

Package ‘randotools’

June 15, 2026

Title Create Randomization Lists

Version 0.2.8

Description Randomization lists are an integral component of randomized clinical trials. 'randotools' provides tools to easily create such lists.

License GPL (>= 3)

URL <https://dcr-unibe-ch.github.io/randotools/>,
<https://github.com/dcr-unibe-ch/randotools>

BugReports <https://github.com/dcr-unibe-ch/randotools/issues>

Encoding UTF-8

Imports cli, dplyr, glue, ggplot2, patchwork, purrr, rlang

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

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 4.1)

LazyData true

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Alan G Haynes [aut, cre] (ORCID:
<<https://orcid.org/0000-0003-1374-081X>>)

Maintainer Alan G Haynes <alan.haynes@unibe.ch>

Repository CRAN

Date/Publication 2026-06-15 14:00:02 UTC

Contents

check_plan	2
imbalance_seq_plots	3
imbalance_test	4
imbalance_test_plot	5

randolist	6
randolist_to_db	8
rando_balance	9
summary.randolist	10

Index	11
--------------	-----------

check_plan	<i>Check randomisation plan</i>
------------	---------------------------------

Description

Before committing to a randomisation plan (in terms of the number of strata, block sizes etc) it can be useful to estimate the imbalance that might be expected. This function simulates trials of a given sample size and returns the imbalance that might be expected.

Usage

```
check_plan(
  n_rando,
  n_strata,
  arms = c("A", "B"),
  blocksizes = c(1, 2),
  n_sim = 1000
)

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

Arguments

n_rando	number of participants to randomise
n_strata	number of strata
arms	arms that will be randomised
blocksizes	number of each randomisation group per block (e.g. 1 = one of each arm per block, 2 = per of each arm per block)
n_sim	number of simulations
x	check_plan object
...	options passed to print.data.frame

Value

list of class checkplan with slots the same slots as input to the function plus mean (mean imbalance), counts (counts of the imbalances) and worst_case (randomisation results with the worst observed imbalance)

Functions

- `print(checkplan)`: Print method for `check_plan` output

See Also

<https://www.sealedenvelope.com/randomisation/simulation/>

Examples

```
check_plan(50, 3, n_sim = 50)
```

`imbalance_seq_plots` *Depict the imbalance of a randomisation sequence through time*

Description

It can be useful to see how imbalance changes through time. This function allows such a depiction by plotting the maximum imbalance as a function of randomisation number (assuming that the observations are in the randomisation order). This is especially useful in the case of randomisation via minimisation. As well as the overall imbalance, the function also depicts the imbalance within each strata (i.e. the interaction among stratifying variables) and within strata identified by each stratifying variable itself.

Usage

```
imbalance_seq_plots(  
  data,  
  randovar,  
  stratavars = NULL,  
  cross = TRUE,  
  stack = TRUE  
)
```

Arguments

<code>data</code>	a data frame
<code>randovar</code>	variable name containing the randomisation result
<code>stratavars</code>	variable names of stratification variables
<code>cross</code>	logical whether to cross the stratification variables to create the individual strata
<code>stack</code>	logical whether to use <code>patchwork::wrap_plots</code> to combine the plots into a single 2x3 figure

Value

Up to six ggplots. Each has the randomisation sequence along the x-axis and imbalance on the y-axis. The different lines denotes different groupings. All plots are paired: the first plot shows the observed balance, the second shows the balance in a simulated dataset. There are up to three pairs of plots.

- First the overall values are shown.
- Second, each line represents a group as defined by the stratification variables. E.g., if there is a 2-level stratification variable and a 3-level variable, there will be 5 lines.
- The third pair shows the individual strata - the combination of all stratification variables. For the 2- and 3-level example mentioned above, this would result in 6 lines. This can be skipped by setting `cross` to `FALSE`. If `stack = FALSE`, a list of ggplots is returned.

Examples

```
data(rando_balance)
# without stratification variables
imbalance_seq_plots(rando_balance, "rando_res")
# with stratification factors
imbalance_seq_plots(rando_balance, "rando_res",
                    c("strat1", "strat2"))
# do not cross the stratification factors
imbalance_seq_plots(rando_balance, "rando_res",
                    c("strat1", "strat2"),
                    cross = FALSE)
```

 imbalance_test

Test the imbalance of randomisation via simulation

Description

This function tests whether the observed imbalance is less than might be expected via a random draw, via a permutation test.

Usage

```
imbalance_test(
  data,
  randovar,
  n_iter = 1000,
  stratavars = NULL,
  arms = NULL,
  cross = TRUE,
  ...
)
```

Arguments

data	a dataframe with the variables indicated in randovar and, optionally, stratavars
randovar	character with the variable name indicating the randomisation
n_iter	integer. number of simulations to perform
stratavars	character vector with the variable names indicating the stratification variables
arms	character vector of arms in the appropriate balance. If NULL the levels in randovar are used and assumed to be balanced
cross	logical. Whether to cross the stratification variables.
...	other arguments passed onto other methods

Value

a list with:

- n_rando: the number of randomisations
- stratavars: the names of the stratification variables
- arms: the arms
- observed: a dataframe with the observed imbalance
- simulated: a dataframe with the simulated imbalances (number of rows = nrow(n_iter))
- tests: a dataframe with the p-values

See Also

[imbalance_test_plot\(\)](#)

Examples

```
data(rando_balance)
# without stratification variables
imbalance_test(rando_balance, "rando_res", n_iter = 50)
imb <- imbalance_test(rando_balance, "rando_res", stratavars = "strat1", n_iter = 50)
imbalance_test(rando_balance, "rando_res", stratavars = c("strat1", "strat2"), n_iter = 50)
imb <- imbalance_test(rando_balance, "rando_res2", stratavars = c("strat1", "strat2"), n_iter = 50)
```

imbalance_test_plot *Plot imbalance and simulation and test results*

Description

Plot histograms of imbalance values from simulated random allocation and a vertical lines to indicate the observed imbalance for each randomisation level (overall, stratification variable level, and strata level, where appropriate). The p-values from the tests are included in the figure captions.

Usage

```
imbalance_test_plot(test, vline_col = "red", stack = TRUE)
```

Arguments

test	imbalance_test object
vline_col	colour for the vertical line indicating the observed imbalance
stack	logical, whether to use patchwork::wrap_plots to stack the plots in one column (TRUE) or return a list of ggplot objects (FALSE)

Value

list of ggplots or a patchwork off ggplots (if stack = TRUE)

See Also

[imbalance_test\(\)](#)

Examples

```
# example code
data(rando_balance)
# without stratification variables
imb <- imbalance_test(rando_balance, "rando_res2", stratavars = c("strat1", "strat2"), n_iter = 50)
imbalance_test_plot(imb)
```

randolist

Generate randomisation lists

Description

Randomisation lists are central to randomised trials. This function allows to generate randomisation lists simply, via (optionally) stratified block randomisation

Usage

```
randolist(
  n,
  arms = LETTERS[1:2],
  strata = NA,
  blocksizes = 1:3,
  pascal = TRUE,
  n_init = 0,
  init_probs = NULL,
  ...
)
```

Arguments

<code>n</code>	total number of randomizations (per stratum)
<code>arms</code>	arms to randomise
<code>strata</code>	named list of stratification variables (see examples)
<code>blocksizes</code>	numbers of each arm to include in blocks (see details)
<code>pascal</code>	logical, whether to use pascal's triangle to determine block sizes
<code>n_init</code>	number of blocks with different blocksize probabilities to initialize the list with
<code>init_probs</code>	probabilities to use for each blocksize for the <code>n_init</code> blocks
<code>...</code>	arguments passed on to other methods

Details

`blocksizes` defines the number of allocations to each arm in a block. For example, if there are two arms, and `blocksizes = 1`, each block will contain 2 randomisations. If `blocksizes = 1:2`, each block will contain either one of each arm, or two of each arm. Total block sizes are therefore `blocksizes * length(arms)`.

By default, frequency of the different block sizes is determined using Pascal's triangle. This has the advantage that small and large block sizes are less common than intermediate sized blocks, which helps with making it more difficult to guess future allocations, and reduces the risk of finishing in the middle of a large block. If `ifpascal = FALSE`, all `blocksizes` have the same frequency.

Unbalanced randomization is possible by specifying the same arm label multiple times.

To disable block randomisation, set `blocksizes` to the same value as `n`.

It can be helpful to use smaller blocks at the start of a randomisation list. `n_init` allows you to define how many blocks to add at the beginning of the list with a different set of blocksize probabilities (entered via `init_probs`). Note that this modifies the final frequencies of the `blocksizes` - they are no longer according to normal settings (pascals triangle, if `pascal = TRUE`, or approximately equal, if `pascal = FALSE`). Depending on the settings, it may even be that some `blocksizes` are not observed at all (e.g. if `n_init` exceeds the total number of blocks required and `init_probs = c(.8, .1, 0)`, there will be none of the larger blocks in the randomisation list.

`n_init` and `init_probs` allow more control over the blocksize probabilities than is otherwise possible (i.e. with the `pascal` argument). Suppose you want primarily blocks of size 2, with some blocks of size 4, you might set `n_init` to `n/2` (as if all blocks were of size 2, just to ensure that there are enough blocks for all randomisations) and set `init_probs = c(0.8, .2)`.

Value

object of class `randolist` which is a dataframe with additional attributes `ratio` (randomisation ratio, e.g. 1:1, 2:1), `arms` (arm labels), `stratified` (logical whether the list is stratified), and `stratavars` (the stratification variables)

Examples

```
# example code
randolist(10)
# one stratifying variable
```

```

randolist(10, strata = list(sex = c("M", "F")))
# two stratifying variables
randolist(10, strata = list(sex = c("M", "F"),
                           age = c("child", "adult")))
# different arm labels
randolist(10, arms = c("arm 1", "arm 2"))

# unbalanced (2:1) randomization
randolist(10, arms = c("arm 1", "arm 1", "arm 2"))

```

randolist_to_db

Reformat a randolist object to the requirements of a database

Description

Databases generally require a specific format to be able to import a randomization list. This function converts the randolist object to the format required by REDCap or secuTrial.

Usage

```

randolist_to_db(
  randolist,
  target_db = c("REDCap", "secuTrial"),
  strata_enc = NA,
  rando_enc = NA
)

```

Arguments

randolist	a randolist object from randolist or blockrand
target_db	the target database, either "REDCap" or "secuTrial"
strata_enc	a list of data frames with the encoding of each stratification variable. Should have two columns - the value used in randolist and code with the values used in the database. See the examples for details.
rando_enc	a data frame with the randomization encoding

Details

rando_enc should contain an arm column containing the values supplied to randolist, and a variable with the name required by the database with the values that map to those in arm. See the examples.

Value

dataframe with columns required for import into target_db

Examples

```
r <- randolist(10,
              strata = list(sex = c("M", "F")),
              arms = c("T1", "T2"))
randolist_to_db(r,
               rando_enc = data.frame(arm = c("T1", "T2"),
                                       rando_res = c(1, 2)),
               strata_enc = list(sex = data.frame(sex = c("M", "F"),
                                                  code = 1:2)),
               target_db = "REDCap")
randolist_to_db(r,
               rando_enc = data.frame(arm = c("T1", "T2"),
                                       rando_res = c(1, 2)),
               strata_enc = list(sex = data.frame(sex = c("M", "F"),
                                                  code = 1:2)),
               target_db = "secuTrial")
```

rando_balance

rando_balance demonstration dataset

Description

A synthetic dataset used in examples. The dataset contains two randomisation result variables (rando_res and rando_res2) and two stratification variables (strat1 and strat2).

Usage

```
rando_balance
```

Format

rando_balance:

A data frame with 100 rows and 3 columns:

strat1, strat2 Binary stratification variables

rando_res Balanced randomisation result

rando_res2 Unbalanced randomisation result

summary.randolist *Summary method fro randolist objects*

Description

Create a short summary report of the aspects of the randomisation list, which could be used for quality control.

Usage

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

Arguments

object	randolist object
...	additional arguments (currently unused)

Value

object of class randolistsum, which is a list with elements

- n_rando: total number of randomisations
- n_blocks: maximum number of blocks
- block_sizes: table of block sizes
- arms: table of arms
- ratio: randomisation ratio (character)
- stratified: logical
- stratavars: names of stratifying variables (character)
- stratavars_tabs: tabulation of arms by each stratification variable
- strata: names of each individual stratum
- stratum_tabs: list with an element for each strata with n_rando, n_blocks, block_sizes, arms and ratio.

Examples

```
r <- randolist(20)  
print(summary(r))  
  
r2 <- randolist(20, strata = list(sex = c("M", "F")))  
print(summary(r2))  
  
# NOTE: explicitly printing isn't technically necessary
```

Index

* datasets

- rando_balance, 9
- check_plan, 2
- imbalance_seq_plots, 3
- imbalance_test, 4
- imbalance_test(), 6
- imbalance_test_plot, 5
- imbalance_test_plot(), 5
- print.checkplan (check_plan), 2
- rando_balance, 9
- randolist, 6
- randolist_to_db, 8
- summary.randolist, 10