Package: parSim (via r-universe)

October 17, 2024

Type Package

Title Parallel Simulation Studies

Version 0.1.5

Author Sacha Epskamp

Maintainer Sacha Epskamp <mail@sachaepskamp.com>

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, see examples on <https://github.com/SachaEpskamp/parSim>.

Imports dplyr, methods, pbapply, snow, data.table, utils

Suggests ggplot2, tidyr

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2023-05-16 12:10:02 UTC

Contents

`parSim` *Parallel Simulator*

Description

Takes a set of conditions and an R expression and returns a data frame with simulation results. parSim is based on dplyr functions, and if you want to use the data.table package to make your simulation a bit faster, use parSim_dt. See details.

Usage

```
parSim(..., expression, reps = 1, write = FALSE, name,
       nCores = 1, export, exclude, debug = FALSE,
       progressbar = TRUE)

parSim_dt(..., expression, reps = 1, write = FALSE, name,
          nCores = 1, export, exclude, debug = FALSE,
          progressbar = TRUE)
```

Arguments

`...`	Vectors indicating any number of conditions. For example, if you want to vary sample size between N = 100, 250, and 1000, and a regression slope between beta = 0, 0.5, and 1, you can assign as first two arguments `sampleSize = c(100, 250, 1000)`, `beta = c(0, 0.5, 1)`.
`expression`	An R expression that uses the conditions as object names. For example, if the conditions `sampleSize = c(100, 250, 1000)` is used, then in the R expression you can use the object `sampleSize`, which may be 100, 250 or 1000 depending on the simulation condition.
`reps`	Number of times each condition has to be replicated.
`write`	Logical, should the results be written to a file instead of returned as a dataframe?
`name`	Name of the file if `write = TRUE`
`nCores`	Number of cores to use. NOTE: Only setting `nCores` to 1 allows you to see a progress bar and to use `browser()` in the R expression for debugging.
`export`	A character string of objects to be exported. Only needed if `nCores` > 1.
`exclude`	A list with logical calls to exclude cases. Written as formula.
`debug`	Allows for some debugging controls and output. Not recommended.
`progressbar`	Logical: should a progress bar be shown. Setting this to `FALSE` will make simulations much faster!

Details

The R expression should use object names assigned as conditions, and should return a list with single values, or a data frame / data table. If you want to output more than one row of results per condition, you may return a data frame / data table with multiple rows. When using multiple cores, note that all packages should be loaded in the R expression, all objects needed should be exported using the export object, and you will not see a progress bar.

Value

`parSim` outputs a data frame with the results of every iteration as a row.

`parSim_dt` outputs a data table with the results of every iteration as a row.

Author(s)

Sacha Epskamp <mail@sachaepskamp.com>

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

Examples

```
# Some function we might use:
bias <- function(x,y){abs(x-y)}

# Run the simulation:
Results <- parSim(
  # Any number of conditions:
  sampleSize = c(50, 100, 250),
  beta = c(0, 0.5, 1),
  sigma = c(0.25, 0.5, 1),

  # Number of repititions?
  reps = 25, # more is better!

  # Parallel?
  nCores = 1,

  # Write to file?
  write = FALSE,

  # Export objects (only needed when nCores > 1):
  export = c("bias"),

  # R expression:
  expression = {
    # Load all R packages in the expression if needed
    # library(...)

    # Want to debug? Enter browser() and run the function. Only works with nCores = 1!
    # browser()

    # Enter whatever codes you want. I can use the conditions as objects.
    X <- rnorm(sampleSize)
    Y <- beta * X + rnorm(sampleSize, sigma)
    fit <- lm(Y ~ X)
    betaEst <- coef(fit)[2]
    Rsquared <- summary(fit)$r.squared

    # Make a data frame with one row to return results (multple rows is
    # also possible but not recommended):
    data.frame(
      betaEst = betaEst,
      bias = bias(beta,betaEst),
      Rsquared = Rsquared
    )
  }
)

# Analyze the results:
library("ggplot2")
library("tidyr")
```

```
# We want to plot bias and R-squared. Let's first make it long format:
Long <- gather(Results,metric,value,bias:Rsquared)

# Make factors with nice labels:
Long$sigmaFac <- factor(Long$sigma,levels = c(0.25,0.5,1),
    labels = c("Sigma: 0.025", "Sigma: 0.5", "Sigma: 1"))

# Now let's plot:
g <- ggplot(Long, aes(x = factor(sampleSize), y = value, fill = factor(beta)))  +
  facet_grid(metric ~ sigmaFac, scales = "free") +
  geom_boxplot() +
  theme_bw() +
  xlab("Sample size") +
  ylab("") +
  scale_fill_discrete("Beta")
print(g)
```

Index