Package 'simsalapar'

April 27, 2023

Version 1.0-12

Date 2023-04-26

Title Tools for Simulation Studies in Parallel

Description Tools for setting up (``design''), conducting, and evaluating large-scale simulation studies with graphics and tables, including parallel computations.

Author Marius Hofert and Martin Maechler <maechler@stat.math.ethz.ch>

Maintainer Marius Hofert <mhofert@hku.hk>

Depends R (>= 3.1.0), graphics

Imports stats, parallel, utils, grDevices, methods, grid, sfsmisc, gridBase (>= 0.4-6), colorspace

Suggests lattice, Rmpi, Hmisc, copula, foreach, doParallel, fGarch, robustbase

SuggestsNote copula is only used for the vignettes, see their VignetteDepends; fGarch: only used in demo(TGforecasts), robustbase in another demo.

KeepSource yes

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2023-04-27 08:30:02 UTC

R topics documented:

`simsalapar-package` *Tools for Simulation Studies in Parallel with* R

Description

Tools for setting up, conducting, and evaluating larger-scale simulation studies, including parallel computations, in R.

Details

The DESCRIPTION file:

Package:	simsalapar
Version:	1.0-12
Date:	2023-04-26
Title:	Tools for Simulation Studies in Parallel
Description:	Tools for setting up ("design"), conducting, and evaluating large-scale simulation studies with graphics and ta
Author:	Marius Hofert and Martin Maechler <maechler@stat.math.ethz.ch>
Maintainer:	Marius Hofert <mhofert@hku.hk>
Depends:	R (>= 3.1.0), graphics
Imports:	stats, parallel, utils, grDevices, methods, grid, sfsmisc, gridBase (>= 0.4-6), colorspace
Suggests:	lattice, Rmpi, Hmisc, copula, foreach, doParallel, fGarch, robustbase
SuggestsNote:	copula is only used for the vignettes, see their VignetteDepends; fGarch: only used in demo(TGforecasts), ro
KeepSource:	yes
License:	GPL-2 | GPL-3
Encoding:	UTF-8

Index of help topics:

```
dev.off.pdf             Cropping and Font Embedding PDF Device
doCallWE                Innermost Computation: Error Catching Version
                        of do.call()
doCheck                 Checking a User's doOne
doLapply                Functions for Iterating Over All Subjobs
expr2latex              Translate 'plotmath' expressions to LaTeX
```

getEl	Tools For Working with Variable Specification Lists
LEseeds	Advancing .Random.seed for "L'Ecuyer-CMRG"
mayplot	Matrix-like Plot for Arrays up to Rank 5
simsalapar-package	Tools for Simulation Studies in Parallel with R
subjob	Subjob - Compute one Row of the Virtual Grid
toLatex.ftable	Convert Flat Contingency Table (ftable) and VarLists to LaTeX Table
tryCatch.W.E	Catching and Storing Warnings and Errors Simultaneously
ul	Tools For Converting To and From Arrays, Lists, and Array of Lists
varlist	Variable Specification List - Generation and Class
wrapLaTable	Wrapper for a floating LaTeX Table

Setting up a simulation:

varlist() creates a variable specification list.

dimnames2varlist() creates a variable specification list from given dimension names.

getEl() extracts elements from a variable list.

mkGrid() function for creating a grid of all variables of type "grid"; see mkGrid().

mkNms() builds a list of names from a variable list; see mkNms().

get.n.sim() extracts "n.sim"; see get.n.sim().

get.nonGrids() extracts all variables not of type "grid"; see get.nonGrids().

Conducting a simulation:

tryCatch.W.E() catching and storing warnings and errors simultaneously; see tryCatch.W.E().

doCallWE() innermost computation (return value of doOne()): returns value, error, warning, and run time; see doCallWE().

LEseeds() create a list of advanced .Random.seed's for "L'Ecuyer-CMRG"; see LEseeds().

printInfo() displays information about the sub-job just finished; see printInfo().

subjob() computes one row of the virtual grid in a simulation; see subjob().

mkTimer() creates a function to be passed to doCallWE() as timer; see mkTimer().

doLapply() sequentially iterates over all subjobs via standard lapply().

doForeach() iterates over all subjobs in parallel (via foreach(), package **foreach**).

doRmpi() iterates over all subjobs in parallel (via **Rmpi**'s mpi.apply()).

doMclapply() iterates over all subjobs in parallel (via mclapply()).

doClusterApply() iterates over all subjobs in parallel (via clusterApply()).

Analysis:

doRes.equal() convenience wrapper for comparing two results of the do* lapply-like functions; see doRes.equal().

mkAL() converts a list of named 5-lists to an array of lists; see mkAL().

saveSim() (optionally) converts a result list to an array of lists using mkAL(); see saveSim().

maybeRead() (optionally) reads the provided .rds; see maybeRead().

getArray() gets an array of 4-lists and computes an array of values, errors, warnings, or run times; see `getArray()`.

`array2df()` conveniently converts an array to a `data.frame`.

`toLatex()`**:** an S3 method for varlist and ftable.

`fftable()` essentially calls `format.ftable()` and adds attributes `ncv` and `nrv` to the return object.

`tablines()` computes ingredients for converting a character matrix with attributes to a LaTeX table.

`wrapLaTable()` wraps a table and tabular environment around the lines of the body of a LaTeX table.

`mayplot()`**:** a matrix-like plot for arrays up to rank 5, with grid and gridBase.

Author(s)

Marius Hofert and Martin Maechler <maechler@stat.math.ethz.ch>

Maintainer: Marius Hofert <marius.hofert@math.ethz.ch>

References

Publication Marius Hofert, Martin Maechler (2016). Parallel and Other Simulations in R Made Easy: An End-to-End Study. *Journal of Statistical Software*, **69**(4), 1–44. doi:10.18637/jss.v069.i04

Preprint (for simsalapar 1.0-0; including timing info): Hofert, M. and Mächler, M. (2013). Parallel and other simulations in R made easy: An end-to-end study. https://arxiv.org/abs/1309.4402

Examples

```
## Not run:
 demo(TGforecasts)

## End(Not run)
```

`array-stuff` *Tools For Converting To and From Arrays, Lists, and Array of Lists*

Description

ul() is a simple wrapper for `unlist()` with `recursive=FALSE`.

mkAL() gets a list `x` with elements that are named lists of length five, see `x` below, and converts it to an array of `list`s.

saveSim() (optionally) converts a result list to an array using `mkAL()` and (optionally) saves it to a file via `saveRDS()`.

maybeRead() if the provided '`.rds`' file exists, this function reads it via `readRDS()`; otherwise, nothing is done.

getArray() gets an array of 4-lists as returned by `mkAL()`, picks out the specified component `comp`, applies the specified function `FUN` (with useful defaults), and builds an `array`.

array2df() auxiliary function to convert an array to a `data.frame` (correctly dealing with `n.sim`).

Usage

```
ul(x)

mkAL   (x, vList, repFirst, check=TRUE)
saveSim(x, vList, repFirst, sfile, check=TRUE, doAL=TRUE)

maybeRead(sfile, msg=TRUE)

getArray(x, comp=c("value", "error", "warning", "time"),
         FUN=NULL, err.value=NA)

array2df(x, responseName = "value")
```

Arguments

`x`	for **ul()** a list. **mkAL(), saveSim()** a list (of length `n.sim * nrow(pGrid)`) where each element is a list of length five, containing the named elements `"value"`, `"error"`, `"warning"`, `"time"`, and `".Random.seed"`, the first four as returned by `doCallWE()`. **getArray()** an array of `list`s as returned by `mkAL()`. **array2df()** a numeric array as returned by `getArray(*, "value")`.
`vList`	a `list` of variable specifications. Each variable specification is itself a named list which must contain a `"value"` component.
`repFirst`	logical; **must** match the value of `repFirst` in the `x <- do*Apply()` call where `x` has been created.
`check`	`logical` activating consistency checks for `x`.
`sfile`	a file name, typically with extension '`.rds`' or `NULL`.
`doAL`	`logical` indicating if `mkAL()` should be called, or rather just `x` be saved.
`msg`	`logical` indicating whether a message is printed when an object is read from `sfile`.
`err.value`	`numeric` which is used to replace the value of the array entry in case of an error.
`comp`	character string denoting the component.
`FUN`	function to be applied right before the resulting array `array` is constructed.
`responseName`	(for `arrady2df()`:) a string specifying the name of the "value" column of the resulting data frame.

Details

`mkAL()` is useful when creating arrays from result lists returned from large(r) simulation studies which use `doCallWE()`. To create a proper argument `x` for `mkAL()`, the function `ul()` turns out to be useful to (stepwise) unlist nested lists.

`getArray()` converts arrays of lists as returned by `mkAL()` to an `array` of `numeric` (or `logical`, see below) after applying the specified `FUN`.
In case of an error, the corresponding entry in the resulting array is replaced by `err.value`.

The default `FUN` converts possible errors and warnings to `logical` (indicating whether there was a error or warning, respectively) and run times to `numeric`. For `comp="value"`, the situation is trickier. First of all, the resulting array contains dimensions for variables of type "inner" and, if greater than 1, for the variable of type "N" (typically called `"n.sim"`); see the vignette for details. Use `FUN = identity` to get at the full error or warning objects, for `comp = "error"` or for `comp = "warning"`, respectively.

`saveSim()` and `maybeRead()` are useful for creating and (re)storing arrays from large(r) simulation studies (to avoid recomputation, to ease the data analysis etc.). `saveSim()` calls `mkAL()`, nowadays wrapped in `tryCatch(.)`, such that the simulation is not lost, even when the resulting format cannot correctly be treated by `mkAL()`. Consequently, `doAL` is not much needed anymore. Note that both `saveSim()` and `maybeRead()` accept `sfile=NULL` in which case nothing is saved or read.

Value

For

ul() the unlisted list; see `unlist()`.

mkAL() an `array` of `list`s.

saveSim() the `array` returned by `mkAL()`.

maybeRead() the object read by `readRDS()` from `sfile` or nothing (if `sfile` does not exist).

getArray() an `array` containing the values of the specified component `comp` after applying `FUN` to them. The default `FUN` produces an `array`, depending on `comp`, of

- **"value":** values or `err.value` (in case of an error)
- **"error":** logicals indicating whether there was an error
- **"warning":** logicals indicating whether there was a warning
- **"time":** timings as returned by `doCallWE()`, i.e., typically (from `mkTimer`'s `proc.time()[1]`) the number of milliseconds of ""CPU user time"".

array2df(x) a `data.frame` with several columns built from the `dimnames(x)` and a column named `responseName` with the values of `x`.

Author(s)

Marius Hofert and Martin Maechler.

References

see `simsalapar-package`.

See Also

getEl() and mkNms() used by mkAL(). saveRDS() and readRDS(), the "workhorses" of saveSim() and maybeRead(), respectively.

Examples

```
## Not run:
## Get at the full error objects, notably (message, call):
errObjs <- getArray(res, "error", FUN=identity)

## End(Not run)

if(FALSE) ## A longer, "interesting" example is in
 demo(robust.mean)
```

---

device *Cropping and Font Embedding PDF Device*

Description

dev.off.pdf() is a wrapper of dev.off() which is meant for closing a pdf device. It also performs cropping and font embedding if chosen.

Usage

```
dev.off.pdf(file="Rplots.pdf", crop=NULL, embedFonts="", ...)
```

Arguments

file	output file name including extension .pdf.
crop	cropping command, can be one of:
	NULL crop with the command "pdfcrop --pdftexcmd pdftex file file 1>/dev/null 2>&1". This is suitable for Unix; for non-Unix, no cropping is done.
	character a string containing the crop command.
	"" do not crop.
embedFonts	font embedding command, can be one of:
	NULL embed fonts with the command embedFonts(file, options="-dSubsetFonts=true -dEmbedAllFonts=true -dPDFSETTINGS=/printer -dUseCIEColor"). This is suitable for Unix; for non-Unix, no font embedding is done.
	character a string containing a font embedding command.
	"" do not embed fonts.
...	additional arguments passed to dev.off().

Value

invisible().

Author(s)

Marius Hofert

See Also

`dev.off()` for closing a device, `embedFonts()` for font embedding. **sfsmisc**'s `pdf.end()` for another approach.

Examples

```
## typical usage
doPDF <- !dev.interactive(orNone=TRUE)
if(doPDF) pdf(file=(file <- "crop_device.pdf"), width=6, height=6)
plot(1)
if(doPDF) dev.off.pdf(file)
if(file.exists(file)) file.remove(file)
```

`doApply` *Functions for Iterating Over All Subjobs*

Description

`doLapply()` iterates over all subjobs (using the non-parallel `lapply()`). Similarly, but in parallel, for `doForeach` (based on CRAN package **foreach**'s `foreach()`), `doRmpi` (based on **Rmpi**'s `mpi.apply()`), `doMclapply` (based on **parallel**'s `mclapply()`), and `doClusterApply` (based on **parallel**'s `clusterApply()`).

`doRes.equal()` is simple convenience wrapper for `all.equal()`, for comparing two results (from the same `varlist` and `doOne` arguments) of the `do*` lapply-like functions above.

Usage

```
doLapply(vList, seed="seq", repFirst=TRUE,
       sfile=NULL, check=TRUE, doAL=TRUE, subjob.=subjob, monitor=FALSE,
       doOne, ...)
doForeach(vList, cluster=makeCluster(detectCores(), type="PSOCK"),
       cores=NULL, block.size = 1, seed="seq", repFirst=TRUE,
       sfile=NULL, check=TRUE, doAL=TRUE, subjob.=subjob, monitor=FALSE,
       doOne, extraPkgs=character(), exports=character(), ...)
doRmpi(vList,
      nslaves = if((sz <- Rmpi::mpi.universe.size()) <= 1) detectCores() else sz,
       load.balancing=TRUE, block.size = 1, seed="seq", repFirst=TRUE,
       sfile=NULL, check=TRUE, doAL=TRUE, subjob.=subjob, monitor=FALSE,
       doOne, exports=character(), ...)
doMclapply(vList, cores = if(.Platform$OS.type == "windows") 1 else detectCores(),
       load.balancing=TRUE, block.size = 1, seed="seq", repFirst=TRUE,
       sfile=NULL, check=TRUE, doAL=TRUE, subjob.=subjob, monitor=FALSE,
       doOne, ...)
```

```
doClusterApply(vList, cluster=makeCluster(detectCores(), type="PSOCK"),
       load.balancing=TRUE, block.size = 1, seed="seq", repFirst=TRUE,
       sfile=NULL, check=TRUE, doAL=TRUE, subjob.=subjob, monitor=FALSE,
       doOne, initExpr, exports=character(), ...)

doRes.equal(x, y, tol = 1e-15, ...)
```

Arguments

vList	a list of variable specifications. Each variable spec is itself a named list which must contain a "value" component.
cluster	cluster object, typically generated by makeCluster(). For doForeach(), this can be NULL as well, see Details below.
cores	the number of cores. For doForeach(), this can be NULL as well, see Details below.
nslaves	the number of workers for doRmpi, passed to package **Rmpi**'s mpi.spawn.Rslaves when no running workers are found.
load.balancing	logical indicating whether load balancing is used: **doRmpi()** mpi.applyLB() is used instead of mpi.apply(). **doMclapply()** here, mc.preschedule=!load.balancing determines load balancing. **doClusterApply()** clusterApplyLB() instead of clusterApply().
block.size	size of blocks of rows in the virtual grid which are computed simultaneously (load-balancing).
seed, repFirst	see subjob().
sfile, check, doAL	see saveSim().
subjob.	a function for computing a subjob (one row of the virtual grid). Typically subjob().
doOne	a user-supplied function for computing one row of the (physical) grid.
monitor	a logical or a function for producing "monitoring" output; the function argument list must contain the one of printInfo[["default"]].
extraPkgs	character vector of packages to be made available on the nodes.
exports	character vector of functions (for doForeach() and doClusterApply()) or objects (for doRmpi()) to export.
initExpr	expression initially evaluated on the cluster (can be missing).
...	additional arguments passed to subjob() (typically further passed on to doOne()), or, for doRes.equal(), to all.equal(*).
x,y	each a result of, say doLapply() which should be compared where sensible, i.e., the first three components "value","error","warning", using all.equal.
tol	passed to all.equal(*).

Details

See the vignette or references in simsalapar-package for how to use these functions.

For reasons to choose "MPI" as cluster type (if not on Windows), see the discussion starting at https://stat.ethz.ch/pipermail/r-sig-hpc/2013-April/001647.html.

For doForeach(), precisely one of cluster or cores has to be not NULL. This will determine whether the parallel computations are carried out on a cluster with multiple nodes or on a multi-core processor.

Value

The result of applying subjob() to all subjobs, converted with saveSim().

Author(s)

Marius Hofert and Martin Maechler.

See Also

subjob() for computing a subjob. doCallWE() for the return value of doOne(). .Random.seed for information about random number generators and seeds.

Examples

```
if(simsalapar:::doExtras()) { ## needs some CPU
  demo(robust.mean) # 512 simulations, differing block sizes, ...
}
```

---

doCallWE *Innermost Computation: Error Catching Version of do.call()*

Description

doCallWE() performs the innermost computation of the simulation study at hand. It is a version of do.call(f, argl, *), with care of catching and storing *both* error and warnings (via tryCatch.W.E()) and measures user time. This is useful in large(r) simulation studies.

mkTimer() returns a *function* to be passed as timer to doCallWE().

Usage

```
doCallWE(f, argl,
         timer = mkTimer(gcFirst=FALSE))

mkTimer(gcFirst)
```

Arguments

f	a function which given data and parameters, computes the statistic we are simulating.
argl	list of arguments for f().
timer	a function similar to system.time(); by default, measure user time in milliseconds.
gcFirst	logical, passed to system.time(), as it is called from the resulting function mkTimer().

Details

Note that gcFirst=FALSE is default for a good reason: if a call to doOne() is relatively fast, calling gc() every time is unnecessarily expensive and may completely dominate the overall simulation run time. For serious run time measurement, gcFirst=TRUE is preferable, as it ensures less variable timings, see system.time.

Value

doCallWE() returns a list with components

value	f(⟨argl⟩), if there was no error, NULL otherwise.
error	error message (see simpleError or stop()), NULL otherwise.
warning	warning message (see simpleWarning or warning()), NULL otherwise.
time	time, as measured by timer(); defaults to milliseconds without garbage collection.

Author(s)

Marius Hofert and Martin Maechler.

See Also

do.call, tryCatch.W.E.

Examples

```
set.seed(61)
L <- log(abs(rt(n=100, df = 1.5)))
r <- doCallWE(quantile, list(L, probs= 0.95))
## set timer for  "no timing" :
u <- doCallWE(quantile, list(L, probs= 0.95), timer = function(E) { E; NULL })
stopifnot(is.null(r$error),
  all.equal(r$value, quantile(L, 0.95)),
  identical(r[1:3], u[1:3]), is.null(u[["time"]]))
```

doCheck *Checking a User's doOne*

Description

`doCheck()` checks, if possible, a user's doOne() function for return objects of correct sizes.

Usage

```
doCheck(doOne, vList, nChks = ng, verbose = TRUE)
```

Arguments

`doOne`	a user-supplied `function` for computing one row of the (physical) grid.
`vList`	a `list` of variable specifications. Each variable spec is itself a named list which must contain a `"value"` component.
`nChks`	number of rows randomly picked from the (physical) grid which are used for a basic test of the evaluation and return value of `doOne()`.
`verbose`	`logical` indicating whether check output is displayed.

Value

None.

Author(s)

Marius Hofert and Martin Maechler.

Examples

```
## definition
doCheck
```

expr2latex *Translate 'plotmath' expressions to LaTeX*

Description

`expr2latex()` translates a "R graphics annotation" expression to the corresponding LaTeX one.

`escapeLatex()`, very similar to its original, `escape_latex()` from **fortunes**, escapes certain character combinations, such that the result can be used in LaTeX.

Usage

```
expr2latex(expr)
escapeLatex(x)
```

Arguments

expr	an R object of class expression or language, typically as from quote(...).
x	a character vector.

Details

The expr2latex() function is recursively rendering (sub) expressions, until it uses the internal renderAtom() for simple symbols (is.symbol).

We currently work with some tables of math annotation expressions, lifted from the corresponding C source of R itself. (Hidden in **simsalpar**'s namespace, we have AccentTable, BinTable, RelTable, Lgreek and Ugreek, currently.)

The current implementation is still incomplete.

Value

a character string with the LaTeX expression corresponding to "R graphics annotation" expression expr.

Author(s)

Martin Maechler.

See Also

plotmath for mathematical expressions to annotate R graphics.

toLatex() and its ftable method, toLatex().

Examples

```
expr2latex( quote( N[sim] ) )
expr2latex( quote( N[sim] ~ O(n) ) )
expr2latex( quote(x %notin% N) )
expr2latex( quote(x %+-% epsilon) )
expr2latex( quote(N[s*m^2]) )
expr2latex( quote( 2^{N[sim] - 3} ~~~ O(n^{n^2}) ) )

escapeLatex(c("#{positives}", "A | B"))
```

Description

From a variable specification list (varlist),

getEl() gets elements of a variable specification list that match the given variable type.

mkGrid() builds a grid, e.g., for parallel evaluation, basically by calling do.call(expand.grid, <list>).

mkNms() builds a list of names, e.g., to be used as dimnames for a corresponding simulation result array.

get.n.sim() extracts n.sim or returns 1 if it is not contained in vList.

set.n.sim() modifies or sets n.sim in vList.

get.nonGrids() extracts all variables not having type="grid" and returns n.sim the same as get.n.sim().

Usage

```
getEl       (vList, type = "ALL", comp = "value")
mkGrid      (vList)
mkNms       (vList, addNms = FALSE)
get.n.sim   (vList)
set.n.sim   (vList, n)
get.nonGrids(vList)
```

Arguments

vList	a list of variable specifications, typically resulting from varlist(). Each variable spec is itself a named list which must contain a "value" component.
type	character vector of variable type or types to restrict the selection to. The default, "ALL" implies no restriction and hence returns *all* variables.
comp	either a character string containing the component name to pick out or NA (in which case all components are picked out).
addNms	logical, specifying if the resulting names should be of the form <var>=<value> instead of just <value>.
n	for set.n.sim(): the value n.sim should be set to; an integer or NULL.

Details

These functions are useful when working with variable specification lists.

Value

For

getEl() a named list containing the selected components of those variables that match the provided type.

mkGrid() a data frame (data.frame).

mkNms() a named list of the same length() and with the same names() as vList.

get.n.sim() `n.sim` if it is contained in `vList`, 1 otherwise.

set.n.sim() the `varlist` `vList` with a *modified* `n.sim`.

get.nonGrids() `list` of length 2 containing the (possibly modified) `n.sim` and a `list` containing all variables not having `type="grid"`.

Author(s)

Marius Hofert and Martin Maechler.

See Also

`varlist`, for construction of variable lists. `expand.grid`, the "workhorse" of `mkGrid()`.

Examples

```
vList <-
 varlist(n.sim = list(type="N", expr = quote(N[sim]), value = 64),
         n     = list(type="grid",
                      value = c(20, 100, 500)), # sample sizes
         p     = list(type="grid",
                      value = c(3, 7, 15, 25)), # dimensions
         meth  = list(type="grid", expr = quote(italic(method)),
                      value = c("classical", "robust")))

getEl(vList, type="grid") # for those of type "grid", get all values
## for those of type "grid", get all components :
str(getEl(vList, type="grid", comp=NA))
stopifnot(identical(as(vList, "list"),
                    getEl(vList, type=c("N","grid"), comp = NA)))

(grd <- mkGrid(vList))
stopifnot(nrow(grd) == 3*4*2, ncol(grd) == 3)

getEl(vList)# -> all "value"s: the same as lapply(., `[[`, "value") :
stopifnot(identical(lapply(vList, `[[`, "value"),
                    getEl(vList)))
mkNms(vList)
mkNms(vList, addNms=TRUE)

get.n.sim(vl. <- set.n.sim(vList, NULL)) # 1
vl.$n.sim # NULL
set.n.sim(vl., 12)
```

`LEseeds` *Advancing .Random.seed for "L'Ecuyer-CMRG"*

Description

`LEseeds()` creates a list of advanced .Random.seed's for "L'Ecuyer-CMRG".

Usage

```
LEseeds(n)
```

Arguments

n	number of steps to advance .Random.seed.

Details

See, for example, Hofert and Mächler (2014) for how to use these functions.

Value

A list of length n containing the advanced .Random.seed's.

Author(s)

Marius Hofert and Martin Maechler.

See Also

.Random.seed for information about random number generators and seeds.

---

mayplot *Matrix-like Plot for Arrays up to Rank 5*

Description

Produces a matrix-like plot for arrays up to rank 5, using **grid** and **gridBase** which allows traditional **graphics**, optionally via a user specified panel function panel.

Usage

```
mayplot(x, vList, row.vars = NULL, col.vars = NULL,
        xvar, method = if(has.n.sim) "boxplot" else "lines",
       panel.first = NULL, panel.last = NULL,
       type = "l", pch = NULL, ylim = "global",
       log = "", do.legend = TRUE,
       spc = c(0.04/max(1,n.x-1), 0.04/max(1,n.y-1)),
       axlabspc=c(0.12, 0.08), labspc=c(0.04, 0.04),
       n.sim.spc = 0.06, auxcol = c("gray40", "gray78", "gray90", "white"),
       pcol = c("black", "blue", "red", "orange"), grid.lwd = 1.6, ax.lwd = 2,
       tx.cex = 1.2, leg.cex = 1, xlab = NULL, ylab = NA,
       do.n.sim = has.n.sim,
       verbose = getOption("verbose"), show.layout = verbose, ...)
```

Arguments

x	numeric array of 'rank' 5, i.e., length(dim(x)) == 5, with *named* dimnames; typically resulting from a call like getArray(doMclapply(..)).
vList	a list of variable specifications, see varlist and mkGrid.
row.vars	a dimension name of x, a string; this variable is plotted in the plot rows.
col.vars	a dimension name of x, a string; this variable is plotted in the plot columns.
xvar	dimension name of x, a string; this variable is plotted on the x axis of each sub-plot.
method	character string indicating the plot method used. Currently available are "boxplot" (the default if vList has n.sim) or "lines" (otherwise; type adjusts the type of lines used).
panel.first, panel.last	function or NULL (default). If specified, panel.first(x, y, col, ...) is called before and panel.last(x, y, col, ...) is called after the main plotting functions (think boxplot.matrix() and lines()) are called in each panel.
type	character indicating the type of plotting in the non-boxplot case; actually any of the types as in plot.default.
pch	logical indicating whether a plot symbol is to be used in the legend (default NULL determines this from type).
ylim	either string "global", "local", or a numeric vector, as for plot.default.
log	logical indicating if logarithmic scales should be used (in the individual plots).
do.legend	logical indicating if a legend should be added.
spc	dimensions (x, y) in "npc" for the space between sub-plots. The default uses a simple adaption to the number of sub-plots in each direction.
axlabspc	vector of length two containing the width of the y axis label and the height of the x axis label in "npc".
labspc	vector of length two containing the width of the box of the row labels and the height of the box of the column labels in "npc".
n.sim.spc	space for n.sim on the bottom right of the plot in "npc" (only if available).
auxcol	auxiliary colors; vector with four components: 1. color of axes and ticks 2. background color for the row and column labels 3. background color for the plots 4. color of grid lines
pcol	plot base colors. If more colors than the provided ones are required, colorRampPalette() is used.
grid.lwd	lwd for grid
ax.lwd	lwd for axes
tx.cex	cex for row and column labels
leg.cex	cex of legend text and n.sim if appropriate

xlab	x axis label (spanned over all plot columns); when NULL, the default is vList[[xvar]]$expr; to suppress, use NA.
ylab	y axis label (spanned over all plot rows): Typically a label for the "value" of the simulation.
do.n.sim	logical indicating whether n.sim is displayed on the bottom right of the plot (only if available).
verbose	logical indicating whether more information is displayed during plotting.
show.layout	logical indicating whether the grid layout is displayed.
...	optional arguments passed to panel().

Value

the layout, invisibly.

Author(s)

Marius Hofert and Martin Maechler.

See Also

matplot unit and grid.layout from package **grid**.

Examples

```
vLis <-
 varlist(d = list(type="grid", value = c(10, 100, 1000)),
         family=list(type="grid", value = c("Clayton", "Gumbel")),
         tau = list(type="grid", value = c(0.25, 0.5)),
         alpha = list(type="inner", value = c(0.95, 0.99, 0.999)))

iP <- c(4, 1:3)# <- permutation, putting alpha first
dNms <- mkNms(vLis)[iP]
## an array as from  x <- getArray( doMclapply(vLis, ..) ) :
x <- array(
    c(6.1981, 8.0478, 8.4265, 46.883, 74.359, 86.4394, 432.585, 743.27, 859.35,
      4.8508, 6.0286, 6.3965, 26.380, 35.132, 47.1517, 243.113, 311.36, 342.84,
      7.8546, 8.9769, 9.2199, 78.235, 89.493, 92.2875, 785.674, 893.63, 923.62,
      7.7164, 8.2866, 8.8169, 75.959, 82.806, 88.0626, 756.786, 831.65, 874.70),
    dim = sapply(dNms, length), dimnames = dNms)

mayplot(x, vLis, row.vars="family", col.vars="tau", xvar="alpha", log="y",
       ylab=bquote(widehat(VaR)[alpha]))
## the same, but no xlab and no ylab :
mayplot(x, vLis, row.vars="family", col.vars="tau", xvar="alpha", log="y", xlab=NA)
```

subjob *Subjob - Compute one Row of the Virtual Grid*

Description

subjob() computes one row of the virtual grid in a simulation study, provides several seeding methods, and sub-job monitoring (information about the sub-job just finished).

printInfo is a named list of functions optionally to be used as monitor in subjob() for printing information at the end of each sub-job.

Usage

```
subjob(i, pGrid, nonGrids, n.sim, seed, keepSeed = FALSE,
       repFirst = TRUE, doOne,
       timer = mkTimer(gcFirst=FALSE), monitor = FALSE, ...)

printInfo # or
# printInfo[["default"]]
```

Arguments

i	row number of the virtual grid. i.sim and j together determine i.
pGrid	"physical grid" of all combinations of variables of type "grid", as returned by mkGrid(<varlist>).
nonGrids	values of non-"grid"-variables (if provided, passed to doOne()), i.e., typically get.nonGrids(<varlist>)[["nonGrids"]].
n.sim	number of simulation replications.
seed	one of:

NULL .Random.seed remains untouched. If it does not exist, generate it by calling runif(1). This case typically leads to **non**-reproducible results.

numeric(n.sim) a numeric vector of length n.sim containing the seed for each simulation replications (same seed for each row in the (physical) grid; this ensures least variance across computations for the same replication). This case leads to reproducible results.

vector("list", n.sim) a list of length n.sim containing seeds (typically numeric vectors) for each of the n.sim simulation replications (same seed for each row in the (physical) grid). The seeds are assigned to .Random.seed in globalenv() and can thus be used for other random number generators such as "L'Ecuyer-CMRG", see set.seed(). This case leads to reproducible results.

NA .Random.seed remains untouched. If it does not exist, so be it. No fifth component is concatenated to the result of the doOne() call in this case even when keepSeed=TRUE (where in all other cases, the seed is appended as 5th component). This method typically leads to **non**-reproducible results.

character **string** a character string specifying a seeding method. Currently only "seq" in which case the seeds 1 to n.sim for the n.sim simulation replications are used. This is the default. Functionally, it is a special case of the "numeric(n.sim)" specification above (with seed = 1:n.sim) and hence leads to reproducible results.

keepSeed — logical indicating if .Random.seed should be appended to each return value of doCallWE() - unless seed = NA.

repFirst — logical; if TRUE (the default), all n.sim replications are computed for a row in the (physical) grid first, before the next row is considered; if FALSE, first all rows of the (physical) grid are computed for a fixed replicate until the next replicate is considered.

doOne — function for computing one row in the (physical) grid; must return a numeric vector, matrix, or array.

timer — a function similar to system.time(), passed to doCallWE().

monitor — logical or function indicating whether or how monitoring output is displayed. TRUE defaults to the printInfo[["default"]] function.

... — additional arguments passed to doOne().

Details

See the vignette or references in simsalapar-package for how to use these functions.

The case where seed is a numeric vector of length n.sim also leads to the same results no matter which variables are of type "grid" or "inner"; see demo(robust.mean) where this is tested. This is important to guarantee since one might want to change certain "inner" variables to "grid" variables due to load-balancing while computing the desired statistics based on the same seed (or generated data from this seed).

Value

printInfo *is* a named list of functions which produce output (cat(..)) containing information about the sub-job which subjob() has just finished. Note that components "gfile" ("global file") and "fileEach" each direct the monitoring output to *files*.

subjob() returns a vector of length five if keepSeed is true and seed is not NA, otherwise (also by default), of length four. The first four components contain the return value of doCallWE(). If keepSeed is true, the fifth component contains .Random.seed before the call of doCallWE() (for reproducibility).

Author(s)

Marius Hofert and Martin Maechler.

See Also

doCallWE(); .Random.seed for information about random number generators and seeds.

For examples of *implicit* use of subjob, see doLapply.

Examples

```
names(printInfo)# currently "default", "gfile", "fileEach"

str(printInfo, give.attr=FALSE)
## the functions in printInfo share a common environment() with utility functions:
ls.str(environment(printInfo$default))
if(FALSE) # show them all
as.list(environment(printInfo$default))
```

`toLatex-ftable` *Convert Flat Contingency Table (ftable) and VarLists to LaTeX Table*

Description

The `ftable` method of `toLatex()` converts an `ftable` to a LaTeX table via `tablines()`.

Analogously, the `varlist` method of `toLatex()` converts an `varlist` to a LaTeX table.

`fftable()` essentially calls `format.ftable()` and adds attributes `ncv` and `nrv` to the return object.

`tablines()` computes ingredients for converting a `character matrix` with attributes to a LaTeX table.

`cattablines()` is a small auxiliary function which creates rows of a LaTeX table from a given matrix.

Usage

```
## S3 method for class 'ftable'
toLatex(object, vList = NULL,
        x.escape = FALSE, exprFUN = expr2latex, escapeFUN = escapeLatex,
        align = NULL, booktabs = TRUE, head = NULL,
rsep = "\\\\", sp = if(booktabs) 3 else 1.25, rsep.sp = NULL,
csep = " & ", quote = FALSE, lsep=" \\textbar\\ ",
do.table = TRUE, placement = "htbp", center = TRUE,
fontsize = "normalsize", caption = NULL, label = NULL, ...)

## S3 method for class 'varlist'
toLatex(object,
col.vars = c("Variable", "expression", "type", "value"),
exprFUN = expr2latex, escapeFUN = escapeLatex,
align = NULL, booktabs = TRUE, head = NULL,
rsep = "\\\\", sp = if(booktabs) 3 else 1.25, rsep.sp = NULL, csep = " & ",
do.table = TRUE, placement = "htbp", center = TRUE,
fontsize = "normalsize", caption = NULL, label = NULL, ...)

fftable(x, lsep = " | ", quote = FALSE, method = "compact", ...)

tablines(x, align = NULL, booktabs = TRUE, head = NULL,
 rsep = "\\\\", sp = if(booktabs) 3 else 1.25, rsep.sp = NULL,
```

```
 csep = " & ", quote = FALSE)

cattablines(x, rsep = "\\\\", csep = " & ", include.rownames = TRUE)
```

Arguments

object	an ftable to be converted to a LaTeX table. This is accomplished via formatting it.
x	for fftable() an ftable object; for tablines() a character matrix with attributes nrv and ncv (as returned by fftable()) giving the number of row and column variables, respectively; for cattablines() a numeric or character matrix.
vList	a variable specification list see varlist.
x.escape	logical indicating if the "body" entries of the table should be escaped by espaceFUN(); if false, as by default, only the column and row variables are escaped.
exprFUN	a function, by default expr2latex, for transforming plotmath expressions to equivalent LaTeX strings.
escapeFUN	a function, by default escapeLatex which "escapes" each of its input character strings to valid LaTeX strings.
align	either a character (e.g., "*{3}{c} S[table-format=1.2]") or character vector (e.g., c("c", "c", "c", "S[table-format=1.2]")), or NULL (default).
booktabs	logical indicating whether a LaTeX table in the format of the LaTeX booktabs package is created (requires the LaTeX booktabs package loaded in the preamble).
head	either character a vector containing the lines of the header. NA do not construct a header. NULL construct a default header.
rsep	character to be inserted at the end of each row.
sp	numeric scaling factor for separating blocks of rows if rsep.sp is NULL.
rsep.sp	numeric of length equal to the number of different groups of rows minus one, giving the spaces (interpreted as pt) between different groups of rows. If NULL, a suitable default is constructed.
csep	character string for separating different cells in a row.
quote, lsep, method	see format.ftable() (R-3.0.0 or later).
col.vars	character vector of length 3 or 4 ("expression" can be omitted), specifying the column names.
do.table	logical indicating whether a LaTeX 'table' environment should be used at all.
placement	(if do.table:) character string containing a LaTeX table placement string such as "htbp".
center	logical indicating whether centering should happen.

fontsize	character string giving a fontsize (such as "tiny", "scriptsize", "footnotesize", "small", "normalsize", "large", "Large", "LARGE", "huge", or "Huge").
caption	(if do.table:) character string containing the table caption or NULL for no caption.
label	(if do.table:) character string containing the table label or NULL for no label.
include.rownames	logical indicating whether row names are included in the first column.
...	additional arguments passed to format.ftable().

Value

toLatex() returns an object as from wrapLaTable().

fftable() returns a formatted flat contingency table as returned by format.ftable() with added attributes ncv (number of column variables) and nrv (number of row variables).

tablines() a list with components

body	character vector of lines of the table body.
body.raw	character matrix of cells of the table body.
head	character vector of lines of the table head.
head.raw	character matrix of cells of the table head..
align	alignment string.
rsepcol	character vector containing the row separators (last entries of each row).

cattablines() outputs the formatted lines for copy-and-paste into a LaTeX table.

Author(s)

Marius Hofert and Martin Maechler.

References

see simsalapar-package.

See Also

wrapLaTable() for how to wrap the lines of a LaTeX table created by tablines() in a LaTeX table and tabular environment.

Examples

```
## Different table layouts for the same content
(ft1 <- ftable(Titanic, col.vars = 1:4))
(ft2 <- ftable(Titanic, row.vars = 1))
(ft3 <- ftable(Titanic, row.vars = 1:2))
(ft4 <- ftable(Titanic, row.vars = 1:3))
(ft5 <- ftable(Titanic, row.vars = 1:4))

## What tablines() returns
```

```
tablines(fftable(ft2))

## LaTeX (booktabs/non-booktabs) versions
toLatex(ft1, do.table=FALSE)
toLatex(ft1, booktabs=FALSE)
toLatex(ft1, method="col.compact")
toLatex(ft1)
toLatex(ft2)
toLatex(ft3)
toLatex(ft4)
toLatex(ft5, booktabs=FALSE)
toLatex(ft5, method="col.compact")
toLatex(ft5)

## ``poor-man's approach'' for creating lines of a LaTeX table
set.seed(271)
tab <- matrix(runif(6), ncol=3)
ftab <- formatC(tab, digits=4, format="f")
cattablines(ftab)
rownames(ftab) <- LETTERS[1:nrow(ftab)]
cattablines(ftab)
```

`tryCatch.W.E` *Catching and Storing Warnings and Errors Simultaneously*

Description

Catches and saves both warnings (`warning`) and errors (`stop`) and in the case of a warning, also the computed result.

Usage

```
tryCatch.W.E(expr)
```

Arguments

`expr`	expression to be evaluated, typically a function call.

Details

This function is particularly useful in large(r) simulation studies to check all computations and guarantee their correctness.

Value

`list` with components

`value`	value of `expr` *or* error message (see `simpleError` or `stop()`).
`warning`	warning message (see `simpleWarning` or `warning()`) or `NULL`.

Author(s)

Marius Hofert and Martin Maechler, based on hints from Luke Tierney and Bill Dunlap, see https://stat.ethz.ch/pipermail/r-help/2010-December/262626.html.

References

see simsalapar-package.

See Also

the base function tryCatch() and demo(error.catching). Also, doCallWE(), of which tryCatch.W.E() is the "workhorse".

Examples

```
## Adapted from demo(error.catching) :
str(r1 <- tryCatch.W.E( log( 2 ) ) )
str(r2 <- tryCatch.W.E( log(-1 ) ) )
str(r3 <- tryCatch.W.E( log("a") ) )
stopifnot(is.null(r1$warning),
          is.na  (r2$value),   inherits(r2$warning, "warning"),
          is.null(r3$warning), inherits(r3$value, "error"))
```

---

varlist *Variable Specification List - Generation and Class*

Description

Generate variable specification lists. These are objects of the formal (aka "S4") class "varlist". This class simply extends "namedList" and has a validity method (see validObject).

Usage

```
varlist(...)
dimnames2varlist(dmn)


## S4 method for signature 'varlist'
show(object)
```

Arguments

...	of the form nam1 = **list(....),** nam2 = **list(....),** **.....** namk = **list(....)**

	i.e, a "list" of variable specifications using "sub lists" `list(....)` = `list(value = <vv>, type = <tp>, expr = <e>)`, see the details and the examples below.
`dmn`	**named** `dimnames`, a `list`.
`object`	a `"varlist"` object.

Details

`value` is typically an atomic vector (`is.atomic`) or a `list`, e.g., of `function`s; in the latter case, typically with `names`.

`type` can be one of `"N"`, `"frozen"`, `"grid"`, or `"inner"`. In short:

`"N"` This type is reserved for a (single) variable named `n.sim` which provides the simulation replications; if it is not given, `n.sim` is implicitly treated as 1.

`"frozen"` Variables of this type remain fixed (they do not vary) throughout the whole simultion study. They affect the final result but do not appear as a *dimension* in the result array of the simulation study. This is the **default** `type` (apart from `n.sim` which defaults to `"N"`).

`"grid"` Variables of this type are used to build a (physical) grid (a `data.frame`) with number of rows equal to the product of the lengths of all variables of this type. The simulation will use this grid to iterate `n.sim` times over all of its rows for conducting the required computations. Conceptually, this corresponds to iterating over a *virtual grid* seen as `n.sim` copies of the (physical) grid pasted together. The computations for one row in this virtual grid form one sub-job. One can use one of `doLapply()`, `doForeach()`, `doRmpi()`, `doMclapply()`, or `doClusterApply()` to iterate over all sub-jobs.

`"inner"` Variables of this type are all dealt with within a sub-job for reasons of convenience, speed, load balancing etc.

The `dimnames2varlist()` functions creates a `varlist` from a *named* list of `character` vectors, typically resulting from `dimnames(tt)` of a `table` `tt`, see the Titanic example below.

For more details, see Hofert and Mächler (2014), and also the examples in `demo(package="simsalapar")`.

Value

an object of formal (aka "S4") class `"varlist"`.

Author(s)

Martin Maechler.

See Also

`namedList`; `getEl` for easy extraction of elements from a `"varlist"`.

The `toLatex` method for `varlists`, `toLatex.varlist`.

`doLapply()`, `doForeach()`, `doRmpi()`, `doMclapply()`, `doClusterApply()` for the functions to iterate over the virtual grid.

Examples

```
showClass("varlist")

vList <- varlist(
    n.sim = list(value = 1000, expr = quote(N[sim])), # type = N
    n     = list(type="grid", value = c(20, 100, 500)), # sample sizes
    meth  = list(type="grid", expr = quote(italic(method)),
                 value = c("classical", "robust")),
    alpha = list(value = 0.95)) # default type = "frozen"

str(vList)# note the default 'expr' for  n and alpha; and type of alpha

## For more extensive examples, see also
demo(package="simsalapar")

## coerce to simple list  .. and back :
lvl <- as(vList, "list")
stopifnot(identical(
    do.call(varlist, lvl),
    vList ))

## From a data.frame to a LaTeX table :
str(dimnames(Titanic))
vlTitan <- dimnames2varlist(dimnames(Titanic))
vlTitan  # default 'type = "grid"' here
toLatex(vlTitan)
```

`wrapLaTable` *Wrapper for a floating LaTeX Table*

Description

`wrapLaTable()` wraps (a table and tabular environment) around the lines of the body of a LaTeX table and utilizes `writeLines()` to write the LaTeX table.

Usage

```
wrapLaTable(x, align, do.table = TRUE, placement = "htbp", center = TRUE,
            fontsize = "normalsize", booktabs = TRUE,
            caption = NULL, label = NULL)
```

Arguments

`x`	a `character vector` containing the lines of the body of the table (for "booktabs" tables, everything strictly between `\midrule` and `\bottomrule`). A table header can be passed via attributes of `x`.
`align`	table columns alignment string (e.g., `"lcccS[table-format=1.2]"`, the notation of `"S[...]"` coming from the LaTeX package `siunitx`).

do.table	logical indicating whether a LaTeX 'table' environment should be used at all.
placement	(if do.table:) character string containing a LaTeX table placement string such as "htbp".
center	logical indicating whether centering should happen.
fontsize	character string giving a fontsize (such as "tiny", "scriptsize", "footnotesize", "small", "normalsize", "large", "Large", "LARGE", "huge", or "Huge").
booktabs	logical indicating whether a LaTeX table in the format of the LaTeX booktabs package is created.
caption	(if do.table:) character string containing the table caption or NULL for no caption.
label	(if do.table:) character string containing the table label or NULL for no label.

Details

Note that necessary LaTeX packages (such as tabularx) have to be loaded in the preambel of the corresponding .tex or .Rnw file.

Value

a "LaTeX table", of class "Latex" (where the print method uses writeLines()).

Author(s)

Marius Hofert.

References

see simsalapar-package.

See Also

toLatex() where it is used to create a LaTeX table.

Examples

```
ftab <- ftable(Titanic, row.vars = 1:2)
fftab <- fftable(ftab)
tlist <- tablines(fftab)

wrapLaTable(structure(tlist$body, head = tlist$head), align = tlist$align,
            caption="The Titanic data set.", label="tab:titanic")
```

Index