Package 'parSim'

July 17, 2026

Title Parallel Simulator

Version 0.4.0

Description Perform flexible simulation studies using one or multiple computer cores. The package is set up to be usable on high-performance clusters in addition to being run locally (i.e., see the package vignettes for more information).

License GPL-2

URL https://github.com/SachaEpskamp/parSim

BugReports https://github.com/SachaEpskamp/parSim/issues

Imports data.table, dplyr, parabar, parallel, utils

Encoding UTF-8

Suggests knitr, rmarkdown, ggplot2, tidyr

VignetteBuilder knitr

NeedsCompilation no

Author Sacha Epskamp [aut, cre] (ORCID: <https://orcid.org/0000-0003-4884-8118>, Website: https://sachaepskamp.com),
Xinkai Du [ctb],
Mihai Constantin [aut] (ORCID: <https://orcid.org/0000-0002-6460-0107>, Website: https://mihaiconstantin.com),
Adela Maria Isvoranu [ctb]

Maintainer Sacha Epskamp <mail@sachaepskamp.com>

Repository CRAN

Date/Publication 2026-07-16 23:00:02 UTC

Contents

configure_bar *Configure The Progress Bar*

Description

This function can be used to conveniently configure the progress bar. It is exported for convenience from the parabar::parabar package. Please consult its documentation at parabar::configure_bar for more information on how to use it.

Usage

```
configure_bar(type = "modern", ...)
```

Arguments

type	A character string specifying the type of progress bar to be used with compatible backends. Possible values are "modern" and "basic". The default value is "modern".
...	A list of named arguments used to configure the progress bar. See the **Details** section for more information.

parSim *Parallel Simulator*

Description

The function parSim takes a set of conditions and an R base::expression and returns a base::data.frame with simulation results. The parSim function is based on dplyr::dplyr functions for handling the results, and the parabar::parabar package for handling parallelization and progress tracking.

Usage

```
parSim(
    ...,
    expression,
    replications = 1,
    reps,
    exclude = NULL,
    export = NULL,
    packages = NULL,
    write = FALSE,
    name,
    nCores = 1,
    progress = TRUE,
    seed = NULL,
    env = parent.frame()
)
```

Arguments

`...`	Any number of R vectors representing the simulation conditions. For example, if you want to vary the sample size between `{100, 250, 1000}` and a regression slope between `{0, 0.5, 1}`, you can assign the first two arguments as `sample_size = c(100, 250, 1000)` and `beta = c(0, 0.5, 1)`. Condition names should not collide with argument names of `parSim` or `parSim_dt`; a warning is given if they do. The columns `id`, `error`, `message` and `replication` are reserved for the output.
`expression`	An R expression that uses the simulation conditions as variable names. For example, if the `...` arguments are used to define `sample_size = c(100, 250, 1000)`, then, within the context of `expression`, you can access the `sample_size` variable, which may hold a value of `100`, `250`, or `1000` depending on the simulation condition. The expression must collect and output the results as a named `base::list` or a `base::data.frame`. See the the *Examples* section for more details.
`replications`	An integer representing the number of times each condition will be replicated. Please ensure that an adequate number of simulations is used (i.e., the more the better). Defaults to `1`.
`reps`	Deprecated alternative to 'replications'
`exclude`	A single (unquoted) logical expression indicating the simulation conditions to exclude. For example, `exclude = sample_size == 100 | beta == 0` will exclude all cases where `sample_size` is equal to `100` or `beta` is equal to `0`. Similarly, `exclude = sample_size == 100 & beta == 0` will exclude all cases where `sample_size` is equal to `100` and `beta` is equal to `0`. Defaults to `NULL` (i.e., no simulation conditions are excluded).
`export`	A character string containing the names of the objects to be exported to the parallel backend. This argument is only relevant when using parallel execution (i.e., `nCores > 1`). Defaults to `NULL` (i.e., indicating that no objects are to be exported).
`packages`	A character vector containing the names of the packages to be loaded on the parallel backend. For example `packages = c("bootnet", "qgraph")`. This argument is only relevant when using parallel execution (i.e., `nCores > 1`). Defaults to `NULL` (i.e., indicating that no packages are to be loaded).
`write`	Logical, should the results be written to a file? If `write = TRUE` and `name` is not provided, results are saved to a temporary file (i.e., created using `base::tempfile`). If `write = TRUE` and `name` is provided, results are saved to `name.txt`. If `write = FALSE` the results are not saved. Upon saving the results, a message is printed to the console indicating the location of the saved file. Defaults to `FALSE`.
`name`	A character string specifying the base name of the file to save the results to (a `.txt` extension is appended automatically). Only used when `write = TRUE`.
`nCores`	An integer value indicating the number of cores to use for parallel execution. Setting this argument to `1` implies that the simulation conditions are run sequentially. Using a value greater than `1` implies that the simulation conditions are run in parallel using a `parabar::parabar` backend with the specified number of cores. Defaults to `1`.

progress — A logical value indicating whether to show a progress bar while running the simulation. Defaults to TRUE.

seed — Optional integer. When supplied, one L'Ecuyer-CMRG random-number substream is derived per simulation condition (per design row), making the results fully reproducible *and identical for any value of* nCores. The caller's RNG state is restored afterwards. Defaults to NULL (no seeding).

env — The environment from which to export variables specified in the export argument, and in which unresolved symbols of expression and exclude are looked up when running sequentially. Defaults to parent.frame() (i.e., the caller's environment). This is relevant when calling parSim from within a function, where variables to export may not be in the global environment; export/env behave identically for sequential (nCores = 1) and parallel runs.

Details

The R base::expression should use object names assigned as conditions, and should return a named list with single values, or a base::data.frame. If you want to output more than one row of results per condition, you may return a base::data.frame with multiple rows. When running the simulation conditions in parallel, note that all packages needed should be loaded via the packages argument and all external objects used within the expression should be exported via the export argument.

Value

The parSim function returns a base::data.frame with the results of every iteration as a row, sorted by the id column (the row number of the condition in the fully expanded design). Besides the simulation conditions, the replication number and the columns returned by expression, the output contains: id (integer identifying the design row), error (logical; TRUE if the expression threw an error for this condition) and message (the error message, NA on success). Check any(output$error) after a run.

See Also

parSim_dt, bootnet::bootnet, parabar::par_lapply, and parabar::configure_bar.

Examples

```
# Determine a function to evaluate for each simulation condition.
bias <- function(x, y) {
    # Perform some computation.
    result <- abs(x - y)

    # Return the result.
    return(result)
}

# Run the simulation.
results <- parSim(
    # The simulation conditions.
    sample_size = c(50, 100, 250),
```

```
    beta = c(0, 0.5, 1),
    sigma = c(0.25, 0.5, 1),

    # The expression to evaluate for each simulation condition.
    expression = {
        # Generate the data.
        x <- rnorm(sample_size)
        y <- beta * x + rnorm(sample_size, sigma)

        # Fit the model.
        fit <- lm(y ~ x)

        # Compute the relevant quantities.
        beta_estimate <- coef(fit)[2]
        r_squared <- summary(fit)$r.squared
        bias <- bias(beta, beta_estimate)

        # Return in a compatible format.
        list(
            beta_estimate = beta_estimate,
            r_squared = r_squared,
            bias = bias
        )
    },

    # The number of replications.
    replications = 10,

    # The conditions to exclude.
    exclude = sample_size == 50 | beta <= 0.5,

    # The variables to export.
    export = c("bias"),

    # No packages are required for export.
    packages = NULL,

    # Do not save the results.
    write = FALSE,

    # Execute the simulation on a single core.
    nCores = 1,

    # Show the progress bar.
    progress = TRUE
)

# Print the head of the results.
head(results)


# Configure the progress bar.
configure_bar(
```

```
    type = "modern",
    format = "[:bar] [:percent] [:elapsed]",
    show_after = 0.15
)

# Run the simulation again with more cores and the updated progress bar.
results <- parSim(
    # The simulation conditions.
    sample_size = c(50, 100, 250),
    beta = c(0, 0.5, 1),
    sigma = c(0.25, 0.5, 1),

    # The expression to evaluate for each simulation condition.
    expression = {
        # Generate the data.
        x <- rnorm(sample_size)
        y <- beta * x + rnorm(sample_size, sigma)

        # Fit the model.
        fit <- lm(y ~ x)

        # Compute the relevant quantities.
        beta_estimate <- coef(fit)[2]
        r_squared <- summary(fit)$r.squared
        bias <- bias(beta, beta_estimate)

        # Return in a compatible format.
        list(
            beta_estimate = beta_estimate,
            r_squared = r_squared,
            bias = bias
        )
    },

    # The number of replications.
    replications = 1000,

    # The conditions to exclude.
    exclude = sample_size == 50 | beta <= 0.5,

    # The variables to export.
    export = c("bias"),

    # No packages are required for export.
    packages = NULL,

    # Save the results to a temporary file.
    write = TRUE,

    # Execute the simulation in parallel.
    nCores = 2,

    # Show the progress bar.
```

```
    progress = TRUE
)

# Print the tail of the results.
tail(results)
```

---

parSim_dt *Parallel Simulator data.table version*

Description

The function uses the more efficient data.table in its internal code instead of dplyr, thereby speeding up the function itself and also allowing researchers to use data.table to speed up the code they want to simulate with parSim.

Usage

```
parSim_dt(
    ...,
    expression,
    replications = 1,
    reps,
    write = FALSE,
    name,
    nCores = 1,
    export = NULL,
    packages = NULL,
    exclude,
    debug = FALSE,
    progress = TRUE,
    progressbar,
    env = parent.frame(),
    seed = NULL
)
```

Arguments

`...`	Any number of R vectors representing the simulation conditions.
`expression`	An R expression that uses the simulation conditions as variable names. The expression must return a named `list` or a `data.frame`. Row names of returned data frames are dropped; put identifiers in a proper column if needed.
`replications`	Integer. The number of replications for each condition. Defaults to 1.
`reps`	Deprecated. Alias for `replications`, kept for backward compatibility; if supplied it overrides `replications` and a deprecation warning is issued.

write	Logical, should the results be written to a file? If TRUE, the results are written to paste0(name, ".txt") when name is provided, or to a temporary file (via tempfile) otherwise, and the path is reported with a message. Unlike previous versions, the results are also returned (invisibly) rather than NULL. Defaults to FALSE.
name	Name of the file if write = TRUE. A .txt extension is appended automatically. If omitted, a temporary file is used.
nCores	An integer value indicating the number of cores to use for parallel execution. Defaults to 1.
packages	A character vector containing the names of packages to be loaded on the parallel backend (only relevant when nCores > 1). Defaults to NULL.
export	A character vector of global objects to export to the cluster. Defaults to NULL (nothing exported).
exclude	A character vector of logical expressions indicating simulation conditions to exclude. The expressions are combined with | (OR) and evaluated in the context of the simulation design: any row matching at least one expression is removed.
debug	Logical. If TRUE, prints iteration details during execution. Only useful when running sequentially (nCores = 1); in parallel runs the output goes to the worker consoles and is not visible. Defaults to FALSE.
progress	Logical. If TRUE, shows a progress bar while running the simulation. Defaults to TRUE.
progressbar	Deprecated. Alias for progress, kept for backward compatibility; if supplied (and progress is not) it sets progress and a deprecation warning is issued.
seed	Optional integer. When supplied, one L'Ecuyer-CMRG random-number substream is derived per simulation condition (per design row), making the results fully reproducible *and identical for any value of* nCores. The caller's RNG state is restored afterwards. Defaults to NULL (no seeding).
env	The environment from which to export variables specified in the export argument, and in which unresolved symbols of expression and exclude are looked up when running sequentially. Defaults to parent.frame() (i.e., the caller's environment). export/env behave identically for sequential (nCores = 1) and parallel runs.

Details

The function uses data.table in its internal code instead of dplyr, thereby speeding up the simulation and allowing researchers to use data.table in the code they want to simulate with parSim.

Value

A data.table with the results of each iteration in each row, sorted by the id column (the row number of the condition in the fully expanded design). Besides the simulation conditions, the replication number and the columns returned by expression, the output contains: id (integer identifying the design row), error (logical; TRUE if the expression threw an error) and message (the error message, NA on success). Check any(output$error) after a run. When write = TRUE the results are also written to a file and returned invisibly.

Author(s)

Xinkai Du <xinkai.du.xd@gmail.com>

See Also

parSim

Index