# Package 'rlecuyer'

March 24, 2023

**Version** 0.3-7

**Date** 2023-03-23

**Title** R Interface to RNG with Multiple Streams

**Description** Provides an interface to the C implementation of the random number generator with multiple independent streams developed by L'Ecuyer et al (2002). The main purpose of this package is to enable the use of this random number generator in parallel R applications.

**License** GPL (>= 2)

**URL** http://www.iro.umontreal.ca/~lecuyer/myftp/papers/streams00.pdf

**NeedsCompilation** yes

**Author** Hana Sevcikova [aut, cre],
Tony Rossini [aut],
Pierre L'Ecuyer [cph] (author of the underlying C code)

**Maintainer** Hana Sevcikova <hanas@uw.edu>

**Repository** CRAN

**Date/Publication** 2023-03-24 20:00:02 UTC

## R topics documented:

---

`rlecuyer-package` *R Interface to Random Number Generator with Multiple Streams*

---

## Description

Provides an interface to the C implementation of the random number generator (RNG) with multiple independent streams developed by L'Ecuyer et al (2002). The package enables to use this random number generator in parallel R applications.

## Details

When the **rlecuyer** package is loaded, the L'Ecuyer RNG is initialized by creating a global table object (`.lec.Random.seed.table`) which allows to keep information and account for multiple random number streams that the user can create. The individual streams are identified by names which must be unique. The workflow starts with initializing the RNG with a seed via `.lec.SetPackageSeed` and creating one or more streams via `.lec.CreateStream`.

When the RNG is deployed in a parallel application, one stream can be used for generating all random numbers (RNs) on one node, or all RNs generated within one task. The master node would hold the global table object, initialize it with a package seed (via `.lec.SetPackageSeed`), create the amount of streams that are needed for the application (via `.lec.CreateStream`) and send each worker information about the stream assigned to it. Alternatively, all workers could be initialized with an identical global table and identical streams, and the master node could be only sending the identifiers of the streams on which each worker should operate. If streams are assigned to tasks instead of nodes, one can assure reproducibility regardless of the number of nodes the application is running on, or regardless if it is run sequentially or in parallel.

To generate RNs from a particular stream, start with the function `.lec.CurrentStream`. This will assure that any subsequent call to standard R functions that use RNs (e.g. `runif` or `rnorm` will draw from the current stream. `.lec.CurrentStreamEnd` unsets the stream. Thus, by using these two functions in pair, one can switch between different streams. Alternatively, if drawing from a uniform distribution is sufficient, by using the function `.lec.uniform` one can omit the CurrentStream and CurrentStreamEnd pair, as the `.lec.uniform` function includes the name of the stream to draw from.

Each stream is given by its current state (see `SetPackageSeed` for description) that can be viewed by `.lec.WriteStateFull`. To reset a stream to its initial state, use `.lec.ResetStartStream`. To extract the curent state of a stream, one can use `.lec.GetState`.

Other useful functions are available. For example, one can advance the state of a stream by given number of steps via the `.lec.AdvanceState` function. Function `.lec.GetStreams` allows to retrieve names of all streams in the global table. To delete a stream from the table, use `.lec.DeleteStream`. If there is a need to delete the whole table of streams and initialize a new one, one can use `.lec.init` and `.lec.exit`.

## References

P. L'Ecuyer, R. Simard, E.J.Chen and W.D.Kelton: An Object-Oriented Random-Number Package With Many Long Streams and Substreams; Operations Research, vol. 50, nr. 6, 2002.

## Examples

```
# Initialize the RNG with package seed
seed <- rep(85424, 6)
.lec.SetPackageSeed(seed)

# Create 5 streams
nstream <- 5
snames <- as.character(1:nstream) # unique stream identifiers
.lec.CreateStream(snames)

# Generate 10 RNs from each of the 5 streams
# (each of the iterations could run on a different node)
rns <- NULL
for (i in 1:nstream) {
    old.kind <- .lec.CurrentStream(snames[i])
    rns <- rbind(rns, runif(10))
   .lec.CurrentStreamEnd(old.kind)
}
rns

# Reproduce results on the second stream
.lec.ResetStartStream("2")
rns2 <- .lec.uniform("2", 10)
all(rns2 == rns[2,])

# Reproduce the last three RNs on stream 5
.lec.ResetStartStream("5")
.lec.AdvanceState("5", 0, 7) # move the state by 7 steps
rns5p <- .lec.uniform("5", 3)
all(rns5p == rns[5, 8:10])
```

---

`AdvanceState` *Advance the state of a stream*

---

## Description

`.lec.AdvanceState` advances the state of a stream by $n$ steps (see bellow).

## Usage

```
.lec.AdvanceState (name, e, c)
```

### Arguments

| | |
|---|---|
| name | name of the stream. |
| e, c | if $e > 0$ then $n = 2^e + c$; if $e < 0$ then $n = -2^{-e} + c$; if $e = 0$ then $n = c$. |

### Details

.lec.AdvanceState is a wrapper function for the C function RngStream_AdvanceState (L'Ecuyer et al, 2002).

### Value

None.

### References

P. L'Ecuyer, R. Simard, E.J.Chen and W.D.Kelton: An Object-Oriented Random-Number Package With Many Long Streams and Substreams; Operations Research, vol. 50, nr. 6, 2002.

---

## CreateStream *Spawn new streams*

---

### Description

.lec.CreateStream creates new streams of random numbers. .lec.StreamExists checks the existence of a stream.

### Usage

```
.lec.CreateStream (names)
.lec.StreamExists (name)
```

### Arguments

| | |
|---|---|
| names | a character string or a vector of character strings naming the streams to be created. The argument must be provided and the names must be unique within the set of existing streams. If for one i a stream of the name names[i] already exists, its state is replaced by the state of the new created stream. |
| name | name of stream |

### Details

.lec.CreateStream is a wrapper function for the C function RngStream_CreateStream (L'Ecuyer et al, 2002). The state of the created stream returned by the C function is stored in the global object .lec.Random.seed.table.

.lec.StreamExists returns TRUE if the stream is found in .lec.Random.seed.table, otherwise FALSE.

## Value

`.lec.StreamExists` returns `TRUE` or `FALSE`.

## References

P. L'Ecuyer, R. Simard, E.J.Chen and W.D.Kelton: An Object-Oriented Random-Number Package With Many Long Streams and Substreams; Operations Research, vol. 50, nr. 6, 2002.

## Examples

```
nstreams <- 10       # number of streams
names <- paste("mystream",1:nstreams,sep="")
.lec.CreateStream(names)
.lec.WriteStateFull(names)
```

---

# CurrentStream *Set/unset the current stream*

---

## Description

`.lec.CurrentStream` sets the current stream for usage with the standard R functions for generating random numbers such as `runif` or `rnorm`. `.lec.CurrentStreamEnd` unsets it.

## Usage

```
.lec.CurrentStream (name)
.lec.CurrentStreamEnd (kind.old = c("Mersenne-Twister",
                          "Kinderman-Ramage"))
```

## Arguments

| | |
|---|---|
| `name` | a character string giving the name of the stream. |
| `kind.old` | a length 2 character vector, the old rng kinds (possiblly returned by `.lec.CurrentStream`). |

## Details

`.lec.CurrentStream` sets the RNGkind to user-defined. All succeeding calls of R built-in generators will generate random numbers from the stream `name`, until `.lec.CurrentStreamEnd` is called. `.lec.CurrentStreamEnd` updates the RNG state of the stream `name` in the table `.lec.Random.seed.table` and sets the RNGkind to `kind.old`. These two functions are meant to be always used as a pair. Thus, one can arbitrarily switch generating between different streams.

## Value

`.lec.CurrentStream` returns a two-element character vector of the RNG and normal kinds in use before the call. `.lec.CurrentStreamEnd` returns a character string giving the name of the unset current stream.

## Examples

```
nstreams <- 10       # number of streams
names <- paste("mystream",1:nstreams,sep="")
.lec.CreateStream(names)
for (i in 1:nstreams) {  # generate 10 RNs from each stream
   old.kind <- .lec.CurrentStream(names[i])
   print(paste("stream no.",i))
   print(runif(10))
   .lec.CurrentStreamEnd(old.kind)
}
```

---

# DeleteStream *Remove streams*

---

## Description

`.lec.DeleteStream` removes streams from the global state table.

## Usage

```
.lec.DeleteStream (names)
```

## Arguments

| | |
|---|---|
| `names` | a character string or a vector of character strings naming the streams to be deleted. |

## Details

All streams given in the argument `names` are removed from the table `.lec.Random.seed.table`.

## Value

None.

---

# GetState *Return current state of a stream*

---

## Description

Returns current state (Cg values) of the stream `name`.

## Usage

```
.lec.GetState (name)
```

### Arguments

| | |
|---|---|
| name | a character string giving the name of the stream. |

### Value

a vector of six integer values that identifies the current state of the stream.

### See Also

SetPackageSeed

---

## GetStreams *Return names of existing streams*

---

### Description

Returns names of existing streams stored in .lec.Random.seed.table.

### Usage

```
.lec.GetStreams ()
```

### Value

a vector of character strings.

---

## IncreasedPrecis *Switch between 32 and 53 bits of resolution*

---

### Description

Switch between 32 and 53 bits of resolution as described in L'Ecuyer et al (2002).

### Usage

```
.lec.IncreasedPrecis (name, incp=FALSE)
```

### Arguments

| | |
|---|---|
| name | name of the stream. |
| incp | see L'Ecuyer et al (2002). |

### Details

.lec.IncreasedPrecis is a wrapper function for the C function RngStream_IncreasedPrecis.

## Value

None.

## References

P. L'Ecuyer, R. Simard, E.J.Chen and W.D.Kelton: An Object-Oriented Random-Number Package With Many Long Streams and Substreams; Operations Research, vol. 50, nr. 6, 2002.

---

`init` *Initialization and Cleaning*

---

## Description

Initialize and remove the RNG

## Usage

```
.lec.init()
.lec.exit()
```

## Details

The package uses a global table object `.lec.Random.seed.table` to keep information about the current state of the streams. Functions `.lec.init` creates this global object and function `.lec.exit` deletes it. However, in most cases these two functions will not be needed, as the table is automatically created when the package is loaded and it is deleted when the package is unloaded. If there is however a need to explicitely delete the table of streams and create a new empty one, these two functions can be used for that purpose.

`.lec.init` initializes the workspace: removes old and creates new global object `.lec.Random.seed.table`. It also allocates memory for the current stream used by `.lec.CurrentStream`.

`.lec.exit` removes the global object `.lec.Random.seed.table` and frees memory used for the current stream.

---

`ResetStream` *Reset the state of a stream*

---

## Description

Resets the state of a stream to its initial state, beginning of the current substream or beginning of the next substream.

### Usage

```
.lec.ResetNextSubstream(name)
.lec.ResetStartStream(name)
.lec.ResetStartSubstream(name)
```

### Arguments

| | |
|---|---|
| name | a character string giving the name of the stream. |

### Details

`.lec.ResetNextSubstream` reinitializes the stream to the beginning of its next substream.

`.lec.ResetStartStream` reinitializes the stream to its initial state.

`.lec.ResetStartSubstream` reinitializes the stream to the beginning of its current substream.

### Value

None.

### See Also

SetPackageSeed

---

## SetAntithetic *Switch between U and 1-U variates*

---

### Description

Switches between $U$ and $1 - U$ variates.

### Usage

```
.lec.SetAntithetic (name, anti=FALSE)
```

### Arguments

| | |
|---|---|
| name | name of the stream. |
| anti | if `anti=TRUE` then antithetic variates are generated (i.e. $1-U$), until this function is called again with `anti=FALSE`. |

### Value

None.

SetPackageSeed *Set RNG Seed*

## Description

Set the initial seed of the package or stream.

## Usage

```
.lec.SetPackageSeed(seed)
.lec.SetSeed (name, seed)
```

## Arguments

| | |
|---|---|
| name | a character string giving the name of the stream. |
| seed | a vector of six integers. If it is shorter, the seed is extended to the length of 6 by default values 12345. If it is longer, it is truncated to the length of 6 by eliminating the last elements. |

## Details

.lec.SetPackageSeed sets the the starting state of the next stream to be created. If there are no streams yet, it is the initial seed of the RNG. .lec.SetSeed sets the seed of a specific stream.

A state of each stream is given by three integer vectors of length 6: Ig gives the initital state of the stream, Bg gives the starting state of the substream that contains the current state, Cg gives the current state. Function .lec.SetPackageSeed sets Cg, Bg and Ig to the value of seed. Function .lec.SetSeed sets Ig to seed. L'Ecuyer recommends to use the ResetStream functions instead of SetSeed.

## Value

The (possibly modified) seed that has been used.

## See Also

ResetNextSubstream

## Examples

```
# Set the seed of the first stream
.lec.SetPackageSeed(1:6)

# Create the first stream and print out its state
.lec.CreateStream("A")
.lec.WriteStateFull("A")

# Create two more streams
.lec.CreateStream(c("B", "C"))
```

```
.lec.WriteStateFull(c("A", "B", "C"))

# Cet the seed of the next stream and create it
.lec.SetPackageSeed(rep(5678, 6))
.lec.CreateStream("D")
.lec.WriteStateFull(c("A", "B", "C", "D"))
```

---

`uniform` *Generate random numbers*

---

## Description

`.lec.uniform` generates $U(0, 1)$ random numbers.

`.lec.uniform.int` generates random numbers from the discrete uniform distribution over integers.

## Usage

```
.lec.uniform (name, n = 1)

.lec.uniform.int (name, n = 1, a = 0, b = 10)
```

## Arguments

| | |
|---|---|
| `name` | name of the stream. |
| `n` | number of random numbers to be generated. |
| `a,b` | interval from which the integer random numbers should be generated. |

## Details

`.lec.uniform` and `.lec.uniform.int`, respectively, are wrapper functions for the C functions `RngStream_RandU01` and `RngStream_RandInt`, respectively (L'Ecuyer et al, 2002).

Note: Since the stream is here identified by `name`, there is no need for using the `CurrentStream` pair.

## Value

A vector of $n$ random numbers.

## References

P. L'Ecuyer, R. Simard, E.J.Chen and W.D.Kelton: An Object-Oriented Random-Number Package With Many Long Streams and Substreams; Operations Research, vol. 50, nr. 6, 2002.

## See Also

`.lec.CurrentStream`

## Examples

```
nstreams <- 10       # number of streams
seed<-rep(1,6)
.lec.SetPackageSeed(seed)
names <- paste("mystream",1:nstreams,sep="")
.lec.CreateStream(names)
for (i in 1:nstreams)   # generate 10 RNs from each stream
      print(.lec.uniform(names[i],10))
.lec.DeleteStream(names)
```

---

`WriteState` *Output of the current state of streams*

---

## Description

`.lec.WriteState` writes the current state of given streams (Cg values).

`.lec.WriteStateFull` writes the values of all internal state variables of given streams.

## Usage

```
.lec.WriteState (names)

.lec.WriteStateFull (names)
```

## Arguments

| | |
|---|---|
| `names` | a character string or a vector of character strings naming the streams. |

## Value

None

## See Also

`SetPackageSeed` for description of a state of a stream and examples

# Index