Package 'adsasi'

February 1, 2026

Title Adaptive Sample Size Simulator

Version 0.9.0.1

Description A simulations-first sample size determination package that aims at making sample size formulae obsolete for most easily computable statistical experiments ; the main envisioned use case is clinical trials. The proposed clinical trial must be written by the user in the form of a function that takes as argument a sample size and returns a boolean (for whether or not the trial is a success). The 'adsasi' functions will then use it to find the correct sample size empirically. The unavoidable misspecification is obviated by trying sample size values close to the right value, the latter being understood as the value that gives the probability of success the user wants (usually 80 or 90% in biostatistics, corresponding to 20 or 10% type II error).

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Depends abind, grDevices, graphics, stats

NeedsCompilation no

Author Skerdi Haviari [aut, cre]

Maintainer Skerdi Haviari <skerdi.haviari@aphp.fr>

Repository CRAN

Date/Publication 2026-02-01 07:50:11 UTC

Contents

adsasi_0d *Adaptive Sample Size Finder For Fixed Designs*

Description

This function empirically finds the relationship between sample size and power, for a given experiental simulation scenario supplied by the user in the form of a function (most typically a clinical trial, but any experiment whose success rate increases with the number of observations can be processed). adsasi_0d will try different sample sizes and progressively zoom on the ones where power is nominal. Power is understood in a broad sense here, as a probability of success of the experiment rather than a strict statistical power.

Usage

```
adsasi_0d(
  simfun,
  tar_power = 0.9,
  ...,
  nsims = 5000,
  verbose = FALSE,
  impNN = Inf,
  capNN = 2000,
  initiation = TRUE,
  savegraphs = FALSE,
  keepsims = FALSE
)
```

Arguments

simfun	(function) The user-supplied function that describes the clinical trial scenario (or similar experiment) that needs to be explored. Must have as named arguments a sample size (named NN) and an arbitrary number of design parameters. Must return a boolean indicating whether the trial is successful or not, after performing any required computations (regressions, bootstraps) as written by the user, and never return NA.
tar_power	(single number between 0 and 1) Target power (or more broadly, probability of success). adsasi_0d will seek regions where simfun returns TRUE with a frequency of tar_power, assuming that higher sample size equals higher probability of success.
...	Additional named arguments to be passed to simfun. Some of these arguments can be functions themselves (e.g. for trying different analysis models). Any simfun argument without a default value must be specified here.
nsims	(single number) Number of simulations to be run. After initialization, simulations are run in batches of 10% of the number of existing simulations, until nsims is reached.
verbose	(boolean) Whether to print extra diagnostics messages throughout the run.

`impNN` (single number, or infinity) Sample size that is considered impossible (either computationnally, or logistically). The simulator will exit if, after 500+ simulations, it looks like the best value is above this. In practice, is mostly useful to avoid expensive computations in situations where `simfun` is not written well or is prohibitively long to compute for large sample sizes.

`capNN` (single number, or infinity) Maximum sample size that will be simulated. Also mostly useful to avoid expensive computations. Values between `capNN` and `impNN` will be extrapolations of unclear validity, so if it looks like the answer is really above `capNN`, try running the wrapper again with a higher `capNN`.

`initiation` (boolean, or numeric matrix) Either a boolean indicating whether or not to keep the first 150 simulations for the relationship inference (those tend to be far from `tar_power`), or a matrix with simulation results from a previous run which the user wants enrich with more simulations (formatted exactly as produced by `adsasi_0d` with the same `simfun`). See `keepsims` and Note below for how to store and retrieve this data.

`savegraphs` (boolean or string) Whether to save graphs on drive (vs. showing them in the console). If string, is interpreted as a typical name to be used (several graphs will be drawn, with iteration number, timestamp and .png file extension appended). The string can contain a filepath, but folders must already exist (e.g. with `dir.create()` from `base`, if automated).

`keepsims` (boolean) Whether to keep the simulations sizes and individual outcomes in the output. See Note for format details.

Value

(list) A list with one (by default) element named `size_estimate` with the sample size to obtain probability of success equal to `tar_power`. If `keepsims=TRUE`, additional elements with fits and simulation results (see Note).

Note

Standard error of estimated sample size is shown on graphs but not saved by default anywhere.

With `keepsims=TRUE`, additional elements will be returned : `trials` for all the used simulations, `confint` for 95% confidence interval of `size_estimate`. The confidence interval is obtained in square root form and is assymetric ; the reverse calculation needs to be done to extract the standard error of its root (take the roots of the bounds & divide their distance by 2*1.96).

The `trials` element can be fed back into `adsasi_0d` using argument `initiation=trials` to resume with the same simulations as before. Note that the second `adsasi_0d` call will need to use the same `simfun` and its arguments for this to make sense.

Examples

```
# First, the user defines a function for their target situation. In this simple example, a 2-sample
# t-test with unequal allocation. Note the syntax to avoid returning NAs.
simulate_unequal_t_test = function(NN=20,ratio_n1_NN=0.5,delta=1)
 {
  n1 = round(ratio_n1_NN*NN) ; n2 = NN-n1
  yy1 = rnorm(n1) ; yy2 = rnorm(n2,delta)
```

```
  pp=NA ; try(pp <- t.test(yy1,yy2)$p.value,silent=TRUE)
  !is.na(pp) & pp<0.05
  }
simulate_unequal_t_test()
# Now we empirically find the relationship between sample size and the parameter of interest.
# Note that we can change the simfun parameters directly from the adsasi_0d call.
# nsims should generally be much higher than in this fast-running example (>5000).
adsasi_0d(simulate_unequal_t_test,delta=1.25,nsims=200)
```

---

adsasi_1d *Adaptive Sample Size Finder With One Floating Parameter*

Description

This function empirically finds the relationship between sample size and a numeric parameter of interest, for a given experiental simulation scenario supplied by the user in the form of a function (most typically a clinical trial, but any experiment whose success rate increases with the number of observations can be processed). adsasi_1d will search the two-dimensional space empirically (sample size x parameter of interest), favoring exploration of low sample size regions, to find the line where power is nominal. Power is understood in a broad sense here, as a probability of success of the experiment rather than a strict statistical power.

Usage

```
adsasi_1d(
  simfun,
  tar_power = 0.9,
  ...,
  optivar,
  optiwin = c(min = 0, max = 1),
  optilog = FALSE,
  optiround = FALSE,
  nsims = 5000,
  verbose = FALSE,
  impNN = Inf,
  capNN = 2000,
  initiation = TRUE,
  savegraphs = FALSE,
  keepsims = FALSE,
  n_slope_coefs = 3,
  n_size_coefs = 5
)
```

Arguments

simfun	(function) The user-supplied function that describes the clinical trial scenario (or similar experiment) that needs to be explored. Must have as named arguments a sample size (named NN) and an arbitrary number of design parameters (one

of which will be optimized). Must return a boolean indicating whether the trial is successful or not, after performing any required computations (regressions, bootstraps) as written by the user, and never return NA.

`tar_power` (single number between 0 and 1) Target power (or more broadly, probability of success). `adsasi_1d` will seek regions where `simfun` returns `TRUE` with a frequency of `tar_power`, assuming that higher sample size equals higher probability of success.

`...` Additional named arguments to be passed to `simfun`. Some of these arguments can be functions themselves (e.g. for trying different analysis models). Any `simfun` argument without a default value must be specified here.

`optivar` (single string) Name of the `simfun` argument that needs to be optimized.

`optiwin` (numeric vector of size 2) Bounds of the region to be explored for values of `optivar`.

`optilog` (boolean) Whether `optivar` is best explored and drawn in log scale (as in the case of a ratio) or linearly. If for example `optiwin` is c(0.1,10) for a ratio, graphs will be drawn with 1 as the middle value if `optilog` is `TRUE` and 5.05 is it is `FALSE`.

`optiround` (boolean) Whether `optivar` needs to be rounded to the nearest integer to make sense for `simfun` (for example, if it is a number of centers in a cluster-randomized trial).

`nsims` (single number) Number of simulations to be run across all values of `optivar`. After initialization, simulations are run in batches of 10% of existing simulations, until `nsims` is reached.

`verbose` (boolean) Whether to print extra diagnostics messages throughout the run.

`impNN` (single number, or infinity) Sample size that is considered impossible (either computationnally, or logistically). The simulator will exit if, after 500+ simulations, it looks like the best value is above this. In practice, is mostly useful to avoid expensive computations in situations where `simfun` is not written well or is prohibitively long to compute for large sample sizes.

`capNN` (single number, or infinity) Maximum sample size that will be simulated. Also mostly useful to avoid expensive computations.

`initiation` (boolean, or numeric matrix) Either a boolean indicating whether or not to keep the first 150 simulations for the relationship inference (those tend to be far from `tar_power`), or a matrix with simulation results from a previous run which the user wants enrich with more simulations (formatted exactly as produced by `adsasi_1d` with the same `opti*` arguments). See Value and Notes below for how to get this data from the output to be able to reuse it.

`savegraphs` (boolean or string) Whether to save graphs on drive (vs. showing them in the console). If string, is interpreted as a typical name to be used (several graphs will be drawn, with iteration number, timestamp and .png file extension appended). The string can contain a filepath, but folders must already exist (e.g. with `dir.create()` from `base`, if automated).

`keepsims` (boolean or string) Whether to keep simulations and last fit in the returned object, which by default only containe the best value.

n_slope_coefs (single integer) Number of coefficients for the slope polynomial. The slope polynomial tries to model the relationship between optivar and the loss of power as sample size locally deviates from the (unknown) target.

n_size_coefs (single integer) Number of coefficients for the size polynomial. The size polynomial tries to model the relationship between optivar and the target sample size. Its shape is the most useful output of the function.

Value

A list with 2 numbers in it : minimum sample size, named min_NN, and corresponding best parameter value, named min_optival. If keepsims=TRUE, several other objects will be appended to the list (see Note).

Note

The graph modelling the relationship between parameter value and sample size is generally the most useful output, and is shown but not saved by default.

With keepsims=TRUE, the function keeps summary simulation results in the returned list, which can, among others, be used to draw the main graph again in a different style (as in Examples). The returned list will have the following extra elements : min_NN (last sample sizes simulated), min_optival (corresponding values for the parameter indicated by optivar, scaled between -1 and 1), trials (all the used simulations, including a rescaled optival), abscissae (natural-scale values for the optimization parameter, for which sample sizes have been computed), slope_natural_estimate_by_optival (slope variation by optivar values, see below for plotting), slope_confint_lower_by_optival (lower bound of confidence interval), slope_confint_higher_by_optival (higher bound of confidence interval), size_natural_estimate_by_optival (sample size variation by optivar values, see below for plotting), size_confint_lower_by_optival (lower bound of confidence interval), size_confint_higher_by_optival (higher bound of confidence interval).

The trials element can be fed back into adsasi_1d using argument initiation=x[["trials"]] (if the previous call was saved in x) to resume with the same simulations as before. Note that the second adsasi_1d call will need to use the same simfun, fixed simfun arguments, optivar, optiwin and optilog arguments for this to make sense, because values for optivar stored in trials are between -1 and +1 and are scaled using these arguments before being passed to simfun or shown to the user. If one wants to widen the window, the rescaling will need to be done manually.

Examples

```
# First, the user defines a function for their target situation. In this simple example, a 2-sample
# t-test with unequal allocation. The design parameter of interest will be the ratio of
# n1 (observations in arm 1) to NN (total sample size). Note the syntax to avoid returning NAs.
simulate_unequal_t_test = function(NN=20,ratio_n1_NN=0.5,delta=1)
 {
  n1 = round(ratio_n1_NN*NN) ; n2 = NN-n1
  yy1 = rnorm(n1) ; yy2 = rnorm(n2,delta)
  pp=NA ; try(pp <- t.test(yy1,yy2)$p.value,silent=TRUE)
  !is.na(pp) & pp<0.05
  }
simulate_unequal_t_test()
# Now we empirically find the relationship between sample size and the parameter of interest.
```

```
# Note that we can change the delta parameter directly from the adsasi_1d call.
# nsims should generally be much higher than in this fast-running example (>5000).
batch=adsasi_1d(simulate_unequal_t_test,delta=1.25,optivar="ratio_n1_NN",nsims=200,keepsims=TRUE)
# Drawing the output in a different style
plot( batch[["abscissae"]],batch[["size_natural_estimate_by_optival"]]
     ,xlab="Optimization parameter",ylab="Estimated sample size",type="o",col="red"
     )
polygon( c(batch[["abscissae"]],rev(batch[["abscissae"]]))
        ,c( batch[["size_confint_higher_by_optival"]]
           ,rev(batch[["size_confint_lower_by_optival"]])
           )
        ,col="#55000088",border=NA
        )
```

Index