Package 'stratallo'

March 12, 2026

Title Optimum Sample Allocation in Stratified Sampling

Version 3.0.1

Description Provides exact analytical algorithms for computing optimum sample allocations in stratified sampling.
Supports classical Neyman-Tschuprow allocation, minimum-cost allocation under a variance constraint, and multi-domain allocation with controlled precision.
Handles lower and upper bounds, cost constraints, and multiple domains.
Includes helper functions for variance computation, allocation summaries, rounding, and example datasets for testing and benchmarking.

License GPL-2

URL https://github.com/wwojciech/stratallo

BugReports https://github.com/wwojciech/stratallo/issues

Depends R (>= 3.5.0)

Imports checkmate, lifecycle, Rdpack

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

VignetteBuilder knitr

RdMacros Rdpack

Config/testthat/edition 3

Copyright Wojciech Wójciak

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.3

Collate 'helpers_dca_rdca.R' 'utils.R' 'alg_dca.R'
'alg_other_authors.R' 'alg_rdca.R' 'alg_rna-sga-coma.R'
'alg_rnabox.R' 'alloc_summary.R' 'check_rdca.R' 'data.R'
'dopt.R' 'opt.R' 'rounding.R' 'stratallo-package.R' 'var.R'

NeedsCompilation no

Author Wojciech Wójciak [aut, cre],
Jacek Wesołowski [sad],
Robert Wieczorkowski [ctb]

Maintainer Wojciech Wójciak <wojciech.wojciak@gmail.com>

Repository CRAN

Date/Publication 2026-03-12 15:20:02 UTC

Contents

`alg_1sided` *Algorithms for Optimum Sample Allocation Under One-Sided Bound Constraints*

Description

[Stable]

Functions implementing selected optimum allocation algorithms for solving the optimum sample allocation problem, formulated as follows:

Minimize

$$f(x_1, \ldots, x_H) = \sum_{h=1}^{H} \frac{A_h^2}{x_h}$$

over $\mathbb{R}_+^H$, subject to

$$\sum_{h=1}^{H} c_h x_h = c,$$

and either

$$x_h \leq M_h, \qquad h = 1, \ldots, H,$$

or

$$x_h \geq m_h, \qquad h = 1, \ldots, H,$$

where $c > 0$, $c_h > 0$, $A_h > 0$, $m_h > 0$, $M_h > 0$, $h = 1, \ldots, H$, are given numbers.

The following is a list of all available algorithms along with the functions that implement them:

- RNA - `rna()`,
- LRNA - `rna()`,
- SGA - `sga()`,
- SGAPLUS - `sgaplus()`,
- COMA - `coma()`.

See the documentation of a specific function for details.

The inequality constraints are optional. The user can choose whether and how they are imposed in the optimization problem, depending on the chosen algorithm:

- Lower bounds $m_1, \ldots, m_H$ can be specified only for the LRNA algorithm (by setting `cmp = .Primitive("<=")` for `rna()`).
- Upper bounds $M_1, \ldots, M_H$ are supported by all other algorithms.
- Simultaneous constraints (both lower and upper bounds) are not supported by these functions.

The costs $c_1, \ldots, c_H$ of surveying one element in a stratum can be specified by the user only for the RNA and LRNA algorithms. For the remaining algorithms, these costs are fixed at 1, i.e., $c_h = 1$, $h = 1, \ldots, H$.

Usage

```
rna(
  total_cost,
  A,
  bounds = NULL,
  unit_costs = 1,
  cmp = .Primitive(">="),
  details = FALSE
)

sga(total_cost, A, M)

sgaplus(total_cost, A, M)

coma(total_cost, A, M)
```

Arguments

`total_cost` (`numeric(1)`)
total survey cost c. Must be strictly positive. Additionally:

- If one-sided lower bounds $m_1, \ldots, m_H$ are imposed, it is required that $c \geq \sum_{h=1}^{H} c_h m_h$, i.e. `total_cost >= sum(unit_costs * bounds)`.
- If one-sided upper bounds $M_1, \ldots, M_H$ are imposed, it is required that $c \leq \sum_{h=1}^{H} c_h M_h$, i.e. `total_cost <= sum(unit_costs * bounds)`.

`A` (`numeric`)
population constants $A_1, \ldots, A_H$. All values must be strictly positive.

`bounds` (`numeric` or `NULL`)
optional lower bounds $m_1, \ldots, m_H$, or upper bounds $M_1, \ldots, M_H$, or `NULL` to indicate that no inequality constraints are imposed. If not `NULL`, `bounds` is interpreted as:

- lower bounds, if `cmp = .Primitive("<=")`, or
- upper bounds, if `cmp = .Primitive(">=")`.

See also `total_cost`.

`unit_costs` (`numeric`)
costs $c_1, \ldots, c_H$ of surveying one element in each stratum. Strictly positive values. May also be of length 1, in which case the value is recycled to match the length of `bounds`.

`cmp` (`function`)
a binary comparison operator used to check for violations of `bounds`. Must be either `.Primitive("<=")` (treating `bounds` as lower bounds and invoking the LRNA algorithm) or `.Primitive(">=")` (treating `bounds` as upper bounds and invoking the RNA algorithm).

The value of this argument has no effect if `bounds` is `NULL`.

`details` (`logical(1)`)
should detailed information on stratum assignments (either take-Neyman or take-bound), values of the set function s, and the number of iterations be included in the output?

`M` (`numeric` or `NULL`)
upper bounds $M_1, \ldots, M_H$ optionally imposed on sample sizes in strata. Set to `NULL` if no upper bounds are imposed. Otherwise, it is required that `total_cost <= sum(unit_costs * M)`.

Details

If no inequality constraints are imposed, the allocation is given by the Neyman allocation:

$$x_h = \frac{A_h}{\sqrt{c_h}} \frac{c}{\sum_{i=1}^{H} A_i \sqrt{c_i}}, \qquad h = 1, \ldots, H.$$

For the *stratified π-estimator* of the population total under *stratified simple random sampling without replacement* design, the parameters of the objective function f are

$$A_h = N_h S_h, \qquad h = 1, \ldots, H,$$

where N_h denotes the size of stratum h and S_h is the standard deviation of the study variable in stratum h.

Value

A numeric vector of optimum sample allocations in strata. In the case of `rna()` only, the return value may also be a `list` containing the optimum allocations and strata assignments.

Functions

- `rna()`: Implements the Recursive Neyman Algorithm (RNA) and its counterpart, the Lower Recursive Neyman Algorithm (LRNA), designed for the optimum allocation problem with one-sided lower-bound constraints. The RNA is described in Wesołowski et al. (2022), whereas the LRNA is introduced in Wójciak (2023).
- `sga()`: The Stenger-Gabler (SGA) algorithm, as proposed by Stenger and Gabler (2005) and described in Wesołowski et al. (2022). This algorithm solves the optimum allocation problem with one-sided upper-bound constraints. It assumes unit costs are constant and equal to 1, i.e., $c_h = 1,\ h = 1, \ldots, H$.
- `sgaplus()`: A modified Stenger-Gabler-type algorithm, described in Wójciak (2019), implemented as the Sequential Allocation (version 1) algorithm. This algorithm solves the optimum allocation problem with one-sided upper-bound constraints. It assumes unit costs are constant and equal to 1, i.e., $c_h = 1,\ h = 1, \ldots, H$.
- `coma()`: The Change of Monotonicity Algorithm (COMA), described in Wesołowski et al. (2022), solves the optimum allocation problem with one-sided upper-bound constraints. It assumes unit costs are constant and equal to 1, i.e., $c_h = 1,\ h = 1, \ldots, H$.

Note

These functions are optimized for internal use and should typically not be called directly by users. Use `opt()` or `optcost()` instead.

References

Wójciak W (2023). "Another Solution for Some Optimum Allocation Problem." *Statistics in Transition new series*, **24**(5), 203-219. doi:10.59170/stattrans2023071.

Wesołowski J, Wieczorkowski R, Wójciak W (2022). "Optimality of the Recursive Neyman Allocation." *Journal of Survey Statistics and Methodology*, **10**(5), 1263-1275. ISSN 2325-0984, doi:10.1093/jssam/smab018.

Wójciak W (2019). *Optimal Allocation in Stratified Sampling Schemes*. Master's thesis, Warsaw University of Technology. http://home.elka.pw.edu.pl/~wwojciak/msc_wwojciech_optimum_alloc.pdf.

Stenger H, Gabler S (2005). "Combining random sampling and census strategies - Justification of inclusion probabilities equal to 1." *Metrika*, **61**(2), 137–156. doi:10.1007/s001840400328.

Särndal C, Swensson B, Wretman J (1992). *Model Assisted Survey Sampling*. Springer New York, NY. ISBN 978-0-387-40620-6.

See Also

opt(), optcost(), rnabox().

Examples

```
A <- c(3000, 4000, 5000, 2000)
m <- c(50, 40, 10, 30) # lower bounds
M <- c(100, 90, 70, 80) # upper bounds

rna(total_cost = 190, A = A, bounds = M)

rna(total_cost = 190, A = A, bounds = m, cmp = .Primitive("<="))

rna(total_cost = 300, A = A, bounds = M)

sga(total_cost = 190, A = A, M = M)

sgaplus(total_cost = 190, A = A, M = M)

coma(total_cost = 190, A = A, M = M)
```

---

alloc_summary *Summarizing the Allocation*

Description

[Stable]

A utility that returns a simple data.frame summarizing the allocation returned by opt() or optcost().

Usage

```
alloc_summary(x, A, m = NULL, M = NULL)
```

Arguments

x	(numeric) sample allocations $x_1, \ldots, x_H$.
A	(numeric) population constants $A_1, \ldots, A_H$.
m	(numeric or NULL) optional lower bounds $m_1, \ldots, m_H$.
M	(numeric or NULL) optional upper bounds $M_1, \ldots, M_H$.

Value

A `data.frame` with $H + 1$ rows and up to seven variables, where H is the number of strata. The first H rows correspond to strata $h = 1, \ldots, H$, while the last row contains column totals (where applicable). The columns include:

A Population constant A_h.

m* Lower bound m_h (if provided).

M* Upper bound M_h (if provided).

allocation The optimized sample size x_h.

take_min* Boolean indicator: $x_h = m_h$.

take_max* Boolean indicator: $x_h = M_h$.

take_Neyman Boolean indicator: $m_h < x_h < M_h$ (or simply the internal Neyman allocation if no bounds were violated).

See Also

`opt()`, `optcost()`.

Examples

```
A <- c(3000, 4000, 5000, 2000)
m <- c(100, 90, 70, 80)
M <- c(200, 150, 300, 210)

xopt_1 <- opt(n = 400, A, m)
alloc_summary(xopt_1, A, m)

xopt_2 <- opt(n = 540, A, m, M)
alloc_summary(xopt_2, A, m, M)
```

---

check_rdca *Internal Diagnostic Functions for Checking Optimality of* `rdca()` *Allocations*

Description

[Stable]

Diagnostic tools to verify the objective function and constraint satisfaction for allocations computed by the `rdca()` algorithm.

Usage

```
rdca_obj_cnstr(x, n, H_counts, N, S, rho2, J = integer(0))

rdca_cnstr_check(x, n, H_counts, N, S, rho2, J = integer(0), tol_max = 0.1)

rdca_optcond_sU(H_counts, S, rho, s, U, return_diff = FALSE)
```

Arguments

x	(numeric) vector of sample allocations.
n	(integerish(1)) total sample size n. Must satisfy `0 < n <= sum(N)`.
H_counts	(integerish) strata counts in each domain.
N	(integerish) strata sizes $(N_{d,h},\ (d,h) \in \mathcal{H})$.
S	(numeric) standard deviations $(S_{d,h},\ (d,h) \in \mathcal{H})$ of surveyed variable in strata.
rho2	(numeric) the square of rho (`rho^2`).
J	(integerish or NULL) vector of domain indices, a subset of $\mathcal{D}$. Specifies domains $d \in \mathcal{D}$ for which values of constraint functions g_{dh} are computed. If `integer(0)`, these values are not computed. If NULL, all domains are included.
tol_max	(integerish(1)) the maximum tolerance exponent (as a power of 10).
rho	(numeric) parameters $(\rho_d,\ d \in \mathcal{D})$ of the optimization problem.
s	(numeric) vector of values for function $s(\mathcal{U}, \boldsymbol{v}, d \mid p)$ calculated only for domains included in U that are not fully blocked by the take-max assignment.
U	(integerish or NULL) a vector of indices identifying the *take-max* strata, i.e., the strata (d,h) for which the allocation is fixed to $x_{d,h} = N_{d,h}$. The indices refer to the positions of strata in the set $\mathcal{H}$, in the same order as in the input vectors (N, S, etc.). For example, if $\mathcal{H} = \{(1,1),\ (2,1)\}$ and stratum $(2,1)$ is a take-max stratum, then U = 2. If U contains all strata from a domain, the dimension of the D matrix is reduced accordingly. U must satisfy one of the following conditions: • n > sum(N[U]), • n = sum(N[U]) and n = sum(N).
return_diff	(logical(1)) If FALSE, the function returns a logical vector indicating whether the optimality condition $$s(\mathcal{U},\ \boldsymbol{v},\ d \mid p) \geq \rho_d / S_{d,h}$$ is satisfied for each unblocked stratum $(d,h) \in \mathcal{U}$. If TRUE, the function returns the **differences** $$s(\mathcal{U},\ \boldsymbol{v},\ d \mid p) - \rho_d / S_{d,h}$$ instead of a logical vector, which can be used to assess by how much the condition is satisfied or violated.

Functions

- `rdca_obj_cnstr()`: Compute the value of the objective function and constraint functions for a given allocation.
- `rdca_cnstr_check()`: Check whether the equality and inequality constraints are satisfied for a given allocation, within a specified tolerance. The tolerance applies to equality constraints only.
- `rdca_optcond_sU()`: Check the optimality condition related to s. Specifically, verifies whether $s(\mathcal{U}, \boldsymbol{v}, d \mid p) \geq \rho_d / S_{d,h}$ for all $(d, h) \in \mathcal{U}$ such that d is not fully blocked by $\mathcal{U}$.

Examples

```
H_counts <- c(2, 2) # 2 domains with 2 strata each
N <- c(140, 110, 135, 190)
S <- sqrt(c(180, 20, 5, 4))
total <- c(2, 3)
kappa <- c(0.4, 0.6)
rho <- total * sqrt(kappa)
rho2 <- total^2 * kappa
n <- 500

(x <- dca(n, H_counts, N, S, rho, rho2))

# internal functions (not exported) - examples skipped

## Not run:
rdca_obj_cnstr(x, n, H_counts, N, S, rho2)
rdca_obj_cnstr(x, n, H_counts, N, S, rho2, 2)
rdca_obj_cnstr(x, n, H_counts, N, S, rho2, NULL)

## End(Not run)

## Not run:
rdca_cnstr_check(x, n, H_counts, N, S, rho2)
rdca_cnstr_check(x, n, H_counts, N, S, rho2, 1)
rdca_cnstr_check(x, n, H_counts, N, S, rho2, 2)
rdca_cnstr_check(x, n, H_counts, N, S, rho2, NULL)

## End(Not run)

## Not run:
(n <- dca_nmax(H_counts, N, S) - 1)

U <- 1
(x <- dca(n, H_counts, N, S, rho, rho2, U = U, details = TRUE))
rdca_optcond_sU(H_counts, S, rho, x$s, U) # TRUE

U <- 2
(x <- dca(n, H_counts, N, S, rho, rho2, U = U, details = TRUE))
rdca_optcond_sU(H_counts, S, rho, x$s, U) # FALSE

U <- 3
```

```
(x <- dca(n, H_counts, N, S, rho, rho2, U = U, details = TRUE))
rdca_optcond_sU(H_counts, S, rho, x$s, U) # TRUE

U <- 1:2 # domain 2 blocked
(x <- dca(n, H_counts, N, S, rho, rho2, U = U, details = TRUE))
rdca_optcond_sU(H_counts, S, rho, x$s, U) # no unblocked strata in `U`

## End(Not run)
```

`data` *Datasets with Example Populations*

Description

[Stable]

Example datasets containing artificial populations for testing and demonstrating optimum sample allocation algorithms.

Usage

```
pop10s_bounds_ucost

pop507s_ucost

pop969s_ucost

pop2d4s

pop9d278s
```

Format

pop10s_bounds_ucost: Population with 10 strata, lower and upper bounds on sample sizes, and associated surveying costs. A matrix with 10 rows and 5 variables:

N stratum size

S standard deviation of study variable in the stratum.

m lower bound for sample size in the stratum.

M upper bound for sample size in the stratum.

unit_cost cost of surveying one element in the stratum.

pop507s_ucost: Population with 507 strata and associated surveying costs. A matrix with 507 rows and 3 columns:

N stratum size.

S standard deviation of study variable in the stratum.

unit_cost cost of surveying one element in the stratum.

pop969s_ucost: Population with 969 strata and associated surveying costs. A matrix with 969 rows and 3 columns:

N stratum size.

S standard deviation of study variable in the stratum.

unit_cost cost of surveying one element in the stratum.

pop2d4s: Population with 2 domains and 4 strata. A list with the following elements:

H_counts strata counts in each domain.

N stratum sizes.

S standard deviations of study variable in strata.

total totals in domains, i.e., the sum of the study variable values for population elements in each domain.

kappa priority weights for domains.

rho `total * sqrt(kappa)`.

rho2 `total^2 * kappa`.

n_max See `dca_nmax()` or `dca()`.

pop9d278s: Population with 9 domains and 278 strata. A list with the following elements:

H_counts strata counts in each domain.

N stratum sizes.

S standard deviations of study variable in strata.

total totals in domains, i.e., the sum of the study variable values for population elements in each domain.

kappa priority weights for domains.

rho `total * sqrt(kappa)`.

rho2 `total^2 * kappa`.

n_max See `dca_nmax()` or `dca()`.

Description

[Stable]

Functions implementing the Domain-Controlled Allocation (DCA) algorithm described in Wesołowski (2019) and Wójciak (2026). The algorithm solves the following optimum allocation problem, formulated in mathematical optimization terms:

Minimize

$$f(T, \boldsymbol{x}) = T$$

over $\mathbb{R} \times \mathbb{R}_+^{|\mathcal{H}|}$, subject to

$$\sum_{(d,h)\in\mathcal{H}} x_{d,h} = n,$$

$$\sum_{h\in\mathcal{H}_d} \left(\frac{1}{x_{d,h}} - \frac{1}{N_{d,h}}\right)\frac{N_{d,h}^2 S_{d,h}^2}{\rho_d^2} = T, \qquad d \in \mathcal{D},$$

where:

$(T, \boldsymbol{x}) = (T, (x_{d,h}, (d,h) \in \mathcal{H}))$ the optimization variable,

$\mathcal{H} \subset \mathbb{N}^2$ the set of domain-stratum indices,

$\mathcal{D} := \{d \in \mathbb{N}\colon\ \exists h, (d,h) \in \mathcal{H}\}$ the set of domain indices,

$\mathcal{H}_d := \{h \in \mathbb{N}\colon\ (d,h) \in \mathcal{H}\}$ the set of strata indices in domain d,

$N_{d,h} > 0$ size of stratum (d,h),

$S_{d,h} > 0$ standard deviation of the study variable in stratum (d,h),

$\rho_d := t_d \sqrt{\kappa_d}$ where t_d denotes the total in domain d, i.e., the sum of the values of the study variable for population elements in domain d, and κ_d is a priority weight for domain d,

$n \in (0, \sum_{(d,h)\in\mathcal{H}} N_{d,h}]$ total sample size.

Usage

```
dca0(n, H_counts, N, S, rho, rho2, details = FALSE)

dca(n, H_counts, N, S, rho, rho2, U = NULL, details = FALSE)

dca_nmax(H_counts, N, S)
```

Arguments

`n`	(`integerish(1)`) total sample size n. Must satisfy `0 < n <= sum(N)`.
`H_counts`	(`integerish`) strata counts in each domain.
`N`	(`integerish`) strata sizes $(N_{d,h}, (d,h) \in \mathcal{H})$.
`S`	(`numeric`) standard deviations $(S_{d,h}, (d,h) \in \mathcal{H})$ of surveyed variable in strata.

`rho` (`numeric`)
parameters $(\rho_d,\ d \in \mathcal{D})$ of the optimization problem.

`rho2` (`numeric`)
the square of `rho` (`rho^2`), provided to reduce potential loss of precision due to finite-precision arithmetic.

`details` (`logical(1)`)
whether to produce detailed debug output.

`U` (`integerish` or `NULL`)
a vector of indices identifying the *take-max* strata, i.e., the strata (d, h) for which the allocation is fixed to $x_{d,h} = N_{d,h}$. The indices refer to the positions of strata in the set $\mathcal{H}$, in the same order as in the input vectors (`N`, `S`, etc.).
For example, if $\mathcal{H} = \{(1,1),\ (2,1)\}$ and stratum $(2,1)$ is a take-max stratum, then `U = 2`.
If `U` contains all strata from a domain, the dimension of the D matrix is reduced accordingly.
`U` must satisfy one of the following conditions:

- `n > sum(N[U])`,
- `n = sum(N[U])` and `n = sum(N)`.

Details

For $n \in (0,\ n_{max})$, the optimal value satisfies $T^* > 0$, where

$$n_{max} := \sum_{d \in \mathcal{D}} \frac{\left(\sum_{h \in \mathcal{H}_d} N_{d,h} S_{d,h}\right)^2}{\sum_{h \in \mathcal{H}_d} N_{d,h} S_{d,h}^2}.$$

See Proposition 2.1 in Wesołowski (2019) or Wójciak (2026) for details. The value n_{max} is less than or equal to `sum(N)` and can be computed with `dca_nmax()`.

Value

If `details = FALSE`, the optimal $\boldsymbol{x}^*$ is returned. Otherwise, a list is returned containing the optimal $\boldsymbol{x}^*$ (element named `x`) along with other internal details of this algorithm. In particular, the `lambda` element of the list corresponds to the optimal T^*.

Functions

- `dca0()`: Domain-Controlled Allocation algorithm by Wesołowski (2019)
- `dca()`: Domain-Controlled Allocation algorithm by Wesołowski (2019), optionally using a set of take-max strata as described in Wójciak (2026).
- `dca_nmax()`: Computes the maximum total sample size n_{max} such that the optimization problem solved by the Domain-Controlled Allocation (DCA) algorithm admits a strictly positive optimal value T^*.

Note

These functions are optimized for internal use and should typically not be called directly by users. They are designed to handle a large number of invocations, specifically recursive calls from `rdca()`, and, as a result, parameter assertions are minimal.

References

Wójciak W (2026). *Multi-Domain Optimum Sample Allocation with Controlled-Precision under Upper-Bound Constraints*. Ph.D. thesis, Warsaw University of Technology. http://home.elka.pw.edu.pl/~wwojciak/phd_wwojciech_optimum_alloc.pdf.

Wesołowski J (2019). "Multi-domain Neyman-Tchuprov optimal allocation." *Statistics in Transition new series*, **20**(4), 1–12. doi:10.21307/stattrans2019031.

Wesołowski J, Wieczorkowski R (2017). "An eigenproblem approach to optimal equal-precision sample allocation in subpopulations." *Communications in Statistics - Theory and Methods*, **46**(5), 2212–2231. doi:10.1080/03610926.2015.1040501.

See Also

rdca()

Examples

```
# Two domains with 1 and 3 strata, respectively,
# that is, H = {(1,1), (2,1), (2,2), (2,3)}.
H_counts <- c(1, 3)
N <- c(140, 110, 135, 190) # (N_{1,1}, N_{2,1}, N_{2,2}, N_{2,3})
S <- sqrt(c(180, 20, 5, 4)) # (S_{1,1}, S_{2,1}, S_{2,2}, S_{2,3})
total <- c(2, 3)
kappa <- c(0.4, 0.6)
rho <- total * sqrt(kappa) # (rho_1, rho_2)
rho2 <- total^2 * kappa
sum(N) # 575
n_max <- dca_nmax(H_counts, N, S) # 519.0416

n <- floor(n_max) - 1

dca0(n, H_counts, N, S, rho, rho2)
x0 <- dca0(n, H_counts, N, S, rho, rho2, details = TRUE)
x0$x
x0$lambda
x0$k
x0$v
x0$s

n <- ceiling(n_max) + 1
x0 <- dca0(n, H_counts, N, S, rho, rho2, details = TRUE)
x0$x
x0$lambda

n <- floor(n_max) - 1

x1 <- dca(n, H_counts, N, S, rho, rho2, details = TRUE)
x1$x
x1$x_Uc
x1$lambda
x1$s
```

```
dca(n, H_counts, N, S, rho, rho2, U = 1)
x2 <- dca(n, H_counts, N, S, rho, rho2, U = 1, details = TRUE)
x2$x
x2$x_Uc
x2$lambda
x2$s
```

---

dca_M *DCA Algorithm for Upper-Bound Constrained Allocations (M <= N)*

Description

[Experimental]

Prototype (under testing).

Usage

```
dca_M(n, H_counts, N, S, rho, rho2, M = N, U = NULL)
```

Arguments

n	(integerish(1)) total sample size n. Must satisfy `0 < n <= sum(N)`.
H_counts	(integerish) strata counts in each domain.
N	(integerish) strata sizes $(N_{d,h},\ (d,h) \in \mathcal{H})$.
S	(numeric) standard deviations $(S_{d,h},\ (d,h) \in \mathcal{H})$ of surveyed variable in strata.
rho	(numeric) parameters $(\rho_d,\ d \in \mathcal{D})$ of the optimization problem.
rho2	(numeric) the square of rho (`rho^2`), provided to reduce potential loss of precision due to finite-precision arithmetic.
M	(integerish) upper bounds on sample sizes in strata. Defaults to N.
U	(integerish or NULL) a vector of indices identifying the *take-max* strata, i.e., the strata (d,h) for which the allocation is fixed to $x_{d,h} = N_{d,h}$. The indices refer to the positions of strata in the set $\mathcal{H}$, in the same order as in the input vectors (N, S, etc.). For example, if $\mathcal{H} = \{(1,1),\ (2,1)\}$ and stratum $(2,1)$ is a take-max stratum, then `U = 2`. If U contains all strata from a domain, the dimension of the D matrix is reduced accordingly. U must satisfy one of the following conditions:

- n > sum(N[U]),
- n = sum(N[U]) and n = sum(N).

Examples

```
H_counts <- c(5, 2)
H_names <- rep(seq_along(H_counts), times = H_counts)
S <- c(154, 178, 134, 213, 124, 102, 12)
N <- c(100, 100, 100, 100, 100, 100, 100)
M <- c(80, 90, 70, 40, 10, 90, 100)
names(M) <- names(N) <- H_names
total <- c(13, 2)
kappa <- c(0.8, 0.2)
n <- 150

# experimental function (not exported) - examples skipped
## Not run:
dca_M(n, H_counts, N, S, total, kappa, M = M, U = 5)
#         1         1         1         1         1         2         2
# 12.754880 14.742653 11.098402 17.641490 10.000000 74.945462  8.817113

## End(Not run)
```

---

dopt *Multi-Domain Optimum Sample Allocation with Controlled-Precision under Upper-Bound Constraints*

Description

[Stable]

Computes the optimum allocation for the following multi-domain optimum allocation problem, formulated in mathematical optimization terms:

Minimize

$$f(T,\, \boldsymbol{x}) = T$$

over $\mathbb{R} \times \mathbb{R}_{+}^{|\mathcal{H}|}$, subject to

$$\sum_{(d,h)\in\mathcal{H}} x_{d,h} = n,$$

$$\sum_{h\in\mathcal{H}_d} \left(\frac{1}{x_{d,h}} - \frac{1}{N_{d,h}}\right)\frac{N_{d,h}^2 S_{d,h}^2}{\rho_d^2} = T, \qquad d \in \mathcal{D},$$

$$x_{d,h} \le N_{d,h}, \qquad (d,h) \in \mathcal{H},$$

where:

$(T,\, \boldsymbol{x}) = (T,\, (x_{d,h},\, (d,h) \in \mathcal{H}))$ the optimization variable,

$\mathcal{H} \subset \mathbb{N}^2$ the set of domain-stratum indices,

$\mathcal{D} := \{d \in \mathbb{N}\colon\ \exists h,\ (d,h) \in \mathcal{H}\}$ the set of domain indices,

$\mathcal{H}_d := \{h \in \mathbb{N}\colon\ (d,h) \in \mathcal{H}\}$ the set of strata indices in domain d,

$N_{d,h} > 0$ size of stratum (d,h),

$S_{d,h} > 0$ standard deviation of the study variable in stratum (d,h),

$\rho_d := t_d \sqrt{\kappa_d}$ where t_d denotes the total in domain d, i.e., the sum of the values of the study variable for population elements in domain d, and κ_d is a priority weight for domain d,

$n \in (0,\ \sum_{(d,h)\in\mathcal{H}} N_{d,h}]$ total sample size.

Usage

```
dopt(n, H_counts, N, S, total, kappa, return_T = FALSE)
```

Arguments

`n`	(`integerish(1)`) total sample size n. Must satisfy `0 < n <= sum(N)`.
`H_counts`	(`integerish`) strata counts in each domain.
`N`	(`integerish`) strata sizes $(N_{d,h},\ (d,h) \in \mathcal{H})$.
`S`	(`numeric`) standard deviations $(S_{d,h},\ (d,h) \in \mathcal{H})$ of surveyed variable in strata.
`total`	(`numeric`) vector of domain totals, $t_d,\ d \in \mathcal{D}$, i.e., the sum of the study variable over all population elements in each domain.
`kappa`	(`numeric`) vector of priority weights for the domains, $\kappa_d,\ d \in \mathcal{D}$.
`return_T`	(`logical(1)`) If `TRUE`, the function returns a list containing the optimal allocation and the optimal value of the objective function T. If `FALSE` (default), only the optimal allocation vector is returned.

Details

The `dopt()` function uses the RDCA algorithm implemented in `rdca()`.

Value

If `return_T = FALSE` (default), a numeric vector containing the optimal sample allocations $x_{d,h}$ for each stratum $(d,h) \in \mathcal{H}$.

If `return_T = TRUE`, a list with components:

xopt numeric vector of optimal sample allocations.

Topt optimal value of the objective function T.

References

Wójciak W (2026). *Multi-Domain Optimum Sample Allocation with Controlled-Precision under Upper-Bound Constraints*. Ph.D. thesis, Warsaw University of Technology. http://home.elka.pw.edu.pl/~wwojciak/phd_wwojciech_optimum_alloc.pdf.

See Also

rdca(), dca(), dca_nmax(), opt(), optcost().

Examples

```
# Three domains with 2, 2, and 3 strata, respectively,
# that is, H = {(1,1), (1,2), (2,1), (2,2), (3,1), (3,2), (3,3)}.
H_counts <- c(2, 2, 3)
# (N_{1,1}, N_{1,2}, N_{2,1}, N_{2,2}, N_{3,1}, N_{3,2}, N_{3,3})
N <- c(140, 110, 135, 190, 200, 40, 70)
# (S_{1,1}, S_{1,2}, S_{2,1}, S_{2,2}, S_{3,1}, S_{3,2}, S_{3,3})
S <- c(180, 20, 5, 4, 35, 9, 40)
total <- c(2, 3, 5)
kappa <- c(0.5, 0.2, 0.3)
n <- 828

# Optimum allocation.
dopt(n, H_counts, N, S, total, kappa)

# Example population with 9 domains and 278 strata
p <- pop9d278s
sum(p$N)
n <- 5000
x <- dopt(n, p$H_counts, p$N, p$S, p$total, p$kappa, return_T = TRUE)
x
all(x$xopt <= p$N)
sum(x$xopt)
```

---

fpia *Fixed-Point Iteration Algorithm*

Description

[Experimental]

Algorithm for optimum sample allocation in stratified sampling under lower- and upper-bound constraints, based on fixed-point iteration.

Usage

```
fpia(
  n,
  Ah,
  mh = NULL,
  Mh = NULL,
  lambda0 = NULL,
  maxiter = 100,
  tol = .Machine$double.eps * 1000
)

fpia2(v0, Nh, Sh, mh = NULL, Mh = NULL, lambda0 = NULL, maxiter = 100)

glambda(lambda, n, Ah, mh = NULL, Mh = NULL)

philambda(lambda, n, Ah, mh = NULL, Mh = NULL)
```

Arguments

`n`	(`integerish(1)`) total sample size.
`Ah`	(`numeric`) products of population stratum sizes and standard deviations of the study variable, $A_h = N_h S_h$.
`mh`	(`numeric` or `NULL`) lower bounds on stratum sample sizes (optional).
`Mh`	(`numeric` or `NULL`) upper bounds on stratum sample sizes (optional).
`lambda0`	(`numeric(1)`) initial value of the parameter λ (optional).
`maxiter`	(`integerish(1)`) maximum number of iterations.
`tol`	(`numeric(1)`) desired convergence tolerance.
`v0`	variance
`Nh`	(`numeric`) population sizes in strata.
`Sh`	(`numeric`) standard deviations of the study variable in strata.
`lambda`	(`numeric(1)`) λ.

Value

A list with elements:

nh Vector of optimal allocation sizes.

iter Number of iterations performed.

Functions

- `fpia()`:
- `fpia2()`: Variant of `fpia()` using variance-based parametrization.
- `glambda()`: Helper function for the `fpia()`
- `philambda()`: Helper function for the `fpia()`.

References

Münnich RT, Sachs EW, Wagner M (2012). "Numerical solution of optimal allocation problems in stratified sampling under box constraints." *AStA Advances in Statistical Analysis*, **96**(3), 435–450. doi:10.1007/s101820110176z.

`has_mixed_signs` *Check for Mixed Signs in a Numeric Vector*

Description

[Stable]

Determines whether a numeric vector contains both negative and positive values. Zero (`0`) is treated as neutral and does not count as either sign.

Usage

```
has_mixed_signs(x)
```

Arguments

`x`	(`numeric`) a vector to check.

Value

`TRUE` if the vector contains both positive and negative values, `FALSE` otherwise.

Examples

```
# internal functions (not exported) - examples skipped
## Not run:
has_mixed_signs(1:5)
has_mixed_signs(-(1:5))
has_mixed_signs(c(-1, -2, 3))
has_mixed_signs(c(0, -1))
has_mixed_signs(c(0, 1))
```

```
has_mixed_signs(c(0, 1, -1))

## End(Not run)
```

---

helpers_dca_rdca *Internal Helper Functions for* dca0(), dca(), *and* rdca()

Description

[Stable]

Internal utility functions used by dca0(), dca(), and rdca() that perform operations on sets of domain–strata indices and manage the mapping between strata and domains.

Usage

```
H_cnt2dind(H_counts)

H_cnt2glbidx(H_counts)

H_get_strata_indices(H_counts, d)
```

Arguments

H_counts	(integerish) strata counts in each domain.
d	(integerish(1)) domain index. Must satisfy 0 < d <= length(H_counts).

Functions

- H_cnt2dind(): Creates a vector of domain indicators from a vector of strata counts per domain; each element of the vector is the index of the domain to which the corresponding stratum belongs.
- H_cnt2glbidx(): Creates unique indices for strata across multiple domains. Returns a list of integer vectors, where the d-th element contains the unique indices of the strata in domain d.
- H_get_strata_indices(): Get the globally unique indices of strata belonging to a specific domain.

Note

These functions are internal and should typically not be called directly by users.

Examples

```
H_counts <- c(2, 2, 3) # three domains with 2, 2, and 3 strata respectively

# internal functions (not exported) - examples skipped
## Not run:
H_cnt2dind(H_counts) # 1 1 2 2 3 3 3

## End(Not run)

# internal functions (not exported) - examples skipped
## Not run:
H_cnt2glbidx(H_counts)

## End(Not run)

# internal functions (not exported) - examples skipped
## Not run:
H_get_strata_indices(H_counts, 3) # 5 6 7

## End(Not run)
```

---

is_equal *Check Numeric Equality within a Tolerance*

Description

[Stable]

Compares two numeric vectors element-wise using an adaptive tolerance sequence ranging from `10^-19` to `10^tol_max`. The smallest tolerance at which the values are considered equal is returned as the corresponding name in the output.

Usage

```
is_equal(x, y, tol_max = -1)
```

Arguments

`x`	(`numeric`) first vector to compare.
`y`	(`numeric`) second vector to compare.
`tol_max`	(`integerish(1)`) the maximum tolerance exponent (as a power of 10).

Value

A logical vector indicating whether each element pair is equal within the detected tolerance. The names reflect the tolerance used.

Examples

```
# internal functions (not exported) - examples skipped
## Not run:
is_equal(c(3, 4), c(3, 4))
is_equal(c(3, 4), c(3.01, 4.11))
is_equal(c(3, 4), c(3.01, 4.11), tol_max = 0)

## End(Not run)
```

`obj_emptiness` *Object Emptiness (*`NULL` *or zero-length)*

Description

[Stable]

Utility functions for checking whether an object is empty, where emptiness is defined as being `NULL` or having length 0.

Usage

```
is_empty(x)

is_nonempty(x)
```

Arguments

`x`	object to test.

Functions

- `is_empty()`: Returns `TRUE` if the object is `NULL` or has length 0, and `FALSE` otherwise.
- `is_nonempty()`: Logical negation of `is_empty()`. This function directly checks if `length(x) > 0L` for performance reasons, avoiding the extra negation step that would occur if using `!is_empty(x)`. It is optimized for repeated use in algorithms where `is_nonempty()` is called many times.

Examples

```
# internal functions (not exported) - examples skipped

## Not run:
is_empty(NULL)
is_empty(character(0))
is_empty(1)

## End(Not run)
```

```
## Not run:
is_nonempty(NULL)
is_nonempty(character(0))
is_nonempty(1)

## End(Not run)
```

---

opt *Optimum Sample Allocation in Stratified Sampling*

Description

[Stable]

Computes the optimum allocation for the following optimum allocation problem, formulated in mathematical optimization terms:

Minimize

$$f(x_1, \ldots, x_H) = \sum_{h=1}^{H} \frac{A_h^2}{x_h}$$

over $\mathbb{R}_+^H$, subject to

$$\sum_{h=1}^{H} x_h = n,$$

$$m_h \leq x_h \leq M_h, \qquad h = 1, \ldots, H,$$

where $n > 0$, $A_h > 0$, $m_h > 0$, $M_h > 0$, such that $m_h < M_h$, $h = 1, \ldots, H$, and $\sum_{h=1}^{H} m_h \leq n \leq \sum_{h=1}^{H} M_h$, are given numbers. Inequality constraints are optional and may be omitted.

Inequality constraints are optional, and the user can choose whether and how they are applied to the optimization problem. This is controlled using the `m` and `M` arguments as follows:

- **No inequality constraints:** both `m` and `M` must be `NULL` (default).
- **Lower bounds only** ($m_1, \ldots, m_H$)**:** specify `m`, and set `M = NULL`.
- **Upper bounds only** ($M_1, \ldots, M_H$)**:** specify `M`, and set `m = NULL`.
- **Box constraints** ($m_h, M_h, h = 1, \ldots, H$)**:** specify both `m` and `M`.

Usage

```
opt(n, A, m = NULL, M = NULL, M_algorithm = "rna")
```

Arguments

n	(integerish(1)) total sample size. Must satisfy n > 0. Additionally: • If bounds_inner is not NULL, then n >= sum(bounds_inner) when bounds_inner are treated as lower bounds, or n <= sum(bounds_inner) when treated as upper bounds. • If bounds_outer is not NULL, then n >= sum(bounds_outer) when bounds_outer are treated as lower bounds, or n <= sum(bounds_outer) when treated as upper bounds.
A	(numeric) population constants $A_1, \ldots, A_H$. All values must be strictly positive.
m	(numeric or NULL) optional lower bounds $m_1, \ldots, m_H$ for the stratum sample sizes. If no lower bounds are desired, set m = NULL. If M is not NULL, it is required that $m_h < M_h$ for all strata.
M	(numeric or NULL) optional upper bounds $M_1, \ldots, M_H$ for the stratum sample sizes. If no upper bounds are desired, set M = NULL. If m is not NULL, it is required that $m_h < M_h$ for all strata.
M_algorithm	(string) Name of the algorithm to use for computing the sample allocation when only upper-bound constraints are imposed. Must be one of "rna" (default), "sga", "sgaplus", or "coma". This parameter is used only when $H > 1$ and n < sum(M).

Details

The opt() function uses different allocation algorithms depending on which inequality constraints are applied. Each algorithm is implemented in a separate R function, which is generally **not intended to be called directly** by the end user. The algorithms are:

- **Lower bounds only** ($m_1, \ldots, m_H$)**:**
  - LRNA - rna()
- **Upper bounds only** ($M_1, \ldots, M_H$)**:**
  - RNA - rna()
  - SGA - sga()
  - SGAPLUS - sgaplus()
  - COMA - coma()
- **Box constraints** ($m_h, M_h, h = 1, \ldots, H$)**:**
  - RNABOX - rnabox()

See the documentation of each specific function for more details about the corresponding algorithm.

Value

A numeric vector of the optimal sample allocations for each stratum.

Note

If no inequality constraints are applied, the allocation follows the Neyman allocation:

$$x_h = A_h \frac{n}{\sum_{i=1}^{H} A_i}, \quad h = 1, \ldots, H.$$

For a **stratified π estimator** of the population total using stratified simple random sampling without replacement design, the objective function parameters A_h are:

$$A_h = N_h S_h, \quad h = 1, \ldots, H,$$

where N_h is the size of stratum h and S_h is the standard deviation of the study variable in stratum h.

References

Särndal C, Swensson B, Wretman J (1992). *Model Assisted Survey Sampling*. Springer New York, NY. ISBN 978-0-387-40620-6.

See Also

optcost(), rna(), sga(), sgaplus(), coma(), rnabox().

Examples

```
A <- c(3000, 4000, 5000, 2000)
m <- c(100, 90, 70, 50)
M <- c(300, 400, 200, 90)

# One-sided lower bounds.
opt(n = 340, A = A, m = m)
opt(n = 400, A = A, m = m)
opt(n = 700, A = A, m = m)

# One-sided upper bounds.
opt(n = 190, A = A, M = M)
opt(n = 700, A = A, M = M)

# Box-constraints.
opt(n = 340, A = A, m = m, M = M)
opt(n = 500, A = A, m = m, M = M)
x <- opt(n = 800, A = A, m = m, M = M)
x

# Variance corresponding to the allocation x.
var_st(x = x, A = A, A0 = 45000)

# Execution-time comparison of different algorithms using the microbenchmark package.
## Not run:
N <- pop969s_ucost[, "N"]
S <- pop969s_ucost[, "S"]
A <- N * S
```

```
nfrac <- c(0.005, seq(0.05, 0.95, 0.05))
n <- setNames(as.integer(nfrac * sum(N)), nfrac)
lapply(
  n,
  function(ni) {
    microbenchmark::microbenchmark(
      RNA = opt(ni, A, M = N, M_algorithm = "rna"),
      SGA = opt(ni, A, M = N, M_algorithm = "sga"),
      SGAPLUS = opt(ni, A, M = N, M_algorithm = "sgaplus"),
      COMA = opt(ni, A, M = N, M_algorithm = "coma"),
      times = 200,
      unit = "us"
    )
  }
)

## End(Not run)
```

`optcost` *Minimum-Cost Allocation in Stratified Sampling*

Description

[Stable]

Computes stratum sample sizes that minimize the total survey cost for a given target variance of a stratified estimator, optionally subject to one-sided upper bounds on the stratum sample sizes. Specifically, the function solves the following optimization problem:

Minimize

$$c(x_1, \ldots, x_H) = \sum_{h=1}^{H} c_h x_h$$

over $\mathbb{R}_+^H$, subject to

$$\sum_{h=1}^{H} \frac{A_h^2}{x_h} - A_0 = V,$$

$$x_h \le M_h, \qquad h = 1, \ldots, H,$$

where A_0, $A_h > 0$, $c_h > 0$, $M_h > 0$, $h = 1, \ldots, H$, and $V > \sum_{h=1}^{H} \frac{A_h^2}{M_h} - A_0$, are given numbers.

The upper-bound constraints $x_h \le M_h$ are optional. If they are not imposed, it is only required that $V > 0$.

Usage

```
optcost(V, A, A0, M = NULL, unit_costs = 1)
```

Arguments

`V`	(`number`) parameter V in the variance constraint. If upper bounds are imposed (`M` is not `NULL`), it must satisfy `V > sum(A^2/M) - A_0`. Otherwise, `V > 0`.
`A`	(`numeric`) population constants $A_1, \ldots, A_H$. All values must be strictly positive.
`A0`	(`number`) population constant A_0.
`M`	(`numeric` or `NULL`) optional upper bounds $M_1, \ldots, M_H$ on the stratum sample sizes. If no upper bounds are imposed, set `M = NULL`.
`unit_costs`	(`numeric`) costs $c_1, \ldots, c_H$ of surveying one element in each stratum. Strictly positive values. May also be of length 1, in which case the value is recycled to match the length of `bounds`.

Details

The allocation is computed using the **LRNA algorithm**, described in Wójciak (2023).

The solution is valid for stratified sampling designs in which the variance V_{st} of the stratified estimator can be expressed as

$$V_{st} = \sum_{h=1}^{H} \frac{A_h^2}{x_h} - A_0,$$

where H is the number of strata, $x_1, \ldots, x_H$ are the stratum sample sizes, and $A_0,\ A_h > 0$ do not depend on x_h.

Value

A numeric vector containing the optimal sample allocation for each stratum.

Note

For the *stratified π-estimator* of the population total under *stratified simple random sampling without replacement* design, the parameters take the form

$$A_h = N_h S_h, \qquad h = 1, \ldots, H,$$

$$A_0 = \sum_{h=1}^{H} N_h S_h^2,$$

where N_h is the size of stratum h and S_h is the standard deviation of the study variable in stratum h.

References

Wójciak W (2023). "Another Solution for Some Optimum Allocation Problem." *Statistics in Transition new series*, **24**(5), 203-219. doi:10.59170/stattrans2023071.

See Also

rna(), opt().

Examples

```
A <- c(3000, 4000, 5000, 2000)
M <- c(100, 90, 70, 80)
x <- optcost(1017579, A = A, A0 = 579, M = M)
x
```

---

rdca *Recursive Domain-Controlled Allocation (RDCA) Algorithm*

Description

[Stable]

Implements the Recursive Domain-Controlled Allocation (RDCA) algorithm described in Wójciak (2026). The algorithm solves the following optimum allocation problem, formulated in mathematical optimization terms:

Minimize

$$f(T,\, \boldsymbol{x}) = T$$

over $\mathbb{R} \times \mathbb{R}_+^{|\mathcal{H}|}$, subject to

$$\sum_{(d,h)\in\mathcal{H}} x_{d,h} = n,$$

$$\sum_{h\in\mathcal{H}_d} \left(\frac{1}{x_{d,h}} - \frac{1}{N_{d,h}}\right)\frac{N_{d,h}^2 S_{d,h}^2}{\rho_d^2} = T, \qquad d \in \mathcal{D},$$

$$x_{d,h} \le N_{d,h}, \qquad (d,h) \in \mathcal{H},$$

where:

$(T,\, \boldsymbol{x}) = (T,\, (x_{d,h},\, (d,h) \in \mathcal{H}))$ the optimization variable,

$\mathcal{H} \subset \mathbb{N}^2$ the set of domain-stratum indices,

$\mathcal{D} := \{d \in \mathbb{N}\colon\ \exists h,\, (d,h) \in \mathcal{H}\}$ the set of domain indices,

$\mathcal{H}_d := \{h \in \mathbb{N}\colon\ (d,h) \in \mathcal{H}\}$ the set of strata indices in domain d,

$N_{d,h} > 0$ size of stratum (d,h),

$S_{d,h} > 0$ standard deviation of the study variable in stratum (d,h),

$\rho_d := t_d \sqrt{\kappa_d}$ where t_d denotes the total in domain d, i.e., the sum of the values of the study variable for population elements in domain d, and κ_d is a priority weight for domain d,

$n \in (0,\, \sum_{(d,h)\in\mathcal{H}} N_{d,h}]$ total sample size.

Usage

```
rdca(n, H_counts, N, S, rho, rho2 = rho^2, U = NULL, J = NULL)
```

Arguments

`n` (`integerish(1)`)
total sample size n. Must satisfy `0 < n <= sum(N)`.

`H_counts` (`integerish`)
strata counts in each domain.

`N` (`integerish`)
strata sizes $(N_{d,h},\ (d,h) \in \mathcal{H})$.

`S` (`numeric`)
standard deviations $(S_{d,h},\ (d,h) \in \mathcal{H})$ of surveyed variable in strata.

`rho` (`numeric`)
parameters $(\rho_d,\ d \in \mathcal{D})$ of the optimization problem.

`rho2` (`numeric`)
the square of `rho` (`rho^2`), provided to reduce potential loss of precision due to finite-precision arithmetic.

`U` (`integerish` or `NULL`)
a vector of indices identifying the *take-max* strata, i.e., the strata (d,h) for which the allocation is fixed to $x_{d,h} = N_{d,h}$. The indices refer to the positions of strata in the set $\mathcal{H}$, in the same order as in the input vectors (`N`, `S`, etc.).

For example, if $\mathcal{H} = \{(1,1),\ (2,1)\}$ and stratum $(2,1)$ is a take-max stratum, then `U = 2`.

If `U` contains all strata from a domain, the dimension of the D matrix is reduced accordingly.

`U` must satisfy one of the following conditions:

- `n > sum(N[U])`,
- `n = sum(N[U])` and `n = sum(N)`.

`J` (`integerish` or `NULL`)
vector of domain indices, subset of $\mathcal{D}$. Specifies the domains for which allocated sample sizes must not exceed the corresponding strata sizes `N`. For domains not included in `J`, allocations may exceed strata sizes. Must not be empty. If `J` is `NULL`, it is treated as containing all domains. `U` must not contain any strata from domains included in `J`.

Details

The upper-bound constraints $x_{d,h} \leq N_{d,h}$ are guaranteed to be preserved only if domain d is in `J`. The parameter `J` is used in the recursion. The specified optimization problem is solved when `J = NULL`, i.e., when `J` contains all domains.

Note

This function is optimized for internal use and should typically not be called directly by users. It is designed to handle a large number of invocations, specifically recursive invocations of `rdca()`, and, as a result, parameter assertions are minimal.

References

Wójciak W (2026). *Multi-Domain Optimum Sample Allocation with Controlled-Precision under Upper-Bound Constraints*. Ph.D. thesis, Warsaw University of Technology. http://home.elka.pw.edu.pl/~wwojciak/phd_wwojciech_optimum_alloc.pdf.

See Also

dca()

Examples

```
# Three domains with 2, 2, and 3 strata, respectively,
# that is, H = {(1,1), (1,2), (2,1), (2,2), (3,1), (3,2), (3,3)}.
H_counts <- c(2, 2, 3)
# (N_{1,1}, N_{1,2}, N_{2,1}, N_{2,2}, N_{3,1}, N_{3,2}, N_{3,3})
N <- c(140, 110, 135, 190, 200, 40, 70)
# (S_{1,1}, S_{1,2}, S_{2,1}, S_{2,2}, S_{3,1}, S_{3,2}, S_{3,3})
S <- c(180, 20, 5, 4, 35, 9, 40)
total <- c(2, 3, 5)
kappa <- c(0.5, 0.2, 0.3)
rho <- total * sqrt(kappa) # (rho_1, rho_2, rho_3)
rho2 <- total^2 * kappa
sum(N)
n <- 828

# Optimum allocation.
rdca(n, H_counts, N, S, rho, rho2)

# Upper bounds enforced only for domain 1.
rdca(n, H_counts, N, S, rho, rho2, J = 1)
```

---

rdca_iter *Iterative RDCA Implementation*

Description

[Experimental]

Iterative implementation of the Recursive Domain-Controlled Allocation (RDCA) algorithm. Not tested.

Usage

```
rdca_iter(n, H_counts, N, S, rho, rho2 = NULL, ref_domain = 1L)
```

Arguments

n	(integerish(1)) total sample size n. Must satisfy `0 < n <= sum(N)`.
H_counts	(integerish) strata counts in each domain.
N	(integerish) strata sizes $(N_{d,h},\ (d,h) \in \mathcal{H})$.
S	(numeric) standard deviations $(S_{d,h},\ (d,h) \in \mathcal{H})$ of surveyed variable in strata.
rho	(numeric) parameters $(\rho_d,\ d \in \mathcal{D})$ of the optimization problem.
rho2	(numeric) the square of `rho` (`rho^2`), provided to reduce potential loss of precision due to finite-precision arithmetic.
ref_domain	(integerish(1)) reference domain (denoted by j in the thesis).

Examples

```
H_counts <- c(2, 2, 3)
N <- c(140, 110, 135, 190, 200, 40, 70)
S <- sqrt(c(180, 20, 5, 4, 35, 9, 40))
total <- c(2, 3, 5)
kappa <- c(0.5, 0.2, 0.3)
rho <- total * sqrt(kappa)
(n <- dca_nmax(H_counts, N, S) - 1)

# experimental function (not exported) - examples skipped
## Not run:
rdca_iter(n, H_counts, N, S, rho)
# 140.0000 103.6139 132.1970 166.4127 195.9701  19.8750  70.0000

## End(Not run)
```

---

rnabox *Recursive Neyman Algorithm for Optimum Sample Allocation under Box Constraints (RNABOX)*

Description

[Stable]

Implements the Recursive Neyman Algorithm for Optimum Sample Allocation under Box Constraints (RNABOX), as proposed in Wesołowski et al. (2024). The algorithm solves the following optimum allocation problem, formulated in mathematical optimization terms:

Minimize

$$f(x_1, \ldots, x_H) = \sum_{h=1}^{H} \frac{A_h^2}{x_h}$$

over $\mathbb{R}_+^H$, subject to

$$\sum_{h=1}^{H} x_h = n,$$

$$m_h \leq x_h \leq M_h, \qquad h = 1, \ldots, H,$$

where $n > 0$, $A_h > 0$, $m_h > 0$, $M_h > 0$, such that $m_h < M_h$, $h = 1, \ldots, H$, and $\sum_{h=1}^{H} m_h \leq n \leq \sum_{h=1}^{H} M_h$, are given numbers. Inequality constraints are optional and may be omitted.

Usage

```
rnabox(
  n,
  A,
  bounds_inner = NULL,
  bounds_outer = NULL,
  cmp_inner = .Primitive(">="),
  cmp_outer = .Primitive("<=")
)
```

Arguments

`n`	(`integerish(1)`) total sample size. Must satisfy `n > 0`. Additionally: • If `bounds_inner` is not `NULL`, then `n >= sum(bounds_inner)` when `bounds_inner` are treated as lower bounds, or `n <= sum(bounds_inner)` when treated as upper bounds. • If `bounds_outer` is not `NULL`, then `n >= sum(bounds_outer)` when `bounds_outer` are treated as lower bounds, or `n <= sum(bounds_outer)` when treated as upper bounds.
`A`	(`numeric`) population constants $A_1, \ldots, A_H$. All values must be strictly positive.
`bounds_inner`	(`numeric` or `NULL`) optional bounds on sample sizes in strata for the **interim (inner) allocation** phase of `rnabox()`. These can be either lower bounds $m_1, \ldots, m_H$ or upper bounds $M_1, \ldots, M_H$, depending on the value of `cmp_inner`. If `bounds_inner` is `NULL`, no bounds are imposed during the interim allocation phase. If both `bounds_inner` and `bounds_outer` are not `NULL`, the following element-wise relationships must hold: • If `bounds_inner` are treated as lower bounds, then `bounds_inner < bounds_outer`. • If `bounds_inner` are treated as upper bounds, then `bounds_inner > bounds_outer`. `bounds_inner` is passed by `rnabox()` to `rna()` as the `bounds` argument, serving as an interim allocation step applied before enforcing the outer `bounds_outer`.

`bounds_outer` (`numeric` or `NULL`)
optional bounds on sample sizes in strata, checked during the **violation check (outer)** phase of the iterative `rnabox()` loop. These can be either lower bounds $m_1, \ldots, m_H$ or upper bounds $M_1, \ldots, M_H$, depending on the value of `cmp_outer`. If `bounds_outer` is `NULL`, no bounds are imposed during the outer phase.

If both `bounds_outer` and `bounds_inner` are not `NULL`, the following element-wise relationships must hold:

- If `bounds_outer` are treated as lower bounds, then `bounds_outer` < `bounds_inner`.
- If `bounds_outer` are treated as upper bounds, then `bounds_outer` > `bounds_inner`.

`cmp_inner` (`function`)
a binary comparison operator used to check for violations of `bounds_inner`. Must be either `.Primitive("<=")` or `.Primitive(">=")`. This operator determines how `bounds_inner` is handled:

- `.Primitive("<=")` treats `bounds_inner` as lower bounds and causes `rnabox()` to apply the LRNA algorithm as an inner allocation step.
- `.Primitive(">=")` treats `bounds_inner` as upper bounds and causes `rnabox()` to apply the RNA algorithm as an inner allocation step.

The value of this argument has no effect if `bounds_inner` is `NULL`.

`cmp_inner` is passed by `rnabox()` to `rna()` as the `cmp` argument, serving as an interim allocation step applied before enforcing the outer `bounds_outer`.

`cmp_outer` (`function`)
a binary comparison operator used to check for violations of `bounds_outer`. It must be the logical complement of `cmp_inner` (up to equality): if `cmp_inner` = `.Primitive("<=")`, then `cmp_outer` = `.Primitive(">=")`, and vice versa. It determines how `bounds_outer` is handled:

- `.Primitive("<=")` treats `bounds_outer` as lower bounds.
- `.Primitive(">=")` treats `bounds_outer` as upper bounds.

The value of this argument has no effect if `bounds_outer` is `NULL`.

`cmp_outer` is provided solely for computational efficiency, as its value is fully determined by `cmp_inner`.

Value

A numeric vector of optimum sample allocations in strata.

Note

The `rnabox()` function is optimized for internal use and should typically not be called directly by users. Use `opt()` instead.

References

Wesołowski J, Wieczorkowski R, Wójciak W (2024). "Recursive Neyman algorithm for optimum sample allocation under box constraints on sample sizes in strata." *Survey Methodology*, **50**(2), 487–511. ISSN 1492-0921, https://www150.statcan.gc.ca/n1/en/catalogue/12-001-X201200111682.

See Also

opt(), optcost(), rna(), sga(), sgaplus(), coma()

Examples

```
N <- c(454, 10, 116, 2500, 2240, 260, 39, 3000, 2500, 400)
S <- c(0.9, 5000, 32, 0.1, 3, 5, 300, 13, 20, 7)
A <- N * S
m <- c(322, 3, 57, 207, 715, 121, 9, 1246, 1095, 294) # lower bounds
M <- N # upper bounds

# Regular allocation.
n <- 6000
opt_regular <- rnabox(n, A, M, m)

# Vertex allocation.
n <- 4076
opt_vertex <- rnabox(n, A, M, m)
```

---

rna_experimental *RNA – Experimental Versions*

Description

[Experimental]

Experimental variants of the Recursive Neyman Algorithm (RNA).

Usage

```
rna_rec(
  total_cost,
  A,
  bounds = NULL,
  unit_costs = rep(1, length(A)),
  cmp = .Primitive(">=")
)

rna_prior(
  total_cost,
  A,
  bounds = NULL,
  check = NULL,
  cmp = .Primitive(">="),
  details = FALSE
)
```

Arguments

`total_cost` (`numeric(1)`)
total survey cost c. Must be strictly positive. Additionally:

- If one-sided lower bounds $m_1, \ldots, m_H$ are imposed, it is required that $c \geq \sum_{h=1}^{H} c_h m_h$, i.e. `total_cost >= sum(unit_costs * bounds)`.
- If one-sided upper bounds $M_1, \ldots, M_H$ are imposed, it is required that $c \leq \sum_{h=1}^{H} c_h M_h$, i.e. `total_cost <= sum(unit_costs * bounds)`.

`A` (`numeric`)
population constants $A_1, \ldots, A_H$. All values must be strictly positive.

`bounds` (`numeric` or `NULL`)
optional lower bounds $m_1, \ldots, m_H$, or upper bounds $M_1, \ldots, M_H$, or `NULL` to indicate that no inequality constraints are imposed. If not `NULL`, `bounds` is interpreted as:

- lower bounds, if `cmp = .Primitive("<=")`, or
- upper bounds, if `cmp = .Primitive(">=")`.

See also `total_cost`.

`unit_costs` (`numeric`)
costs $c_1, \ldots, c_H$ of surveying one element in each stratum. Strictly positive values. May also be of length 1, in which case the value is recycled to match the length of `bounds`.

`cmp` (`function`)
a binary comparison operator used to check for violations of `bounds`. Must be either `.Primitive("<=")` (treating `bounds` as lower bounds and invoking the LRNA algorithm) or `.Primitive(">=")` (treating `bounds` as upper bounds and invoking the RNA algorithm).

The value of this argument has no effect if `bounds` is `NULL`.

`check` (`integer`)
indices of strata for which allocation constraints may be violated. For all other strata, constraint violations are not checked.

`details` (`logical(1)`)
should detailed information on stratum assignments (either take-Neyman or take-bound), values of the set function s, and the number of iterations be included in the output?

Functions

- `rna_rec()`: Recursive implementation of the RNA.
- `rna_prior()`: A variant of the Recursive Neyman Algorithm (RNA) that uses prior information about strata for which allocation constraints may be violated. For all other strata, allocations are assumed to satisfy the bounds.

Note

This code has not been extensively tested and may change in future releases.

Examples

```
A <- c(3000, 4000, 5000, 2000)
M <- c(100, 90, 70, 80) # upper bounds

# experimental function (not exported) - examples skipped
## Not run:
rna_rec(total_cost = 190, A = A, bounds = M)
rna_rec(total_cost = 312, A = A, bounds = M)
rna_rec(total_cost = 339, A = A, bounds = M)
rna_rec(total_cost = 340, A = A, bounds = M)

## End(Not run)
```

---

rounding *Rounding of Numbers*

Description

[Experimental]

Usage

```
round_ran(x)

round_oric(x)
```

Arguments

x	(numeric) numeric vector.

Value

An integer vector.

Functions

- `round_ran()`: Random rounding of numbers. A number x is rounded to an integer y according to the following rule:

$$y = \lfloor x \rfloor + I(u < x - \lfloor x \rfloor)\,,$$

where the indicator function $I : \{FALSE, TRUE\} \to \{0, 1\}$ is defined as

$$I(b) := \begin{cases} 0, & b \text{ is } FALSE, \\ 1, & b \text{ is } TRUE, \end{cases}$$

and u is a random number drawn from the $\mathrm{Uniform}(0, 1)$ distribution.

- `round_oric()`: Optimal rounding under integer constraints, as proposed by Cont and Heidari (2014).

References

Cont R, Heidari M (2014). "Optimal rounding under integer constraints." 1501.00014, https://arxiv.org/abs/1501.00014.

Examples

```
x <- c(4.5, 4.1, 4.9)

set.seed(5)
round_ran(x) # 5 4 4

set.seed(6)
round_ran(x) # 4 4 5

round_oric(x) # 4 4 5
```

`simplegreedy-capacityscaling`

SimpleGreedy and CapacityScaling Algorithms

Description

[Experimental]

Fast integer-valued algorithms for optimum allocations under constraints in stratified sampling proposed in Friedrich et al. (2015).

Usage

```
SimpleGreedy(
  n,
  Ah,
  mh = rep(1, length(Ah)),
  Mh = rep(Inf, length(Ah)),
  nh = mh
)

SimpleGreedy2(v0, Nh, Sh, mh = rep(1, length(Nh)), Mh = Nh, nh = mh)

CapacityScaling(n, Ah, mh = rep(1, length(Ah)), Mh = rep(Inf, length(Ah)))

CapacityScaling2(
  v0,
  Nh,
  Sh,
  mh = rep(1, length(Nh)),
  Mh = rep(Inf, length(Nh))
)
```

Arguments

`n`	(`integerish(1)`) total sample size.
`Ah`	(`numeric`) products of population stratum sizes and standard deviations of the study variable, $A_h = N_h S_h$.
`mh`	(`integerish`) lower bounds on stratum sample sizes.
`Mh`	(`integerish`) upper bounds on stratum sample sizes.
`nh`	(`integerish`) initial allocation. Defaults to `mh`.
`v0`	(`numeric(1)`) upper bound on the variance of the estimator.
`Nh`	(`numeric`) population sizes in strata.
`Sh`	(`numeric`) standard deviations of the study variable in strata.

Value

For the `fpia()` - an integer vector of optimum sample sizes allocated to each stratum.

Functions

- `SimpleGreedy()`:
- `SimpleGreedy2()`: Variant of the *SimpleGreedy* algorithm based on a variance stopping rule.
- `CapacityScaling()`:
- `CapacityScaling2()`: Variant of the *CapacityScaling* algorithm based on a variance stopping rule.

References

Friedrich U, Münnich R, de Vries S, Wagner M (2015). "Fast integer-valued algorithms for optimal allocations under constraints in stratified sampling." *Computational Statistics & Data Analysis*, **92**, 1-12. ISSN 0167-9473, doi:10.1016/j.csda.2015.06.003.

stratallo *Optimum Sample Allocation in Stratified Sampling*

Description

Optimum Sample Allocation in Stratified Sampling

Author(s)

Wojciech Wójciak <wojciech.wojciak@gmail.com>

References

Wesołowski J, Wieczorkowski R, Wójciak W (2026). "R package stratallo - source code." https://github.com/wwojciech/stratallo.

Wesołowski J, Wieczorkowski R, Wójciak W (2023). "Numerical Performance of the RNABOX Algorithm." https://github.com/rwieczor/recursive_Neyman_rnabox.

Wesołowski J, Wieczorkowski R, Wójciak W (2022). "Optimality of the Recursive Neyman Allocation." *Journal of Survey Statistics and Methodology*, **10**(5), 1263-1275. ISSN 2325-0984, doi:10.1093/jssam/smab018.

Wójciak W (2023). "Another Solution for Some Optimum Allocation Problem." *Statistics in Transition new series*, **24**(5), 203-219. doi:10.59170/stattrans2023071.

Wójciak W (2019). *Optimal Allocation in Stratified Sampling Schemes*. Master's thesis, Warsaw University of Technology. http://home.elka.pw.edu.pl/~wwojciak/msc_wwojciech_optimum_alloc.pdf.

Wójciak W (2026). *Multi-Domain Optimum Sample Allocation with Controlled-Precision under Upper-Bound Constraints*. Ph.D. thesis, Warsaw University of Technology. http://home.elka.pw.edu.pl/~wwojciak/phd_wwojciech_optimum_alloc.pdf.

Stenger H, Gabler S (2005). "Combining random sampling and census strategies - Justification of inclusion probabilities equal to 1." *Metrika*, **61**(2), 137–156. doi:10.1007/s001840400328.

Särndal C, Swensson B, Wretman J (1992). *Model Assisted Survey Sampling*. Springer New York, NY. ISBN 978-0-387-40620-6.

See Also

Useful links:

- https://github.com/wwojciech/stratallo
- Report bugs at https://github.com/wwojciech/stratallo/issues

var_st *Variance of the Stratified π Estimator of the Population Total*

Description

[Stable]

Computes the value of the variance function of the stratified π estimator of the population total, which has the following generic form:

$$V_{st} = \sum_{h=1}^{H} \frac{A_h^2}{x_h} - A_0,$$

where H denotes the total number of strata, $x_1, \ldots, x_H$ are the stratum sample sizes, and A_0 and $A_h > 0$, for $h = 1, \ldots, H$, are population constants that do not depend on the x_h.

Usage

```
var_st(x, A, A0)

var_stsi(x, N, S)
```

Arguments

`x`	(`numeric`) sample allocations $x_1, \ldots, x_H$.
`A`	(`numeric`) population constants $A_1, \ldots, A_H$.
`A0`	(`numeric(1)`) population constant A_0.
`N`	(`integerish`) strata sizes $N_1, \ldots, N_H$.
`S`	(`numeric`) strata standard deviations of a given study variable $S_1, \ldots, S_H$.

Value

The value of the variance V_{st} for a given allocation vector $x_1, \ldots, x_H$.

Functions

- `var_st()`: The value of the variance V_{st}.
- `var_stsi()`: The value of the variance V_{st} for the case of *simple random sampling without replacement* design within each stratum.
  This particular case yields:

$$A_h = N_h S_h, \qquad h = 1, \ldots, H,$$

$$A_0 = \sum_{h=1}^{H} N_h S_h^2,$$

where N_h denotes the size of stratum h and S_h is the corresponding stratum standard deviation of the study variable, for $h = 1, \ldots, H$.

References

Särndal C, Swensson B, Wretman J (1992). *Model Assisted Survey Sampling*. Springer New York, NY. ISBN 978-0-387-40620-6.

Examples

```
N <- c(300, 400, 500, 200)
S <- c(2, 5, 3, 1)
x <- c(27, 88, 66, 9)
A <- N * S
A0 <- sum(N * S^2)

var_st(x, A, A0)

N <- c(3000, 4000, 5000, 2000)
S <- rep(1, 4)
M <- c(100, 90, 70, 80)
x <- opt(n = 320, A = N * S, M = M)

var_stsi(x = x, N, S)
```

Index