Package 'adaptr'

August 21, 2023

Title Adaptive Trial Simulator

Version 1.3.2

Date 2023-08-21

Description Package that simulates adaptive (multi-arm, multi-stage) clinical trials using adaptive stopping, adaptive arm dropping, and/or adaptive randomisation. Developed as part of the INCEPT (Intensive Care Platform Trial) project (<https://incept.dk/>), which is primarily supported by a grant from Sygeforsikringen ``danmark'' (<https://www.sygeforsikring.dk/>).

License GPL (>= 3)

Imports stats, parallel, utils

Encoding UTF-8

Language en-GB

NeedsCompilation no

URL https://inceptdk.github.io/adaptr/, https://github.com/INCEPTdk/adaptr/, https://incept.dk/

BugReports https://github.com/INCEPTdk/adaptr/issues/

RoxygenNote 7.2.3

Suggests ggplot2, covr, rmarkdown, knitr, testthat, vdiffr

VignetteBuilder knitr

Config/testthat/edition 3

Author Anders Granholm [aut, cre] (<https://orcid.org/0000-0001-5799-7655>), Benjamin Skov Kaas-Hansen [aut] (<https://orcid.org/0000-0003-1023-0371>), Aksel Karl Georg Jensen [ctb] (<https://orcid.org/0000-0002-1459-0465>), Theis Lange [ctb] (<https://orcid.org/0000-0001-6807-8347>)

Maintainer Anders Granholm <andersgran@gmail.com>

Repository CRAN

Date/Publication 2023-08-21 12:10:04 UTC

R topics documented:

`adaptr-package` *adaptr: Adaptive Trial Simulator*

Description

The `adaptr` package simulates adaptive (multi-arm, multi-stage) randomised clinical trials using adaptive stopping, adaptive arm dropping and/or response-adaptive randomisation. The package is developed as part of the INCEPT (Intensive Care Platform Trial) project, funded primarily by a grant from Sygeforsikringen "danmark".

Details

The `adaptr` package contains the following primary functions (in order of typical use):

1. The `setup_cluster()` initiates a parallel computation cluster that can be used to run simulations and post-processing in parallel, increasing speed. Details on parallelisation and other options for running `adaptr` functions in parallel are described in the `setup_cluster()` documentation.
2. The `setup_trial()` function is the general function that sets up a trial specification. The simpler, special-case functions `setup_trial_binom()` and `setup_trial_norm()` may be used for easier specification of trial designs using binary, binomially distributed or continuous, normally distributed outcomes, respectively, with some limitations in flexibility.

3. The `calibrate_trial()` function calibrates a trial specification to obtain a certain value for a performance metric (typically used to calibrate the Bayesian type 1 error rate in a scenario with no between-arm differences), using the functions below.
4. The `run_trial()` and `run_trials()` functions are used to conduct single or multiple simulations, respectively, according to a trial specification setup as described in #2.
5. The `extract_results()`, `check_performance()` and `summary()` functions are used to extract results from multiple trial simulations, calculate performance metrics, and summarise results. The `plot_convergence()` function assesses stability of performance metrics according to the number of simulations conducted. The `plot_metrics_ecdf()` function plots empirical cumulative distribution functions for numerical performance metrics. The `check_remaining_arms()` function summarises all combinations of remaining arms across multiple trials simulations.
6. The `plot_status()` and `plot_history()` functions are used to plot the overall trial/arm statuses for multiple simulated trials or the history of trial metrics over time for single/multiple simulated trials, respectively.

For further information see the documentation of each function or the **Overview** vignette (`vignette("Overview", package = "adaptr")`) for an example of how the functions work in combination. For further examples and guidance on setting up trial specifications, see the `setup_trial()` documentation, the **Basic examples** vignette (`vignette("Basic-examples", package = "adaptr")`) and the **Advanced example** vignette (`vignette("Advanced-example", package = "adaptr")`).

If using the package, please consider citing it using `citation(package = "adaptr")`.

References

Granholm A, Jensen AKG, Lange T, Kaas-Hansen BS (2022). adaptr: an R package for simulating and comparing adaptive clinical trials. Journal of Open Source Software, 7(72), 4284. doi:10.21105/joss.04284

Granholm A, Kaas-Hansen BS, Lange T, Schjørring OL, Andersen LW, Perner A, Jensen AKG, Møller MH (2022). An overview of methodological considerations regarding adaptive stopping, arm dropping and randomisation in clinical trials. J Clin Epidemiol. doi:10.1016/j.jclinepi.2022.11.002

Website/manual

GitHub repository

See Also

`setup_cluster()`, `setup_trial()`, `setup_trial_binom()`, `setup_trial_norm()`, `calibrate_trial()`, `run_trial()`, `run_trials()`, `extract_results()`, `check_performance()`, `summary()`, `check_remaining_arms()`, `plot_convergence()`, `plot_metrics_ecdf()`, `print()`, `plot_status()`, `plot_history()`.

`calibrate_trial` *Calibrate trial specification*

Description

This function calibrates a trial specification using a Gaussian process-based Bayesian optimisation algorithm. The function calibrates an input trial specification object (using repeated calls to `run_trials()` while adjusting the trial specification) to a `target` value within a `search_range` in a single input dimension (`x`) in order to find an optimal value (`y`).
The default (and expectedly most common use case) is to calibrate a trial specification to adjust the `superiority` and `inferiority` thresholds to obtain a certain probability of superiority; if used with a trial specification with identical underlying outcomes (no between-arm differences), this probability is an estimate of the Bayesian analogue of the total type-1 error rate for the outcome driving the adaptations, and if between-arm differences are present, this corresponds to an estimate of the Bayesian analogue of the power.
The default is to perform the calibration while varying single, constant, symmetric thresholds for `superiority` / `inferiority` throughout a trial design, as described in **Details**, and the default values have been chosen to function well in this case.
Advanced users may use the function to calibrate trial specifications according to other metrics - see **Details** for how to specify a custom function used to modify (or recreate) a trial specification object during the calibration process.
The underlying Gaussian process model and its control hyperparameters are described under **Details**, and the model is partially based on code from Gramacy 2020 (with permission; see **References**).

Usage

```
calibrate_trial(
  trial_spec,
  n_rep = 1000,
  cores = NULL,
  base_seed = NULL,
  fun = NULL,
  target = 0.05,
  search_range = c(0.9, 1),
  tol = target/10,
  dir = 0,
  init_n = 2,
  iter_max = 25,
  resolution = 5000,
  kappa = 0.5,
  pow = 1.95,
  lengthscale = 1,
  scale_x = TRUE,
  noisy = is.null(base_seed),
  narrow = !noisy & !is.null(base_seed),
  prev_x = NULL,
  prev_y = NULL,
  path = NULL,
  overwrite = FALSE,
  version = NULL,
  compress = TRUE,
```

```
  sparse = TRUE,
  progress = NULL,
  export = NULL,
  export_envir = parent.frame(),
  verbose = FALSE,
  plot = FALSE
)
```

Arguments

trial_spec	trial_spec object, generated and validated by the setup_trial(), setup_trial_binom() or setup_trial_norm() function.
n_rep	single integer, the number of simulations to run at each evaluation. Values < 100 are not permitted; values < 1000 are permitted but recommended against.
cores	NULL or single integer. If NULL, a default value/cluster set by setup_cluster() will be used to control whether simulations are run in parallel on a default cluster or sequentially in the main process; if a cluster/value has not been specified by setup_cluster(), cores will then be set to the value stored in the global "mc.cores" option (if previously set by options(mc.cores = <number of cores>), and 1 if that option has not been specified. If the resulting number of cores = 1, computations will be run sequentially in the primary process, and if cores > 1, a new parallel cluster will be setup using the parallel library and removed once the function completes. See setup_cluster() for details.
base_seed	single integer or NULL (default); the random seed used as the basis for all simulation runs (see run_trials()) and random number generation within the rest of the calibration process; if used, the global random seed will be restored after the function has been run. **Note:** providing a base_seed is highly recommended, as this will generally lead to faster and more stable calibration.
fun	NULL (the default), in which case the trial specification will be calibrated using the default process described above and further in **Details**; otherwise a user-supplied function used during the calibration process, which should have a structure as described in **Details**.
target	single finite numeric value (defaults to 0.05); the target value for y to calibrate the trial_spec object to.
search_range	finite numeric vector of length 2; the lower and upper boundaries in which to search for the best x. Defaults to c(0.9, 1.0).
tol	single finite numeric value (defaults to target / 10); the accepted tolerance (in the direction(s) specified by dir) accepted; when a y-value within the accepted tolerance of the target is obtained, the calibration stops. **Note:** tol should be specified to be sensible considering n_rep; e.g., if the probability of superiority is targeted with n_rep == 1000, a tol of 0.01 will correspond to 10 simulated trials. A too low tol relative to n_rep may lead to very slow calibration or calibration that cannot succeed regardless of the number of iterations. **Important:** even when a large number of simulations are conducted, using a

very low `tol` may lead to calibration not succeeding as it may also be affected by other factors, e.g., the total number of simulated patients, the possible maximum differences in simulated outcomes, and the number of posterior draws (`n_draws` in the `setup_trial()` family of functions), which affects the minimum differences in posterior probabilities when simulating trials and thus can affect calibration, including when using the default calibration function. Increasing the number of posterior draws or the number of repetitions should be attempted if the desired tolerance cannot be achieved with lower numbers.

`dir` single numeric value; specifies the direction(s) of the tolerance range. If `0` (the default) the tolerance range will be `target - tol` to `target + tol`. If `< 0`, the range will be `target - tol` to `target`, and if `> 0`, the range will be `target` to `target + tol`.

`init_n` single integer `>= 2`. The number of initial evaluations evenly spread over the `search_range`, with one evaluation at each boundary (thus, the default value of `2` is the minimum permitted value; if calibrating according to a different target than the default, a higher value may be sensible).

`iter_max` single integer `> 0` (default `25`). The maximum number of new evaluations after the initial grid (with size specified by `init_n`) has been set up. If calibration is unsuccessful after the maximum number of iterations, the `prev_x` and `prev_y` arguments (described below) may be used to to start a new calibration process re-using previous evaluations.

`resolution` single integer (defaults to `5000`), size of the grid at which the predictions used to select the next value to evaluate at are made.
Note: memory use will substantially increase with higher values. See also the `narrow` argument below.

`kappa` single numeric value `> 0` (default `0.5`); corresponding to the width of the uncertainty bounds used to find the next target to evaluate. See **Details**.

`pow` single numerical value in the `[1, 2]` range (default `1.95`), controlling the smoothness of the Gaussian process. See **Details**.

`lengthscale` single numerical value (defaults to `1`) or numerical vector of length `2`; values must be finite and non-negative. If a single value is provided, this will be used as the `lengthscale` hyperparameter; if a numerical vector of length `2` is provided, the second value must be higher than the first and the optimal `lengthscale` in this range will be found using an optimisation algorithm. If any value is `0`, a small amount of noise will be added as lengthscales must be `> 0`. Controls smoothness in combination with `pow`. See **Details**.

`scale_x` single logical value; if `TRUE` (the default) the `x`-values will be scaled to the `[0, 1]` range according to the minimum/maximum values provided. If `FALSE`, the model will use the original scale. If distances on the original scale are small, scaling may be preferred. The returned values will always be on the original scale. See **Details**.

`noisy` single logical value; if `FALSE`, a noiseless process is assumed, and interpolation between values is performed (i.e., with no uncertainty at the `x`-values assumed). If `TRUE`, the `y`-values are assumed to come from a noisy process, and regression is performed (i.e., some uncertainty at the evaluated `x`-values will be assumed and included in the predictions). Specifying `FALSE` requires a `base_seed` supplied,

and is generally recommended, as this will usually lead to faster and more stable calibration. If a low `n_rep` is used (or if trials are calibrated to other metrics other than the default), specifying `TRUE` may be necessary even when using a valid `base_seed`. Defaults to `TRUE` if a `base_seed` is supplied and `FALSE` if not.

`narrow` single logical value. If `FALSE`, predictions are evenly spread over the full `x`-range. If `TRUE`, the prediction grid will be spread evenly over an interval consisting of the two `x`-values with corresponding `y`-values closest to the target in opposite directions. Can only be `TRUE` when a `base_seed` is provided and `noisy` is `FALSE` (the default value is `TRUE` in that case, otherwise it is `FALSE`), and only if the function can safely be assumed to be only monotonically increasing or decreasing (which is generally reasonable if the default is used for `fun`), in which case this will lead to a faster search and a smoother prediction grid in the relevant region without increasing memory use.

`prev_x, prev_y` numeric vectors of equal lengths, corresponding to previous evaluations. If provided, these will be used in the calibration process (added before the initial grid is setup, with values in the grid matching values in `prev_x` leading to those evaluations being skipped).

`path` single character string or `NULL` (the default); if a valid file path is provided, the calibration results will either be saved to this path (if the file does not exist or if `overwrite` is `TRUE`, see below) or the previous results will be loaded and returned (if the file exists, `overwrite` is `FALSE`, and if the input `trial_spec` and central control settings are identical to the previous run, otherwise an error is produced). Results are saved/loaded using the `saveRDS()` / `readRDS()` functions.

`overwrite` single logical, defaults to `FALSE`, in which case previous results are loaded if a valid file path is provided in `path` and the object in `path` contains the same input `trial_spec` and the previous calibration used the same central control settings (otherwise, the function errors). If `TRUE` and a valid file path is provided in `path`, the complete calibration function will be run with results saved using `saveRDS()`, regardless of whether or not a previous result was saved in `path`.

`version` passed to `saveRDS()` when saving calibration results, defaults to `NULL` (as in `saveRDS()`), which means that the current default version is used. Ignored if calibration results are not saved.

`compress` passed to `saveRDS()` when saving calibration results, defaults to `TRUE` (as in `saveRDS()`), see `saveRDS()` for other options. Ignored if calibration results are not saved.

`sparse, progress, export, export_envir`
passed to `run_trials()`, see description there.

`verbose` single logical, defaults to `FALSE`. If `TRUE`, the function will print details on calibration progress.

`plot` single logical, defaults to `FALSE`. If `TRUE`, the function will print plots of the Gaussian process model predictions and return them as part of the final object; requires the `ggplot2` package installed.

Details

Default calibration

If fun is NULL (as default), the default calibration strategy will be employed. Here, the target y is the probability of superiority (as described in check_performance() and summary()), and the function will calibrate constant stopping thresholds for superiority and inferiority (as described in setup_trial(), setup_trial_binom(), and setup_trial_norm()), which corresponds to the Bayesian analogues of the type 1 error rate if there are no differences between arms in the trial specification, which we expect to be the most common use case, or the power, if there are differences between arms in the trial specification.

The stopping calibration process will, in the default case, use the input x as the stopping threshold for superiority and 1 - x as the stopping threshold for inferiority, respectively, i.e., stopping thresholds will be constant and symmetric.

The underlying default function calibrated is typically essentially noiseless if a high enough number of simulations are used with an appropriate random base_seed, and generally monotonically decreasing. The default values for the control hyperparameters have been set to normally work well in this case (including init_n, kappa, pow, lengthscale, narrow, scale_x, etc.). Thus, few initial grid evaluations are used in this case, and if a base_seed is provided, a noiseless process is assumed and narrowing of the search range with each iteration is performed, and the uncertainty bounds used in the acquisition function (corresponding to quantiles from the posterior predictive distribution) are relatively narrow.

Specifying calibration functions

A user-specified calibration function should have the following structure:

```
# The function must take the arguments x and trial_spec
# trial_spec is the original trial_spec object which should be modified
# (alternatively, it may be re-specified, but the argument should still
# be included, even if ignored)
function(x, trial_spec) {
  # Calibrate trial_spec, here as in the default function
  trial_spec$superiority <- x
  trial_spec$inferiority <- 1 - x

  # If relevant, known y values corresponding to specific x values may be
  # returned without running simulations (here done as in the default
  # function). In that case, a code block line the one below can be included,
  # with changed x/y values - of note, the other return values should not be
  # changed
  if (x == 1) {
    return(list(sims = NULL, trial_spec = trial_spec, y = 0))
  }

  # Run simulations - this block should be included unchanged
  sims <- run_trials(trial_spec, n_rep = n_rep, cores = cores,
                     base_seed = base_seed, sparse = sparse,
                     progress = progress, export = export,
                     export_envir = export_envir)
```

```
 # Return results - only the y value here should be changed
 # summary() or check_performance() will often be used here
 list(sims = sims, trial_spec = trial_spec,
      y = summary(sims)$prob_superior)
}
```

Note: changes to the trial specification are **not validated**; users who define their own calibration function need to ensure that changes to calibrated trial specifications does not lead to invalid values; otherwise, the procedure is prone to error when simulations are run. Especially, users should be aware that changing `true_ys` in a trial specification generated using the simplified `setup_trial_binom()` and `setup_trial_norm()` functions requires changes in multiple places in the object, including in the functions used to generate random outcomes, and in these cases (and otherwise if in doubt) re-generating the `trial_spec` instead of modifying should be preferred as this is safer and leads to proper validation.

Note: if the `y` values corresponding to certain `x` values are known, then the user may directly return these values without running simulations (e.g., in the default case an `x` of `1` will require `>100%` or `<0%` probabilities for stopping rules, which is impossible, and hence the `y` value in this case is by definition `1`).

Gaussian process optimisation function and control hyperparameters

The calibration function uses a relatively simple Gaussian optimisation function with settings that should work well for the default calibration function, but can be changed as required, which should be considered if calibrating according to other targets (effects of using other settings may be evaluated in greater detail by setting `verbose` and `plot` to `TRUE`).
The function may perform both interpolation (i.e., assuming a noiseless, deterministic process with no uncertainty at the values already evaluated) or regression (i.e., assuming a noisy, stochastic process), controlled by the `noisy` argument.

The covariance matrix (or kernel) is defined as:

```
exp(-||x - x'||^pow / lengthscale)
```

with `||x -x'||` corresponding to a matrix containing the absolute Euclidean distances of values of `x` (and values on the prediction grid), scaled to the `[0, 1]` range if `scale_x` is `TRUE` and on their original scale if `FALSE`. Scaling i generally recommended (as this leads to more comparable and predictable effects of `pow` and `lengthscale`, regardless of the true scale), and also recommended if the range of values is smaller than this range. The absolute distances are raised to the power `pow`, which must be a value in the `[1, 2]` range. Together with `lengthscale`, `pow` controls the smoothness of the Gaussian process model, with `1` corresponding to less smoothing (i.e., piecewise straight lines between all evaluations if `lengthscale` is `1`) and values `> 1` corresponding to more smoothing. After raising the absolute distances to the chosen power `pow`, the resulting matrix is divided by `lengthscale`. The default is `1` (no change), and values `< 1` leads to faster decay in correlations and thus less smoothing (more wiggly fits), and values `> 1` leads to more smoothing (less wiggly fits). If a single specific value is supplied for `lengthscale` this is used; if a range of values is provided, a secondary optimisation process determines the value to use within that range.

Some minimal noise ("jitter") is always added to the diagonals of the matrices where relevant to ensure numerical stability; if `noisy` is `TRUE`, a "nugget" value will be determined using a secondary optimisation process

Predictions will be made over an equally spaced grid of `x` values of size `resolution`; if `narrow` is `TRUE`, this grid will only be spread out between the `x` values with corresponding `y` values closest to and below and closes to and above `target`, respectively, leading to a finer grid in the range of relevance (as described above, this should only be used for processes that are assumed to be noiseless and should only be used if the process can safely be assumed to be monotonically increasing or decreasing within the `search_range`). To suggest the next `x` value for evaluations, the function uses an acquisition function based on bi-directional uncertainty bounds (posterior predictive distributions) with widths controlled by the `kappa` hyperparameter. Higher `kappa`/wider uncertainty bounds leads to increased *exploration* (i.e., the algorithm is more prone to select values with high uncertainty, relatively far from existing evaluations), while lower `kappa`/narrower uncertainty bounds leads to increased *exploitation* (i.e., the algorithm is more prone to select values with less uncertainty, closer to the best predicted mean values). The value in the `x` grid leading with one of the boundaries having the smallest absolute distance to the `target` is chosen (within the narrowed range, if `narrow` is `TRUE`). See Greenhill et al, 2020 under **References** for a general description of acquisition functions.

IMPORTANT: we recommend that control hyperparameters are explicitly specified, even for the default calibration function. Although the default values should be sensible for the default calibration function, these may change in the future. Further, we generally recommend users to perform small-scale comparisons (i.e., with fewer simulations than in the final calibration) of the calibration process with different hyperparameters for specific use cases beyond the default (possibly guided by setting the `verbose` and `plot` options to `TRUE`) before running a substantial number of calibrations or simulations, as the exact choices may have important influence on the speed and likelihood of success of the calibration process.
It is the responsibility of the user to specify sensible values for the settings and hyperparameters.

Value

A list of special class `"trial_calibration"`, which contains the following elements that can be extracted using `$` or `[[`:

- `success`: single logical, `TRUE` if the calibration succeeded with the best result being within the tolerance range, `FALSE` if the calibration process ended after all allowed iterations without obtaining a result within the tolerance range.
- `best_x`: single numerical value, the `x`-value (on the original, input scale) at which the best `y`-value was found, regardless of `success`.
- `best_y`: single numerical value, the best `y`-value obtained, regardless of `success`.
- `best_trial_spec`: the best calibrated version of the original `trial_spec` object supplied, regardless of `success` (i.e., the returned trial specification object is only adequately calibrated if `success` is `TRUE`).
- `best_sims`: the trial simulation results (from `run_trials()`) leading to the best `y`-value, regardless of `success`. If no new simulations have been conducted (e.g., if the best `y`-value is from one of the `prev_y`-values), this will be `NULL`.

- `evaluations`: a two-column `data.frame` containing the variables `x` and `y`, corresponding to all `x`-values and `y`-values (including values supplied through `prev_x`/`prev_y`).
- `input_trial_spec`: the unaltered, uncalibrated, original `trial_spec`-object provided to the function.
- `elapsed_time`: the total run time of the calibration process.
- `control`: list of the most central settings provided to the function.
- `fun`: the function used for calibration; if `NULL` was supplied when starting the calibration, the default function (described in **Details**) is returned after being used in the function.
- `adaptr_version`: the version of the `adaptr` package used to run the calibration process.
- `plots`: list containing `ggplot2` plot objects of each Gaussian process suggestion step, only included if `plot` is `TRUE`.

References

Gramacy RB (2020). Chapter 5: Gaussian Process Regression. In: Surrogates: Gaussian Process Modeling, Design and Optimization for the Applied Sciences. Chapman Hall/CRC, Boca Raton, Florida, USA. Available online.

Greenhill S, Rana S, Gupta S, Vellanki P, Venkatesh S (2020). Bayesian Optimization for Adaptive Experimental Design: A Review. IEEE Access, 8, 13937-13948. doi:10.1109/ACCESS.2020.2966228

Examples

```
## Not run:
# Setup a trial specification to calibrate
# This trial specification has similar event rates in all arms
# and as the default calibration settings are used, this corresponds to
# assessing the Bayesian type 1 error rate for this design and scenario
binom_trial <- setup_trial_binom(arms = c("A", "B"),
                                 true_ys = c(0.25, 0.25),
                                 data_looks = 1:5 * 200)

# Run calibration using default settings for most parameters
res <- calibrate_trial(binom_trial, n_rep = 1000, base_seed = 23)

# Print calibration summary result
res

## End(Not run)
```

`check_performance` *Check performance metrics for trial simulations*

Description

Calculates performance metrics for a trial specification based on simulation results from the `run_trials()` function, with bootstrapped uncertainty measures if requested. Uses `extract_results()`, which may be used directly to extract key trial results without summarising. This function is also used by `summary()` to calculate the performance metrics presented by that function.

Usage

```
check_performance(
  object,
  select_strategy = "control if available",
  select_last_arm = FALSE,
  select_preferences = NULL,
  te_comp = NULL,
  raw_ests = FALSE,
  final_ests = NULL,
  restrict = NULL,
  uncertainty = FALSE,
  n_boot = 5000,
  ci_width = 0.95,
  boot_seed = NULL,
  cores = NULL
)
```

Arguments

`object` `trial_results` object, output from the `run_trials()` function.

`select_strategy`
single character string. If a trial was not stopped due to superiority (or had only 1 arm remaining, if `select_last_arm` is set to `TRUE` in trial designs with a common `control` arm; see below), this parameter specifies which arm will be considered selected when calculating trial design performance metrics, as described below; this corresponds to the consequence of an inconclusive trial, i.e., which arm would then be used in practice.
The following options are available and must be written exactly as below (case sensitive, cannot be abbreviated):

- `"control if available"` (default): selects the **first** `control` arm for trials with a common `control` arm **if** this arm is active at end-of-trial, otherwise no arm will be selected. For trial designs without a common `control`, no arm will be selected.
- `"none"`: selects no arm in trials not ending with superiority.
- `"control"`: similar to `"control if available"`, but will throw an error if used for trial designs without a common `control` arm.
- `"final control"`: selects the **final** `control` arm regardless of whether the trial was stopped for practical equivalence, futility, or at the maximum sample size; this strategy can only be specified for trial designs with a common `control` arm.

- "control or best": selects the **first** control arm if still active at end-of-trial, otherwise selects the best remaining arm (defined as the remaining arm with the highest probability of being the best in the last adaptive analysis conducted). Only works for trial designs with a common control arm.
- "best": selects the best remaining arm (as described under "control or best").
- "list or best": selects the first remaining arm from a specified list (specified using select_preferences, technically a character vector). If none of these arms are are active at end-of-trial, the best remaining arm will be selected (as described above).
- "list": as specified above, but if no arms on the provided list remain active at end-of-trial, no arm is selected.

select_last_arm
: single logical, defaults to FALSE. If TRUE, the only remaining active arm (the last control) will be selected in trials with a common control arm ending with equivalence or futility, before considering the options specified in select_strategy. Must be FALSE for trial designs without a common control arm.

select_preferences
: character vector specifying a number of arms used for selection if one of the "list or best" or "list" options are specified for select_strategy. Can only contain valid arms available in the trial.

te_comp
: character string, treatment-effect comparator. Can be either NULL (the default) in which case the **first** control arm is used for trial designs with a common control arm, or a string naming a single trial arm. Will be used when calculating sq_err_te (the squared error of the treatment effect comparing the selected arm to the comparator arm, as described below).

raw_ests
: single logical. If FALSE (default), the posterior estimates (post_ests or post_ests_all, see setup_trial() and run_trial()) will be used to calculate sq_err (the squared error of the estimated compared to the specified effect in the selected arm) and sq_err_te (the squared error of the treatment effect comparing the selected arm to the comparator arm, as described for te_comp and below). If TRUE, the raw estimates (raw_ests or raw_ests_all, see setup_trial() and run_trial()) will be used instead of the posterior estimates.

final_ests
: single logical. If TRUE (recommended) the final estimates calculated using outcome data from all patients randomised when trials are stopped are used (post_ests_all or raw_ests_all, see setup_trial() and run_trial()); if FALSE, the estimates calculated for each arm when an arm is stopped (or at the last adaptive analysis if not before) using data from patients having reach followed up at this time point and not all patients randomised are used (post_ests or raw_ests, see setup_trial() and run_trial()). If NULL (the default), this argument will be set to FALSE if outcome data are available immediate after randomisation for all patients (for backwards compatibility, as final posterior estimates may vary slightly in this situation, even if using the same data); otherwise it will be said to TRUE. See setup_trial() for more details on how these estimates are calculated.

`restrict` single character string or `NULL`. If `NULL` (default), results are summarised for all simulations; if `"superior"`, results are summarised for simulations ending with superiority only; if `"selected"`, results are summarised for simulations ending with a selected arm only (according to the specified arm selection strategy for simulations not ending with superiority). Some summary measures (e.g., `prob_conclusive`) have substantially different interpretations if restricted, but are calculated nonetheless.

`uncertainty` single logical; if `FALSE` (default) uncertainty measures are not calculated, if `TRUE`, non-parametric bootstrapping is used to calculate uncertainty measures.

`n_boot` single integer (default `5000`); the number of bootstrap samples to use if `uncertainty = TRUE`. Values `< 100` are not allowed and values `< 1000` will lead to a warning, as results are likely to be unstable in those cases.

`ci_width` single numeric `>= 0` and `< 1`, the width of the percentile-based bootstrapped confidence intervals. Defaults to `0.95`, corresponding to 95% confidence intervals.

`boot_seed` single integer, `NULL` (default), or `"base"`. If a value is provided, this value will be used to initiate random seeds when bootstrapping with the global random seed restored after the function has run. If `"base"` is specified, the `base_seed` specified in `run_trials()` is used. Regardless of whether simulations are run sequentially or in parallel, bootstrapped results will be identical if a `boot_seed` is specified.

`cores` `NULL` or single integer. If `NULL`, a default value set by `setup_cluster()` will be used to control whether extractions of simulation results are done in parallel on a default cluster or sequentially in the main process; if a value has not been specified by `setup_cluster()`, `cores` will then be set to the value stored in the global `"mc.cores"` option (if previously set by `options(mc.cores = <number of cores>)`, and `1` if that option has not been specified.
If `cores = 1`, computations will be run sequentially in the primary process, and if `cores > 1`, a new parallel cluster will be setup using the `parallel` library and removed once the function completes. See `setup_cluster()` for details.

Details

The ideal design percentage (IDP) returned is based on *Viele et al, 2020* doi:10.1177/1740774519877836 (and also described in *Granholm et al, 2022* doi:10.1016/j.jclinepi.2022.11.002, which also describes the other performance measures) and has been adapted to work for trials with both desirable/undesirable outcomes and non-binary outcomes. Briefly, the expected outcome is calculated as the sum of the true outcomes in each arm multiplied by the corresponding selection probabilities (ignoring simulations with no selected arm). The IDP is then calculated as:

- For desirable outcomes (`highest_is_best` is `TRUE`):
  `100 * (expected outcome - lowest true outcome) / (highest true outcome - lowest true outcome)`
- For undesirable outcomes (`highest_is_best` is `FALSE`):
  `100 - IDP calculated for desirable outcomes`

Value

A tidy data.frame with added class trial_performance (to control the number of digits printed, see print()), with the columns "metric" (described below), "est" (estimate of each metric), and the following four columns if uncertainty = TRUE: "err_sd"(bootstrapped SDs), "err_mad" (bootstrapped MAD-SDs, as described in setup_trial() and stats::mad()), "lo_ci", and "hi_ci", the latter two corresponding to the lower/upper limits of the percentile-based bootstrapped confidence intervals. Bootstrap estimates are **not** calculated for the mininum (_p0) and maximum values (_p100) of size, sum_ys, and ratio_ys, as non-parametric bootstrapping for mininum/maximum values is not sensible - bootstrap estimates for these values will be NA.
The following performance metrics are calculated:

- n_summarised: the number of simulations summarised.
- size_mean, size_sd, size_median, size_p25, size_p75, size_p0, size_p100: the mean, standard deviation, median as well as 25-, 75-, 0- (min), and 100- (max) percentiles of the sample sizes (number of patients randomised in each simulated trial) of the summarised trial simulations.
- sum_ys_mean, sum_ys_sd, sum_ys_median, sum_ys_p25, sum_ys_p75, sum_ys_p0, sum_ys_p100: the mean, standard deviation, median as well as 25-, 75-, 0- (min), and 100- (max) percentiles of the total sum_ys across all arms in the summarised trial simulations (e.g., the total number of events in trials with a binary outcome, or the sums of continuous values for all patients across all arms in trials with a continuous outcome). Always uses all outcomes from all randomised patients regardless of whether or not all patients had outcome data available at the time of trial stopping (corresponding to sum_ys_all in results from run_trial()).
- ratio_ys_mean, ratio_ys_sd, ratio_ys_median, ratio_ys_p25, ratio_ys_p75, ratio_ys_p0, ratio_ys_p100: the mean, standard deviation, median as well as 25-, 75-, 0- (min), and 100- (max) percentiles of the final ratio_ys (sum_ys as described above divided by the total number of patients randomised) across all arms in the summarised trial simulations.
- prob_conclusive: the proportion (0 to 1) of conclusive trial simulations, i.e., simulations not stopped at the maximum sample size without a superiority, equivalence or futility decision.
- prob_superior, prob_equivalence, prob_futility, prob_max: the proportion (0 to 1) of trial simulations stopped for superiority, equivalence, futility or inconclusive at the maximum allowed sample size, respectively.
  Note: Some metrics may not make sense if summarised simulation results are restricted.
- prob_select_*: the selection probabilities for each arm and for no selection, according to the specified selection strategy. Contains one element per arm, named prob_select_arm_<arm name> and prob_select_none for the probability of selecting no arm.
- rmse, rmse_te: the root mean squared error of the estimates for the selected arm and for the treatment effect, as described in extract_results().
- idp: the ideal design percentage (IDP; 0-100%), see **Details**.

See Also

extract_results(), summary(), plot_convergence(), plot_metrics_ecdf(), check_remaining_arms().

Examples

```
# Setup a trial specification
binom_trial <- setup_trial_binom(arms = c("A", "B", "C", "D"),
                                 control = "A",
                                 true_ys = c(0.20, 0.18, 0.22, 0.24),
                                 data_looks = 1:20 * 100)

# Run 10 simulations with a specified random base seed
res <- run_trials(binom_trial, n_rep = 10, base_seed = 12345)

# Check performance measures, without assuming that any arm is selected in
# the inconclusive simulations, with bootstrapped uncertainty measures
# (unstable in this example due to the very low number of simulations
# summarised):
check_performance(res, select_strategy = "none", uncertainty = TRUE,
n_boot = 1000, boot_seed = "base")
```

---

check_remaining_arms *Check remaining arm combinations*

Description

This function summarises the numbers and proportions of all combinations of remaining arms (i.e., excluding arms dropped for inferiority or futility at any analysis, and arms dropped for equivalence at earlier analyses in trials with a common `control`) across multiple simulated trial results. The function supplements the `extract_results()`, `check_performance()`, and `summary()` functions, and is especially useful for designs with `> 2` arms, where it provides details that the other functionality mentioned do not.

Usage

```
check_remaining_arms(object, ci_width = 0.95)
```

Arguments

`object`	`trial_results` object, output from the `run_trials()` function.
`ci_width`	single numeric `>= 0` and `< 1`, the width of the approximate confidence intervals for the proportions of combinations (calculated analytically). Defaults to `0.95`, corresponding to 95% confidence intervals.

Value

a `data.frame` containing the combinations of remaining arms, sorted in descending order of, with the following columns:

- arm_*, one column per arm, each named as arm_<arm name>. These columns will contain an empty character string "" for dropped arms (including arms dropped at the final analysis), and otherwise be "superior", "control", "equivalence" (only if equivalent at the final analysis), or "active", as described in run_trial().
- n integer vector, number of trial simulations ending with the combination of remaining arms as specified by the preceding columns.
- prop numeric vector, the proportion of trial simulations ending with the combination of remaining arms as specified by the preceding columns.
- se,lo_ci,hi_ci: the standard error of prop and the confidence intervals of the width specified by ci_width.

See Also

extract_results(), check_performance(), summary(), plot_convergence(), plot_metrics_ecdf().

Examples

```
# Setup a trial specification
binom_trial <- setup_trial_binom(arms = c("A", "B", "C", "D"),
                                 control = "A",
                                 true_ys = c(0.20, 0.18, 0.22, 0.24),
                                 data_looks = 1:20 * 200,
                                 equivalence_prob = 0.7,
                                 equivalence_diff = 0.03,
                                 equivalence_only_first = FALSE)

# Run 35 simulations with a specified random base seed
res <- run_trials(binom_trial, n_rep = 25, base_seed = 12345)

# Check remaining arms (printed with fewer digits)
print(check_remaining_arms(res), digits = 3)
```

---

extract_results *Extract simulation results*

Description

This function extracts relevant information from multiple simulations of the same trial specification in a tidy data.frame (1 simulation per row). See also the check_performance() and summary() functions, that uses the output from this function to further summarise simulation results.

Usage

```
extract_results(
  object,
  select_strategy = "control if available",
  select_last_arm = FALSE,
```

```
  select_preferences = NULL,
  te_comp = NULL,
  raw_ests = FALSE,
  final_ests = NULL,
  cores = NULL
)
```

Arguments

`object` `trial_results` object, output from the `run_trials()` function.

`select_strategy`

single character string. If a trial was not stopped due to superiority (or had only 1 arm remaining, if `select_last_arm` is set to `TRUE` in trial designs with a common `control` arm; see below), this parameter specifies which arm will be considered selected when calculating trial design performance metrics, as described below; this corresponds to the consequence of an inconclusive trial, i.e., which arm would then be used in practice.
The following options are available and must be written exactly as below (case sensitive, cannot be abbreviated):

- `"control if available"` (default): selects the **first** `control` arm for trials with a common `control` arm **if** this arm is active at end-of-trial, otherwise no arm will be selected. For trial designs without a common `control`, no arm will be selected.
- `"none"`: selects no arm in trials not ending with superiority.
- `"control"`: similar to `"control if available"`, but will throw an error if used for trial designs without a common `control` arm.
- `"final control"`: selects the **final** `control` arm regardless of whether the trial was stopped for practical equivalence, futility, or at the maximum sample size; this strategy can only be specified for trial designs with a common `control` arm.
- `"control or best"`: selects the **first** `control` arm if still active at end-of-trial, otherwise selects the best remaining arm (defined as the remaining arm with the highest probability of being the best in the last adaptive analysis conducted). Only works for trial designs with a common `control` arm.
- `"best"`: selects the best remaining arm (as described under `"control or best"`).
- `"list or best"`: selects the first remaining arm from a specified list (specified using `select_preferences`, technically a character vector). If none of these arms are are active at end-of-trial, the best remaining arm will be selected (as described above).
- `"list"`: as specified above, but if no arms on the provided list remain active at end-of-trial, no arm is selected.

`select_last_arm`

single logical, defaults to `FALSE`. If `TRUE`, the only remaining active arm (the last `control`) will be selected in trials with a common `control` arm ending with `equivalence` or `futility`, before considering the options specified in `select_strategy`. Must be `FALSE` for trial designs without a common `control` arm.

select_preferences
: character vector specifying a number of arms used for selection if one of the "list or best" or "list" options are specified for select_strategy. Can only contain valid arms available in the trial.

te_comp
: character string, treatment-effect comparator. Can be either NULL (the default) in which case the **first** control arm is used for trial designs with a common control arm, or a string naming a single trial arm. Will be used when calculating sq_err_te (the squared error of the treatment effect comparing the selected arm to the comparator arm, as described below).

raw_ests
: single logical. If FALSE (default), the posterior estimates (post_ests or post_ests_all, see setup_trial() and run_trial()) will be used to calculate sq_err (the squared error of the estimated compared to the specified effect in the selected arm) and sq_err_te (the squared error of the treatment effect comparing the selected arm to the comparator arm, as described for te_comp and below). If TRUE, the raw estimates (raw_ests or raw_ests_all, see setup_trial() and run_trial()) will be used instead of the posterior estimates.

final_ests
: single logical. If TRUE (recommended) the final estimates calculated using outcome data from all patients randomised when trials are stopped are used (post_ests_all or raw_ests_all, see setup_trial() and run_trial()); if FALSE, the estimates calculated for each arm when an arm is stopped (or at the last adaptive analysis if not before) using data from patients having reach followed up at this time point and not all patients randomised are used (post_ests or raw_ests, see setup_trial() and run_trial()). If NULL (the default), this argument will be set to FALSE if outcome data are available immediate after randomisation for all patients (for backwards compatibility, as final posterior estimates may vary slightly in this situation, even if using the same data); otherwise it will be said to TRUE. See setup_trial() for more details on how these estimates are calculated.

cores
: NULL or single integer. If NULL, a default value set by setup_cluster() will be used to control whether extractions of simulation results are done in parallel on a default cluster or sequentially in the main process; if a value has not been specified by setup_cluster(), cores will then be set to the value stored in the global "mc.cores" option (if previously set by options(mc.cores = <number of cores>), and 1 if that option has not been specified.
If cores = 1, computations will be run sequentially in the primary process, and if cores > 1, a new parallel cluster will be setup using the parallel library and removed once the function completes. See setup_cluster() for details.

Value

A data.frame containing the following columns:

- sim: the simulation number (from 1 to the total number of simulations).
- final_n: the final sample size in each simulation.
- sum_ys: the sum of the total counts in all arms, e.g., the total number of events in trials with a binary outcome (setup_trial_binom()) or the sum of the arm totals in trials with a continuous outcome (setup_trial_norm()). Always uses all outcome data from all randomised

patients regardless of whether or not all patients had outcome data available at the time of trial stopping (corresponding to `sum_ys_all` in results from `run_trial()`).

- `ratio_ys`: calculated as `sum_ys/final_n` (as described above).
- `final_status`: the final trial status for each simulation, either `"superiority"`, `"equivalence"`, `"futility"`, or `"max"`, as described in `run_trial()`.
- `superior_arm`: the final superior arm in simulations stopped for superiority. Will be `NA` in simulations not stopped for superiority.
- `selected_arm`: the final selected arm (as described above). Will correspond to the `superior_arm` in simulations stopped for superiority and be `NA` if no arm is selected. See `select_strategy` above.
- `sq_err`: the squared error of the estimate in the selected arm, calculated as `(estimated effect - true effect)^2` for the selected arms.
- `sq_err_te`: the squared error of the treatment effect comparing the selected arm to the comparator arm (as specified in `te_comp`). Calculated as:
`((estimated effect in the selected arm - estimated effect in the comparator arm) - (true effect in the selected arm - true effect in the comparator arm))^2`
Will be `NA` for simulations without a selected arm, with no comparator specified (see `te_comp` above), and when the selected arm is the comparator arm.

See Also

`check_performance()`, `summary()`, `plot_convergence()`, `plot_metrics_ecdf()`, `check_remaining_arms()`.

Examples

```
# Setup a trial specification
binom_trial <- setup_trial_binom(arms = c("A", "B", "C", "D"),
                                 control = "A",
                                 true_ys = c(0.20, 0.18, 0.22, 0.24),
                                 data_looks = 1:20 * 100)

# Run 10 simulations with a specified random base seed
res <- run_trials(binom_trial, n_rep = 10, base_seed = 12345)

# Extract results and Select the control arm if available
# in simulations not ending with superiority
extract_results(res, select_strategy = "control")
```

`find_beta_params` *Find beta distribution parameters from thresholds*

Description

Helper function to find a beta distribution with parameters corresponding to the fewest possible patients with events/non-events and a specified event proportion. Used in the **Advanced example** vignette (`vignette("Advanced-example", "adaptr")`) to derive `beta` prior distributions for use in *beta-binomial conjugate models*, based on a belief that the true event probability lies within a specified percentile-based interval (defaults to `95%`). May similarly be used by users to derive other `beta` priors.

Usage

```
find_beta_params(
  theta = NULL,
  boundary_target = NULL,
  boundary = "lower",
  interval_width = 0.95,
  n_dec = 0,
  max_n = 10000
)
```

Arguments

`theta`	single numeric `> 0` and `< 1`, expected true event probability.
`boundary_target`	single numeric `> 0` and `< 1`, target lower or upper boundary of the interval.
`boundary`	single character string, either `"lower"` (default) or `"upper"`, used to select which boundary to use when finding appropriate parameters for the `beta` distribution.
`interval_width`	width of the credible interval whose lower/upper boundary should be used (see `boundary_target`); must be `> 0` and `< 1`; defaults to `0.95`.
`n_dec`	single non-negative integer; the returned parameters are rounded to this number of decimals. Defaults to `0`, in which case the parameters will correspond to whole number of patients.
`max_n`	single integer `> 0` (default `10000`), the maximum total sum of the parameters, corresponding to the maximum total number of patients that will be considered by the function when finding the optimal parameter values. Corresponds to the maximum number of patients contributing information to a beta prior; more than the default number of patients are unlikely to be used in a beta prior.

Value

A single-row `data.frame` with five columns: the two shape parameters of the beta distribution (`alpha`, `beta`), rounded according to `n_dec`, and the actual lower and upper boundaries of the interval and the median (with appropriate names, e.g. `p2.5`, `p50`, and `p97.5` for a `95%` interval), when using those rounded values.

plot_convergence *Plot convergence of performance metrics*

Description

Plots performance metrics according to the number of simulations conducted for multiple simulated trials. The simulated trial results may be split into a number of batches to illustrate stability of performance metrics across different simulations. Calculations are done according to specified selection and restriction strategies as described in `extract_results()` and `check_performance()`. Requires the `ggplot2` package installed.

Usage

```
plot_convergence(
  object,
  metrics = "size mean",
  resolution = 100,
  select_strategy = "control if available",
  select_last_arm = FALSE,
  select_preferences = NULL,
  te_comp = NULL,
  raw_ests = FALSE,
  final_ests = NULL,
  restrict = NULL,
  n_split = 1,
  nrow = NULL,
  ncol = NULL,
  cores = NULL
)
```

Arguments

`object`	`trial_results` object, output from the `run_trials()` function.
`metrics`	the performance metrics to plot, as described in `check_performance()`. Multiple metrics may be plotted at the same time. Valid metrics include: `size_mean`, `size_sd`, `size_median`, `size_p25`, `size_p75`, `size_p0`, `size_p100`, `sum_ys_mean`, `sum_ys_sd`, `sum_ys_median`, `sum_ys_p25`, `sum_ys_p75`, `sum_ys_p0`, `sum_ys_p100`, `ratio_ys_mean`, `ratio_ys_sd`, `ratio_ys_median`, `ratio_ys_p25`, `ratio_ys_p75`, `ratio_ys_p0`, `ratio_ys_p100`, `prob_conclusive`, `prob_superior`, `prob_equivalence`, `prob_futility`, `prob_max`, `prob_select_*` (with `*` being either "`arm_<name>` for all `arm` names or `none`), `rmse`, `rmse_te`, and `idp`. All may be specified as above, case sensitive, but with either spaces or underlines. Defaults to `"size mean"`.
`resolution`	single positive integer, the number of points calculated and plotted, defaults to `100` and must be `>= 10`. Higher numbers lead to smoother plots, but increases computation time. If the value specified is higher than the number of simulations (or simulations per split), the maximum possible value will be used instead.

select_strategy

single character string. If a trial was not stopped due to superiority (or had only 1 arm remaining, if select_last_arm is set to TRUE in trial designs with a common control arm; see below), this parameter specifies which arm will be considered selected when calculating trial design performance metrics, as described below; this corresponds to the consequence of an inconclusive trial, i.e., which arm would then be used in practice.
The following options are available and must be written exactly as below (case sensitive, cannot be abbreviated):

- "control if available" (default): selects the **first** control arm for trials with a common control arm **if** this arm is active at end-of-trial, otherwise no arm will be selected. For trial designs without a common control, no arm will be selected.
- "none": selects no arm in trials not ending with superiority.
- "control": similar to "control if available", but will throw an error if used for trial designs without a common control arm.
- "final control": selects the **final** control arm regardless of whether the trial was stopped for practical equivalence, futility, or at the maximum sample size; this strategy can only be specified for trial designs with a common control arm.
- "control or best": selects the **first** control arm if still active at end-of-trial, otherwise selects the best remaining arm (defined as the remaining arm with the highest probability of being the best in the last adaptive analysis conducted). Only works for trial designs with a common control arm.
- "best": selects the best remaining arm (as described under "control or best").
- "list or best": selects the first remaining arm from a specified list (specified using select_preferences, technically a character vector). If none of these arms are are active at end-of-trial, the best remaining arm will be selected (as described above).
- "list": as specified above, but if no arms on the provided list remain active at end-of-trial, no arm is selected.

select_last_arm

single logical, defaults to FALSE. If TRUE, the only remaining active arm (the last control) will be selected in trials with a common control arm ending with equivalence or futility, before considering the options specified in select_strategy. Must be FALSE for trial designs without a common control arm.

select_preferences

character vector specifying a number of arms used for selection if one of the "list or best" or "list" options are specified for select_strategy. Can only contain valid arms available in the trial.

te_comp

character string, treatment-effect comparator. Can be either NULL (the default) in which case the **first** control arm is used for trial designs with a common control arm, or a string naming a single trial arm. Will be used when calculating sq_err_te (the squared error of the treatment effect comparing the selected arm to the comparator arm, as described below).

raw_ests — single logical. If FALSE (default), the posterior estimates (post_ests or post_ests_all, see setup_trial() and run_trial()) will be used to calculate sq_err (the squared error of the estimated compared to the specified effect in the selected arm) and sq_err_te (the squared error of the treatment effect comparing the selected arm to the comparator arm, as described for te_comp and below). If TRUE, the raw estimates (raw_ests or raw_ests_all, see setup_trial() and run_trial()) will be used instead of the posterior estimates.

final_ests — single logical. If TRUE (recommended) the final estimates calculated using outcome data from all patients randomised when trials are stopped are used (post_ests_all or raw_ests_all, see setup_trial() and run_trial()); if FALSE, the estimates calculated for each arm when an arm is stopped (or at the last adaptive analysis if not before) using data from patients having reach followed up at this time point and not all patients randomised are used (post_ests or raw_ests, see setup_trial() and run_trial()). If NULL (the default), this argument will be set to FALSE if outcome data are available immediate after randomisation for all patients (for backwards compatibility, as final posterior estimates may vary slightly in this situation, even if using the same data); otherwise it will be said to TRUE. See setup_trial() for more details on how these estimates are calculated.

restrict — single character string or NULL. If NULL (default), results are summarised for all simulations; if "superior", results are summarised for simulations ending with superiority only; if "selected", results are summarised for simulations ending with a selected arm only (according to the specified arm selection strategy for simulations not ending with superiority). Some summary measures (e.g., prob_conclusive) have substantially different interpretations if restricted, but are calculated nonetheless.

n_split — single positive integer, the number of consecutive batches the simulation results will be split into, which will be plotted separately. Default is 1 (no splitting); maximum value is the number of simulations summarised (after restrictions) divided by 10.

nrow, ncol — the number of rows and columns when plotting multiple metrics in the same plot (using faceting in ggplot2). Defaults to NULL, in which case this will be determined automatically.

cores — NULL or single integer. If NULL, a default value set by setup_cluster() will be used to control whether extractions of simulation results are done in parallel on a default cluster or sequentially in the main process; if a value has not been specified by setup_cluster(), cores will then be set to the value stored in the global "mc.cores" option (if previously set by options(mc.cores = <number of cores>), and 1 if that option has not been specified.
If cores = 1, computations will be run sequentially in the primary process, and if cores > 1, a new parallel cluster will be setup using the parallel library and removed once the function completes. See setup_cluster() for details.

Value

A ggplot2 plot object.

See Also

check_performance(), summary(), extract_results(), check_remaining_arms().

Examples

```
#### Only run examples if ggplot2 is installed ####
if (requireNamespace("ggplot2", quietly = TRUE)){

  # Setup a trial specification
  binom_trial <- setup_trial_binom(arms = c("A", "B", "C", "D"),
                                   control = "A",
                                   true_ys = c(0.20, 0.18, 0.22, 0.24),
                                   data_looks = 1:20 * 100)

  # Run multiple simulation with a fixed random base seed
  res_mult <- run_trials(binom_trial, n_rep = 25, base_seed = 678)

  # NOTE: the number of simulations in this example is smaller than
  # recommended - the plots reflect that, and show that performance metrics
  # are not stable and have likely not converged yet

  # Convergence plot of mean sample sizes
  plot_convergence(res_mult, metrics = "size mean")

}

if (requireNamespace("ggplot2", quietly = TRUE)){

  # Convergence plot of mean sample sizes and ideal design percentages,
  # with simulations split in 2 batches
  plot_convergence(res_mult, metrics = c("size mean", "idp"), n_split = 2)

}
```

---

plot_history *Plot trial metric history*

Description

Plots the history of relevant metrics over the progress of a single or multiple trial simulations. Simulated trials **only** contribute until the time they are stopped, i.e., if some trials are stopped earlier than others, they will not contribute to the summary statistics at later adaptive looks. Data from individual arms in a trial contribute until the complete trial is stopped.
These history plots require non-sparse results (`sparse` set to `FALSE`; see run_trial() and run_trials()) and the `ggplot2` package installed.

Usage

```
plot_history(object, x_value = "look", y_value = "prob", line = NULL, ...)

## S3 method for class 'trial_result'
plot_history(object, x_value = "look", y_value = "prob", line = NULL, ...)

## S3 method for class 'trial_results'
plot_history(
  object,
  x_value = "look",
  y_value = "prob",
  line = NULL,
  ribbon = list(width = 0.5, alpha = 0.2),
  cores = NULL,
  ...
)
```

Arguments

object	trial_results object, output from the run_trials() function.
x_value	single character string, determining whether the number of adaptive analysis looks ("look", default), the total cumulated number of patients randomised ("total n") or the total cumulated number of patients with outcome data available at each adaptive analysis ("followed n") are plotted on the x-axis.
y_value	single character string, determining which values are plotted on the y-axis. The following options are available: allocation probabilities ("prob", default), the total number of patients with outcome data available ("n") or randomised ("n all") to each arm, the percentage of patients with outcome data available ("pct") or randomised ("pct all") to each arm out of the current total, the sum of all available ("sum ys") outcome data or all outcome data for randomised patients including outcome data not available at the time of the current adaptive analysis ("sum ys all"), the ratio of outcomes as defined for "sum ys"/"sum ys all" divided by the corresponding number of patients in each arm.
line	list styling the lines as per ggplot2 conventions (e.g., linetype, linewidth).
...	additional arguments, not used.
ribbon	list, as line but only appropriate for trial_results objects (i.e., when multiple simulations are run). Also allows to specify the width of the interval: must be between 0 and 1, with 0.5 (default) showing the inter-quartile ranges.
cores	NULL or single integer. If NULL, a default value set by setup_cluster() will be used to control whether extractions of simulation results are done in parallel on a default cluster or sequentially in the main process; if a value has not been specified by setup_cluster(), cores will then be set to the value stored in the global "mc.cores" option (if previously set by options(mc.cores = <number of cores>), and 1 if that option has not been specified. If cores = 1, computations will be run sequentially in the primary process, and if cores > 1, a new parallel cluster will be setup using the parallel library and removed once the function completes. See setup_cluster() for details.

Value

A ggplot2 plot object.

See Also

plot_status().

Examples

```
#### Only run examples if ggplot2 is installed ####
if (requireNamespace("ggplot2", quietly = TRUE)){

  # Setup a trial specification
  binom_trial <- setup_trial_binom(arms = c("A", "B", "C", "D"),
                                   control = "A",
                                   true_ys = c(0.20, 0.18, 0.22, 0.24),
                                   data_looks = 1:20 * 100)



  # Run a single simulation with a fixed random seed
  res <- run_trial(binom_trial, seed = 12345)

  # Plot total allocations to each arm according to overall total allocations
  plot_history(res, x_value = "total n", y_value = "n")

}

if (requireNamespace("ggplot2", quietly = TRUE)){

  # Run multiple simulation with a fixed random base seed
  # Notice that sparse = FALSE is required
  res_mult <- run_trials(binom_trial, n_rep = 15, base_seed = 12345, sparse = FALSE)

  # Plot allocation probabilities at each look
  plot_history(res_mult, x_value = "look", y_value = "prob")

  # Other y_value options are available but not shown in these examples

}
```

---

plot_metrics_ecdf *Plot empirical cumulative distribution functions of performance metrics*

Description

Plots empirical cumulative distribution functions (ECDFs) of numerical performance metrics across multiple simulations from a "trial_results" object returned by run_trials(). Requires the ggplot2 package installed.

Usage

```
plot_metrics_ecdf(
  object,
  metrics = c("size", "sum_ys", "ratio_ys"),
  restrict = NULL,
  nrow = NULL,
  ncol = NULL,
  cores = NULL
)
```

Arguments

object	trial_results object, output from the run_trials() function.
metrics	the performance metrics to plot, as described in extract_results(). Multiple metrics may be plotted at the same time. Valid metrics include: size, sum_ys, and ratio_ys_mean. All may be specified using either spaces or underlines (case sensitive). Defaults to plotting all three.
restrict	single character string or NULL. If NULL (default), results are summarised for all simulations; if "superior", results are summarised for simulations ending with superiority only; if "selected", results are summarised for simulations ending with a selected arm only (according to the specified arm selection strategy for simulations not ending with superiority). Some summary measures (e.g., prob_conclusive) have substantially different interpretations if restricted, but are calculated nonetheless.
nrow, ncol	the number of rows and columns when plotting multiple metrics in the same plot (using faceting in ggplot2). Defaults to NULL, in which case this will be determined automatically.
cores	NULL or single integer. If NULL, a default value set by setup_cluster() will be used to control whether extractions of simulation results are done in parallel on a default cluster or sequentially in the main process; if a value has not been specified by setup_cluster(), cores will then be set to the value stored in the global "mc.cores" option (if previously set by options(mc.cores = <number of cores>), and 1 if that option has not been specified. If cores = 1, computations will be run sequentially in the primary process, and if cores > 1, a new parallel cluster will be setup using the parallel library and removed once the function completes. See setup_cluster() for details.

Value

A ggplot2 plot object.

See Also

check_performance(), summary(), extract_results(), plot_convergence(), check_remaining_arms().

Examples

```
#### Only run examples if ggplot2 is installed ####
if (requireNamespace("ggplot2", quietly = TRUE)){

  # Setup a trial specification
  binom_trial <- setup_trial_binom(arms = c("A", "B", "C", "D"),
                                   control = "A",
                                   true_ys = c(0.20, 0.18, 0.22, 0.24),
                                   data_looks = 1:20 * 100)

  # Run multiple simulation with a fixed random base seed
  res_mult <- run_trials(binom_trial, n_rep = 25, base_seed = 678)

  # NOTE: the number of simulations in this example is smaller than
  # recommended - the plots reflect that, and would likely be smoother if
  # a larger number of trials had been simulated

  # Plot ECDFs of continuous performance metrics
  plot_metrics_ecdf(res_mult)

}
```

`plot_status` *Plot statuses*

Description

Plots the statuses over time of multiple simulated trials (overall or for one or more specific arms). Requires the `ggplot2` package installed.

Usage

```
plot_status(
  object,
  x_value = "look",
  arm = NULL,
  area = list(alpha = 0.5),
  nrow = NULL,
  ncol = NULL
)

## S3 method for class 'trial_results'
plot_status(
  object,
  x_value = "look",
  arm = NULL,
  area = list(alpha = 0.5),
```

```
  nrow = NULL,
  ncol = NULL
)
```

Arguments

object	trial_results object, output from the run_trials() function.
x_value	single character string, determining whether the number of adaptive analysis looks ("look", default), the total cumulated number of patients randomised ("total n") or the total cumulated number of patients with outcome data available at each adaptive analysis ("followed n") are plotted on the x-axis.
arm	character vector containing one or more unique, valid arm names, NA, or NULL (default). If NULL, the overall trial statuses are plotted, otherwise the specified arms or all arms (if NA is specified) are plotted.
area	list of styling settings for the area as per ggplot2 conventions (e.g., alpha, linewidth). The default (list(alpha = 0.5)) sets the transparency to 50% so overlain shaded areas are visible.
nrow, ncol	the number of rows and columns when plotting statuses for multiple arms in the same plot (using faceting in ggplot2). Defaults to NULL, in which case this will be determined automatically where relevant.

Value

A ggplot2 plot object.

See Also

plot_history().

Examples

```
#### Only run examples if ggplot2 is installed ####
if (requireNamespace("ggplot2", quietly = TRUE)){

  # Setup a trial specification
  binom_trial <- setup_trial_binom(arms = c("A", "B", "C", "D"),
                                   control = "A",
                                   true_ys = c(0.20, 0.18, 0.22, 0.24),
                                   data_looks = 1:20 * 100)

  # Run multiple simulation with a fixed random base seed
  res_mult <- run_trials(binom_trial, n_rep = 25, base_seed = 12345)

  # Plot trial statuses at each look according to total allocations
  plot_status(res_mult, x_value = "total n")

}

if (requireNamespace("ggplot2", quietly = TRUE)){
```

```
  # Plot trial statuses for all arms
  plot_status(res_mult, arm = NA)

}
```

---

print *Print methods for adaptive trial objects*

Description

Prints contents of the first input x in a human-friendly way, see **Details** for more information.

Usage

```
## S3 method for class 'trial_spec'
print(x, prob_digits = 3, ...)

## S3 method for class 'trial_result'
print(x, prob_digits = 3, ...)

## S3 method for class 'trial_performance'
print(x, digits = 3, ...)

## S3 method for class 'trial_results'
print(
  x,
  select_strategy = "control if available",
  select_last_arm = FALSE,
  select_preferences = NULL,
  te_comp = NULL,
  raw_ests = FALSE,
  final_ests = NULL,
  restrict = NULL,
  digits = 1,
  cores = NULL,
  ...
)

## S3 method for class 'trial_results_summary'
print(x, digits = 1, ...)

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

Arguments

x	object to print, see **Details**.
prob_digits	single integer (default is 3), the number of digits used when printing probabilities, allocation probabilities and softening powers (with 2 extra digits added for stopping rule probability thresholds in trial specifications and for outcome rates in summarised results from multiple simulations).
...	additional arguments, not used.
digits	single integer, the number of digits used when printing the numeric results. Default is 3 for outputs from `check_performance()` and 1 for outputs from `run_trials()` and the accompanying `summary()` method.
select_strategy	single character string. If a trial was not stopped due to superiority (or had only 1 arm remaining, if `select_last_arm` is set to `TRUE` in trial designs with a common `control` arm; see below), this parameter specifies which arm will be considered selected when calculating trial design performance metrics, as described below; this corresponds to the consequence of an inconclusive trial, i.e., which arm would then be used in practice. The following options are available and must be written exactly as below (case sensitive, cannot be abbreviated): • `"control if available"` (default): selects the **first** `control` arm for trials with a common `control` arm **if** this arm is active at end-of-trial, otherwise no arm will be selected. For trial designs without a common `control`, no arm will be selected. • `"none"`: selects no arm in trials not ending with superiority. • `"control"`: similar to `"control if available"`, but will throw an error if used for trial designs without a common `control` arm. • `"final control"`: selects the **final** `control` arm regardless of whether the trial was stopped for practical equivalence, futility, or at the maximum sample size; this strategy can only be specified for trial designs with a common `control` arm. • `"control or best"`: selects the **first** `control` arm if still active at end-of-trial, otherwise selects the best remaining arm (defined as the remaining arm with the highest probability of being the best in the last adaptive analysis conducted). Only works for trial designs with a common `control` arm. • `"best"`: selects the best remaining arm (as described under `"control or best"`). • `"list or best"`: selects the first remaining arm from a specified list (specified using `select_preferences`, technically a character vector). If none of these arms are are active at end-of-trial, the best remaining arm will be selected (as described above). • `"list"`: as specified above, but if no arms on the provided list remain active at end-of-trial, no arm is selected.
select_last_arm	single logical, defaults to `FALSE`. If `TRUE`, the only remaining active arm (the last `control`) will be selected in trials with a common `control` arm ending

with equivalence or futility, before considering the options specified in select_strategy. Must be FALSE for trial designs without a common control arm.

select_preferences
: character vector specifying a number of arms used for selection if one of the "list or best" or "list" options are specified for select_strategy. Can only contain valid arms available in the trial.

te_comp
: character string, treatment-effect comparator. Can be either NULL (the default) in which case the **first** control arm is used for trial designs with a common control arm, or a string naming a single trial arm. Will be used when calculating sq_err_te (the squared error of the treatment effect comparing the selected arm to the comparator arm, as described below).

raw_ests
: single logical. If FALSE (default), the posterior estimates (post_ests or post_ests_all, see setup_trial() and run_trial()) will be used to calculate sq_err (the squared error of the estimated compared to the specified effect in the selected arm) and sq_err_te (the squared error of the treatment effect comparing the selected arm to the comparator arm, as described for te_comp and below). If TRUE, the raw estimates (raw_ests or raw_ests_all, see setup_trial() and run_trial()) will be used instead of the posterior estimates.

final_ests
: single logical. If TRUE (recommended) the final estimates calculated using outcome data from all patients randomised when trials are stopped are used (post_ests_all or raw_ests_all, see setup_trial() and run_trial()); if FALSE, the estimates calculated for each arm when an arm is stopped (or at the last adaptive analysis if not before) using data from patients having reach followed up at this time point and not all patients randomised are used (post_ests or raw_ests, see setup_trial() and run_trial()). If NULL (the default), this argument will be set to FALSE if outcome data are available immediate after randomisation for all patients (for backwards compatibility, as final posterior estimates may vary slightly in this situation, even if using the same data); otherwise it will be said to TRUE. See setup_trial() for more details on how these estimates are calculated.

restrict
: single character string or NULL. If NULL (default), results are summarised for all simulations; if "superior", results are summarised for simulations ending with superiority only; if "selected", results are summarised for simulations ending with a selected arm only (according to the specified arm selection strategy for simulations not ending with superiority). Some summary measures (e.g., prob_conclusive) have substantially different interpretations if restricted, but are calculated nonetheless.

cores
: NULL or single integer. If NULL, a default value set by setup_cluster() will be used to control whether extractions of simulation results are done in parallel on a default cluster or sequentially in the main process; if a value has not been specified by setup_cluster(), cores will then be set to the value stored in the global "mc.cores" option (if previously set by options(mc.cores = <number of cores>), and 1 if that option has not been specified.
If cores = 1, computations will be run sequentially in the primary process, and if cores > 1, a new parallel cluster will be setup using the parallel library and removed once the function completes. See setup_cluster() for details.

Details

The behaviour depends on the class of `x`:

- `trial_spec`: prints a trial specification setup by `setup_trial()`, `setup_trial_binom()` or `setup_trial_norm()`.
- `trial_result`: prints the results of a single trial simulated by `run_trial()`. More details are saved in the `trial_result` object and thus printed if the `sparse` argument in `run_trial()` or `run_trials()` is set to `FALSE`; if `TRUE`, fewer details are printed, but the omitted details are available by printing the `trial_spec` object created by `setup_trial()`, `setup_trial_binom()` or `setup_trial_norm()`.
- `trial_results`: prints the results of multiple simulations generated using `run_trials()`. Further documentation on how multiple trials are summarised before printing can be found in the `summary()` function documentation.
- `trial_results_summary`: print method for summary of multiple simulations of the same trial specification, generated by using the `summary()` function on an object generated by `run_trials()`.

Value

Invisibly returns `x`.

Methods (by class)

- `print(trial_spec)`: Trial specification
- `print(trial_result)`: Single trial result
- `print(trial_performance)`: Trial performance metrics
- `print(trial_results)`: Multiple trial results
- `print(trial_results_summary)`: Summary of multiple trial results
- `print(trial_calibration)`: Trial calibration

`run_trial` *Simulate a single trial*

Description

This function conducts a single trial simulation using a trial specification as specified by `setup_trial()`, `setup_trial_binom()` or `setup_trial_norm()`.
During simulation, the function randomises "patients", randomly generates outcomes, calculates the probabilities that each `arm` is the best (and better than the control, if any). This is followed by checking inferiority, superiority, equivalence and/or futility as desired; dropping arms, and re-adjusting allocation probabilities according to the criteria specified in the trial specification. If there is no common `control` arm, the trial simulation will be stopped at the final specified adaptive analysis, when 1 arm is superior to the others, or when all arms are considered equivalent (if equivalence is

assessed). If a common control arm is specified, all other arms will be compared to that, and if 1 of these pairwise comparisons crosses the applicable superiority threshold at an adaptive analysis, that arm will become the new control and the old control will be considered inferior and dropped. If multiple non-control arms cross the applicable superiority threshold in the same adaptive analysis, the one with the highest probability of being the overall best will become the new control. Equivalence/futility will also be checked if specified, and equivalent or futile arms will be dropped in designs with a common control arm and the entire trial will be stopped if all remaining arms are equivalent in designs without a common control arm. The trial simulation will be stopped when only 1 arm is left, when the final arms are all equivalent, or after the final specified adaptive analysis.
After stopping (regardless of reason), a final analysis including outcome data from all patients randomised to all arms will be conducted (with the final control arm, if any, used as the control in this analysis). Results from this analysis will be saved, but not used with regards to the adaptive stopping rules. This is particularly relevant if less patients have available outcome data at the last adaptive analyses than the total number of patients randomised (as specified in setup_trial(), setup_trial_binom(), or setup_trial_norm()), as the final analysis will then include all patients randomised, which may be more than in the last adaptive analysis conducted.

Usage

```
run_trial(trial_spec, seed = NULL, sparse = FALSE)
```

Arguments

trial_spec	trial_spec object, generated and validated by the setup_trial(), setup_trial_binom() or setup_trial_norm() function.
seed	single integer or NULL (default). If a value is provided, this value will be used as the random seed when running and the global random seed will be restored after the function has run, so it is not affected.
sparse	single logical; if FALSE (default) everything listed below is included in the returned object. If TRUE, only a limited amount of data are included in the returned object. This can be practical when running many simulations and saving the results using the run_trials() function (which relies on this function), as the output file will thus be substantially smaller. However, printing of individual trial results will be substantially less detailed for sparse results and non-sparse results are required by plot_history().

Value

A trial_result object containing everything listed below if sparse (as described above) is FALSE. Otherwise only final_status, final_n, followed_n, trial_res, seed, and sparse are included.

- final_status: either "superiority", "equivalence", "futility", or "max" (stopped at the last possible adaptive analysis), as calculated during the adaptive analyses.
- final_n: the total number of patients randomised.
- followed_n: the total number of patients with available outcome data at the last adaptive analysis conducted.

- max_n: the pre-specified maximum number of patients with outcome data available at the last possible adaptive analysis.
- max_randomised: the pre-specified maximum number of patients randomised at the last possible adaptive analysis.
- looks: numeric vector, the total number of patients with outcome data available at each conducted adaptive analysis.
- planned_looks: numeric vector, the cumulated number of patients planned to have outcome data available at each adaptive analysis, even those not conducted if the simulation is stopped before the final possible analysis.
- randomised_at_looks: numeric vector, the cumulated number of patients randomised at each conducted adaptive analysis (only including the relevant numbers for the analyses actually conducted).
- start_control: character, initial common control arm (if specified).
- final_control: character, final common control arm (if relevant).
- control_prob_fixed: fixed common control arm probabilities (if specified; see setup_trial()).
- inferiority, superiority, equivalence_prob, equivalence_diff, equivalence_only_first, futility_prob, futility_diff, futility_only_first, highest_is_best, and soften_power: as specified in setup_trial().
- best_arm: the best arm(s), as described in setup_trial().
- trial_res: a data.frame containing most of the information specified for each arm in setup_trial() including true_ys (true outcomes as specified in setup_trial()) and for each arm the sum of the outcomes (sum_ys/sum_ys_all; i.e., the total number of events for binary outcomes or the totals of continuous outcomes) and sum of patients (ns/ns_all), summary statistics for the raw outcome data (raw_ests/raw_ests_all, calculated as specified in setup_trial(), defaults to mean values, i.e., event rates for binary outcomes or means for continuous outcomes) and posterior estimates (post_ests/post_ests_all, post_errs/post_errs_all, lo_cri/lo_cri_all, and hi_cri/hi_cri_all, calculated as specified in setup_trial()), final_status of each arm ("inferior", "superior", "equivalence", "futile", "active", or "control" (currently active control arm, including if the current control when stopped for equivalence)), status_look (specifying the cumulated number of patients with outcome data available when an adaptive analysis changed the final_status to "superior", "inferior", "equivalence", or "futile"), status_probs, the probability (in the last adaptive analysis for each arm) that each arm was the best/better than the common control arm (if any)/equivalent to the common control arm (if any and stopped for equivalence; NA if the control arm was stopped due to the last remaining other arm(s) being stopped for equivalence)/futile if stopped for futility at the last analysis it was included in, final_alloc, the final allocation probability for each arm the last time patients were randomised to it, including for arms stopped at the maximum sample size, and probs_best_last, the probabilities of each remaining arm being the overall best in the last conducted adaptive analysis (NA for previously dropped arms).
  Note: for the variables in the data.frame where a version including the _all-suffix is included, the versions WITHOUT this suffix are calculated using patients with available outcome data at the time of analysis, while the versions WITH the _all-suffixes are calculated using outcome data for all patients randomised at the time of analysis, even if they have not reached the time of follow-up yet (see setup_trial()).

- all_looks: a list of lists containing one list per conducted trial look (adaptive analysis). These lists contain the variables arms, old_status (status before the analysis of the current round was conducted), new_status (as specified above, status after current analysis has been conducted), sum_ys/sum_ys_all (as described above), ns/ns_all (as described above), old_alloc (the allocation probability used during this look), probs_best (the probabilities of each arm being the best in the current adaptive analysis), new_alloc (the allocation probabilities after updating these in the current adaptive analysis; NA for all arms when the trial is stopped and no further adaptive analyses will be conducted), probs_better_first (if a common control is provided, specifying the probabilities that each arm was better than the control in the first analysis conducted during that look), probs_better (as probs_better_first, but updated if another arm becomes the new control), probs_equivalence_first and probs_equivalence (as for probs_better/probs_better_first, but for equivalence if equivalence is assessed). The last variables are NA if the arm was not active in the applicable adaptive analysis or if they would not be included during the next adaptive analysis.
- allocs: a character vector containing the allocations of all patients in the order of randomization.
- ys: a numeric vector containing the outcomes of all patients in the order of randomization (0 or 1 for binary outcomes).
- seed: the random seed used, if specified.
- description, add_info, cri_width, n_draws, robust: as specified in setup_trial(), setup_trial_binom() or setup_trial_norm().
- sparse: single logical, corresponding to the sparse input.

Examples

```
# Setup a trial specification
binom_trial <- setup_trial_binom(arms = c("A", "B", "C", "D"),
                                 true_ys = c(0.20, 0.18, 0.22, 0.24),
                                 data_looks = 1:20 * 100)

# Run trial with a specified random seed
res <- run_trial(binom_trial, seed = 12345)

# Print results with 3 decimals
print(res, digits = 3)
```

---

run_trials *Simulate multiple trials*

Description

This function conducts multiple simulations using a trial specification as specified by setup_trial(), setup_trial_binom() or setup_trial_norm(). This function essentially manages random seeds and runs multiple simulation using run_trial() - additional details on individual simulations are provided in that function's description. This function allows simulating trials in parallel using multiple cores, automatically saving and re-loading saved objects, and "growing" already saved simulation files (i.e., appending additional simulations to the same file).

Usage

```
run_trials(
  trial_spec,
  n_rep,
  path = NULL,
  overwrite = FALSE,
  grow = FALSE,
  cores = NULL,
  base_seed = NULL,
  sparse = TRUE,
  progress = NULL,
  version = NULL,
  compress = TRUE,
  export = NULL,
  export_envir = parent.frame()
)
```

Arguments

trial_spec	trial_spec object, generated and validated by the setup_trial(), setup_trial_binom() or setup_trial_norm() function.
n_rep	single integer; the number of simulations to run.
path	single character string; if specified (defaults to NULL), files will be written to and loaded from this path using the saveRDS() / readRDS() functions.
overwrite	single logical; defaults to FALSE, in which case previous simulations saved in the same path will be re-loaded (if the same trial specification was used). If TRUE, the previous file is overwritten (even if the the same trial specification was not used). If grow is TRUE, this argument must be set to FALSE.
grow	single logical; defaults to FALSE. If TRUE and a valid path to a valid previous file containing less simulations than n_rep, the additional number of simulations will be run (appropriately re-using the same base_seed, if specified) and appended to the same file.
cores	NULL or single integer. If NULL, a default value/cluster set by setup_cluster() will be used to control whether simulations are run in parallel on a default cluster or sequentially in the main process; if a cluster/value has not been specified by setup_cluster(), cores will then be set to the value stored in the global "mc.cores" option (if previously set by options(mc.cores = <number of cores>), and 1 if that option has not been specified. If the resulting number of cores = 1, computations will be run sequentially in the primary process, and if cores > 1, a new parallel cluster will be setup using the parallel library and removed once the function completes. See setup_cluster() for details.
base_seed	single integer or NULL (default); a random seed used as the basis for simulations. Regardless of whether simulations are run sequentially or in parallel, random number streams will be identical and appropriate (see setup_cluster() for details).

sparse	single logical, as described in run_trial(); defaults to TRUE when running multiple simulations, in which case only the data necessary to summarise all simulations are saved for each simulation. If FALSE, more detailed data for each simulation is saved, allowing more detailed printing of individual trial results and plotting using plot_history() (plot_status() does not require non-sparse results).
progress	single numeric > 0 and <= 1 or NULL. If NULL (default), no progress is printed to the console. Otherwise, progress messages are printed to the control at intervals proportional to the value specified by progress. **Note:** as printing is not possible from within clusters on multiple cores, the function conducts batches of simulations on multiple cores (if specified), with intermittent printing of statuses. Thus, all cores have to finish running their current assigned batches before the other cores may proceed with the next batch. If there are substantial differences in the simulation speeds across cores, using progress may thus increase total run time (especially with small values).
version	passed to saveRDS() when saving simulations, defaults to NULL (as in saveRDS()), which means that the current default version is used. Ignored if simulations are not saved.
compress	passed to saveRDS() when saving simulations, defaults to TRUE (as in saveRDS()), see saveRDS() for other options. Ignored if simulations are not saved.
export	character vector of names of objects to export to each parallel core when running in parallel; passed as the varlist argument to parallel::clusterExport(). Defaults to NULL (no objects exported), ignored if cores == 1. See **Details** below.
export_envir	environment where to look for the objects defined in export when running in parallel and export is not NULL. Defaults to the environment from where the function is called.

Details

Exporting objects when using multiple cores

If setup_trial() is used to define a trial specification with custom functions (in the fun_y_gen, fun_draws, and fun_raw_est arguments of setup_trial()) and run_trials() is run with cores > 1, it is necessary to export additional functions or objects used by these functions and defined by the user outside the function definitions provided. Similarly, functions from external packages loaded using library() or require() must be exported or called prefixed with the namespace, i.e., package::function. The export and export_envir arguments are used to export objects calling the parallel::clusterExport()-function. See also setup_cluster(), which may be used to setup a cluster and export required objects only once per session.

Value

A list of a special class "trial_results", which contains the trial_results (results from all simulations; note that seed will be NULL in the individual simulations), trial_spec (the trial specification), n_rep, base_seed, elapsed_time (the total simulation run time), sparse (as described above) and adaptr_version (the version of the adaptr package used to run the simulations). These results may be extracted, summarised, and plotted using the extract_results(),

check_performance(), summary(), print.trial_results(), plot_convergence(), check_remaining_arms(), plot_status(), and plot_history() functions. See the definitions of these functions for additional details and details on additional arguments used to select arms in simulations not ending in superiority and other summary choices.

Examples

```
# Setup a trial specification
binom_trial <- setup_trial_binom(arms = c("A", "B", "C", "D"),
                                 true_ys = c(0.20, 0.18, 0.22, 0.24),
                                 data_looks = 1:20 * 100)

# Run 10 simulations with a specified random base seed
res <- run_trials(binom_trial, n_rep = 10, base_seed = 12345)

# See ?extract_results, ?check_performance, ?summary and ?print for details
# on extracting resutls, summarising and printing
```

---

setup_cluster *Setup default cluster for use in parallelised adaptr functions*

Description

This function setups (or removes) a default cluster for use in all parallelised functions in adaptr using the parallel package. The function also exports objects that should be available on the cluster and sets the random number generator appropriately. See **Details** for further info on how adaptr handles sequential/parallel computation.

Usage

```
setup_cluster(cores, export = NULL, export_envir = parent.frame())
```

Arguments

cores	can be either unspecified, NULL, or a single integer > 0. If NULL or 1, an existing default cluster is removed (if any), and the default will subsequently be to run functions sequentially in the main process if cores = 1, and according to getOption("mc.cores") if NULL (unless otherwise specified in individual functions calls). The parallel::detectCores() function may be used to see the number of available cores, although this comes with some caveats (as described in the function documentation), including that the number of cores may not always be returned and may not match the number of cores that are available for use. In general, using less cores than available may be preferable if other processes are run on the machine at the same time.
export	character vector of names of objects to export to each parallel core when running in parallel; passed as the varlist argument to parallel::clusterExport(). Defaults to NULL (no objects exported), ignored if cores == 1. See **Details** below.

export_envir environment where to look for the objects defined in export when running in parallel and export is not NULL. Defaults to the environment from where the function is called.

Details

Using sequential or parallel computing in adaptr

All parallelised adaptr functions have a cores argument that defaults to NULL. If a non-NULL integer > 0 is provided to the cores argument in any of those (except setup_cluster()), the package will run calculations sequentially in the main process if cores = 1, and otherwise initiate a new cluster of size cores that will be removed once the function completes, regardless of whether or not a default cluster or the global "mc.cores" option have been specified.

If cores is NULL in any adaptr function (except setup_cluster()), the package will use a default cluster if one exists or run computations sequentially if setup_cluster() has last been called with cores = 1. If setup_cluster() has not been called or last called with cores = NULL, then the package will check if the global "mc.cores" option has been specified (using options(mc.cores = <number of cores>)). If this option has been set with a value > 1, then a new, temporary cluster of that size is setup, used, and removed once the function completes. If this option has not been set or has been set to 1, then computations will be run sequentially in the main process.

Generally, we recommend using the setup_cluster() function as this avoids the overhead of re-initiating new clusters with every call to one of the parallelised adaptr functions. This is especially important when exporting many or large objects to a parallel cluster, as this can then be done only once (with the option to export further objects to the same cluster when calling run_trials()).

Type of clusters used and random number generation

The adaptr package solely uses parallel socket clusters (using parallel::makePSOCKcluster()) and thus does not use forking (as this is not available on all operating systems and may cause crashes in some situations). As such, user-defined objects that should be used by the adaptr functions when run in parallel need to be exported using either setup_cluster() or run_trials(), if not included in the generated trial_spec object.

The adaptr package uses the "L'Ecuyer-CMRG" kind (see RNGkind()) for safe random number generation for all parallelised functions. This is also the case when running adaptr functions sequentially with a seed provided, to ensure that the same results are obtained regardless of whether sequential or parallel computation is used. All functions restore both the random number generator kind and the global random seed after use if called with a seed.

Value

Invisibly returns the default parallel cluster or NULL, as appropriate. This may be used with other functions from the parallel package by advanced users, for example to load certain libraries on the cluster prior to calling run_trials().

Examples

```
# Setup a cluster using 2 cores
setup_cluster(cores = 2)

# Get existing default cluster (printed here as invisibly returned)
```

```
print(setup_cluster())

# Remove existing default cluster
setup_cluster(cores = NULL)

# Specify preference for running computations sequentially
setup_cluster(cores = 1)

# Remove default cluster preference
setup_cluster(cores = NULL)

# Set global option to default to using 2 new clusters each time
# (only used if no default cluster preference is specified)
options(mc.cores = 2)
```

`setup_trial` *Setup a generic trial specification*

Description

Specifies the design of an adaptive trial with any type of outcome and validates all inputs. Use `calibrate_trial()` to calibrate the trial specification to obtain a specific value for a certain performance metric (e.g., the Bayesian type 1 error rate). Use `run_trial()` or `run_trials()` to conduct single/multiple simulations of the specified trial, respectively.
See `setup_trial_binom()` and `setup_trial_norm()` for simplified setup of trial designs for common outcome types. For additional trial specification examples, see the the **Basic examples** vignette (`vignette("Basic-examples", package = "adaptr")`) and the **Advanced example** vignette (`vignette("Advanced-example", package = "adaptr")`).

Usage

```
setup_trial(
  arms,
  true_ys,
  fun_y_gen = NULL,
  fun_draws = NULL,
  start_probs = NULL,
  fixed_probs = NULL,
  min_probs = rep(NA, length(arms)),
  max_probs = rep(NA, length(arms)),
  data_looks = NULL,
  max_n = NULL,
  look_after_every = NULL,
  randomised_at_looks = NULL,
  control = NULL,
  control_prob_fixed = NULL,
  inferiority = 0.01,
```

```
  superiority = 0.99,
  equivalence_prob = NULL,
  equivalence_diff = NULL,
  equivalence_only_first = NULL,
  futility_prob = NULL,
  futility_diff = NULL,
  futility_only_first = NULL,
  highest_is_best = FALSE,
  soften_power = 1,
  fun_raw_est = mean,
  cri_width = 0.95,
  n_draws = 5000,
  robust = TRUE,
  description = NULL,
  add_info = NULL
)
```

Arguments

arms	character vector with unique names for the trial arms.
true_ys	numeric vector specifying true outcomes (e.g., event probabilities, mean values, etc.) for all trial arms.
fun_y_gen	function, generates outcomes. See setup_trial() **Details** for information on how to specify this function. **Note:** this function is called once during setup to validate its output (with the global random seed restored afterwards).
fun_draws	function, generates posterior draws. See setup_trial() **Details** for information on how to specify this function. **Note:** this function is called up to three times during setup to validate its output (with the global random seed restored afterwards).
start_probs	numeric vector, allocation probabilities for each arm at the beginning of the trial. The default (NULL) automatically generates equal randomisation probabilities for each arm.
fixed_probs	numeric vector, fixed allocation probabilities for each arm. Must be either a numeric vector with NA for arms without fixed probabilities and values between 0 and 1 for the other arms or NULL (default), if adaptive randomisation is used for all arms or if one of the special settings ("sqrt-based", "sqrt-based start", "sqrt-based fixed", or "match") is specified for control_prob_fixed (described below).
min_probs	numeric vector, lower threshold for adaptive allocation probabilities; lower probabilities will be rounded up to these values. Must be NA (default for all arms) if no lower threshold is wanted and for arms using fixed allocation probabilities.
max_probs	numeric vector, upper threshold for adaptive allocation probabilities; higher probabilities will be rounded down to these values. Must be NA (default for all arms) if no threshold is wanted and for arms using fixed allocation probabilities.
data_looks	vector of increasing integers, specifies when to conduct adaptive analyses (= the total number of patients with available outcome data at each adaptive analysis).

The last number in the vector represents the final adaptive analysis, i.e., the final analysis where superiority, inferiority, practical equivalence, or futility can be claimed. Instead of specifying `data_looks`, the `max_n` and `look_after_every` arguments can be used in combination (in which case `data_looks` must be `NULL`, the default value).

`max_n` single integer, number of patients with available outcome data at the last possible adaptive analysis (defaults to `NULL`). Must only be specified if `data_looks` is `NULL`. Requires specification of the `look_after_every` argument.

`look_after_every`
single integer, specified together with `max_n`. Adaptive analyses will be conducted after every `look_after_every` patients have available outcome data, and at the total sample size as specified by `max_n` (`max_n` does not need to be a multiple of `look_after_every`). If specified, `data_looks` must be `NULL` (default).

`randomised_at_looks`
vector of increasing integers or `NULL`, specifying the number of patients randomised at the time of each adaptive analysis, with new patients randomised using the current allocation probabilities at said analysis. If `NULL` (the default), the number of patients randomised at each analysis will match the number of patients with available outcome data at said analysis, as specified by `data_looks` or `max_n` and `look_after_every`, i.e., outcome data will be available immediately after randomisation for all patients.
If not `NULL`, the vector must be of the same length as the number of adaptive analyses specified by `data_looks` or `max_n` and `look_after_every`, and all values must be larger than or equal to the number of patients with available outcome data at each analysis.

`control` single character string, name of one of the `arms` or `NULL` (default). If specified, this arm will serve as a common control arm, to which all other arms will be compared and the inferiority/superiority/equivalence thresholds (see below) will be for those comparisons. See `setup_trial()` **Details** for information on behaviour with respect to these comparisons.

`control_prob_fixed`
if a common `control` arm is specified, this can be set `NULL` (the default), in which case the control arm allocation probability will not be fixed if control arms change (the allocation probability for the first control arm may still be fixed using `fixed_probs`). If not `NULL`, a vector of probabilities of either length `1` or `number of arms - 1` can be provided, or one of the special arguments `"sqrt-based"`, `"sqrt-based start"`, `"sqrt-based fixed"` or `"match"`. See `setup_trial()` **Details** for details on how this affects trial behaviour.

`inferiority` single numeric value or vector of numeric values of the same length as the maximum number of possible adaptive analyses, specifying the probability threshold(s) for inferiority (default is `0.01`). All values must be `>= 0` and `<= 1`, and if multiple values are supplied, no values may be lower than the preceding value. If a common `control`is not used, all values must be `< 1 / number of arms`. An arm will be considered inferior and dropped if the probability that it is best (when comparing all arms) or better than the control arm (when a common `control` is used) drops below the inferiority threshold at an adaptive analysis.

`superiority` single numeric value or vector of numeric values of the same length as the maximum number of possible adaptive analyses, specifying the probability threshold(s) for superiority (default is `0.99`). All values must be `>= 0` and `<= 1`, and if multiple values are supplied, no values may be higher than the preceding value. If the probability that an arm is best (when comparing all arms) or better than the control arm (when a common `control` is used) exceeds the superiority threshold at an adaptive analysis, said arm will be declared the winner and the trial will be stopped (if no common `control` is used or if the last comparator is dropped in a design with a common control) *or* become the new control and the trial will continue (if a common control is specified).

`equivalence_prob`
single numeric value, vector of numeric values of the same length as the maximum number of possible adaptive analyses or `NULL` (default, corresponding to no equivalence assessment), specifying the probability threshold(s) for equivalence. If not `NULL`, all values must be `> 0` and `<= 1`, and if multiple values are supplied, no value may be higher than the preceding value. If not `NULL`, arms will be dropped for equivalence if the probability of either *(a)* equivalence compared to a common `control` or *(b)* equivalence between all arms remaining (designs without a common `control`) exceeds the equivalence threshold at an adaptive analysis. Requires specification of `equivalence_diff` and `equivalence_only_first`.

`equivalence_diff`
single numeric value (`> 0`) or `NULL` (default, corresponding to no equivalence assessment). If a numeric value is specified, estimated absolute differences smaller than this threshold will be considered equivalent. For designs with a common `control` arm, the differences between each non-control arm and the `control` arm is used, and for trials without a common `control` arm, the difference between the highest and lowest estimated outcome rates are used and the trial is only stopped for equivalence if all remaining arms are equivalent.

`equivalence_only_first`
single logical in trial specifications where `equivalence_prob` and `equivalence_diff` are specified and a common `control` arm is included, otherwise `NULL` (default). If a common `control` arm is used, this specifies whether equivalence will only be assessed for the first control (if `TRUE`) or also for subsequent `control` arms (if `FALSE`) if one arm is superior to the first control and becomes the new control.

`futility_prob` single numeric value, vector of numeric values of the same length as the maximum number of possible adaptive analyses or `NULL` (default, corresponding to no futility assessment), specifying the probability threshold(s) for futility. All values must be `> 0` and `<= 1`, and if multiple values are supplied, no value may be higher than the preceding value. If not `NULL`, arms will be dropped for futility if the probability for futility compared to the common `control` exceeds the futility threshold at an adaptive analysis. Requires a common `control` arm (otherwise this argument must be `NULL`), specification of `futility_diff`, and `futility_only_first`.

`futility_diff` single numeric value (`> 0`) or `NULL` (default, corresponding to no futility assessment). If a numeric value is specified, estimated differences below this threshold in the *beneficial* direction (as specified in `highest_is_best`) will be considered

futile when assessing futility in designs with a common `control` arm. If only 1 arm remains after dropping arms for futility, the trial will be stopped without declaring the last arm superior.

`futility_only_first`
: single logical in trial specifications designs where `futility_prob` and `futility_diff` are specified, otherwise `NULL` (default and required in designs without a common `control` arm). Specifies whether futility will only be assessed against the first `control` (if `TRUE`) or also for subsequent control arms (if `FALSE`) if one arm is superior to the first control and becomes the new control.

`highest_is_best`
: single logical, specifies whether larger estimates of the outcome are favourable or not; defaults to `FALSE`, corresponding to, e.g., an undesirable binary outcomes (e.g., mortality) or a continuous outcome where lower numbers are preferred (e.g., hospital length of stay).

`soften_power`
: either a single numeric value or a numeric vector of exactly the same length as the maximum number of looks/adaptive analyses. Values must be between `0` and `1` (default); if `< 1`, then re-allocated non-fixed allocation probabilities are all raised to this power (followed by rescaling to sum to `1`) to make adaptive allocation probabilities less extreme, in turn used to redistribute remaining probability while respecting limits when defined by `min_probs` and/or `max_probs`. If `1`, then no *softening* is applied.

`fun_raw_est`
: function that takes a numeric vector and returns a single numeric value, used to calculate a raw summary estimate of the outcomes in each `arm`. Defaults to `mean()`, which is always used in the `setup_trial_binom()` and `setup_trial_norm()` functions.
Note: the function is called one time per arm during setup to validate the output structure.

`cri_width`
: single numeric `>= 0` and `< 1`, the width of the percentile-based credible intervals used when summarising individual trial results. Defaults to `0.95`, corresponding to 95% credible intervals.

`n_draws`
: single integer, the number of draws from the posterior distributions for each arm used when running the trial. Defaults to `5000`; can be reduced for a speed gain (at the potential loss of stability of results if too low) or increased for increased precision (increasing simulation time). Values `< 100` are not allowed and values `< 1000` are not recommended and warned against.

`robust`
: single logical, if `TRUE` (default) the medians and median absolute deviations (scaled to be comparable to the standard deviation for normal distributions; MAD_SDs, see `stats::mad()`) are used to summarise the posterior distributions; if `FALSE`, the means and standard deviations (SDs) are used instead (slightly faster, but may be less appropriate for posteriors skewed on the natural scale).

`description`
: optional single character string describing the trial design, will only be used in print functions if not `NULL` (the default).

`add_info`
: optional single string containing additional information regarding the trial design or specifications, will only be used in print functions if not `NULL` (the default).

Details

How to specify the `fun_y_gen` function

The function must take the following arguments:

- `allocs`: character vector, the trial `arms` that new patients allocated since the last adaptive analysis are randomised to.

The function must return a single numeric vector, corresponding to the outcomes for all patients allocated since the last adaptive analysis, in the same order as `allocs`.
See the **Advanced example** vignette (`vignette("Advanced-example", package = "adaptr")`) for an example with further details.

How to specify the `fun_draws` function

The function must take the following arguments:

- `arms`: character vector, the unique trial `arms`, in the same order as above, but only the **currently active** arms are included when the function is called.
- `allocs`: a vector of allocations for all patients, corresponding to the trial `arms`, including patients allocated to both **currently active AND inactive** `arms` when called.
- `ys`: a vector of outcomes for all patients in the same order as `allocs`, including outcomes for patients allocated to both **currently active AND inactive** `arms` when called.
- `control`: single character, the current `control` arm, will be `NULL` for designs without a common control arm, but required regardless as the argument is supplied by `run_trial()`/`run_trials()`.
- `n_draws`: single integer, the number of posterior draws for each arm.

The function must return a `matrix` (containing numeric values) with `arms` named columns and `n_draws` rows. The `matrix` must have columns **only for currently active arms** (when called). Each row should contain a single posterior draw for each arm on the original outcome scale: if they are estimated as, e.g., the *log(odds)*, these estimates must be transformed to probabilities and similarly for other measures.
Important: the `matrix` cannot contain `NA`s, even if no patients have been randomised to an arm yet. See the provided example for one way to alleviate this.
See the **Advanced examples** vignette (`vignette("Advanced-example", package = "adaptr")`) for an example with further details.

Notes

- Different estimation methods and prior distributions may be used; complex functions will lead to slower simulations compared to simpler methods for obtaining posterior draws, including those specified using the `setup_trial_binom()` and `setup_trial_norm()` functions.
- Technically, using log relative effect measures — e.g. log(odds ratios) or log(risk ratios) - or differences compared to a reference arm (e.g., mean differences or absolute risk differences) instead of absolute values in each arm will work to some extent (**be cautious!**):
- Stopping for superiority/inferiority/max sample sizes will work.
- Stopping for equivalence/futility may be used with relative effect measures on the log scale, but thresholds have to be adjusted accordingly.
- Several summary statistics from `run_trial()` (`sum_ys` and posterior estimates) may be nonsensical if relative effect measures are used (depending on calculation method; see the `raw_ests` argument in the relevant functions).

- In the same vein, `extract_results()` (`sum_ys`, `sq_err`, and `sq_err_te`), and `summary()` (`sum_ys_mean/sd/median/q25/q75/q0/q100`, `rmse`, and `rmse_te`) may be equally nonsensical when calculated on the relative scale (see the `raw_ests` argument in the relevant functions.

Using additional custom or functions from loaded packages in the custom functions

If the `fun_y_gen`, `fun_draws`, or `fun_raw_est` functions calls other user-specified functions (or uses objects defined by the user outside these functions or the `setup_trial()`-call) or functions from external packages and simulations are conducted on multiple cores, these objects or functions must be exported or prefixed with their namespaces, respectively, as described in `setup_cluster()` and `run_trials()`.

More information on arguments

- `control`: if one or more treatment arms are superior to the control arm (i.e., passes the superiority threshold as defined above), this arm will become the new control (if multiple arms are superior, the one with the highest probability of being the overall best will become the new control), the previous control will be dropped for inferiority, and all remaining arms will be immediately compared to the new control in the same adaptive analysis and dropped if inferior (or possibly equivalent/futile, see below) compared to this new control arm. Only applies in trials with a common `control`.
- `control_prob_fixed`: If the length is 1, then this allocation probability will be used for the `control` group (including if a new arm becomes the control and the original control is dropped). If multiple values are specified the first value will be used when all arms are active, the second when one arm has been dropped, and so forth. If 1 or more values are specified, previously set `fixed_probs`, `min_probs` or `max_probs` for new control arms will be ignored. If all allocation probabilities do not sum to 1 (e.g, due to multiple limits) they will be rescaled to do so.
  Can also be set to one of the special arguments `"sqrt-based"`, `"sqrt-based start"`, `"sqrt-based fixed"` or `"match"` (written exactly as one of those, case sensitive). This requires `start_probs` to be `NULL` and relevant `fixed_probs` to be `NULL` (or `NA` for the control arm).
  If one of the `"sqrt-based"`/`"sqrt-based start"`/`"sqrt-based fixed"` options are used, the function will set *square-root-transformation-based* starting allocation probabilities. These are defined as:
  `square root of number of non-control arms to 1-ratio for other arms`
  scaled to sum to 1, which will generally increase power for comparisons against the common `control`, as discussed in, e.g., *Park et al, 2020* doi:10.1016/j.jclinepi.2020.04.025.
  If `"sqrt-based"`, square-root-transformation-based allocation probabilities will also be used for new controls when arms are dropped. If `"sqrt-based start"`, the control arm will be fixed to this allocation probability at all times (also after arm dropping, with rescaling as necessary, as specified above). If `"sqrt-based fixed"` is chosen, square-root-transformation-based allocation probabilities will be used and all allocation probabilities will be fixed throughout the trial (with rescaling when arms are dropped).
  If `"match"` is specified, the control group allocation probability will always be *matched* to be similar to the highest non-control arm allocation probability.

Superiority and inferiority

In trial designs without a common control arm, superiority and inferiority are assessed by comparing all **currently active** groups. This means that if a "final" analysis of a trial without a common

control and > 2 arms is conducted including all arms (as will often be done in practice) *after* an adaptive trial has stopped, the final probabilities of the best arm being superior may differ slightly. For example, in a trial with three arms and no common control arm, one arm may be dropped early for inferiority defined as < 1% probability of being the overall best arm. The trial may then continue with the two remaining arms, and stopped when one is declared superior to the other defined as > 99% probability of being the overall best arm. If a final analysis is then conducted including all arms, the final probability of the best arm being overall superior will generally be slightly lower as the probability of the first dropped arm being the best will often be > 0%, even if very low and below the inferiority threshold.
This is less relevant trial designs *with* a common control, as pairwise assessments of superiority/inferiority compared to the common control will not be influenced similarly by previously dropped arms (and previously dropped arms may be included in the analyses, even if posterior distributions are not returned for those). Similarly, in actual clinical trials and when randomised_at_looks is specified with numbers higher than the number of patients with available outcome data at each analysis, final probabilities may change somewhat when the all patients are have completed follow-up and are included in a final analysis.

Equivalence

Equivalence is assessed **after** both inferiority and superiority have been assessed (and in case of superiority, it will be assessed against the new control arm in designs with a common control, if specified - see above).

Futility

Futility is assessed **after** inferiority, superiority, **and** equivalence have been assessed (and in case of superiority, it will be assessed against the new control arm in designs with a common control, if specified - see above). Arms will thus be dropped for equivalence before futility.

Varying probability thresholds

Different probability thresholds (for superiority, inferiority, equivalence, and futility) may be specified for different adaptive analyses. This may be used, e.g., to apply more strict probability thresholds at earlier analyses (or make one or more stopping rules not apply at earlier analyses), similar to the use of monitoring boundaries with different thresholds used for interim analyses in conventional, frequentist group sequential trial designs. See the **Basic examples** vignette (vignette("Basic-examples", package = "adaptr")) for an example.

Value

A trial_spec object used to run simulations by run_trial() or run_trials(). The output is essentially a list containing the input values (some combined in a data.frame called trial_arms), but its class signals that these inputs have been validated and inappropriate combinations and settings have been ruled out. Also contains best_arm, holding the arm(s) with the best value(s) in true_ys. Use str() to peruse the actual content of the returned object.

Examples

```
# Setup a custom trial specification with right-skewed, log-normally
# distributed continuous outcomes (higher values are worse)

# Define the function that will generate the outcomes in each arm
# Notice: contents should match arms/true_ys in the setup_trial() call below
```

```
get_ys_lognorm <- function(allocs) {
  y <- numeric(length(allocs))
  # arms (names and order) and values (except for exponentiation) should match
  # those used in setup_trial (below)
  means <- c("Control" = 2.2, "Experimental A" = 2.1, "Experimental B" = 2.3)
  for (arm in names(means)) {
    ii <- which(allocs == arm)
    y[ii] <- rlnorm(length(ii), means[arm], 1.5)
  }
  y
}

# Define the function that will generate posterior draws
# In this example, the function uses no priors (corresponding to improper
# flat priors) and calculates results on the log-scale, before exponentiating
# back to the natural scale, which is required for assessments of
# equivalence, futility and general interpretation
get_draws_lognorm <- function(arms, allocs, ys, control, n_draws) {
  draws <- list()
  logys <- log(ys)
  for (arm in arms){
    ii <- which(allocs == arm)
    n <- length(ii)
    if (n > 1) {
      # Necessary to avoid errors if too few patients randomised to this arm
    draws[[arm]] <- exp(rnorm(n_draws, mean = mean(logys[ii]), sd = sd(logys[ii])/sqrt(n - 1)))
    } else {
      # Too few patients randomised to this arm - extreme uncertainty
    draws[[arm]] <- exp(rnorm(n_draws, mean = mean(logys), sd = 1000 * (max(logys) - min(logys))))
    }
  }
  do.call(cbind, draws)
}

# The actual trial specification is then defined
lognorm_trial <- setup_trial(
  # arms should match those above
  arms = c("Control", "Experimental A", "Experimental B"),
  # true_ys should match those above
  true_ys = exp(c(2.2, 2.1, 2.3)),
  fun_y_gen = get_ys_lognorm, # as specified above
  fun_draws = get_draws_lognorm, # as specified above
  max_n = 5000,
  look_after_every = 200,
  control = "Control",
  # Square-root-based, fixed control group allocation ratio
  # and response-adaptive randomisation for other arms
  control_prob_fixed = "sqrt-based",
  # Equivalence assessment
  equivalence_prob = 0.9,
  equivalence_diff = 0.5,
  equivalence_only_first = TRUE,
  highest_is_best = FALSE,
```

```
  # Summarise raw results by taking the mean on the
  # log scale and back-transforming
  fun_raw_est = function(x) exp(mean(log(x))) ,
  # Summarise posteriors using medians with MAD-SDs,
  # as distributions will not be normal on the actual scale
  robust = TRUE,
  # Description/additional info used when printing
  description = "continuous, log-normally distributed outcome",
  add_info = "SD on the log scale for all arms: 1.5"
)

# Print trial specification with 3 digits for all probabilities
print(lognorm_trial, prob_digits = 3)
```

`setup_trial_binom` *Setup a trial specification using a binary, binomially distributed outcome*

Description

Specifies the design of an adaptive trial with a binary, binomially distributed outcome and validates all inputs. Uses *beta-binomial* conjugate models with `beta(1, 1)` prior distributions, corresponding to a uniform prior (or the addition of 2 patients, 1 with an event and 1 without, in each `arm`) to the trial. Use `calibrate_trial()` to calibrate the trial specification to obtain a specific value for a certain performance metric (e.g., the Bayesian type 1 error rate). Use `run_trial()` or `run_trials()` to conduct single/multiple simulations of the specified trial, respectively.
Note: `add_info` as specified in `setup_trial()` is set to `NULL` for trial specifications setup by this function.
Further details: please see `setup_trial()`. See `setup_trial_norm()` for simplified setup of trials with a normally distributed continuous outcome.
For additional trial specification examples, see the the **Basic examples** vignette (`vignette("Basic-examples", package = "adaptr")`) and the **Advanced example** vignette (`vignette("Advanced-example", package = "adaptr")`).

Usage

```
setup_trial_binom(
  arms,
  true_ys,
  start_probs = NULL,
  fixed_probs = NULL,
  min_probs = rep(NA, length(arms)),
  max_probs = rep(NA, length(arms)),
  data_looks = NULL,
  max_n = NULL,
  look_after_every = NULL,
  randomised_at_looks = NULL,
```

```
  control = NULL,
  control_prob_fixed = NULL,
  inferiority = 0.01,
  superiority = 0.99,
  equivalence_prob = NULL,
  equivalence_diff = NULL,
  equivalence_only_first = NULL,
  futility_prob = NULL,
  futility_diff = NULL,
  futility_only_first = NULL,
  highest_is_best = FALSE,
  soften_power = 1,
  cri_width = 0.95,
  n_draws = 5000,
  robust = TRUE,
  description = "generic binomially distributed outcome trial"
)
```

Arguments

arms	character vector with unique names for the trial arms.
true_ys	numeric vector, true probabilities (between 0 and 1) of outcomes in all trial arms.
start_probs	numeric vector, allocation probabilities for each arm at the beginning of the trial. The default (NULL) automatically generates equal randomisation probabilities for each arm.
fixed_probs	numeric vector, fixed allocation probabilities for each arm. Must be either a numeric vector with NA for arms without fixed probabilities and values between 0 and 1 for the other arms or NULL (default), if adaptive randomisation is used for all arms or if one of the special settings ("sqrt-based", "sqrt-based start", "sqrt-based fixed", or "match") is specified for control_prob_fixed (described below).
min_probs	numeric vector, lower threshold for adaptive allocation probabilities; lower probabilities will be rounded up to these values. Must be NA (default for all arms) if no lower threshold is wanted and for arms using fixed allocation probabilities.
max_probs	numeric vector, upper threshold for adaptive allocation probabilities; higher probabilities will be rounded down to these values. Must be NA (default for all arms) if no threshold is wanted and for arms using fixed allocation probabilities.
data_looks	vector of increasing integers, specifies when to conduct adaptive analyses (= the total number of patients with available outcome data at each adaptive analysis). The last number in the vector represents the final adaptive analysis, i.e., the final analysis where superiority, inferiority, practical equivalence, or futility can be claimed. Instead of specifying data_looks, the max_n and look_after_every arguments can be used in combination (in which case data_looks must be NULL, the default value).
max_n	single integer, number of patients with available outcome data at the last possible adaptive analysis (defaults to NULL). Must only be specified if data_looks is NULL. Requires specification of the look_after_every argument.

look_after_every
: single integer, specified together with max_n. Adaptive analyses will be conducted after every look_after_every patients have available outcome data, and at the total sample size as specified by max_n (max_n does not need to be a multiple of look_after_every). If specified, data_looks must be NULL (default).

randomised_at_looks
: vector of increasing integers or NULL, specifying the number of patients randomised at the time of each adaptive analysis, with new patients randomised using the current allocation probabilities at said analysis. If NULL (the default), the number of patients randomised at each analysis will match the number of patients with available outcome data at said analysis, as specified by data_looks or max_n and look_after_every, i.e., outcome data will be available immediately after randomisation for all patients.
If not NULL, the vector must be of the same length as the number of adaptive analyses specified by data_looks or max_n and look_after_every, and all values must be larger than or equal to the number of patients with available outcome data at each analysis.

control
: single character string, name of one of the arms or NULL (default). If specified, this arm will serve as a common control arm, to which all other arms will be compared and the inferiority/superiority/equivalence thresholds (see below) will be for those comparisons. See setup_trial() **Details** for information on behaviour with respect to these comparisons.

control_prob_fixed
: if a common control arm is specified, this can be set NULL (the default), in which case the control arm allocation probability will not be fixed if control arms change (the allocation probability for the first control arm may still be fixed using fixed_probs). If not NULL, a vector of probabilities of either length 1 or number of arms - 1 can be provided, or one of the special arguments "sqrt-based", "sqrt-based start", "sqrt-based fixed" or "match". See setup_trial() **Details** for details on how this affects trial behaviour.

inferiority
: single numeric value or vector of numeric values of the same length as the maximum number of possible adaptive analyses, specifying the probability threshold(s) for inferiority (default is 0.01). All values must be >= 0 and <= 1, and if multiple values are supplied, no values may be lower than the preceding value. If a common controlis not used, all values must be < 1 / number of arms. An arm will be considered inferior and dropped if the probability that it is best (when comparing all arms) or better than the control arm (when a common control is used) drops below the inferiority threshold at an adaptive analysis.

superiority
: single numeric value or vector of numeric values of the same length as the maximum number of possible adaptive analyses, specifying the probability threshold(s) for superiority (default is 0.99). All values must be >= 0 and <= 1, and if multiple values are supplied, no values may be higher than the preceding value. If the probability that an arm is best (when comparing all arms) or better than the control arm (when a common control is used) exceeds the superiority threshold at an adaptive analysis, said arm will be declared the winner and the trial will be stopped (if no common control is used or if the last comparator is dropped in a design with a common control) *or* become the new control and the trial will continue (if a common control is specified).

equivalence_prob
: single numeric value, vector of numeric values of the same length as the maximum number of possible adaptive analyses or NULL (default, corresponding to no equivalence assessment), specifying the probability threshold(s) for equivalence. If not NULL, all values must be > 0 and <= 1, and if multiple values are supplied, no value may be higher than the preceding value. If not NULL, arms will be dropped for equivalence if the probability of either *(a)* equivalence compared to a common control or *(b)* equivalence between all arms remaining (designs without a common control) exceeds the equivalence threshold at an adaptive analysis. Requires specification of equivalence_diff and equivalence_only_first.

equivalence_diff
: single numeric value (> 0) or NULL (default, corresponding to no equivalence assessment). If a numeric value is specified, estimated absolute differences smaller than this threshold will be considered equivalent. For designs with a common control arm, the differences between each non-control arm and the control arm is used, and for trials without a common control arm, the difference between the highest and lowest estimated outcome rates are used and the trial is only stopped for equivalence if all remaining arms are equivalent.

equivalence_only_first
: single logical in trial specifications where equivalence_prob and equivalence_diff are specified and a common control arm is included, otherwise NULL (default). If a common control arm is used, this specifies whether equivalence will only be assessed for the first control (if TRUE) or also for subsequent control arms (if FALSE) if one arm is superior to the first control and becomes the new control.

futility_prob
: single numeric value, vector of numeric values of the same length as the maximum number of possible adaptive analyses or NULL (default, corresponding to no futility assessment), specifying the probability threshold(s) for futility. All values must be > 0 and <= 1, and if multiple values are supplied, no value may be higher than the preceding value. If not NULL, arms will be dropped for futility if the probability for futility compared to the common control exceeds the futility threshold at an adaptive analysis. Requires a common control arm (otherwise this argument must be NULL), specification of futility_diff, and futility_only_first.

futility_diff
: single numeric value (> 0) or NULL (default, corresponding to no futility assessment). If a numeric value is specified, estimated differences below this threshold in the *beneficial* direction (as specified in highest_is_best) will be considered futile when assessing futility in designs with a common control arm. If only 1 arm remains after dropping arms for futility, the trial will be stopped without declaring the last arm superior.

futility_only_first
: single logical in trial specifications designs where futility_prob and futility_diff are specified, otherwise NULL (default and required in designs without a common control arm). Specifies whether futility will only be assessed against the first control (if TRUE) or also for subsequent control arms (if FALSE) if one arm is superior to the first control and becomes the new control.

highest_is_best
: single logical, specifies whether larger estimates of the outcome are favourable

or not; defaults to FALSE, corresponding to, e.g., an undesirable binary outcomes (e.g., mortality) or a continuous outcome where lower numbers are preferred (e.g., hospital length of stay).

soften_power — either a single numeric value or a numeric vector of exactly the same length as the maximum number of looks/adaptive analyses. Values must be between 0 and 1 (default); if < 1, then re-allocated non-fixed allocation probabilities are all raised to this power (followed by rescaling to sum to 1) to make adaptive allocation probabilities less extreme, in turn used to redistribute remaining probability while respecting limits when defined by min_probs and/or max_probs. If 1, then no *softening* is applied.

cri_width — single numeric >= 0 and < 1, the width of the percentile-based credible intervals used when summarising individual trial results. Defaults to 0.95, corresponding to 95% credible intervals.

n_draws — single integer, the number of draws from the posterior distributions for each arm used when running the trial. Defaults to 5000; can be reduced for a speed gain (at the potential loss of stability of results if too low) or increased for increased precision (increasing simulation time). Values < 100 are not allowed and values < 1000 are not recommended and warned against.

robust — single logical, if TRUE (default) the medians and median absolute deviations (scaled to be comparable to the standard deviation for normal distributions; MAD_SDs, see stats::mad()) are used to summarise the posterior distributions; if FALSE, the means and standard deviations (SDs) are used instead (slightly faster, but may be less appropriate for posteriors skewed on the natural scale).

description — character string, default is "generic binomially distributed outcome trial". See arguments of setup_trial().

Value

A trial_spec object used to run simulations by run_trial() or run_trials(). The output is essentially a list containing the input values (some combined in a data.frame called trial_arms), but its class signals that these inputs have been validated and inappropriate combinations and settings have been ruled out. Also contains best_arm, holding the arm(s) with the best value(s) in true_ys. Use str() to peruse the actual content of the returned object.

Examples

```
# Setup a trial specification using a binary, binomially
# distributed, undesirable outcome
binom_trial <- setup_trial_binom(
  arms = c("Arm A", "Arm B", "Arm C"),
  true_ys = c(0.25, 0.20, 0.30),
  # Minimum allocation of 15% in all arms
  min_probs = rep(0.15, 3),
  data_looks = seq(from = 300, to = 2000, by = 100),
  # Stop for equivalence if > 90% probability of
  # absolute differences < 5 percentage points
  equivalence_prob = 0.9,
  equivalence_diff = 0.05,
  soften_power = 0.5 # Limit extreme allocation ratios
```

```
)

# Print using 3 digits for probabilities
print(binom_trial, prob_digits = 3)
```

`setup_trial_norm` *Setup a trial specification using a continuous, normally distributed outcome*

Description

Specifies the design of an adaptive trial with a continuous, normally distributed outcome and validates all inputs. Uses normally distributed posterior distributions for the mean values in each trial arm; technically, no priors are used (as using *normal-normal* conjugate prior models with extremely wide or uniform priors gives similar results for these simple, unadjusted estimates). This corresponds to the use of improper, flat priors, although not explicitly specified as such. Use `calibrate_trial()` to calibrate the trial specification to obtain a specific value for a certain performance metric (e.g., the Bayesian type 1 error rate). Use `run_trial()` or `run_trials()` to conduct single/multiple simulations of the specified trial, respectively.
Note: `add_info` as specified in `setup_trial()` is set to the arms and standard deviations used for trials specified using this function.
Further details: please see `setup_trial()`. See `setup_trial_binom()` for simplified setup of trials with binomially distributed binary outcomes.
For additional trial specification examples, see the the **Basic examples** vignette (`vignette("Basic-examples", package = "adaptr")`) and the **Advanced example** vignette (`vignette("Advanced-example", package = "adaptr")`).

Usage

```
setup_trial_norm(
  arms,
  true_ys,
  sds,
  start_probs = NULL,
  fixed_probs = NULL,
  min_probs = rep(NA, length(arms)),
  max_probs = rep(NA, length(arms)),
  data_looks = NULL,
  max_n = NULL,
  look_after_every = NULL,
  randomised_at_looks = NULL,
  control = NULL,
  control_prob_fixed = NULL,
  inferiority = 0.01,
  superiority = 0.99,
  equivalence_prob = NULL,
```

```
  equivalence_diff = NULL,
  equivalence_only_first = NULL,
  futility_prob = NULL,
  futility_diff = NULL,
  futility_only_first = NULL,
  highest_is_best = FALSE,
  soften_power = 1,
  cri_width = 0.95,
  n_draws = 5000,
  robust = FALSE,
  description = "generic normally distributed outcome trial"
)
```

Arguments

arms	character vector with unique names for the trial arms.
true_ys	numeric vector, simulated means of the outcome in all trial arms.
sds	numeric vector, true standard deviations (must be > 0) of the outcome in all trial arms.
start_probs	numeric vector, allocation probabilities for each arm at the beginning of the trial. The default (NULL) automatically generates equal randomisation probabilities for each arm.
fixed_probs	numeric vector, fixed allocation probabilities for each arm. Must be either a numeric vector with NA for arms without fixed probabilities and values between 0 and 1 for the other arms or NULL (default), if adaptive randomisation is used for all arms or if one of the special settings ("sqrt-based", "sqrt-based start", "sqrt-based fixed", or "match") is specified for control_prob_fixed (described below).
min_probs	numeric vector, lower threshold for adaptive allocation probabilities; lower probabilities will be rounded up to these values. Must be NA (default for all arms) if no lower threshold is wanted and for arms using fixed allocation probabilities.
max_probs	numeric vector, upper threshold for adaptive allocation probabilities; higher probabilities will be rounded down to these values. Must be NA (default for all arms) if no threshold is wanted and for arms using fixed allocation probabilities.
data_looks	vector of increasing integers, specifies when to conduct adaptive analyses (= the total number of patients with available outcome data at each adaptive analysis). The last number in the vector represents the final adaptive analysis, i.e., the final analysis where superiority, inferiority, practical equivalence, or futility can be claimed. Instead of specifying data_looks, the max_n and look_after_every arguments can be used in combination (in which case data_looks must be NULL, the default value).
max_n	single integer, number of patients with available outcome data at the last possible adaptive analysis (defaults to NULL). Must only be specified if data_looks is NULL. Requires specification of the look_after_every argument.
look_after_every	single integer, specified together with max_n. Adaptive analyses will be conducted after every look_after_every patients have available outcome data, and

at the total sample size as specified by max_n (max_n does not need to be a multiple of look_after_every). If specified, data_looks must be NULL (default).

randomised_at_looks
: vector of increasing integers or NULL, specifying the number of patients randomised at the time of each adaptive analysis, with new patients randomised using the current allocation probabilities at said analysis. If NULL (the default), the number of patients randomised at each analysis will match the number of patients with available outcome data at said analysis, as specified by data_looks or max_n and look_after_every, i.e., outcome data will be available immediately after randomisation for all patients.
If not NULL, the vector must be of the same length as the number of adaptive analyses specified by data_looks or max_n and look_after_every, and all values must be larger than or equal to the number of patients with available outcome data at each analysis.

control
: single character string, name of one of the arms or NULL (default). If specified, this arm will serve as a common control arm, to which all other arms will be compared and the inferiority/superiority/equivalence thresholds (see below) will be for those comparisons. See setup_trial() **Details** for information on behaviour with respect to these comparisons.

control_prob_fixed
: if a common control arm is specified, this can be set NULL (the default), in which case the control arm allocation probability will not be fixed if control arms change (the allocation probability for the first control arm may still be fixed using fixed_probs). If not NULL, a vector of probabilities of either length 1 or number of arms - 1 can be provided, or one of the special arguments "sqrt-based", "sqrt-based start", "sqrt-based fixed" or "match". See setup_trial() **Details** for details on how this affects trial behaviour.

inferiority
: single numeric value or vector of numeric values of the same length as the maximum number of possible adaptive analyses, specifying the probability threshold(s) for inferiority (default is 0.01). All values must be >= 0 and <= 1, and if multiple values are supplied, no values may be lower than the preceding value. If a common controlis not used, all values must be < 1 / number of arms. An arm will be considered inferior and dropped if the probability that it is best (when comparing all arms) or better than the control arm (when a common control is used) drops below the inferiority threshold at an adaptive analysis.

superiority
: single numeric value or vector of numeric values of the same length as the maximum number of possible adaptive analyses, specifying the probability threshold(s) for superiority (default is 0.99). All values must be >= 0 and <= 1, and if multiple values are supplied, no values may be higher than the preceding value. If the probability that an arm is best (when comparing all arms) or better than the control arm (when a common control is used) exceeds the superiority threshold at an adaptive analysis, said arm will be declared the winner and the trial will be stopped (if no common control is used or if the last comparator is dropped in a design with a common control) *or* become the new control and the trial will continue (if a common control is specified).

equivalence_prob
: single numeric value, vector of numeric values of the same length as the maximum number of possible adaptive analyses or NULL (default, corresponding to

no equivalence assessment), specifying the probability threshold(s) for equivalence. If not NULL, all values must be > 0 and <= 1, and if multiple values are supplied, no value may be higher than the preceding value. If not NULL, arms will be dropped for equivalence if the probability of either *(a)* equivalence compared to a common control or *(b)* equivalence between all arms remaining (designs without a common control) exceeds the equivalence threshold at an adaptive analysis. Requires specification of equivalence_diff and equivalence_only_first.

equivalence_diff

single numeric value (> 0) or NULL (default, corresponding to no equivalence assessment). If a numeric value is specified, estimated absolute differences smaller than this threshold will be considered equivalent. For designs with a common control arm, the differences between each non-control arm and the control arm is used, and for trials without a common control arm, the difference between the highest and lowest estimated outcome rates are used and the trial is only stopped for equivalence if all remaining arms are equivalent.

equivalence_only_first

single logical in trial specifications where equivalence_prob and equivalence_diff are specified and a common control arm is included, otherwise NULL (default). If a common control arm is used, this specifies whether equivalence will only be assessed for the first control (if TRUE) or also for subsequent control arms (if FALSE) if one arm is superior to the first control and becomes the new control.

futility_prob single numeric value, vector of numeric values of the same length as the maximum number of possible adaptive analyses or NULL (default, corresponding to no futility assessment), specifying the probability threshold(s) for futility. All values must be > 0 and <= 1, and if multiple values are supplied, no value may be higher than the preceding value. If not NULL, arms will be dropped for futility if the probability for futility compared to the common control exceeds the futility threshold at an adaptive analysis. Requires a common control arm (otherwise this argument must be NULL), specification of futility_diff, and futility_only_first.

futility_diff single numeric value (> 0) or NULL (default, corresponding to no futility assessment). If a numeric value is specified, estimated differences below this threshold in the *beneficial* direction (as specified in highest_is_best) will be considered futile when assessing futility in designs with a common control arm. If only 1 arm remains after dropping arms for futility, the trial will be stopped without declaring the last arm superior.

futility_only_first

single logical in trial specifications designs where futility_prob and futility_diff are specified, otherwise NULL (default and required in designs without a common control arm). Specifies whether futility will only be assessed against the first control (if TRUE) or also for subsequent control arms (if FALSE) if one arm is superior to the first control and becomes the new control.

highest_is_best

single logical, specifies whether larger estimates of the outcome are favourable or not; defaults to FALSE, corresponding to, e.g., an undesirable binary outcomes (e.g., mortality) or a continuous outcome where lower numbers are preferred (e.g., hospital length of stay).

soften_power — either a single numeric value or a numeric vector of exactly the same length as the maximum number of looks/adaptive analyses. Values must be between 0 and 1 (default); if < 1, then re-allocated non-fixed allocation probabilities are all raised to this power (followed by rescaling to sum to 1) to make adaptive allocation probabilities less extreme, in turn used to redistribute remaining probability while respecting limits when defined by min_probs and/or max_probs. If 1, then no *softening* is applied.

cri_width — single numeric >= 0 and < 1, the width of the percentile-based credible intervals used when summarising individual trial results. Defaults to 0.95, corresponding to 95% credible intervals.

n_draws — single integer, the number of draws from the posterior distributions for each arm used when running the trial. Defaults to 5000; can be reduced for a speed gain (at the potential loss of stability of results if too low) or increased for increased precision (increasing simulation time). Values < 100 are not allowed and values < 1000 are not recommended and warned against.

robust — single logical, if TRUE (default) the medians and median absolute deviations (scaled to be comparable to the standard deviation for normal distributions; MAD_SDs, see stats::mad()) are used to summarise the posterior distributions; if FALSE, the means and standard deviations (SDs) are used instead (slightly faster, but may be less appropriate for posteriors skewed on the natural scale).

description — character string, default is "generic normally distributed outcome trial". See arguments of setup_trial().

Details

Because the posteriors used in this type of trial (with a generic, continuous, normally distributed outcome) are by definition normally distributed, FALSE is used as the default value for the robust argument.

Value

A trial_spec object used to run simulations by run_trial() or run_trials(). The output is essentially a list containing the input values (some combined in a data.frame called trial_arms), but its class signals that these inputs have been validated and inappropriate combinations and settings have been ruled out. Also contains best_arm, holding the arm(s) with the best value(s) in true_ys. Use str() to peruse the actual content of the returned object.

Examples

```
# Setup a trial specification using a continuous, normally distributed, desirable outcome
norm_trial <- setup_trial_norm(
  arms = c("Control", "New A", "New B", "New C"),
  true_ys = c(15, 20, 14, 13),
  sds = c(2, 2.5, 1.9, 1.8), # SDs in each arm
  max_n = 500,
  look_after_every = 50,
  control = "Control", # Common control arm
  # Square-root-based, fixed control group allocation ratios
  control_prob_fixed = "sqrt-based fixed",
```

```
  # Desirable outcome
  highest_is_best = TRUE,
  soften_power = 0.5 # Limit extreme allocation ratios
)

# Print using 3 digits for probabilities
print(norm_trial, prob_digits = 3)
```

---

summary *Summary of simulated trial results*

Description

Summarises simulation results from the run_trials() function. Uses extract_results() and check_performance(), which may be used directly to extract key trial results without summarising or to calculate performance metrics (with uncertainty measures if desired) and return them in a tidy data.frame.

Usage

```
## S3 method for class 'trial_results'
summary(
  object,
  select_strategy = "control if available",
  select_last_arm = FALSE,
  select_preferences = NULL,
  te_comp = NULL,
  raw_ests = FALSE,
  final_ests = NULL,
  restrict = NULL,
  cores = NULL,
  ...
)
```

Arguments

object	trial_results object, output from the run_trials() function.
select_strategy	single character string. If a trial was not stopped due to superiority (or had only 1 arm remaining, if select_last_arm is set to TRUE in trial designs with a common control arm; see below), this parameter specifies which arm will be considered selected when calculating trial design performance metrics, as described below; this corresponds to the consequence of an inconclusive trial, i.e., which arm would then be used in practice. The following options are available and must be written exactly as below (case sensitive, cannot be abbreviated):

- "control if available" (default): selects the **first** control arm for trials with a common control arm **if** this arm is active at end-of-trial, otherwise no arm will be selected. For trial designs without a common control, no arm will be selected.
- "none": selects no arm in trials not ending with superiority.
- "control": similar to "control if available", but will throw an error if used for trial designs without a common control arm.
- "final control": selects the **final** control arm regardless of whether the trial was stopped for practical equivalence, futility, or at the maximum sample size; this strategy can only be specified for trial designs with a common control arm.
- "control or best": selects the **first** control arm if still active at end-of-trial, otherwise selects the best remaining arm (defined as the remaining arm with the highest probability of being the best in the last adaptive analysis conducted). Only works for trial designs with a common control arm.
- "best": selects the best remaining arm (as described under "control or best").
- "list or best": selects the first remaining arm from a specified list (specified using select_preferences, technically a character vector). If none of these arms are are active at end-of-trial, the best remaining arm will be selected (as described above).
- "list": as specified above, but if no arms on the provided list remain active at end-of-trial, no arm is selected.

select_last_arm

single logical, defaults to FALSE. If TRUE, the only remaining active arm (the last control) will be selected in trials with a common control arm ending with equivalence or futility, before considering the options specified in select_strategy. Must be FALSE for trial designs without a common control arm.

select_preferences

character vector specifying a number of arms used for selection if one of the "list or best" or "list" options are specified for select_strategy. Can only contain valid arms available in the trial.

te_comp

character string, treatment-effect comparator. Can be either NULL (the default) in which case the **first** control arm is used for trial designs with a common control arm, or a string naming a single trial arm. Will be used when calculating sq_err_te (the squared error of the treatment effect comparing the selected arm to the comparator arm, as described below).

raw_ests

single logical. If FALSE (default), the posterior estimates (post_ests or post_ests_all, see setup_trial() and run_trial()) will be used to calculate sq_err (the squared error of the estimated compared to the specified effect in the selected arm) and sq_err_te (the squared error of the treatment effect comparing the selected arm to the comparator arm, as described for te_comp and below). If TRUE, the raw estimates (raw_ests or raw_ests_all, see setup_trial() and run_trial()) will be used instead of the posterior estimates.

final_ests

single logical. If TRUE (recommended) the final estimates calculated using outcome data from all patients randomised when trials are stopped are used (post_ests_all

or raw_ests_all, see setup_trial() and run_trial()); if FALSE, the estimates calculated for each arm when an arm is stopped (or at the last adaptive analysis if not before) using data from patients having reach followed up at this time point and not all patients randomised are used (post_ests or raw_ests, see setup_trial() and run_trial()). If NULL (the default), this argument will be set to FALSE if outcome data are available immediate after randomisation for all patients (for backwards compatibility, as final posterior estimates may vary slightly in this situation, even if using the same data); otherwise it will be said to TRUE. See setup_trial() for more details on how these estimates are calculated.

restrict — single character string or NULL. If NULL (default), results are summarised for all simulations; if "superior", results are summarised for simulations ending with superiority only; if "selected", results are summarised for simulations ending with a selected arm only (according to the specified arm selection strategy for simulations not ending with superiority). Some summary measures (e.g., prob_conclusive) have substantially different interpretations if restricted, but are calculated nonetheless.

cores — NULL or single integer. If NULL, a default value set by setup_cluster() will be used to control whether extractions of simulation results are done in parallel on a default cluster or sequentially in the main process; if a value has not been specified by setup_cluster(), cores will then be set to the value stored in the global "mc.cores" option (if previously set by options(mc.cores = <number of cores>), and 1 if that option has not been specified.
If cores = 1, computations will be run sequentially in the primary process, and if cores > 1, a new parallel cluster will be setup using the parallel library and removed once the function completes. See setup_cluster() for details.

... — additional arguments, not used.

Value

A "trial_results_summary" object containing the following values:

- n_rep: the number of simulations.
- n_summarised: as described in check_performance().
- highest_is_best: as specified in setup_trial().
- elapsed_time: the total simulation time.
- size_mean, size_sd, size_median, size_p25, size_p75, size_p0, size_p100, sum_ys_mean, sum_ys_sd, sum_ys_median, sum_ys_p25, sum_ys_p75, sum_ys_p0, sum_ys_p100, ratio_ys_mean, ratio_ys_sd, ratio_ys_median, ratio_ys_p25, ratio_ys_p75, ratio_ys_p0, ratio_ys_p100, prob_conclusive, prob_superior, prob_equivalence, prob_futility, prob_max, prob_select_* (with * being either "arm_<name> for all arm names or none), rmse, rmse_te, and idp: performance metrics as described in check_performance(). Note that all sum_ys_ and ratio_ys_ measures use outcome data from all randomised patients, regardless of whether they had outcome data available at the last analysis or not, as described in extract_results().
- select_strategy, select_last_arm, select_preferences, te_comp, raw_ests, final_ests, restrict: as specified above.

- control: the control arm specified by setup_trial(), setup_trial_binom() or setup_trial_norm(); NULL if no control.
- equivalence_assessed, futility_assessed: single logicals, specifies whether the trial design specification includes assessments of equivalence and/or futility.
- base_seed: as specified in run_trials().
- cri_width, n_draws, robust, description, add_info: as specified in setup_trial(), setup_trial_binom() or setup_trial_norm().

See Also

extract_results(), check_performance(), plot_convergence(), plot_metrics_ecdf(), check_remaining_arms().

Examples

```
# Setup a trial specification
binom_trial <- setup_trial_binom(arms = c("A", "B", "C", "D"),
                                 control = "A",
                                 true_ys = c(0.20, 0.18, 0.22, 0.24),
                                 data_looks = 1:20 * 100)

# Run 10 simulations with a specified random base seed
res <- run_trials(binom_trial, n_rep = 10, base_seed = 12345)

# Summarise simulations - select the control arm if available in trials not
# ending with a superiority decision
res_sum <- summary(res, select_strategy = "control")

# Print summary
print(res_sum, digits = 1)
```

---

update_saved_trials *Update previously saved simulation results*

Description

This function updates a previously saved "trial_results" object created and saved by run_trials() using a previous version of adaptr, allowing the results from these previous simulations to be post-processed (including performance metric calculation, printing and plotting) without errors by this version of the package. The function should be run only once per saved simulation object and will issue a warning if the object is already up to date. And overview of the changes made according to the adaptr package version used to generate the original object is provided in **Details**.
NOTE: some values cannot be updated and will be set to NA (the posterior estimates from the 'final' analysis conducted after the last adaptive analysis and including outcome data for all patients), and thus using both raw_ests = TRUE and final_ests = TRUE in the extract_results() and summary() functions will lead to missing values for some of the values calculated for updated simulation objects.
NOTE: other objects created by the adaptr package, i.e., trial specifications generated by setup_trial()

/ setup_trial_binom() / setup_trial_norm() and single simulation results from run_trials() when not included in as part of the returned output from run_trials() should be re-created by re-running the relevant code using the updated version of adaptr; if manually re-loaded from previous sessions, they may cause errors and problems with the updated version of the package.

Usage

```
update_saved_trials(path, version = NULL, compress = TRUE)
```

Arguments

path	single character; the path to the saved "trial_results"-object containing the simulations saved by run_trials().
version	passed to saveRDS() when saving the updated object, defaults to NULL (as in saveRDS()), which means that the current default version is used.
compress	passed to saveRDS() when saving the updated object, defaults to TRUE (as in saveRDS()), see saveRDS() for other options.

Details

The following changes are made according to the version of adaptr used to generate the original "trial_results" object:

- v1.2.0+: only updates the version number.
- v1.1.1 or earlier: updates version number and everything related to follow-up and data collection lag (in these versions, the randomised_at_looks argument in the setup_trial() functions did not exist, but for practical purposes was identical to the number of patients with available data at each look).

Value

Invisibly returns the updated "trial_results"-object.

See Also

run_trials().

Index