Package: MonteCarlo (via r-universe)

October 29, 2024

Type Package

Title Automatic Parallelized Monte Carlo Simulations

Version 1.0.6

Date 2019-01-29

Author Christian Hendrik Leschinski

Maintainer Christian Hendrik Leschinski <christian_leschinski@gmx.de>

Description Simplifies Monte Carlo simulation studies by automatically setting up loops to run over parameter grids and parallelising the Monte Carlo repetitions. It also generates LaTeX tables.

URL http://github.com/FunWithR/MonteCarlo

BugReports http://github.com/FunWithR/MonteCarlo/issues

License GPL-2

Imports abind(>= 1.4-0), codetools(>= 0.2-8), rlecuyer(>= 0.3-4), snowfall(>= 1.84-4), stats(>= 3.0.2), utils(>= 3.0.2), reshape(>= 0.8.6)

Depends snow(>= 0.4-1)

RoxygenNote 6.1.1

Suggests knitr, rmarkdown, testthat, dplyr, ggplot2

VignetteBuilder knitr

Repository https://funwithr.r-universe.dev

RemoteUrl https://github.com/funwithr/montecarlo

RemoteRef HEAD

RemoteSha f349d54ebe3648b34d32a49d094f691f00c997ab

Contents

MakeFrame *Conversion of MonteCarlo outputs to data.frame.*

Description

MakeFrame takes the output of MonteCarlo as its argument and returns a data.frame that contains the simulation results.

Usage

```
MakeFrame(output)
```

Arguments

output	A MonteCarlo object returned by the MonteCarlo() function.

Details

Each row of the data.frame contains the values returned by func for one repetition of the simulation and the respective values of the parameters.

Value

A data.frame that contains the simulation results.

Examples

```
test_func<-function(n,loc,scale){
 sample<-rnorm(n, loc, scale)
 stat<-sqrt(n)*mean(sample)/sd(sample)
 decision<-abs(stat)>1.96
 return(list("decision"=decision, "stat"=stat))
}

n_grid<-c(50,100,250,500)
loc_grid<-c(0,1)
scale_grid<-c(1,2)

param_list=list("n"=n_grid, "loc"=loc_grid, "scale"=scale_grid)
erg<-MonteCarlo(func=test_func, nrep=250, param_list=param_list, ncpus=1)
df<-MakeFrame(erg)
head(df)

library(dplyr)
library(ggplot2)
tbl <- tbl_df(df)
ggplot(filter(tbl, loc==0)) + geom_density(aes(x=stat, col=factor(n)))
```

MakeTable *Create LaTeX Tables From MonteCarlo Output.*

Description

MakeTable generates LaTeX tables with user determined ordering from the output of MonteCarlo.

Usage

```
MakeTable(output, rows, cols, digits = 4, collapse = NULL,
  transform = NULL, include_meta = TRUE, width_mult = 1,
  partial_grid = NULL)
```

Arguments

output	List of class MonteCarlo generated by MonteCarlo.
rows	Vector of parameter names to be stacked in the rows of the table. Ordered from the inside to the outside.
cols	Vector of parameter names to be stacked in the cols of the table. Ordered from the inside to the outside.
digits	Maximal number of digits displayed in table. Default is digits=4.
collapse	Optional list of the same length as the list returned by the function *func* supplied to MonteCarlo(). This list specifies the names of functions to be applied to the repective components of output when collapsing the results to a table. By default means are taken. Another example could be sd(). Currently, functions supplied have to return a scalar.
transform	Optional argument to transform the output table (for example from MSE to RMSE). If a function is supplied it is applied to all tables. Alternatively, a list of functions can be supplied that has the same length as the list returned by the function *func* supplied to MonteCarlo(). For tables that are supposed to stay unchanged set list element to NULL.
include_meta	Boolean that determines whether the meta data provided by summary() is included in comments below the table. Default is include_meta==TRUE.
width_mult	Scaling factor for width of the output table. Default is width_mult=1.
partial_grid	Optional list with the elements named after the parameters for which only a part of the grid values is supposed to be included in the table. Each component of the list is a vector that specifies the grid values of interest.

Details

To generate a two-dimensional table from the high dimensional array of simulation results in output, the results have to be stacked into rows and columns. The ordering of the resulting table is defined by the ordering in rows and cols that are ordered from the inside of the desired table to the outside.

The first two elements specify a matrix for all possible combinations from the grids for the two desired parameters. For a third parameter, the matrices for the first two can be stacked in columns

- one over the other - or in rows - one next to the other. The result of this is a larger matrix. This matrix produced for each value of the grid for the fourth parameter can again be stacked into rows or columns and so on. Consult the example.

To compile a Tex document containing the generated table include '\usepackage{multirow}' in the preamble.

To make the resultig tables more comprehensive, parameter grids of length one are dropped from the table (unless they are the only value in either cols or rows) and the information is added to the caption.

In case that the simulation function `func` used in `MonteCarlo` returns a list with more than one element (for example the results of two competing estimators or tests) separate tables are generated for each list element.

If it is desired to include the list elements in a single table, this behavior can be modified by adding "list" in one of the vectors `rows` or `cols` (see examples).

Examples

```
test_func<-function(n,loc,scale){
 sample<-rnorm(n, loc, scale)
 stat<-sqrt(n)*mean(sample)/sd(sample)
 decision<-abs(stat)>1.96
 return(list("decision"=decision))
}

n_grid<-c(50,100,250,500)
loc_grid<-seq(0,1,0.2)
scale_grid<-c(1,2)

param_list=list("n"=n_grid, "loc"=loc_grid, "scale"=scale_grid)
erg<-MonteCarlo(func=test_func, nrep=250, param_list=param_list, ncpus=1)
str(erg)

rows<-c("n")
cols<-c("loc","scale")
MakeTable(output=erg, rows=rows, cols=cols, digits=2)


#-------- Further Examples: Compare Mean and Median as Estimators for the Expected Value

# define func

func<-function(n,loc,scale){

 # generate sample
 sample<-rnorm(n, loc, scale)

 # calculate estimators
 mean_sample<-mean(sample)
 median_sample<-median(sample)

 # calculate bias
```

```
 bias_mean_sample<-mean_sample-loc
 bias_median_sample<-median_sample-loc

 # return results
 return(list("mean for calculation of sd"=mean_sample,
 "bias_mean"=bias_mean_sample,
 "median for calculation of sd"=median_sample,
 "bias_median"=bias_median_sample))
}

n_grid<-c(50,100,250,500)
loc_grid<-seq(0,1,0.2)
scale_grid<-c(1,2)

param_list=list("n"=n_grid, "loc"=loc_grid, "scale"=scale_grid)
erg_mean_median<-MonteCarlo(func=func, nrep=250, param_list=param_list, ncpus=1)

rows<-c("n")
cols<-c("loc","scale")

# use partial_grid

MakeTable(output=erg_mean_median, rows=rows, cols=cols, digits=2,
          partial_grid=list("n"=c(1,3), "loc"=c(1,3,5)), include_meta=FALSE)

# use collapse to calculate standard deviation and bias

collapse<-list("sd", "mean", "sd", "mean")
MakeTable(output=erg_mean_median, rows=rows, cols=cols, digits=2,
          collapse=collapse, include_meta=FALSE)

# merge all results in one table

MakeTable(output=erg_mean_median, rows=c("n","loc"), cols=c("scale","list"),
          digits=2, collapse=collapse, include_meta=FALSE)

# transform the results for better scaling

scale_table_10<-function(x){x*10}

MakeTable(output=erg_mean_median, rows=c("n","loc"), cols=c("scale","list"),
          digits=2, collapse=collapse,
          transform=list(scale_table_10, NULL, function(x){x*10}, NULL),
          include_meta=FALSE)
```

MergeResults *Merge Lists Returned From Separate Simulations With MonteCarlo().*

Description

`MergeResults` is a utility function that allows to merge the output from separate simulations using the same function and parameter grid.

Usage

```
MergeResults(identifier, path)
```

Arguments

`identifier`	String that is common to the names of the files that are supposed to be merged.
`path`	String specifying the path to directory that contains the files.

Details

To merge two or more files with simulation results they have to be saved using `save`. The identifier string has to be part of the name of all targeted files, but not part of the names of any other files in the directory.

Examples

```
out<-MergeResults(identifier="MonteCarloResults", path="C:/Users/")
summary(out)
```

`MonteCarlo` *Parallized Monte Carlo Simulation*

Description

`MonteCarlo` runs a Monte Carlo simulation study for a correctly specified function and the desired parameter grids. See details for instructions on the specification of the function.

Usage

```
MonteCarlo(func, nrep, param_list, ncpus = 1, max_grid = 1000,
  time_n_test = FALSE, save_res = FALSE, raw = TRUE,
  export_also = NULL)
```

Arguments

`func`	The function to be evaluated. See details.
`nrep`	An integer that specifies the desired number of Monte Carlo repetitions.
`param_list`	A list whose components are named after the parameters of `func` and each component is a vector containing the desired grid values for that parameter
`ncpus`	An integer specifying the number of cpus to be used. Default is `ncpus=1`. For `ncpus>1` the simulation is parallized automatically using `ncpus` cpu units.

max_grid	Integer that specifies for which grid size to throw an error, if grid becomes to large. Default is max_grid=1000.
time_n_test	Boolean that specifies whether the required simulation time should be estimated (useful for large simulations or slow functions). See details. Default is time_n_test=FALSE.
save_res	Boolean that specifies whether the results of time_n_test should be saved to the current directory. Default is save_res=FALSE.
raw	Boolean that specifies whether the output should be averaged over the nrep repetitions. Default is raw=TRUE.
export_also	List specifying additional objects that are supposed to be exported to the cluster. This allows to export data or to bypass the automatic export of functions. Default is export_also=NULL. See details.

Details

The user defined function func handles the generation of data, the application of the method of interest and the evaluation of the result for a single repetition and parameter combination. MonteCarlo handles the generation of loops over the desired parameter grids and the repetition of the Monte Carlo experiment for each of the parameter constellations.

There are two important formal requirements that func has to fulfill.

1. The arguments of func have to be scalar.

2. The value returned by func has to be list of (unnamed) scalars (The list elements can be named).

For the estimation of the required simulation time, a separate simulation is run on a reduced grid that only contains the extreme points for each parameter, e.g. the smallest and the largest sample size. This test simulation is carried out with nrep/10 repetitions and the required simulation time is estimated by a linear interpolation. Since the computational complexity is usually a convex function of the sample size and the dimension of the process, this approach tends to overestimate the time required.

export_also allows to export data to the cluster in case parallized computations on a dataset are desired. It also allows to bypass the automatic export of functions and packages. To manually export a function or dataset or to load a package, pass a list to export_also where the list elements are named "functions", "data" and/or "packages". For example: export_also=list("functions"=c("function_name_1", "function_name_2"),"packages"="package_name", "data"="mtcars".

Value

A list of type MonteCarlo.

Examples

```
test_func<-function(n,loc,scale){
 sample<-rnorm(n, loc, scale)
 stat<-sqrt(n)*mean(sample)/sd(sample)
 decision<-abs(stat)>1.96
 return(list("decision"=decision))
}

# Example without parallization
```

```
n_grid<-c(50,100,250,500)
loc_grid<-seq(0,1,0.2)
scale_grid<-c(1,2)

param_list=list("n"=n_grid, "loc"=loc_grid, "scale"=scale_grid)
erg<-MonteCarlo(func=test_func, nrep=250, param_list=param_list, ncpus=1)
summary(erg)

rows<-c("n")
cols<-c("loc","scale")
MakeTable(output=erg, rows=rows, cols=cols, digits=2)

# Note that parallized computation is not always faster,
# due to the computational costs of the overhead
# that is needed to manage multiple CPUs.
```

Index