Package 'simTool'

March 26, 2018

Type Package

Title Conduct Simulation Studies with a Minimal Amount of Source Code

Version 1.1.0

Date 2018-03-26

Maintainer Marsel Scheer <scheer@freescience.de>

Description The simTool package is designed for statistical simulations that have two components. One component generates the data and the other one analyzes the data. The main aims of the simTool package are the reduction of the administrative source code (mainly loops and management code for the results) and a simple applicability of the package that allows the user to quickly learn how to work with the simTool package. Parallel computing is also supported. Finally, convenient functions are provided to summarize the simulation results.

Depends R (>= 2.14.0)

Imports plyr (>= 1.8.1), reshape (>= 0.8.5), dplyr (>= 0.7.2), purrr (>= 0.2.3), tidyr (>= 0.6.2), tibble (>= 1.4.2), parallel

Suggests ggplot2, knitr, boot, broom, testthat, rmarkdown

License GPL-3

VignetteBuilder knitr

RoxygenNote 6.0.1

URL https://github.com/MarselScheer/simTool

BugReports https://github.com/MarselScheer/simTool/issues

NeedsCompilation no

Author Marsel Scheer [aut, cre]

Repository CRAN

Date/Publication 2018-03-26 14:58:24 UTC

R topics documented:

`simTool-package` *Conduct Simulation Studies with a Minimal Amount of Source Code*

Description

The `simTool` package is designed for statistical simulations that have two components. One component generates the data and the other one analyzes the data. The main aims of the `simTool` package are the reduction of the administrative source code (mainly loops and management code for the results) and a simple applicability of the package that allows the user to quickly learn how to work with the `simTool` package. Parallel computing is also supported. Finally, convenient functions are provided to summarize the simulation results.

Details

Package:	simTool
Type:	Package
Version:	1.0.3
Date:	2013-02-21
License:	GPL-3

`evalGrids` is the workhorse. `as.data.frame` is function coercing the result object of evalGrids to a data.frame. `expandGrid` is only a convenient function

Author(s)

Marsel Scheer

Maintainer: scheer@freescience.de

Examples

```
dg = expand_tibble(fun="rexp", n=c(10, 20), rate=1:2)
pg = expand_tibble(proc="summary")
```

```
eval_tibbles(dg, pg, replications=3)
eval_tibbles(dg, pg, replications=3, summary_fun = list(mean = mean))
eval_tibbles(dg, pg, replications=3, summary_fun = list(mean = mean, sd = sd))
```

`as.data.frame.evalGrid` *Converts an* `evalGrid` *object into a* `data.frame`

Description

Converts the results contained in the object returned by `evalGrids` into a `data.frame`. If the results can not be coerced automatically into a `data.frame`, the user can provide a function to pre-process the results (see `convert.result.fun`). Furthermore, univariate functions to summarize the results over the replications can be specified via `summary.fun`.

Usage

```
## S3 method for class 'evalGrid'
as.data.frame(x, ..., convert.result.fun = identity,
  summary.fun = NULL, progress = FALSE)
```

Arguments

`x`	an object returned by `evalGrids`
`...`	only for S3 method consistency
`convert.result.fun`	a functions that converts the result object contained in `x` into a `data.frame`
`summary.fun`	univariate functions to summarize the results (numeric or logical) over the replications, e.g. mean, sd. Alternatively, `summary.fun` can be one function that may return a vector.
`progress`	if `TRUE` a progress bar is shown in the console.

Value

a `data.frame` with the parameter constellations for the data generation and evaluation and the results (probably summarized).

Author(s)

Marsel Scheer

See Also

`evalGrids`

Examples

```
genRegData <- function(){
 data.frame(
     x = 1:10,
     y = rnorm(10, mean=1:10))
}

eg <- evalGrids(
 expandGrid(fun="genRegData"),
 expandGrid(proc="lm", formula=c("y ~ x", "y ~ x + I(x^2)")),
 replications=5)

lm2df = function(lm.object) {
 ret = coef(summary.lm(lm.object))[, 1:2]
 data.frame(covariable = rownames(ret), ret, check.names=FALSE)
}
as.data.frame(eg, convert.result.fun=lm2df, progress=TRUE)
as.data.frame(eg, convert.result.fun=lm2df, summary.fun=c(mean, sd), progress=TRUE)
```

`evalGrids` *Workhorse for simulation studies*

Description

Generates data according to all provided constellations in `dataGrid` and applies all provided constellations in `procGrid` to them.

Usage

```
evalGrids(dataGrid, procGrid = expandGrid(proc = "length"),
  replications = 1, discardGeneratedData = FALSE, progress = FALSE,
  summary.fun = NULL, ncpus = 1L, cluster = NULL,
  clusterSeed = rep(12345, 6), clusterLibraries = NULL,
  clusterGlobalObjects = NULL, fallback = NULL, envir = globalenv(), ...)
```

Arguments

`dataGrid`	a `data.frame` where the first column is a character vector with function names. The other columns contain parameters for the functions specified in the first column. Parameters with NA are ignored.
`procGrid`	similar as `dataGrid` the first column must contain function names. The other columns contain parameters for the functions specified in the first column. The data generated according to `dataGrid` will always be passed to the first unspecified argument of the functions specified in the first column of `procGrid`.
`replications`	number of replications for the simulation

`discardGeneratedData`
: if `TRUE` the generated data is deleted after all function constellations in `procGrid` have been applied. Otherwise, ALL generated data sets will be part of the returned object.

`progress`
: if `TRUE` a progress bar is shown in the console.

`summary.fun`
: univariate functions to summarize the results (numeric or logical) over the replications, e.g. mean, sd. Alternatively, `summary.fun` can be one function that may return a vector.

`ncpus`
: a cluster of `ncpus` workers (R-processes) is created on the local machine to conduct the simulation. If `ncpus` equals one no cluster is created and the simulation is conducted by the current R-process.

`cluster`
: a cluster generated by the `parallel` package that will be used to conduct the simulation. If `cluster` is specified, then `ncpus` will be ignored.

`clusterSeed`
: if the simulation is done in parallel manner, then the combined multiple-recursive generator from L'Ecuyer (1999) is used to generate random numbers. Thus `clusterSeed` must be a (signed) integer vector of length 6. The 6 elements of the seed are internally regarded as 32-bit unsigned integers. Neither the first three nor the last three should be all zero, and they are limited to less than 4294967087 and 4294944443 respectively.

`clusterLibraries`
: a character vector specifying the packages that should be loaded by the workers.

`clusterGlobalObjects`
: a character vector specifying the names of R objects in the global environment that should be exported to the global environment of every worker.

`fallback`
: must be missing or a character specifying a file. Every time when the data generation function is changed, the results so far obtained are saved in the file specified by `fallback`.

`envir`
: must be provided if the functions specified in `dataGrid` or `procGrid` are not part of the global environment.

`...`
: only needed to alert the user if some deprecated arguments were used.

Value

The returned object is a list of the class `evalGrid`, where the fourth element is a list of lists named `simulation`. `simulation[[i]][[r]]` contains:

`data`
: the data set that was generated by the `i`th constellation (`i`th row) of `dataGrid` in the `r`th replication

`results`
: a list containing `nrow(procGrid)` objects. The `j`th object is the returned value of the function specified by the `j`th constellation (`j`th row) of `procGrid` applied to the data set contained in `data`

Note

If `cluster` is provided by the user the function `evalGrids` will NOT stop the cluster. This has to be done by the user. Conducting parallel simulations by specifying `ncpus` will internally create a cluster and stop it after the simulation is done.

Author(s)

Marsel Scheer

See Also

as.data.frame.evalGrid

Examples

```
rng = function(data, ...) {
ret = range(data)
names(ret) = c("min", "max")
ret
}

# call runif(n=1), runif(n=2), runif(n=3)
# and range on the three "datasets"
# generated by runif(n=1), runif(n=2), runif(n=3)
eg = evalGrids(
 expandGrid(fun="runif", n=1:3),
 expandGrid(proc="rng"),
 rep=10
)
eg

# summarizing the results in a data.frame
as.data.frame(eg)

# we now generate data for a regression
# and fit different regression models

# not that we use SD and not sd (the
# reason for this is the cast() call below)
regData = function(n, SD){
 data.frame(
   x=seq(0,1,length=n),
   y=rnorm(n, sd=SD))
}

eg = evalGrids(
 expandGrid(fun="regData", n=20, SD=1:2),
 expandGrid(proc="lm", formula=c("y~x", "y~I(x^2)")),
 replications=2)

# can not be converted to data.frame, because
# an object of class "lm" can not converted to
# a data.frame
try(as.data.frame(eg))

# for the data.frame we just extract the r.squared
# from the fitted model
```

```
as.data.frame(eg, convert.result.fun=function(fit) c(rsq=summary(fit)$r.squared))

# for the data.frame we just extract the coefficients
# from the fitted model
df = as.data.frame(eg, convert.result.fun=coef)

# since we have done 2 replication we can calculate
# sum summary statistics
library("reshape")
df$replication=NULL
mdf = melt(df, id=1:7, na.rm=TRUE)
cast(mdf, ... ~ ., c(mean, length, sd))

# note if the data.frame would contain the column
# named "sd" instead of "SD" the cast will generate
# an error
names(df)[5] = "sd"
mdf = melt(df, id=1:7, na.rm=TRUE)
try(cast(mdf, ... ~ ., c(mean, length, sd)))


# extracting the summary of the fitted.model
as.data.frame(eg, convert.result.fun=function(x) {
 ret = coef(summary(x))
 data.frame(valueName = rownames(ret), ret, check.names=FALSE)
})



# we now compare to methods for
# calculating quantiles

# the functions and parameters
# that generate the data
N = c(10, 50, 100)
library("plyr")
dg = rbind.fill(
 expandGrid(fun="rbeta", n=N, shape1=4, shape2=4),
 expandGrid(fun="rnorm", n=N))

# definition of the two quantile methods
emp.q = function(data, probs) c(quantile(data, probs=probs))
nor.q = function(data, probs) {
 ret = qnorm(probs, mean=mean(data), sd=sd(data))
 names(ret) = names(quantile(1, probs=probs))
 ret
}

# the functions and parameters that are
# applied to the generate data
pg = rbind.fill(expandGrid(proc=c("emp.q", "nor.q"), probs=c(0.01, 0.025, 0.05)))

# generate data and apply quantile methods
```

```
set.seed(1234)
eg = evalGrids(dg, pg, replication=50, progress=TRUE)

# convert the results to a data.frame
df = as.data.frame(eg)
df$replication=NULL
mdf = melt(df, id=1:8, na.rm=TRUE)

# calculate, print and plot summary statistics
require("ggplot2")
print(a <- arrange(cast(mdf, ... ~ ., c(mean, sd)), n))
ggplot(a, aes(x=fun, y=mean, color=proc)) + geom_point(size=I(3)) + facet_grid(probs ~ n)
```

`eval_tibbles` *Workhorse for simulation studies*

Description

Generates data according to all provided constellations in `data_tibble` and applies all provided constellations in `proc_tibble` to them.

Usage

```
eval_tibbles(data_grid, proc_grid = expand_tibble(proc = "length"),
  replications = 1, discard_generated_data = FALSE,
  post_analyze = identity, summary_fun = NULL, group_for_summary = NULL,
  ncpus = 1L, cluster = NULL, cluster_seed = rep(12345, 6),
  cluster_libraries = NULL, cluster_global_objects = NULL,
  envir = globalenv(), simplify = TRUE)
```

Arguments

`data_grid`	a `data.frame` or `tibble` where the first column is a character vector with function names. The other columns contain parameters for the functions specified in the first column. Parameters with NA are ignored.
`proc_grid`	similar as `data_grid` the first column must contain function names. The other columns contain parameters for the functions specified in the first column. The data generated according to `data_grid` will always be passed to the first unspecified argument of the functions specified in the first column of `proc_grid`.
`replications`	number of replications for the simulation
`discard_generated_data`	if `TRUE` the generated data is deleted after all function constellations in `proc_grid` have been applied. Otherwise, ALL generated data sets will be part of the returned object.
`post_analyze`	this is a convenience function, that is applied directly after the data analyzing function.

summary_fun — named list of univariate function to summarize the results (numeric or logical) over the replications, e.g. list(mean = mean, sd = sd).

group_for_summary — if the result returned by the data analyzing function or post_analyze is a data.frame with more than one row, one usually is interested in summarizing the results while grouping for some variables. This group variables can be passed as a character vector into group_for_summary

ncpus — a cluster of ncpus workers (R-processes) is created on the local machine to conduct the simulation. If ncpus equals one no cluster is created and the simulation is conducted by the current R-process.

cluster — a cluster generated by the parallel package that will be used to conduct the simulation. If cluster is specified, then ncpus will be ignored.

cluster_seed — if the simulation is done in parallel manner, then the combined multiple-recursive generator from L'Ecuyer (1999) is used to generate random numbers. Thus cluster_seed must be a (signed) integer vector of length 6. The 6 elements of the seed are internally regarded as 32-bit unsigned integers. Neither the first three nor the last three should be all zero, and they are limited to less than 4294967087 and 4294944443 respectively.

cluster_libraries — a character vector specifying the packages that should be loaded by the workers.

cluster_global_objects — a character vector specifying the names of R objects in the global environment that should be exported to the global environment of every worker.

envir — must be provided if the functions specified in data_grid or proc_grid are not part of the global environment.

simplify — usually the result column is nested, by default it is tried to unnest it.

Value

The returned object list of the class eval_tibbles, where the element simulations contain the results of the simulation.

Note

If cluster is provided by the user the function eval_tibbles will NOT stop the cluster. This has to be done by the user. Conducting parallel simulations by specifying ncpus will internally create a cluster and stop it after the simulation is done.

Author(s)

Marsel Scheer

Examples

```
rng = function(data, ...) {
ret = range(data)
names(ret) = c("min", "max")
```

```
ret
}

dg <- expand_tibble(fun = "rnorm", n = c(5L, 10L))
pg <- expand_tibble(proc = c("rng", "median", "length"))

eval_tibbles(dg, pg,rep = 2)
eval_tibbles(dg, pg,rep = 2, post_analyze = purrr::compose(tibble::as_tibble, t))
eval_tibbles(dg, pg,rep = 2, summary_fun = list(mean = mean, sd = sd))

regData = function(n, SD){
  data.frame(
  x=seq(0,1,length=n),
  y=rnorm(n, sd=SD))
}

eg <- eval_tibbles(
  expand_tibble(fun="regData", n=5L, SD=1:2),
  expand_tibble(proc="lm", formula=c("y~x", "y~I(x^2)")),
  group_for_summary = "term",
  replications=3
)
eg

presever_rownames = function(mat)
{
  rn = rownames(mat)
  ret = tibble::as_tibble(mat)
  ret$term = rn
  ret
}

eg <- eval_tibbles(
  expand_tibble(fun="regData", n=5L, SD=1:2),
  expand_tibble(proc="lm", formula=c("y~x", "y~I(x^2)")),
  post_analyze = purrr::compose(presever_rownames, coef, summary),
  #post_analyze = broom::tidy, # is a nice out of the box alternative
  summary_fun = list(mean = mean, sd = sd),
  group_for_summary = "term",
  replications=3
)
tidyr::unnest(eg$simulation)
```

`expandGrid` *Creates a* `data.frame` *from All Combinations*

Description

Actually a wrapper for `expand.grid`, but character vectors will stay as characters.

Usage

```
expandGrid(...)
```

Arguments

...	vectors, factors or a list containing these.

Value

See expand.grid

Author(s)

Marsel Scheer

See Also

expand.grid

Examples

```
expandGrid(fun="rnorm", mean=1:4, sd=2:5)
```

---

expand_tibble *Creates a* tibble *from All Combinations*

Description

Actually a wrapper for expand.grid, but character vectors will stay as characters.

Usage

```
expand_tibble(...)
```

Arguments

...	vectors, factors or a list containing these.

Value

See expand.grid but instead of a data.frame a tibble is returned.

Author(s)

Marsel Scheer

See Also

expand.grid

Examples

```
expand_tibble(fun="rnorm", mean=1:4, sd=2:5)
```

---

meanAndNormCI *A convenient function to calculate the mean and a 95% confidence interval*

Description

The 95% confidence interval is based on a normal approximation.

Usage

```
meanAndNormCI(results)
```

Arguments

results	a numeric or logical vector

Author(s)

Marsel Scheer

Examples

```
meanAndNormCI(rexp(10^4, rate=2))
```

---

print.eval_tibbles *Printing simulation results*

Description

Prints objects created by eval_tibbles()

Usage

```
## S3 method for class 'eval_tibbles'
print(x, ...)
```

Arguments

`x`	object of class `eval_tibbles`
`...`	not used. only necessary to define the function consistently with respect to `print(x, ...)`

Author(s)

Marsel Scheer

Index