

Package: SpaDES.experiment (via r-universe)

October 16, 2026

Type Package

Title Simulation Experiments Within The SpaDES Ecosystem (Deprecated)

Description DEPRECATED and no longer maintained: this functionality has moved to the 'SpaDES.project' package, which is now its maintained home. The experiment functions here (experiment(), experiment2(), simInitAndExperiment()) forward to 'SpaDES.project'. Historically, this package provided tools to do simulation experiments within the SpaDES ecosystem, including replication, parameter sweeps, scenario analysis, pattern oriented modeling, and simulation experiments, and introduced the simLists class (an environment that contains many simList class objects) plus tools to do post hoc analyses of such objects.

URL <https://spades-experiment.predictiveecology.org/>,
<https://github.com/PredictiveEcology/SpaDES.experiment>

Date 2026-05-22

Version 0.0.2.9006

Depends R (>= 4.1), reproducible (>= 0.2.11), SpaDES.core (>= 0.2.7)

Imports data.table (>= 1.10.4), DEoptim (>= 2.2-4), future,
future.apply, googledrive, methods, parallel, purrr, raster (>= 2.5-8), utils

Suggests covr, future.callr, ggplot2, igraph, knitr, NLMR (>= 1.1.1),
quickPlot, RColorBrewer, rmarkdown, sf, SpaDES.project,
SpaDES.tools (>= 0.2.0), testthat (>= 3.0.0)

Remotes ropensci/NLMR, PredictiveEcology/quickPlot@development,
cran/RandomFields, cran/RandomFieldsUtils,
PredictiveEcology/reproducible@development,
PredictiveEcology/SpaDES.core@development,
PredictiveEcology/SpaDES.project,
PredictiveEcology/SpaDES.tools@development

Additional_repositories <https://predictiveecology.r-universe.dev/>

Encoding UTF-8

Language en-CA
License GPL-3
VignetteBuilder knitr, rmarkdown
BugReports <https://github.com/PredictiveEcology/SpaDES.experiment/issues>
ByteCompile yes
Collate 'POM.R' 'helpers.R' 'simLists-class.R' 'as.data.table.R'
 'experiment.R' 'experiment2.R' 'simInitAndExperiment.R'
 'spades-experiment-package.R'
RoxygenNote 7.3.3
Roxygen list(markdown = TRUE)
Config/testthat/edition 3
Config/pak/sysreqs cmake libgdal-dev gdal-bin libgeos-dev libglpk-dev make libuv1-dev libxml2-
 dev libssl-dev libproj-dev libsqlite3-dev
Repository <https://predictiveecology.r-universe.dev>
Date/Publication 2026-09-18 03:46:41 UTC
RemoteUrl <https://github.com/PredictiveEcology/SpaDES.experiment>
RemoteRef development
RemoteSha 5ef3508f98e82842dc724195bdf6a464ad8b510c

Contents

SpaDES.experiment-package	2
as.data.table.simLists	4
experiment	6
experiment2	8
initialize,simLists-method	9
POM	10
show,simLists-method	16
simInitAndExperiment	17
simLists-class	18
Index	19

Description



SpaDES.experiment **is deprecated and no longer maintained**. Its experiment functionality has moved to the **SpaDES.project** package, which is now its maintained home: `experiment()`, `experiment2()`, `simInitAndExperiment()`, the `simLists` class and `as.data.table.simLists()` all live there. The same-named functions here are thin shims that forward to SpaDES.project (when installed). Please migrate: `remotes::install_github("PredictiveEcology/SpaDES.project")`.

Historically, this package provided tools to do simulation experiments within the SpaDES ecosystem – replication, parameter sweeps, scenario analysis, pattern oriented modeling, and simulation experiments – and introduced the `simLists` class (an environment holding many `simList` objects) plus tools for post hoc analyses of such objects.

Bug reports: <https://github.com/PredictiveEcology/SpaDES.experiment/issues>

Module repository: <https://github.com/PredictiveEcology/SpaDES-modules>

Wiki: <https://github.com/PredictiveEcology/SpaDES/wiki>

Author(s)

Maintainer: Eliot J B McIntire <eliot.mcintire@canada.ca> ([ORCID](#))

Authors:

- Alex M Chubaty <achubaty@for-cast.ca> ([ORCID](#))

Other contributors:

- Her Majesty the Queen in Right of Canada, as represented by the Minister of Natural Resources Canada [copyright holder]

See Also

Useful links:

- <https://spades-experiment.predictiveecology.org/>
- <https://github.com/PredictiveEcology/SpaDES.experiment>
- Report bugs at <https://github.com/PredictiveEcology/SpaDES.experiment/issues>

as.data.table.simLists

Coerce elements of a simLists object to a data.table

Description

This is particularly useful to build plots using the **tidyverse**, e.g., **ggplot2**.

Usage

```
## S3 method for class 'simLists'
as.data.table(x, vals, objectsFromSim = NULL, objectsFromOutputs = NULL, ...)
```

Arguments

x	An R object.
vals	A (named) list of object names to extract from each simList, or a named list of quoted expressions to calculate for each simList, or a mix of character and quoted expressions.
objectsFromSim	Character vector of objects to extract from the simLists. If omitted, it will extract all objects from each simList in order to calculate the vals. This may have a computational cost. If NA, then no objects will be accessed from the simList. Objects identified here will only be as they are in the simList, i.e., at end(sim).
objectsFromOutputs	List of (named) character vectors of objects to load from the outputs(sim) prior to evaluating vals. If there already is an object with that same name in the simList, then it will be overwritten with the object loaded from outputs(sim). If there are many objects with the same name, specifically from several saveTime values in the outputs(sim), these will all be loaded, one at a time, vals evaluated one at a time, and each of the values will be returned from each saveTime. A column, saveTime, will be part of the returned data.table. For cases where more than one object is required at a given saveTime, all should be identified here, without time specified. This function will take all identified objects from the same time period.
...	Additional arguments. Currently unused.

Details

See examples.

Value

This returns a data.table class object with

Examples

```
## Not run:
if (require("ggplot2", quietly = TRUE) &&
    require("NLMR", quietly = TRUE) &&
    require("RColorBrewer", quietly = TRUE)) {
  library(SpaDES.core)
  library(SpaDES.experiment)

  tmpdir <- file.path(tempdir(), "examples")
  # Make 3 simLists -- set up scenarios
  endTime <- 2

  # Example of changing parameter values
  # Make 3 simLists with some differences between them
  mySim <- lapply(c(10, 20, 30), function(nFires) {
    simInit(
      times = list(start = 0.0, end = endTime, timeunit = "year"),
      params = list(
        .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
        # Turn off interactive plotting
        fireSpread = list(.plotInitialTime = NA, spreadprob = c(0.2), nFires = c(10)),
        caribouMovement = list(.plotInitialTime = NA),
        randomLandscapes = list(.plotInitialTime = NA, .useCache = "init")
      ),
      modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
      paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"),
                    outputPath = tmpdir),
      # Save final state of landscape and caribou
      outputs = data.frame(
        objectName = c(rep("landscape", endTime), "caribou", "caribou"),
        saveTimes = c(seq_len(endTime), unique(c(ceiling(endTime / 2), endTime))),
        stringsAsFactors = FALSE
      )
    )
  })

  planTypes <- c("sequential") # try others! ?future::plan
  sims <- experiment2(sim1 = mySim[[1]], sim2 = mySim[[2]], sim3 = mySim[[3]],
                      replicates = 3)

  # Try pulling out values from simulation experiments
  # 2 variables
  df1 <- as.data.table(sims, vals = c("nPixelsBurned", NCaribou = quote(length(caribou$x1))))

  # Now use objects that were saved to disk at different times during spades call
  df1 <- as.data.table(sims,
                      vals = c("nPixelsBurned", NCaribou = quote(length(caribou$x1))),
                      objectsFromOutputs = list(nPixelsBurned = NA, NCaribou = "caribou"))

  # now calculate 4 different values, some from data saved at different times
  # Define new function -- this calculates perimeter to area ratio
```

```

fn <- quote({
  landscape$Fires[landscape$Fires[] == 0] <- NA;
  a <- boundaries(landscape$Fires, type = "inner");
  a[landscape$Fires[] > 0 & a[] == 1] <- landscape$Fires[landscape$Fires[] > 0 & a[] == 1];
  peri <- table(a[]);
  area <- table(landscape$Fires[]);
  keep <- match(names(area), names(peri));
  mean(peri[keep]/area)
})

df1 <- as.data.table(sims,
  vals = c("nPixelsBurned",
    perimToArea = fn,
    meanFireSize = quote(mean(table(landscape$Fires[])[-1])),
    caribouPerHaFire = quote({
      NROW(caribou) /
      mean(table(landscape$Fires[])[-1])
    })),
  objectsFromOutputs = list(NA, c("landscape"), c("landscape"),
    c("landscape", "caribou")),
  objectsFromSim = "nPixelsBurned")

if (interactive()) {
  # with an unevaluated string
  library(ggplot2)
  p <- lapply(unique(df1$vals), function(var) {
    ggplot(df1[vals == var,],
      aes(x = saveTime, y = value, group = simList, color = simList)) +
    stat_summary(geom = "point", fun.y = mean) +
    stat_summary(geom = "line", fun.y = mean) +
    stat_summary(geom = "errorbar", fun.data = mean_se, width = 0.2) +
    ylab(var)
  })

  # Arrange all 4 -- could use gridExtra::grid.arrange -- easier
  pushViewport(viewport(layout = grid.layout(2, 2)))
  vplotout <- function(x, y) viewport(layout.pos.row = x, layout.pos.col = y)
  print(p[[1]], vp = vplotout(1, 1))
  print(p[[2]], vp = vplotout(1, 2))
  print(p[[3]], vp = vplotout(2, 1))
  print(p[[4]], vp = vplotout(2, 2))
}
}

## End(Not run)

```

Description

Deprecated – moved to SpaDES.project.

experiment() has moved to the **SpaDES.project** package, which is now its maintained home. The version here is a thin shim: it emits a deprecation message and forwards to [SpaDES.project::experiment\(\)](#) when that package is installed. Please update your code to call SpaDES.project::experiment() directly.

Usage

```
experiment(  
  sim,  
  replicates = 1,  
  params,  
  modules,  
  objects = list(),  
  inputs,  
  dirPrefix = "simNum",  
  substrLength = 3,  
  saveExperiment = TRUE,  
  experimentFile = "experiment.RData",  
  clearSimEnv = FALSE,  
  notOlderThan,  
  cl,  
  ...  
)  
  
## S4 method for signature 'simList'  
experiment(  
  sim,  
  replicates = 1,  
  params,  
  modules,  
  objects = list(),  
  inputs,  
  dirPrefix = "simNum",  
  substrLength = 3,  
  saveExperiment = TRUE,  
  experimentFile = "experiment.RData",  
  clearSimEnv = FALSE,  
  notOlderThan,  
  cl,  
  ...  
)
```

Arguments

sim	A simList.
replicates	Number of replicates. See SpaDES.project::experiment() .

params, modules, objects, inputs
 Alternatives to vary. See [SpaDES.project::experiment\(\)](#).
 dirPrefix, substrLength, saveExperiment, experimentFile, clearSimEnv
 See [SpaDES.project::experiment\(\)](#).
 notOlderThan, cl
 Deprecated; see [SpaDES.project::experiment\(\)](#).
 ... Passed to [SpaDES.project::experiment\(\)](#).

Value

See [SpaDES.project::experiment\(\)](#).

Author(s)

Eliot McIntire

See Also

[SpaDES.project::experiment\(\)](#), [SpaDES.project::experiment2\(\)](#)

experiment2

Run experiment, algorithm 2 (moved to SpaDES.project)

Description

Deprecated – moved to SpaDES.project.

experiment2() has moved to the **SpaDES.project** package, which is now its maintained home. The version here is a thin shim: it emits a deprecation message and forwards to [SpaDES.project::experiment2\(\)](#) when that package is installed. Please update your code to call [SpaDES.project::experiment2\(\)](#) directly.

Usage

```
experiment2(
  ...,
  replicates = 1,
  clearSimEnv = FALSE,
  createUniquePaths = c("outputPath"),
  useCache = FALSE,
  debug = getOption("spades.debug"),
  drive_auth_account,
  meanStaggerIntervalInSecs
)

## S4 method for signature 'simList'
experiment2(
  ...,
```

```

    replicates = 1,
    clearSimEnv = FALSE,
    createUniquePaths = c("outputPath"),
    useCache = FALSE,
    debug = getOption("spades.debug"),
    drive_auth_account = NULL,
    meanStaggerIntervalInSecs = 1
  )

```

Arguments

... One or more `simList` objects (and any named arguments to forward to `spades()`, e.g. `events`). See [SpaDES.project::experiment2\(\)](#).

`replicates`, `clearSimEnv`, `createUniquePaths`, `useCache`, `debug`,
`drive_auth_account`, `meanStaggerIntervalInSecs`
 See [SpaDES.project::experiment2\(\)](#).

Value

See [SpaDES.project::experiment2\(\)](#).

Author(s)

Eliot McIntire

See Also

[SpaDES.project::experiment2\(\)](#), [SpaDES.project::experiment\(\)](#)

initialize,simLists-method

Generate a simLists object

Description

Given the name or the definition of a class, plus optionally data to be included in the object, `new` returns an object from that class.

Usage

```

## S4 method for signature 'simLists'
initialize(.Object, ...)

```

Arguments

`.Object` A `simList` object.

... Optional Values passed to any or all slot

Description

This is very much in alpha condition. It has been tested on simple problems, as shown in the examples, with up to 2 parameters. It appears that DEoptim is the superior package for the stochastic problems. This should be used with caution as with all optimization routines. This function can nevertheless take optim as optimizer, using stats::optim. However, these latter approaches do not seem appropriate for stochastic problems, and have not been widely tested and are not supported within POM.

Usage

```
POM(
  sim,
  params,
  objects = NULL,
  objFn,
  cl,
  optimizer = "DEoptim",
  sterr = FALSE,
  ...,
  objFnCompare = "MAD",
  optimControl = NULL,
  NaNRetries = NA,
  logObjFnVals = FALSE,
  weights,
  useLog = FALSE
)

## S4 method for signature 'simList,character'
POM(
  sim,
  params,
  objects = NULL,
  objFn,
  cl,
  optimizer = "DEoptim",
  sterr = FALSE,
  ...,
  objFnCompare = "MAD",
  optimControl = NULL,
  NaNRetries = NA,
  logObjFnVals = FALSE,
  weights,
  useLog = FALSE
```

)

Arguments

<code>sim</code>	A <code>simList</code> simulation object, generally produced by <code>simInit</code> .
<code>params</code>	Character vector of parameter names that can be changed by the optimizer. These must be accessible with <code>params(sim)</code> internally.
<code>objects</code>	A optional named list (must be specified if <code>objFn</code> is not). The names of each list element must correspond to an object in the <code>.GlobalEnv</code> and the list elements must be objects or functions of objects that can be accessed in the <code>ls(sim)</code> internally. These will be used to create the objective function passed to the optimizer. See details and examples.
<code>objFn</code>	An optional objective function to be passed into optimizer. If missing, then POM will use <code>objFnCompare</code> and <code>objects</code> instead. If using POM with a <code>SpaDES</code> simulation, this <code>objFn</code> must contain a <code>spades</code> call internally, followed by a derivation of a value that can be minimized but the optimizer. It must have, as first argument, the values for the parameters. See example.
<code>cl</code>	A cluster object. Optional. This would generally be created using <code>parallel::makeCluster</code> or equivalent. This is an alternative way, instead of <code>beginCluster()</code> , to use parallelism for this function, allowing for more control over cluster use.
<code>optimizer</code>	The function to use to optimize. Default is <code>"DEoptim"</code> . Currently it can also be <code>"optim"</code> , which uses <code>stats::optim</code> . The latter two do not seem optimal for stochastic problems and have not been widely tested.
<code>sterr</code>	Logical. If using <code>optimizer = "optim"</code> , the hessian can be calculated. If this is <code>TRUE</code> , then the standard errors can be estimated using that hessian, assuming normality.
<code>...</code>	All objects needed in <code>objFn</code>
<code>objFnCompare</code>	Character string. Either, <code>"MAD"</code> (default) or <code>"RMSE"</code> indicating that inside the objective function, data and prediction will be compared by Mean Absolute Deviation or Root Mean Squared Error.
<code>optimControl</code>	List of control arguments passed into the control of each optimization routine. Currently, only passed to <code>DEoptim::DEoptim.control()</code> when optimizer is <code>"DEoptim"</code>
<code>NaNRetries</code>	Numeric. If greater than 1, then the function will retry the objective function for a total of that number of times if it results in an <code>NaN</code> . In general this should not be used as the objective function should be made so that it doesn't produce <code>NaN</code> . But, sometimes it is difficult to diagnose stochastic results.
<code>logObjFnVals</code>	Logical or Character string indicating a filename to log the outputs. Ignored if <code>objFn</code> is supplied. If <code>TRUE</code> (and there is no <code>objFn</code> supplied), then the value of the individual patterns will be output the console if being run interactively or to a tab delimited text file named <code>ObjectiveFnValues.txt</code> (or that passed by the user here) at each evaluation of the POM created objective function. See details.
<code>weights</code>	Numeric. If provided, this vector will be multiplied by the standardized deviations (possibly <code>MAD</code> or <code>RMSE</code>) as described in <code>objects</code> . This has the effect of weighing each standardized deviation (pattern–data pair) to a user specified amount in the objective function.

`useLog` Logical. Should the data patterns and output patterns be logged (log) before calculating the `objFnCompare`. I.e., `mean(abs(log(output) - log(data)))`. This should be length 1 or length objects. It will be recycled if length > 1, less than objects.

Details

There are two ways to use this function: via 1) `objFn` or 2) objects.

1. The user can pass the entire objective function to the `objFn` argument that will be passed directly to the optimizer. For this, the user will likely need to pass named objects as part of the
2. The slightly simpler approach is to pass a list of 'actual data-simulated data' pairs as a named list in objects and specify how these objects should be compared via `objFnCompare` (whose default is Mean Absolute Deviation or "MAD").

Option 1 offers more control to the user, but may require more knowledge. Option 1 should likely contain a call to `simInit(Copy(simList))` and `spades` internally. See examples that show simple examples of each type, option 1 and option 2. In both cases, `params` is required to indicate which parameters can be varied in order to achieve the fit.

Currently, option 1 only exists when optimizer is "DEoptim", the default.

The upper and lower limits for parameter values are taken from the metadata in the module. Thus, if the module metadata does not define the upper and lower limits, or these are very wide, then the optimization may have troubles. Currently, there is no way to override these upper and lower limits; the module metadata should be changed if there needs to be different parameter limits for optimization.

objects is a named list of data-pattern pairs. Each of these pairs will be assessed against one another using the `objFnCompare`, after standardizing each independently. The standardization, which only occurs if the `abs(data value < 1)`, is: `mean(abs(derived value - data value))/mean(data value)`. If the data value is between -1 and 1, then there is no standardization. If there is more than one data-pattern pair, then they will simply be added together in the objective function. This gives equal weight to each pair. If the user wishes to put different weight on each pattern, a `weights` vector can be provided. This will be used to multiply the standardized values described above. Alternatively, the user may wish to weight them differently, in which case, their relative scales can be adjusted.

There are many options that can be passed to `DEoptim:DEoptim()`, (the details of which are in the help), using `optimControl`. The defaults sent from POM to DEoptim are: `steptol = 3` (meaning it will start assessing convergence after 3 iterations (*which may not be sufficient for your problem*)), `NP = 10 * length(params)` (meaning the population size is 10 x the number of parameters) and `itermax = 200` (meaning it won't go past 200 iterations). These and others may need to be adjusted to obtain good values. **NOTE:** DEoptim does not provide a direct estimate of confidence intervals. Also, convergence may be unreliable, and may occur because `itermax` is reached. Even when convergence is indicated, the estimates are not guaranteed to be global optima. This is different than other optimizers that will normally indicate if convergence was not achieved at termination of the optimization.

Using this function with a parallel cluster currently requires that you pass `optimControl = list(parallelType = 1)`, and possibly package and variable names (and does not yet accept the `cl` argument). See examples. This setting will use all available threads on your computer. Future versions of this will

allow passing of a custom cluster object via `cl` argument. POM will automatically determine packages to load in the spawned cluster (via `SpaDES.core::packages()`) and it will load all objects in the cluster that are necessary, by sending `names(objects)` to `parVar` in `DEoptim.control`.

Setting `logObjFnVals` to `TRUE` may help diagnosing some problems. Using the POM derived objective function, essentially all patterns are treated equally. This may not give the correct behaviour for the objective function. Because POM weighs the patterns equally, it may be useful to use the log files to examine the behaviour of the pattern–data pairs. The first file, `ObjectiveFnValues.txt`, shows the result of each of the (possibly logged), pattern–data deviations, standardized, and weighted. The second file, `'ObjectiveFnValues_RawPatterns.txt'`, shows the actual value of the pattern (unstandardized, unweighted, unlogged). If `weights` is passed, then these weighted values will be reflected in the `'ObjectiveFnValues.txt'` file.

Value

A list with at least 2 elements. The first (or first several) will be the returned object from the optimizer. The second (or last if there are more than 2), named `args` is the set of arguments that were passed into the control of the optimizer.

Author(s)

Eliot McIntire

See Also

`SpaDES.core::spades()`, `parallel::makeCluster()`, `SpaDES.core::simInit()`

Examples

```
if (interactive() &&
    require("NLMR", quietly = TRUE) &&
    require("RColorBrewer", quietly = TRUE)) {
  set.seed(89462)
  library(parallel)
  library(raster)
  mySim <- simInit(
    times = list(start = 0.0, end = 2.0, timeunit = "year"),
    params = list(
      .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
      fireSpread = list(nfires = 5),
      randomLandscapes = list(nx = 300, ny = 300)
    ),
    modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
    paths = list(modulePath = system.file("sampleModules", package = "SpaDES.core"))
  )

  # Since this is a made up example, we don't have real data
  # to run POM against. Instead, we will run the model once,
  # take the values at the end of the simulation as if they
  # are real data, then rerun the POM function next,
  # comparing these "data" with the simulated values
  # using Mean Absolute Deviation
```

```

outData <- spades(reproducible::Copy(mySim), .plotInitialTime = NA)

# Extract the "true" data, in this case, the "proportion of cells burned"
# Function defined that will use landscape$Fires map from simList,
# i.e., sim$landscape$Fires
# the return value being compared via MAD with propCellBurnedData
propCellBurnedFn <- function(landscape) {
  sum(getValues(landscape$Fires) > 0) / ncell(landscape$Fires)
}
# visualize the burned maps of true "data"
propCellBurnedData <- propCellBurnedFn(outData$landscape)
clearPlot()
if (interactive()) {
  library(quickPlot)

  fires <- outData$landscape$Fires # Plot doesn't do well with many nested layers
  Plot(fires)
}

# Example 1 - 1 parameter
# In words, this says, "find the best value of spreadprob such that
# the proportion of the area burned in the simulation
# is as close as possible to the proportion area burned in
# the "data", using DEoptim().

# Can use cluster if computer is multi-threaded.
# This example can use parallelType = 1 in DEoptim. For this, you must manually
# pass all packages and variables as character strings.
# cl <- makeCluster(detectCores() - 1) # not implemented yet in DEoptim
out1 <- POM(mySim, "spreadprob",
  list(propCellBurnedData = propCellBurnedFn), # data = pattern pair
  #optimControl = list(parallelType = 1),
  logObjFnVals = TRUE)

## Once cl arg is available from DEoptim, this will work:
# out1 <- POM(mySim, "spreadprob", cl = cl,
#   list(propCellBurnedData = propCellBurnedFn)) # data = pattern pair

# Example 2 - 2 parameters
# Function defined that will use caribou from sim$caribou, with
# the return value being compared via MAD with nPattern
# module, parameter N, is from 10 to 1000)
caribouFn <- function(caribou) length(caribou)

# Extract "data" from simList object (normally, this would be actual data)
nPattern <- caribouFn(outData$caribou)

aTime <- Sys.time()
parsToVary <- c("spreadprob", "N")
out2 <- POM(mySim, parsToVary,
  list(propCellBurnedData = propCellBurnedFn,
    nPattern = caribouFn), logObjFnVals = TRUE)
  #optimControl = list(parallelType = 1))

```

```

      #cl = cl) # not yet implemented, waiting for DEoptim
bTime <- Sys.time()
# check that population overlaps known values (0.225 and 100)
apply(out2$member$pop, 2, quantile, c(0.025, 0.975))
hists <- apply(out2$member$pop, 2, hist, plot = FALSE)
clearPlot()
for (i in seq_along(hists)) Plot(hists[[i]], addTo = parsToVary[i],
                                title = parsToVary[i], axes = TRUE)

print(paste("DEoptim", format(bTime - aTime)))
#stopCluster(cl) # not yet implemented, waiting for DEoptim

# Example 3 - using objFn instead of objects

# list all the parameters in the simList, from these, we select to vary
params(mySim)

# Objective Function Example:
#   objective function must have several elements
#   - first argument must be parameter vector, passed to and used by DEoptim
#   - likely needs to take sim object, likely needs a copy
#     because of pass-by-reference semantics of sim objects
#   - pass data that will be used internally for objective function
objFnEx <- function(pars, # param values
                    sim, # simList object
                    nPattern, propCellBurnedData, caribouFn, propCellBurnedFn) {
  ### data

  # make a copy of simList because it will possibly be altered by spades call
  sim1 <- reproducible::Copy(sim)

  # take the parameters and assign them to simList
  params(sim1)$fireSpread$spreadprob <- pars[1]
  params(sim1)$caribouMovement$N <- pars[2]

  # run spades, without plotting
  out <- spades(sim1, .plotInitialTime = NA)

  # calculate outputs
  propCellBurnedOut <- propCellBurnedFn(out$landscape)
  nPattern_Out <- caribouFn(out$caribou)

  minimizeFn <- abs(nPattern_Out - nPattern) +
    abs(propCellBurnedOut - propCellBurnedData)

  # have more info reported to console, if desired
  # cat(minimizeFn)
  # cat(" ")
  # cat(pars)

  return(minimizeFn)
}

```

```

# Run DEoptim with custom objFn, identifying 2 parameters to allow
# to vary, and pass all necessary objects required for the
# objFn

# choose 2 of them to vary. Need to identify them in params & inside objFn
# Change optimization parameters to alter how convergence is achieved
out5 <- POM(mySim, params = c("spreadprob", "N"),
  objFn = objFnEx,
  nPattern = nPattern,
  propCellBurnedData = propCellBurnedData,
  caribouFn = caribouFn,
  propCellBurnedFn = propCellBurnedFn,
  #cl = cl, # uncomment for cluster # not yet implemented, waiting for DEoptim
  # see ?DEoptim.control for explanation of these options
  optimControl = list(
    NP = 100, # run 100 populations, allowing quantiles to be calculated
    initialpop = matrix(c(runif(100, 0.2, 0.24), runif(100, 80, 120)), ncol = 2),
    parallelType = 1
  )
)

# Can also use an optimizer directly -- miss automatic parameter bounds,
# and automatic objective function using option 2
library(DEoptim)
out7 <- DEoptim(fn = objFnEx,
  sim = mySim,
  nPattern = nPattern,
  propCellBurnedData = propCellBurnedData,
  caribouFn = caribouFn,
  propCellBurnedFn = propCellBurnedFn,
  # cl = cl, # uncomment for cluster
  # see ?DEoptim.control for explanation of these options
  control = DEoptim.control(
    steptol = 3,
    parallelType = 1, # parallelType = 3,
    packages = list("raster", "SpaDES.core", "RColorBrewer"),
    parVar = list("objFnEx"),
    initialpop = matrix(c(runif(40, 0.2, 0.24),
      runif(40, 80, 120)), ncol = 2)),
  lower = c(0.2, 80), upper = c(0.24, 120))
}

```

show,simLists-method *Show method for simLists*

Description

Show method for simLists

Usage

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

Arguments

object simLists

Author(s)

Eliot McIntire

simInitAndExperiment *Run simInit and experiment in one step (moved to SpaDES.project)*

Description

Deprecated – moved to SpaDES.project.

simInitAndExperiment() has moved to the **SpaDES.project** package, which is now its main-tained home. The version here is a thin shim: it emits a deprecation message and forwards to [SpaDES.project::simInitAndExperiment\(\)](#) when that package is installed. Please update your code to call SpaDES.project::simInitAndExperiment() directly.

Usage

```
simInitAndExperiment(
  times,
  params,
  modules,
  objects,
  paths,
  inputs,
  outputs,
  loadOrder,
  notOlderThan,
  replicates,
  dirPrefix,
  substrLength,
  saveExperiment,
  experimentFile,
  clearSimEnv,
  cl,
  ...
)
```

Arguments

times, params, modules, objects, paths, inputs, outputs, loadOrder,
notOlderThan

See [SpaDES.core::simInit\(\)](#).

replicates, dirPrefix, substrLength, saveExperiment, experimentFile,
clearSimEnv, cl

See [SpaDES.project::experiment\(\)](#).

... Passed onward.

simLists-class

The simLists class

Description

This is a grouping of simList objects. Normally this class will be made using experiment2, but can be made manually if there are existing simList objects.

Slots

paths Named list of modulePath, inputPath, and outputPath paths. Partial matching is performed. These will be prepended to the relative paths of each simList

.xData Environment holding the simLists.

Accessor Methods

None yet defined:

[SpaDES.core::simList-accessors-envir\(\)](#) Simulation environment.

Author(s)

Eliot McIntire

Index

`as.data.table.simLists`, 4

`DEoptim::DEoptim()`, 12

`DEoptim::DEoptim.control()`, 11

`experiment`, 6

`experiment, simList-method (experiment)`, 6

`experiment2`, 8

`experiment2, simList-method (experiment2)`, 8

`initialize, simLists-method`, 9

`parallel::makeCluster()`, 13

`POM`, 10

`POM, simList, character-method (POM)`, 10

`show, simLists-method`, 16

`simInitAndExperiment`, 17

`simLists (simLists-class)`, 18

`simLists-class`, 18

`SpaDES.core::packages()`, 13

`SpaDES.core::simInit()`, 13, 18

`SpaDES.core::simList-accessors-envir()`, 18

`SpaDES.core::spades()`, 6, 13

`SpaDES.experiment`
`(SpaDES.experiment-package)`, 2

`SpaDES.experiment-package`, 2

`SpaDES.project::experiment()`, 7–9, 18

`SpaDES.project::experiment2()`, 8, 9

`SpaDES.project::simInitAndExperiment()`, 17