Package ‘harvestr’
August 29, 2016

Type Package
Title A Parallel Simulation Framework
Version 0.7.1
Date 2016-08-29
Author Andrew Redd
Maintainer Andrew Redd <andrew.redd@hsc.utah.edu>
Description Functions for easy and reproducible simulation.
License GPL (>= 2)
VignetteBuilder knitr
Imports parallel, plyr, digest, foreach, stats
Suggests testthat, dostats, doParallel, MCMCpack, knitr, boot, withr
 ’harvestr-package.R’ ’option.R’ ’utils.R’
RoxygenNote 5.0.1
NeedsCompilation no
Repository CRAN
Date/Publication 2016-08-29 18:36:54

R topics documented:

bale .................................................. 2
called_from ........................................ 2
farm .................................................... 3
gather .................................................. 4
getAttr ............................................... 4
harvest ............................................... 5
harvestr ............................................. 5
Interactive .......................................... 7
is_seeded ........................................... 7
noattr ............................................... 7
plant .................................................. 8
called_from

Description

Test if a function was called from others.

Usage

called_from(..., FUNS = list(...))

is_knitting()

Arguments

... functions to pass
FUNS functions as a list

bale

Combine results into a data frame

Description

Combine results into a data frame

Usage

bale(l, .check = T)

Arguments

l a list, from a harvestr function.
.check should checks be run on the object.

See Also

ldply
Farm

**Functions**

- **is_knitting**: Determine if the function is called while knitting a document

---

**Farm**

*Evaluate an expression for a set of seeds.*

---

**Description**

For each seed, set the seed, then evaluate the expression. The `farm` function is used to generate data. The seeds for the state of the random number generator is stored in the attribute `'ending.seed'`, and will be used by `harvestr` functions for any other random number generation that is needed.

**Usage**

```r
farm(seeds, expr, envir = parent.frame(), ..., 
    cache = getOption("harvestr.use.cache", defaults$cache()), 
    time = getOption("harvestr.time", defaults$time()), 
    parallel = getOption("harvestr.parallel", defaults$parallel()), 
    progress = getOption("harvestr.progress", defaults$progress()))
```

**Arguments**

- **seeds**: a list of seeds can be obtained through `gather`
- **expr**: an expression to evaluate with the different seeds.
- **envir**: an environment within which to evaluate `expr`.
- **...**: extra arguments
- **cache**: should cached results be used or generated?
- **time**: should results be timed?
- **.parallel**: should the computations be run in parallel?
- **.progress**: Show a progress bar?

**See Also**

Other harvest: `gather, harvest, plant, sprout`
gather

Gather independent seeds.

Description

Collect seeds representing independent random number streams. These seeds can them be used in farm or plant.

Usage

gather(x, seed = get.seed(), ..., .starting = F)

Arguments

x number of seeds, or an object with seeds to gather
seed a seed to use to set the seed, must be compatible with "L'Ecuyer-CMRG"
... passed on
.starting if TRUE starting seeds with be gathered rather than ending seeds.

See Also

RNG
Other harvest: farm, harvest, plant, sprout

getAttr

Retrieve an attribute or a default if not present.

Description

Behaves similar to getOption, but is a simple wrapper for attr.

Usage

getAttr(object, name, default = NULL)

Arguments

object An R Object.
name Name of the Attribute
default The value if the attribute is not set or NULL
harvest

Harvest the results.

Description

Harvest the results.

Usage

harvest(.list, fun, ..., time = getOption("harvest.rtime", defaults$time()),
          .parallel = getOption("harvest.rparallel", defaults$parallel()),
          .progress = getOption("harvest.rprogress", defaults$progress()))

Arguments

- .list: a list of data.frames See details.
- fun: a function to apply
- ...: passed to fun
- time: should results be timed?
- .parallel: should the computations be run in parallel?
- .progress: Show a progress bar?

Details

harvest is functionaly equivalent to llply, but takes on additional capability when used with the other functions from this package. When an object comes from withseed the ending seed is extracted and used to continue evaluation.

See Also

Other harvest: farm, gather, plant, sprout

harvestr

A Simple Reproducible Parallel Simulation Framework

Description

harvestr package
Caching

The functions in harvestr can cache results for faster and interruptible simulations. This option defaults to \texttt{FALSE} but can be chosen by specifying the cache parameter in any of the functions that produce results.

The caching is performed by saving a RData file in a specified caching directory. The default directory is named "harvestr-cache" and resides under the \texttt{working directory}. This can be specified by setting the \texttt{harvestr.cache.dir} option. Files in this directory use file names derived from hashes of the expression to evaluate. Do not modify the file names.

Options

The following options control behavior and default values for harvestr.

1. \texttt{harvestr.use.cache} = \texttt{FALSE} - Should results be cached for fault tolerance and accelerated reproducibility?
2. \texttt{harvestr.cache.dir} = "harvestr-cache" - The directory to use for storing cached results.
3. \texttt{harvestr.time} = \texttt{FALSE} - Should results be timed?
4. \texttt{harvestr.use.try} = \texttt{!interactive()} - Should the vectorized calls use try to increase fault tolerance?
5. \texttt{harvestr.try.silent} = \texttt{FALSE} - Should try be run silently?
6. \texttt{harvestr.try.summary} = \texttt{TRUE} - Print a result if errors were found?
7. \texttt{harvestr.parallel} - Run results in parallel? Default is to run in parallel if a parallel backend is registered and the call is the uppermost harvestr call.
8. \texttt{harvestr.progress} - Use a progress bar? Default is to show a bar in interactive mode for top level call, but the type is platform dependent.

Author(s)

Andrew Redd <amredd_at_gmail.com>

The harvestr package is a framework for parallel reproducible simulations.

The functions to know about are:

1. \texttt{gather} - which gathers parallel seeds.
2. \texttt{farm} - which uses the saved seeds from gather to replicate an expression, once for each seed.
3. \texttt{harvest} - which uses objects from farm, that have saved seed attributes, to continue evaluation from where farm finished.
4. \texttt{reap} - is used by harvest for a single item
5. \texttt{plant} - is used to set seeds for a list of predefined objects so that harvest can be used on it.
6. \texttt{sprout} - Generate independent sub-streams.
7. \texttt{graft} - Replicate and object in independent substreams of random numbers.
Interactive

Smarter interactive test

Description
This is a smarter version of `interactive`, but also excludes cases inside knit or in startup `.First`, or others specified in dots. You can also specify functions to exclude in the option `harvestr::Interactive::exclude`.

Usage
`Interactive(exclude.calls = getOption("harvestr::Interactive::exclude"))`

Arguments
- `exclude.calls` functions to pass to `called_from`

is_seeded

Check if an object or list of objects has seed attributes

Description
Check if an object or list of objects has seed attributes

Usage
`is_seeded(x)`

Arguments
- `x` an object or list to check

noattr

Strip attributes from an object.

Description
Strip attributes from an object.

Usage
`noattr(x)`

Arguments
- `x`, any object
See Also

attributes

---

**plant**

Assign elements of a list with seeds

---

**Description**

The function `plant` assigns each element in list set seed. This will replace and ending seeds values already set for the objects in the list.

The `graft` function replicates an object with independent substreams. The result from `graft` should be used with `harvest`.

**Usage**

```r
plant(.list, seeds = gather(length(.list), ...), ...)
graft(x, n, seeds = sprout(x, n))
```

**Arguments**

- `.list` a list to set seeds on
- `seeds` to plant from `gather` or `sprout`
- `...` passed to `gather` to generate seeds.
- `x` an objects that already has seeds.
- `n` number of seeds to create

See Also

Other harvest: `farm`, `gather`, `harvest`, `sprout`

---

**plow**

Apply over rows of a data frame

---

**Description**

Apply over rows of a data frame

**Usage**

```r
plow(df, f, ..., seed = get.seed(), seeds = gather(nrow(df), seed = seed),
     time = getOption("harvest\n.time", defaults$time()),
     .parallel = getOption("harvest\n.parallel", defaults$parallel()),
     .progress = getOption("harvest\n.progress", defaults$progress()))
```
**reap**

**Arguments**
- **df**: a data frame of parameters
- **f**: a function
- **...**: additional parameters
- **seed**: passed to `gather` to generate seeds.
- **seeds**: seeds to use.
- **time**: should results be timed?
- **.parallel**: should the computations be run in parallel?
- **.progress**: Show a progress bar?

**Value**
a list with `f` applied to each row of `df`.

---

**Description**
The `reap` function is the central function to harvest. It takes an object, `x`, extracts the previous seed, i.e., state of the random number generator, sets the seed, and continues any evaluation. This creates a continuous random number stream, that is completely reproducible.

**Usage**

```r
reap(x, fun, ..., hash = digest(list(x, fun, ..., source = "harvestr::reap"), "md5"), cache = getOption("harvestr.use.cache", defaults$cache()),
    cache.dir = getOption("harvestr.cache.dir", defaults$cache.dir()),
    time = getOption("harvestr.time", defaults$time()),
    use.try = getOption("harvestr.use.try", !interactive()))
```

**Arguments**
- **x**: an object
- **fun**: a function to call on object
- **...**: passed onto function
- **hash**: hash of the list to retrieve the cache from.
- **cache**: use cache, see Caching in `harvestr`
- **cache.dir**: directory for the cache.
- **time**: should results be timed?
- **use.try**: Should the call be wrapped in `try`?
sprout

Create substreams of numbers based of a current stream.

**Description**

Seeds from `gather` can be used to generate another set of independent streams. These seeds can be given to `graft`.

**Usage**

`sprout(seed, n)`

**Arguments**

- `seed`: a current random number stream compatible with `nextRNGSubStream`
- `n`: number of new streams to create.

**Details**

As a convenience seed can be an object that has a seed attached, ie. the result of any `harvestr` function.

**See Also**

- `nextRNGSubStream`
- Other harvest: `farm, gather, harvest, plant`

---

**total_time**

retrieve the total time for a simulation

**Description**

retrieve the total time for a simulation

**Usage**

`total_time(x)`

**Arguments**

- `x`: a list from `harvest`
use_method

Use a reference class method

Description

Use a reference class method

Usage

use_method(method, ...)

Arguments

method name of the method to call
... additional arguments to pass along

Value

a function that calls the designated method

See Also

ReferenceClasses

Examples

library(harvestr)
library(plyr)
mr <- setRefClass("HelloWorld",
  fields = list(
    x = 'integer',
    name = 'character'),
  methods = list(
    hello = function(){
      invisible(name)
    },
    times = function(y){
      x*y
    },
    babble = function(n){
      paste(sample(letters), collapse='')
    }
  )
)
p <- data.frame(x=as.integer(1:26), name=letters, stringsAsFactors=FALSE)
# create list of objects
objs <- mply(p, mr$new)
# plant seeds to prep objects for harvest
objs <- plant(objs)
withseed

Do a computation with a given seed.

Description
Do a computation with a given seed.
Safe version of retrieving the .Random.seed
Get or Set Current Seed - Safe Version

Usage
```
withseed(seed, expr, envir = parent.frame(),
  cache = getOption("harvestr.use.cache", defaults$cache),
  cache.dir = getOption("harvestr.cache.dir", defaults$cache.dir),
  time = getOption("harvestr.time", defaults$time))
```

get.seed()
replace.seed(seed, delete = TRUE)
GetOrSetSeed()

Arguments
- `seed` a valid seed value
- `expr` expression to evaluate.
- `envir` the environment to evaluate the code in.
- `cache` should results be cached or retrieved from cache.
- `cache.dir` Where should cached results be saved to/retrieve from.
- `time` should results be timed?
- `delete` logical to delete if seed is null.

Details
Compute the expr with the given seed, replacing the global seed after computations are finished.
Does not replace the global .Random.seed
Replaces the .Random.seed with seed unless seed is null, then it will delete the .Random.seed if delete=T
Always returns a valid seed. Useful for grabbing a seed used to generate a random object.
withseed

Value

the .Random.seed if defined, otherwise NULL

a valid .Random.seed value.

Note

Not parallel compatible, this modifies the global environment, while processing.

See Also

set.seed
Index

.First, 7
apply, 10
attr, 4
attributes, 8
bale, 2
called_from, 2, 7
environment, 12
farm, 3, 4–6, 8, 10
gather, 3, 4, 5, 6, 8, 10
get.seed(withseed), 12
getAttr, 4
getOption, 4
GetOrSetSeed(withseed), 12
graft, 6, 10
graft(plant), 8
harvest, 3, 4, 5, 6, 8, 10
harvestr, 5, 9
harvestr-package(harvestr), 5
Interactive, 7
interactive, 6, 7
is_knitting(called_from), 2
is_seeded, 7
nextRNGSubStream, 10
noattr, 7
option, 6
package-harvestr(harvestr), 5
plant, 3–6, 8, 10
plow, 8
reap, 6, 9
ReferenceClasses, 11
replace.seed(withseed), 12
RNG, 4
set.seed, 13
sprout, 3–6, 8, 10
total_time, 10
use_method, 11
with, 10
withseed, 5, 10, 12
working directory, 6