Package 'rngtools'

July 2, 2014

Maintainer Renaud Gaujoux <renaud@tx.technion.ac.il>

Author Renaud Gaujoux

Version 1.2.4

License GPL-3

Title Utility functions for working with Random Number Generators

Description This package contains a set of functions for working with Random Number Generators (RNGs). In particular, it defines a generic S4 framework for getting/setting the current RNG, or RNG data that are embedded into objects for reproducibility.
Notably, convenient default methods greatly facilitate the way current RNG settings can be changed.

URL https://renozao.github.io/rngtools

BugReports http://github.com/renozao/rngtools/issues

SCM github:renozao, r-forge

Depends R (>= 3.0.0), methods, pkgmaker (>= 0.20)

Imports stringr, digest

Suggests parallel, RUnit, knitr

Collate 'rngtools-package.r' 'format.R' 'RNG.R' 'RNGseq.R'

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2014-03-06 22:18:10

R topics documented:

---

checkRNG *Checking RNG Differences in Unit Tests*

Description

checkRNG checks if two objects have the same RNG settings and should be used in unit tests, e.g., with the **RUnit** package.

Usage

```
checkRNG(x, y = getRNG(), ...)
```

Arguments

x,y	objects from which RNG settings are extracted.
...	extra arguments passed to rng.equal.

Examples

```
# check for differences in RNG
set.seed(123)
checkRNG(123)
try( checkRNG(123, 123) )
try( checkRNG(123, 1:3) )
```

getRNG *Getting/Setting RNGs*

Description

getRNG returns the Random Number Generator (RNG) settings used for computing an object, using a suitable .getRNG S4 method to extract these settings. For example, in the case of objects that result from multiple model fits, it would return the RNG settings used to compute the best fit.

hasRNG tells if an object has embedded RNG data.

.getRNG is an S4 generic that extract RNG settings from a variety of object types. Its methods define the workhorse functions that are called by getRNG.

getRNG1 is defined to provide separate access to the RNG settings as they were at the very beginning of a whole computation, which might differ from the RNG settings returned by getRNG, that allows to reproduce the result only.

nextRNG returns the RNG settings as they would be after seeding with object.

setRNG set the current RNG with a seed, using a suitable .setRNG method to set these settings.

.setRNG is an S4 generic that sets the current RNG settings, from a variety of specifications. Its methods define the workhorse functions that are called by setRNG.

Usage

```
getRNG(object, ..., num.ok = FALSE, extract = TRUE,
  recursive = TRUE)

hasRNG(object)

.getRNG(object, ...)

getRNG1(object, ...)

nextRNG(object, ..., ndraw = 0L)

setRNG(object, ..., verbose = FALSE, check = TRUE)

.setRNG(object, ...)
```

Arguments

object	an R object from which RNG settings can be extracted, e.g. an integer vector containing a suitable value for .Random.seed or embedded RNG data, e.g., in S3/S4 slot rng or rng$noise.
...	extra arguments to allow extension and passed to a suitable S4 method .getRNG or .setRNG.

num.ok	logical that indicates if single numeric (not integer) RNG data should be considered as a valid RNG seed (TRUE) or passed to set.seed into a proper RNG seed (FALSE) (See details and examples).
extract	logical that indicates if embedded RNG data should be looked for and extracted (TRUE) or if the object itself should be considered as an RNG specification.
recursive	logical that indicates if embedded RNG data should be extracted recursively (TRUE) or only once (FASE).
ndraw	number of draws to perform before returning the RNG seed.
check	logical that indicates if only valid RNG kinds should be accepted, or if invalid values should just throw a warning. Note that this argument is used only on R >= 3.0.2.
verbose	a logical that indicates if the new RNG settings should be displayed.

Details

This function handles single number RNG specifications in the following way:

integers Return them unchanged, considering them as encoded RNG kind specification (see RNG). No validity check is performed.

real numbers If num.ok=TRUE return them unchanged. Otherwise, consider them as (pre-)seeds and pass them to set.seed to get a proper RNG seed. Hence calling getRNG(1234) is equivalent to set.seed(1234); getRNG() (See examples).

Think of a sequence of separate computations, from which only one result is used for the result (e.g. the one that maximises a likelihood): getRNG1 would return the RNG settings to reproduce the complete sequence of computations, while getRNG would return the RNG settings necessary to reproduce only the computation whose result has maximum likelihood.

Value

getRNG, getRNG1, nextRNG and setRNG usually return an integer vector of length > 2L, like .Random.seed.

getRNG and getRNG1 return NULL if no RNG data was found.

setRNG invisibly returns the old RNG settings as they were before changing them.

Methods

.getRNG signature(object = "ANY"): Default method that tries to extract RNG information from object, by looking sequentially to a slot named 'rng', a slot named 'rng.seed' or an attribute names 'rng'.

It returns NULL if no RNG data was found.

.getRNG signature(object = "missing"): Returns the current RNG settings.

.getRNG signature(object = "list"): Method for S3 objects, that aims at reproducing the behaviour of the function getRNG of the package getRNG.

It sequentially looks for RNG data in elements 'rng', noise$rng if element 'noise' exists and is a list, or in attribute 'rng'.

.getRNG `signature(object = "numeric")`: Method for numeric vectors, which returns the object itself, coerced into an integer vector if necessary, as it is assumed to already represent a value for `.Random.seed`.

getRNG1 `signature(object = "ANY")`: Default method that is identical to `getRNG(object, ...)`.

.setRNG `signature(object = "character")`: Sets the RNG to kind `object`, assuming is a valid RNG kind: it is equivalent to `RNGkind(object, ...)`. All arguments in `...` are passed to `RNGkind`.

.setRNG `signature(object = "numeric")`: Sets the RNG settings using `object` directly the new value for `.Random.seed` or to initialise it with `set.seed`.

See Also

`.Random.seed`, `showRNG`

Examples

```
# get current RNG settings
s <- getRNG()
head(s)
showRNG(s)

# get RNG from a given single numeric seed
s1234 <- getRNG(1234)
head(s1234)
showRNG(s1234)
# this is identical to the RNG seed as after set.seed()
set.seed(1234)
identical(s1234, .Random.seed)
# but if num.ok=TRUE the object is returned unchanged
getRNG(1234, num.ok=TRUE)

# single integer RNG data = encoded kind
head(getRNG(1L))

# embedded RNG data
s <- getRNG(list(1L, rng=1234))
identical(s, s1234)
# test for embedded RNG data
hasRNG(1)
hasRNG( structure(1, rng=1:3) )
hasRNG( list(1, 2, 3) )
hasRNG( list(1, 2, 3, rng=1:3) )
hasRNG( list(1, 2, 3, noise=list(1:3, rng=1)) )
head(nextRNG())
head(nextRNG(1234))
head(nextRNG(1234, ndraw=10))
obj <- list(x=1000, rng=123)
setRNG(obj)
rng <- getRNG()
runif(10)
set.seed(123)
```

```
rng.equal(rng)
# set RNG kind
old <- setRNG('Marsaglia')
# restore
setRNG(old)
# directly set .Random.seed
rng <- getRNG()
r <- runif(10)
setRNG(rng)
rng.equal(rng)

# initialise from a single number (<=> set.seed)
setRNG(123)
rng <- getRNG()
runif(10)
set.seed(123)
rng.equal(rng)
```

`rng.equal` *Comparing RNG Settings*

Description

`rng.equal` compares the RNG settings associated with two objects.

`rng1.equal` tests whether two objects have identical **initial** RNG settings.

Usage

```
rng.equal(x, y)

rng1.equal(x, y)
```

Arguments

`x`	objects from which RNG settings are extracted
`y`	object from which RNG settings are extracted

Details

These functions return `TRUE` if the RNG settings are identical, and `FALSE` otherwise. The comparison is made between the hashes returned by `RNGdigest`.

Value

`rng.equal` and `rng.equal1` return a `TRUE` or `FALSE`.

RNGseed *Directly Getting or Setting the RNG Seed*

Description

RNGseed directly gets/sets the current RNG seed .Random.seed. It can typically be used to backup and restore the RNG state on exit of functions, enabling local RNG changes.

RNGrecovery recovers from a broken state of .Random.seed, and reset the RNG settings to defaults.

Usage

```
RNGseed(seed)

RNGrecovery()
```

Arguments

seed	an RNG seed, i.e. an integer vector. No validity check is performed, so it **must** be a valid seed.

Value

invisibly the current RNG seed when called with no arguments, or the – old – value of the seed before changing it to seed.

Examples

```
# get current seed
RNGseed()
# directly set seed
old <- RNGseed(c(401L, 1L, 1L))
# show old/new seed description
showRNG(old)
showRNG()

# set bad seed
RNGseed(2:3)
try( showRNG() )
# recover from bad state
RNGrecovery()
showRNG()

# example of backup/restore of RNG in functions
f <- function(){
orng <- RNGseed()
 on.exit(RNGseed(orng))
RNGkind('Marsaglia')
runif(10)
}
```

```
sample(NA)
s <- .Random.seed
f()
identical(s, .Random.seed)
```

RNGseq *Generate Sequence of Random Streams*

Description

Create a given number of seeds for L'Ecuyer's RNG, that can be used to seed parallel computation, making them fully reproducible.

`RNGseq_seed` generates the – next – random seed used as the first seed in the sequence generated by `RNGseq`.

Usage

```
RNGseq(n, seed = NULL, ..., simplify = TRUE, version = 2)

RNGseq_seed(seed = NULL, normal.kind = NULL,
  verbose = FALSE, version = 2)
```

Arguments

`n`	Number of streams to be created
`seed`	seed specification used to initialise the set of streams using `RNGseq_seed`.
`simplify`	a logical that specifies if sequences of length 1 should be unlisted and returned as a single vector.
`...`	extra arguments passed to `RNGseq_seed`.
`normal.kind`	Type of Normal random generator. See `RNG`.
`verbose`	logical to toggle verbose messages
`version`	version of the function to use, to reproduce old behaviours. Version 1 had a bug which made the generated stream sequences share most of their seeds (!), as well as being not equivalent to calling `set.seed(seed); RNGseq_seed(NULL)`. Version 2 fixes this bug.

Details

This ensures complete reproducibility of the set of run. The streams are created using L'Ecuyer's RNG, implemented in R core since version 2.14.0 under the name `"L'Ecuyer-CMRG"` (see `RNG`).

Generating a sequence without specifying a seed uses a single draw of the current RNG. The generation of a sequence using seed (a single or 6-length numeric) a should not affect the current RNG state.

Value

a list of integer vectors (or a single integer vector if n=1 and unlist=TRUE).

a 7-length numeric vector.

See Also

RNGseq

Examples

```
RNGseq(3)
RNGseq(3)
RNGseq(3, seed=123)
# or identically
set.seed(123)
identical(RNGseq(3), RNGseq(3, seed=123))


RNGseq(3, seed=1:6, verbose=TRUE)
# select Normal kind
RNGseq(3, seed=123, normal.kind="Ahrens")
## generate a seed for RNGseq
# random
RNGseq_seed()
RNGseq_seed()
RNGseq_seed(NULL)
# fixed
RNGseq_seed(1)
RNGseq_seed(1:6)

# `RNGseq_seed(1)` is identical to
set.seed(1)
s <- RNGseq_seed()
identical(s, RNGseq_seed(1))
```

RNGstr *Formatting RNG Information*

Description

These functions retrieve/prints formated information about RNGs.

RNGtype returns the same type of values as RNGkind() (character strings), except that it can extract the RNG settings from an object. If object is missing it returns the kinds of the current RNG settings, i.e. it is identical to RNGkind().

showRNG displays human readable information about RNG settings. If object is missing it displays information about the current RNG.

RNGinfo is equivalent to RNGtype but returns a named list instead of an unnamed character vector.

RNGdigest computes a hash from the RNG settings associated with an object.

Usage

```
RNGstr(object, n = 7L, ...)

RNGtype(object, ..., provider = FALSE)

showRNG(object = getRNG(), indent = "#", ...)

RNGinfo(object = getRNG(), ...)

RNGdigest(object = getRNG())
```

Arguments

object	RNG seed (i.e. an integer vector), or an object that contains embedded RNG data. For RNGtype this must be either a valid RNG seed or a single integer that must be a valid encoded RNG kind (see RNGkind).
n	maximum length for a seed to be showed in full. If the seed has length greater than n, then only the first three elements are shown and a digest hash of the complete seed is appended to the string.
provider	logical that indicates if the library that provides the RNG should also be returned as a third element.
indent	character string to use as indentation prefix in the output from showRNG.
...	extra arguments passed to RNGtype.

Details

All functions can retrieve can be called with objects that are – valid – RNG seeds or contain embedded RNG data, but none of them change the current RNG setting. To effectively change the current settings on should use setRNG.

RNGstr returns a description of an RNG seed as a single character string.

RNGstr formats seeds by collapsing them in a comma separated string. By default, seeds that contain more than 7L integers, have their 3 first values collapsed plus a digest hash of the complete seed.

Value

a single character string

RNGtype returns a 2 or 3-long character vector.

Examples

```
# default is a 626-long integer
RNGstr()
# what would be the seed after seeding with set.seed(1234)
RNGstr(1234)
# another RNG (short seed)
RNGstr(c(401L, 1L, 1L))
```

```
# no validity check is performed
RNGstr(2:3)
# get RNG type
RNGtype()
RNGtype(provider=TRUE)
RNGtype(1:3)

# type from encoded RNG kind
RNGtype(107L)
# this is different from the following which treats 107 as a seed for set.seed
RNGtype(107)
showRNG()
# as after set.seed(1234)
showRNG(1234)
showRNG()
set.seed(1234)
showRNG()
# direct seeding
showRNG(1:3)
# this does not change the current RNG
showRNG()
showRNG(provider=TRUE)
# get info as a list
RNGinfo()
RNGinfo(provider=TRUE)
# from encoded RNG kind
RNGinfo(107)
# compute digest hash from RNG settings
RNGdigest()
RNGdigest(1234)
# no validity check is performed
RNGdigest(2:3)
```

`rngtools` *Utility functions for working with Random Number Generators*

Description

This package contains a set of functions for working with Random Number Generators (RNGs). In particular, it defines a generic S4 framework for getting/setting the current RNG, or RNG data that are embedded into objects for reproducibility.

Details

Notably, convenient default methods greatly facilitate the way current RNG settings can be changed.

Examples

```
showRNG()
s <- getRNG()
```

```
RNGstr(s)
RNGtype(s)

# get what would be the RNG seed after set.seed
s <- nextRNG(1234)
showRNG(s)
showRNG( nextRNG(1234, ndraw=10) )

# change of RNG kind
showRNG()
k <- RNGkind()
k[2L] <- 'Ahrens'
try( RNGkind(k) )
setRNG(k)
showRNG()
# set encoded kind
setRNG(501L)
showRNG()

# use as set seed
setRNG(1234)
showRNG()
r <- getRNG()

# extract embedded RNG specifications
runif(10)
setRNG(list(1, rng=1234))
rng.equal(r)

# restore default RNG (e.g., after errors)
RNGrecovery()
showRNG()
```

Index