Package 'SiMRiv'

July 21, 2025

Version 1.0.7

Date 2024-09-09

Title Simulating Multistate Movements in River/Heterogeneous Landscapes

Depends R (>= 3.5.0), terra

Imports methods, stats, mco, parallel

Suggests adehabitatLT, moveHMM, testthat, sf

Description Provides functions to generate and analyze spatially-explicit individual-based multistate movements in rivers, heterogeneous and homogeneous spaces. This is done by incorporating landscape bias on local behaviour, based on resistance rasters. Although originally conceived and designed to simulate trajectories of species constrained to linear habitats/dendritic ecological networks (e.g. river networks), the simulation algorithm is built to be highly flexible and can be applied to any (aquatic, semi-aquatic or terrestrial) organism, independently on the landscape in which it moves. Thus, the user will be able to use the package to simulate movements either in homogeneous landscapes, heterogeneous landscapes (e.g. semi-aquatic animal moving mainly along rivers but also using the matrix), or even in highly contrasted landscapes (e.g. fish in a river network). The algorithm and its input parameters are the same for all cases, so that results are comparable. Simulated trajectories can then be used as mechanistic null models (Potts & Lewis 2014, <DOI:10.1098/rspb.2014.0231>) to test a variety of 'Movement Ecology' hypotheses (Nathan et al. 2008, <DOI:10.1073/pnas.0800375105>), including landscape effects (e.g. resources, infrastructures) on animal movement and species site fidelity, or for predictive purposes (e.g. road mortality risk, dispersal/connectivity). The package should be relevant to explore a broad spectrum of ecological phenomena, such as those at the interface of animal behaviour, management, landscape and movement ecology, disease and invasive species

spread, and population dynamics.

License GPL (>= 2)

URL https://www.r-project.org, https://github.com/miguel-porto/SiMRiv

BugReports https://github.com/miguel-porto/SiMRiv/issues

NeedsCompilation yes

Author Lorenzo Quaglietta [aut],
Miguel Porto [aut, cre],
Erida Gjini [ctb]

Maintainer Miguel Porto <mpbertolo@gmail.com>

Repository CRAN

Date/Publication 2024-09-10 09:30:05 UTC

Contents

SiMRiv-package *Simulating Multistate Movements in River/Heterogeneous Landscapes*

Description

Provides functions to generate and analyze individual-based, spatially-explicit simulations of multi-state movements in homogeneous or heterogeneous landscapes, based on "resistance" rasters. Although originally conceived and designed to simulate spatially-explicit trajectories of species constrained to linear habitats or dendritic ecological networks (e.g., river networks), the simulation algorithm is built to be highly flexible and can be applied to any (aquatic, semi-aquatic or terrestrial) organism, independently of the landscape in which it moves. Thus, the user will be able to use the package to simulate movements either in homogeneous landscapes, heterogeneous landscapes (e.g. semi-aquatic animal moving mainly along rivers but also using the matrix), or even in highly contrasted landscapes (e.g. fish in a river network). The algorithm and its input parameters are the

same for all cases, so that results are comparable. Simulated trajectories can then be used as mechanistic null models (Moorcroft and Lewis 2006) to test e.g. for species site fidelity (Powell 2000) and other 'Movement Ecology' hypotheses (Nathan et al. 2008), or for other predictive purposes. The package should thus be relevant to explore a broad spectrum of ecological phenomena, such as those at the interface of animal behaviour, landscape, spatial and movement ecology, disease and invasive species spread, and population dynamics.

Details

`simulate` is the central function. See the examples in `?simulate` to quickly get started, or the vignette for a more verbose tutorial.

Index of help topics:

`Arith-methods`	`Shortcuts for defining species movement states`
`SiMRiv-package`	`Simulating Multistate Movements in River/Heterogeneous Landscapes`
`adjustModel`	`Finds ("estimates") simulation input parameters able to replicate a given (real) trajectory, assuming the given species model`
`binCounts`	`Count values in given bins`
`generationPlot`	`Plots input parameter optimization results`
`perceptualRange`	`Define a perceptual range`
`resistanceFromShape`	`Build resistance raster by combining shapefiles`
`sampleMovement`	`Resample a simulated movement and compute step-wise statistics`
`simulate`	`Simulate movements in river networks, homogeneous, or heterogeneous landscapes`
`species`	`Create a species`
`speciesModel`	`Defines a species model to adjust to a real trajectory`
`state`	`Define a movement state`
`transitionMatrix`	`Define a state transition matrix`

Further information is available in the following vignettes:

`SiMRiv`	Usage of the SiMRiv package (source)

References

- Powell, R. A. 2000. Animal home ranges and territories and home range estimators. In: Research techniques in animal ecology: controversies and consequences, 442. Boitani, L., & Fuller, T. (Eds.). Columbia university press, New York: pp.65-110.
- Moorcroft, P. R. & Lewis, M. A. 2006. Mechanistic Home Range Analysis. Monographs in Population Biology 43. Eds. Levin S.A. and H.S. Horn. Princeton University Press. pp 172.
- Nathan, R., Getz, W. M., Revilla, E., Holyoak, M., Kadmon, R., Saltz, D., & Smouse, P. E. 2008. A movement ecology paradigm for unifying organismal movement research. Proceedings of the National Academy of Sciences,105(49), 19052-19059.

Examples

```
## a simple Levy-like movement in homogeneous space
## see ?simulate for more complex examples

LevyWalker <- species(
    state.RW() + state.CRW(0.99),
    transitionMatrix(0.005, 0.02))

sim <- simulate(LevyWalker, 20000)
plot(sim, type="l", asp=1)
```

`adjustModel` *Finds ("estimates") simulation input parameters able to replicate a given (real) trajectory, assuming the given species model*

Description

Given a trajectory, a type of movement and the time resolution at which the user wants to simulate, this function approximates the values for the simulation input parameters so that the simulated movement is maximally similar to the given trajectory, in terms of general non-spatial patterns. If the user wants to simulate at a higher frequency than real data (which is the norm), the function maximizes the similarity between the real trajectory and the simulated trajectories (at a higher frequency) after downsampled to the same frequency as the real. It does so by running a genetic optimization algorithm.

Usage

```
adjustModel(realData, species.model, resolution = 10
  , resistance = NULL, coords = NULL, angles = NULL
  , nrepetitions = 6
  , nbins.hist = if(aggregate.obj.hist) c(7, 7, 0) else c(3, 3, 0)
  , step.hist.log = TRUE, nlags = 100
  , window.size = dim(reference$stats)[1]%/%nlags
  , aggregate.obj.hist = TRUE, step.hist.range = c(0, 1)
  , popsize = 100, generations = seq(5, 1000, by=5), mprob = 0.2
  , parallel = is.null(resistance), trace = TRUE
)
```

Arguments

`realData`	the given trajectory for which the simulation input parameters are to be "estimated", given as a matrix with two columns (coordinates) and assuming that relocations are equally spaced in time.
`species.model`	the species model to adjust, created with `speciesModel`. This defines the type of movement that is to be adjusted (e.g. how many, and which type of, behavioral states, see details).

`resolution` the desired time frequency of the simulations for which parameters will be approximated, as a fraction of the real data, i.e. a value of 20 will simulate movements at a 20-fold higher frequency than real data.

`resistance` the resistance raster to use in simulations during parameter approximation.

`coords` the initial coordinates of the simulated individuals (only relevant if `resistance` is provided because the metrics used in optimization are spatially-agnostic).

`angles` the initial angle to which the individual is facing in that start of all simulations (only relevant if `resistance` is provided because the metrics used in optimization are spatially-agnostic).

`nrepetitions` the number of simulations conducted for each solution evaluation during optimization. If >1, the quality of the solutions is computed by comparing the averaged histograms across repetitions, with the real histograms.

`nbins.hist` a vector with three positive integers defining the number of histogram bins for turning angle histograms, step length histograms and turning angle variation histograms. These bins will be used during optimization to compare simulated trajectories with the real trajectory to infer the quality of the "fit".

`step.hist.log` set to `TRUE` to use the histogram of the logarithm of the step lengths rather than of the raw values in the comparisons. Setting to `TRUE` usually results in more detail in the comparisons.

`nlags` the number of time lags within which the standard deviation of the turning angles will be computed during optimization, if `nbins.hist[3] > 0`. Ignored if `nbins.hist[3] == 0` or if `window.size` is provided.

`window.size` the size (in steps of the real sampling frequency) of the time lags within which the standard deviation of the turning angles will be computed during optimization, if `nbins.hist[3] > 0`. A different way of providing `nlags`. See details.

`aggregate.obj.hist` if `FALSE`, comparison of histograms is done bin by bin (each bin absolute difference is an objective to minimize), if `TRUE` the absolute differences of the bins are summed in each histogram to a single number which is the objective being minimized (the overall absolute difference in each of the histograms).

`step.hist.range` the quantiles used to define the range of the step length histogram computation. Used if the user wants to exclude outliers. The default is not to exclude outliers, thus `c(0, 1)`.

`popsize` number of solutions to optimize, to pass to `nsga2`

`generations` number of algorithm generations to run, to pass to `nsga2`. The default is a vector, so that convergence of results can be assessed along generations.

`mprob` mutation probability, to pass to `nsga2`

`parallel` set to `TRUE` to use multicore processing.

`trace` set to `TRUE` to print the matrix of optimization objectives (rows) for each solution (columns) in each generation along optimization. These are the values that are being internally minimized. The number of rows is `sum(nbins.hist)`; the first `nbins.hist[1]` are the turning angle variation objectives, the last are the step length objectives.

Details

This function finds possible parameters for the simulation (solutions), so that the resulting movements are as similar as possible to the given real trajectory, in terms of their intrinsic properties measured by step lengths and variation in turning angles. The input parameter approximations are found using a multiobjective genetic algorithm (NSGA-II, *Deb et al. 2002*). The algorithm minimizes a vector of N objectives (N=sum(nbins.hist) if aggregate.obj.hist == FALSE, N=sum(nbins.hist > 0) otherwise) whose values are computed by the absolute differences between each pair of bins (real and simulated) of up to three histograms: an histogram of the step lengths, an histogram of turning angles and/or and histogram of the standard deviation in turning angles computed in a moving time window along each trajectory. Using either of these histograms (and the respective number of bins) is specified in the parameter `nbins.hist`, which has three elements, one for each histogram. A value of 0 tells the function not to use that corresponding histogram.

SiMRiv simulations are intended to reproduce the fine-scale movement steps, unlike what is normally collected in field data. Hence, simulations should be conducted with a much higher time frequency than provided by the real data, which poses challenges for parameterization. This function incorporates this difference in the time scale during optimization (`resolution`), allowing the user to find the input parameters for simulations at a much higher frequency, which, when downsampled to the real data's time frequency, will present similar patterns. The higher the frequency, the more flexibility the model has to adjust to real data, but the more possible solutions may exist to achieve the same result.

There are no limits to the number of parameters that can be approximated, but obviously, the higher the number, the larger the solution space, so, in theory, the longer the algorithm has to run in order to converge. The number of parameters to approximate is defined by the user by providing a `speciesModel`. This defines how many states and which types of states are to be "fit" to the data. See `speciesModel` for details. Trials have shown that even when the number of parameters to approximate is high (e.g. 12 parameters for "fitting" a 3-state movement model), the algorithm converges rapidly if the real movement suits such model. However, as in any other method, a compromise should be sought. A good starting point is to provide a two-state species model, in which both states are Correlated Random Walks. This model involves the approximation of 6 parameters and is sufficiently flexible for simulating a variety of movements, while not overly complex. Note that all complex models can accomodate to simpler ones (i.e. the simpler models are special cases of the complex ones).

To assess the convergence of the algorithm, an utility plotting function is provided, see the example below and `generationPlot` for details.

Value

The object returned by `nsga2` (package `mco`), see details therein. If `generations` is a vector (which is recommended, for assessing convergence), this object contains the approximated input parameter values in each generation given in the vector. See examples for easily plotting results.

Note

This function is an experimental feature, here provided only to guide the user on how to parameterize the simulations. Care must be taken when interpreting the results, at least by assessing algorithm convergence and visually comparing simulations with the approximated parameters to the real data (see examples).

References

- Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2), 182-197.

See Also

simulate, generationPlot.

Examples

```
## Not run:
library(SiMRiv)
library(adehabitatLT)

## simulate "real" data, a Levy walk, for which we want to
## parameterize our model

real.data <- simm.levy(1:500)[[1]][, 1:2]

## Define a species model to adjust. Let's assume we don't know
## much about what kind of real data we have, hence define
## a flexible model: a two-state correlated random walk model
## with variable step lengths.
## This model implies "estimating" 6 parameters:
##   - turning angle correlation of state 1 [0, 1]
##   - turning angle correlation of state 2 [0, 1]
##   - switching probability S1 -> S2 [0, 1]
##   - switching probability S2 -> S1 [0, 1]
##   - maximum step length of state 1 [0, ?]
##   - maximum step length of state 2 [0, ?]

## Let's assume we want to simulate at a 20 times higher time frequency
## than real data.
## In order to allow our model to adjust to real data, we have
## to provide a maximum allowable step length to the optimization algorithm
## that allows to recover real data after downsampling 20 times.
## Let's make a simple calculation of the step lengths of real data:

tmp <- sampleMovement(real.data)

## and compute a good maximum allowed step length during optimization
## using the observed maximum divided by 20 (because each real step
## will comprise 20 simulated steps)

max.step.length <- max(tmp$stat[, "steplengths"]) / 20

## and finally build the species model with it.
## Note: "CRW.CRW.sl" is the short name for the model we want,
## as defined above

species.model <- speciesModel("CRW.CRW.sl", steplength = max.step.length)
```

```
## now run optimization

sol <- adjustModel(real.data, species.model, resol = 20
, nbins.hist = c(3, 3, 0), step.hist.log = TRUE)

## After finishing, we can extract the input parameters of the optimized
## solutions (100 by default) in the last generation (generation 1000
## by default):

pars <- sol[[length(sol)]]$par

## now we can take the optimized solutions and reconstruct species
## based on them:

optimized.species <- apply(pars, 1, species.model)

## and make some simulations with those optimized species.
## Plot real trajectory

par(mfrow = c(2, 2), mar = c(0, 0, 1, 0))
plot(real.data, type = "l", asp = 1, axes = F, main = "Real")

## plot three simulated trajectories with optimized species

for(i in 1:3) {
# remember we want to simulate at a 20 times higher frequency
# so we do 500 (real data) x 20 steps
sim <- simulate(optimized.species[[i]], 500 * 20)

# now we downsample frequency to match real
samp <- sampleMovement(sim, 20)

# and plot the simulated trajectory before and after
# downsampling 20 times.

plot(sim[, 1:2], type = "l", asp = 1, axes = F, col = "gray"
, main = "Simulated and downsampled")
lines(samp$relocs, col = "black")
}

## Now plot the evolution of parameters along algorithm's generations.
## This is good to assess whether the final solutions converged
## but see ?generationPlot for details

generationPlot(sol, species.model)


## End(Not run)
```

Arith-methods *Shortcuts for defining species movement states*

Description

Shortcuts for defining species movement states with the arithmetic operators `+`, `*`.

The `+` applied to states or species and a number defines the step length of one behavioural state or of all states of a species.

The `*` applied to states or species and a number defines the radius of the perceptual range of one behavioural state or of all states of a species.

Further, states can be combined with `+` to define multistate movements in a species. See examples.

Methods

`signature(e1 = "list", e2 = "state")` Adds one state to a list of states.

`signature(e1 = "species", e2 = "ANY")` Applies the operation to all states in the given species.

`signature(e1 = "state", e2 = "list")` Adds one state to a list of states.

`signature(e1 = "state", e2 = "numeric")` Sets the step length of the given state to the given value.

`signature(e1 = "state", e2 = "state")` Combines two states in a list of states to be used with `species`.

See Also

`species`, `state`.

Examples

```
# define a species with two states
twostatespecies <- species(state.RW() + state.CRW(0.9)
  , transitionMatrix(0.01, 0.02))

# set the perception window radius of both states to 200
# and the step length to 10
twostatespecies <- twostatespecies * 200 + 10
```

`binCounts` *Count values in given bins*

Description

A convenience function to count the values (optionally log-transformed) falling within given equal-range bins.

Usage

```
binCounts(data, range, nbins, log = FALSE)
```

Arguments

data	a numeric vector with the data.
range	the closed data range in which to divide bins, as a two-element numeric vector. Values falling outside range are discarded.
nbins	the number of bins to split the data. Result is guaranteed to have this number of bins, irrespective of data.
log	whether or not to log-transform data and ranges before binning.

Details

This is just a convenience function to simplify the task of couting data in equal bins. The same result could be achieved with hist with the right arguments. It is mostly used internally by adjustModel during the approximation on input parameters, for the computation of objectives.

Value

A named vector with the counts of values in each bin.

See Also

adjustModel.

---

generationPlot *Plots input parameter optimization results*

Description

Plots the evolution of the optimized solutions (sets of input parameters) along the adjustModel algorithm's generations.

Usage

```
generationPlot(solutions, species.model
  , plot.quantiles = c(0.10, 0.5, 0.90), only.pareto = FALSE
  , show.legend = TRUE, lwd = 1.5, mar = c(2.3, 2.3, 0.2, 2.3)
  , mgp = c(1.2, 0.2, 0), tcl = -0.25, ...)
```

Arguments

solutions	the result of adjustModel.
species.model	the species model that was adjusted and that was passed to adjustModel.
plot.quantiles	the three quantiles to plot. The middle is plotted as a solid line, the extremes define the shaded area.
only.pareto	whether to display the quantiles only of the Pareto front (TRUE) or of all solutions (FALSE).

show.legend	whether to show a legend indicating the correspondence of colors - input parameters.
lwd	line width to be used to draw the middle quantile (usually the median).
mar, mgp, tcl, ...	arguments to pass to par.

Details

The generation plot depicts, for each generation of the optimization algorithm, the given quantiles of each input parameter being optimized in the population of solutions. This is not the ideal plot because input parameters are plotted independently, while they are only supposed to make sense in the context of a given solution (i.e. it is the combination of the input parameters that is being optimized, not the parameters in isolation).

Nonetheless, this plot is still a good way to assess whether the final solutions converged to stable values, which can indicate that the algorithm succeeded in replicating the real trajectory provided. However, note that depending on how the problem is formulated, the solutions might not be expected to converge to a single solution "type": for example, there can be two types of solutions, corresponding to two different ways of achieving similar results.

Value

Returns, invisibly, a 3-D matrix with the computed quantiles for all input parameters along generations.

See Also

adjustModel.

Examples

```
## see ?adjustModel for a complete example
```

---

perceptualRange *Define a perceptual range*

Description

Defines the perceptual range to be used in a movement state.

Usage

```
perceptualRange(type = "circular", radius)
```

Arguments

type	defines the type of weights that are given to each pixel, according to the distance to its center. One of circular or gaussian.
radius	the radius of the circular perceptual range, or the sigma of the gaussian perceptual range, in map units.

Details

The perceptual range is often defined as the distance (radius) at which the species perceives the environment, based, e.g., on sense of smell, vision, audition, etc. (*Lima & Zollner 1996*; *Powell 2000*). In SiMRiv, perceptual range should be seen as the distance (radius) at which the species evaluates the landscape resistance around its current location, influencing species next heading. Perceptual range size can be defined (in meters or other map units) based on available literature (on species perceptual range, or, as its surrogate, species home range size), on expert-based criteria, or be estimated from real data.

A `circular` range gives equal weight to all pixels, which form a circle centered on current individual's position. A `gaussian` range gives weights corresponding to a gaussian kernel centered on current individual's position.

References

- Lima, S. L., & Zollner, P. A. (1996). Towards a behavioral ecology of ecological landscapes. Trends in Ecology & Evolution, 11(3), 131-135.
- Powell, R. A. 2000. Animal home ranges and territories and home range estimators. In: Research techniques in animal ecology: controversies and consequences, 442. Boitani, L., & Fuller, T. (Eds.). Columbia university press, New York: pp.65-110.

See Also

`state`.

`resistanceFromShape` *Build resistance raster by combining shapefiles*

Description

Creates a resistance raster to be used in simulations, by rasterizing and combining different shapefiles. It is basically a helper function that uses the functions from package `terra-package` to create and manipulate such raster.

Usage

```
  resistanceFromShape(shp, baseRaster, res, binary = is.na(field)
, field = NA, background = 1, buffer = NA, margin = 0
, mapvalues = NA, extend = TRUE, ...)
```

Arguments

`shp`	either a character string specifying the shapefile filename or a shapefile object itself.
`baseRaster`	if provided, a raster onto which to stack the given rasterized shapefile. If not provided, a new raster will be created.

res	the desired pixel resolution of the raster to be created, when baseRaster is not provided.
binary	if TRUE, the shapefile will be rasterized so that any feature is assigned a value of 0, and the background 1.
field	either a number in the range [0-1], in which case it will be assigned to all pixels covered by features of the shapefile; or the name of the numeric shapefile field from which to extract such number; or the name of the factor shapefile field containing classes to map to resistance values (see mapvalues).
background	the value in the range [0-1] to assign to all pixels that are not covered by any shapefile feature.
buffer	the size of a buffer to build around every shapefile feature before rasterizing.
margin	the margin to leave around the shapefile's extent when rasterizing (i.e. how much to increase shapefile's extent).
mapvalues	a named vector specifying the resistance value mapping from the classes of field.
extend	set to TRUE to extend baseRaster if the shapefile has a larger extent. If FALSE, the shapefile will be clipped to baseRaster's extent.
...	other arguments to pass to rasterize.

Details

This function rasterizes the given shapefile using provided options and optionally stacks it onto the provided baseRaster. The produced raster does not contain NAs and all values are in the range [0, 1]. All the areas of the raster for which data is not provided are assigned the value of background.

When combining a shapefile to an existing baseRaster, only the areas covered by features are updated in the base raster; all the remaining areas are left with the original values of baseRaster. If the shapefile to combine has a larger extent than baseRaster, those extra pixels will be assigned the background value defined for the shapefile (not use the original background of baseRaster).

Value

The resistance raster, an object of class SpatRaster-class.

See Also

simulate.

Examples

```
## Example taken from the vignette; see the vignette
## for more details and examples.
## In this example we read a land cover shapefile and
## assign resistance values based on each polygon's
## land cover class (provided in the field 'coverclass')

landcover <- resistanceFromShape(
  system.file("doc/landcover.shp", package="SiMRiv")
```

```
  , res = 150, field = "coverclass", mapvalues = c(
    "forest" = 0.5, "urban" = 1, "dam" = 0
    , "shrubland" = 0.75), background = 0.95)

## We then combine it with a river network from another
## shapefile, assigning a value of 0 to all rivers

river.landcover <- resistanceFromShape(
  system.file("doc/river-sample.shp", package="SiMRiv")
  , baseRaster = landcover, buffer = 100, field = 0
  , background = 0.95, margin = 1000)

plot(river.landcover, axes = FALSE, mar = c(0, 0, 0, 2))
```

`sampleMovement` *Resample a simulated movement and compute step-wise statistics*

Description

Resamples a movement simulated with the function `simulate` to a lower temporal resolution (frequency). Simultaneously, computes step length, turning angles (*Turchin 1998*) and accumulated resistance for each resampled step, assuming that a straight line connects each re-sampled location.

Usage

```
sampleMovement(relocs, resolution = 1, resist = NULL)
```

Arguments

`relocs`	the simulated movement, an object returned by `simulate`. NOTE: currently this is only implemented for simulations of single individuals.
`resolution`	movement will be resampled every this number of time ticks. If 1, no resampling is done (but metrics are computed).
`resist`	a landscape resistance raster, usually the same that was used in `simulate`

Details

This function mimics what happens in real world movement data: it resamples the simulated movement (which is supposed to be infinitesimal) into a lower temporal resolution, so that it is comparable to real world field data (e.g. telemetry data). During the process, it computes, for each resampled step:

- the step length
- the turning angle
- the accumulated resistance along the step, assuming a straight line is taken from start to end

Value

A list with the components `relocs` and `stats`.

`relocs` contains the resampled positions, `stats` contains the metrics for each step (which has N-2 rows because of the turning angles).

Note

These metrics are only meaningful for `resolution` » 1, otherwise they are just a consequence of the simulation input parameters.

References

- Turchin, P. 1998. Quantitative analysis of movement: measuring and modeling population redistribution in animals and plants (Vol. 1). Sinauer Associates, Sunderland, MA.

See Also

`simulate`.

Examples

```
library(SiMRiv)

LevyWalker <- species(
  state.RW() + state.CRW(0.99),
  trans = transitionMatrix(0.005, 0.02))

sim <- simulate(LevyWalker, 10000)
resamp <- sampleMovement(sim, 50)
plot(sim, type="l", asp=1, col = "#777777")
lines(resamp$relocs, col = "red")
```

`simulate` *Simulate movements in river networks, homogeneous, or heterogeneous landscapes*

Description

Performs fast and spatially-explicit simulation of multi-state random movements (*Morales et al. 2004*, *McClintock et al. 2012*) of individuals created with the function `species` in an optional landscape resistance raster.

Usage

```
simulate(individuals, time, coords = NULL
  , states = NULL, resist = NULL, angles = NULL
  , start.resistance)
```

Arguments

individuals	a species or a list of N species whose movements are to be simulated
time	the number of time steps to be simulated
coords	a N x 2 matrix giving the initial coordinates of the simulation, for each individual. Can be a vector of the form c(x, y) if only one individual is provided.
resist	an optional landscape resistance raster of class RasterLayer. If not provided, movements are simulated in an homogeneous environment.
angles	an optional numeric vector of length N defining the initial heading of each individual, in radians. Zero is north and angles increase clockwise.
start.resistance	an optional scalar in the range [0, 1] giving the maximum resistance value in which the individuals are allowed to start, if coords is NULL and resist is provided
states	Not implemented yet. An optional numeric vector of length N defining the initial state of each individual.

Details

Performs a mechanistic simulation of the movement of the given individual/s (when more than one individual is given, their movements are simulated simultaneously) in the given landscape raster defining physical resistance values. At present, multiple individuals do not interact, but in the upcoming version it will be possible to define positive and negative interactions between simulated individuals, thus accounting for spatial bias.

The simulation runs in a series of micro-steps, and is intended to be a high-resolution simulation (which can be later sampled with sampleMovement to emulate real field data, e.g. telemetry data).

In summary, at each micro-step the individual chooses a random direction which is based on the previous step heading and in the resistance context at the current position, such that it will avoid heading to areas with high resistance. This evaluation depends on the individual's perceptual range in the current movement state. At each step, a test to see if the individual changes its state is also performed, based on the provided transitionMatrix. Each state may have its own perceptual range, step length and angular correlation (with previous step heading). It's up to the user the definition of these values (e.g., expert- or literature-based), but we provide an experimental function to numerically approximate these values from real data, see adjustModel.

In more detail, in each of the time steps, the procedure is as follows:

1. Draw state for the current step according to state transition matrix and previous state
2. Compute empirical probability density for changing heading, from the landscape raster values around current position (resistance component). See details below.
3. Compute probability density for changing heading, given the previous step heading and correlation defined in the current state (correlated walk component)
4. Intersect the resistance component with the correlated walk component to make a compound probability density for changing heading
5. Draw the new heading from the probability density distribution computed above
6. Compute the length of the step that will be taken as a fraction of current step's defined length proportional to mean resistance of the starting and ending points in the chosen heading

7. Move to the new position, defined by the drawn heading and length of step.

Details of the simulation algorithm:

The landscape resistance raster: This raster represents the amount of physical resistance that is offered to the simulated individuals. The values must be between 0 (no resistance) and 1 (infinite resistance). In the future, other types of rasters can be provided, for example rasters for resource availability, habitat suitability and points of attraction/repulsion, allowing to conduct simulations with various types of spatial bias.
A careful choice of pixel size must be taken for the resistance raster. If rasterizing from vector lines (e.g. river network), please be sure to adjust the pixel size so that there are no gaps between river pixels and all pixels of the river are connected orthogonally.

The empirical probability density: The empirical probability density for a given point (resistance component) is computed by summing the `1 - resistance` values along a set of discrete radial lines departing from that point, forming a circle. The length of the lines (i.e. radius) and weighting given to each pixel are defined in the current state's perceptual range. The sums are packed and used as the circular empirical distribution of the resistance component. This will be crossed with the correlated walk component to yield the final empirical probability distribution from which heading will be drawn.

Value

A matrix with 3 columns for each simulated individual, in the order x1, y1, state1, x2, y2, state2, ...; and the same number of rows as the simulation length (given by `time`).

Note

The structure of the returned object will change in the upcoming version.

References

- McClintock, B. T., King, R., Thomas, L., Matthiopoulos, J., McConnell, B. J., & Morales, J. M. 2012. A general discrete-time modeling framework for animal movement using multistate random walks. Ecological Monographs, 82(3), 335-349.
- Morales, J. M., Haydon, D. T., Frair, J., Holsinger, K. E., & Fryxell, J. M. 2004. Extracting more out of relocation data: building movement models as mixtures of random walks. Ecology, 85(9), 2436-2445.
- Sims, D. W., Southall, E. J., Humphries, N. E., Hays, G. C., Bradshaw, C. J., Pitchford, J. W., ... & Morritt, D. 2008. Scaling laws of marine predator search behaviour. Nature, 451(7182), 1098-1102.
- Turchin, P. 1998. Quantitative analysis of movement: measuring and modeling population redistribution in animals and plants (Vol. 1). Sinauer Associates, Sunderland, MA.

See Also

`species`, `sampleMovement`, `adjustModel`.

Examples

```
library(SiMRiv)

## A classic: simple random walk (Brownian motion) (Turchin 1998)
## i.e. single-state uncorrelated movement in an homogeneous landscape
######################################################################

## a single state, other parameters set to defaults

rand.walker <- species(state.RW())
sim <- simulate(rand.walker, 10000)
plot(sim, type="l", asp=1)

## two random walkers
#####################

sim <- simulate(list(rand.walker, rand.walker), 10000)
plot(sim[,1:2], type="l", asp=1, xlim=range(sim), ylim=range(sim), col=2)
lines(sim[,4:5], col=3)

## Another classic: Levy walk-like movement (e.g. Sims et al. 2008)
## i.e. two-state movement: composition of small-scale random walks
## with bursts of longer, correlated random walks
###################################################################

LevyWalker <- species(
    state.RW() + state.CRW(0.99),
    transitionMatrix(0.005, 0.02))

sim <- simulate(LevyWalker, 10000)
plot(sim, type="l", asp=1)

## Linear habitats, e.g. fish in a river network
################################################

## load sample river raster in a fish's perspective,
## i.e. resistance is 0 within the river, 1 otherwise.

river <- terra::rast(system.file("doc/river.tif", package="SiMRiv"))

## let's try a Levy-like movement in a river network
## note: perceptual range radii and step lengths must be
## adequate to the raster resolution!

LevyWalker <- species(list(
    state(0, perceptualRange("cir", 100), 10, "RandomWalk")
    ,state(0.97, perceptualRange("cir", 500), 20, "CorrelatedRW")
), transitionMatrix(0.005, 0.001))

## NOTE: the following lines do exactly the same as above, but
## using the more convenient arithmetic operator shortcuts
```

```
LevyWalker <- species(
(state.RW() * 100 + 10) + (state.CRW(0.97) * 500 + 20)
, transitionMatrix(0.005, 0.001))

sim <- simulate(LevyWalker, 20000, resist = river
    , coords = c(280635, 505236))

## plot movement; we use a high-res TIFF so that it
## can be viewed in detail
## Not run:
tiff("movement.tif", wid=5000, hei=5000, comp="lzw")
par(mar = c(0, 0, 0, 0))
plot(river, asp = 1, col = gray(seq(1, 0.5, len = 2))
    , ylim = range(sim[,2]), xlim = range(sim[,1]), axes = FALSE)
lines(sim, lwd = 2, col = "#0000ffcc")
dev.off()

## End(Not run)

## if we want the kernel density overlaid,
## uncomment these and put before dev.off()
# library(ks)
# d <- kde(sim[,1:2])
# plot(d, disp = "image", add=TRUE
#    , col = rgb(1, 0, 0, seq(0, 1, len = 15)))
```

`species` *Create a species*

Description

Creates a species, characterized by one or more behavioral states, to be simulated with function `simulate`.

Usage

```
species(states, trans = transitionMatrix(), name = "<unnamed>"
  , resistanceMap = NULL)
```

Arguments

`states`	a list of `states` characterizing the behavior of the species, or a single `state`, for simple movements
`trans`	a square state transition matrix, defining the probability of changing between states. For convenience, use the function `transitionMatrix`. Can be omitted if this species has a single-state movement.
`name`	the name of the species
`resistanceMap`	not used. Will be implemented in future versions.

Details

The rows and columns of the transition matrix correspond in the same order to the list of states. The matrix is not symmetric, and is read along the rows, i.e. the probability of changing from state 2 to state 1 is located in row 2, column 1; hence rows must sum to 1 but columns not.

Value

An object of class `species`.

See Also

`simulate`, `perceptualRange`, `state`, `transitionMatrix`, `Arith-methods`.

Examples

```
## example from 'simulate'

## note: perceptual range radii and step lengths must be
## adequate to the raster resolution!

LevyWalker <- species(
(state.RW() * 100 + 10) + (state.CRW(0.97) * 500 + 20)
, transitionMatrix(0.005, 0.001))
```

---

speciesModel *Defines a species model to adjust to a real trajectory*

Description

The sole purpose of this function is to be used in conjunction with `adjustModel`. It is used to tell the optimization algorithm which parameters are to be approximated, and which are constant.

Usage

```
speciesModel(type, perceptual.range = 0, steplength = 1
  , prob.upperbound = 0.5, max.concentration = 0.99)
```

Arguments

`type`	the type of movement to "fit". One of `CRW`, `RW.CRW`, `CRW.CRW`, `RW.CRW.sl`, `CRW.CRW.sl`
`perceptual.range`	the perceptual range for all states.
`steplength`	the fixed step length for fixed step length types `CRW`, `RW.CRW`, `CRW.CRW` or the maximum allowed value for variable step length types `RW.CRW.sl` , `CRW.CRW.sl`, `CRW.CRW.CRW.sl` and `CRW.RW.Rest.sl`.

`prob.upperbound`
: the maximum allowed value for the state switching probabilities. The default is 0.5 because very high state switching probabilities don't make much sense from a biological point of view.

`max.concentration`
: the maximum allowed value for the turning angle concentration parameter for CRWs [0, 1]. By default it is set to 0.99 because values higher than this (for technical reasons) result in straight line paths, which is a technical artifact.

Details

This function defines the type of movement to be adjusted with `adjustModel`. Before choosing the type, it is good practice to plot the real trajectory and visually assess which would be the most adequate model to try. Currently included movement types are:

- `CRW`: single state CRW, fixed step length (1 parameter)
- `RW.CRW`: two state RW/CRW, fixed step length (3 parameters)
- `CRW.CRW`: two state CRW/CRW, fixed step length, (4 parameters)
- `RW.CRW.sl`: two state RW/CRW, variable step length, (5 parameters)
- `CRW.CRW.sl`: two state CRW/CRW, variable step length (6 parameters)
- `CRW.CRW.CRW.sl`: three state CRW/CRW/CRW, variable step length (12 parameters)
- `CRW.RW.Rest.sl`: three state CRW/RW/Rest, variable step length (7 parameters)

However, the user can easily write any custom function for addressing other movement types, see the code for details.

Value

Returns a function that creates a species from a vector of parameter values. This function is normally used to create species from the `adjustModel` results, see examples there.

See Also

`adjustModel`.

Examples

```
library(SiMRiv)

model <- speciesModel("RW.CRW.sl")

# this shows the parameters that will be approximated
model

# this creates a species with 2 states
# RW and a CRW with correlation 0.9
# with the switching probabilities RW->CRW = 0.01, CRW->RW = 0.05
# and the step lengths RW = 15, CRW = 50.

species <- model(c(0.9, 0.01, 0.05, 15, 50))
```

`state` *Define a movement state*

Description

Defines a behavioral state to be used when creating `species`.

Usage

```
state(concentration, pwind = perceptualRange("circular", 0)
  , steplen = 1, name = "")
state.Resting()    # still state
state.RW()         # uniform random walk (brownian motion),
                   # independent of resistance
state.CRW(concentration)  # correlated random walk,
                        # independent of resistance
```

Arguments

`concentration`	turning angle concentration, a value between 0 (uniform distribution resulting in random walk) and 1 (only one value possible resulting in a straight line path)
`pwind`	a `perceptualRange` definition
`steplen`	the base (maximum) step length of this state in map units. Note that the actual step length depends on the resistance in each step.
`name`	the name of the state

Details

See `Arith-methods` for more convenient ways of setting parameters, instead of using `state`.

Value

An object of class `state`.

Note

The perceptual range radius and step length must be adequate to the resolution of the resistance raster (if provided in simulations). If no raster will be provided, then the perceptual range is irrelevant, and the step length has solely a relative meaning (in relation to other states or other species).

For a review of different random walks, see Codling et al. (2008)

References

- Codling, E. A., Plank, M. J., & Benhamou, S. 2008. Random walk models in biology. Journal of the Royal Society Interface, 5(25), 813-834.

See Also

species, perceptualRange, Arith-methods.

Examples

```
## a correlated random walk influenced by landscape

state(0.97, perceptualRange("cir", 500), 10, "CorrelatedRW")

## the same, but using the shortcut form

state.CRW(0.97) * 500 + 10
```

---

transitionMatrix *Define a state transition matrix*

Description

Defines a state transition matrix to be used when creating species.

Usage

```
transitionMatrix(...)
```

Arguments

...	the probabilities that will form the matrix, see Details. If none given, returns a 1-element matrix (for one state only)

Details

The transition matrix (Markov matrix) is a square, non-symmetric matrix with all elements between 0 and 1, and whose rows must sum to 1 (but not columns). It defines the probability of the individual changing from each behavioral state to another, and this is tested in each time step of the simulation, hence probabilities should be small.

This function is just a helper to create such matrix. The arguments are probabilities given in the following order (example for 3 states):

Probability of changing from:

state 1 –> state 2

state 1 –> state 3

state 2 –> state 1

state 2 –> state 3

state 3 –> state 1

state 3 –> state 2

The diagonal (probablity of remaining in the same state) is computed so that rows sum to 1.

Value

A numeric matrix.

See Also

species.

Examples

```
## a 3-state transition matrix

transitionMatrix(0.01,0.02,0,0.03,0.0001,0)
```

Index