Package 'POSSA'

February 28, 2022

Title Power Simulation for Sequential Analysis

Version 0.1.0

Date 2022-02-15

Description This R package serves to calculate, via simulation, power and appropriate ``stopping'' alpha boundaries (and including, optionally, futility bounds) for sequential analyses (i.e., ``group sequential design''), given any specified global error rate. This enables the sequential use of practically any significance test, as long as the underlying data can be simulated in advance to a reasonable approximation. Note: Since the specific functions are rather complex, it is much easier to understand the procedure with the examples given via https://github.com/gasparl/possa.

URL `https://github.com/gasparl/possa`

Depends R (>= 3.6.0)

Imports data.table, methods

License BSD_2_clause + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.2

R topics documented:

`pow` *Power calculation*

Description

Calculates power and local alphas based on simulated p values (which should be provided as created by the POSSA::sim function). The calculation for sequential testing involves a staircase procedure during which an initially provided set of local alphas is continually adjusted until the (approximate) specified global error rate (e.g., global alpha = .05) is reached: the value of adjustment is decreasing while global error rate is larger than specified, and increasing while global error rate is smaller than specified; a smaller step is chosen whenever the direction (increase vs. decrease) changes; the procedure stops when the global error rate is close enough to the specified one (e.g., matches it up to 4 fractional digits) or when the specified smallest step is passed. The adjustment works via a dedicated ("adjust") function that either replaces missing (NA) values with varying alternatives or (when there are no missing values) in some manner varyingly modifies the initial values (e.g. by addition or multiplication).

Usage

```
pow(
  p_values,
  alpha_locals = NULL,
  alpha_global = 0.05,
  adjust = NULL,
  adj_init = NULL,
  staircase_steps = NULL,
  alpha_precision = 5,
  fut_locals = NULL,
  multi_logic = "all",
  multi_logic_fut = "all",
  group_by = NULL,
  alpha_locals_extra = NULL,
  design_fix = NULL,
  design_seq = TRUE,
  descr_cols = TRUE,
  descr_func = summary,
  round_to = 3,
  seed = 8
)
```

Arguments

p_values	A data.frame containing the simulated iterations, looks, and corresponding H0 and H1 p value outcomes, as returned by the POSSA::sim function. (Custom data frames are also accepted, but may not work as expected.)
alpha_locals	A number, a numeric vector, or a named list of numeric vectors, that specify the initial set of local alphas that decide on statistical significance (for interim look as well as for the final look), and, if significant, stop the experiment at the given interim looks; to be adjusted via the adjust function; see the adjust parameter below. Any of the numbers included can always be NA values as well (which indicates alphas to be calculated; again, see the related adjust parameter

below). In case of a vector or a list of vectors, the length of each vector must correspond exactly to the maximum number of looks in the `p_values` data frame. When a `list` is given, the names of the list element(s) must correspond to the root of the related H0 and H1 p value column name pair(s) (in the `p_values` data frame), that is, without the `"_h0"` and `"_h1"` suffixes: for example, if the column name pairs is `"p_test4_h0"` and `"p_test4_h1"`, the name of the corresponding list element should be `"p_test4"`. If a single number or a single numeric vector is given, all potential p value column pairs are automatically detected as starting with `"p_"` prefix and ending with `"_h0"` and `"_h1"`. In case of a single vector given, each such automatically detected p value pair receives this same vector. In case of a single number given, this all elements of all vectors will be this same number. The default `NULL` value specifies "fixed design" (no interim stopping alphas) with final alpha as specified as `alpha_global`. (This is useful for cases where only futility bounds are to be set for stopping.)

`alpha_global` Global alpha (expected error rate in total); `0.05` by default.

`adjust` The function via which the initial vector local alphas is modified with each step of the staircase procedure. Three arguments are passed to it: `adj`, `orig`, and `prev`. The `adj` parameter is mandatory; it passes the pivotal changing value that, starting from an initial value (see `adj_init`), should, via the staircase steps, decrease when the global error rate is too large, and increase when the global error rate is too small. The `orig` parameter always passes the same original vector of alphas as they were provided via `alpha_locals`. The `prev` parameter (optional) passes the "latest" vector of local alphas, which were obtained in the previous adjustment step (or, in the initial run, it is the original vector, i.e., the same as `orig`). When `NULL` (default), function replaces `NA`s with the varying adjustment value (as `{ prev[is.na(orig)] = adj; return(prev) }`).

`adj_init` The initial adjustment value that is used as the `"adj"` parameter in the `"adjust"` function and is continually adjusted via the staircase steps (see `staircase_steps` parameter). When `NULL` (default), it is calculated as the global alpha divided by the maximum number of looks (Bonferroni correction), as a rough initial approximation with the assumption that `"adj"` is used as a replacement for `NA`s.

`staircase_steps`

Numeric vector that specifies the (normally decreasing) sequence of step sizes for the staircase that narrows down on the specified global error error by decreasing or increasing the adjustment value (initially: `adj_init`): the step size (numeric value) is added for increase, and subtracted for decrease. Whenever the direction (decrease vs. increase) is changed, the staircase moves on to the next step size. When the direction changes and there are no more steps remaining, the procedure is finished (regardless of the global error rate). By default (`NULL`), the `staircase_steps` is either `"0.01 * (0.5 ^ (seq(0,11,1)))"` (giving: `0.01,0.005,0.0025,...`) or `"0.5 * (0.5 ^ (seq(0,11,1)))"` (giving: `0.05,0.025,0.0125,...`). The latter is chosen when adjustment via multiplication is assumed, which is simply based on finding any multiplication sign (`\*`) in a given custom `adjust` function. The former is chosen in any other case.

`alpha_precision`

During the error rate staircase procedure, at any point when the simulated global error rate first matches the given `alpha_global` at least for the number of fractional digits given here (`alpha_precision`; default: `5`), the procedure stops and

	the results are printed. (Otherwise, the procedures finishes only when all steps given as `staircase_steps` have been used.)
`fut_locals`	Specifies local futility bounds that may stop the experiment at the given interim looks if the corresponding p value is above the given futility bound value. When `NULL` (default), sets no futility bounds. Otherwise, it follows the same logic as `alpha_locals` and has the same input possibilities (number, numeric vector, or named list of numeric vectors).
`multi_logic`	When multiple p values are evaluated for stopping rules, `multi_logic` specifies the function used for how to evaluate the multiple outcomes as a single `TRUE` or `FALSE` value that decides whether or not to stop at a given look. The default, `'all'`, specifies that all of the p values must pass the boundary for stopping. The other acceptable character input is `'any'`, which specifies that the collection stops when any of the p values pass the boundary for stopping. Instead of these strings, the actual `all` and `any` would lead to identical outcomes, respectively, but the processing would be far slower (since the string `'all'` or `'any'` inputs specify a dedicated faster internal solution). For custom combinations, any custom function can be given, which will take, as arguments, the p value columns in their given order (either in the `p_values` data frame, or as specified in `alpha_locals`), and should return a single `TRUE` or `FALSE` value.
`multi_logic_fut`	Same as `multi_logic`, but for futility bounds (for the columns specified in `fut_locals`).
`group_by`	When given as a character element or vector, specifies the factors by which to group the analysis: the `p_values` data will be divided into parts by these factors and these parts will be analyzed separately, with power and error information (and descriptives, if specified) printed per each part. By default `NULL`, it identifies factors, if any, given to the `sim` function (via `fun_obs`) that produced the given `p_values` data.
`alpha_locals_extra`	Optional extra and "non-stopper" alphas via which to evaluate p values per look, but without stopping the data collection regardless of statistical significance.
`design_fix`	Whether to calculate fixed design (as comparable alternative(s) to given "looks" of the sequential design). If `NULL` (default), shows power for maximum sample (last "look") only. If set to `TRUE`, shows power for each given "look" (interim and final) Altogether omitted when set to `FALSE`.
`design_seq`	Whether to calculate sequential design (default: `TRUE`). (Although this R package is designed for sequential analysis, it can be used for fixed designs alone too.) Altogether omitted when set to `FALSE`.
`descr_cols`	When given as a character element or vector, specifies the factors for which descriptive data should be shown (by group, if applicable). By default `TRUE`, it identifies (similar as `group_by`) factors, if any, given to the `sim` function (via `fun_obs`) that produced the given `p_values` data.
`descr_func`	Function used for printing descriptives (see `descr_cols`). By default, it uses the `summary` `base` function.
`round_to`	Number to round to.
`seed`	Number for `set.seed`; `8` by default. Set to `NULL` for random seed.

Value

Returns `data.frame`, etc.

Note

This function uses, internally, the `data.table` R package.

References

Lakens, D. (2014). Performing high-powered studies efficiently with sequential analyses: Sequential analyses. European Journal of Social Psychology, 44(7), 701–710. doi: 10.1002/ejsp.2023

See Also

`sim`

Examples

```
# some pow
```

`sim` *Simulation procedure*

Description

This function performs the simulation procedure in order to get the p values that will eventually serve for power calculations (via `pow`). The "sample" values to be tested are simulated via the given `fun_obs` function, and the significance testing is performed via the given `fun_test` function. The numbers of observations per look (for a sequential design) are specified in `n_obs`.

Usage

```
sim(fun_obs, n_obs, fun_test, n_iter = 5000, seed = 8)
```

Arguments

`fun_obs`	A `function` that creates the observations (i.e., the "sample"; all values for the dependent variable(s)). The arguments that specify the observation numbers in this function have to be provided via `n_obs`: the respective maximum observation number, given in `n_obs`, will be passed to the `fun_obs`, and, in case of sequential testing, the resulting observations will be reduced to the specified (smaller) number(s) of observations for each given interim "look" (as a simulation for what would happen if collection was stopped at that given look). Importantly, a special "`_h`" suffix (e.g., "`var1_h`") is required to specify the variable(s) that differ(s) depending on whether or not the null hypothesis ("H0") or the alternative hypothesis ("H1") is true. This will be internally transformed into "`_h0`" and "`_h1`" endings (e.g., `var1_h0`, `var1_h1`) for the arguments of the `fun_test`

	function. Optionally, the `fun_obs` can be passed additional arguments (via a `list`); see Details.
`n_obs`	A numeric vector or a (named) list of numeric vectors. Specifies the numbers of observations (i.e., samples sizes) that are to be generated from `fun_obs`, to be then tested in `fun_test`. If a single vector is given, this will be used for all arguments in the `fun_obs` function. Otherwise, if a named list of numeric vectors is given, the names must correspond exactly to the argument names in `fun_obs`, so that the respective numeric vectors are used for each given variable.
`fun_test`	The function for significance testing. The observation values created in `fun_obs` (with observation numbers specified in `n_obs`) will be passed into this `fun_test` function as arguments (with `_h` suffixes extended to `_h0` and `_h1`), to be used in the given statistical significance tests in this function. The test results must include a pair (or pairs) of p values for H0 and H1 outcomes, where each p value must be specified with a "`p_`" prefix and a "`_h0`" suffix for H0 outcome or a "`_h1`" suffix for H1 outcome (e.g., `p_h0`, `p_h1`; `p_ttest_h0`, `p_ttest_h1`). Each of these p values will be separately stored in a dedicated column of the `data.frame` returned by the `sim` function. Optionally, the `fun_test` can return other miscellaneous outcomes too, such as effect sizes or confidence interval limits; these will then be stored in dedicated columns in the resulting `data.frame`.
`n_iter`	Number of iterations (default: 5000).
`seed`	Number for `set.seed`; `8` by default. Set to `NULL` for random seed.

Details

Optionally, `fun_obs` can be provided in `list` format for the convenience of exploring varying factors (e.g., different effect sizes, correlations) at once, without writing a dedicated `fun_obs` function for each combination, and each time separately running the simulation and the power calculation. In this case, the first element of the list must be the actual `function`, which contains certain parameters for specifying varying factors, while the rest of the elements should contain the various argument values for these parameters of the function as named elements of the list (e.g., `list(my_function,factor1,factor2)`), with the name corresponding to the parameter name in the function, and the varying values. When so specified, a separate simulation procedure will be run for each combination of the given factors (or, if only one factor is given, for each element of that factor). The `POSSA::pow` function will be able to automatically detect (by default) the factors generated this way in the present `POSSA::sim` function, in order to calculate power separately for each factor combination.

Value

Returns a `data.frame` that includes the columns `iter` (the iterations of the simulation procedure numbered from `1` to `n_iter`), `look` (the interim "looks" numbered from `1` to the maximum number of looks, including the final one), and the information returned by the `fun_test` function for H0 and H1 outcomes (mainly p values; but also other, optional information, if any) and the corresponding observation numbers.

References

Lakens, D. (2014). Performing high-powered studies efficiently with sequential analyses: Sequential analyses. European Journal of Social Psychology, 44(7), 701–710. doi: 10.1002/ejsp.2023

See Also

pow

Examples

```
# user-defined function to specify sample(s)
custom_sample_v1 = function(v1, v2_h) {
    list(v1 = rnorm(v1, mean = 0, sd = 10),
         v2_h0 = rnorm(v2_h, mean = 0, sd = 10),
         v2_h1 = rnorm(v2_h, mean = 5, sd = 10))
}


# user-defined function to specify significance test(s)
custom_test_v1 = function(sampl) {
    c(
        p_h0 = t.test(sampl$v2_h0, sampl$v1, var.equal = TRUE)$p.value,
        p_h1 = t.test(sampl$v2_h1, sampl$v1, var.equal = TRUE)$p.value
    )
}

df_ps_v1 = sim(
    fun_obs = custom_sample_v1,
    n_obs = c(33, 65, 98),
    fun_test = custom_test_v1
)
```

Index