Package ‘sps’

November 19, 2025

Title Sequential Poisson Sampling

Version 0.6.3

Description Sequential Poisson sampling is a variation of Poisson sampling for drawing probability-proportional-to-size samples with a given number of units, and is commonly used for price-index surveys. This package gives functions to draw stratified sequential Poisson samples according to the method by Ohlsson (1998, ISSN:0282-423X), as well as other order sample designs by Rosén (1997, <doi:10.1016/S0378-3758(96)00186-3>), and generate approximate bootstrap replicate weights according to the generalized bootstrap method by Beaumont and Patak (2012, <doi:10.1111/j.1751-5823.2011.00166.x>).

Depends R (>= 4.2)

Imports stats

Suggests kit (>= 0.0.10), knitr, quarto, testthat (>= 3.0.0)

Config/testthat/edition 3

License MIT + file LICENSE

Encoding UTF-8

URL `https://marberts.github.io/sps/`, `https://github.com/marberts/sps`

BugReports `https://github.com/marberts/sps/issues`

VignetteBuilder quarto

RoxygenNote 7.3.3

NeedsCompilation no

Author Steve Martin [aut, cre, cph] (ORCID: <`https://orcid.org/0000-0003-2544-9480`>), Justin Francis [ctb]

Maintainer Steve Martin <`marberts@protonmail.com`>

Repository CRAN

Date/Publication 2025-11-19 04:50:02 UTC

Contents

`expected_coverage` *Expected coverage*

Description

Find the expected number of strata covered by ordinary Poisson sampling without stratification. As sequential and ordinary Poisson sampling have the same sample size on average, this gives an approximation for the coverage under sequential Poisson sampling.

This function can also be used to calculate, e.g., the expected number of enterprises covered within a stratum when sampling business establishments.

Usage

```
expected_coverage(x, n, strata, alpha = 0.001, cutoff = Inf)
```

Arguments

`x`	A positive and finite numeric vector of sizes for units in the population (e.g., revenue for drawing a sample of businesses).
`n`	A positive integer giving the sample size.
`strata`	A factor, or something that can be coerced into one, giving the strata associated with units in the population. The default is to place all units into a single stratum.
`alpha`	A numeric vector with values between 0 and 1 for each stratum, ordered according to the levels of `strata`. Units with inclusion probabilities greater than or equal to 1 - `alpha` are set to 1 for each stratum. A single value is recycled for all strata. The default is slightly larger than 0.
`cutoff`	A positive numeric vector of cutoffs for each stratum, ordered according to the levels of `strata`. Units with `x` >= `cutoff` get an inclusion probability of 1 for each stratum. A single value is recycled for all strata. The default does not apply a cutoff.

Value

The expected number of strata covered by the sample design.

See Also

`prop_allocation()` for generating proportional-to-size allocations.

Examples

```
# Make a population with units of different size
x <- c(rep(1:9, each = 3), 100, 100, 100)

# ... and 10 strata
s <- rep(letters[1:10], each = 3)

# Should get about 7 to 8 strata in a sample on average
expected_coverage(x, 15, s)
```

`inclusion_prob` *Calculate inclusion probabilities*

Description

Calculate stratified (first-order) inclusion probabilities.

Usage

```
inclusion_prob(x, n, strata = NULL, alpha = 0.001, cutoff = Inf)

becomes_ta(x, alpha = 0.001, cutoff = Inf)
```

Arguments

`x`	A positive and finite numeric vector of sizes for units in the population (e.g., revenue for drawing a sample of businesses).
`n`	A positive integer vector giving the sample size for each stratum, ordered according to the levels of `strata`. A single value is recycled for all strata. Non-integers are truncated towards 0.
`strata`	A factor, or something that can be coerced into one, giving the strata associated with units in the population. The default is to place all units into a single stratum.
`alpha`	A numeric vector with values between 0 and 1 for each stratum, ordered according to the levels of `strata`. Units with inclusion probabilities greater than or equal to 1 - `alpha` are set to 1 for each stratum. A single value is recycled for all strata. The default is slightly larger than 0.
`cutoff`	A positive numeric vector of cutoffs for each stratum, ordered according to the levels of `strata`. Units with x >= `cutoff` get an inclusion probability of 1 for each stratum. A single value is recycled for all strata. The default does not apply a cutoff.

Details

Within a stratum, the inclusion probability for a unit is given by $\pi = nx/\sum x$. These values can be greater than 1 in practice, and so they are constructed iteratively by taking units with $\pi \geq 1 - \alpha$ (from largest to smallest) and assigning these units an inclusion probability of 1, with the remaining inclusion probabilities recalculated at each step. See `vignette("take-all")` for details. If $\alpha > 0$, then any ties among units with the same size are broken by their position.

The `becomes_ta()` function reverses this operations and finds the critical sample size at which a unit enters the take-all stratum. This value is undefined for units that are always included in the sample (because their size exceeds `cutoff`) or never included.

Value

`inclusion_prob()` returns a numeric vector of inclusion probabilities for each unit in the population.

`becomes_ta()` returns an integer vector giving the sample size at which a unit enters the take-all stratum.

Note

`kit::topn()` is used if available to improve performance in the normal case when the sample size is small relative to the population.

See Also

`sps()` for drawing a sequential Poisson sample.

Examples

```
# Make inclusion probabilities for a population with units
# of different size
x <- c(1:10, 100)
(pi <- inclusion_prob(x, 5))

# The last unit is sufficiently large to be included in all
# samples with two or more units
becomes_ta(x)

# Determine the number of take-all units before drawing a sample
n_ta <- function(x, n, ...) {
  sum(becomes_ta(x, ...) <= n, na.rm = TRUE)
}

n_ta(x, 7)

# Use the inclusion probabilities to calculate the variance of the
# sample size for Poisson sampling
sum(pi * (1 - pi))
```

`prop_allocation` *Construct a proportional allocation*

Description

Generate a proportional-to-size allocation for stratified sampling.

Usage

```
prop_allocation(
  x,
  n,
  strata,
  initial = 0L,
  divisor = divisor_method("Jefferson/D'Hondt"),
  ties = c("largest", "first")
)

divisor_method(
  name = c(
    "Jefferson/D'Hondt",
    "Webster/Sainte-Lague",
    "Imperiali",
    "Huntington-Hill",
    "Danish",
    "Adams",
    "Dean"
    )
  )
```

Arguments

`x`	A positive and finite numeric vector of sizes for units in the population (e.g., revenue for drawing a sample of businesses).
`n`	A positive integer giving the sample size.
`strata`	A factor, or something that can be coerced into one, giving the strata associated with units in the population. The default is to place all units into a single stratum.
`initial`	A positive integer vector giving the initial (or minimal) allocation for each stratum, ordered according to the levels of `strata`. A single integer is recycled for each stratum using a special algorithm to ensure a feasible allocation; see details. Non-integers are truncated towards 0. The default allows for no units to be allocated to a stratum.
`divisor`	A function for the divisor (highest-averages) apportionment method. The default uses the Jefferson/D'Hondt method. See details for other possible functions.
`ties`	Either 'largest' to break ties in favor of the stratum with the largest size (the default), or 'first' to break ties in favor of the ordering of `strata`.
`name`	Name of the divisor function. See details.

Details

The `prop_allocation()` function gives a sample size for each level in `strata` that is proportional to the sum of `x` across strata and adds up to `n`. This is done using the divisor (highest-averages) apportionment method (Balinksi and Young, 1982, Appendix A), for which there are a number of different divisor functions:

Jefferson/D'Hondt `\(a) a + 1`

Webster/Sainte-Laguë `\(a) a + 0.5`

Imperiali `\(a) a + 2`

Huntington-Hill `\(a) sqrt(a * (a + 1))`

Danish `\(a) a + 1 / 3`

Adams `\(a) a`

Dean `\(a) a * (a + 1) / (a + 0.5)`

Note that a divisor function with $d(0) = 0$ (i.e., Huntington-Hill, Adams, Dean) should have an initial allocation of at least 1 for all strata. In all cases, ties are broken according to the sum of `x` if `ties = 'largest'`; otherwise, if `ties = 'first'`, then ties are broken according to the levels of `strata`.

In cases where the number of units with non-zero size in a stratum is smaller than its allocation, the allocation for that stratum is set to the number of available units, with the remaining sample size reallocated to other strata proportional to `x`. This is similar to `PROC SURVEYSELECT` in SAS with `ALLOC = PROPORTIONAL`.

Passing a single integer for the initial allocation first checks that recycling this value for each stratum does not result in an allocation larger than the sample size. If it does, then the value is reduced so that recycling does not exceed the sample size. This recycled vector can be further reduced in cases where it exceeds the number of units in a stratum, the result of which is the initial allocation. This special recycling ensures that the initial allocation is feasible.

Value

`prop_allocation()` returns a named integer vector of sample sizes for each stratum in `strata`.

`divisor_method()` returns a function giving the desired divisor function.

References

Balinksi, M. L. and Young, H. P. (1982). *Fair Representation: Meeting the Ideal of One Man, One Vote*. Yale University Press.

See Also

`sps()` for stratified sequential Poisson sampling.

`expected_coverage()` to calculate the expected number of strata in a sample without stratification.

`strAlloc()` in the **PracTools** package and the **optimall** package for other allocation methods.

Examples

```
# Make a population with units of different size
x <- c(rep(1:9, each = 3), 100, 100, 100)

# ... and 10 strata
s <- rep(letters[1:10], each = 3)

# Generate an allocation
prop_allocation(x, 15, s, initial = 1)
```

`sps` *Stratified sequential Poisson sampling*

Description

Draw a stratified probability-proportional-to-size sample using the sequential and ordinary Poisson methods, and generate other order sampling schemes.

Usage

```
sps(x, n, strata = NULL, prn = NULL, alpha = 0.001, cutoff = Inf)

ps(x, n, strata = NULL, prn = NULL, alpha = 0.001, cutoff = Inf)

order_sampling(dist)
```

Arguments

`x`	A positive and finite numeric vector of sizes for units in the population (e.g., revenue for drawing a sample of businesses).
`n`	A positive integer vector giving the sample size for each stratum, ordered according to the levels of `strata`. A single value is recycled for all strata. Non-integers are truncated towards 0.
`strata`	A factor, or something that can be coerced into one, giving the strata associated with units in the population. The default is to place all units into a single stratum.
`prn`	A numeric vector of permanent random numbers for units in the population, distributed uniform between 0 and 1. The default does not use permanent random numbers, instead generating a random vector when the function is called.
`alpha`	A numeric vector with values between 0 and 1 for each stratum, ordered according to the levels of `strata`. Units with inclusion probabilities greater than or equal to 1 - `alpha` are set to 1 for each stratum. A single value is recycled for all strata. The default is slightly larger than 0.
`cutoff`	A positive numeric vector of cutoffs for each stratum, ordered according to the levels of `strata`. Units with `x` >= `cutoff` get an inclusion probability of 1 for each stratum. A single value is recycled for all strata. The default does not apply a cutoff.

`dist`	A function giving the fixed order distribution shape for an order sampling scheme. See details.

Details

The `sps()` function draws a sample according to the sequential Poisson procedure, the details of which are given by Ohlsson (1998). It is also called uniform order sampling, as it is a type of order sampling; see Rosén (1997, 2000) for a more general presentation of the method. This is the same method used by `PROC SURVEYSELECT` in SAS with `METHOD = SEQ_POISSON`.

For each stratum, the sequential Poisson procedure starts by stratifying units in the population based on their (target) inclusion probabilities π. Units with $\pi = 0$ are placed into a take-none stratum, units with $0 < \pi < 1$ are placed into a take-some stratum, and units with $\pi = 1$ are placed into a take-all stratum. As noted by Ohlsson (1998), it can be useful to set α to a small positive value when calculating inclusion probabilities, and this is the default behavior.

After units are appropriately stratified, a sample of take-some units is drawn by assigning each unit a value $\xi = u/\pi$, where u is a random deviate from the uniform distribution between 0 and 1. The units with the smallest values for ξ are included in the sample, along with the take-all units. (Ties in ξ are technically a measure-zero event—in practice these are broken by position.) This results in a fixed sample size at the expense of the sampling procedure being only approximately probability-proportional-to-size (i.e., the inclusion probabilities from the sample design are close but not exactly equal to π; see Matei and Tillé, 2007, for details on the exact computation).

Ordinary Poisson sampling follows the same procedure as above, except that all units with $\xi < 1$ are included in the sample; consequently, while it does not contain a fixed number of units, the procedure is strictly probability-proportional-to-size. Despite this difference, the standard Horvitz-Thompson estimator for the total (of the take-some stratum) is asymptotically unbiased, normally distributed, and equally efficient under both procedures. The `ps()` function draws a sample using the ordinary Poisson method.

A useful feature of sequential and ordinary Poisson sampling is the ability to coordinate samples by using permanent random numbers for u. Keeping u fixed when updating a sample retains a larger number of overlapping units, whereas switching u for $u - z \bmod 1$ or $1 - (u - z \bmod 1)$, for some z between 0 and 1, when drawing different samples from the same frame reduces the number of overlapping units.

Despite the focus on sequential Poisson sampling, all order sampling procedures follow the same approach as sequential Poisson sampling. The `order_sampling()` function can be used to generate other order sampling functions by passing an appropriate function to make the ranking variable ξ:

Sequential Poisson sampling `\(x) x`
Successive sampling `\(x) log(1 - x)`
Pareto sampling `\(x) x / (1 - x)`

Value

`sps()` and `ps()` return an object of class `sps_sample`. This is an integer vector of indices for the units in the population that form the sample, along with a `weights` attribute that gives the design (inverse probability) weights for each unit in the sample (keeping in mind that sequential Poisson sampling is only approximately probability-proportional-to-size). `weights()` can be used to access the design weights attribute of an `sps_sample` object, and `levels()` can be used to determine which units are in the take-all or take-some strata. Mathematical and binary/unary operators strip attributes, as does replacement.

`order_sampling` returns a function the with the same interface as `sps()` and `ps()`.

Note

`kit::topn()` is used if available to improve performance in the normal case when the sample size is small relative to the population.

References

Matei, A., and Tillé, Y. (2007). Computational aspects of order πps sampling schemes. *Computational Statistics & Data Analysis*, 51: 3703-3717.

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

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

Rosén, B. (2000). On inclusion probabilities for order πps sampling. *Journal of Statistical Planning and Inference*, 90(1): 117-143.

See Also

`prop_allocation()` for generating proportional-to-size allocations.

`inclusion_prob()` for calculating the inclusion probabilities.

`sps_repweights()` for generating bootstrap replicate weights.

The `UPpoisson()` and `UPopips()` functions in the **sampling** package for ordinary and sequential Poisson sampling, respectively. Note that the algorithm for order sampling in the `UPopips()` function is currently incorrect, giving a worse approximation for the inclusion probabilities than it should.

The `UP*` functions in the **sampling** package, the `S.*` functions in the **TeachingSampling** package, and the **pps** package for other probability-proportional-to-size sampling methods.

The `pps()` function in the **prnsamplr** package for Pareto order sampling with permanent random numbers.

Examples

```
# Make a population with units of different size
x <- c(1:10, 100)

#---- Sequential Poisson sampling ----
# Draw a sequential Poisson sample
(samp <- sps(x, 5))

# Get the design (inverse probability) weights
weights(samp)

# All units except 11 are in the take-some (TS) stratum
levels(samp)
```

```
# Ensure that the top 10% of units are in the sample
sps(x, 5, cutoff = quantile(x, 0.9))

#---- Ordinary Poisson sampling ----
# Ordinary Poisson sampling gives a random sample size for the
# take-some stratum
ps(x, 5)

#---- Stratified Sequential Poisson sampling ----
# Draw a stratified sample with a proportional allocation
strata <- rep(letters[1:4], each = 5)
(allocation <- prop_allocation(1:20, 12, strata))
(samp <- sps(1:20, allocation, strata))

# Use the Horvitz-Thompson estimator to estimate the total
y <- runif(20) * 1:20
sum(weights(samp) * y[samp])

#---- Useful properties of Sequential Poisson sampling ----
# It can be useful to set 'prn' in order to extend the sample
# to get a fixed net sample
u <- runif(11)
(samp <- sps(x, 6, prn = u))

# Removing unit 5 gives the same net sample
sps(x[-samp[5]], 6, prn = u[-samp[5]])

# Also useful for topping up a sample
all(samp %in% sps(x, 7, prn = u))

#---- Other order-sampling methods ----
# Generate new order-sampling functions from the parameters of
# the inverse generalized Pareto distribution
igpd <- function(shape, scale = 1, location = 0) {
  if (shape == 0) {
    function(x) -scale * log(1 - x) + location
  } else {
    function(x) scale * (1 - (1 - x)^shape) / shape + location
  }
}

order_sampling2 <- function(x) order_sampling(igpd(x))

order_sampling2(1)(x, 6, prn = u) # sequential Poisson
order_sampling2(0)(x, 6, prn = u) # successive
order_sampling2(-1)(x, 6, prn = u) # Pareto
```

`sps_iterator` *Iteratively draw a sequential Poisson sample*

Description

Create a function that draws new units, one at a time, according to the sequential Poisson method without replacing previously sampled units.

Usage

```
sps_iterator(x, n = 0L, prn = NULL, alpha = 0.001, cutoff = Inf)
```

Arguments

`x`	A positive and finite numeric vector of sizes for units in the population (e.g., revenue for drawing a sample of businesses).
`n`	A positive integer giving the initial sample size for the iterator.
`prn`	A numeric vector of permanent random numbers for units in the population, distributed uniform between 0 and 1. The default does not use permanent random numbers, instead generating a random vector when the function is called.
`alpha`	A number between 0 and 1. Units with inclusion probabilities greater than or equal to 1 - `alpha` are set to 1. The default is slightly larger than 0.
`cutoff`	A numeric cutoff. Units with `x >= cutoff` get an inclusion probability of 1. The default does not apply a cutoff.

Value

A function that returns the next unit in the sample. It take a single argument giving the sentinel value to indicate that there are no units left to sample (default `NULL`).

Examples

```
prn <- runif(5)
s <- sps_iterator(1:5, prn = prn)
s()
s()
s()

# Same as drawing the sample with 3 units.
sps(1:5, 3, prn = prn)
```

`sps_repweights` *Bootstrap replicate weights for sequential Poisson sampling*

Description

Produce bootstrap replicate weights that are appropriate for Poisson sampling, and therefore approximately correct for sequential Poisson sampling.

Usage

```
sps_repweights(w, replicates = 1000L, tau = min_tau(1e-04), dist = NULL)

min_tau(tol)
```

Arguments

`w`	A numeric vector of design (inverse probability) weights for a (sequential) Poisson sample.
`replicates`	A positive integer that gives the number of bootstrap replicates (1,000 by default). Non-integers are truncated towards 0.
`tau`	A number greater than or equal to 1 that gives the rescale factor for the bootstrap weights. Setting to 1 does not rescale the weights. This can also be a function that takes a vector of bootstrap adjustments and returns a number larger than 1. The default automatically picks the smallest feasible rescale factor (up to a small tolerance).
`dist`	A function that produces random deviates with mean 0 and standard deviation 1, such as `rnorm()`. The default uses the pseudo-population method from section 4.1 of Beaumont and Patak (2012); see details.
`tol`	A non-negative number, strictly less than 1, that gives the tolerance for determining the minimum feasible value of `tau`.

Details

Replicate weights are constructed using the generalized bootstrap method by Beaumont and Patak (2012). Their method takes a vector of design weights w, finds a vector of adjustments a for each bootstrap replicate, and calculates the replicate weights as aw.

There are two ways to calculate the adjustments a. The default pseudo-population method randomly rounds w for each replicate to produce a collection of integer weights w' that are used to generate a random vector b from the binomial distribution. The vector of adjustments is then $a = 1 + b - w'/w$. Specifying a deviates-generating function for `dist` uses this function to produce a random vector d that is then used to make an adjustment $a = 1 + d\sqrt{1 - 1/w}$.

The adjustments can be rescaled by a value $\tau \geq 1$ to prevent negative replicate weights. With this rescaling, the adjustment becomes $(a + \tau - 1)/\tau$. If $\tau > 1$ then the resulting bootstrap variance estimator should be multiplied by τ^2.

Value

`sps_repweights()` returns a matrix of bootstrap replicate weights with `replicates` columns (one for each replicate) and `length(w)` rows (one for each unit in the sample), with the value of `tau` as an attribute.

`min_tau()` returns a function that takes a vector of bootstrap adjustments and returns the smallest value for τ such that the rescaled adjustments are greater than or equal to `tol`.

Note

As an alternative to the bootstrap, Ohlsson (1998, equations 2.13) proposes an analytic estimator for the variance of the total $\hat{Y} = \sum wy$ (for the take-some units) under sequential Poisson sampling:

$$V(\hat{Y}) = \frac{n}{n-1} \sum \left(1 - \frac{1}{w}\right) \left(wy - \frac{\hat{Y}}{n}\right)^2.$$

See Rosén (1997, equation 3.11) for a more general version of this estimator that can be applied to other order sampling schemes. Replacing the left-most correction by $n/(m-1)$, where m is the number of units in the sample, gives a similar estimator for the total under ordinary Poisson sampling, $\hat{Y} = n/m \sum wy$.

References

Beaumont, J.-F. and Patak, Z. (2012). On the Generalized Bootstrap for Sample Surveys with Special Attention to Poisson Sampling. *International Statistical Review*, 80(1): 127-148.

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

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

See Also

`sps()` for drawing a sequential Poisson sample.

`bootstrapFP()` (with `method = "wGeneralised"`) in the **bootstrapFP** package for calculating the variance of Horvitz-Thompson estimators using the generalized bootstrap and `make_gen_boot_factors()` in the **svrep** package.

Examples

```
# Make a population with units of different size
x <- c(1:10, 100)

# Draw a sequential Poisson sample
(samp <- sps(x, 5))

# Make some bootstrap replicates
dist <- list(
  pseudo_population = NULL,
  standard_normal = rnorm,
  exponential = \(x) rexp(x) - 1,
  uniform = \(x) runif(x, -sqrt(3), sqrt(3))
)

lapply(dist, sps_repweights, w = weights(samp), replicates = 5, tau = 2)
```

Index