Package 'mcstatsim'

November 6, 2026

Type Package

Title Monte Carlo Statistical Simulation Tools Using a Functional Approach

Version 0.9.2

Description A lightweight toolkit for Monte Carlo simulation studies built around a higher-order, functional-style interface: the simulation is expressed as a function of its parameters and mapped over a grid of conditions by a single driver function, which handles iteration, parallelism, reproducible per-replication random-number streams, error capture and result assembly. Also provides ready-to-use performance measures with Monte Carlo standard errors, a summary layer, and diagnostic plots.

License AGPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Imports functionals

Suggests testthat (>= 3.0.0), ggplot2, knitr, rmarkdown

URL https://CRAN.R-project.org/package=mcstatsim

BugReports https://github.com/ielbadisy/mcstatsim/issues

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Imad El Badisy [aut, cre, cph]

Maintainer Imad El Badisy <elbadisyimad@gmail.com>

Repository CRAN

Date/Publication 2026-11-06 12:50:09 UTC

Contents

calc_bias . . . 2
calc_coverage . . . 3
calc_mse . . . 4
calc_rejection_rate . . . 5
calc_relative_bias . . . 6
calc_relative_mse . . . 7
calc_relative_rmse . . . 8
calc_rmse . . . 9
calc_variance . . . 9
calc_width . . . 10
check_estimates . . . 11
combine_df . . . 12
failure_summary . . . 13
mcpmap . . . 14
mcse_target . . . 15
plot_convergence . . . 15
plot_performance . . . 16
plot_zip . . . 17
runsim . . . 18
summarise_sim . . . 20

Index **23**

`calc_bias` *Calculate Bias and Bias Monte Carlo Standard Error*

Description

This function computes the bias and the Monte Carlo Standard Error (MCSE) of the bias for a set of estimates relative to a true parameter value. The bias is the difference between the mean of the estimates and the true parameter. The MCSE of the bias is calculated as the square root of the variance of the estimates divided by the number of estimates, providing a measure of the precision of the bias estimate.

Usage

```
calc_bias(estimates, true_param, digits = 4)
```

Arguments

`estimates`	A numeric vector of estimates from the simulation or sampling process.
`true_param`	The true parameter value that the estimates are intended to approximate.
`digits`	Number of decimal places to round the returned values to; 'NULL' returns them unrounded. Default 4.

Value

A list with two components: 'bias', the calculated bias of the estimates, and 'bias_mcse', the Monte Carlo Standard Error of the bias, indicating the uncertainty associated with the bias estimate.

Examples

```
estimates <- rnorm(100, mean = 50, sd = 10)
true_param <- 50
bias_info <- calc_bias(estimates, true_param)
print(bias_info)
```

`calc_coverage` *Calculate Coverage Probability and its Monte Carlo Standard Error*

Description

Computes the coverage probability of a confidence interval, defined as the proportion of times the true parameter value falls within the calculated lower and upper bounds across a set of simulations. Additionally, calculates the Monte Carlo Standard Error (MCSE) of the coverage probability to assess the uncertainty associated with this coverage estimate. This function is useful for evaluating the accuracy and reliability of confidence intervals generated by statistical models or estimation procedures.

Usage

```
calc_coverage(lower_bound, upper_bound, true_param, digits = 4)
```

Arguments

`lower_bound`	A numeric vector of lower bounds of confidence intervals.
`upper_bound`	A numeric vector of upper bounds of confidence intervals, corresponding to 'lower_bound'.
`true_param`	The true parameter value that the confidence intervals are intended to estimate.
`digits`	Number of decimal places to round the returned values to; 'NULL' returns them unrounded. Default 4.

Value

A list with two components: 'coverage', the calculated coverage probability of the confidence intervals, and 'coverage_mcse', the Monte Carlo Standard Error of the coverage. This MCSE provides a measure of the precision of the coverage probability estimate.

Examples

```
set.seed(123) # For reproducibility
estimates <- rnorm(100, mean = 50, sd = 10)
ci_lower <- estimates - 1.96 * 10
ci_upper <- estimates + 1.96 * 10
true_param <- 50
coverage_info <- calc_coverage(ci_lower, ci_upper, true_param)
print(coverage_info)
```

`calc_mse` *Calculate Mean Squared Error and its Monte Carlo Standard Error*

Description

Computes the Mean Squared Error (MSE) of a set of estimates relative to a true parameter value, along with the Monte Carlo Standard Error (MCSE) for the MSE. The MCSE takes into account the variance, skewness, and kurtosis of the estimates to provide a more accurate measure of uncertainty. This function is useful for assessing the accuracy of simulation or estimation methods by comparing the squared deviations of estimated values from a known parameter.

Usage

```
calc_mse(estimates, true_param, digits = 4)
```

Arguments

`estimates`	A numeric vector of estimates from a simulation or sampling process.
`true_param`	The true parameter value that the estimates are intended to approximate.
`digits`	Number of decimal places to round the returned values to; 'NULL' returns them unrounded. Default 4.

Value

A list with two components: 'mse', the calculated Mean Squared Error of the estimates, and 'mse_mcse', the Monte Carlo Standard Error of the MSE, offering insight into the reliability of the MSE calculation.

Examples

```
estimates <- rnorm(100, mean = 50, sd = 10)
true_param <- 50
mse_info <- calc_mse(estimates, true_param)
print(mse_info)
```

`calc_rejection_rate` *Calculate Rejection Rate and its Monte Carlo Standard Error*

Description

Computes the rejection rate of hypotheses tests based on a vector of p-values and a specified significance level (alpha). The rejection rate is the proportion of p-values that are lower than alpha, indicating significant results. Additionally, the function calculates the Monte Carlo Standard Error (MCSE) for the rejection rate, which quantifies the uncertainty associated with the estimated rejection rate. This function is useful for assessing the overall type I error rate or the power of a statistical test across multiple simulations or experimental replications.

Usage

```
calc_rejection_rate(p_values, alpha = 0.05, digits = 4)
```

Arguments

`p_values`	A numeric vector of p-values from multiple hypothesis tests.
`alpha`	The significance level used to determine if a p-value indicates a significant result. Default is 0.05.
`digits`	Number of decimal places to round the returned values to; 'NULL' returns them unrounded. Default 4.

Value

A list with two components: 'rejection_rate', the proportion of tests that resulted in rejection of the null hypothesis, and 'rejection_rate_mcse', the Monte Carlo Standard Error of the rejection rate, providing an estimate of its variability.

Examples

```
set.seed(123) # For reproducibility
p_values <- runif(100, min = 0, max = 1) # Simulated p-values
rejection_info <- calc_rejection_rate(p_values)
print(rejection_info)
```

`calc_relative_bias` *Calculate Relative Bias and its Monte Carlo Standard Error*

Description

Computes the relative bias of a set of estimates with respect to a true parameter value, along with the Monte Carlo Standard Error (MCSE) of the relative bias. Relative bias is the ratio of the mean of the estimates to the true parameter, providing a scale-independent measure of bias. This function is particularly useful for evaluating the accuracy of estimates in situations where the magnitude of the true parameter is crucial to the interpretation of bias. The function gracefully handles cases where the true parameter is zero by returning 'NA' for both relative bias and its MCSE, avoiding division by zero errors.

Usage

```
calc_relative_bias(estimates, true_param, digits = 4)
```

Arguments

`estimates`	A numeric vector of estimates from a simulation or sampling process.
`true_param`	The true parameter value that the estimates are intended to approximate. Note that 'true_param' must not be zero, as relative bias calculation involves division by the true parameter value.
`digits`	Number of decimal places to round the returned values to; 'NULL' returns them unrounded. Default 4.

Value

A list with two components: 'rel_bias', the calculated relative bias of the estimates, and 'rel_bias_mcse', the Monte Carlo Standard Error of the relative bias. If 'true_param' is zero, both 'rel_bias' and 'rel_bias_mcse' will be 'NA'.

Examples

```
estimates <- rnorm(100, mean = 50, sd = 10)
true_param <- 50 # Non-zero true parameter
relative_bias_info <- calc_relative_bias(estimates, true_param)
print(relative_bias_info)
```

`calc_relative_mse` *Calculate Relative Mean Squared Error and its Monte Carlo Standard Error*

Description

Computes the Relative Mean Squared Error (Relative MSE) of a set of estimates with respect to a true parameter value, along with the Monte Carlo Standard Error (MCSE) of the Relative MSE. The Relative MSE is a normalized measure of error that scales the Mean Squared Error (MSE) by the square of the true parameter value, making it particularly useful for comparing the accuracy of estimates across different scales. The function also calculates the MCSE for the Relative MSE, taking into account the variance, skewness, and kurtosis of the estimates to provide a measure of uncertainty. The function returns 'NA' for both Relative MSE and its MCSE if the true parameter is zero, to avoid division by zero.

Usage

```
calc_relative_mse(estimates, true_param, digits = 4)
```

Arguments

`estimates`	A numeric vector of estimates from a simulation or sampling process.
`true_param`	The true parameter value that the estimates are intended to approximate. Note that 'true_param' must not be zero, as the calculation involves division by the true parameter value.
`digits`	Number of decimal places to round the returned values to; 'NULL' returns them unrounded. Default 4.

Value

A list with two components: 'rel_mse', the calculated Relative Mean Squared Error of the estimates, and 'rel_mse_mcse', the Monte Carlo Standard Error of the Relative MSE. If 'true_param' is zero, both 'rel_mse' and 'rel_mse_mcse' will be 'NA'.

Examples

```
estimates <- rnorm(100, mean = 50, sd = 10)
true_param <- 50 # Non-zero true parameter
relative_mse_info <- calc_relative_mse(estimates, true_param)
print(relative_mse_info)
```

`calc_relative_rmse` *Calculate Relative Root Mean Squared Error and its Monte Carlo Standard Error*

Description

Computes the Relative Root Mean Squared Error (Relative RMSE) of a set of estimates with respect to a true parameter value. The Relative RMSE is the square root of the Relative Mean Squared Error (see [calc_relative_mse()]), a scale-independent measure of error that facilitates comparisons across different scales of the true parameter. This function is especially useful for evaluating the accuracy of estimates when the magnitude of the true parameter varies significantly across different scenarios. The function gracefully handles cases where the true parameter is zero by returning 'NA' for both Relative RMSE and its MCSE, to avoid division by zero.

Usage

```
calc_relative_rmse(estimates, true_param, digits = 4)
```

Arguments

`estimates`	A numeric vector of estimates from a simulation or sampling process.
`true_param`	The true parameter value that the estimates are intended to approximate. Note that 'true_param' must not be zero, as the calculation involves division by the true parameter value.
`digits`	Number of decimal places to round the returned values to; 'NULL' returns them unrounded. Default 4.

Value

A list with two components: 'rel_rmse', the calculated Relative Root Mean Squared Error of the estimates, and 'rel_rmse_mcse', the Monte Carlo Standard Error of the Relative RMSE, obtained from the MCSE of the Relative MSE by the delta method ('rel_mse_mcse / (2 * rel_rmse)'). Both are 'NA' when 'true_param' is zero, and 'rel_rmse_mcse' is 'NA' when 'rel_rmse' is zero.

Examples

```
estimates <- rnorm(100, mean = 50, sd = 10)
true_param <- 50 # Non-zero true parameter
relative_rmse_info <- calc_relative_rmse(estimates, true_param)
print(relative_rmse_info)
```

calc_rmse — *Calculate Root Mean Squared Error and its Monte Carlo Standard Error*

Description

Computes the Root Mean Squared Error (RMSE) of a set of estimates relative to a true parameter value, along with the Monte Carlo Standard Error (MCSE) for the RMSE. The RMSE is a measure of the accuracy of the estimates, representing the square root of the average squared differences between the estimated values and the true parameter. The MCSE for the RMSE is calculated using jackknife estimates, providing an assessment of the uncertainty associated with the RMSE value.

Usage

```
calc_rmse(estimates, true_param, digits = 4)
```

Arguments

estimates	A numeric vector of estimates from a simulation or sampling process.
true_param	The true parameter value that the estimates are intended to approximate.
digits	Number of decimal places to round the returned values to; 'NULL' returns them unrounded. Default 4.

Value

A list with two components: 'rmse', the calculated Root Mean Squared Error of the estimates, and 'rmse_mcse', the Monte Carlo Standard Error of the RMSE. This MCSE is derived from jackknife estimates, offering insight into the reliability of the RMSE calculation.

Examples

```
estimates <- rnorm(100, mean = 50, sd = 10)
true_param <- 50
rmse_info <- calc_rmse(estimates, true_param)
print(rmse_info)
```

calc_variance — *Calculate Variance and its Monte Carlo Standard Error*

Description

Computes the sample variance of a set of estimates and the Monte Carlo Standard Error (MCSE) for the variance. The MCSE is adjusted by the sample kurtosis to account for the shape of the distribution of the estimates. This function is particularly useful in simulation studies where understanding the variability of an estimator and the precision of this variability estimate is crucial.

Usage

```
calc_variance(estimates, digits = 4)
```

Arguments

estimates	A numeric vector of estimates from a simulation or sampling process.
digits	Number of decimal places to round the returned values to; 'NULL' returns them unrounded. Default 4.

Value

A list containing two elements: 'variance', the calculated sample variance of the estimates, and 'variance_mcse', the Monte Carlo Standard Error of the variance. The MCSE provides a measure of the uncertainty associated with the variance estimate, adjusted for kurtosis.

Examples

```
estimates <- rnorm(100, mean = 50, sd = 10)
variance_info <- calc_variance(estimates)
print(variance_info)
```

---

calc_width *Calculate Average Width of Confidence Intervals and its Monte Carlo Standard Error*

Description

This function computes the average width of a set of confidence intervals, represented by their lower and upper bounds, along with the Monte Carlo Standard Error (MCSE) of this average width. The average width provides insight into the precision of the estimation process, with narrower intervals typically indicating more precise estimates. The MCSE of the width offers a measure of the uncertainty associated with the average width calculation, useful for assessing the variability of interval estimates across simulations or bootstrap samples.

Usage

```
calc_width(lower_bound, upper_bound, digits = 4)
```

Arguments

lower_bound	A numeric vector of lower bounds of confidence intervals.
upper_bound	A numeric vector of upper bounds of confidence intervals, corresponding to 'lower_bound'.
digits	Number of decimal places to round the returned values to; 'NULL' returns them unrounded. Default 4.

Value

A list with two components: ‘width‘, the average width of the confidence intervals, and ‘width_mcse‘, the Monte Carlo Standard Error of the average width. This MCSE provides a quantification of the uncertainty in the average width estimate.

Examples

```
set.seed(123) # For reproducibility
estimates <- rnorm(100, mean = 50, sd = 10)
ci_lower <- estimates - 1.96 * 10
ci_upper <- estimates + 1.96 * 10
width_info <- calc_width(ci_lower, ci_upper)
print(width_info)
```

---

check_estimates *Flag Non-Finite or Degenerate Estimates in Simulation Output*

Description

Scans the numeric columns of a replicate-level [runsim()] result for values that will quietly distort performance measures: ‘NA‘, ‘NaN‘, and ‘Inf‘. Reports the count and proportion per condition, so a handful of blown-up replications do not pass unnoticed into a summary table.

Usage

```
check_estimates(data, cols = NULL, by = NULL, digits = 4)
```

Arguments

data	A dataframe of replicate-level results, typically from [runsim()].
cols	Character vector of columns to check. Defaults to every numeric column except ‘ID‘ and the grouping columns in ‘by‘.
by	Character vector of column names identifying a simulation condition. If ‘NULL‘ (default) the whole dataframe is treated as one group.
digits	Number of decimal places to round ‘prop_bad‘ to; ‘NULL‘ leaves it unrounded. Default 4.

Value

A dataframe with one row per (condition, column) that has at least one non-finite value: the ‘by‘ columns, ‘column‘, ‘n_bad‘, ‘n_total‘, and ‘prop_bad‘, ordered by decreasing ‘prop_bad‘. A zero-row dataframe (with those columns) means every checked value was finite.

Examples

```
sim_func <- function(mu) {
  x <- rnorm(10, mu)
  data.frame(est = mean(x), ratio = mu / (mu - 1))  # Inf at mu == 1
}
grid <- expand.grid(mu = c(0, 1, 2))
res <- runsim(20, grid, sim_func, show_progress = FALSE, num_cores = 1)
check_estimates(res, by = "mu")
```

`combine_df` *Combine Dataframes in a Nested List*

Description

This function combines dataframes that are nested within a list of lists into a single dataframe.

Usage

```
combine_df(nested_list)
```

Arguments

`nested_list`	A list of lists, where each sublist contains dataframes to be combined.

Details

The function first checks if the input is a non-empty list of lists containing dataframes. It then iterates through each sublist, combining the dataframes and adding an ID column to indicate their source. Finally, all combined dataframes are row-bound into a single dataframe.

Value

A combined dataframe where each original dataframe is augmented with an ID column indicating its source list.

Examples

```
df1 <- data.frame(a = 1:3, b = 4:6)
df2 <- data.frame(a = 7:9, b = 10:12)
nested_list <- list(list(df1, df2), list(df1, df2))
combine_df(nested_list)
```

`failure_summary` *Summarise Failed Jobs from a Simulation Run*

Description

Aggregates the per-job failure log attached to a [runsim()] result (when it was called with ‘on_error = "warn"‘ or ‘on_error = "omit"‘) into one row per simulation condition, with the number of failed replications, the failure rate, and an example error message.

Usage

```
failure_summary(x, n = NULL, digits = 4)
```

Arguments

`x`	A dataframe returned by [runsim()].
`n`	Optional total number of replications per condition, used to turn failure counts into rates. If ‘NULL‘ (default) it is taken as ‘max(x$ID)‘.
`digits`	Number of decimal places to round ‘failure_rate‘ to; ‘NULL‘ leaves it unrounded. Default 4.

Value

A dataframe with the grid columns that describe each condition, plus ‘n_failed‘, ‘failure_rate‘, and ‘example_error‘. If ‘x‘ has no ‘"errors"‘ attribute (no job failed, or ‘on_error = "stop"‘ was used) a zero-row dataframe with those columns is returned.

Examples

```
sim_func <- function(mu) {
  if (mu < 0) stop("mu must be non-negative")
  data.frame(est = mean(rnorm(10, mu)))
}
params <- expand.grid(mu = c(-1, 0, 1))
res <- runsim(5, params, sim_func, show_progress = FALSE, num_cores = 1,
              on_error = "omit")
failure_summary(res)
```

mcpmap *Parallel Map Function using functionals::fmapn*

Description

This function applies a given function over a list of parameters in parallel using multiple cores.

Usage

```
mcpmap(
  lists,
  func,
  num_cores = parallel::detectCores() - 1,
  show_progress = TRUE
)
```

Arguments

lists	A list of lists containing the parameters for the function.
func	The function to be applied.
num_cores	The number of cores to use for parallel execution. Default is one less than the total number of available cores.
show_progress	Logical indicating whether to display the progress bar. Default is TRUE.

Details

The function ensures that all elements in the list have the same length and uses ‘functionals::fmapn‘ for parallel processing. It sets the number of cores based on the operating system and then applies the function in parallel.

Value

A list of results from applying the function over the parameters.

Examples

```
params <- list(a = 1:3, b = 4:6)
mcpmap(params, function(a, b) a + b, num_cores = 2)
```

mcse_target *Replications Needed to Reach a Target Monte Carlo Standard Error*

Description

Monte Carlo standard errors shrink like ‘1 / sqrt(n_sim)‘. Given the MCSE actually achieved with ‘n_current‘ replications, this returns the number of replications a future run would need to bring the MCSE down to ‘target‘.

Usage

```
mcse_target(mcse_current, n_current, target)
```

Arguments

mcse_current	The Monte Carlo standard error observed so far. May be a vector.
n_current	The number of replications that produced ‘mcse_current‘. A single positive number, or a vector aligned with ‘mcse_current‘.
target	The desired Monte Carlo standard error. A single positive number, or a vector aligned with ‘mcse_current‘.

Value

An integer vector (‘ceiling‘) of the replications required to reach ‘target‘. Entries are ‘NA‘ where ‘mcse_current‘ is ‘NA‘ or not positive.

Examples

```
# A coverage estimate with MCSE 0.011 from 1000 reps; how many reps for MCSE 0.005?
mcse_target(0.011, 1000, 0.005)

# Vectorised over a summarise_sim() column
# mcse_target(tab$coverage_mcse, tab$n_sim, target = 0.005)
```

plot_convergence *Convergence Trace of a Monte Carlo Estimate*

Description

Plots the running mean of a quantity against the number of replications, with a Monte Carlo standard error band, one panel per simulation condition. It shows whether a run used enough replications for the quantity of interest to settle.

Usage

```
plot_convergence(data, var, by = NULL, se_mult = 1)
```

Arguments

data	A replicate-level dataframe, typically from [runsim()]. Replication order is taken from the ‘ID‘ column when present, otherwise from row order.
var	Name of the numeric column to trace.
by	Character vector of condition columns; one panel is drawn per combination. ‘NULL‘ (default) traces the whole dataframe as one panel.
se_mult	Half-width of the band in Monte Carlo standard errors (default 1).

Value

A ‘ggplot‘ object.

Examples

```
## Not run:
sim_func <- function(mu) data.frame(est = mean(rnorm(30, mu)))
res <- runsim(500, expand.grid(mu = c(0, 2)), sim_func,
              show_progress = FALSE, num_cores = 1, seed = 1)
plot_convergence(res, "est", by = "mu")

## End(Not run)
```

---

plot_performance *Point-Range Plot of a Performance Measure Across Conditions*

Description

Takes a summary table from [summarise_sim()] and plots one performance measure per condition as a point with a Monte Carlo error bar (‘estimate +/- se_mult * MCSE‘), the standard "does the method work across the design" picture.

Usage

```
plot_performance(
  summary,
  measure,
  x,
  colour = NULL,
  facet = NULL,
  se_mult = 2,
  ref = NULL
)
```

Arguments

summary	A dataframe from [summarise_sim()].
measure	Name of the estimate column to plot (e.g. ‘"coverage"‘). Its Monte Carlo standard error is taken from ‘paste0(measure, "_mcse")‘ when that column exists.
x	Name of the column to place on the x axis (a design factor).
colour	Optional column mapped to colour.
facet	Optional column to facet by.
se_mult	Half-height of the error bar in Monte Carlo standard errors (default 2).
ref	Optional numeric reference line (e.g. ‘0.95‘ for coverage, ‘0‘ for bias).

Value

A ‘ggplot‘ object.

Examples

```
## Not run:
tab <- summarise_sim(res, by = c("mu", "n_obs"),
                     measures = list(coverage = ~ calc_coverage(lo, hi, true_val[1])))
plot_performance(tab, "coverage", x = "n_obs", colour = "mu", ref = 0.95)

## End(Not run)
```

---

plot_zip *Zip Plot of Confidence-Interval Coverage*

Description

Draws the zip plot of Morris, White and Crowther (2019): within each condition the confidence intervals are sorted by how far their implied z-statistic sits from the target value, stacked vertically by that fractional centile, and coloured by whether they cover. A well-calibrated procedure fills the panel to the nominal line with the covering colour and shows the non-covering intervals only in the top tail.

Usage

```
plot_zip(data, lower, upper, true, by = NULL, conf_level = 0.95)
```

Arguments

data	A replicate-level dataframe, typically from [runsim()].
lower, upper, true	Column names of the interval bounds and the target value.
by	Character vector of condition columns; one panel per combination.
conf_level	Nominal confidence level the intervals were built at (default 0.95).

Value

A ‘ggplot‘ object.

References

Morris TP, White IR, Crowther MJ (2019). Using simulation studies to evaluate statistical methods. *Statistics in Medicine*, 38(11), 2074-2102.

Examples

```
## Not run:
sim_func <- function(mu) {
  x <- rnorm(25, mu); est <- mean(x); se <- sd(x) / 5
  data.frame(lo = est - 1.96 * se, hi = est + 1.96 * se, mu = mu)
}
res <- runsim(300, expand.grid(mu = c(0, 1)), sim_func,
              show_progress = FALSE, num_cores = 1, seed = 1)
plot_zip(res, "lo", "hi", "mu", by = "mu")

## End(Not run)
```

---

runsim *Run Monte Carlo Simulations in Parallel*

Description

This function executes a series of Monte Carlo simulations in parallel, providing detailed progress updates.

Usage

```
runsim(
  n,
  grid_params,
  sim_func,
  show_progress = TRUE,
  num_cores = parallel::detectCores() - 1,
  seed = NULL,
  on_error = c("stop", "warn", "omit"),
  save_path = NULL,
  checkpoint_every = 1L
)
```

Arguments

n	The number of times the simulation function should be executed for each set of parameters. Must be a positive integer.

`grid_params` A dataframe where each row corresponds to a unique combination of parameters for the simulation. Typically generated using ‘expand.grid‘.

`sim_func` The simulation function to be applied. This function should accept parameters corresponding to a row in ‘grid_params‘ and return a dataframe or a list that can be row-bound.

`show_progress` Logical indicating whether to display progress messages during the execution of the simulations.

`num_cores` The number of cores to use for parallel execution. The default is one less than the total number of cores available on the system.

`seed` Optional single number. When supplied, every job is assigned its own independent "L'Ecuyer-CMRG" random-number stream derived from ‘seed‘, so the whole run is reproducible and the result does not depend on ‘num_cores‘ or on the order in which jobs finish. When ‘NULL‘ (default) the session RNG is left untouched.

`on_error` How to handle a job in which ‘sim_func‘ throws an error. ‘"stop"‘ (default) aborts the whole run, reproducing the behaviour of earlier versions. ‘"warn"‘ drops the failed jobs, keeps the rest, and emits a warning. ‘"omit"‘ drops the failed jobs silently. With ‘"warn"‘ or ‘"omit"‘, details of every failed job are stored in ‘attr(x, "errors")‘.

`save_path` Optional path to an ‘.rds‘ file used as a checkpoint. When set, ‘runsim()‘ runs the replications in chunks and rewrites this file after each chunk, so a run that is interrupted (crash, time limit, ‘Ctrl-C‘) can be resumed simply by calling ‘runsim()‘ again with the same ‘save_path‘ and matching ‘n‘ / ‘grid_params‘ / ‘seed‘: completed replications are loaded from disk and skipped. When ‘NULL‘ (default) nothing is written and all jobs run in a single pass.

`checkpoint_every` Positive integer: how many replications to run between checkpoint writes when ‘save_path‘ is set. Larger values mean less I/O but more lost work if the run is interrupted. Ignored when ‘save_path‘ is ‘NULL‘.

Details

The function first validates the input parameters. It then builds a single flat list of ‘n * nrow(grid_params)‘ jobs (replication-major order: all conditions for replication 1, then all conditions for replication 2, and so on) and dispatches it in one load-balanced parallel pass via [mcpmap()]. Results are row-bound into a single dataframe, the condition’s ‘grid_params‘ columns are prepended (unless ‘sim_func‘ already returned a column of that name), and the replication index is tagged in the ‘ID‘ column.

If ‘seed‘ is supplied, ‘n * nrow(grid_params)‘ independent RNG streams are generated up front with [parallel::nextRNGStream()] and handed to the jobs by position. Because job positions are fixed (replication-major), a given ‘(replication, condition)‘ cell always draws from the same stream regardless of core count, so runs are bit-for-bit reproducible. Do not call ‘set.seed()‘ inside ‘sim_func‘ when using ‘seed‘, as that would defeat the per-stream design.

Each job is wrapped so that an error in ‘sim_func‘ is caught rather than lost in a worker process, and any warnings it emits are recorded. See ‘on_error‘ for how failures are handled.

When ‘save_path‘ is set the replications are run in chunks of ‘checkpoint_every‘ and the checkpoint file is rewritten after each chunk. Per-job RNG streams are keyed by position, so a resumed run produces exactly the same results as an uninterrupted one.

Value

A combined dataframe of all simulation results. Every ‘grid_params‘ column that ‘sim_func‘ did not itself return is prepended, so each row identifies its own condition, and an ‘ID‘ column gives the replication index (1 to ‘n‘). When ‘seed‘ is supplied it is stored in ‘attr(x, "seed")‘. Any warnings emitted by ‘sim_func‘ are collected in ‘attr(x, "warnings")‘, and, when ‘on_error‘ is not ‘"stop"‘, failed jobs are described in ‘attr(x, "errors")‘ (see [failure_summary()]).

Examples

```
## Not run:
library(mcstatsim)

# Define a simple simulation function
sim_function <- function(a, b) {
  Sys.sleep(0.1)  # Simulate a time-consuming process
  return(data.frame(result = a + b))
}

# Generate a grid of parameters
params <- expand.grid(a = 1:3, b = 4:6)

# Run simulations
results <- runsim(n = 1, grid_params = params, sim_func = sim_function)
print(results)

## End(Not run)
```

---

summarise_sim *Summarise Replicate-Level Simulation Output into a Performance Table*

Description

Collapses the raw, replicate-level dataframe returned by [runsim()] into one row per simulation condition, evaluating one or more performance measures (bias, coverage, rejection rate, ...) per condition. This is the bridge between ‘runsim()‘ output and the ‘calc_*‘ functions in this package.

Usage

```
summarise_sim(data, by = NULL, measures, digits = 4)
```

Arguments

data	A dataframe of replicate-level results, typically from [runsim()]: one row per replication (per condition), with the condition columns, the quantities returned by ‘sim_func‘, and usually an ‘ID‘ column.
by	Character vector of column names identifying a simulation condition. Rows sharing the same values in these columns form one group. If ‘NULL‘ (default) the whole dataframe is treated as a single group.
measures	A named list of performance measures to evaluate per group. Each element is either • a one-sided formula whose right-hand side is evaluated with the group’s columns in scope, e.g. ‘~ calc_bias(est, true_val[1])‘; or • a function taking the group dataframe and returning a named numeric vector or a list of scalars, e.g. ‘function(d) calc_coverage(dlo, dhi, d$true_val[1])‘. Columns hold one value per replicate, so a quantity that is constant across a condition (a true parameter value) must be indexed, e.g. ‘true_val[1]‘. The element names are labels only. A measure that returns a length-one value becomes a column named after the measure; a measure that returns a named vector of length > 1 (as the ‘calc_*‘ functions do) contributes one column per name. Column-name collisions across measures are an error.
digits	Number of decimal places to round the performance-measure columns to (the ‘by‘ columns and ‘n_sim‘ are left alone); ‘NULL‘ returns them unrounded. Default 4. Note that measures built from the ‘calc_*‘ functions are already rounded to those functions’ own ‘digits‘ (also 4 by default); pass ‘digits = NULL‘ inside such a measure, e.g. ‘~ calc_bias(est, true_val[1], digits = NULL)‘, if you need more precision than that.

Value

A dataframe with one row per group: the ‘by‘ columns, then ‘n_sim‘ (the number of distinct ‘ID‘ values in the group, or the row count if there is no ‘ID‘ column), then one column per performance-measure output.

Examples

```
sim_func <- function(mu, n_obs) {
  x <- rnorm(n_obs, mean = mu)
  est <- mean(x)
  se <- sd(x) / sqrt(n_obs)
  data.frame(est = est, lo = est - 1.96 * se, hi = est + 1.96 * se, true_val = mu)
}
grid <- expand.grid(mu = c(0, 1), n_obs = c(20, 50))
res <- runsim(100, grid, sim_func, show_progress = FALSE, num_cores = 1, seed = 1)

summarise_sim(
  res,
  by = c("mu", "n_obs"),
  measures = list(
    bias     = ~ calc_bias(est, true_val[1]),
```

```
    coverage = ~ calc_coverage(lo, hi, true_val[1])
  )
)
```

Index

calc_bias, 2
calc_coverage, 3
calc_mse, 4
calc_rejection_rate, 5
calc_relative_bias, 6
calc_relative_mse, 7
calc_relative_rmse, 8
calc_rmse, 9
calc_variance, 9
calc_width, 10
check_estimates, 11
combine_df, 12

failure_summary, 13

mcpmap, 14
mcse_target, 15

plot_convergence, 15
plot_performance, 16
plot_zip, 17

runsim, 18

summarise_sim, 20