Package 'sondage'

November 13, 2026

Type Package

Title Survey Sampling Algorithms

Version 0.15.0

Description Fast implementations of survey sampling algorithms for single-stage probability sampling from finite populations, written in C.
Provides equal probability methods (simple random sampling, systematic, Bernoulli), unequal probability methods (conditional Poisson / maximum entropy, Sampford, Brewer, systematic PPS, Pareto, sequential Poisson, Poisson, Chromy's sequential and systematic PPS with minimum replacement, multinomial), balanced sampling via the cube method, and spatially balanced sampling via the local pivotal method and spatially correlated Poisson sampling.
All sampling functions return design objects carrying sample indices, inclusion probabilities, and design metadata. Generics compute joint inclusion probabilities, pairwise expectations, and sampling covariances for variance estimation. Mostly based on algorithms from Tillé (2006, <doi:10.1007/0-387-34240-0>).

Depends R (>= 4.1.0)

License GPL (>= 2)

Encoding UTF-8

Language en-US

URL https://dickoa.gitlab.io/sondage, https://gitlab.com/dickoa/sondage

BugReports https://gitlab.com/dickoa/sondage/-/work_items

NeedsCompilation yes

Suggests knitr, rmarkdown, sampling, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

Author Ahmadou Dicko [aut, cre] (ORCID:
<https://orcid.org/0000-0002-9654-7582>),
Thomas Lumley [ctb],
Wilmer Prentius [ctb, cph] (Author of the BalancedSampling 2.0.6 (GPL >= 2) cube implementation from which src/cube.c was ported)

Maintainer Ahmadou Dicko <mail@ahmadoudicko.com>

Repository CRAN

Date/Publication 2026-11-13 10:20:02 UTC

Contents

sondage-package . . . 2
balanced_wor . . . 4
equal_prob_wor . . . 10
equal_prob_wr . . . 12
expected_hits . . . 13
hajek_jip . . . 14
he_jip . . . 15
inclusion_prob . . . 17
is_registered_method . . . 19
joint_expected_hits . . . 20
joint_inclusion_prob . . . 21
method_spec . . . 23
print.sondage_sample . . . 24
registered_methods . . . 25
register_method . . . 25
sampling_cov . . . 30
sondage_sample . . . 31
unequal_prob_wor . . . 33
unequal_prob_wr . . . 35
unregister_method . . . 37

Index **39**

`sondage-package` *sondage: Survey Sampling Algorithms*

Description

Fast implementations of survey sampling algorithms for single-stage probability sampling from finite populations, written in C. Provides equal probability methods (simple random sampling, systematic, Bernoulli), unequal probability methods (conditional Poisson / maximum entropy, Sampford, Brewer, systematic PPS, Pareto, sequential Poisson, Poisson, Chromy's sequential and systematic PPS with minimum replacement, multinomial), balanced sampling via the cube method,

and spatially balanced sampling via the local pivotal method and spatially correlated Poisson sampling. All sampling functions return design objects carrying sample indices, inclusion probabilities, and design metadata. Generics compute joint inclusion probabilities, pairwise expectations, and sampling covariances for variance estimation. Mostly based on algorithms from Tillé (2006, doi:10.1007/0387342400).

Unequal Probability Sampling

- `unequal_prob_wor()` - Without replacement: CPS (maximum entropy), Sampford, Brewer, systematic PPS, Poisson, SPS (sequential Poisson), Pareto
- `unequal_prob_wr()` - With replacement: Chromy and systematic PPS (both with minimum replacement), multinomial PPS

Equal Probability Sampling

- `equal_prob_wor()` - Without replacement: SRS, systematic, Bernoulli (random size)
- `equal_prob_wr()` - With replacement: SRS

Balanced Sampling

- `balanced_wor()` - Cube method (Deville & Tillé, 2004) for balanced sampling with unequal probabilities, with optional stratification (Chauvet, 2009), the local pivotal method 2 (Grafström, Lundström & Schelin, 2012), and spatially correlated Poisson sampling (Grafström, 2012) for spatially balanced, well-spread sampling

Design Queries

All sampling functions return objects of class `"sondage_sample"`. Use these generics to query the design:

- `inclusion_prob()` - First-order inclusion probabilities
- `expected_hits()` - Expected number of selections (WR)
- `joint_inclusion_prob()` - Joint inclusion probabilities (WOR)
- `joint_expected_hits()` - Pairwise expectations (WR)
- `sampling_cov()` - Sampling covariance matrix

Joint Probability Approximations

Standalone approximation functions for joint inclusion probabilities, useful as `joint_fn` arguments to `register_method()`:

- `he_jip()` - High-entropy approximation (Brewer & Donadio, 2003). Recommended default for most designs.
- `hajek_jip()` - Hajek (1964) approximation based on conditional Poisson (rejective) sampling theory. Simpler formula, slightly less accurate.

For without-replacement designs, the stored `pik` vector is the design-defining target. For most methods this equals the true first-order inclusion probabilities. For order-sampling methods (`"sps"`, `"pareto"`), the true probabilities are only approximately equal to the target.

Size-to-probability conversion

- `inclusion_prob()` - Compute inclusion probabilities from size measures (with capping for certainty selections)
- `expected_hits()` - Compute expected hits from size measures (simple proportional allocation, no capping)

Author(s)

Maintainer: Ahmadou Dicko <mail@ahmadoudicko.com> (ORCID)

Other contributors:

- Thomas Lumley <t.lumley@auckland.ac.nz> (contributor)
- Wilmer Prentius (contributor and copyright holder), author of the BalancedSampling 2.0.6 (GPL >= 2) cube implementation from which src/cube.c was ported

References

Tillé, Y. (2006). *Sampling Algorithms*. Springer Series in Statistics.

Chromy, J.R. (2009). Some generalizations of the Horvitz-Thompson estimator. *Proceedings of the Survey Research Methods Section, American Statistical Association*.

See Also

Useful links:

- https://dickoa.gitlab.io/sondage
- https://gitlab.com/dickoa/sondage
- Report bugs at https://gitlab.com/dickoa/sondage/-/work_items

---

balanced_wor *Balanced Sampling Without Replacement*

Description

Draws a balanced sample using the cube method (Deville & Tillé, 2004), or a spatially balanced (well-spread) sample using the local pivotal method (Grafström, Lundström & Schelin, 2012) or spatially correlated Poisson sampling (Grafström, 2012). A balanced sample satisfies (approximately) the balancing equations $\sum_{k \in S} x_k/\pi_k \approx \sum_{k \in U} x_k$ for each auxiliary variable x. A well-spread sample selects units that are far apart in the space spanned by the spreading variables.

Usage

```
balanced_wor(
  pik,
  aux = NULL,
  strata = NULL,
  spread = NULL,
  bounds = NULL,
  method = c("cube", "lpm2", "scps"),
  nrep = 1L,
  prn = NULL,
  ...
)
```

Arguments

pik	A numeric vector of inclusion probabilities (length N). sum(pik) must be an integer to floating-point accuracy. See unequal_prob_wor() for the exact-0/1 handling of boundary values. The target sample size, sum(pik), must be at least 1.
aux	An optional numeric matrix (N x p) of auxiliary balancing variables. Each column defines a balancing constraint. The sample size constraint is always included automatically and aux specifies additional variables to balance on. When NULL, only the sample size is balanced (equivalent to an unbalanced fixed-size design).
strata	An optional integer vector (length N) of stratum indicators (positive integers). Uses the stratified cube method (Chauvet & Tillé, 2006; Chauvet, 2009) to preserve within-stratum sample sizes while balancing on aux. Requires sum(pik) within each stratum to be close to an integer for exact sizes. If not, a warning is issued and fixed_size is set to FALSE.
spread	An optional numeric matrix (N x d) of spatial coordinates (or other spreading variables) for well-spread, spatially balanced sampling. Required by the built-in "lpm2" and "scps" methods, and supported by methods registered via register_method() with supports_spread = TRUE. The built-in "cube" method does not use it and will error. A non-matrix vector is treated as a single spreading variable.
bounds	An optional list with elements B, lower, and upper describing linear inequality constraints on the realized sample (Tripet & Tillé, 2026): B is a numeric matrix (N x q) whose columns are constraint variables, and the sample s is drawn so that $lower_j \leq \sum_{k \in s} B_{kj} \leq upper_j$ for every constraint j. -Inf / Inf entries make a constraint one-sided, and lower[j] == upper[j] enforces an exact equality. The starting probabilities must be feasible: lower <= colSums(B * pik) <= upper. Only supported by the built-in "cube" method. See **Inequality constraints** below.
method	The sampling method. "cube" (the default) balances on aux. "lpm2" (local pivotal method 2) and "scps" (spatially correlated Poisson sampling) spread on spread. The name of a balanced method added via register_method() is also accepted.

`nrep` Number of replicate samples (default 1). When `nrep > 1`, `$sample` holds a matrix (n x nrep) for fixed-size designs, or a list of integer vectors when within-stratum sizes are not exact.

`prn` Optional vector of permanent random numbers (length N, values in the open interval (0, 1)) for sample coordination. Supported by `"scps"` and by methods registered with `supports_prn = TRUE`. With `"scps"`, the sample is then a function of `pik`, `spread`, `prn` and the row order alone. See **Sample coordination** below. Cannot be combined with `nrep > 1`.

`...` Additional arguments passed to methods:

`eps` Boundary tolerance (default `1e-10`): decides when an updated *working* probability has numerically reached 0 or 1 (during the cube flight phase, or during the pivotal steps of `"lpm2"` or `"scps"`). It never reclassifies the input. Supplied `pik` inside `(0, eps]` or `[1 - eps, 1)` are rejected. Only `pik` of exactly 0 or exactly 1 enter the algorithm as already-resolved units.

`condition_aux` Logical. If `TRUE`, pre-conditions `aux` to improve numerical stability with ill-conditioned or collinear auxiliary variables (default `FALSE`). Each column is centered and scaled in the space the cube balances, `aux / pik`, so that the balancing equations are unchanged, and columns that depend on earlier ones or on the sample size constraint are dropped. The other columns keep their order.

`qr_tol` Tolerance for QR rank detection when `condition_aux = TRUE`. It must be one finite, non-negative number (default `sqrt(.Machine$double.eps)`).

Details

The cube method proceeds in two phases:

Flight phase Probabilities are moved toward 0 or 1 while maintaining all balancing constraints. Each step resolves at least one unit. Terminates when fewer than p+1 undecided units remain.

Landing phase Remaining undecided units are resolved by progressively relaxing balancing constraints, starting from the **last** column of `aux`. Users should order auxiliary variables by importance (most important first).

The sample size constraint is always placed first (never relaxed during landing). For stratified designs, within-stratum size constraints are also placed first.

Joint inclusion probabilities are approximated via the high-entropy approximation (Brewer & Donadio, 2003). It uses only `pik`, so it ignores `aux`, `strata`, and `bounds`, and it returns the same matrix as for an unconstrained design. It can give a positive probability to a pair the design excludes. For example, with strata of two units and one selection in each, two units of the same stratum never occur together. The marginal-defect warning of `joint_inclusion_prob()` does not detect this. For a stratified or strongly constrained design, prefer Monte Carlo estimates of the joint probabilities or a variance method suited to balanced sampling.

Value

An object of class `c("balanced", "unequal_prob", "wor", "sondage_sample")`. When `nrep = 1`, `$sample` is an integer vector of selected unit indices. When `nrep > 1`, `$sample` is a matrix (n x nrep) for fixed-size designs, or a list of integer vectors when `fixed_size` is `FALSE` (e.g., stratified with non-integer per-stratum sizes).

Inequality constraints

`bounds` implements the cube method with inequality constraints of Tripet & Tillé (2026). During the flight phase, steps are capped so every constraint stays feasible, and a constraint whose slack reaches zero becomes an equality from then on. The inclusion probabilities are respected exactly (`E(s) = pik`), unlike rejective procedures.

`B` applies to the realized sample directly (counts or raw sums), so it is **not** divided by `pik`. To bound a Horvitz-Thompson estimator, pass `x / pik` as the constraint column.

One application is controlled selection with bounding category counts to the integers adjacent to their expectation. With indicator columns `B` and `S <- colSums(B * pik)`, use `lower = floor(S)`, `upper = ceiling(S)`. Categories may overlap (e.g. row and column margins of a two-way control table, as in NAEP-style designs).

Integer-valued bound systems on partitions or two-way margins are satisfied exactly. For structures with no exact integer solution, some three-way controlled rounding problems, or continuous-valued constraints that end the flight phase tight against a boundary, bounds that provably block the landing phase are relaxed one at a time, with a warning, and `E(s) = pik` still holds. In other words, the bounds are guaranteed whenever no relaxation warning is raised.

The high-entropy approximation used by `joint_inclusion_prob()` is less accurate under tight bounds, which distort the design away from maximum entropy. Tripet & Tillé (2026) recommend Monte Carlo estimation of joint inclusion probabilities in that case.

Spatially balanced sampling (`lpm2` and `scps`)

`method = "lpm2"` implements the local pivotal method 2 of Grafström, Lundström & Schelin (2012). Repeatedly, a randomly chosen undecided unit and its nearest undecided neighbor in the `spread` space compete in a pivotal step (Deville & Tillé, 1998) that resolves at least one of them to 0 or 1 while preserving the inclusion probabilities exactly (`E(s) = pik`). Nearby units thereby tend to exclude each other, spreading the sample over the population. Nearest-neighbor distance ties (common on gridded coordinates) are broken uniformly at random.

Spreading variables should be on comparable scales, since nearness is plain Euclidean distance in the `spread` columns. Rescale them (e.g. with `scale()`) when they are not. `"lpm2"` and `"scps"` use `spread` only: they do not accept `aux`, `strata`, or `bounds`. To exactly balance covariate totals *and* spread spatially, register a method that supports both `aux` and `spread` (for example, a local-cube implementation). Running `"cube"` with coordinates in `aux` balances their totals but does not itself enforce spatial spread.

`method = "scps"` implements spatially correlated Poisson sampling with the maximal-weight strategy of Grafström (2012). At each step, a randomly chosen undecided unit is accepted or rejected using its current conditional probability. Its probability displacement is then distributed to the nearest undecided units, subject to feasibility bounds that keep every working probability in $[0, 1]$. Equal-distance units share weight as evenly as their bounds allow. Random selection of the step unit avoids dependence on input row order. With `prn`, units are visited in row order instead (see **Sample coordination** below).

Both spatial methods deliberately drive joint inclusion probabilities of nearby units toward zero, so the design is *not* high entropy and no joint-probability approximation is provided: `joint_inclusion_prob()` errors for these designs. Variance for well-spread samples is usually estimated with local-neighborhood estimators (e.g. Grafström & Schelin, 2014).

Both methods are *spread-only*: neither exactly balances the totals of `aux`. They are dispatched by `balanced_wor()` because "spatially balanced sampling" is the standard name for well-spread fixed-size designs, and because the same interface accommodates local-cube methods that combine exact balancing with spread. Capability metadata keeps the distinction explicit: both report `supports_aux = FALSE` and `supports_spread = TRUE` through `method_spec()`.

LPM2 finds nearest neighbors with a k-d tree (Friedman et al., 1977) when `spread` has at most eight columns, so a draw costs about O(N log N) time. Many units at the same distance, as with duplicated points, make the tree visit all of them, and a draw then approaches O(N^2). With more columns a tree prunes too little and LPM2 scans the undecided units instead, at O(N^2 * d) per draw. SCPS uses weighted quickselect to find the distance at which its maximal weights sum to one, avoiding a full sort of the remaining units at each step. Its expected cost is O(N^2 * d). Both implementations use O(N) workspace and store no distance matrix. Units at the cutoff distance share their weight equally, up to their feasibility bounds, and SCPS sorts that group only when one of its units cannot take an equal share.

Sample coordination

With `prn`, `method = "scps"` is the list-sequential correlated Poisson scheme of Bondesson & Thorburn (2008) with Grafström's maximal weights, coordinated through permanent random numbers as in Grafström & Matei (2018). Units are visited in row order. Unit k is selected when $u_k < \pi_k^{(k-1)}$, its current conditional probability, and its probability displacement goes only to undecided units later in the list. The inclusion probabilities are still `pik` exactly. As with the `"poisson"`, `"sps"` and `"pareto"` methods of `unequal_prob_wor()`, a small permanent random number makes selection more likely.

Drawing two samples with the same `prn` coordinates them positively (large overlap), and drawing the second with `1 - prn` coordinates them negatively (small overlap). The sample depends on the row order, so coordinated draws must use the same ordering of the frame, for example sorted by a permanent unit identifier.

Matei, Smith, Smeets & Klingwort (2023) replace the coordinates in `spread` by a measure of response burden, such as an indicator of highly burdened units or the inclusion probabilities, and draw the surveys with `prn` and `1 - prn`. Spreading on the burden measure keeps burdened units out of the same sample, and negative coordination keeps them out of both samples. This adapted SCP sampling is `balanced_wor(pik, spread = burden, method = "scps", prn = u)`.

References

Bondesson, L. and Thorburn, D. (2008). A list sequential sampling method suitable for real-time sampling. *Scandinavian Journal of Statistics*, 35(3), 466-483. doi:10.1111/j.14679469.2008.00596.x

Deville, J.C. and Tillé, Y. (1998). Unequal probability sampling without replacement through a splitting method. *Biometrika*, 85(1), 89-101.

Deville, J.C. and Tillé, Y. (2004). Efficient balanced sampling: the cube method. *Biometrika*, 91(4), 893-912.

Chauvet, G. and Tillé, Y. (2006). A fast algorithm for balanced sampling. *Computational Statistics*, 21(1), 53-62.

Chauvet, G. (2009). Stratified balanced sampling. *Survey Methodology*, 35, 115-119.

Friedman, J.H., Bentley, J.L. and Finkel, R.A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3), 209-226. doi:10.1145/355744.355745

Grafström, A., Lundström, N.L.P. and Schelin, L. (2012). Spatially balanced sampling through the pivotal method. *Biometrics*, 68(2), 514-520. doi:10.1111/j.15410420.2011.01699.x

Grafström, A. (2012). Spatially correlated Poisson sampling. *Journal of Statistical Planning and Inference*, 142(1), 139-147. doi:10.1016/j.jspi.2011.07.003

Grafström, A. and Matei, A. (2018). Coordination of spatially balanced samples. *Survey Methodology*, 44(2), 215-238.

Grafström, A. and Schelin, L. (2014). How to select representative samples. *Scandinavian Journal of Statistics*, 41(2), 277-290. doi:10.1111/sjos.12016

Matei, A., Smith, P.A., Smeets, M.J.E. and Klingwort, J. (2023). Targetted double control of burden in multiple surveys. *Survey Methodology*, 49(2), 363-384.

Tripet, A. and Tillé, Y. (2026). Balanced sampling with inequalities: application to category bounding, matrix rounding, and spread sampling. *Journal of the American Statistical Association*, 121(553), 796-806. doi:10.1080/01621459.2025.2550667

See Also

`unequal_prob_wor()` for unbalanced designs, `inclusion_prob()` to compute inclusion probabilities from size measures.

Examples

```
# Unequal probability balanced sample
pik <- c(0.3, 0.6, 0.2, 0.4, 0.5)
x <- matrix(c(10, 20, 15, 25, 30))
set.seed(1)
s <- balanced_wor(pik, aux = x)
s$sample

# Check balancing: HT estimate of aux totals vs population totals
colSums(x[s$sample, , drop = FALSE] / pik[s$sample]) - colSums(x)

# Stratified balanced sample
N <- 20
pik <- rep(0.4, N)
x <- matrix(as.double(1:N), ncol = 1)
strata <- rep(1:4, each = 5)
set.seed(1)
s <- balanced_wor(pik, aux = x, strata = strata)
s$sample

# Controlled selection: bound category counts to the integers
# adjacent to their expectation (floor/ceil)
pik <- rep(0.5, 12)  # n = 6
groups <- rep(c("a", "b", "c"), each = 4)
B <- sapply(unique(groups), function(g) as.double(groups == g))
S <- colSums(B * pik)  # 2, 2, 2 per category
```

```
set.seed(1)
s <- balanced_wor(
  pik,
  bounds = list(B = B, lower = floor(S), upper = ceiling(S))
)
table(groups[s$sample])  # exactly 2 per category

# Spatially balanced (well-spread) sample: local pivotal method 2
N <- 100
coords <- cbind(runif(N), runif(N))
pik <- rep(0.1, N)
set.seed(1)
s <- balanced_wor(pik, spread = coords, method = "lpm2")
s$sample

# Spatially correlated Poisson sampling uses the same spread contract
set.seed(1)
s_scps <- balanced_wor(pik, spread = coords, method = "scps")
s_scps$sample

# Two surveys negatively coordinated with permanent random numbers,
# spread on an indicator of highly burdened units (adapted SCP sampling)
burdened <- as.double(seq_len(N) %in% c(4, 12, 21, 22, 32, 44))
pik1 <- inclusion_prob(runif(N, 1, 5), 10)
pik2 <- inclusion_prob(runif(N, 1, 5), 6)
u <- runif(N)
s1 <- balanced_wor(pik1, spread = burdened, method = "scps", prn = u)
s2 <- balanced_wor(pik2, spread = burdened, method = "scps", prn = 1 - u)
intersect(s1$sample, s2$sample)
```

`equal_prob_wor` *Equal Probability Sampling Without Replacement*

Description

Draws a sample with equal inclusion probabilities, without replacement.

Usage

```
equal_prob_wor(
  N,
  n,
  method = c("srs", "systematic", "bernoulli"),
  nrep = 1L,
  prn = NULL,
  ...
)
```

Arguments

N	Population size (positive integer).
n	Sample size. For "srs" and "systematic", a non-negative integer not exceeding N (exact realized size). For "bernoulli", a non-negative number (not required to be an integer) interpreted as the expected sample size, and p = n / N is used as the selection probability.
method	The sampling method: "srs" Simple Random Sampling. Each possible sample of size n has equal probability. Fixed sample size. "systematic" Systematic sampling with interval k = N/n. A random start is drawn from (0, k]. Fixed sample size. Implicit stratification based on unit ordering. "bernoulli" Bernoulli sampling. Each unit selected independently with probability p = n/N. Random sample size. Note: the realized sample size varies across draws.
nrep	Number of replicate samples (default 1). When nrep > 1, $sample holds a matrix (fixed-size) or list (random-size) of all replicates. The design object and all generics remain usable.
prn	Optional vector of permanent random numbers (length N, values in the open interval (0, 1)) for sample coordination. Only supported by "bernoulli" method. Cannot be used with nrep > 1 (identical PRN would produce identical replicates).
...	Reserved for extensions. Built-in methods currently take no additional arguments.

Value

An object of class c("equal_prob", "wor", "sondage_sample"). When nrep = 1, $sample is an integer vector. When nrep > 1, $sample is a matrix (n x nrep) for fixed-size methods, or a list of integer vectors of varying lengths for "bernoulli". $n is an integer for "srs" and "systematic" (realized size) and a double for "bernoulli" (expected size). See sondage_sample.

See Also

equal_prob_wr() for with-replacement designs, unequal_prob_wor() for unequal probability designs.

Examples

```
set.seed(1)
s <- equal_prob_wor(10, 3)
s$sample

# Systematic sampling
s <- equal_prob_wor(12, 3, method = "systematic")
s$sample
```

```
# Bernoulli sampling (random size, expected n = 30)
s <- equal_prob_wor(100, 30, method = "bernoulli")
length(s$sample)
```

`equal_prob_wr` *Equal Probability Sampling With Replacement*

Description

Draws a simple random sample with replacement.

Usage

```
equal_prob_wr(N, n, method = "srs", nrep = 1L, prn = NULL, ...)
```

Arguments

`N`	Population size (positive integer).
`n`	Sample size (non-negative integer).
`method`	The sampling method. Currently only `"srs"`.
`nrep`	Number of replicate samples (default 1).
`prn`	Optional vector of permanent random numbers for sample coordination. No equal-probability WR method currently supports `prn`; supplying it is an error.
`...`	Reserved for extensions. Built-in methods currently take no additional arguments.

Value

An object of class `c("equal_prob", "wr", "sondage_sample")`. When `nrep = 1`, `$sample` is an integer vector and `$hits` is an integer vector. When `nrep > 1`, `$sample` is a matrix (n x nrep) and `$hits` is a matrix (N x nrep).

See Also

`equal_prob_wor()` for without-replacement designs, `unequal_prob_wr()` for unequal probability designs.

Examples

```
set.seed(1)
s <- equal_prob_wr(10, 3)
s$sample
s$hits
```

`expected_hits` *Expected Hits*

Description

Compute expected hits from a size measure, or extract them from a with-replacement design object.

Usage

```
expected_hits(x, ...)

## Default S3 method:
expected_hits(x, n, ...)

## S3 method for class 'wr'
expected_hits(x, ...)

## S3 method for class 'wor'
expected_hits(x, ...)
```

Arguments

`x`	A numeric vector of positive size measures, or a with-replacement design object (class `"wr"`).
`...`	Reserved for methods. The methods provided by sondage currently take no additional arguments.
`n`	The desired sample size. Required when `x` is a numeric vector; it must not be supplied when `x` is a design object.

Value

A numeric vector of expected hits. Values can exceed 1 for with-replacement designs.

See Also

`inclusion_prob()` for the without-replacement analogue, `unequal_prob_wr()` for sampling with expected hits.

Examples

```
# From size measures
x <- c(40, 80, 50, 60, 70)
hits <- expected_hits(x, n = 3)
sum(hits)  # 3

# From a design object
s <- unequal_prob_wr(hits, method = "chromy")
expected_hits(s)
```

hajek_jip *Hajek Approximation for Joint Inclusion Probabilities*

Description

Computes the joint inclusion probability matrix using the Hajek (1964) approximation based on conditional Poisson (rejective) sampling theory:

$$\pi_{ij} \approx \pi_i \pi_j \left[1 - \frac{(1-\pi_i)(1-\pi_j)}{D}\right]$$

where $D = \sum_k \pi_k(1-\pi_k)$.

Usage

```
hajek_jip(pik, sample_idx = NULL, eps = 1e-06, ...)
```

Arguments

pik	Numeric vector of inclusion probabilities ($0 \leq \pi_k \leq 1$).
sample_idx	Unique integer vector of 1-based indices for the sampled units, or NULL (default) for the full population matrix. When non-NULL, returns the submatrix for those units only, without allocating the full N x N matrix. The returned matrix has at most 10,000 rows, as for joint_inclusion_prob(), so a population above that size needs sample_idx.
eps	Boundary tolerance (default 1e-6). Units with $\pi_k \geq 1-\varepsilon$ are treated as certainty selections, and units with $\pi_k \leq \varepsilon$ are treated as zero.
...	Additional arguments (ignored). Present so that the function matches the joint_fn signature required by register_method(). sampled_only is refused, since it belongs to joint_inclusion_prob(). Use sample_idx here instead.

Details

The Hajek approximation is simpler and computationally lighter than the high-entropy approximation (he_jip()), but generally slightly less accurate. It is derived from the asymptotic theory of rejective (conditional Poisson) sampling, where the design is obtained by conditioning independent Bernoulli trials (Poisson sampling) on the total sample size.

The formula is valid for any high-entropy design, but is most accurate when the design is close to rejective sampling. For CPS, the maximum-entropy design, and for other high-entropy designs such as Sampford's, he_jip() tends to give tighter results. Both CPS and Sampford have exact joint probabilities in joint_inclusion_prob(). In practice, both approximations agree closely for moderate to large populations with well-spread inclusion probabilities.

Like he_jip(), this function matches the joint_fn signature required by register_method():

```
register_method("my_method", sample_fn = my_fn, joint_fn = hajek_jip)
```

Value

A symmetric matrix of joint inclusion probabilities: N x N when `sample_idx` is `NULL`, or `length(sample_idx)` x `length(sample_idx)` otherwise. Diagonal entries are π_i. Rows and columns are named after `names(pik)`, as in `joint_inclusion_prob()`. A submatrix of unnamed `pik` is labeled with the indices in `sample_idx`.

Properties

- Symmetric: $\pi_{ij} = \pi_{ji}$
- Diagonal: $\pi_{ii} = \pi_i$ (set directly)
- Bounded: $0 \leq \pi_{ij} \leq \min(\pi_i, \pi_j)$ (clamped)
- Marginal defect: $|\sum_{j \neq i} \pi_{ij} - (n-1)\pi_i| = O(1/N)$ for well-spread π_k

References

Hajek, J. (1964). Asymptotic theory of rejective sampling with varying probabilities from a finite population. *Annals of Mathematical Statistics*, 35(4), 1491–1523.

See Also

`he_jip()` for the high-entropy approximation, `joint_inclusion_prob()` for design-based dispatch, `register_method()` for custom method registration.

Examples

```
pik <- inclusion_prob(c(2, 3, 4, 5, 6, 7, 8, 9), n = 4)

# Full N x N matrix
pikl <- hajek_jip(pik)
round(pikl, 4)

# Compare with high-entropy approximation
he <- he_jip(pik)
max(abs(pikl - he))
```

---

he_jip *High-Entropy Approximation for Joint Inclusion Probabilities*

Description

Computes the joint inclusion probability matrix using the high-entropy approximation of Brewer & Donadio (2003, eq. 18):

$$\pi_{ij} \approx \pi_i \pi_j \frac{c_i + c_j}{2}$$

where $c_k = \frac{n-1}{n - \frac{2n-1}{n-1}\pi_k + \frac{\sum_\ell \pi_\ell^2}{n-1}}$ and $n = \sum_k \pi_k$.

Usage

```
he_jip(pik, sample_idx = NULL, eps = 1e-06, ...)
```

Arguments

pik	Numeric vector of inclusion probabilities ($0 \leq \pi_k \leq 1$).
sample_idx	Unique integer vector of 1-based indices for the sampled units, or NULL (default) for the full population matrix. When non-NULL, returns the submatrix for those units only, without allocating the full N x N matrix. The returned matrix has at most 10,000 rows, as for joint_inclusion_prob(), so a population above that size needs sample_idx.
eps	Boundary tolerance (default 1e-6). Units with $\pi_k \geq 1-\varepsilon$ are treated as certainty selections, and units with $\pi_k \leq \varepsilon$ are treated as zero.
...	Additional arguments (ignored). Present so that the function matches the joint_fn signature required by register_method(). sampled_only is refused, since it belongs to joint_inclusion_prob(). Use sample_idx here instead.

Details

The high-entropy (HE) approximation is the recommended default for designs that are close to maximum entropy, which includes most common unequal-probability without-replacement designs: Brewer, Sampford, Tillé, SPS, Pareto, cube, and CPS itself.

The approximation guarantees symmetry, $0 \leq \pi_{ij} \leq \min(\pi_i, \pi_j)$, and correct diagonal ($\pi_{ii} = \pi_i$), but does not exactly satisfy the marginal identity $\sum_{j \neq i} \pi_{ij} = (n-1)\pi_i$. The defect is typically small but grows with skewed π_k and small n.

Internally, this calls the same C implementation used by joint_inclusion_prob() for Brewer, SPS, Pareto, and cube methods. Exported so that custom methods registered via register_method() can use it directly as their joint_fn:

```
register_method("my_method", sample_fn = my_fn, joint_fn = he_jip)
```

For a lighter alternative based on conditional Poisson theory, see hajek_jip().

Value

A symmetric matrix of joint inclusion probabilities: N x N when sample_idx is NULL, or length(sample_idx) x length(sample_idx) otherwise. Diagonal entries are π_i. Rows and columns are named after names(pik), as in joint_inclusion_prob(). A submatrix of unnamed pik is labeled with the indices in sample_idx.

References

Brewer, K.R.W. and Donadio, M.E. (2003). The high entropy variance of the Horvitz-Thompson estimator. *Survey Methodology*, 29(2), 189–196.

See Also

hajek_jip() for the Hajek approximation, joint_inclusion_prob() for design-based dispatch, register_method() for custom method registration.

Examples

```
pik <- inclusion_prob(c(2, 3, 4, 5, 6, 7, 8, 9), n = 4)

# Full N x N matrix
pikl <- he_jip(pik)
round(pikl, 4)

# Submatrix for specific units
he_jip(pik, sample_idx = c(1, 3, 5))

# Use as joint_fn in register_method()
register_method("my_method", sample_fn = function(pik, n, prn, ...) {
  sample.int(length(pik), n, prob = pik)
}, joint_fn = he_jip)
unregister_method("my_method")
```

---

inclusion_prob *Inclusion Probabilities*

Description

Compute inclusion probabilities from a size measure, or extract them from a without-replacement design object.

Usage

```
inclusion_prob(x, ...)

## S3 method for class 'wor'
inclusion_prob(x, ...)

## S3 method for class 'wr'
inclusion_prob(x, ...)

## Default S3 method:
inclusion_prob(x, n, ...)
```

Arguments

x	A numeric vector of positive size measures, or a without-replacement design object (class "wor").

`...` Reserved for methods. The methods provided by sondage currently take no additional arguments.

`n` The desired sample size, or the expected sample size of a random-size design, which may be fractional. Required when `x` is a numeric vector. It must not be supplied when `x` is a design object.

Details

When `x` is a numeric vector and `n` is provided, computes inclusion probabilities via iterative capping: units with $\pi_k \geq 1$ are set to 1 (certainty selections) and the remaining probabilities are recomputed with reduced n. The result sums to exactly n. This differs from `expected_hits()`, which does simple proportional allocation without capping. Negative values in `x` are treated as zero (with a warning).

`n` may be any real number from 0 to the number of units with positive size. A fractional `n` is the expected size of a random-size design, such as Poisson sampling. Fixed-size samplers, `unequal_prob_wor()` among them, require probabilities that sum to a whole number and refuse others.

Value

A numeric vector of inclusion probabilities. When applied to a design object, returns the stored `pik` vector. For most methods this equals the true first-order inclusion probabilities. For order-sampling methods (`sps`, `pareto`), it is the target used to define the design. The true probabilities are approximately equal to it and converge as N grows. Names on a numeric `x` are preserved.

With-replacement designs

`inclusion_prob()` applies only to without-replacement (`wor`) designs. Calling it on a with-replacement design object (e.g. from `equal_prob_wr()` or `unequal_prob_wr()`) is an error; use `expected_hits()` to obtain the expected number of selections.

See Also

`expected_hits()` for the with-replacement analogue, `unequal_prob_wor()` for sampling with these probabilities.

Examples

```
# From size measures
size <- c(10, 20, 30, 40)
pik <- inclusion_prob(size, n = 2)
sum(pik)  # 2

# From a design object
s <- unequal_prob_wor(pik, method = "cps")
inclusion_prob(s)

# With certainty selections (large units)
size <- c(1, 1, 1, 100)
pik <- inclusion_prob(size, n = 2)
```

```
pik  # Unit 4 gets probability 1

# The expected size of a random-size design need not be a whole number
pik <- inclusion_prob(rep(1, 10), n = 2.6)
sum(pik)  # 2.6
unequal_prob_wor(pik, method = "poisson")       # valid
try(unequal_prob_wor(pik, method = "sampford")) # refused: fixed size
```

`is_registered_method` *Check Whether a Method Is Registered*

Description

Check Whether a Method Is Registered

Usage

```
is_registered_method(name)
```

Arguments

`name`	Method name (character string).

Value

`TRUE` if the method has been registered via `register_method()`, `FALSE` otherwise.

See Also

`register_method()`

Examples

```
is_registered_method("foo")
```

`joint_expected_hits` *Joint Expected Hits*

Description

Computes the matrix of pairwise expectations $E(n_i n_j)$ for a with-replacement sampling design, where n_k is the number of times unit k is selected.

Usage

```
joint_expected_hits(x, ...)

## S3 method for class 'wr'
joint_expected_hits(x, sampled_only = FALSE, nsim = 10000L, ...)

## S3 method for class 'wor'
joint_expected_hits(x, ...)

## Default S3 method:
joint_expected_hits(x, ...)
```

Arguments

`x`	A with-replacement design object (class `"wr"`).
`...`	Additional arguments passed to methods (e.g., `nsim` for simulation-based methods).
`sampled_only`	If `TRUE`, return only the submatrix for units selected at least once (requires `nrep = 1`). Useful when N is large but the number of distinct selected units is manageable. Default `FALSE`.
`nsim`	Number of simulations for Chromy's pairwise expectations (default 10000).

Details

The computation depends on the method:

Exact `multinomial` ($E(n_i n_j) = n(n-1)p_i p_j$), `srs` ($E(n_i n_j) = n(n-1)/N^2$), and `systematic_mr`, from the whole part of each unit's expected hits and the overlap of the arcs the units cover on the systematic walk. For `systematic_mr`, two units below one expected hit whose arcs do not overlap are never selected together, and their entry is exactly zero.

Simulation `chromy`: estimated by Monte Carlo (controlled by `nsim`, default 10 000).

When `sampled_only = TRUE`, only the submatrix for units with `hits > 0` is returned. All methods compute it directly without allocating the full N x N matrix.

For `method = "chromy"`, `sampled_only = TRUE` reduces memory (the n_s x n_s accumulator replaces an N x N one) but not simulation cost. Each of `nsim` draws still covers the full population.

Value

A symmetric N x N matrix (or n_s x n_s if `sampled_only = TRUE`, where n_s is the number of distinct selected units). Diagonal entries are $E(n_i^2)$ and off-diagonal entries are $E(n_i n_j)$. When the stored probability vector is named, those names are used as row and column names. An otherwise unnamed sampled-only matrix is labeled with the selected population indices.

See Also

`joint_inclusion_prob()` for the without-replacement analogue, `sampling_cov()` for the covariance matrix.

Examples

```
x <- c(40, 80, 50, 60, 70)
hits <- expected_hits(x, n = 3)
s <- unequal_prob_wr(hits, method = "chromy")
joint_expected_hits(s)

# Only the submatrix for selected units
joint_expected_hits(s, sampled_only = TRUE)
```

---

joint_inclusion_prob *Joint Inclusion Probabilities*

Description

Computes the matrix of joint inclusion probabilities $\pi_{ij} = P(i \in S \text{ and } j \in S)$ for a without-replacement sampling design.

Usage

```
joint_inclusion_prob(x, ...)

## S3 method for class 'wor'
joint_inclusion_prob(x, sampled_only = FALSE, eps = 1e-06, ...)

## S3 method for class 'wr'
joint_inclusion_prob(x, ...)

## Default S3 method:
joint_inclusion_prob(x, ...)
```

Arguments

x	A without-replacement design object (class "wor").
...	Additional arguments passed to the joint_fn of methods registered via register_method().
sampled_only	If TRUE, return only the n x n submatrix for the sampled units (requires nrep = 1). Useful when N is large but n is manageable. Default FALSE.
eps	High-entropy approximation boundary tolerance (default 1e-6), used only by the HE-approximate methods (brewer, sps, pareto, cube). See he_jip(). The exact cps, sampford, and systematic methods ignore it and treat only exact 0/1 as excluded/certainty units.

Details

The computation depends on the method stored in the design object:

Exact cps (probability-domain Poisson-binomial recurrences), sampford (elementary-symmetric recurrences), systematic (circular interval overlap), poisson ($\pi_{ij} = \pi_i \pi_j$), srs, and bernoulli.

Approximate brewer, sps, pareto, and cube use the high-entropy approximation (Brewer & Donadio, 2003, eq. 18): $\pi_{ij} \approx \pi_i \pi_j (c_i + c_j)/2$. This guarantees symmetry, $0 \leq \pi_{ij} \leq \min(\pi_i, \pi_j)$, and correct diagonal, but **not** the marginal identity $\sum_{j \neq i} \pi_{ij} = (n-1)\pi_i$. The defect is typically small but grows with skewed π_k and small n. A warning is issued when it exceeds 5% of n. For cube, the approximation ignores strata and balancing constraints, see balanced_wor(). Use method = "cps" when exact second-order probabilities are needed. For sps and pareto, the approximation uses the stored target pik, not the exact finite-population probabilities.

For **systematic PPS**, some off-diagonal entries may be exactly zero (pairs that never co-occur). See sampling_cov().

When sampled_only = TRUE, only the n x n submatrix for sampled units is returned. All methods compute this directly without allocating the full N x N matrix, so large N is feasible (e.g. N = 50 000 with n = 200). The marginal defect diagnostic is skipped because the row-sum identity only holds for the full matrix.

For cps, the fixed-point calibration used to produce the joint probabilities can emit a "CPS calibration did not reach tolerance" warning for pik values very close to 0 or 1. The realized joint probabilities differ from their exact values by up to the reported max_diff, see unequal_prob_wor() for context and remediation.

Value

A symmetric N x N matrix (or n x n if sampled_only = TRUE) of joint inclusion probabilities. Diagonal entries are the first-order inclusion probabilities π_i. When the stored probability vector is named, those names are used as row and column names. An otherwise unnamed sampled-only matrix is labeled with the sampled population indices.

See Also

joint_expected_hits() for the with-replacement analogue, sampling_cov() for the covariance matrix.

Examples

```
pik <- c(0.2, 0.4, 0.6, 0.8)
s <- unequal_prob_wor(pik, method = "cps")
joint_inclusion_prob(s)

# Only the n x n submatrix for sampled units
joint_inclusion_prob(s, sampled_only = TRUE)
```

---

method_spec *Query Method Metadata*

Description

Return the capabilities of a sampling method. Works for built-in methods and methods added via `register_method()`.

Usage

```
method_spec(name, dispatcher = NULL)
```

Arguments

`name`	Method name (character string), as used by the sondage dispatchers (e.g. `"brewer"`, `"cube"`, `"srs"`).
`dispatcher`	Optional sampling entry point. One of `"equal_prob_wor"`, `"equal_prob_wr"`, `"unequal_prob_wor"`, `"unequal_prob_wr"`, or `"balanced_wor"`. Required when `name` is available through more than one entry point, as with `"srs"` and `"systematic"`. Values must match exactly.

Value

A list with elements `dispatcher` (the sampling entry point), `type` (`"wor"`, `"wr"`, or `"balanced"`), `fixed_size` (logical), `variance_family` (one of `"srs"`, `"pps_brewer"`, `"poisson"`, `"wr"`, `"unsupported"`, or `NULL` for a registered method that did not declare one, as `register_method()` describes), `supports_prn` (logical), `supports_aux` (logical), `supports_strata` (logical), `supports_spread` (logical), and `probabilities` (where the method sits in the first-order probability taxonomy: `"exact"` for every built-in except `"sps"` and `"pareto"`, which honor `pik` to a documented approximation and report `"approximate"`; for a registered method the declared tier, `"unknown"` when the author did not establish one), plus `sample_fn` and `joint_fn` (the registered implementation functions for a registered method, `NULL` for built-ins, whose implementations are internal dispatch paths). Returns `NULL` if the method is unknown. The aux/strata/spread capabilities are only `TRUE` for balanced methods. An ambiguous built-in name without `dispatcher` is an error rather than silently selecting one variant.

See Also

`register_method()`, `registered_methods()`

Examples

```
method_spec("brewer")
method_spec("cube")
method_spec("srs", dispatcher = "equal_prob_wr")
method_spec("systematic", dispatcher = "equal_prob_wor")
method_spec("nonexistent")
```

`print.sondage_sample` *Print Sampling Design Objects*

Description

Print Sampling Design Objects

Usage

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

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

Arguments

`x`	A sampling design object of class `"sondage_sample"`.
`...`	Further arguments passed to or from other methods.

Value

`invisible(x)`.

See Also

sondage_sample for the documented object structure and fields.

registered_methods *List Registered Custom Methods*

Description

List Registered Custom Methods

Usage

```
registered_methods()
```

Value

A character vector of registered method names (empty if none).

See Also

`register_method()`

Examples

```
registered_methods()
```

register_method *Register a Custom Sampling Method*

Description

Register a user-defined sampling method so it can be used through `unequal_prob_wor()`, `unequal_prob_wr()`, or `balanced_wor()` and their associated generics.

Usage

```
register_method(
  name,
  type = c("wor", "wr", "balanced"),
  sample_fn,
  joint_fn = NULL,
  fixed_size = TRUE,
  variance_family = NULL,
  supports_prn = NULL,
  supports_aux = NULL,
  supports_strata = NULL,
  supports_spread = NULL,
  probabilities = c("unknown", "exact", "approximate")
)
```

Arguments

name	A unique method name (character string). Must not collide with a built-in method name.
type	"wor" (without replacement), "wr" (with replacement), or "balanced" (balanced without replacement, dispatched through balanced_wor()).
sample_fn	A function that draws a sample. See **Contracts** below.
joint_fn	An optional function that computes joint inclusion probabilities (WOR) or joint expected hits (WR). If NULL, joint_inclusion_prob() / joint_expected_hits() will error for this method.
fixed_size	Does this method always produce exactly n units? Must be TRUE for type = "wr".
variance_family	Optional declaration of how design-based variance should be estimated for this method, for downstream packages that export designs for variance estimation. One of "srs", "pps_brewer", "poisson", "wr", or "unsupported". See **Variance families** below. NULL (the default) means undeclared: consumers fall back on inferring a treatment from type and fixed_size.
supports_prn	Does this method support permanent random numbers for sample coordination? TRUE or FALSE, for every method type. NULL (the default) resolves to FALSE.
supports_aux	Does this method use auxiliary balancing variables? Set to TRUE or FALSE for type = "balanced". NULL (the default) resolves to TRUE for balanced methods and FALSE otherwise. Set it to FALSE for spread-only (spatially balanced) methods such as the local pivotal method, so that passing aux to balanced_wor() is an error instead of being silently ignored. Leave NULL for "wor" and "wr" methods.
supports_strata	Does this method support stratified balanced sampling? Set to TRUE or FALSE for type = "balanced". NULL (the default) resolves to FALSE. When FALSE, passing strata to balanced_wor() with this method is an error, and sample_fn does not need a strata argument. Leave NULL for "wor" and "wr" methods.
supports_spread	Does this method support spatial spreading (well-spread / spatially balanced sampling)? Set to TRUE or FALSE for type = "balanced". NULL (the default) resolves to FALSE. When FALSE, passing spread to balanced_wor() with this method is an error, and sample_fn does not need a spread argument. When TRUE, the method may use spread but is still called without it, so a sample_fn that cannot work without spread should check for it. Leave NULL for "wor" and "wr" methods.
probabilities	Where the method sits in the first-order probability taxonomy, for downstream packages that weight or record per-unit selection probabilities: • "exact": the true first-order inclusion probabilities (types "wor" and "balanced") or expected hits (type "wr") equal the pik or hits vector passed to sample_fn, as for Sampford or the cube method. • "approximate": pik or hits is the method's first-order target, achieved up to a documented approximation, as for Pareto or sequential Poisson order sampling. Design weights 1/pik remain standard practice.

- `"unknown"` (the default): `pik` or `hits` is an input weight or preference only, so the corresponding first-order quantities are not known. The toy WOR sampler in the examples is such a method: successive sampling with `prob = pik` does not yield inclusion probabilities equal to `pik`, so its design weights `1/pik` would be systematically biased, not merely noisy. (The same draw *with* replacement does honor expected hits, so a multinomial-style `type = "wr"` method declares `"exact"`.)

The default is deliberately strict: if you have not established which tier your method is in, its selection probabilities are unknown, and downstream packages that weight estimation by `1/pik` may refuse to draw with it rather than produce biased weights. Sampling through sondage itself is unaffected: the declaration describes the method, it never disables it. Like `variance_family`, the declaration is an assertion by the method author that sondage cannot verify. The package vignette shows how to check a first-order contract by simulation.

Details

Registrations last for the current R session. Registering an existing custom method name is an error. Call `unregister_method()` first when deliberately replacing a method.

Value

Invisible `NULL`, called for its side effect.

Contracts

`sample_fn(pik, n = NULL, prn = NULL, ...)` (type `"wor"`)

`pik` Inclusion probabilities, numeric vector of length N.

`n` Sample size: `round(sum(pik))` for a fixed-size method, and the expected size `sum(pik)`, a double, for a method registered with `fixed_size = FALSE`.

`prn` Permanent random numbers (numeric vector length N, values in (0,1)), or `NULL`. Supplying `prn` is an error when the method is registered with `supports_prn = FALSE`, and `sample_fn` is not called.

Returns Integer vector of selected unit indices (1-based). Indices are distinct and have length `n` for fixed-size methods, or varying length for random-size methods. The dispatcher validates the type, range, size, and replacement rules before constructing the sample object.

`sample_fn(hits, n = NULL, prn = NULL, ...)` (type `"wr"`)

`hits` Expected hits, numeric vector of length N.

`n` Target sample size, equal to `round(sum(hits))`.

`prn` Permanent random numbers (numeric vector length N, values in (0,1)), or `NULL`. Supplying `prn` is an error when the method is registered with `supports_prn = FALSE`, and `sample_fn` is not called.

Returns Integer vector of `n` selected unit indices (1-based), with possible repeats. The dispatcher validates the type, range, size, and replacement rules before constructing the sample object.

`sample_fn(pik, n = NULL, aux = NULL, ...)` (type `"balanced"`)

`pik` Inclusion probabilities, numeric vector of length N.

`n` Target sample size (integer when `fixed_size`, otherwise `sum(pik)`).

`aux` Auxiliary balancing matrix (N x p, double), or `NULL`. Passed through as supplied to `balanced_wor()` after validation. The sample-size constraint is **not** prepended, so add it yourself if your algorithm needs it (e.g. `cbind(pik, aux)`). Methods registered with `supports_aux = FALSE` always receive `aux = NULL`.

`strata` Only when registered with `supports_strata = TRUE` and the caller supplies `strata`: an integer vector (length N) of dense stratum labels `1:H`. Declare it as `strata = NULL` in your function signature.

`spread` Only when registered with `supports_spread = TRUE` and the caller supplies `spread`: a numeric matrix (N x d, double) of spatial coordinates (or other spreading variables). Declare it as `spread = NULL` in your function signature.

`prn` Only when registered with `supports_prn = TRUE` and the caller supplies `prn`: permanent random numbers (numeric vector length N, values in (0,1)). Declare it as `prn = NULL` in your function signature.

Returns Integer vector of distinct selected unit indices (1-based). The dispatcher validates the returned indices.

`joint_fn(pik, sample_idx = NULL, ...)` (optional, types `"wor"` and `"balanced"`)

`pik` Inclusion probabilities, numeric vector of length N.

`eps` If the function explicitly declares an `eps` formal, it receives the value supplied to `joint_inclusion_prob()`.

`sample_idx` When non-NULL, an integer vector of sampled unit indices. Return only the submatrix for these units.

Returns Symmetric matrix of joint inclusion probabilities (N x N when `sample_idx` is NULL, `length(sample_idx)` x `length(sample_idx)` otherwise). The dispatcher validates that the matrix has the required dimensions and contains finite, symmetric numeric values.

`joint_fn(hits, sample_idx = NULL, ...)` (optional, type `"wr"`)

`hits` Expected hits, numeric vector of length N, renormalized to sum exactly to the integer sample size `n`.

`nsim` If the function explicitly declares an `nsim` formal, it receives the value supplied to `joint_expected_hits()`.

`sample_idx` When non-NULL, an integer vector of distinct sampled unit indices. Return only the submatrix for these units.

Returns Symmetric matrix of joint expected hits (N x N when `sample_idx` is NULL, `length(sample_idx)` x `length(sample_idx)` otherwise). The dispatcher validates that the matrix has the required dimensions and contains finite, symmetric numeric values.

Variance families

`variance_family` names the estimator treatment a variance consumer (such as a survey-export package) should apply to samples drawn with this method. Selection metadata alone cannot determine it: a fixed-size WOR method may need Brewer's unequal-probability approximation or an SRS-style variance, and for a random-size WOR method no safe inference exists at all, because a Poisson-type method (independent selections) and a correlated random-size scheme need different estimators.

`"srs"` Equal-probability fixed-size WOR. SRS-style variance with a finite population correction. Requires `fixed_size = TRUE`.

`"pps_brewer"` Fixed-size unequal-probability WOR. Brewer's approximation from the marginal inclusion probabilities. Requires `fixed_size = TRUE`.

`"poisson"` Random-size WOR with **independent** selections (Poisson-type). Exact Poisson linearization. Requires `type = "wor"` and `fixed_size = FALSE`.

`"wr"` With-replacement (or minimum-replacement) selection. Hansen-Hurwitz variance, no finite population correction. Requires `type = "wr"`. It is design-unbiased for independent draws such as `"multinomial"`, and an approximation, often conservative, for minimum-replacement methods such as `"chromy"` and `"systematic_mr"`, whose hits are dependent.

`"unsupported"` No linearization treatment is valid. Consumers should refuse to linearize and point to replicate methods instead. The correct declaration for correlated random-size schemes. Always allowed.

Balanced methods allow only `"pps_brewer"` or `"unsupported"`: balancing constraints couple selections across units, so `"poisson"` can never hold for them.

The declaration is an assertion by the method author, not something sondage can verify. `"poisson"` in particular asserts that units are selected independently, and a wrong declaration produces silently wrong variance estimates for every user of the method. The package vignette (`vignette("custom-methods", package = "sondage")`) shows how to check a declared family by simulation.

See Also

`registered_methods()`, `is_registered_method()`, `unregister_method()`, `method_spec()`, `unequal_prob_wor()`, `unequal_prob_wr()`, `balanced_wor()`

Examples

```
# Register a toy random sampler. Successive sampling with
# prob = pik does not give first-order inclusion probabilities
# equal to pik, so its probabilities stay "unknown" (the default).
my_sampler <- function(pik, n = NULL, prn = NULL, ...) {
  sample.int(length(pik), size = n, prob = pik)
}
register_method(
  "toy",
  type = "wor",
  sample_fn = my_sampler
)
s <- unequal_prob_wor(c(0.3, 0.3, 0.4), method = "toy")
s$method

# Register a toy balanced sampler (ignores aux, keeps size fixed)
my_balanced <- function(pik, n = NULL, aux = NULL, ...) {
  sample.int(length(pik), size = n, prob = pik)
}
register_method("toy_bal", type = "balanced", sample_fn = my_balanced)
s <- balanced_wor(c(0.3, 0.3, 0.4), method = "toy_bal")
s$method
```

```
# Clean up
unregister_method("toy")
unregister_method("toy_bal")
```

---

sampling_cov *Sampling Covariance Matrix*

Description

Computes the sampling covariance matrix used in variance estimation.

Usage

```
sampling_cov(x, ...)

## S3 method for class 'wor'
sampling_cov(x, weighted = FALSE, sampled_only = FALSE, ...)

## S3 method for class 'wr'
sampling_cov(x, weighted = FALSE, sampled_only = FALSE, ...)

## Default S3 method:
sampling_cov(x, ...)
```

Arguments

x	A sampling design object (class "sondage_sample").
...	Additional arguments passed to joint_inclusion_prob() or joint_expected_hits().
weighted	If FALSE (default), returns the raw covariance matrix. If TRUE, returns the weighted check quantities used in the Sen-Yates-Grundy variance estimator.
sampled_only	If TRUE, compute only the submatrix for sampled units. Passed to joint_inclusion_prob() or joint_expected_hits(). Default FALSE.

Details

For without-replacement designs: $\Delta_{ij} = \pi_{ij} - \pi_i \pi_j$.

For with-replacement designs: $E(n_i n_j) - E(n_i)E(n_j)$.

When weighted = TRUE, returns the Sen-Yates-Grundy check quantities: $1 - \pi_i \pi_j / \pi_{ij}$ for WOR, $1 - E(n_i)E(n_j)/E(n_i n_j)$ for WR.

Accuracy depends on the underlying joint probabilities. For cps, sampford, systematic, poisson, srs, bernoulli, multinomial, and systematic_mr, joint probabilities or pairwise expectations are exact and so is the covariance. For brewer, sps, pareto, and cube, they use the high-entropy approximation. For chromy, they are simulation-based (see nsim). lpm2 and scps expose no joint

probabilities, so `sampling_cov()` stops with an error for them. See `joint_inclusion_prob()` and `joint_expected_hits()`.

Some designs (notably systematic PPS) produce $\pi_{ij} = 0$ for pairs that never co-occur, and `systematic_mr` gives such pairs zero pairwise expected hits. When `weighted = TRUE`, the SYG quantity $1 - \pi_i\pi_j/\pi_{ij}$ is undefined for such pairs and set to `NA` with a warning. The raw covariance (`weighted = FALSE`) is unaffected. The Sen-Yates-Grundy estimator is not applicable for these designs (Tillé, 2006, Ch. 5).

Value

A symmetric N x N matrix (or n x n if `sampled_only = TRUE`). For WOR designs with `weighted = FALSE`, off-diagonal entries are typically negative for well-behaved designs. With `weighted = TRUE`, off-diagonal entries are typically non-positive (entries where $\pi_{ij} = 0$ are set to `NA`). Unit labels are inherited from the underlying joint matrix.

References

Chromy, J.R. (2009). Some generalizations of the Horvitz-Thompson estimator. *Proceedings of the Survey Research Methods Section, American Statistical Association*.

Tillé, Y. (2006). *Sampling Algorithms*. Springer.

See Also

`joint_inclusion_prob()` for the without-replacement joint probabilities this is computed from, `joint_expected_hits()` for the with-replacement analogue.

Examples

```
pik <- c(0.2, 0.4, 0.6, 0.8)
s <- unequal_prob_wor(pik, method = "cps")

# Raw covariance
sampling_cov(s)

# SYG check quantities
sampling_cov(s, weighted = TRUE)

# Covariance for sampled units only
sampling_cov(s, sampled_only = TRUE)
```

---

sondage_sample *Sampling Design Objects*

Description

All sampling functions in sondage return objects inheriting from class `"sondage_sample"`. These objects store the realized sample together with the design-defining quantities needed by the query generics.

Details

All sondage_sample objects contain:

$sample Sample indices (integer vector, or matrix/list when nrep > 1). Built-in without-replacement methods return the indices in ascending order, except equal_prob_wor(method = "srs"), which keeps the order drawn by sample.int(). With-replacement methods and registered methods keep the order in which their sampler returns the units.

$n Sample size. For designs with $fixed_size = TRUE, an integer equal to the realized sample size. For random-size designs ($fixed_size = FALSE), a double equal to the expected sample size: sum(pik) for "poisson" and for registered methods with fixed_size = FALSE, the user-supplied target for "bernoulli".

$N Population size.

$method Sampling method name.

$fixed_size Whether the sample size is fixed by design.

Without-replacement ("wor") objects also contain:

$pik Inclusion probability vector. For most methods this equals the true first-order probabilities. For "sps" and "pareto", this is the target vector.

With-replacement ("wr") objects also contain:

$prob Per-draw selection probability vector.

$hits Realized selection counts (integer vector, or N x nrep matrix when nrep > 1).

The class vector records the design family:

c("equal_prob", "wor", "sondage_sample") Equal probability, without replacement.

c("equal_prob", "wr", "sondage_sample") Equal probability, with replacement.

c("unequal_prob", "wor", "sondage_sample") Unequal probability, without replacement.

c("balanced", "unequal_prob", "wor", "sondage_sample") Balanced or spatially balanced sampling, including cube, LPM2, SCPS, and registered methods. Dispatch falls through to unequal_prob / wor methods unless a balanced-specific method is defined.

c("unequal_prob", "wr", "sondage_sample") Unequal probability, with replacement.

See Also

equal_prob_wor(), equal_prob_wr(), unequal_prob_wor(), unequal_prob_wr(), balanced_wor(), inclusion_prob(), expected_hits(), joint_inclusion_prob(), joint_expected_hits(), sampling_cov(), print.sondage_sample()

unequal_prob_wor *Unequal Probability Sampling Without Replacement*

Description

Draws a sample with unequal inclusion probabilities, without replacement.

Usage

```
unequal_prob_wor(
  pik,
 method = c("cps", "sampford", "brewer", "systematic", "poisson", "sps", "pareto"),
  nrep = 1L,
  prn = NULL,
  ...
)
```

Arguments

pik	A numeric vector of inclusion probabilities. For fixed-size methods, `sum(pik)` must be an integer to floating-point accuracy: an exact fixed-size design cannot have a non-integer sum, so looser sums are rejected rather than silently rounded. Their target sample size, `sum(pik)`, must be at least 1. Units with `pik` of exactly 0 are never selected and units with exactly 1 are always selected and values in between are sampled as given.
method	The sampling method:

`"cps"` Conditional Poisson Sampling (maximum entropy, Chen et al., 1994). Fixed size, exact joint probabilities. Two units with probabilities strictly between 0 and 1 have $\pi_{ij} > 0$ when the design selects at least two such units. Calibration and the conditional probability table are O(Nn) (probability-domain Poisson-binomial recurrence), each draw is O(N) thereafter. Equal `pik` are drawn directly as SRS.

`"sampford"` Sampford's (1967) fixed-size PPS design. Gives the supplied first-order inclusion probabilities exactly and has exact joint inclusion probabilities. A native C kernel combines the fast rejection construction with Grafström's non-rejective conditional-Poisson fallback and draws the smaller of the sample and its complement. Typical O(N + n), and O(Nn) in the fallback.

`"brewer"` Brewer's (1975) draw-by-draw method. Fixed size, approximate joint probabilities (high-entropy approximation, see `joint_inclusion_prob()`). O(Nn).

`"systematic"` Systematic PPS. Fixed size, exact joint probabilities but **some may be zero** (pairs that never co-occur), making the SYG estimator inapplicable. See `sampling_cov()`. O(N).

`"poisson"` Poisson sampling. **Random** sample size (expected $n = \sum \pi_k$). Units selected independently, so $\pi_{ij} = \pi_i \pi_j$. Supports PRN. O(N).

"sps" Sequential Poisson Sampling (Ohlsson, 1998). Order sampling with key $\xi_k = u_k/\pi_k$, where the n smallest are selected. Fixed size, high-entropy. Supports PRN. Approximate joint probabilities. The true first-order inclusion probabilities are approximately equal to the supplied pik. See inclusion_prob(). Expected O(N). Tied keys (possible with duplicated prn values) are broken toward the smallest population index.

"pareto" Pareto sampling (Rosen, 1997). Order sampling with odds-ratio key $\xi_k = [u_k/(1-u_k)]/[\pi_k/(1-\pi_k)]$. Same properties as "sps". Expected O(N).

nrep — Number of replicate samples (default 1). When nrep > 1, $sample holds a matrix (fixed-size) or list (random-size) of all replicates. The design object and all generics remain usable.

prn — Optional vector of permanent random numbers (length N, values in the open interval (0, 1)) for sample coordination. Supported by methods "sps", "pareto", and "poisson". When NULL, random numbers are generated internally. Cannot be used with nrep > 1 (identical PRN would produce identical replicates). Use a loop with different PRN vectors for coordinated repeated sampling.

... — Additional arguments passed to methods registered via register_method(). Built-in methods take no additional arguments. The former eps boundary-trimming argument was removed because it silently changed the design.

Value

An object of class c("unequal_prob", "wor", "sondage_sample"). When nrep = 1, $sample is an integer vector of selected unit indices. When nrep > 1, $sample is a matrix (n x nrep) for fixed-size methods, or a list of integer vectors of varying lengths for random-size methods ("poisson" and registered methods with fixed_size = FALSE). $n is an integer for fixed-size methods (realized size) and a double for random-size methods (expected size, sum(pik)). See sondage_sample.

Near-certainty inclusion probabilities in CPS

The CPS fixed-point calibration converges geometrically for well-spread pik, but asymptotes at a non-zero defect when some pik are within a few decimal digits of 0 or 1 (e.g. 0.9999). When this happens the function emits a "CPS calibration did not reach tolerance" warning reporting the achieved max_diff. The realized first-order inclusion probabilities differ from the target by up to max_diff, typically 1e-5 or smaller for inputs in the 0.999-range, well within Monte Carlo error for most estimators. If the warning is unwanted, clip pik away from 0/1 before calling.

References

Chen, X. H., Dempster, A. P., & Liu, J. S. (1994). Weighted finite population sampling to maximize entropy. *Biometrika*, 81(3), 457-469.

Brewer, K.R.W. (1975). A simple procedure for sampling pi-ps wor. *Australian Journal of Statistics*, 17(3), 166-172.

Sampford, M.R. (1967). On sampling without replacement with unequal probabilities of selection. *Biometrika*, 54(3/4), 499-513.

Grafstrom, A. (2009). Non-rejective implementations of the Sampford sampling design. *Journal of Statistical Planning and Inference*, 139(6), 2111-2114.

Ohlsson, E. (1998). Sequential Poisson sampling. *Journal of Official Statistics*, 14(2), 149-162.

Rosen, B. (1997). On sampling with probability proportional to size. *Journal of Statistical Planning and Inference*, 62(2), 159-191.

Tillé, Y. (2006). *Sampling Algorithms*. Springer.

See Also

`unequal_prob_wr()` for with-replacement designs, `equal_prob_wor()` for equal probability designs, `inclusion_prob()` to compute inclusion probabilities from size measures.

Examples

```
pik <- c(0.2, 0.4, 0.6, 0.8)

# Conditional Poisson Sampling
set.seed(123)
s <- unequal_prob_wor(pik, method = "cps")
s$sample

# Brewer's method
s <- unequal_prob_wor(pik, method = "brewer")
s$sample

# Sequential Poisson Sampling with PRN coordination
prn <- runif(4)
s <- unequal_prob_wor(pik, method = "sps", prn = prn)
s$sample

# Pareto sampling
s <- unequal_prob_wor(pik, method = "pareto", prn = prn)
s$sample


# Batch mode for simulations
sim <- unequal_prob_wor(pik, method = "cps", nrep = 1000)
dim(sim$sample)  # 2 x 1000
```

---

unequal_prob_wr *Unequal Probability Sampling With Replacement*

Description

Draws a sample with unequal selection probabilities, with replacement or minimum replacement.

Usage

```
unequal_prob_wr(
  hits,
  method = c("chromy", "multinomial", "systematic_mr"),
  nrep = 1L,
  prn = NULL,
  ...
)
```

Arguments

`hits`	A numeric vector of expected hits (expected number of selections per unit). Typically computed via `expected_hits()`. `sum(hits)` is the sample size, so it must be a whole number of at least 1 to floating-point accuracy. Looser sums are rejected rather than silently rounded.
`method`	The sampling method: `"chromy"` Chromy's (1979) sequential PPS with minimum replacement. Pairwise expectations $E(n_i n_j)$ are estimated by simulation. See `joint_expected_hits()`. Complexity: O(N + n). `"multinomial"` Multinomial PPS (independent draws). Units can be selected any number of times. Pairwise expectations are exact: $E(n_i n_j) = n(n-1)p_i p_j$. Complexity: O(n). `"systematic_mr"` Systematic PPS with minimum replacement (Madow, 1949), the design of `METHOD=PPS_SYS` in SAS SURVEYSELECT. One random start walks the cumulated expected hits in the order of `hits`, which acts as implicit stratification. Each unit receives the whole part of its expected hits or one more. Unlike `unequal_prob_wor(method = "systematic")`, which takes a unit larger than the sampling interval once, with certainty, this method hits it once per interval it spans. When every expected hit is below 1, both draw the same units from the same seed. Pairwise expectations are exact, and some are zero for units that are never selected together. See `sampling_cov()`. Complexity: O(N + n), since the sample lists each hit.
`nrep`	Number of replicate samples (default 1).
`prn`	Optional vector of permanent random numbers for sample coordination. No built-in WR method currently supports `prn`; supplying it is an error. Registered methods can opt into PRN support.
`...`	Additional arguments passed to methods registered via `register_method()`. Built-in methods take no additional arguments.

Value

An object of class `c("unequal_prob", "wr", "sondage_sample")`. When `nrep = 1`, `$sample` is an integer vector and `$hits` is an integer vector. When `nrep > 1`, `$sample` is a matrix (n x nrep) and `$hits` is a matrix (N x nrep).

References

Chromy, J.R. (1979). Sequential sample selection methods. *Proceedings of the Survey Research Methods Section, ASA*, 401-406.

Chromy, J.R. (2009). Some generalizations of the Horvitz-Thompson estimator. *Proceedings of the Survey Research Methods Section, American Statistical Association*.

Madow, W.G. (1949). On the theory of systematic sampling, II. *Annals of Mathematical Statistics*, 20(3), 333-354.

See Also

`unequal_prob_wor()` for without-replacement designs, `expected_hits()` to compute expected hits from size measures.

Examples

```
x <- c(40, 80, 50, 60, 70)
hits <- expected_hits(x, n = 3)

set.seed(12345)
s <- unequal_prob_wr(hits, method = "chromy")
s$sample
s$hits

# A unit larger than the sampling interval can be hit twice
hits <- expected_hits(c(40, 80, 50, 60, 270), n = 3)
s <- unequal_prob_wr(hits, method = "systematic_mr")
s$hits
```

---

unregister_method *Remove a Registered Method*

Description

Remove a Registered Method

Usage

```
unregister_method(name)
```

Arguments

`name`	Method name to unregister.

Value

Invisible `TRUE` if the method was removed, `FALSE` if it was not registered.

See Also

register_method()

Examples

```
unregister_method("nonexistent")
```

Index

balanced_wor, 4
balanced_wor(), *3*, *22*, *25, 26*, *28, 29*, *32*

equal_prob_wor, 10
equal_prob_wor(), *3*, *12*, *32*, *35*
equal_prob_wr, 12
equal_prob_wr(), *3*, *11*, *18*, *32*
expected_hits, 13
expected_hits(), *3, 4*, *18*, *32*, *36, 37*

hajek_jip, 14
hajek_jip(), *3*, *16, 17*
he_jip, 15
he_jip(), *3*, *14, 15*, *22*

inclusion_prob, 17
inclusion_prob(), *3, 4*, *9*, *13*, *32*, *34, 35*
is_registered_method, 19
is_registered_method(), *29*

joint_expected_hits, 20
joint_expected_hits(), *3*, *22*, *26*, *28*, *30–32*, *36*
joint_inclusion_prob, 21
joint_inclusion_prob(), *3*, *6, 7*, *14–17*, *21*, *26*, *28*, *30–33*

method_spec, 23
method_spec(), *8*, *29*

print.equal_prob *(*print.sondage_sample*)*, 24
print.sondage_sample, 24
print.sondage_sample(), *32*
print.unequal_prob *(*print.sondage_sample*)*, 24

register_method, 25
register_method(), *3*, *5*, *14–17*, *19*, *22, 23*, *25*, *34*, *36*, *38*
registered_methods, 25
registered_methods(), *23*, *29*

sample.int(), *32*
sampling_cov, 30
sampling_cov(), *3*, *21, 22*, *32, 33*, *36*
scale(), *7*
sondage *(*sondage-package*)*, 2
sondage-package, 2
sondage_sample, *11*, *24*, 31, *34*

unequal_prob_wor, 33
unequal_prob_wor(), *3*, *5*, *8, 9*, *11*, *18*, *22*, *25*, *29*, *32*, *37*
unequal_prob_wr, 35
unequal_prob_wr(), *3*, *12, 13*, *18*, *25*, *29*, *32*, *35*
unregister_method, 37
unregister_method(), *27*, *29*