Package 'mcstatsim'

February 12, 2024

Type Package

Title Monte Carlo Simulation Tools Using a Functional Approach

Version 0.1.0

Author Author

Maintainer Author <authour@mail.com>

Description The mcstatsim package offers an efficient, functional programming-based approach for statistical simulations, centralizing the process in a single higher-order function for better manageability. Besides, it includes ready-to-use functions for well-known simulation targets, enhancing its utility. The core `runsim` function processes simulation parameters via expanded grid, mapping them to the simulation function.
Replication management is performed through the `replicater()` function, which estimates the remaining time (ETA) with an option to hide progress via `show_progress = FALSE`.
Outputs are deliberately structured as dataframe to simplify analysis and visualization, addressing the limitations of list outputs in data manipulation.

License AGPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

R topics documented:

`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)
```

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.

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)
```

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.

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)
```

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.

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)
```

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.

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)
```

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.

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)
```

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.

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 derived from the Relative Mean Squared Error (MSE), providing 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. The MCSE for the Relative RMSE is not directly computed in this function and is marked as a placeholder for future implementation.

Usage

```
calc_relative_rmse(estimates, true_param)
```

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.

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. The MCSE is currently not calculated and returned as 'NA'. This is a placeholder for future implementation.

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)
```

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.

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)
```

Arguments

estimates	A numeric vector of estimates from a simulation or sampling process.

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)
```

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‘.

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)
```

---

combine_df *Combine nested lists of dataframes into a single dataframe*

Description

This function takes a nested list of data frames (a list of lists, where each inner list contains data frames) and combines them into a single data frame. Each data frame within the same sublist is combined row-wise, and an ID column is added to identify the source sublist. The function ensures that all elements of the input are proper lists containing data frames, and it stops with an error message if the input does not meet these criteria.

Usage

```
combine_df(nested_list)
```

Arguments

nested_list	A non-empty list of lists, where each inner list contains one or more data frames. It is expected that all data frames within the same sublist can be row-bound.

Value

A single data frame that is the row-wise combination of all input data frames, with an additional column ‘ID‘ indicating the originating sublist.

Examples

```
df1 <- data.frame(a = 1:3, b = letters[1:3])
df2 <- data.frame(a = 4:6, b = letters[4:6])
df3 <- data.frame(a = 7:9, b = letters[7:9])
df4 <- data.frame(a = 10:12, b = letters[10:12])

nested_list <- list(list(df1, df2), list(df3, df4))
combined_df <- combine_df(nested_list)
print(combined_df)
```

mcpmap *Parallel map over a list using multiple cores*

Description

Applies a function in parallel over elements of a list of elements with the same length with support for multiple cores. The function uses ‘parallel::mcmapply‘ under the hood to perform the parallel computation, automatically determining the number of cores to use if not specified.

Usage

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

Arguments

lists	A list of lists containing the arguments to pass to the function ‘func‘. Each inner list should correspond to an argument of ‘func‘, and all lists must be of the same length.
func	The function to apply to each set of arguments contained in ‘lists‘.
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 to leave resources for other processes.

Value

The result of applying ‘func‘ to the elements of ‘lists‘ in parallel. The return value is a list of the same length as the input lists, where each element is the result of applying ‘func‘ to the corresponding elements of the input lists.

Examples

```
# Define a function to apply
sum_func <- function(x, y) x + y

# Create a list of arguments
args_list <- list(c(1,2,3), c(4,5,6))

# Apply the function in parallel
results <- mcpmap(args_list, sum_func, num_cores = 2)
print(results)
```

replicater *Replicate expressions with progress reporting*

Description

Executes an R expression ‘n‘ times in a loop, optionally displaying progress and estimated time until completion. This function is particularly useful for simulations where the progress of repetitive tasks is beneficial for monitoring execution time and estimating completion. Progress information includes the percentage completed and an estimated time of arrival (ETA) calculated based on the average time per iteration.

Usage

```
replicater(n, expr, show_progress = TRUE)
```

Arguments

n	An integer specifying the number of times the expression should be replicated.
expr	The expression to be evaluated. This can be any valid R expression. Note that the expression is evaluated in the environment from which ‘replicater‘ is called.
show_progress	Logical, indicating whether to show progress and ETA information. Defaults to ‘TRUE‘.

Value

A list of length ‘n‘ containing the results of each evaluation of ‘expr‘.

Examples

```
# Simple example without progress (for quick execution)
results <- replicater(5, rnorm(1), show_progress = FALSE)

# Example with progress - might take longer due to Sys.sleep()
results_with_delay <- replicater(10, {Sys.sleep(0.5); rnorm(1)}, show_progress = TRUE)
```

runsim *Run simulations in parallel*

Description

Executes a series of simulations in parallel, based on a set of parameters specified in a dataframe. This function is designed to facilitate large-scale simulation studies by leveraging multiple cores to distribute the computational load.

Usage

```
runsim(
  n,
  grid_params,
  sim_func,
  show_progress = TRUE,
  num_cores = parallel::detectCores() - 1
)
```

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 the ‘expand.grid()‘ function.
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 to leave resources for other processes.

Details

This function first validates the input parameters, ensuring that ‘n‘ is a positive integer, ‘grid_params‘ is a dataframe, and ‘sim_func‘ is a valid function. It then uses parallel processing to apply ‘sim_func‘ to each combination of parameters specified in ‘grid_params‘, repeating each simulation ‘n‘ times. The results are combined into a single dataframe, which is returned to the caller. The parallel execution mode utilized is multicore, through the ‘parallel‘ package.

Value

A dataframe combining the results of all simulations, with an additional column to identify the set of parameters (from ‘grid_params‘) used for each simulation.

Note

The multicore parallel mode is only supported on macOS and Linux-based operating systems. Users on other operating systems, such as Windows, may not experience the same parallel processing capabilities and should consider alternative methods for parallel execution.

See Also

mcmapply for the underlying parallel apply mechanism.

Examples

```
# Define a simple simulation function
sim_function <- function(a, b) {
  Sys.sleep(0.1) # Simulating 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 = 2, grid_params = params, sim_func = sim_function, num_cores = 2)
print(results)
```

Index