Package 'simhelpers'

January 10, 2025

Type Package

Title Helper Functions for Simulation Studies

Version 0.3.1

Description Calculates performance criteria measures and associated Monte Carlo standard errors for simulation results. Includes functions to help run simulation studies, following a general simulation workflow that closely aligns with the approach described by Morris, White, and Crowther (2019) <DOI:10.1002/sim.8086>. Also includes functions for calculating bootstrap confidence intervals (including normal, basic, studentized, percentile, bias-corrected, and bias-corrected-and-accelerated) with tidy output, as well as for extrapolating confidence interval coverage rates and hypothesis test rejection rates following techniques suggested by Boos and Zhang (2000) <DOI:10.1080/01621459.2000.10474226>.

URL https://meghapsimatrix.github.io/simhelpers/

BugReports https://github.com/meghapsimatrix/simhelpers/issues

Depends R (>= 2.10)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

SystemRequirements RStudio

Imports stats, furrr, tidyr, rstudioapi, Rdpack

Suggests dplyr, tibble, purrr, future, knitr, rmarkdown, pkgdown, covr, testthat, kableExtra, ggplot2, broom, boot

RdMacros Rdpack

VignetteBuilder knitr

NeedsCompilation no

Author Megha Joshi [aut, cre] (<https://orcid.org/0000-0001-7936-076X>), James Pustejovsky [aut] (<https://orcid.org/0000-0003-0591-9465>)

Maintainer Megha Joshi <megha.j456@gmail.com>

Repository CRAN

Date/Publication 2025-01-10 21:50:02 UTC

Contents

`alpha_res` *Cronbach's alpha simulation results*

Description

A dataset containing simulation results from estimating Cronbach's alpha and its variance.

Usage

```
alpha_res
```

Format

A tibble with 1,000 rows and 3 variables:

A estimate of alpha.

Var_A estimate of the variance of alpha.

true_param true alpha used to generate the data.

bootstrap_CIs *Calculate one or multiple bootstrap confidence intervals*

Description

Calculate one or multiple bootstrap confidence intervals, given a sample of bootstrap replications.

Usage

```
bootstrap_CIs(
  boot_est,
  boot_se = NULL,
  est = NULL,
  se = NULL,
  influence = NULL,
  CI_type = "percentile",
  level = 0.95,
  B_vals = length(boot_est),
  reps = 1L,
  format = "wide",
  seed = NULL
)
```

Arguments

boot_est	vector of bootstrap replications of an estimator.
boot_se	vector of estimated standard errors from each bootstrap replication.
est	numeric value of the estimate based on the original sample. Required for CI_type = "normal", CI_type = "basic", CI_type = "student", and CI_type = "bias-corrected".
se	numeric value of the estimated standard error based on the original sample. Required for CI_type = "student".
influence	vector of empirical influence values for the estimator. Required for CI_type = "BCa".
CI_type	Character string or vector of character strings indicating types of confidence intervals to calculate. Options are "normal", "basic", "student", "percentile" (the default), "bias-corrected", or "BCa".
level	numeric value between 0 and 1 for the desired coverage level, with a default of 0.95.
B_vals	vector of sub-sample sizes for which to calculate confidence intervals. Setting B_vals = length(boot_est) (the default) will return bootstrap confidence intervals calculated on the full set of bootstrap replications. For B_vals < length(boot_est), confidence intervals will be calculated after sub-sampling (without replacement) the bootstrap replications.
reps	integer value for the number of sub-sample confidence intervals to generate when B_vals < length(boot_est), with a default of reps = 1.

format	character string controlling the format of the output. If format = "wide" (the default), then different types of confidence intervals will be returned in separate columns. If format = "long", then confidence intervals of different types will appear on different rows of dataset. If format = "wide-list", then different types of confidence intervals will be returned in separate columns and the result will be wrapped in an unnamed list.
seed	Single numeric value to which the random number generator seed will be set. Default is NULL, which does not set a seed.

Details

Confidence intervals are calculated following the methods described in Chapter 5 of Davison and Hinkley (1997). For basic non-parametric bootstraps, the methods are nearly identical to the implementation in boot.ci from the boot package.

Value

If format = "wide", the function returns a data.frame with reps rows per entry of B_vals, where each row contains confidence intervals for one sub-sample replication.

If format = "long", the function returns a data.frame with one row for each CI_type, each replication, and each entry of B_vals, where each row contains a single confidence interval for one sub-sample replication.

If format = "wide-list", then the output will be structured as in format = "wide" but will be wrapped in an unnamed list, which makes it easier to sore the output in a tibble, and will be assigned the class "bootstrap_CIs".

References

Davison, A.C. and Hinkley, D.V. (1997). _Bootstrap Methods and Their Application_, Chapter 5. Cambridge University Press.

Examples

```
# generate t-distributed data
N <- 50
mu <- 2
nu <- 5
dat <- mu + rt(N, df = nu)

# create bootstrap replications
f <- \(x) {
 c(
   M = mean(x, trim = 0.1),
   SE = sd(x) / sqrt(length(x))
 )
}

booties <- replicate(399, {
  sample(dat, replace = TRUE, size = N) |>
  f()
```

```
})

res <- f(dat)

# calculate bootstrap CIs from full set of bootstrap replicates
bootstrap_CIs(
  boot_est = booties[1,],
  boot_se = booties[2,],
  est = res[1],
  se = res[2],
  CI_type = c("normal","basic","student","percentile","bias-corrected"),
  format = "long"
)

# Calculate bias-corrected-and-accelerated CIs
inf_vals <- res[1] - sapply(seq_along(dat), \(i) f(dat[-i])[1])
bootstrap_CIs(
  boot_est = booties[1,],
  est = res[1],
  influence = inf_vals,
  CI_type = c("percentile","bias-corrected","BCa"),
  format = "long"
)

# calculate multiple bootstrap CIs using sub-sampling of replicates
bootstrap_CIs(
  boot_est = booties[1,],
  boot_se = booties[2,],
  est = res[1],
  se = res[2],
  CI_type = c("normal","basic","student","percentile","bias-corrected"),
  B_vals = 199,
  reps = 4L,
  format = "long"
)

# calculate multiple bootstrap CIs using sub-sampling of replicates,
# for each of several sub-sample sizes.
bootstrap_CIs(
  boot_est = booties[1,],
  boot_se = booties[2,],
  est = res[1],
  se = res[2],
  CI_type = c("normal","basic","student","percentile"),
  B_vals = c(49,99,199),
  reps = 4L,
  format = "long"
)
```

`bootstrap_pvals` *Calculate one or multiple bootstrap p-values*

Description

Calculate one or multiple bootstrap p-values, given a bootstrap sample of test statistics.

Usage

```
bootstrap_pvals(
  boot_stat,
  stat,
  alternative = "two-sided",
  B_vals = length(boot_stat),
  reps = 1L,
  enlist = FALSE,
  seed = NULL
)
```

Arguments

boot_stat	vector of bootstrap replications of a test statistic.
stat	numeric value of the test statistic based on the original sample.
alternative	a character string specifying the alternative hypothesis, must be one of "two-sided" (the default), "greater" or "less".
B_vals	vector of sub-sample sizes for which to calculate p-values. Setting B_vals = length(boot_stat) (the default) will return a single p-value calculated on the full set of bootstrap replications. For B_vals < length(boot_stat), p-values will be calculated after sub-sampling (without replacement) the bootstrap replications.
reps	integer value for the number of sub-sample p-values to generate when B_vals < length(boot_stat), with a default of reps = 1.
enlist	logical indicating whether to wrap the returned values in an unnamed list, with a default of FALSE. Setting enlist = TRUE makes it easier to store the output as a single entry in a tibble.
seed	Single numeric value to which the random number generator seed will be set. Default is NULL, which does not set a seed.

Details

p-values are calculated by comparing stat to the distribution of boot_stat, which is taken to represent the null distribution of the test statistic. If alternative = "two-sided" (the default), then the p-value is the proportion of the bootstrap sample where the absolute value of the bootstrapped statistic exceeds the absolute value of the original statistic. If alternative = "greater", then the p-value is the proportion of the bootstrap sample where the value of the bootstrapped statistic is larger than the original statistic. If alternative = "less", then the p-value is the proportion of the bootstrap sample where the value of the bootstrapped statistic is less than the original statistic.

Value

The format of the output depends on several contingencies. If only a single value of `B_vals` is specified and `reps = 1`, then the function returns a vector with a single p-value. If only a single value of `B_vals` is specified but `B_vals < length(boot_stat)` and `reps > 1`, then the function returns a vector p-values, with an entry for each sub-sample replication. If `B_vals` is a vector of multiple values, then the function returns a list with one entry per entry of `B_vals`, where each entry is a vector of length `reps` with entries for each sub-sample replication.

If `enlist = TRUE`, then results will be wrapped in an unnamed list, which makes it easier to sore the output in a tibble.

References

Davison, A.C. and Hinkley, D.V. (1997). _Bootstrap Methods and Their Application_, Chapter 4. Cambridge University Press.

Examples

```
# generate data from two distinct populations
dat <- data.frame(
  group = rep(c("A","B"), c(40, 50)),
  y = c(
    rgamma(40, shape = 7, scale = 2),
    rgamma(50, shape = 3, scale = 4)
  )
)
stat <- t.test(y ~ group, data = dat)$statistic

# create bootstrap replications under the null of no difference
boot_dat <- dat
booties <- replicate(399, {
  boot_dat$group <- sample(dat$group)
  t.test(y ~ group, data = boot_dat)$statistic
})

# calculate bootstrap p-values from full set of bootstrap replicates
bootstrap_pvals(boot_stat = booties, stat = stat)

# calculate multiple bootstrap p-values using sub-sampling of replicates
bootstrap_pvals(
  boot_stat = booties, stat = stat,
  B_vals = 199,
  reps = 4L
)

# calculate multiple bootstrap p-values using sub-sampling of replicates,
# for each of several sub-sample sizes.
bootstrap_pvals(
  boot_stat = booties, stat = stat,
  B_vals = c(49,99,199),
  reps = 4L
)
```

bundle_sim *Bundle functions into a simulation driver function*

Description

Bundle a data-generation function, a data-analysis function, and (optionally) a performance summary function into a simulation driver.

Usage

```
bundle_sim(
  f_generate,
  f_analyze,
  f_summarize = NULL,
  reps_name = "reps",
  seed_name = "seed",
  summarize_opt_name = "summarize",
  row_bind_reps = TRUE
)
```

Arguments

f_generate	function for data-generation
f_analyze	function for data-analysis. The first argument must be the data, in the format generated by f_analyze().
f_summarize	function for calculating performance summaries across replications. The first argument must be the replicated data analysis results. Default is NULL, so that no summary function is used.
reps_name	character string to set the name of the argument for the number of replications, with a default value of "reps".
seed_name	character string to set the name of the argument for the seed option, with a default value of "seed". Set to NULL to remove the argument from the simulation driver.
summarize_opt_name	character string to set the name of the argument for where to apply f_summarize to the simulation results, with a default value of "summarize". Ignored if no f_summarize function is specified. Set to NULL to remove the argument from the simulation driver.
row_bind_reps	logical indicating whether to combine the simulation results into a data frame using rbind(), with a default value of TRUE. If FALSE, then the function will return replications in a list and so f_summarize must be able to take a list as its first argument.

Value

A function to repeatedly run the ‘f_generate‘ and ‘f_analyze‘ functions and (optionally) apply ‘f_summarize‘ to the resulting replications.

Examples

```
f_G <- rnorm
f_A <- function(x, trim = 0) data.frame(y_bar = mean(x, trim = trim))
f_S <- function(x, calc_sd = FALSE) {
  if (calc_sd) {
    res_SD <- apply(x, 2, sd)
    res <- data.frame(M = colMeans(x), SD = res_SD)
  } else {
    res <- data.frame(M = colMeans(x))
  }
  res
}

# bundle data-generation and data-analysis functions
sim1 <- bundle_sim(f_generate = f_G, f_analyze = f_A)
args(sim1)
res1 <- sim1(4, n = 70, mean = 0.5, sd = 1, trim = 0.2)
res1

# bundle data-generation, data-analysis, and performance summary functions
sim2 <- bundle_sim(f_generate = f_G, f_analyze = f_A, f_summarize = f_S)
args(sim2)
res2 <- sim2(24, n = 7, mean = 0, sd = 1, trim = 0.2, calc_sd = TRUE)
res2

# bundle data-generation and data-analysis functions, returning results as a list
sim3 <- bundle_sim(f_generate = f_G, f_analyze = f_A, row_bind_reps = FALSE)
args(sim3)
res3 <- sim3(4, n = 70, mean = 0.5, sd = 3, trim = 0.2)
res3
```

---

calc_absolute *Calculate absolute performance criteria and MCSE*

Description

Calculates absolute bias, variance, mean squared error (mse) and root mean squared error (rmse). The function also calculates the associated Monte Carlo standard errors.

Usage

```
calc_absolute(
  data,
```

```
  estimates,
  true_param,
  criteria = c("bias", "variance", "stddev", "mse", "rmse"),
  winz = Inf
)
```

Arguments

data	data frame or tibble containing the simulation results.
estimates	vector or name of column from data containing point estimates.
true_param	vector or name of column from data containing corresponding true parameters.
criteria	character or character vector indicating the performance criteria to be calculated, with possible options "bias", "variance", "stddev", "mse", and "rmse".
winz	numeric value for winsorization constant. If set to a finite value, estimates will be winsorized at the constant multiple of the inter-quartile range below the 25th percentile or above the 75th percentile of the distribution. For instance, setting winz = 3 will truncate estimates that fall below P25 - 3 * IQR or above P75 + 3 * IQR.

Value

A tibble containing the number of simulation iterations, performance criteria estimate(s) and the associated MCSE.

Examples

```
calc_absolute(data = t_res, estimates = est, true_param = true_param)
```

---

calc_coverage *Calculate confidence interval coverage, width and MCSE*

Description

Calculates confidence interval coverage and width. The function also calculates the associated Monte Carlo standard errors. The confidence interval percentage is based on how you calculated the lower and upper bounds.

Usage

```
calc_coverage(
  data,
  lower_bound,
  upper_bound,
  true_param,
  criteria = c("coverage", "width"),
  winz = Inf
)
```

Arguments

data	data frame or tibble containing the simulation results.
lower_bound	vector or name of column from data containing lower bounds of confidence intervals.
upper_bound	vector or name of column from data containing upper bounds of confidence intervals.
true_param	vector or name of column from data containing corresponding true parameters.
criteria	character or character vector indicating the performance criteria to be calculated, with possible options "coverage" and "width".
winz	numeric value for winsorization constant. If set to a finite value, estimates will be winsorized at the constant multiple of the inter-quartile range below the 25th percentile or above the 75th percentile of the distribution. For instance, setting winz = 3 will truncate estimates that fall below P25 - 3 * IQR or above P75 + 3 * IQR.

Value

A tibble containing the number of simulation iterations, performance criteria estimate(s) and the associated MCSE.

Examples

```
calc_coverage(data = t_res, lower_bound = lower_bound,
              upper_bound = upper_bound, true_param = true_param)
```

---

calc_rejection *Calculate rejection rate and MCSE*

Description

Calculates rejection rate. The function also calculates the associated Monte Carlo standard error.

Usage

```
calc_rejection(data, p_values, alpha = 0.05, format = "wide")
```

Arguments

data	data frame or tibble containing the simulation results.
p_values	vector or name of column from data containing p-values.
alpha	scalar or vector indicating the nominal alpha level(s). Default value is set to the conventional .05.
format	option "wide" (the default) will produce a tibble with one row, with separate variables for each specified alpha. Option "long" will produce a tibble with one row per specified alpha.

Value

A tibble containing the number of simulation iterations, performance criteria estimate and the associated MCSE.

Examples

```
calc_rejection(data = t_res, p_values = p_val)
```

`calc_relative` *Calculate relative performance criteria and MCSE*

Description

Calculates relative bias, mean squared error (relative mse), and root mean squared error (relative rmse). The function also calculates the associated Monte Carlo standard errors.

Usage

```
calc_relative(
  data,
  estimates,
  true_param,
  criteria = c("relative bias", "relative mse", "relative rmse"),
  winz = Inf
)
```

Arguments

`data`	data frame or tibble containing the simulation results.
`estimates`	vector or name of column from `data` containing point estimates.
`true_param`	vector or name of column from `data` containing corresponding true parameters.
`criteria`	character or character vector indicating the performance criteria to be calculated, with possible options `"relative bias"`, `"relative mse"`, and `"relative rmse"`.
`winz`	numeric value for winsorization constant. If set to a finite value, estimates will be winsorized at the constant multiple of the inter-quartile range below the 25th percentile or above the 75th percentile of the distribution. For instance, setting `winz = 3` will truncate estimates that fall below P25 - 3 * IQR or above P75 + 3 * IQR.

Value

A tibble containing the number of simulation iterations, performance criteria estimate(s) and the associated MCSE.

Examples

```
calc_relative(data = t_res, estimates = est, true_param = true_param)
```

`calc_relative_var` *Calculate jack-knife Monte Carlo SE for variance estimators*

Description

Calculates relative bias, mean squared error (relative mse), and root mean squared error (relative rmse) of variance estimators. The function also calculates the associated jack-knife Monte Carlo standard errors.

Usage

```
calc_relative_var(
  data,
  estimates,
  var_estimates,
  criteria = c("relative bias", "relative mse", "relative rmse"),
  winz = Inf,
  var_winz = winz
)
```

Arguments

`data`	data frame or tibble containing the simulation results.
`estimates`	vector or name of column from `data` containing point estimates.
`var_estimates`	vector or name of column from `data` containing variance estimates for point estimator in `estimates`.
`criteria`	character or character vector indicating the performance criteria to be calculated, with possible options `"relative bias"`, `"relative mse"`, and `"relative rmse"`.
`winz`	numeric value for winsorization constant. If set to a finite value, estimates will be winsorized at the constant multiple of the inter-quartile range below the 25th percentile or above the 75th percentile of the distribution. For instance, setting `winz = 3` will truncate estimates that fall below P25 - 3 * IQR or above P75 + 3 * IQR.
`var_winz`	numeric value for winsorization constant for the variance estimates. If set to a finite value, variance estimates will be winsorized at the constant multiple of the inter-quartile range below the 25th percentile or above the 75th percentile of the distribution. For instance, setting `var_winz = 3` will truncate variance estimates that fall below P25 - 3 * IQR or above P75 + 3 * IQR. By default `var_winz` is set to the same constant as `winsorize`.

Value

A tibble containing the number of simulation iterations, performance criteria estimate(s) and the associated MCSE.

Examples

```
calc_relative_var(data = alpha_res, estimates = A, var_estimates = Var_A)
```

`create_skeleton` *Open a simulation skeleton*

Description

Creates and opens a .R file containing a skeleton for writing a Monte Carlo simulation study.

Usage

```
create_skeleton()
```

Examples

```
## Not run:
create_skeleton()

## End(Not run)
```

`evaluate_by_row` *Evaluate a simulation function on each row of a data frame or tibble*

Description

Evaluates a simulation function on each row of a data frame or tibble containing parameter values. Returns a single tibble with parameters and simulation results. The function uses `furrr::future_pmap`, which allows for easy parallelization.

Usage

```
evaluate_by_row(
  params,
  sim_function,
  ...,
  results_name = ".results",
  .progress = FALSE,
  .options = furrr::furrr_options(),
  system_time = TRUE
)
```

Arguments

`params`	data frame or tibble containing simulation parameter values. Each row should represent a separate set of parameter values.
`sim_function`	function to be evaluated, with argument names matching the variable names in `params`. The function must return a `data.frame`, `tibble`, or vector.
`...`	additional arguments passed to `sim_function`.
`results_name`	character string to set the name of the column storing the results of the simulation. Default is `".results"`.
`.progress`	A single logical. Should a progress bar be displayed? Only works with multisession, multicore, and multiprocess futures. Note that if a multicore/multisession future falls back to sequential, then a progress bar will not be displayed. **Warning:** The `.progress` argument will be deprecated and removed in a future version of furrr in favor of using the more robust progressr package.
`.options`	The `future` specific options to use with the workers. This must be the result from a call to `furrr_options()`.
`system_time`	logical indicating whether to print computation time. `TRUE` by default.

Value

A tibble containing parameter values and simulation results.

Examples

```
df <- data.frame(
  n = 3:5,
  lambda = seq(8, 16, 4)
)

evaluate_by_row(df, rpois)
```

---

extrapolate_coverage *Extrapolate coverage and width using sub-sampled bootstrap confidence intervals.*

Description

Given a set of bootstrap confidence intervals calculated across sub-samples with different numbers of replications, extrapolates confidence interval coverage and width of bootstrap confidence intervals to a specified (larger) number of bootstraps. The function also calculates the associated Monte Carlo standard errors. The confidence interval percentage is based on how you calculated the lower and upper bounds.

Usage

```
extrapolate_coverage(
  data,
  CI_subsamples,
  true_param,
  B_target = Inf,
  criteria = c("coverage", "width"),
  winz = Inf,
  nested = FALSE,
  format = "wide",
  width_trim = 0,
  cover_na_val = NA,
  width_na_val = NA
)
```

Arguments

data	data frame or tibble containing the simulation results.
CI_subsamples	list or name of column from data containing list of confidence intervals calculated based on sub-samples with different numbers of replications.
true_param	vector or name of column from data containing corresponding true parameters.
B_target	number of bootstrap replications to which the criteria should be extrapolated, with a default of B = Inf.
criteria	character or character vector indicating the performance criteria to be calculated, with possible options "coverage" and "width".
winz	numeric value for winsorization constant. If set to a finite value, estimates will be winsorized at the constant multiple of the inter-quartile range below the 25th percentile or above the 75th percentile of the distribution. For instance, setting winz = 3 will truncate estimates that fall below P25 - 3 * IQR or above P75 + 3 * IQR.
nested	logical value controlling the format of the output. If FALSE (the default), then the results will be returned as a data frame with rows for each distinct number of bootstraps. If TRUE, then the results will be returned as a data frame with a single row, with each performance criterion containing a nested data frame.
format	character string controlling the format of the output when CI_subsamples has results for more than one type of confidence interval. If "wide" (the default), then each performance criterion will have a separate column for each CI type. If "long", then each performance criterion will be a single variable, with separate rows for each CI type.
width_trim	numeric value specifying the trimming percentage to use when summarizing CI widths across replications from a single set of bootstraps, with a default of 0.0 (i.e., use the regular arithmetic mean).
cover_na_val	numeric value to use for calculating coverage if bootstrap CI end-points are missing. Default is NA.
width_na_val	numeric value to use for calculating width if bootstrap CI end-points are missing. Default is NA.

Value

A tibble containing the number of simulation iterations, performance criteria estimate(s) and the associated MCSE.

References

Boos DD, Zhang J (2000). "Monte Carlo evaluation of resampling-based hypothesis tests." *Journal of the American Statistical Association*, **95**(450), 486–492. doi:10.1080/01621459.2000.10474226.

Examples

```
dgp <- function(N, mu, nu) {
  mu + rt(N, df = nu)
}

estimator <- function(
   dat,
    B_vals = c(49,59,89,99),
    m = 4,
    trim = 0.1
) {


  # compute estimate and standard error
  N <- length(dat)
  est <- mean(dat, trim = trim)
  se <- sd(dat) / sqrt(N)

  # compute booties
  booties <- replicate(max(B_vals), {
    x <- sample(dat, size = N, replace = TRUE)
    data.frame(
      M = mean(x, trim = trim),
      SE = sd(x) / sqrt(N)
    )
  }, simplify = FALSE) |>
    dplyr::bind_rows()

  # confidence intervals for each B_vals
  CIs <- bootstrap_CIs(
    boot_est = booties$M,
    boot_se = booties$SE,
    est = est,
    se = se,
    CI_type = c("normal","basic","student","percentile"),
    B_vals = B_vals,
    reps = m,
    format = "wide-list"
  )

  res <- data.frame(
    est = est,
```

```
    se = se
  )
  res$CIs <- CIs

  res
}

#' build a simulation driver function
simulate_bootCIs <- bundle_sim(
  f_generate = dgp,
  f_analyze = estimator
)

boot_results <- simulate_bootCIs(
  reps = 50, N = 20, mu = 2, nu = 3,
  B_vals = seq(49, 199, 50),
)

extrapolate_coverage(
  data = boot_results,
  CI_subsamples = CIs,
  true_param = 2
)

extrapolate_coverage(
  data = boot_results,
  CI_subsamples = CIs,
  true_param = 2,
  B_target = 999,
  format = "long"
)
```

`extrapolate_rejection` *Extrapolate coverage and width using sub-sampled bootstrap confidence intervals.*

Description

Given a set of bootstrap confidence intervals calculated across sub-samples with different numbers of replications, extrapolates confidence interval coverage and width of bootstrap confidence intervals to a specified (larger) number of bootstraps. The function also calculates the associated Monte Carlo standard errors. The confidence interval percentage is based on how you calculated the lower and upper bounds.

Usage

```
extrapolate_rejection(
  data,
  pvalue_subsamples,
```

```
  B_target = Inf,
  alpha = 0.05,
  nested = FALSE,
  format = "wide"
)
```

Arguments

data	data frame or tibble containing the simulation results.
pvalue_subsamples	list or name of column from data containing list of confidence intervals calculated based on sub-samples with different numbers of replications.
B_target	number of bootstrap replications to which the criteria should be extrapolated, with a default of B = Inf.
alpha	scalar or vector indicating the nominal alpha level(s). Default value is set to the conventional .05.
nested	logical value controlling the format of the output. If FALSE (the default), then the results will be returned as a data frame with rows for each distinct number of bootstraps. If TRUE, then the results will be returned as a data frame with a single row, with each performance criterion containing a nested data frame.
format	character string controlling the format of the output when CI_subsamples has results for more than one type of confidence interval. If "wide" (the default), then each performance criterion will have a separate column for each CI type. If "long", then each performance criterion will be a single variable, with separate rows for each CI type.

Value

A tibble containing the number of simulation iterations, performance criteria estimate(s) and the associated MCSE.

References

Boos DD, Zhang J (2000). "Monte Carlo evaluation of resampling-based hypothesis tests." *Journal of the American Statistical Association*, **95**(450), 486–492. doi:10.1080/01621459.2000.10474226.

Examples

```
# function to generate data from two distinct populations
dgp <- function(N_A, N_B, shape_A, scale_A, shape_B, scale_B) {
  data.frame(
    group = rep(c("A","B"), c(N_A, N_B)),
      y = c(
        rgamma(N_A, shape = shape_A, scale = scale_A),
        rgamma(N_B, shape = shape_B, scale = scale_B)
      )
  )
}
```

```
# function to do a bootstrap t-test
estimator <- function(
    dat,
    B_vals = c(49,59,89,99), # number of booties to evaluate
    pval_reps = 4L
) {
  stat <- t.test(y ~ group, data = dat)$statistic

  # create bootstrap replications under the null of no difference
  boot_dat <- dat
  booties <- replicate(max(B_vals), {
    boot_dat$group <- sample(dat$group)
    t.test(y ~ group, data = boot_dat)$statistic
  })

  # calculate multiple bootstrap p-values using sub-sampling of replicates
  res <- data.frame(stat = stat)

  res$pvalue_subsamples <- bootstrap_pvals(
    boot_stat = booties,
    stat = stat,
    B_vals = B_vals,
    reps = pval_reps,
    enlist = TRUE
  )

  res
}

# create simulation driver
simulate_boot_pvals <- bundle_sim(
  f_generate = dgp,
  f_analyze = estimator
)

# replicate the bootstrap process
x <- simulate_boot_pvals(
  reps = 50L,
  N_A = 20, N_B = 25,
  shape_A = 7, scale_A = 2,
  shape_B = 4, scale_B = 3,
  B_vals = c(49, 99, 149, 199),
  pval_reps = 2L
)

extrapolate_rejection(
  data = x,
  pvalue_subsamples = pvalue_subsamples,
  B_target = 1999,
  alpha = c(.01, .05, .10)
)

extrapolate_rejection(
```

```
  data = x,
  pvalue_subsamples = pvalue_subsamples,
  B_target = Inf,
  alpha = c(.01, .05, .10),
  nested = TRUE
)
```

---

repeat_and_stack *Repeat an expression multiple times and (optionally) stack the results.*

Description

Repeat an expression (usually involving random number generation) multiple times. Optionally, organize the results into a `data.frame` that stacks the output from all replications of the expression.

Usage

```
repeat_and_stack(n, expr, stack = TRUE)
```

Arguments

`n`	Number of times to repeat the expression
`expr`	An expression to be evaluated.
`stack`	Logical value indicating whether to organize the results into a `data.frame`.

Value

If `stack = TRUE` (the default), the results of each evaluation of `expr` will be stacked together using `rbind`. If `stack = FALSE`, a list of length `n` with entries corresponding to the output of each replication of `expr`.

Examples

```
repeat_and_stack(n = 3, data.frame(x = rexp(2)))

repeat_and_stack(n = 3, data.frame(x = rexp(2)), stack = FALSE)
```

Tipton_Pusto *Results for Figure 2 of Tipton & Pustejovsky (2015)*

Description

A dataset containing simulation results comparing small sample correction methods for cluster robust variance estimation in meta-analysis.

Usage

```
Tipton_Pusto
```

Format

A tibble with 15,300 rows and 8 variables:

num_studies the number of studies included in the meta-analysis.

r correlation between outcomes.

Isq measure of heterogeneity of true effects.

contrast type of contrast that was tested.

test small sample method used.

q the number of parameters in the hypothesis test.

rej_rate the Type 1 error rate.

mcse the Monte Carlo standard error for the estimate of the Type 1 error rate.

Source

Tipton E, Pustejovsky JE (2015). "Small-sample adjustments for tests of moderators and model fit using robust variance estimation in meta-regression." *Journal of Educational and Behavioral Statistics*, **40**(6), 604–634. doi:10.3102/1076998615606099.

t_res *t-test simulation results*

Description

A dataset containing simulation results from a study that just runs a t-test.

Usage

```
t_res
```

Format

A tibble with 1,000 rows and 5 variables:

est estimate of the mean difference.

p_val p-value from the t-test.

lower_bound lower bound of the confidence interval.

upper_bound upper bound of the confidence interval.

true_param true mean difference used to generate the data.

---

welch_res *Welch t-test simulation results*

Description

A dataset containing simulation results from a study comparing Welch t-test to the conventional t-test.

Usage

```
welch_res
```

Format

A tibble with 16,000 rows and 11 variables:

n1 sample size for Group 1.

n2 sample size for Group 2.

mean_diff true difference in means of two groups used to generate the data.

iterations number of iterations.

seed seed used to generate data.

method indicates whether Welch or conventional t-test was used.

est estimate of the mean difference.

var variance of the estimate.

p_val p-value from the t-test.

lower_bound lower bound of the confidence interval.

upper_bound upper bound of the confidence interval.

Index