Package ‘harvestr’

February 20, 2015

Type Package
Title A Parallel Simulation Framework
Version 0.6.0
Date 2014-05-27
Author Andrew Redd
Maintainer Andrew Redd <andrew.redd@hsc.utah.edu>
Description Functions for easy simulation and reproducability.
License GPL (>= 2)
Imports parallel, plyr, digest
Suggests testthat, dostats, doParallel, foreach, MCMCpack, boot
Collate 'withseed.R' 'gather.R' 'harvestr-package.R' 'utils.R'
NeedsCompilation no
Repository CRAN
Date/Publication 2014-05-28 00:50:50

R topics documented:

bale ......................................................... 2
farm ......................................................... 2
gather ..................................................... 3
harvest ................................................... 3
harvestr ................................................... 4
is_seeded ............................................... 5
noattr .................................................... 5
plant ....................................................... 6
plow ....................................................... 6
reap ....................................................... 7
sprout ..................................................... 8
total_time ............................................... 8
use_method ............................................... 9
withseed ............................................... 10

Index 12
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
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
farm(seeds, expr, envir = parent.frame(), ..., cache = getOption("harvestr.use.cache", FALSE), time = getOption("harvestr.time", FALSE), .parallel = getOption("harvestr.parallel", FALSE))

Arguments
seeds  a list of seeds can be obtained though 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?
**gather**

_Gather independent seeds._

**Description**

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

**Usage**

```r
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 will be gathered rather than ending seeds.

**See Also**

RNG

Other harvest: `farm`, `plant`, `harvest`, `sprout`

---

**harvest**

_Harvest the results._

**Description**

Harvest the results.

**Usage**

```r
harvest(.list, fun, ..., time = getOption("harvest.r.time", FALSE),
        .parallel = getOption("harvest.r.parallel", FALSE))
```

**Arguments**

- `.list` : a list of `data.frame`es See details.
- `fun` : a function to apply
- `...` : passed to `fun`
- `time` : should results be timed?
- `.parallel` : should the computations be run in parallel?
Details

harvest is functionally 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; graft, plant; sprout

Description

harvestr package

Caching

The functions in harvestr can cache results for faster and interruptible simulations. This option defaults to 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 working directory. This can be specified by setting the 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. harvestr.use.cache=FALSE - Should results be cached for fault tolerance and accelerated reproducability?
2. harvestr.cache.dir="harvestr-cache" - The directory to use for storing cached results.
3. harvestr.time=FALSE - Should results be timed?
4. harvestr.use.try=TRUE - Should the vectorized calls use try to increase fault tolerance?
5. harvestr.try.silent=FALSE - Should try be run silently?
6. harvestr.try.summary=TRUE - Print a result if errors were found?
7. harvestr.parallel=FALSE - Run results in parallel?
The harvestr package is a framework for parallel reproducible simulations.

The functions to know about are:

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

---

### 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**

```r
is_seeded(x)
```

**Arguments**

- `x` an object or list to check

---

### noattr

**Strip attributes from an object.**

**Description**

Strip attributes from an object.

**Usage**

```r
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`
- `x` an objects that already has seeds.
- `n` number of seeds to create

**See Also**

Other harvest: `farm`; `gather`; `harvest`; `sprout`

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, ..., seeds = gather(nrow(df)))
```

**Arguments**

- `df` a data frame of parameters
- `f` a function
- `...` additional parameters
- `seeds` seeds to use.
Value

a list with \( f \) applied to each row of \( df \).

---

**reap**

*Call a function continuing the random number stream.*

---

**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", FALSE), cache.dir = getOption("harvestr.cache.dir", "harvestr-cache"), time = getOption("harvestr.time", FALSE))
```

**Arguments**

- \( x \) an object
- \( \text{fun} \) 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?

**Details**

The function calling works the same as the `apply` family of functions.

**See Also**

`withseed`, `harvest`, and `with`
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, i.e. the result of any `harvestr` function.

**See Also**

- `nextRNGSubStream`
- Other harvest: `farm; gather; graft; plant; harvest`

---

**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**

```r
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 <- mapply(p, mr$new)
# plant seeds to prep objects for harvest
objs <- plant(objs)
```
# run methods on objects
talk <- harvest(objs, use_method(babble))
unlist(talk)

# and to show reproducibility
more.talk <- harvest(objs, use_method(babble))
identical(unlist(talk), unlist(more.talk))

## 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

```r
withseed(seed, expr, envir = parent.frame(),
         cache = getOption("harvestr.use.cache", FALSE),
         cache.dir = getOption("harvestr.cache.dir", "harvestr-cache"),
         time = getOption("harvestr.time", FALSE))
```

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

apply, 7
attributes, 5
bale, 2
environment, 10
farm, 2, 3–6, 8
gather, 2, 3, 3, 4–6, 8
goingseed (withseed), 10
GetorSetSeed (withseed), 10
graft, 3–5, 8
graft (plant), 6
harvest, 3, 3, 5–8
harvestr, 4, 7
harvestr-package (harvestr), 4
is_seeded, 5
nextRNGSubStream, 8
noattr, 5
option, 4
package-harvestr (harvestr), 4
plant, 3–5, 6, 8
plow, 6
reap, 5, 7
ReferenceClasses, 9
replace.seed (withseed), 10
RNG, 3
set.seed, 11
sprout, 3–6, 8
total_time, 8
use_method, 9
with, 7
withseed, 4, 7, 10
working directory, 4