

Package: RASbound (via r-universe)

October 29, 2026

Title Bounded RAS / Biproportional Matrix Balancing with Cell-Level Constraints

Version 0.0.1

Description Implements the RAS method for adjusting a matrix to match target row and column totals. In RAS, a matrix A is multiplied on the left by a diagonal matrix R and on the right by a diagonal matrix S , which gives the method its name. It is also known as biproportional matrix balancing, iterative proportional fitting (IPF) or raking. The package extends it with lower and upper bounds on individual cells. Provides several algorithmic variants for handling bound violations: hard cell-freezing with simultaneous, sequential, or hybrid fixation; two dependency-free clamp-and-compensate heuristics (proportional redistribution and iterative renormalization); and a naive unconstrained-recompute baseline; together with thin wrapper functions exposing equivalent bounded-calibration methods from the 'survey' package under a common interface. Intended for updating or regionalizing input-output tables, social accounting matrices (SAMs), and other applications where classic RAS/IPF is used but cell-level plausibility ranges must be respected. References: Deming and Stephan (1940) <doi:10.1214/aoms/1177731829> Deville, Saerndal and Sautory (1993) <doi:10.1080/01621459.1993.10476369> Acknowledgements: The author wishes to thank Conselleria de Educacion, Cultura y Universidades, Generalitat Valenciana (grant CIAICO/2023/031) for supporting this research.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 3.6)

Imports stats, graphics, grDevices, utils

Suggests survey, MASS, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3
VignetteBuilder knitr
URL <https://github.com/JMPavia/RASbound>
BugReports <https://github.com/JMPavia/RASbound/issues>
Repository <https://jmpavia.r-universe.dev>
Date/Publication 2026-10-18 17:13:10 UTC
RemoteUrl <https://github.com/jmpavia/rasbound>
RemoteRef HEAD
RemoteSha b04135744753ba0e1a504075d3971f828509bbd1

Contents

bras_calibrate_linear	2
bras_calibrate_logit	4
bras_calibrate_raking	6
bras_cap_iter	7
bras_fit	9
bras_freeze_hybrid	10
bras_freeze_seq	11
bras_freeze_sim	12
bras_naive	13
bras_soft	15
plot.bras_fit	16
ras	17
summary.bras_fit	17
summary.bras_fit	18
Index	19

bras_calibrate_linear	<i>Bounded RAS via <code>survey::calibrate(calfun = "linear")</code> (wrapper)</i>
-----------------------	--

Description

This function is a thin wrapper around `survey::calibrate()`, for the same reason as `bras_calibrate_logit()`. Like the logit version, it enforces lower and upper as real constraints inside the same Newton-Raphson solve (Deville, Særndal and Sautory, 1993). It is not a correction made afterwards. It differs from `bras_calibrate_logit()` in the distance function (linear, that is, least squares) and in one practical point: **its bounds are optional and Inf is accepted**, so no replacement is needed. See the section "This is not the same solution as classic RAS/IPF" in `bras_calibrate_logit()`. Even without bounds, linear calibration generally returns a different matrix from classic RAS, with the same margins. See `citation("survey")`.

Usage

```
bras_calibrate_linear(
  x0,
  row_targets,
  col_targets,
  lower = 0,
  upper = Inf,
  tol = 1e-06,
  ...
)
```

Arguments

<code>x0</code>	Numeric matrix, the base/seed matrix (non-negative).
<code>row_targets, col_targets</code>	Numeric vectors of target row/column totals.
<code>lower, upper</code>	Scalar or matrix of per-cell lower/upper bounds. Defaults to 0 and Inf.
<code>tol</code>	Tolerance used to decide converged. It is checked against the margin error of the returned matrix. It is looser than the default of the freeze functions, because the solution comes from survey and is reshaped back to a matrix.
<code>...</code>	Additional arguments passed on to survey::calibrate() .

Details

Rows and columns with a target of exactly 0 are removed before calling survey and put back as zeros afterwards.

Value

An object of class `bras_fit`; see `?bras_fit`. `converged` is computed from the margin error of the returned matrix. `feasible` is always NA, because the bounds are optional here and a successful return does not prove that they could be met. Check `converged` instead.

Examples

```
if (requireNamespace("survey", quietly = TRUE) &&
    requireNamespace("MASS", quietly = TRUE)) {
  x0 <- matrix(c(2, 4, 6, 8), nrow = 2)
  fit <- bras_calibrate_linear(x0, c(20, 20), c(15, 25),
                             upper = matrix(c(Inf, Inf, 12, Inf), nrow = 2))
  fit$matrix
  fit$converged
}
```

```
bras_calibrate_logit  Bounded RAS via survey::calibrate(calfun = "logit") (wrapper)
```

Description

This function is a thin wrapper around `survey::calibrate()`. It gives the same arguments and the same `bras_fit` result as the rest of `RASbound`, but it is not an independent implementation. The algorithm, its guarantees and its limits are those of the `survey` package. For example, `survey` documents that the Newton-Raphson iterations of `calfun = "logit"` can fail to converge when the bounds are very tight. If that happens, the error comes from `survey`, not from `RASbound`. See `citation("survey")`.

Usage

```
bras_calibrate_logit(
  x0,
  row_targets,
  col_targets,
  lower = 0,
  upper = Inf,
  tol = 1e-06,
  unbounded_mult = 20,
  ...
)
```

Arguments

<code>x0</code>	Numeric matrix, the base/seed matrix (non-negative).
<code>row_targets, col_targets</code>	Numeric vectors of target row/column totals.
<code>lower, upper</code>	Scalar or matrix of per-cell lower/upper bounds. Defaults to 0 and Inf.
<code>tol</code>	Tolerance used to decide converged. It is checked against the margin error of the returned matrix. It is looser than the default of the freeze functions, because the solution comes from <code>survey</code> and is reshaped back to a matrix.
<code>unbounded_mult</code>	Multiplier used to replace an infinite upper (see "Logit calibration cannot represent an unbounded cell"). The replacement is <code>unbounded_mult * max(row_targets, col_targets)</code> . Increase it if <code>survey::calibrate()</code> still fails to converge. Decrease it only for a specific reason, since a tighter replacement is more fragile.
<code>...</code>	Additional arguments passed on to <code>survey::calibrate()</code> .

Details

Internally, each cell of `x0` becomes one row of a long table, with the cell value as its calibration weight and a formula with row and column effects only. `survey` enforces the bounds as real constraints inside its Newton-Raphson solve, not as a correction afterwards. The "logit" and

"linear" versions differ only in the distance function they minimize. This is a different approach from `bras_freeze_sim()` and the other freeze functions. It optimizes the whole matrix in one step instead of solving, finding violations, fixing them and repeating.

Rows and columns with a target of exactly 0 are removed before calling `survey` and put back as zeros afterwards. An error is raised if such a row or column has a positive lower bound.

Value

An object of class `bras_fit`; see `?bras_fit`. `converged` is computed from the margin error of the returned matrix. `feasible` is `TRUE` with the default `force = FALSE`, because `survey::calibrate()` stops with an error instead of returning an unconverged result. If you pass `force = TRUE` through ..., this `TRUE` is optimistic, so check `converged` as well.

This is not the same solution as classic RAS/IPF

Only `bras_calibrate_raking()` minimizes the same multiplicative distance as classic RAS. Without bounds, it gives the same matrix as `ras()`, up to numerical tolerance. `calfun = "logit"` (this function) and `calfun = "linear"` (`bras_calibrate_linear()`) minimize other distance functions (a bounded logistic one and a least-squares one). Without bounds, both still meet the same margins as classic RAS, but they generally return a different matrix. See `vignette("RASbound-diagnostics")`.

Known fragility on real large tables

On a real 57 x 57 input-output-style table, all three `bras_calibrate_*`() wrappers failed inside `survey::calibrate()`, while the `bras_freeze_*`() functions converged. Removing the zero-target rows and columns did not fix this. The cause is not known. Differences in scale between cells are one possibility that has not been tested. If a wrapper fails on your data, use `bras_freeze_sim()`, `bras_freeze_seq()` or `bras_freeze_hybrid()`.

Logit calibration cannot represent an unbounded cell

The logit calibration of `survey` maps each weight onto the interval `(lower, upper)` with a logistic transform. That transform is not defined for an infinite bound, and `survey` stops with the error "Logit calibration requires finite bounds". The other functions of this package use `upper = Inf` for "no upper bound". So this wrapper replaces every infinite upper bound by `unbounded_mult * max(row_targets, col_targets)` (20 times by default). No cell can reach such a value, and it is far enough from the saturation zone of the logistic transform. It also scales with the marginal totals, not with the matrix size.

A tighter replacement is not used. The exact maximum of a cell, `min(row_targets[i], col_targets[j])`, is correct but numerically fragile: a bound that close to the true supremum is where the logistic transform saturates and the Newton-Raphson iterations stall (`survey` then reports "Failed to converge"). Increase `unbounded_mult` if that still happens on your data. An infinite lower bound (unusual, since the default is 0) is replaced by a large negative number.

Failure mode

a cell at a real, tight bound: Suppose a cell has a real, tight upper and its seed in `x0` is at or above it. The seed is moved to exactly that bound before the calibration starts (see `vignette("RASbound-diagnostics")`). If most other cells have no bound and get the replacement above, one cell sits on a tight bound next

to cells whose bound is tens of times larger. This combination can make the Newton-Raphson solve diverge, with the error "infinite or missing values" from an internal routine, instead of converging slowly. `vignette("RASbound-diagnostics")` has a reproducible example.

If you know that some cells need a tight bound, give real finite bounds to the other cells too, even generous ones, instead of relying on the replacement of `Inf`. Or use `bras_calibrate_raking()` or `bras_calibrate_linear()`. Both accept `Inf` and need no replacement, so they are more robust when most cells are effectively unbounded.

Examples

```
if (requireNamespace("survey", quietly = TRUE) &&
    requireNamespace("MASS", quietly = TRUE)) {
  x0 <- matrix(c(2, 4, 6, 8), nrow = 2)
  # upper = Inf on most cells is fine here: internally substituted with
  # unbounded_mult * max(row_target, col_target), since logit
  # calibration requires finite bounds (see "Logit calibration cannot
  # represent an unbounded cell").
  fit <- bras_calibrate_logit(x0, c(20, 20), c(15, 25),
                           upper = matrix(c(Inf, Inf, 12, Inf), nrow = 2))
  fit$matrix
}
```

`bras_calibrate_raking` *Bounded RAS via survey::calibrate(calfun = "raking") (wrapper)*

Description

This function is a thin wrapper around `survey::calibrate()`, for the same reason as `bras_calibrate_logit()` and `bras_calibrate_linear()`. Unlike those two, `calfun = "raking"` minimizes the same multiplicative distance as classic RAS/IPF. Without bounds, it is **expected to give essentially the same matrix as `ras()`**, up to numerical tolerance. It gets there with a different algorithm: the Newton-Raphson solve of `survey` (Deville, Saerndal and Sautory, 1993) instead of this package's alternating row and column scaling.

Usage

```
bras_calibrate_raking(
  x0,
  row_targets,
  col_targets,
  lower = 0,
  upper = Inf,
  tol = 1e-06,
  ...
)
```

Arguments

<code>x0</code>	Numeric matrix, the base/seed matrix (non-negative).
<code>row_targets, col_targets</code>	Numeric vectors of target row/column totals.
<code>lower, upper</code>	Scalar or matrix of per-cell lower/upper bounds. Defaults to 0 and Inf.
<code>tol</code>	Tolerance used to decide converged. It is checked against the margin error of the returned matrix. It is looser than the default of the freeze functions, because the solution comes from survey and is reshaped back to a matrix.
<code>...</code>	Additional arguments passed on to <code>survey::calibrate()</code> .

Details

The package's cross-validation test checks this agreement (see `vignette("RASbound-diagnostics")`). That is why this wrapper exists. It is the one external result that should match this package's own engine on a problem without bounds, so it is the most direct check against an independent implementation.

With bounds (what survey calls bounded raking), the bounds are enforced as real constraints inside the same solve. As with `bras_calibrate_linear()`, they are optional and Inf is accepted, so no replacement is needed, unlike `bras_calibrate_logit()`. See `citation("survey")`.

Rows and columns with a target of exactly 0 are removed before calling survey and put back as zeros afterwards.

Value

An object of class `bras_fit`; see `?bras_fit`. `converged` is computed from the margin error of the returned matrix. `feasible` is always NA, because the bounds are optional here and a successful return does not prove that they could be met. Check `converged` instead.

Examples

```
if (requireNamespace("survey", quietly = TRUE) &&
    requireNamespace("MASS", quietly = TRUE)) {
  x0 <- matrix(c(2, 4, 6, 8), nrow = 2)
  fit <- bras_calibrate_raking(x0, c(20, 20), c(15, 25))
  fit$matrix
  # expected to match our own engine on the same problem:
  ras(x0, c(20, 20), c(15, 25))$matrix
}
```

Description

After each row and column scaling step, cells outside `[lower, upper]` are set to the nearest bound ("capped"). Then only the cells of that row or column that are not capped are rescaled, all by the same factor, to make up for the mass lost or gained. This differs from `bras_naive()`, which leaves the margin off, and from `bras_soft()`, which shares the difference in proportion to the remaining headroom. No cell is removed for good: a cell capped in one pass can move again in the next.

Usage

```
bras_cap_iter(
  x0,
  row_targets,
  col_targets,
  lower = 0,
  upper = Inf,
  tol = 1e-08,
  max_iter = 200,
  verbose = FALSE
)
```

Arguments

<code>x0</code>	Numeric matrix, the base/seed matrix (non-negative).
<code>row_targets, col_targets</code>	Numeric vectors of target row/column totals.
<code>lower, upper</code>	Scalar or matrix of per-cell lower/upper bounds. Defaults to 0 and Inf.
<code>tol</code>	Convergence tolerance on margins for the inner RAS solve.
<code>max_iter</code>	Maximum inner RAS iterations per outer pass.
<code>verbose</code>	Print progress messages.

Details

This is RASbound's own implementation, without extra dependencies, of the general "cap and iterate" idea used by some external weight-raking tools. It is not a wrapper of, nor a copy of, any specific package. See `vignette("RASbound-intro")` for how it compares to `bras_naive()` and `bras_soft()`.

Value

An object of class `bras_fit`; see `?bras_fit`. `feasible` is always NA here, since this variant does not attempt to diagnose infeasibility.

Examples

```
x0 <- matrix(c(2, 4, 6, 8), nrow = 2)
fit <- bras_cap_iter(x0, c(20, 20), c(15, 25),
  upper = matrix(c(Inf, Inf, 12, Inf), nrow = 2))
fit$matrix
```

bras_fit

The bras_fit result object

Description

Every function in RASbound returns an object of class `bras_fit`. This includes `ras()`, the `bras_freeze_*`() functions, the reference variants and the `bras_calibrate_*`() wrappers. It is a plain list with the fields below, so code that uses one method's result also works with the result of any other method.

Details

`feasible = TRUE` together with `converged = FALSE` specifically means the freeze algorithm ran out of `max_outer` passes before it found a solution or proved that none exists. Increasing `max_outer` may solve it. `feasible = FALSE` means that the bounds were shown to admit no solution (see `diagnostics$infeasible_at`). More iterations cannot fix that. You have to change the bounds or the targets.

A third, distinct situation (hard-freeze family only): `diagnostics$stalled = TRUE` means no bound violations remained, but the inner alternating row/column solve reached a genuine fixed point that does not satisfy the margins, and stayed there. The margin error stopped changing across iterations; it was not just slow to converge. This happens when the free cells left after fixation form a structurally degenerate subsystem: row-wise pressure to scale up is exactly cancelled by column-wise pressure to scale down (or vice versa). This is a known limitation of alternating-projection RAS/IPF methods in general, not specific to this package. The feasibility check (`sum(upper) >= residual target` for each row and column) is necessary but not sufficient, and a stall is a case it cannot detect. A stall often means that no matrix satisfies the targets and the bounds together. Unlike `max_outer` exhaustion, increasing `max_iter` or `max_outer` will not help; see `vignette("RASbound-diagnostics")`.

Fields

`matrix` The adjusted matrix. For the hard-freeze family this exactly matches `row_targets/col_targets` and respects `lower/upper` whenever `converged` is `TRUE`; for the reference variants and the wrappers, check `diagnostics` for how close it actually got.

`converged` `TRUE` if the method finished having satisfied its own stopping criterion (exact margins for the hard-freeze family and `ras()`; margin tolerance `tol` for the reference variants).

`feasible` For the hard-freeze family: `TRUE` unless a bound configuration was proven to admit no solution (see `diagnostics$infeasible_at`); can also be `TRUE` while `converged` is `FALSE` if `max_outer` ran out before a solution was found or ruled out. These are different situations; see Details. For methods that do not attempt to diagnose infeasibility (the reference variants, the wrappers), this is `NA`.

`n_outer_iter` Number of outer fixation passes (hard-freeze family) or scaling iterations (reference variants) used.

`fixed_cells` A data frame with one row per cell permanently fixed at a bound: `row`, `col`, `bound` ("`lower`"/"`upper`"), `value`, and `outer_pass` (which pass fixed it). Empty for methods that do not freeze cells.

diagnostics A list with `row_margin_error` and `col_margin_error` (absolute deviation from `row_targets/col_targets` in the returned matrix), `infeasible_at` (NULL unless infeasibility was proven, in which case a list with rows/cols naming the offending margins), and `stalled` (hard-freeze family only; see Details).

method A short string identifying which function produced this fit (e.g. "freeze_simultaneous", "ras_classic").

call The original function call, as from `match.call()`.

<code>bras_freeze_hybrid</code>	<i>Bounded RAS: hybrid hard-freeze variant (simultaneous with sequential fallback)</i>
---------------------------------	--

Description

Tries simultaneous fixation each pass; if that makes the residual problem locally infeasible *within that one pass*, that pass is retried fixing only the most extreme violating cell before continuing. That per-pass safety net is not always enough: fixing slightly more than necessary in several *different* passes can add up to an infeasible or non-convergent run, even though no single pass looked infeasible (see vignette("RASbound-diagnostics")). So if the per-pass approach ends without a solution, the **entire problem is retried once, from scratch, with `bras_freeze_seq()`'s one-cell-at-a-time fixation** before giving up. This costs nothing in the common case, because it only happens after the faster approach has failed. The result is as robust as `bras_freeze_seq()` and as fast as `bras_freeze_sim()` whenever that is safe. This is the recommended default when you are unsure whether to pick `bras_freeze_sim()` or `bras_freeze_seq()`.

Usage

```
bras_freeze_hybrid(
  x0,
  row_targets,
  col_targets,
  lower = 0,
  upper = Inf,
  tol = 1e-08,
  max_iter = 100,
  max_outer = 50,
  on_infeasible = c("error", "warning", "force"),
  verbose = FALSE
)
```

Arguments

<code>x0</code>	Numeric matrix, the base/seed matrix (non-negative).
<code>row_targets, col_targets</code>	Numeric vectors of target row/column totals.
<code>lower, upper</code>	Scalar or matrix of per-cell lower/upper bounds. Defaults to 0 and Inf.

tol	Convergence tolerance on margins for the inner RAS solve.
max_iter	Maximum inner RAS iterations per outer pass.
max_outer	Maximum number of outer fixation passes.
on_infeasible	One of "error", "warning", "force": what to do if no bound-respecting solution exists.
verbose	Print progress messages.

Value

An object of class `bras_fit`; see `?bras_fit` for the fields. method is "freeze_hybrid" normally, or "freeze_hybrid_retry_sequential" if the global retry was the one that found the solution.

Examples

```
x0 <- matrix(c(2, 4, 6, 8), nrow = 2)
fit <- bras_freeze_hybrid(x0, c(20, 20), c(15, 25),
                        upper = matrix(c(Inf, Inf, 12, Inf), nrow = 2))
fit$matrix
```

bras_freeze_seq	<i>Bounded RAS: sequential hard-freeze variant (most-extreme cell first)</i>
-----------------	--

Description

Same two-stage idea as `bras_freeze_sim()` (solve the free subproblem, then fix violators, then repeat), but fixes only the single most extreme violating cell per outer pass instead of all of them at once. Slower, but never over-fixes: prefer this over `bras_freeze_sim()` when you suspect simultaneous fixation is reporting a false infeasibility, or when you want the most conservative, auditable fixation trail.

Usage

```
bras_freeze_seq(
  x0,
  row_targets,
  col_targets,
  lower = 0,
  upper = Inf,
  tol = 1e-08,
  max_iter = 100,
  max_outer = 50,
  on_infeasible = c("error", "warning", "force"),
  verbose = FALSE
)
```

Arguments

<code>x0</code>	Numeric matrix, the base/seed matrix (non-negative).
<code>row_targets, col_targets</code>	Numeric vectors of target row/column totals.
<code>lower, upper</code>	Scalar or matrix of per-cell lower/upper bounds. Defaults to 0 and Inf.
<code>tol</code>	Convergence tolerance on margins for the inner RAS solve.
<code>max_iter</code>	Maximum inner RAS iterations per outer pass.
<code>max_outer</code>	Maximum number of outer fixation passes.
<code>on_infeasible</code>	One of "error", "warning", "force": what to do if no bound-respecting solution exists.
<code>verbose</code>	Print progress messages.

Value

An object of class `bras_fit`; see `?bras_fit` for the fields.

Examples

```
x0 <- matrix(c(2, 4, 6, 8), nrow = 2)
fit <- bras_freeze_seq(x0, c(20, 20), c(15, 25),
                      upper = matrix(c(Inf, Inf, 12, Inf), nrow = 2))
fit$fixed_cells
```

<code>bras_freeze_sim</code>	<i>Bounded RAS: simultaneous hard-freeze variant</i>
------------------------------	--

Description

Solves the free subproblem to convergence, fixes **all** free cells that violate a bound in that pass, subtracts them from the residual margins, and repeats. See `vignette("RASbound-intro")` for details and trade-offs relative to `bras_freeze_seq()` and `bras_freeze_hybrid()`.

Usage

```
bras_freeze_sim(
  x0,
  row_targets,
  col_targets,
  lower = 0,
  upper = Inf,
  tol = 1e-08,
  max_iter = 100,
  max_outer = 50,
  on_infeasible = c("error", "warning", "force"),
  verbose = FALSE
)
```

Arguments

<code>x0</code>	Numeric matrix, the base/seed matrix (non-negative).
<code>row_targets, col_targets</code>	Numeric vectors of target row/column totals.
<code>lower, upper</code>	Scalar or matrix of per-cell lower/upper bounds. Defaults to 0 and Inf.
<code>tol</code>	Convergence tolerance on margins for the inner RAS solve.
<code>max_iter</code>	Maximum inner RAS iterations per outer pass.
<code>max_outer</code>	Maximum number of outer fixation passes.
<code>on_infeasible</code>	One of "error", "warning", "force": what to do if no bound-respecting solution exists.
<code>verbose</code>	Print progress messages.

Value

An object of class `bras_fit`; see `?bras_fit` for the fields.

Examples

```
x0 <- matrix(c(2, 4, 6, 8), nrow = 2)
row_targets <- c(20, 20)
col_targets <- c(15, 25)
upper <- matrix(c(Inf, Inf, 12, Inf), nrow = 2) # cap cell [1, 2] at 12

fit <- bras_freeze_sim(x0, row_targets, col_targets, upper = upper)
fit$matrix
fit$fixed_cells
summary(fit)

# An infeasible configuration is reported, not silently misapplied:
too_tight <- matrix(1, nrow = 2, ncol = 2)
tryCatch(
  bras_freeze_sim(matrix(1, 2, 2), c(100, 2), c(51, 51), upper = too_tight),
  error = function(e) conditionMessage(e)
)
```

<code>bras_naive</code>	<i>Bounded RAS: naive recompute-every-pass variant (baseline, not recommended)</i>
-------------------------	--

Description

Recomputes the full RAS on **every** cell each pass, including cells that already sit on a bound. After every row and column scaling step it sets each cell outside `[lower, upper]` to the nearest bound, and never removes a cell from further adjustment. The next row or column correction can push such a cell outside its bound again. So this method can oscillate and fail to reach the target margins, even when a solution that respects the bounds exists.

Usage

```
bras_naive(
  x0,
  row_targets,
  col_targets,
  lower = 0,
  upper = Inf,
  tol = 1e-08,
  max_iter = 200,
  verbose = FALSE
)
```

Arguments

<code>x0</code>	Numeric matrix, the base/seed matrix (non-negative).
<code>row_targets, col_targets</code>	Numeric vectors of target row/column totals.
<code>lower, upper</code>	Scalar or matrix of per-cell lower/upper bounds. Defaults to 0 and Inf.
<code>tol</code>	Convergence tolerance on margins for the inner RAS solve.
<code>max_iter</code>	Maximum inner RAS iterations per outer pass.
<code>verbose</code>	Print progress messages.

Details

It is included as a reference baseline for comparing against [bras_freeze_sim\(\)](#), [bras_freeze_seq\(\)](#) and [bras_freeze_hybrid\(\)](#). For actual data adjustment, prefer one of the `bras_freeze_*`() functions, which guarantee exact margins and respected bounds whenever a solution exists.

Value

An object of class `bras_fit`; see `?bras_fit`. `feasible` is always NA here, since this variant does not attempt to diagnose infeasibility.

Examples

```
x0 <- matrix(c(2, 4, 6, 8), nrow = 2)
fit <- bras_naive(x0, c(20, 20), c(15, 25),
  upper = matrix(c(Inf, Inf, 12, Inf), nrow = 2))
fit$converged # may be FALSE even though a solution exists
```

bras_soft

*Bounded RAS: soft proportional redistribution variant***Description**

Sets free cells outside their bounds to the nearest bound, and shares the excess or deficit among the other free cells of the same row and column, in proportion to their remaining headroom. It is similar in spirit to the bounded trimming of `survey::calibrate(calfun = "linear")`. No cell is fixed for good. It is a comparator for `bras_freeze_sim()` that needs no extra packages; see `vignette("RASbound-intro")` for the trade-offs.

Usage

```
bras_soft(
  x0,
  row_targets,
  col_targets,
  lower = 0,
  upper = Inf,
  tol = 1e-08,
  max_iter = 200,
  verbose = FALSE
)
```

Arguments

<code>x0</code>	Numeric matrix, the base/seed matrix (non-negative).
<code>row_targets, col_targets</code>	Numeric vectors of target row/column totals.
<code>lower, upper</code>	Scalar or matrix of per-cell lower/upper bounds. Defaults to 0 and Inf.
<code>tol</code>	Convergence tolerance on margins for the inner RAS solve.
<code>max_iter</code>	Maximum inner RAS iterations per outer pass.
<code>verbose</code>	Print progress messages.

Value

An object of class `bras_fit`; see `?bras_fit`. `feasible` is always NA here, since this variant does not attempt to diagnose infeasibility.

Examples

```
x0 <- matrix(c(2, 4, 6, 8), nrow = 2)
fit <- bras_soft(x0, c(20, 20), c(15, 25),
                upper = matrix(c(Inf, Inf, 12, Inf), nrow = 2))
fit$matrix
```

plot.bras_fit	<i>Plot which cells were fixed at a bound</i>
---------------	---

Description

Draws the fitted matrix as an image and marks the cells that were fixed at a bound.

Usage

```
## S3 method for class 'bras_fit'
plot(x, labels = TRUE, cex.axis = 0.6, ...)
```

Arguments

x	A bras_fit object.
labels	Logical. If TRUE (the default), show the row and column names of the matrix as axis labels when it has them. Use FALSE to hide them.
cex.axis	Size of the axis labels. Reduce it for large matrices.
...	Not used.

Value

x, invisibly.

How to read the plot

- Each square is one cell of the matrix. Row 1 is at the top and column 1 is on the left. If the fitted matrix has row or column names (rownames() and colnames()), they are shown as axis labels. Without names there are no labels.
- The colour shows the value of the cell in the fitted matrix. Lower values are pale yellow and higher values are dark red. A few very large cells can make all the others look similar, since the colours follow the raw values.
- An upward triangle marks a cell fixed at its **upper** bound. A downward triangle marks a cell fixed at its **lower** bound. Cells without a marker were never fixed.
- Methods that do not fix cells (for example [ras\(\)](#) or [bras_naive\(\)](#)) show no markers.

The plot gives a quick view of where the bounds are binding. The exact cells, bounds and values are in fit\$fixed_cells.

Examples

```
x0 <- matrix(c(2, 4, 6, 8), nrow = 2)
fit <- bras_freeze_sim(x0, c(20, 20), c(15, 25),
                      upper = matrix(c(Inf, Inf, 12, Inf), nrow = 2))
plot(fit)
```

ras	<i>Classic RAS (no bounds)</i>
-----	--------------------------------

Description

Plain biproportional RAS with no cell-level bounds; provided as the baseline/control condition for comparison against the bounded variants.

Usage

```
ras(x0, row_targets, col_targets, tol = 1e-08, max_iter = 100)
```

Arguments

x0	Numeric matrix, the base/seed matrix (non-negative).
row_targets, col_targets	Numeric vectors of target row/column totals.
tol	Convergence tolerance on margins for the inner RAS solve.
max_iter	Maximum inner RAS iterations per outer pass.

Value

An object of class `bras_fit` (with an empty `fixed_cells` table, since there are no bounds to violate); see `?bras_fit`.

Examples

```
x0 <- matrix(c(2, 4, 6, 8), nrow = 2)
fit <- ras(x0, row_targets = c(20, 20), col_targets = c(15, 25))
fit$matrix
rowSums(fit$matrix) # matches row_targets exactly
```

<code>summary.bras_fit</code>	<i>Summarize a fitted bounded-RAS object</i>
-------------------------------	--

Usage

```
## S3 method for class 'bras_fit'
summary(object, ...)
```

Arguments

object A bras_fit object.
... Unused, present for S3 consistency.
A data.frame with one row and the columns method (the method used), converged (logical), feasible (logical, NA for methods that do not check feasibility), n_outer_iter (number of outer passes), n_fixed_cells (number of cells fixed at a bound) and max_row_margin_error and max_col_margin_error (the largest absolute error in the row and column totals of the fitted matrix).
Summarize a fitted bounded-RAS object

summary.bras_fit	<i>Summarize a fitted bounded-RAS object</i>
------------------	--

Description

Summarize a fitted bounded-RAS object

Usage

```
## S3 method for class 'bras_fit'  
summary(object, ...)
```

Arguments

object A bras_fit object.
... Unused, present for S3 consistency.

Value

A data.frame with one row and the columns method (the method used), converged (logical), feasible (logical, NA for methods that do not check feasibility), n_outer_iter (number of outer passes), n_fixed_cells (number of cells fixed at a bound) and max_row_margin_error and max_col_margin_error (the largest absolute error in the row and column totals of the fitted matrix).

Index

`bras_calibrate_linear`, 2
`bras_calibrate_linear()`, 5–7
`bras_calibrate_logit`, 4
`bras_calibrate_logit()`, 2, 6, 7
`bras_calibrate_raking`, 6
`bras_calibrate_raking()`, 5, 6
`bras_cap_iter`, 7
`bras_fit`, 9
`bras_fit-class (bras_fit)`, 9
`bras_freeze_hybrid`, 10
`bras_freeze_hybrid()`, 5, 12, 14
`bras_freeze_seq`, 11
`bras_freeze_seq()`, 5, 10, 12, 14
`bras_freeze_sim`, 12
`bras_freeze_sim()`, 5, 10, 11, 14, 15
`bras_naive`, 13
`bras_naive()`, 8, 16
`bras_soft`, 15
`bras_soft()`, 8

`match.call()`, 10

`plot.bras_fit`, 16

`ras`, 17
`ras()`, 5, 6, 9, 16

`summary.bras_fit`, 17, 18
`survey::calibrate()`, 2–4, 6, 7