé, Y. (2014). "A new resampling method for sampling designs without replacement: The doubled half bootstrap." **Computational Statistics**, *29*(5), 1345-1363. https://doi.org/10.1007/s00180-014-0495-0

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 = "Antal-Tille".

Use estimate_boot_reps_for_target_cv to help choose the number of bootstrap replicates.

Examples

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

 # Example 1: A cluster sample

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

   replicate_factors <- make_doubled_half_bootstrap_weights(
     num_replicates      = 5,
     samp_unit_ids       = library_multistage_sample$PSU_ID,
     strata_ids          = rep(1, times = nrow(library_multistage_sample)),
     samp_unit_sel_probs = library_multistage_sample$PSU_SAMPLING_PROB,
     output              = "factors"
   )

 # 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_doubled_half_bootstrap_weights(
     num_replicates      = 5000,
     samp_unit_ids       = pps_wor_design$cluster[,1],
     strata_ids          = pps_wor_design$strata[,1],
     samp_unit_sel_probs = pps_wor_design$prob
   )

   ## 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)
```

`make_fays_gen_rep_factors`
Form replication factors using Fay's generalized replication method

---

Description

Generate a matrix of replication factors using Fay's generalized replication method. This method yields a fully efficient variance estimator if a sufficient number of replicates is used.

Usage

```
make_fays_gen_rep_factors(Sigma, max_replicates = Inf, balanced = TRUE)
```

Arguments

`Sigma`	A quadratic form matrix corresponding to a target variance estimator. Must be positive semidefinite.
`max_replicates`	The maximum number of replicates to allow. The function will attempt to create the minimum number of replicates needed to produce a fully-efficient variance estimator. If more replicates are needed than `max_replicates`, then the full number of replicates needed will be created, but only a random subsample will be retained.
`balanced`	If `balanced=TRUE`, the replicates will all contribute equally to variance estimates, but the number of replicates needed may slightly increase.

Value

A matrix of replicate factors, with the number of rows matching the number of rows of `Sigma` and the number of columns less than or equal to `max_replicates`. To calculate variance estimates using these factors, use the overall scale factor given by calling `attr(x, "scale")` on the result.

Statistical Details

See Fay (1989) for a full explanation of Fay's generalized replication method. This documentation provides a brief overview.

Let $\mathbf{\Sigma}$ be the quadratic form matrix for a target variance estimator, which is assumed to be positive semidefinite. Suppose the rank of $\mathbf{\Sigma}$ is k, and so $\mathbf{\Sigma}$ can be represented by the spectral decomposition

of k eigenvectors and eigenvalues, where the r-th eigenvector and eigenvalue are denoted $\mathbf{v}_{(r)}$ and λ_r, respectively.

$$\mathbf{\Sigma} = \sum_{r=1}^{k} \lambda_r \mathbf{v}_{(r)} \mathbf{v'}_{(r)}$$

If `balanced = FALSE`, then we let $\mathbf{H}$ denote an identity matrix with $k' = k$ rows/columns. If `balanced = TRUE`, then we let $\mathbf{H}$ be a Hadamard matrix (with all entries equal to 1 or -1), of order $k' \geq k$. Let $\mathbf{H}_{mr}$ denote the entry in row m and column r of $\mathbf{H}$.

Then k' replicates are formed as follows. Let r denote a given replicate, with $r = 1, ..., k'$, and let c denote some positive constant (yet to be specified).

The r-th replicate adjustment factor $\mathbf{f}_r$ is formed as:

$$\mathbf{f}_r = 1 + c \sum_{m=1}^{k} H_{mr} \lambda_{(m)}^{\frac{1}{2}} \mathbf{v}_{(m)}$$

If `balanced = FALSE`, then $c = 1$. If `balanced = TRUE`, then $c = \frac{1}{\sqrt{k'}}$.

If any of the replicates are negative, you can use `rescale_reps`, which recalculates the replicate factors with a smaller value of c.

If all k' replicates are used, then variance estimates are calculated as:

$$v_{rep}\left(\hat{T}_y\right) = \sum_{r=1}^{k'} \left(\hat{T}_y^{*(r)} - \hat{T}_y\right)^2$$

For population totals, this replication variance estimator will *exactly* match the target variance estimator if the number of replicates k' matches the rank of Σ.

The Number of Replicates

If `balanced=TRUE`, the number of replicates created may need to increase slightly. This is due to the fact that a Hadamard matrix of order $k' \geq k$ is used to balance the replicates, and it may be necessary to use order $k' > k$.

If the number of replicates k' is too large for practical purposes, then one can simply retain only a random subset of R of the k' replicates. In this case, variances are calculated as follows:

$$v_{rep}\left(\hat{T}_y\right) = \frac{k'}{R} \sum_{r=1}^{R} \left(\hat{T}_y^{*(r)} - \hat{T}_y\right)^2$$

This is what happens if `max_replicates` is less than the matrix rank of `Sigma`: only a random subset of the created replicates will be retained.

Subsampling replicates is only recommended when using `balanced=TRUE`, since in this case every replicate contributes equally to variance estimates. If `balanced=FALSE`, then randomly subsampling replicates is valid but may produce large variation in variance estimates since replicates in that case may vary greatly in their contribution to variance estimates.

Reproducibility

If `balanced=TRUE`, a Hadamard matrix is used as described above. The Hadamard matrix is deterministically created using the function `hadamard()` from the 'survey' package. However, the order of rows/columns is randomly permuted before forming replicates.

In general, column-ordering of the replicate weights is random. To ensure exact reproducibility, it is recommended to call `set.seed()` before using this function.

References

Fay, Robert. 1989. "Theory And Application Of Replicate Weighting For Variance Calculations." In, 495-500. Alexandria, VA: American Statistical Association. http://www.asasrms.org/Proceedings/papers/1989_033.pdf

See Also

Use `rescale_reps` to eliminate negative adjustment 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 replication adjustment factors ----

  adjustment_factors <- make_fays_gen_rep_factors(
    Sigma = horvitz_thompson_matrix,
    max_replicates = 50
  )
  attr(adjustment_factors, 'scale')

# Compute the Horvitz-Thompson estimate and the replication estimate
```

```
ht_estimate <- svydesign(data = election_pps, ids = ~ 1,
                         prob = diag(election_jointprob),
                         pps = ppsmat(election_jointprob)) |>
  svytotal(x = ~ Kerry)

rep_estimate <- svrepdesign(
  data = election_pps,
  weights = ~ wt,
  repweights = adjustment_factors,
  combined.weights = FALSE,
  scale = attr(adjustment_factors, 'scale'),
  rscales = rep(1, times = ncol(adjustment_factors)),
  type = "other",
  mse = TRUE
) |>
  svytotal(x = ~ Kerry)

SE(rep_estimate)
SE(ht_estimate)
SE(rep_estimate) / SE(ht_estimate)

## End(Not run)
```

`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 semidefinite (this covers a large class of variance estimators).

Usage

```
make_gen_boot_factors(Sigma, num_replicates, tau = 1, exact_vcov = FALSE)
```

Arguments

`Sigma`	The matrix of the quadratic form used to represent the variance estimator. Must be positive semidefinite.
`num_replicates`	The number of bootstrap replicates to create.
`tau`	Either `"auto"`, or a single number; the default value is 1. 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`. Instead of using `tau="auto"`, the user can instead use the function `rescale_reps()` to rescale the replicates later.

`exact_vcov` If `exact_vcov=TRUE`, the replicate factors will be generated such that their variance-covariance matrix exactly matches the target variance estimator's quadratic form (within numeric precision). This is desirable as it causes variance estimates for totals to closely match the values from the target variance estimator. This requires that `num_replicates` exceeds the rank of `Sigma`. The replicate factors are generated by applying PCA-whitening to a collection of draws from a multivariate Normal distribution, then applying a coloring transformation to the whitened collection of draws.

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 semidefinite.

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

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_kernel_var_matrix`

Make a quadratic form matrix for the kernel-based variance estimator of Breidt, Opsomer, and Sanchez-Borrego (2016)

Description

Constructs the quadratic form matrix for the kernel-based variance estimator of Breidt, Opsomer, and Sanchez-Borrego (2016). The bandwidth is automatically chosen to result in the smallest possible nonempty kernel window.

Usage

```
make_kernel_var_matrix(x, kernel = "Epanechnikov", bandwidth = "auto")
```

Arguments

`x`	A numeric vector, giving the values of an auxiliary variable.
`kernel`	The name of a kernel function. Currently only "Epanechnikov" is supported.

`bandwidth` The bandwidth to use for the kernel. The default value is `"auto"`, which means that the bandwidth will be chosen automatically to produce the smallest window size while ensuring that every unit has a nonempty window, as suggested by Breidt, Opsomer, and Sanchez-Borrego (2016). Otherwise, the user can supply their own value, which can be a single positive number.

Details

This kernel-based variance estimator was proposed by Breidt, Opsomer, and Sanchez-Borrego (2016), for use with samples selected using systematic sampling or where only a single sampling unit is selected from each stratum (sometimes referred to as "fine stratification").

Suppose there are n sampled units, and for each unit i there is a numeric population characteristic x_i and there is a weighted total $\hat{Y}_i$, where $\hat{Y}_i$ is only observed in the selected sample but x_i is known prior to sampling.

The variance estimator has the following form:

$$\hat{V}_{ker} = \frac{1}{C_d} \sum_{i=1}^{n} (\hat{Y}_i - \sum_{j=1}^{n} d_j(i) \hat{Y}_j)^2$$

The terms $d_j(i)$ are kernel weights given by

$$d_j(i) = \frac{K(\frac{x_i - x_j}{h})}{\sum_{j=1}^{n} K(\frac{x_i - x_j}{h})}$$

where $K(\cdot)$ is a symmetric, bounded kernel function and h is a bandwidth parameter. The normalizing constant C_d is computed as:

$$C_d = \frac{1}{n} \sum_{i=1}^{n} (1 - 2d_i(i) + \sum_{j=1}^{H} d_j^2(i))$$

If $n = 2$, then the estimator is simply the estimator used for simple random sampling without replacement.

If $n = 1$, then the matrix simply has an entry equal to 0.

Value

The quadratic form matrix for the variance estimator, with dimension equal to the length of `x`. The resulting object has an attribute `bandwidth` that can be retrieved using `attr(Q, 'bandwidth')`

References

Breidt, F. J., Opsomer, J. D., & Sanchez-Borrego, I. (2016). "*Nonparametric Variance Estimation Under Fine Stratification: An Alternative to Collapsed Strata*." **Journal of the American Statistical Association**, 111(514), 822-833. https://doi.org/10.1080/01621459.2015.1058264

Examples

```
# The auxiliary variable has the same value for all units
make_kernel_var_matrix(c(1, 1, 1))

# The auxiliary variable differs across units
make_kernel_var_matrix(c(1, 2, 3))

# View the bandwidth that was automatically selected
Q <- make_kernel_var_matrix(c(1, 2, 4))
attr(Q, 'bandwidth')
```

---

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",
  probs = NULL,
  joint_probs = NULL,
  cluster_ids = NULL,
  strata_ids = NULL,
  strata_pop_sizes = NULL,
  sort_order = NULL,
  aux_vars = 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_pop_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_pop_sizes`. Optionally, to use finite population correction factors, one can also use the argument `strata_pop_sizes`.
- **"Deville-1"**: A variance estimator for unequal-probability sampling without replacement, described in Matei and Tillé (2005) as "Deville 1". If this option is used, then it is necessary to also use the arguments `strata_ids`, `cluster_ids`, and `probs`.
- **"Deville-2"**: A variance estimator for unequal-probability sampling without replacement, described in Matei and Tillé (2005) as "Deville 2". If this option is used, then it is necessary to also use the arguments `strata_ids`, `cluster_ids`, and `probs`.
- **"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.
- **"BOSB"**:
  The kernel-based variance estimator proposed by Breidt, Opsomer, and Sanchez-Borrego (2016) for use with systematic samples or other finely stratified designs. Uses the Epanechnikov kernel with the bandwidth automatically chosen to result in the smallest possible nonempty kernel window.
- **"Deville-Tille"**: The estimator of Deville and Tillé (2005), developed for balanced sampling using the cube method.
- **"Beaumont-Emond"**: The variance estimator of Beaumont and Emond (2022) for multistage unequal-probability sampling without replacement.

`probs` Required if `variance_estimator` equals `"Deville-1"`, `"Deville-2"`, or `"Deville-Tille"`. This should be a matrix or data frame of sampling probabilities. If there are multiple stages of sampling, then `probs` can have multiple columns, with one column for each level of sampling to be accounted for by the variance estimator.

`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.

`aux_vars`
: Required if `variance_estimator` equals `"Deville-Tille"`. A matrix of auxiliary variables.

Value

The matrix of the quadratic form representing the variance estimator.

Variance Estimators

See variance-estimators for a description of each variance estimator.

Arguments required for each variance estimator

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

variance_estimator	probs	joint_probs	cluster_ids	strata_ids	strata_pop_sizes	sort_order	aux_vars
Stratified Multistage SRS			Required	Required	Required		
Ultimate Cluster			Required	Required	Optional		
SD1			Required	Optional	Optional	Required	
SD2			Required	Optional	Optional	Required	
Deville-1	Required		Required	Optional			
Deville-2	Required		Required	Optional			
Beaumont-Emond	Required		Required	Optional			
Deville-Tille	Required		Required	Optional			Required
BOSB			Required	Optional			Required
Yates-Grundy		Required					
Horvitz-Thompson		Required					

See Also

See variance-estimators for a description of each variance estimator.

For a two-phase design, the function make_twophase_quad_form combines the quadratic form matrix from each phase.

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

  ##_ Compare against result from 'survey' package
  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

# Example 4: Deville estimators ----

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

 deville_quad_form <- make_quad_form_matrix(
     variance_estimator = 'Deville-1',
     cluster_ids = library_multistage_sample[,c("PSU_ID", "SSU_ID")],
     strata_ids = cbind(rep(1, times = nrow(library_multistage_sample)),
                        library_multistage_sample$PSU_ID),
     probs = library_multistage_sample[,c("PSU_SAMPLING_PROB",
                                          "SSU_SAMPLING_PROB")]
 )

## 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).

The formulas and algorithms for the replication factors are described by Beaumont and Émond (2022). Below, we list the relevant equations and sections of the paper for each sampling method.

- "SRSWR" - Equation 19 of Beaumont and Émond (2022)
- "PPSWR" - Equation 19 of Beaumont and Émond (2022)
- "SRSWOR" - Equation 20 of Beaumont and Émond (2022)
- "PPSWOR" - Equations 24 and 21 of Beaumont and Émond (2022)
- "Poisson" - See section 3.1 of Beaumont and Émond (2022)

For stratified sampling, the replicate factors are generated independently in each stratum. For cluster sampling at a given stage, the replicate factors are generated at the cluster level and then the cluster's replicate factors are applied to all units in the cluster.

For multistage sampling, replicate factors are generated using the method described in Section 7 ("Bootstrap for Multistage Sampling").

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)
```

`make_sdr_replicate_factors` *Create matrix of replicate factors to use for Successive Difference Replication Method*

Description

Create matrix of replicate factors to use for Successive Difference Replication Method

Usage

```
make_sdr_replicate_factors(
  n,
  target_number_of_replicates,
  use_normal_hadamard = FALSE
)
```

Arguments

n	The number of sampling units
target_number_of_replicates	The target number of replicates to create. This will determine the order of the Hadamard matrix to use when creating replicate factors. The actual number of replicates will be a multiple of 4. If use_normal_hadamard = FALSE, then the actual number of replicates will be 4×2^k for some integer k, which means that the actual number of replicates might be much larger than the target.
use_normal_hadamard	Whether to use a normal Hadamard matrix: that is, a matrix whose first row and first column only have entries equal to 1.

Value

A matrix of replicate factors, with n rows and the number of columns corresponding to the order of the Hadamard matrix used (which is greater than or equal to target_number_of_replicates).

Examples

```
# Note that one of the replicates has every factor equal to 1
# Also note that this matches Table 1 in Ash (2014)
make_sdr_replicate_factors(
  n = 4,
  target_number_of_replicates = 4,
  use_normal_hadamard = TRUE
)

# Note the difference when using a non-normal Hadamard matrix
rep_factors <- make_sdr_replicate_factors(
  n = 4,
  target_number_of_replicates = 4,
  use_normal_hadamard = FALSE
)
print(rep_factors)

# These replicate factors are equivalent
# to the SD2 variance estimator
tcrossprod(rep_factors - 1)

# Compare to the quadratic form of the SD2 estimator
sd2_quad_form <- make_quad_form_matrix(
  variance_estimator = "SD2",
```

```
  cluster_ids        = matrix(1:4, ncol = 1),
  sort_order         = matrix(1:4, ncol = 1)
)
print(sd2_quad_form)
```

`make_twophase_quad_form` *Combine quadratic forms from each phase of a two phase design*

Description

This function combines quadratic forms from each phase of a two phase design, so that the combined variance of the entire two-phase sampling design can be estimated.

Usage

```
make_twophase_quad_form(
  sigma_1,
  sigma_2,
  phase_2_joint_probs,
  ensure_psd = TRUE
)
```

Arguments

`sigma_1`	The quadratic form for the first phase variance estimator, subsetted to only include cases selected in the phase two sample.
`sigma_2`	The quadratic form for the second phase variance estimator, conditional on the selection of the first phase sample.
`phase_2_joint_probs`	The matrix of conditional joint inclusion probabilities for the second phase, given the selected first phase sample.
`ensure_psd`	If `TRUE` (the default), ensures that the result is a positive semidefinite matrix. This is necessary if the quadratic form is used as an input for replication methods such as the generalized bootstrap. For details, see the help section entitled "Ensuring the Result is Positive Semidefinite".

Value

A quadratic form matrix that can be used to estimate the sampling variance from a two-phase sample design.

Statistical Details

The two-phase variance estimator has a quadratic form matrix $\mathbf{\Sigma}_{ab}$ given by:

$$\mathbf{\Sigma}_{ab} = W_b^{-1}(\mathbf{\Sigma}_{a'} \circ D_b)W_b^{-1} + \mathbf{\Sigma}_b$$

The first term estimates the variance contribution from the first phase of sampling, while the second term estimates the variance contribution from the second phase of sampling.

The full quadratic form of the variance estimator is:

$$v(\hat{t_y}) = \breve{y}' \mathbf{\Sigma}_{ab} \breve{y}$$

where the weighted variable $\breve{y}_k = \frac{y_k}{\pi_{ak}\pi_{bk}}$, is formed using the first phase inclusion probability, denoted π_{ak}, and the conditional second phase inclusion probability (given the selected first phase sample), denoted π_{bk}.

The notation for this estimator is as follows:

- n_a denotes the first phase sample size.
- n_b denotes the second phase sample size.
- $\mathbf{\Sigma}_a$ denotes the matrix of dimension $n_a \times n_a$ representing the quadratic form for the variance estimator used for the full first-phase design.
- $\mathbf{\Sigma}_{a'}$ denotes the matrix of dimension $n_b \times n_b$ formed by subsetting the rows and columns of $\mathbf{\Sigma}_a$ to only include cases selected in the second-phase sample.
- $\mathbf{\Sigma}_b$ denotes the matrix of dimension $n_b \times n_b$ representing the Horvitz-Thompson estimator of variance for the second-phase sample, conditional on the selected first-phase sample.
- $\boldsymbol{D}_b$ denotes the $n_b \times n_b$ matrix of weights formed by the inverses of the second-phase joint inclusion probabilities, with element kl equal to π_{bkl}^{-1}, where π_{bkl} is the conditional probability that units k and l are included in the second-phase sample, given the selected first-phase sample. Note that this matrix will often not be positive semidefinite, and so the two-phase variance estimator has a quadratic form which is not necessarily positive semidefinite.
- $\boldsymbol{W}_b$ denotes the diagonal $n_b \times n_b$ matrix whose k-th diagonal entry is the second-phase weight π_{bk}^{-1}, where π_{bk} is the conditional probability that unit k is included in the second-phase sample, given the selected first-phase sample.

Ensuring the Result is Positive Semidefinite

Note that the matrix $(\mathbf{\Sigma}_{a'} \circ D_b)$ may not be positive semidefinite, since the matrix D_b is not guaranteed to be positive semidefinite. If $(\mathbf{\Sigma}_{a'} \circ D_b)$ is found not to be positive semidefinite, then it is approximated by the nearest positive semidefinite matrix in the Frobenius norm, using the method of Higham (1988).

This approximation is discussed by Beaumont and Patak (2012) in the context of forming replicate weights for two-phase samples. The authors argue that this approximation should lead to only a small overestimation of variance.

Since $(\mathbf{\Sigma}_{a'} \circ D_b)$ is a real, symmetric matrix, this is equivalent to "zeroing out" negative eigenvalues. To be more precise, denote $A = (\mathbf{\Sigma}_{a'} \circ D_b)$. Then we can form the spectral decomposition $A = \Gamma \Lambda \Gamma'$, where Λ is the diagonal matrix whose entries are eigenvalues of A. The method of Higham (1988) is to approximate A with $\tilde{A} = \Gamma \Lambda_+ \Gamma'$, where the ii-th entry of Λ_+ is $\max(\Lambda_{ii}, 0)$.

References

See Section 7.5 of Tillé (2020) or Section 9.3 of Särndal, Swensson, and Wretman (1992) for an overview of variance estimation for two-phase sampling. In the case where the Horvitz-Thompson variance estimator is used for both phases, the method used in this function is equivalent to equation (9.3.8) of Särndal, Swensson, and Wretman (1992) and equation (7.7) of Tillé (2020). However, this function can be used for any combination of first-phase and second-phase variance estimators, provided that the joint inclusion probabilities from the second-phase design are available and are all nonzero.

- 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.

- Higham, N. J. (1988). "*Computing a nearest symmetric positive semidefinite matrix.*" Linear Algebra and Its Applications, 103, 103-118.

- Särndal, C.-E., Swensson, B., & Wretman, J. (1992). "*Model Assisted Survey Sampling*." Springer New York.

- Tillé, Y. (2020). "*Sampling and estimation from finite populations*." (I. Hekimi, Trans.). Wiley.

See Also

For each phase of sampling, the function make_quad_form_matrix can be used to create the appropriate quadratic form matrix.

Examples

```
## Not run:

## ---------------------- Example 1 ------------------------##
## First phase is a stratified multistage sample           ##
## Second phase is a simple random sample                  ##
##---------------------------------------------------------##
data('library_multistage_sample', package = 'svrep')

# Load first-phase sample
```

```
  twophase_sample <- library_multistage_sample

# Select second-phase sample
  set.seed(2022)

  twophase_sample[['SECOND_PHASE_SELECTION']] <- sampling::srswor(
    n = 100,
    N = nrow(twophase_sample)
  ) |> as.logical()

# Declare survey design
  twophase_design <- twophase(
    method = "full",
    data = twophase_sample,
    # Identify the subset of first-phase elements
    # which were selected into the second-phase sample
    subset = ~ SECOND_PHASE_SELECTION,
    # Describe clusters, probabilities, and population sizes
    # at each phase of sampling
    id = list(~ PSU_ID + SSU_ID,
              ~ 1),
    probs = list(~ PSU_SAMPLING_PROB + SSU_SAMPLING_PROB,
                 NULL),
    fpc = list(~ PSU_POP_SIZE + SSU_POP_SIZE,
               NULL)
  )

# Get quadratic form matrix for the first phase design
  first_phase_sigma <- get_design_quad_form(
    design = twophase_design$phase1$full,
    variance_estimator = "Stratified Multistage SRS"
  )

# Subset to only include cases sampled in second phase

  first_phase_sigma <- first_phase_sigma[twophase_design$subset,
                                         twophase_design$subset]

# Get quadratic form matrix for the second-phase design
  second_phase_sigma <- get_design_quad_form(
    design = twophase_design$phase2,
    variance_estimator = "Ultimate Cluster"
  )

# Get second-phase joint probabilities
  n <- twophase_design$phase2$fpc$sampsize[1,1]
  N <- twophase_design$phase2$fpc$popsize[1,1]

  second_phase_joint_probs <- Matrix::Matrix((n/N)*((n-1)/(N-1)),
                                     nrow = n, ncol = n)
  diag(second_phase_joint_probs) <- rep(n/N, times = n)

# Get quadratic form for entire two-phase variance estimator
```

```
  twophase_quad_form <- make_twophase_quad_form(
   sigma_1 = first_phase_sigma,
   sigma_2 = second_phase_sigma,
   phase_2_joint_probs = second_phase_joint_probs
  )

  # Use for variance estimation

    rep_factors <- make_gen_boot_factors(
      Sigma = twophase_quad_form,
      num_replicates = 500
    )

    library(survey)

    combined_weights <- 1/twophase_design$prob

    twophase_rep_design <- svrepdesign(
      data = twophase_sample |>
        subset(SECOND_PHASE_SELECTION),
      type = 'other',
      repweights = rep_factors,
      weights = combined_weights,
      combined.weights = FALSE,
      scale = attr(rep_factors, 'scale'),
      rscales = attr(rep_factors, 'rscales')
    )

    svymean(x = ~ LIBRARIA, design = twophase_rep_design)


## ---------------------- Example 2 ------------------------##
## First phase is a stratified systematic sample            ##
## Second phase is nonresponse, modeled as Poisson sampling ##
##----------------------------------------------------------##

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

# Determine quadratic form for full first-phase sample variance estimator

  full_phase1_quad_form <- make_quad_form_matrix(
    variance_estimator = "SD2",
    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
  )

# Identify cases included in phase two sample
# (in this example, respondents)
  phase2_inclusion <- (
    library_stsys_sample$RESPONSE_STATUS == "Survey Respondent"
  )
```

```
  phase2_sample <- library_stsys_sample[phase2_inclusion,]

# Estimate response propensities

  response_propensities <- glm(
    data = library_stsys_sample,
    family = quasibinomial('logit'),
    formula = phase2_inclusion ~ 1,
    weights = 1/library_stsys_sample$SAMPLING_PROB
  ) |>
    predict(type = "response",
            newdata = phase2_sample)

# Estimate conditional joint inclusion probabilities for second phase

  phase2_joint_probs <- outer(response_propensities, response_propensities)
  diag(phase2_joint_probs) <- response_propensities

# Determine quadratic form for variance estimator of second phase
# (Horvitz-Thompson estimator for nonresponse modeled as Poisson sampling)

  phase2_quad_form <- make_quad_form_matrix(
    variance_estimator = "Horvitz-Thompson",
    joint_probs = phase2_joint_probs
  )

# Create combined quadratic form for entire design

 twophase_quad_form <- make_twophase_quad_form(
   sigma_1 = full_phase1_quad_form[phase2_inclusion, phase2_inclusion],
   sigma_2 = phase2_quad_form,
   phase_2_joint_probs = phase2_joint_probs
 )

 combined_weights <- 1/(phase2_sample$SAMPLING_PROB * response_propensities)

# Use for variance estimation

  rep_factors <- make_gen_boot_factors(
    Sigma = twophase_quad_form,
    num_replicates = 500
  )

  library(survey)

  twophase_rep_design <- svrepdesign(
    data = phase2_sample,
    type = 'other',
    repweights = rep_factors,
    weights = combined_weights,
    combined.weights = FALSE,
    scale = attr(rep_factors, 'scale'),
    rscales = attr(rep_factors, 'rscales')
```

```
  )

  svymean(x = ~ LIBRARIA, design = twophase_rep_design)

## 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")
)
```

---

rescale_replicates *Rescale replicate factors*

Description

Rescale replicate factors. The main application of this rescaling is to ensure that all replicate weights are strictly positive.

Note that this rescaling has no impact on variance estimates for totals (or other linear statistics), but variance estimates for nonlinear statistics will be affected by the rescaling.

Usage

```
rescale_replicates(x, new_scale = NULL, min_wgt = 0.01, digits = 2)
```

Arguments

x	Either a replicate survey design object, or a numeric matrix of replicate weights. If x is a matrix, then it must have an attribute scale specifying the scale factor for variance estimation.

`new_scale` Either a single positive number, or `NULL`. If supplied, `new_scale` will be the new scale factor used for estimating variances. For example, with `B=10` bootstrap replicates, specifying `new_scale=0.2` will change the scale factor for variance estimation from $B^{-1} = 0.1$ to 0.2.
If `new_scale=NULL` or is left unspecified, then the argument `min_wgt` should be used instead.

`min_wgt` Should only be used if `new_scale=NULL` or `new_scale` is left unspecified. Specifies the minimum acceptable value for the rescaled weights, which will be used to automatically determine the new scale. Must be at least zero and must be less than one.

`digits` Only used if the argument `min_wgt` is used. Specifies the number of decimal places to use for the scale adjustment factor, so that the ratio `new_scale/orig_scale` will be a number with at most `digits` decimal places. For example, let `orig_scale` denote the original scale and `new_scale` denote the new scale. Then if the user specifies `digits=2`, the ratio `new_scale/orig_scale` will be a number such as 0.25 or 0.10. This results in documentation that is easier to read.

Details

Let $\mathbf{A} = \left[\mathbf{a}^{(1)} \cdots \mathbf{a}^{(b)} \cdots \mathbf{a}^{(B)}\right]$ denote the $(n \times B)$ matrix of replicate adjustment factors. The overall scale factor C is used for estimating variances with the formula

$$v(\hat{y}) = C \sum_{b=1}^{B} (\hat{y}_b - \hat{y})^2$$

The scale factor is changed from C to C' by rescaling the replicate factor matrix $\mathbf{A}$ using the transformation

$$1 + \sqrt{\frac{C}{C'}}(\mathbf{A} - 1)$$

Value

If the input is a numeric matrix, returns the rescaled matrix. If the input is a replicate survey design object, returns an updated replicate survey design object.

For a replicate survey design object, results depend on whether the object has a matrix of replicate factors rather than a matrix of replicate weights (which are the product of replicate factors and sampling weights). If the design object has `combined.weights=FALSE`, then the replication factors are adjusted. If the design object has `combined.weights=TRUE`, then the replicate weights are adjusted. It is strongly recommended to only use the rescaling method for replication factors rather than the weights.

For a replicate survey design object, the `scale` element of the design object will be updated appropriately.

References

Rescaling was suggested by Fay (1989) for the specific application of creating replicate factors using his generalized replication method. This kind of rescaling is commony used in balanced repeated replication to implement Fay's method of balanced repeated replication. Beaumont and

Patak (2012) provided an extended discussion of rescaling methods in the context of rescaling generalized bootstrap replication factors to avoid negative replicate weights.

- 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.

- Fay, Robert. 1989. "Theory And Application Of Replicate Weighting For Variance Calculations." In, 495-500. Alexandria, VA: American Statistical Association. http://www.asasrms.org/Proceedings/papers/1989_033.pdf

Examples

```
# Example 1: Rescaling a matrix of replicate weights to avoid negative weights

rep_wgts <- matrix(
  c(1.69742746694909, -0.230761178913411, 1.53333377634192,
    0.0495043413294782, 1.81820367441039, 1.13229198793703,
    1.62482013925955, 1.0866133494029, 0.28856654131668,
    0.581930729719006, 0.91827012312825, 1.49979905894482,
    1.26281337410693, 1.99327362761477, -0.25608700039304),
  nrow = 3, ncol = 5
)
attr(rep_wgts, 'scale') <- 1/5

rescaled_wgts <- rescale_replicates(rep_wgts, min_wgt = 0.01)

print(rep_wgts)
print(rescaled_wgts)

# Example 2: Rescaling replicate weights with a specified value of 'tau'

rescaled_wgts <- rescale_replicates(rep_wgts, new_scale = 1/10)
print(rescaled_wgts)

# Example 3: Rescaling replicate weights of a survey design object
set.seed(2023)
library(survey)
data('mu284', package = 'survey')

## First create a bootstrap design object
svy_design_object <- svydesign(
  data = mu284,
  ids = ~ id1 + id2,
  fpc = ~  n1 +  n2
)

boot_design <- as_gen_boot_design(
  design = svy_design_object,
  variance_estimator = "Stratified Multistage SRS",
  replicates = 5
)

## Rescale the weights
```

```
rescaled_boot_design <- boot_design |>
  rescale_replicates(min_wgt = 0.01)

boot_wgts <- weights(boot_design, "analysis")
rescaled_boot_wgts <- weights(rescaled_boot_design, 'analysis')

print(boot_wgts)
print(rescaled_boot_wgts)
```

`rescale_reps` *Rescale replicate factors*

Description

Deprecated: Please use the function `rescale_replicates` instead.

Rescale replicate factors. The main application of this rescaling is to ensure that all replicate weights are strictly positive.

Note that this rescaling has no impact on variance estimates for totals (or other linear statistics), but variance estimates for nonlinear statistics will be affected by the rescaling.

Usage

```
rescale_reps(x, tau = NULL, min_wgt = 0.01, digits = 2)
```

Arguments

`x`	Either a replicate survey design object, or a numeric matrix of replicate weights.
`tau`	Either a single positive number, or `NULL`. This is the rescaling constant τ used in the transformation $\frac{w+\tau-1}{\tau}$, where w is the original weight. If `tau=NULL` or is left unspecified, then the argument `min_wgt` should be used instead, in which case, τ is automatically set to the smallest value needed to rescale the replicate weights such that they are all at least `min_wgt`.
`min_wgt`	Should only be used if `tau=NULL` or `tau` is left unspecified. Specifies the minimum acceptable value for the rescaled weights, which will be used to automatically determine the value τ used in the transformation $\frac{w+\tau-1}{\tau}$, where w is the original weight. Must be at least zero and must be less than one.
`digits`	Only used if the argument `min_wgt` is used. Specifies the number of decimal places to use for choosing `tau`. Using a smaller number of `digits` is useful simply for producing easier-to-read documentation.

Details

Let $\mathbf{A} = \left[\mathbf{a}^{(1)} \cdots \mathbf{a}^{(b)} \cdots \mathbf{a}^{(B)}\right]$ denote the $(n \times B)$ matrix of replicate 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 adjustment factors.

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. For example, for bootstrap replicates, the adjustment factor becomes $\frac{\tau^2}{B}$ instead of $\frac{1}{B}$.

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

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

Value

If the input is a numeric matrix, returns the rescaled matrix. If the input is a replicate survey design object, returns an updated replicate survey design object.

For a replicate survey design object, results depend on whether the object has a matrix of replicate factors rather than a matrix of replicate weights (which are the product of replicate factors and sampling weights). If the design object has `combined.weights=FALSE`, then the replication factors are adjusted. If the design object has `combined.weights=TRUE`, then the replicate weights are adjusted. It is strongly recommended to only use the rescaling method for replication factors rather than the weights.

For a replicate survey design object, the `scale` element of the design object will be updated appropriately, and an element `tau` will also be added. If the input is a matrix instead of a survey design object, the result matrix will have an attribute named `tau` which can be retrieved using `attr(x, 'tau')`.

References

This method was suggested by Fay (1989) for the specific application of creating replicate factors using his generalized replication method. Beaumont and Patak (2012) provided an extended discussion on this rescaling method in the context of rescaling generalized bootstrap replication factors to avoid negative replicate weights.

The notation used in this documentation is taken from Beaumont and Patak (2012).

- 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.

- Fay, Robert. 1989. "Theory And Application Of Replicate Weighting For Variance Calculations." In, 495-500. Alexandria, VA: American Statistical Association. http://www.asasrms.org/Proceedings/papers/1989_033.pdf

Examples

```
## Not run:
```

```
# Example 1: Rescaling a matrix of replicate weights to avoid negative weights

rep_wgts <- matrix(
  c(1.69742746694909, -0.230761178913411, 1.53333377634192,
    0.0495043413294782, 1.81820367441039, 1.13229198793703,
    1.62482013925955, 1.0866133494029, 0.28856654131668,
    0.581930729719006, 0.91827012312825, 1.49979905894482,
    1.26281337410693, 1.99327362761477, -0.25608700039304),
  nrow = 3, ncol = 5
)

rescaled_wgts <- rescale_reps(rep_wgts, min_wgt = 0.01)

print(rep_wgts)
print(rescaled_wgts)

# Example 2: Rescaling replicate weights with a specified value of 'tau'

rescaled_wgts <- rescale_reps(rep_wgts, tau = 2)
print(rescaled_wgts)

# Example 3: Rescaling replicate weights of a survey design object
set.seed(2023)
library(survey)
data('mu284', package = 'survey')

## First create a bootstrap design object
svy_design_object <- svydesign(
  data = mu284,
  ids = ~ id1 + id2,
  fpc = ~ n1 + n2
)

boot_design <- as_gen_boot_design(
  design = svy_design_object,
  variance_estimator = "Stratified Multistage SRS",
  replicates = 5, tau = 1
)

## Rescale the weights
rescaled_boot_design <- boot_design |>
  rescale_reps(min_wgt = 0.01)

boot_wgts <- weights(boot_design, "analysis")
rescaled_boot_wgts <- weights(rescaled_boot_design, 'analysis')

print(boot_wgts)
print(rescaled_boot_wgts)

## End(Not run)
```

`shuffle_replicates` *Shuffle the order of replicates in a survey design object*

Description

Shuffle the order of replicates in a survey design object. In other words, the order of the columns of replicate weights is randomly permuted.

Usage

```
shuffle_replicates(design)
```

Arguments

`design`	A survey design object, created with either the `survey` or `srvyr` packages.

Value

An updated survey design object, where the order of the replicates has been shuffled (i.e., the order has been randomly permuted).

Examples

```
library(survey)
set.seed(2023)

# Create an example survey design object

  sample_data <- data.frame(
    STRATUM = c(1,1,1,1,2,2,2,2),
    PSU     = c(1,2,3,4,5,6,7,8)
  )

  survey_design <- svydesign(
    data = sample_data,
    strata = ~ STRATUM,
    ids = ~ PSU,
    weights = ~ 1
  )

  rep_design <- survey_design |>
    as_fays_gen_rep_design(variance_estimator = "Ultimate Cluster")

# Inspect replicates before shuffling

  rep_design |> getElement("repweights")

# Inspect replicates after shuffling

  rep_design |>
```

```
shuffle_replicates() |>
getElement("repweights")
```

`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")
```

`subsample_replicates` *Retain only a random subset of the replicates in a design*

Description

Randomly subsamples the replicates of a survey design object, to keep only a subset. The scale factor used in estimation is increased to account for the subsampling.

Usage

```
subsample_replicates(design, n_reps)
```

Arguments

`design`	A survey design object, created with either the `survey` or `srvyr` packages.
`n_reps`	The number of replicates to keep after subsampling

Value

An updated survey design object, where only a random selection of the replicates has been retained. The overall 'scale' factor for the design (accessed with `design$scale`) is increased to account for the sampling of replicates.

Statistical Details

Suppose the initial replicate design has L replicates, with respective constants c_k for $k = 1, \ldots, L$ used to estimate variance with the formula

$$v_R = \sum_{k=1}^{L} c_k \left(\hat{T}_y^{(k)} - \hat{T}_y \right)^2$$

With subsampling of replicates, L_0 of the original L replicates are randomly selected, and then variances are estimated using the formula:

$$v_R = \frac{L}{L_0} \sum_{k=1}^{L_0} c_k \left(\hat{T}_y^{(k)} - \hat{T}_y \right)^2$$

This subsampling is suggested for certain replicate designs in Fay (1989). Kim and Wu (2013) provide a detailed theoretical justification and also propose alternative methods of subsampling replicates.

References

Fay, Robert. 1989. "Theory And Application Of Replicate Weighting For Variance Calculations." In, 495-500. Alexandria, VA: American Statistical Association. http://www.asasrms.org/Proceedings/papers/1989_033.pdf

Kim, J.K. and Wu, C. 2013. "Sparse and Efficient Replication Variance Estimation for Complex Surveys." **Survey Methodology**, Statistics Canada, 39(1), 91-120.

Examples

```
library(survey)
set.seed(2023)

# Create an example survey design object

  sample_data <- data.frame(
    STRATUM = c(1,1,1,1,2,2,2,2),
    PSU     = c(1,2,3,4,5,6,7,8)
  )

  survey_design <- svydesign(
    data = sample_data,
    strata = ~ STRATUM,
    ids = ~ PSU,
    weights = ~ 1
  )
```

```
  rep_design <- survey_design |>
    as_fays_gen_rep_design(variance_estimator = "Ultimate Cluster")

# Inspect replicates before subsampling

  rep_design |> getElement("repweights")

# Inspect replicates after subsampling

  rep_design |>
    subsample_replicates(n_reps = 4) |>
    getElement("repweights")
```

`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)
```

---

svrep-package-options *Package-level options for svrep*

Description

This help page describes the overall options that can be set for your R session, using the function `options()`.

Options for using the 'torch' package to speed up certain operations

The 'torch' package provides access to fast linear algebra routines and is a particularly convenient approach to working with GPUs or conducting multithreaded linear algebra operations.

Setting the following options will allow functions in 'svrep' to use the 'torch' package to speed up certain computationally intensive operations that occur when creating replicate weights, particularly for Fay's generalized replication method or generalized bootstrap methods.

The option `svrep.torch_device` accepts the following options:

- `options(svrep.torch_device = 'none')`: The 'torch' package will not be used.
- `options(svrep.torch_device = 'cpu')`: The 'torch' package will be used with all operations done on the CPU.
- `options(svrep.torch_device = 'cuda')`: The 'torch' package will be used, with operations conducted on the GPU if possible. This requires the user's computer to have a CUDA-enabled GPU.

Note that precise values for matrix decompositions can vary between different linear algebra libraries (including among common BLAS/LAPACK), and so the replicate weights created with 'torch' may not exactly match those created without 'torch'; differences will generally be small.

The following function from 'torch' will control the number of threads used for computations, which can have a large impact on speed.

- `torch::set_num_threads()`: Sets the number of threads that 'torch' can use.

Relevant options from the 'survey' package

The 'survey' package has the following options which are of particular relevance to users of 'svrep'.

- `options(survey.replicates.mse = TRUE/FALSE)`: The default value for this option is `FALSE`. This option controls the default value used for the `mse` argument in the functions `svrepdesign()` and `as.svrepdesign()`.

Call `help('survey.replicates.mse', package = 'survey')` for more details.

In nearly all cases, it is safer to use `options(survey.replicates.mse = TRUE)`, or–better yet–to always specify `svrepdesign(..., mse = TRUE)` or `as.svrepdesign(..., mse = TRUE)` when using functions with an `mse` argument.

For replicate weights created using Fay's generalized replication method or the generalized bootstrap, using `mse = FALSE` can result in badly biased variance estimates.

- `options(survey.multicore = TRUE/FALSE)`: The default value for this option is `FALSE`. Setting this option to `TRUE` means that multiple processors will be used for certain variance calculations involving replicate weights, such as in `svyglm()`.

This can potentially speed up calculations but is not guaranteed to do so.

Call `help('survey.multicore', package = 'survey')` for more details.

`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)
```

`variance-estimators` *Variance Estimators*

Description

This help page describes variance estimators which are commonly used for survey samples. These variance estimators can be used as the basis of the generalized replication methods, implemented with the functions `as_fays_gen_rep_design()`, `as_gen_boot_design()`, `make_fays_gen_rep_factors()`, or `make_gen_boot_factors()`

Shared Notation

Let s denote the selected sample of size n, with elements $i = 1, \ldots, n$. Element i in the sample had probability π_i of being included in the sample. The *pair* of elements ij was sampled with probability π_{ij}.

The population total for a variable is denoted $Y = \sum_{i \in U} y_i$, and the Horvitz-Thompson estimator for $\hat{Y}$ is denoted $\hat{Y} = \sum_{i \in s} y_i/\pi_i$. For convenience, we denote $\breve{y}_i = y_i/\pi_i$.

The true sampling variance of $\hat{Y}$ is denoted $V(\hat{Y})$, while an estimator of this sampling variance is denoted $v(\hat{Y})$.

Horvitz-Thompson

The **Horvitz-Thompson** variance estimator:

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

Yates-Grundy

The **Yates-Grundy** variance estimator:

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

Poisson Horvitz-Thompson

The **Poisson Horvitz-Thompson** variance estimator is simply the Horvitz-Thompson variance estimator, but where $\pi_{ij} = \pi_i \times \pi_j$, which is the case for Poisson sampling.

Stratified Multistage SRS

The **Stratified Multistage SRS** variance estimator is the recursive variance estimator proposed by Bellhouse (1985) and used in the 'survey' package's function svyrecvar. In the case of simple random sampling without replacement (with one or more stages), this estimator exactly matches the Horvitz-Thompson estimator.

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 $y_{hi.}$.

Ultimate Cluster

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.

SD1 and SD2 (Successive Difference Estimators)

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 and is more conservative. 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.

Beaumont-Emond

The **"Beaumont-Emond"** variance estimator was proposed by Beaumont and Emond (2022), intended for designs that use fixed-size, unequal-probability random sampling without replacement. The variance estimator is simply the Horvitz-Thompson variance estimator with the following approximation for the joint inclusion probabilities.

$$\pi_{kl} \approx \pi_k \pi_l \frac{n-1}{(n-1) + \sqrt{(1-\pi_k)(1-\pi_l)}}$$

In the case of cluster sampling, this approximation is applied to the clusters rather than the units within clusters, with n denoting the number of sampled clusters. and the probabilities π referring to the cluster's sampling probability. For stratified samples, the joint probability for units k and l in different strata is simply the product of π_k and π_l.

For multistage samples, this approximation is 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 probabilities from earlier stages of sampling. For example, at a third stage of sampling, the variance estimate from a third-stage stratum is multiplied by $\pi_1 \times \pi_{(2|1)}$, where π_1 is the sampling probability of the first-stage unit and $\pi_{(2|1)}$ is the sampling probability of the second-stage unit within the first-stage unit.

Deville 1 and Deville 2

The **"Deville-1"** and **"Deville-2"** variance estimators are clearly described in Matei and Tillé (2005), and are intended for designs that use fixed-size, unequal-probability random sampling without replacement. These variance estimators have been shown to be effective for designs that use a fixed sample size with a high-entropy sampling method. This includes most PPSWOR sampling methods, but unequal-probability systematic sampling is an important exception.

These variance estimators take the following form:

$$\hat{v}(\hat{Y}) = \sum_{i=1}^{n} c_i (\breve{y}_i - \frac{1}{\sum_{i=k}^{n} c_k} \sum_{k=1}^{n} c_k \breve{y}_k)^2$$

where $\breve{y}_i = y_i/\pi_i$ is the weighted value of the the variable of interest, and c_i depend on the method used:

- **"Deville-1"**:

$$c_i = (1 - \pi_i) \frac{n}{n-1}$$

- **"Deville-2"**:

$$c_i = (1 - \pi_i) \left[1 - \sum_{k=1}^{n} \left(\frac{1 - \pi_k}{\sum_{k=1}^{n} (1 - \pi_k)} \right)^2 \right]^{-1}$$

In the case of simple random sampling without replacement (SRSWOR), these estimators are both identical to the usual stratified multistage SRS estimator (which is itself a special case of the Horvitz-Thompson estimator).

For multistage samples, "Deville-1" and "Deville-2" 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 probabilities from earlier stages of sampling. For example, at a third stage of sampling, the variance estimate from a third-stage stratum is multiplied by $\pi_1 \times \pi_{(2|1)}$, where π_1 is the sampling probability of the first-stage unit and $\pi_{(2|1)}$ is the sampling probability of the second-stage unit within the first-stage unit.

BOSB

This kernel-based variance estimator was proposed by Breidt, Opsomer, and Sanchez-Borrego (2016), for use with samples selected using systematic sampling or where only a single sampling unit is selected from each stratum (sometimes referred to as "fine stratification").

Suppose there are n sampled units, and for each unit i there is a numeric population characteristic x_i and there is a weighted total $\hat{Y}_i$, where $\hat{Y}_i$ is only observed in the selected sample but x_i is known prior to sampling.

The variance estimator has the following form:

$$\hat{V}_{ker} = \frac{1}{C_d} \sum_{i=1}^{n} (\hat{Y}_i - \sum_{j=1}^{n} d_j(i) \hat{Y}_j)^2$$

The terms $d_j(i)$ are kernel weights given by

$$d_j(i) = \frac{K(\frac{x_i - x_j}{h})}{\sum_{j=1}^{n} K(\frac{x_i - x_j}{h})}$$

where $K(\cdot)$ is a symmetric, bounded kernel function and h is a bandwidth parameter. The normalizing constant C_d is computed as:

$$C_d = \frac{1}{n} \sum_{i=1}^{n} (1 - 2d_i(i) + \sum_{j=1}^{H} d_j^2(i))$$

For most functions in the 'svrep' package, the kernel function is the Epanechnikov kernel and the bandwidth is automatically selected to yield the smallest possible nonempty kernel window, as was recommended by Breidt, Opsomer, and Sanchez-Borrego (2016). That's the case for the functions `as_fays_gen_rep_design()`, `as_gen_boot_design()`, `make_quad_form_matrix()`, etc. However, users can construct the quadratic form matrix of this variance estimator using a different kernel and a different bandwidth by directly working with the function `make_kernel_var_matrix()`.

Deville-Tillé

See Section 6.8 of Tillé (2020) for more detail on this estimator, including an explanation of its quadratic form. See Deville and Tillé (2005) for the results of a simulation study comparing this and other alternative estimators for balanced sampling.

The estimator can be written as follows:

$$v(\hat{Y}) = \sum_{k \in S} \frac{c_k}{\pi_k^2} \left(y_k - \hat{y}_k^*\right)^2,$$

where

$$\hat{y}_k^* = \mathbf{z}_k^\top \left(\sum_{\ell \in S} c_\ell \frac{\mathbf{z}_\ell \mathbf{z}_\ell'}{\pi_\ell^2} \right)^{-1} \sum_{\ell \in S} c_\ell \frac{\mathbf{z}_\ell y_\ell}{\pi_\ell^2}$$

and $\mathbf{z}_k$ denotes the vector of auxiliary variables for observation k included in sample S, with inclusion probability π_k. The value c_k is set to $\frac{n}{n-q}(1-\pi_k)$, where n is the number of observations and q is the number of auxiliary variables.

References

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

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

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

Breidt, F. J., Opsomer, J. D., & Sanchez-Borrego, I. (2016). "*Nonparametric Variance Estimation Under Fine Stratification: An Alternative to Collapsed Strata*." **Journal of the American Statistical Association**, 111(514), 822-833. https://doi.org/10.1080/01621459.2015.1058264

Deville, J.C., and Tillé, Y. (2005). "*Variance approximation under balanced sampling.*" **Journal of Statistical Planning and Inference**, 128, 569-591.

Tillé, Y. (2020). "*Sampling and estimation from finite populations*." (I. Hekimi, Trans.). Wiley.

Matei, Alina, and Yves Tillé. (2005). "*Evaluation of Variance Approximations and Estimators in Maximum Entropy Sampling with Unequal Probability and Fixed Sample Size.*" **Journal of Official Statistics**, 21(4):543-70.

Index