Package: sandwichrand (via r-universe)

October 14, 2026

Type Package

Title Sandwich Mixed Randomization for Allocation Sequence Generation

Version 1.0.0

Description Generates allocation sequences for randomized controlled trials using the sandwich mixed randomization (SMR) scheme, together with verification, diagnostics, and release utilities.

License Apache License (>= 2.0)

Encoding UTF-8

URL https://github.com/wangbshun/sandwichrand

BugReports https://github.com/wangbshun/sandwichrand/issues

Depends R (>= 3.6.0)

Imports graphics, parallel, stats, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

NeedsCompilation no

Config/roxygen2/version 8.1.0

Author Bingshun Wang [aut, cre] (Designer of the sandwich mixed randomization scheme), Jinliang Xie [ctb], Wei Wang [ctb]

Maintainer Bingshun Wang <wangbshun@foxmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-10-14 17:23:44 UTC

RemoteUrl https://github.com/cran/sandwichrand

RemoteRef HEAD

RemoteSha ecce17b8d27f5b08bfd2f64ed3f9573385ca30e0

Contents

sandwichrand-package . . . 2
sandwichrand-methods . . . 4
smr_alloc . . . 5
smr_balance . . . 8
smr_blocks . . . 9
smr_correct_guesses . . . 9
smr_imbalance . . . 10
smr_pcg . . . 11
smr_reproduce . . . 11
smr_rtf . . . 12
smr_simulate . . . 13
smr_spread . . . 14
smr_streams . . . 14
smr_verify . . . 15
smr_write_csv . . . 16

Index **17**

`sandwichrand-package` *Allocation sequences by sandwich mixed randomization*

Description

sandwichrand generates treatment allocation sequences (randomization lists) for randomized controlled trials with the sandwich mixed randomization (SMR) scheme, and provides the routines needed to verify, document and release them.

Details

The SMR sequence is a sandwich. Writing `n1` for the size of the first block and `N` for the total sample size, the layout is

```
|<--- n1 --->|<-------- N - 2*n1 -------->|<--- n1 --->|
+------------+---------------------------+------------+
| block 1    | blocks 2 .. K             | block K+1  |
| imbalanced | permuted blocks of        | complement |
|            | varying sizes             | of block 1 |
+------------+---------------------------+------------+
```

Block 1 and block K+1 are the two halves of one balanced pool of `2*n1` slots, so the last block is the exact complement of the first one: balance over the whole trial is guaranteed while the next allocation stays unpredictable. Two further paths handle the boundary cases: `N < 4 * ratio_sum` gives complete randomization in one single block (path 1), and `N == 2 * n1` gives head and tail blocks only (path 3).

The head-block size `n1` is always sampled at random from the admissible multiples of `ratio_sum` (there is no need to pre-specify it and no user override is accepted), and the imbalance of the

first block is measured by the *spread* of the deviations from the target allocation, a generalisation that equals the between-group count difference for a 1:1 allocation and that is zero for an exactly proportional block.

Main functions

`smr_alloc` generate one sequence;

`smr_streams` derive independent random streams for high-frequency calling;

`smr_verify` re-check every structural invariant of a sequence;

`smr_balance`, `smr_imbalance`, `smr_blocks`, `smr_spread` balance and diagnostics;

`smr_correct_guesses`, `smr_pcg` evaluate allocation unpredictability (proportion of correct guesses);

`smr_simulate` assess a design by replication, reporting both the imbalance and the correct-guess proportion;

`smr_rtf`, `smr_write_csv` release the list, with or without the block structure;

`sandwichrand-methods` `print`, `summary` and `plot` methods.

Reproducibility and independent streams

Every sequence is a deterministic function of a single seed (or of a stream that is handed to the call). The modules of one call draw from independent L'Ecuyer-CMRG sub-streams derived from that seed, so the modules are independent and a call to `smr_alloc` never changes the random number state of the calling session. For high-frequency calling, `smr_streams` derives a family of non-overlapping sub-streams from one base seed; handing one stream to each call guarantees that no two calls ever draw from the same random numbers, so a loop stays both reproducible and free of repeated streams.

Confidentiality

ICH E9 and good clinical practice require the block structure and the seed to remain confidential. `keep.blocks = FALSE` drops the block columns from `$alloc` and from every exported artefact, while the statistician's copy of the structure stays available in `$blocks` for quality control.

Clinical use and validation disclaimer

sandwichrand is research software provided "AS IS", without warranty of any kind, express or implied. It has not been validated as a computerised system under Good Clinical Practice (GCP), Good Clinical/Manufacturing Practice (GxP), 21 CFR Part 11, EU GMP Annex 11, or any other regulatory framework.

Before using **sandwichrand** to generate allocation sequences for a registration trial, or for any trial whose results may support a regulatory submission, the sponsor is solely responsible for: (i) independently validating the software and the generated sequences under its own quality system; (ii) maintaining allocation concealment (ICH E9), including keeping the seed, the block structure and any stream information confidential; and (iii) ensuring compliance with all applicable regulations.

The authors accept no liability for any consequences arising from the use of this software.

Author(s)

Bingshun Wang <wangbshun@foxmail.com>

References

Wang B, Wang X, Ni C, Xie J, Wu S, Wu Y, Shen Y, Dong Z, Lv W, He Y (2026). "Reducing selection bias risk to enhance RCT validity: sandwich mixed randomization outperforms permuted block design." *BMC Medical Research Methodology*, **26**(1), 152. doi:10.1186/s12874026028772.

ICH (1998). *E9: Statistical Principles for Clinical Trials*.

See Also

smr_alloc

---

sandwichrand-methods *Methods for* **sandwichrand** *objects*

Description

`print` shows the design and the head of the allocation list, `summary` adds the arm balance and the verification table, `as.data.frame` extracts the allocation list itself, and `plot` draws the running imbalance.

Usage

```
## S3 method for class 'sandwichrand'
format(x, ...)

## S3 method for class 'sandwichrand'
print(x, n = 6L, ...)

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

## S3 method for class 'summary.sandwichrand'
print(x, ...)

## S3 method for class 'sandwichrand'
plot(
  x,
  y = NULL,
  type = "imbalance",
  main = "Running imbalance",
  xlab = "allocation number",
  ylab = NULL,
  col = 1,
  lwd = 2,
  ...
)

## S3 method for class 'sandwichrand'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`, `object`	An object of class `"sandwichrand"`.
`...`	Further arguments passed to or from other methods.
`n`	Integer. Number of allocation rows echoed by `print`.
`y`	Ignored, present for compatibility with `plot`.
`type`	Character. Which curve to draw: `"imbalance"` (the default) for the running imbalance, or `"cumulative"` for the cumulative deviation of each arm from its target.
`xlab`, `ylab`, `main`, `col`, `lwd`	Graphical parameters.
`row.names`	Passed to `as.data.frame`.
`optional`	Passed to `as.data.frame`.

Value

`as.data.frame` returns the allocation data frame; `format` returns a character vector; `summary` returns a `"summary.sandwichrand"` object; the print and plot methods return their input invisibly.

---

smr_alloc *Allocation sequences by sandwich mixed randomization*

Description

Draws a reproducible allocation sequence (randomization list) for a randomized controlled trial with the sandwich mixed randomization (SMR) scheme.

Usage

```
smr_alloc(
  N,
  arms,
  seed = NULL,
  stream = NULL,
  block.sizes = NULL,
  imbalance = c(1, 4),
  max.cand = 10L,
  max.iter = 1000L,
  keep.blocks = TRUE,
  verbose = FALSE
)
```

Arguments

`N`	Integer. Total sample size.
`arms`	Specification of the treatment arms and of the allocation ratio. Either a character vector with one token per allocation-ratio unit – `c("TestTrt", "Placebo")` for 1:1 and `c("A", "A", "B", "B","C")` for 2:2:1, a single blank-separated string `"A A B B C"` is also accepted – or a named vector of ratio units such as `c(TestTrt = 2, Placebo = 2, Control = 1)`. Arm identity is case-insensitive.
`seed`	Integer. Random seed. `NULL` (the default) draws a seed from the system clock; the seed actually used is stored in the result and is what has to be archived to reproduce the list.
`stream`	Numeric vector. The raw state of an independent L'Ecuyer-CMRG sub-stream (for example one of the values returned by `smr_streams`). Supplying `stream` is the most efficient way to give one call an independent random stream when the package is called at high frequency: each stream is non-overlapping with every other, so a loop never re-uses the same random numbers. At most one of `seed` and `stream` may be supplied.
`block.sizes`	Integer vector. Candidate sizes of the middle blocks. Defaults to `c(2, 3, 4) * ratio_sum`.
`imbalance`	Numeric vector of length two, `c(min, max)`. Admissible imbalance of the first block (default `c(1, 4)`; 4 is the safety factor proposed in the SMR paper). The imbalance is the *spread* of the deviations from the target allocation, which for a 1:1 allocation is the between-group count difference and is zero for an exactly proportional block.
`max.cand`	Integer. Maximum number of candidate values kept for `n1` (default 10).
`max.iter`	Integer. Maximum number of rejection-sampling attempts per module (default 1000). If no candidate qualifies, the closest one is used, so a sequence is always produced.
`keep.blocks`	Logical. `TRUE` (default) keeps the `block` and `block_size` columns in `$alloc`; `FALSE` drops them, as required by ICH E9 and good clinical practice when the list is released to the trial sites. The block structure is still kept in `$blocks`, which is the copy for the statistician and is never written out by `smr_write_csv` or `smr_rtf` when `keep.blocks = FALSE`.
`verbose`	Logical. Emit the design notes and warnings as messages (default `FALSE`; they are always stored in the returned object).

Details

The SMR sequence is a sandwich. Writing `n1` for the size of the first block and `N` for the total sample size, the layout is

```
|<--- n1 --->|<-------- N - 2*n1 -------->|<--- n1 --->|
+------------+---------------------------+------------+
| block 1    | blocks 2 .. K             | block K+1  |
| imbalanced | permuted blocks of        | complement |
|            | varying sizes             | of block 1 |
+------------+---------------------------+------------+
```

Block 1 and block K+1 are the two halves of one balanced pool of `2*n1` slots, so the last block is the exact complement of the first one: overall balance is guaranteed by construction, while the imbalance deliberately introduced in block 1 keeps the next allocation unpredictable.

The head-block size `n1` is always sampled at random from the admissible multiples of `ratio_sum` in `[2*ratio_sum, N/2]`, so there is no need to pre-specify it and the block structure is never revealed in advance.

Two further paths handle the boundary cases:

path 1 `N < 4 * ratio_sum`: complete randomization with a pre-determined `N`, in one single block.

path 3 `N == 2 * n1`: head and tail blocks only, no middle section.

Value

An object of class `"sandwichrand"`, a list with the elements

`call` the matched call.

`design` list describing the design actually used: `N`, `arms`, `ratio`, `ratio_sum`, `n_arms`, `path`, `n1`, `n_mid`, `n_mid_blocks`, `imbalance_observed`, `imbalance_target`, `seed`, ...

`alloc` data frame with columns `id`, `block`, `block_size` (the latter two only when `keep.blocks = TRUE`) and `arm`.

`balance` data frame with `arm`, `n`, `expected` and `deviation`.

`blocks` data frame with one row per block: `block`, `size`, `first_id`, `last_id` and the arm counts.

`notes`, `warnings` character vectors.

`spec` the normalized specification, used by `smr_reproduce`.

References

Wang B, Wang X, Ni C, Xie J, Wu S, Wu Y, Shen Y, Dong Z, Lv W, He Y (2026). "Reducing selection bias risk to enhance RCT validity: sandwich mixed randomization outperforms permuted block design." *BMC Medical Research Methodology*, **26**(1), 152. doi:10.1186/s12874026028772.

See Also

`smr_verify`, `smr_balance`, `smr_imbalance`, `smr_blocks`, `smr_rtf`, `smr_write_csv`, `smr_streams`.

Examples

```
## Two-arm 1:1 trial, N = 48 -- the illustration of the SMR paper
x <- smr_alloc(N = 48, arms = c("TestTrt", "Placebo"), seed = 84529052)
x
head(x$alloc)
x$balance

## Three-arm 2:2:1 trial, N = 150, released without the block structure
y <- smr_alloc(N = 150, arms = c("A", "A", "B", "B", "C"), seed = 123456,
               keep.blocks = FALSE)
names(y$alloc)
smr_balance(y)
```

```
## Small trial: path 1, complete randomization in one block
z <- smr_alloc(N = 9, arms = c("Drug", "Placebo"), seed = 777)
z$design$path

## High-frequency calling: derive independent streams up front so that no
## two calls ever draw from the same random stream
ss <- smr_streams(seed = 42, n = 3)
head(smr_alloc(N = 48, arms = c("A", "B"), stream = ss[[1]])$alloc)
head(smr_alloc(N = 48, arms = c("A", "B"), stream = ss[[2]])$alloc)
```

`smr_balance` *Balance summary of an allocation sequence*

Description

Returns the arm counts, the expected counts under the target allocation and the deviation of each arm from its target.

Usage

```
smr_balance(x)
```

Arguments

`x`	An object of class `"sandwichrand"`.

Value

A data frame with columns `arm`, `n`, `expected` and `deviation`.

Examples

```
x <- smr_alloc(N = 150, arms = c("A", "A", "B", "B", "C"), seed = 3)
smr_balance(x)
```

smr_blocks — *Block structure of an allocation sequence*

Description

Returns one row per block with its size, the first and last allocation ids and the arm counts. This is the statistician's copy of the block structure (kept even when `keep.blocks = FALSE` was used at generation time).

Usage

```
smr_blocks(x)
```

Arguments

`x`	An object of class `"sandwichrand"`.

Value

A data frame, or `NULL` if no block structure is stored.

Examples

```
x <- smr_alloc(N = 48, arms = c("A", "B"), seed = 1)
smr_blocks(x)
```

smr_correct_guesses — *Per-step correct guesses under the convergence strategy*

Description

Evaluates allocation unpredictability of a generated sequence with the Blackwell-Hodges "convergence" guessing strategy. At each allocation step the next assignment is predicted to go to the treatment arm that is most under-allocated relative to the target allocation ratio; ties are broken uniformly at random. `CG(i)` equals 1 when the guess is right, 0 when it is wrong, and `1/k` when `k` arms are tied for the largest deficit. For a two-arm 1:1 trial this is exactly the definition of Wang et al. (2026): guess the arm with the fewer prior allocations, with `CG(i) = 0.5` when the two counts are equal.

Usage

```
smr_correct_guesses(x)
```

Arguments

`x`	An object of class `"sandwichrand"`.

Value

A numeric vector of length `N` with the per-step correct-guess indicators `CG(1), ..., CG(N)`. The first element is always `1/n_arms` (no prior information), which is `0.5` for 1:1.

References

Blackwell D, Hodges JL (1957). Design for the control of selection bias. *Annals of Mathematical Statistics*, **28**(2), 449–460.

Wang B, Wang X, Ni C, Xie J, Wu S, Wu Y, Shen Y, Dong Z, Lv W, He Y (2026). "Reducing selection bias risk to enhance RCT validity: sandwich mixed randomization outperforms permuted block design." *BMC Medical Research Methodology*, **26**(1), 152. doi:10.1186/s12874026028772.

See Also

`smr_pcg`, `smr_imbalance`, `smr_simulate`.

Examples

```
x <- smr_alloc(N = 48, arms = c("A", "B"), seed = 84529052)
smr_correct_guesses(x)
```

---

smr_imbalance *Running imbalance of an allocation sequence*

Description

For each allocation position, computes the spread of the cumulative deviations of the arms from their targets. For a 1:1 allocation this is the running difference between the two arms; it is the quantity the SMR scheme is designed to keep small and that starts (by construction) at the head block's deliberate imbalance.

Usage

```
smr_imbalance(x)
```

Arguments

`x`	An object of class `"sandwichrand"`.

Value

A numeric vector of length `N`, the running imbalance.

Examples

```
x <- smr_alloc(N = 48, arms = c("A", "B"), seed = 1)
tail(smr_imbalance(x))
```

smr_pcg — *Proportion of correct guesses (allocation predictability)*

Description

The mean of smr_correct_guesses, i.e. the expected proportion of correct guesses (PCG). A value close to 0.5 means the sequence is about as unpredictable as complete randomization; values well above 0.5 mean the next allocation can be anticipated from the running imbalance, which is the predictability that the SMR scheme is designed to reduce.

Usage

```
smr_pcg(x)
```

Arguments

x	An object of class "sandwichrand".

Value

A single number in [0, 1], the proportion of correct guesses.

See Also

smr_correct_guesses, smr_simulate.

Examples

```
x <- smr_alloc(N = 48, arms = c("A", "B"), seed = 84529052)
smr_pcg(x)

## Expected PCG over many replications
s <- smr_simulate(N = 48, arms = c("A", "B"), reps = 200, seed = 11)
s
```

smr_reproduce — *Reproduce an allocation sequence*

Description

Re-runs the generator with the specification and the seed stored in an "sandwichrand" object. The result is identical to the original object, which is the property an auditor needs: the seed plus the package version fully determine the list.

Usage

```
smr_reproduce(x)
```

Arguments

x	An object of class "sandwichrand".

Value

A new object of class "sandwichrand", identical to x.

Examples

```
x <- smr_alloc(N = 48, arms = c("A", "B"), seed = 2026)
identical(smr_reproduce(x)$alloc, x$alloc)
```

---

smr_rtf — *Write an allocation sequence to an RTF report*

Description

Produces the deliverable randomization list: the allocation numbers with their treatment (and, optionally, the block structure), followed by the arm balance table. The seed is printed in the report, which therefore has to be handled as confidential trial material (ICH E9 / GCP).

Usage

```
smr_rtf(
  x,
  file,
  title1 = "The sandwich mixed random-allocation schedule",
  title2 = NULL,
  keep.blocks = x$design$keep.blocks
)
```

Arguments

x	An object of class "sandwichrand".
file	Path of the RTF file to write.
title1	Main title of the report.
title2	Subtitle; NULL omits the line.
keep.blocks	Logical. Print the block columns. Defaults to the setting used when the sequence was generated; FALSE releases a list without any block information.

Value

The path `file`, invisibly.

Examples

```
x <- smr_alloc(N = 12, arms = c("Drug", "Placebo"), seed = 777)
f <- tempfile(fileext = ".rtf")
smr_rtf(x, f)
```

---

smr_simulate *Assess an SMR design by simulation*

Description

Runs `reps` independent replications of `smr_alloc` with independent random streams and summarises the two key performance metrics of a randomization procedure: the final and the maximum running imbalance (allocation balance) and the proportion of correct guesses, PCG (allocation unpredictability). This is the routine to use when a design (sample size, arms, imbalance window) needs to be assessed before a trial is started.

Usage

```
smr_simulate(N, arms, reps = 100L, seed = NULL, ...)
```

Arguments

`N`	Integer. Total sample size.
`arms`	Treatment-arm specification, see `smr_alloc`.
`reps`	Integer. Number of replications.
`seed`	Integer. Base seed; the `reps` independent streams are derived from it, so the assessment is fully reproducible.
`...`	Further arguments passed to `smr_alloc`.

Value

An object of class `"smr_simulation"`, a list with `reps`, `final_imbalance`, `max_imbalance`, `pcg` (the proportion of correct guesses in each replication), the mean running curves `imbalance_curve` and `pcg_curve` (one value per allocation step), and `n1`.

Examples

```
s <- smr_simulate(N = 48, arms = c("A", "B"), reps = 200, seed = 11)
s
```

smr_spread — *The spread of deviations from the target allocation*

Description

`spread` is the difference between the largest and the smallest deviation of the arm counts from the target allocation `count_g - n * mult_g / ratio_sum`. For a 1:1 allocation it is exactly the difference of the two arm sizes (the imbalance used in the SMR paper); for unequal ratios it is the natural generalisation (`spread = 0` means exactly proportional).

Usage

```
smr_spread(cnt, n, mult, ratio_sum)
```

Arguments

`cnt`	Numeric vector of arm counts, in the declared arm order.
`n`	Numeric. Number of slots the counts are expressed for.
`mult`	Integer vector. Allocation-ratio units of each arm.
`ratio_sum`	Numeric. Sum of `mult`.

Value

A single numeric value, the spread.

Examples

```
## 1:1, 10 slots, counts 7 / 3 -> spread 4
smr_spread(c(7, 3), 10, c(1, 1), 2)
## 2:2:1, 10 slots, counts 5 / 4 / 1 -> spread 2
smr_spread(c(5, 4, 1), 10, c(2, 2, 1), 5)
```

smr_streams — *Independent random streams for high-frequency calling*

Description

Derives `n` non-overlapping L'Ecuyer-CMRG sub-streams from one base seed. Each returned value is the raw RNG state of a stream and can be passed directly to `smr_alloc` via its `stream` argument. Because the streams do not overlap, a loop that hands one stream per call never re-uses the same random numbers, which is what keeps high-frequency calling safe and reproducible.

Usage

```
smr_streams(seed, n)
```

Arguments

seed	Integer. Base random seed.
n	Integer. Number of independent streams to derive.

Value

A list of n elements, each the numeric state of an independent L'Ecuyer-CMRG stream.

See Also

smr_alloc, smr_simulate.

Examples

```
ss <- smr_streams(seed = 1, n = 3)
a <- smr_alloc(N = 48, arms = c("A", "B"), stream = ss[[1]])
b <- smr_alloc(N = 48, arms = c("A", "B"), stream = ss[[2]])
## the two sequences are different even though they share the base seed
!identical(a$alloc, b$alloc)
```

---

smr_verify — *Verify the structural invariants of an allocation sequence*

Description

Re-checks every invariant that the SMR scheme promises: the length and the numbering of the sequence, the final balance, and – when the block structure is available – the consistency of the blocks and the complementarity of the first and the last block. The function works on any "sandwichrand" object, so a list received from a third party can be re-checked independently of how it was produced.

Usage

```
smr_verify(x, stop.on.fail = FALSE)
```

Arguments

x	An object of class "sandwichrand".
stop.on.fail	Logical. Stop with an error when at least one check fails, instead of returning the table (default FALSE).

Value

A data frame of class "smr_check" with one row per check and the columns check, status ("PASS" or "FAIL") and detail. Invisibly the same value when stop.on.fail = TRUE and every check passes.

Examples

```
x <- smr_alloc(N = 48, arms = c("TestTrt", "Placebo"), seed = 84529052)
v <- smr_verify(x)
all(v$status == "PASS")
```

`smr_write_csv` *Write an allocation sequence to a CSV file*

Description

Releases the allocation list as a comma-separated file. The block columns are included only when `keep.blocks = TRUE`; otherwise the released file contains only the allocation number and the treatment, as required by ICH E9 and good clinical practice.

Usage

```
smr_write_csv(x, file, keep.blocks = x$design$keep.blocks)
```

Arguments

`x`	An object of class `"sandwichrand"`.
`file`	Path of the CSV file to write.
`keep.blocks`	Logical. Write the block columns. Defaults to the setting used when the sequence was generated.

Value

The path `file`, invisibly.

Examples

```
x <- smr_alloc(N = 12, arms = c("Drug", "Placebo"), seed = 777,
               keep.blocks = FALSE)
f <- tempfile(fileext = ".csv")
smr_write_csv(x, f)
```

Index

as.data.frame.sandwichrand
(sandwichrand-methods), 4

format.sandwichrand
(sandwichrand-methods), 4

plot.sandwichrand
(sandwichrand-methods), 4
print.sandwichrand
(sandwichrand-methods), 4
print.summary.sandwichrand
(sandwichrand-methods), 4

sandwichrand (sandwichrand-package), 2
sandwichrand-methods, *3*, 4
sandwichrand-package, 2
smr_alloc, *3, 4*, 5, *13–15*
smr_balance, *3*, *7*, 8
smr_blocks, *3*, *7*, 9
smr_correct_guesses, *3*, 9, *11*
smr_imbalance, *3*, *7*, 10, *10*
smr_pcg, *3*, *10*, 11
smr_reproduce, *7*, 11
smr_rtf, *3*, *6, 7*, 12
smr_simulate, *3*, *10, 11*, 13, *15*
smr_spread, *3*, 14
smr_streams, *3*, *6, 7*, 14
smr_verify, *3*, *7*, 15
smr_write_csv, *3*, *6, 7*, 16
summary.sandwichrand
(sandwichrand-methods), 4