Package 'POSSA'

April 22, 2023

Title Power Simulation for Sequential Analyses and Multiple Hypotheses

Version 0.6.4

Date 2023-04-22

Description Calculates, via simulation, power and appropriate stopping alpha boundaries (and/or futility bounds) for sequential analyses (i.e., group sequential design) as well as for multiple hypotheses (multiple tests included in an analysis), 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. Lukács (2022) <doi:10.21105/joss.04643>.

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 7.2.2

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

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Gáspár Lukács [aut, cre]

Maintainer Gáspár Lukács <`lkcsgaspar@gmail.com`>

Repository CRAN

Date/Publication 2023-04-22 07:50:02 UTC

R topics documented:

`get_p` *Extract p-value from tests*

Description

This function attempts to extract p values from certain tests where it could otherwise be complicated to do so. Please make sure, in case of each new test, whether the function actually returns the values you want.

Usage

```
get_p(x)

## S3 method for class 'aov'
get_p(x)

## S3 method for class 'aovlist'
get_p(x)
```

Arguments

`x`	Test results.

Details

Supported functions: all tests that return the p value as `p.value` (including most R `stats` test functions, having `htest` class), and the `aov()` function (`aov` and `aovlist` classes).

Value

Returns either a single p value or, in case of multiple p values, a list or nested list with each p value.

Methods (by class)

- `get_p(aov)`: get_p method for class 'aov'
- `get_p(aovlist)`: get_p method for class 'aovlist'

Examples

```
get_p(t.test(extra ~ group, data = sleep))
# returns 0.07939414
# same as printed via t.test(extra ~ group, data = sleep)

get_p(prop.test(c(83, 90, 129, 70), c(86, 93, 136, 82)))
# returns 0.005585477,
# same as printed prop.test(c(83, 90, 129, 70), c(86, 93, 136, 82))

get_p(aov(yield ~ block + N * P * K, npk))
# returns list of p values
# corresponds to summary(aov(yield ~ block + N * P * K, npk))

get_p(aov(yield ~ N * P * K + Error(block), npk))
# returns nested list of p values (effects per error term)
# again corresponds printed p values via summary()
```

---

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 type 1 error rate (e.g., global alpha = .05) is reached: the value of adjustment is decreasing while global type 1 error rate is larger than specified, and increasing while global type 1 error rate is smaller than specified; a smaller step is chosen whenever the direction (increase vs. decrease) changes; the procedure stops when the global type 1 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 = TRUE,
  adj_init = NULL,
  staircase_steps = NULL,
  alpha_precision = 5,
  fut_locals = NULL,
  multi_logic_a = "all",
  multi_logic_fut = "all",
  multi_logic_global = "any",
  group_by = NULL,
  alpha_loc_nonstop = NULL,
  round_to = 5,
  iter_limit = 100,
  seed = 8,
  prog_bar = FALSE,
  hush = FALSE
)
```

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 looks as well as for the final look), and, if significant, stop the experiment at the given interim look; 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 pair 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, all elements of all vectors will be assigned this same number (up to the maximum number of looks). If a list is given and any of the elements contain just one number, it will be extended into a vector (up to the maximum number of looks). The default `NULL` value specifies "fixed design" (no interim stopping alphas) with final alpha as specified as `alpha_global`, without adjustment procedure as long as the `adjust` argument is also left as default `TRUE`. (This is useful for cases where only futility bounds are to be set for stopping.)
`alpha_global`	Global alpha (expected type 1 error rate in total); `0.05` by default. See also `multi_logic_global` for when multiple p values are being evaluated.
`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 type 1 error rate is too large, and increase when the global type 1 error rate is too small. The `orig` parameter (optional) 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 TRUE (default), if the given alpha_locals contains any NAs, an adjust function is given internally that simply replaces NAs with the varying adjustment value (as { prev[is.na(orig)] = adj; return(prev) }). If alpha_locals contains no NAs, an adjust function is given that multiplies each original local alpha with the varying adjustment value (as { return(orig * adj) }). When set to FALSE, there will be no adjustment (staircase procedure omitted): this is useful to calculate the global type 1 error rate for any given set of local alphas. Furthermore, if both adjust and alpha_locals are left as default (TRUE and NULL), the staircase procedure will be omitted.

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), assuming that "adj" is used as a replacement for NAs, adj_init is calculated as the global alpha divided by the maximum number of looks (Bonferroni correction), as a rough initial approximation. However, multiplication is assumed when finding any multiplication sign (*) in a given custom adjust function: in such a case, adj_init will be 1 by default.

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 staircase procedure, at any point when the simulated global type 1 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_a
: When multiple p values are evaluated for local alpha stopping rules, multi_logic_a specifies the function used for how to evaluate the multiple significance outcomes (p values being below or above the given local alphas) 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 be below the local 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_a (again with 'all' as default), but for futility bounds (for the columns specified in fut_locals).

multi_logic_global
: Similar as multi_logic_a, but for the calculation of the global type 1 error rate (again: in case of multiple p values being evaluated; otherwise this parameter is not relevant), and with 'any' by default. This default means that if any of the p values under evaluation (specified via alpha_locals or detected automatically) is significant (p value below the given local alpha at the stopping of the simulated "experiment" iteration) in case of the H0 scenario, this is calculated as a type 1 error. If 'all' were specified, only cases with all p evaluated values being significant are counted as type 1 errors. In either case, the ratio of outcomes with such type 1 errors (out of all iterations) gives the global type 1 error rate, which is intended to (approximately) match (via the adjustment procedure) the value specified as alpha_global. This global type 1 error is also what is printed to the console in the end as the "combined" global error rate. Furthermore, the logic given here is also used for the calculation of the "combined" global power printed to the console. In this case, the 'any' logic, for example, would mean that, if any of the p values are significant at the end of the experiment, this is a positive finding. The global power is then the ratio of iterations with such positive findings.

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 calculated per each part. By default (NULL), it identifies factors, if any, given to the sim function (via fun_obs) that created the given p_values data.

alpha_loc_nonstop
: Optional "non-stopper" alphas via which to evaluate p values per look, but without stopping the data collection regardless of statistical significance. Must be a list with names indicating p value column name pairs, similarly as for the alpha_locals argument; see alpha_locals for details.

round_to
: Number of fractional digits (default: 5) to round to, for the displayed numeric information (such as alphas and power; mainly for default value for printing).

iter_limit
: In some specific cases of unideal/wrong input, the staircase may get stuck at a given step's loop process. The iter_limit parameter specifies the number (by default 100) at which the script pauses the loop and offers to the user that the procedure be ceased. If the user chooses to continue, the offer will always be posed again after the same number of iterations (e.g., by default, after 100, at 200, then 300, etc.).

seed	Number for set.seed; 8 by default. Set to NULL for random seed.
prog_bar	Logical, FALSE by default. If TRUE, shows progress bar.
hush	Logical. If TRUE, prevents printing any details (or the progress bar) to console.

Value

The returns a list (with class "possa_pow_list") that includes all details of the calculated power, T1ER, and sample information. This list can be printed legibly (via POSSA's print() method).

Note

For the replicability, in case the adjust function uses any randomization, set.seed is executed in the beginning of this function, each time it is called; see the seed parameter.

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

See Also

sim

Examples

```
# below is a (very) minimal example
# for more, see the vignettes via https://github.com/gasparl/possa#usage

# create sampling function
customSample = function(sampleSize) {
    list(
        sample1 = rnorm(sampleSize, mean = 0, sd = 10),
        sample2_h0 = rnorm(sampleSize, mean = 0, sd = 10),
        sample2_h1 = rnorm(sampleSize, mean = 5, sd = 10)
    )
}

# create testing function
customTest = function(sample1, sample2_h0, sample2_h1) {
 c(
   p_h0 = t.test(sample1, sample2_h0, 'less', var.equal = TRUE)$p.value,
   p_h1 = t.test(sample1, sample2_h1, 'less', var.equal = TRUE)$p.value
 )
}

# run simulation
dfPvals = sim(
    fun_obs = customSample,
    n_obs = 80,
    fun_test = customTest,
    n_iter = 1000
)

# get power info
```

```
pow(dfPvals)
```

`print.possa_pow_df` *Print pow results data frame*

Description

Prints, in a readable manner, the main information from any of the data frames containing power information from the list created by the `POSSA::pow` function. This is an extension (method) of the base R `print` function, so it can be called simply as `print()`.

Usage

```
## S3 method for class 'possa_pow_df'
print(x, round_to = 5, possa_title = TRUE, ...)
```

Arguments

`x`	Power information `data.frame` included in the list returned by the `POSSA::pow` function.
`round_to`	Number of fractional digits to round to, for the displayed numbers (5 by default).
`possa_title`	Set to `FALSE` to omit title printing.
`...`	(Allow additional arguments for technical reasons.)

Value

Returns nothing (`NULL`); this method serves only to print information to the console.

See Also

`pow`, `print.possa_pow_list`

`print.possa_pow_list` *Print pow results*

Description

Prints, in a readable manner, the main information from the list created by the `POSSA::pow` function, calling `print.possa_pow_df` for each of the POSSA power information data frames in the list. This is an extension (method) of the base R `print` function, so it can be called simply as `print()`.

Usage

```
## S3 method for class 'possa_pow_list'
print(x, round_to = NA, ...)
```

Arguments

`x`	The `list` returned by the `POSSA::pow` function.
`round_to`	Number of fractional digits to round to, for the displayed numbers. The default is the value passed from the `POSSA::pow` function (stored in the returned list).
`...`	(Allow additional arguments for technical reasons.)

Value

Returns nothing (`NULL`); this method serves only to print information to the console.

See Also

`pow`, `print.possa_pow_df`

`print.possa_sim_df` *Print sim results*

Description

Prints information about the simulated p values created by the `POSSA::sim` function. This is an extension (method) of the base R `print` function, so it can be called simply as `print()`.

Usage

```
## S3 method for class 'possa_sim_df'
print(x, group_by = NULL, descr_cols = TRUE, descr_func = summary, ...)
```

Arguments

`x`	The `data.frame` returned by the `POSSA::sim` function.
`group_by`	When given as a character element or vector, specifies the factors by which to group the descriptives: the `x` data will be divided into parts by these factors and these parts will be analyzed separately, with descriptives printed per each part. By default (`NULL`), it identifies factors, if any, given to the `sim` function (via `fun_obs`) that created the given `x` data.
`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 `NULL`, it identifies (similar as `group_by`) factors, if any, given to the `sim` function (via `fun_obs`) that produced the given `x` data.
`descr_func`	Function used for printing descriptives (see `descr_cols`). By default, it uses the `summary` (`base`) function.
`...`	(Allow additional arguments for technical reasons.)

Value

Returns nothing (`NULL`); this method serves only to print information to the console.

See Also

sim

---

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 observation values ("sample") 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 = 45000,
  adjust_n = 1,
  seed = 8,
  pair = NULL,
  ignore_suffix = FALSE,
  prog_bar = FALSE,
  hush = FALSE
)
```

Arguments

fun_obs	A function that creates the observations (i.e., the "sample"; all values for the dependent variable(s)). The respective maximum observation number(s), given in n_obs, will be passed to the fun_obs. For this, the returned value must be a named list, where the names correspond exactly to the arguments in fun_test. In case of sequential testing, the observations returned by fun_obs 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), to be used in fun_test. 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 by fun_obs and then tested in fun_test. If a single vector is given, this will be used for all observation number arguments in the fun_obs and for the sample size adjustments for the arguments in the fun_test functions. Otherwise, if a named list of numeric vectors is given, the names must correspond exactly to the argument names in fun_obs and fun_test, so that the respective numeric vectors are used for each

given sample variable. For convenience, in case of a "_h" suffix, the variable will be divided into names with "_h0" and "_h1" suffixes for fun_test (but not for fun_obs); see Details.

fun_test
: The function for significance testing. The list of samples returned by fun_obs (with observation numbers specified in n_obs) will be passed into this fun_test function as arguments, to be used in the given statistical significance tests in this function. To correctly calculate the sample sizes in POSSA::pow, the argument names for the sample that varies depending on whether the null (H0) and alternative (H1) hypothesis is true should be indicated with "_h0" and "_h1" suffixes, respectively, with a common root (so, e.g., "var_x_h0" and "var_x_h1"). Then, in the resulting data.frame, their sample size (which must always be identical) will be automatically merged into a single column with a trimmed "_h" suffix (e.g., "var_x_h"). (Otherwise, the sample sizes of both H0 and H1 would be calculated toward the total expected sample in either case, which is of course incorrect. There are internal checks to prevent this, but the intended total sample size can also be double-checked in the returned data.frame's .n_total column.) Within-subject observations, i.e., multiple observations per group, should be specified with "GRP" prefix for a single group (e.g., simply "GRP", or "GRP_mytest") and, for multiple groups, "grp_" prefix with a following group name (e.g., "grp_1" or "grp_alpha"); the numbers of multiple observations in each group can then be specified in fun_obs via their group name (since the respective numbers of observations should always be the same anyway); see Examples. To be recognized by the POSSA::pow function, the fun_test must return a named vector including a pair (or pairs) of p values for H0 and H1 outcomes, where each p value's name 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). The simulated outcomes (per iteration) for 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: 45000).

adjust_n
: Adjust total number of observations via simple multiplication. Might be useful in some specific cases, e.g. if for some reason multiple p values are derived from the same sample without specifying grouping (GRP or grp_ in fun_test), which would then lead to incorrect (too many, multiplied) totals; for example, in case of four observations obtained from the same sample, the value 1/4 could be given. (The default value is 1.)

seed
: Number for set.seed; 8 by default. Set to NULL for random seed.

pair
: Logical or NULL. By default NULL, the algorithm assumes paired samples included among the observations in case of any grouping via the fun_test parameters ("GRP"/"grp"), and no paired samples otherwise. In case of paired samples included, within each look, the same vector indexes to remove elements from the given observations. In general, this should not substantially affect the outcomes of independent samples (assuming that their order is truly independent), but this depends on how the random samples are generated in the fun_obs function. To be safe and avoid any potential bias, it is best to avoid this paired sampling

	mechanism when no paired samples are included. To override the default, set to TRUE for paired samples scenario (paired sampling), or to FALSE for no paired samples scenario (random subsampling of each sample). (Might be useful for testing or some very specific procedures, e.g. where grouping is not indicated despite paired samples.)
ignore_suffix	Set to NULL to give warnings instead of errors for internally detected consistency problems with the _h0/_h1 suffixes in the fun_test function arguments. Set to TRUE to completely ignore these (neither error nor warning). (Might be useful for testing or some very specific procedures.)
prog_bar	Logical, FALSE by default. If TRUE, shows progress bar.
hush	Logical, FALSE by default. If TRUE, prevents printing any details to console.

Details

To specify a variable that differs depending on whether the null hypothesis ("H0") or the alternative hypothesis ("H1") is true, a pair of samples are needed for fun_test, for which the argument names should have an identical root and "_h0" and "_h1" endings, such as "var_x_h0" (for sample in case of H0) and "var_x_h1" (for sample in case of H1). Then, since the observation number for this pair will always be the same, as a convenience, parameters with "_h0" and "_h1" endings specifically can be specified together in n_obs with the last "0"/"1" character dropped, hence ending with "_h". So, for example, "var_x_h = c(30, 60, 90)" will be automatically adjusted to specify the observation numbers for both "var_x_h0" and "var_x_h1". In that case, fun_obs must have a single argument "var_x_h", while fun_test must have both full names as arguments ("var_x_h0" and "var_x_h1").

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=c(1, 2, 3), factor2=c(0,5))), with the name corresponding to the parameter name in the function, and the varying values (numbers or strings). 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 (with class "possa_sim_df") 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, as well as the total observation number per each look under a dedicated .n_total column. When this data frame is printed to the console (via POSSA's print() method), the head (first few lines) of the data is shown, as well as, in case of any varying factors included, summary information per factor combination.

Note

For the replicability (despite the randomization), `set.seed` is executed in the beginning of this function, each time it is called; see the seed parameter.

See Also

`pow`

Examples

```
# below is a (very) minimal example
# for more, see the vignettes via https://github.com/gasparl/possa#usage

# create sampling function
customSample = function(sampleSize) {
    list(
        sample1 = rnorm(sampleSize, mean = 0, sd = 10),
        sample2_h0 = rnorm(sampleSize, mean = 0, sd = 10),
        sample2_h1 = rnorm(sampleSize, mean = 5, sd = 10)
    )
}

# create testing function
customTest = function(sample1, sample2_h0, sample2_h1) {
 c(
   p_h0 = t.test(sample1, sample2_h0, 'less', var.equal = TRUE)$p.value,
   p_h1 = t.test(sample1, sample2_h1, 'less', var.equal = TRUE)$p.value
 )
}

# run simulation
dfPvals = sim(
    fun_obs = customSample,
    n_obs = 80,
    fun_test = customTest,
    n_iter = 1000
)

# get power info
pow(dfPvals)
```

Index