Package ‘facilitation’

February 9, 2018

Title A C++ Framework for Plant-Plant Interaction IBMs

Version 0.5.2

Description A tool for simulating a variety of spatial individual-based models of plant-plant interactions. User-created models can include any number of species, each of which can be structured in any number of life-stages, where each life-stage has specific death, growth and reproduction rates, as well as specific interaction radius, dispersal radius, and interaction effects over each other species/life-stage. Life stages were modeled so as to be a stochastic, individual-based version of differential Matrix Population Models (Caswell 2001, ISBN:0-87893-096-5). Interactions can be positive (facilitation) or negative (competition) and can affect death rates, growth rates or reproduction rates. Interactions from multiple numbers are additive, so as to best approximate classic population dynamics models such as the logistic model and Lotka-Volterra model (Britton 2004, ISBN:9781852335366). All models work in continuous time, implemented as an optimized version of the Gillespie algorithm (Gillespie 1976 <doi:10.1016/0021-9991(76)90041-3>) for independent exponential times, and continuous space.

Imports Rcpp (>= 0.12.14), Matrix, grid, animation

LinkingTo Rcpp

Suggests knitr, rmarkdown

Depends R (>= 3.1.0)

SystemRequirements C++11

License GPL-2

LazyData true

RoxygenNote 6.0.1

URL https://github.com/Lobz/facilitation

VignetteBuilder knitr

NeedsCompilation yes

Author Mali Salles [aut, cre],
Andre Chalom [aut],
Alexandre Adalardo [aut],
Camila Castanho [ctb]

Maintainer Mali Salles <marinacs@gmail.com>

Repository CRAN

Date/Publication 2018-02-09 18:25:21 UTC

R topics documented:

`abundance.matrix` *abundance matrix*

Description

Returns a matrix with abundances of each life stage/species over time

Usage

```
abundance.matrix(data, times = seq(0, data$maxtime, length.out = 50),
  by.age = FALSE, cap.living = FALSE)
```

Arguments

`data`	result of a simulation, created by `community`
`times`	array of times at which the abundances will be calculated
`by.age`	T/F. Use this option to get the number of individuals to reach each age, instead of abundances for each time.
`cap.living`	Logical. Use this option with by.age=T, to set the time of death of living individuals to max simulation time. Otherwise, living individuals are excluded from the data. Either way, this data will be more representative if only a small fraction of total individuals is living at the end of simulation.

Details

The rows in the matrix are the lifestages/species id. The times are in the row names. To visualize the abundance matrix data we recomment the function `stackplot`.

Examples

```
data(malthusian)
times <- seq(0,malthusian$maxtime,by=0.1)
ab <- abundance.matrix(malthusian,times)

ab.by.age <- abundance.matrix(malthusian,times,by.age=TRUE)
```

`community` *community*

Description

Runs a simulation with any number of structured populations, for a limited time.

Usage

```
community(maxtime, numstages, parameters, init, interactionsD, interactionsG,
  interactionsR, height = 100, width = 100, boundary = c("reflexive",
  "absortive", "periodic"), dispKernel = c("exponential", "random"),
  starttime = 0, maxpop = 30000)
```

Arguments

`maxtime`	How long the simulation must run
`numstages`	Array of number of stages for each population
`parameters`	Data.frame or matrix with one row for each stage. Columns: D,G,R,dispersal distance,radius(optional),maxstressefect (optional)
`init`	Either an array of initial numbers for each stage of each population, or a data.frame with the history of a simulation
`interactionsD`	Optional. A square matrix of effects of life stages over each other, where element [i,j] is the effect of stage i over stage j. Positive values equal facilitation, negative ones, competition. The interactions occur only if the affected individual is within the affecting individual's radius, and are additive. Affects death rates (is subtracted from D).
`interactionsG`	Same as above, but affecting growth rates (is added to G).
`interactionsR`	Same as above, but affecting reproduction rates (is added to R) .
`height`	Arena height
`width`	Arena width

`boundary`	Type of boundary condition. Options are "reflexive", "absortive" and "periodic". Default is reflexive.
`dispKernel`	Type of dispersion kernel. Options are "exponential" and "random", in which seeds are dispersed randomly regardless of parent position (note: "random" option ignores dispersal parameter)
`starttime`	use for proceeding simulations. Time when simulation begins.
`maxpop`	If the simulation reaches this many individuals total, it will stop. Default is 30000.

Examples

```
param <- data.frame(D=c(2,1,2,1),G=c(2,0,2,0),R=c(0,3,0,3),dispersal=c(0,2,0,20))
malth <- community(2,c(2,2),param,init=c(10,10,10,10))
ab <- abundance.matrix(malth)
stackplot(ab[,1:2]) # species 1
stackplot(ab[,3:4]) # species 2
```

`create.parameters` *Create Parameters*

Description

Structures the parameters into the correct format for use in community

Usage

```
create.parameters(Ds, Gs, Rs, dispersal, radius, stress, n)
```

Arguments

`Ds`	An array of death rates for the structured population, of length `n`
`Gs`	An array of growth rates for the structured population, of length `n-1`
`Rs`	Either the seed production rate of adults in the population, or an array of seed production rates, of length `n`.
`dispersal`	Dispersal distances
`radius`	Optional (use if there are any interactions). Either one radius of interactions or an array of interaction radiuses, of length `n`.
`stress`	Optional (use to create a stress gradient). An array of values of stress gradient slope. The full value will be added to death rate at the right of the plot, half value at the middle of the plot, and so on, proportionally.
`n`	Number of stages in the population

Examples

```
# create a sample parameters
create.parameters(n=3)

# structure parameters from arrays
create.parameters(Ds=c(10,5,2),Gs=c(2,2),Rs=20,radius=2)
```

`limiting.rate` *Limiting Rate*

Description

This function returns the real dominant eigenvalue of a Matrix Population Model matrix. That is a real number that corresponds to the per-capita growth rate that a population approaches as time passes, in a model with no interactions.

Usage

```
limiting.rate(mat)
```

Arguments

`mat`	a square matrix

Details

A structured population can grow at exactly this rate if the distribution between stages corresponds exactly to the distribution of the dominant eigenvector. The models that can be simulated by this package are of a class that always has a real dominant eigenvector. Note that these are continuous-time models, in which $r > 0$ means the population will grow, and $r < 0$ means it will decrease. This function doesn't throw errors, instead it returns 'NA'.

Examples

```
mat <- mat.model.base(5)
limiting.rate(mat)
```

`longevity` *longevity*

Description

Calculates the lifespan of each individual. Returns a data.frame with the individual's id, the last stage reached by that individual, the time of birth, time of death (if dead), and longevity (if dead).

Usage

```
longevity(data)
```

Arguments

`data`	result of a simulation, created by `community`

Examples

```
data(malthusian)
longevity(malthusian)
```

`malthusian` *A single species, 3 stages, no interactions simulation result*

Description

A single species, 3 stages, no interactions simulation result

Usage

```
malthusian
```

Format

A list of data and parameters, generated by community

Examples

```
## Simple mathusian one species
init <- c(0,0,100)
################# D G R disp
param <- matrix(c(5,1,0,5, 1,1,0,5, .5,0,10,5),nrow=3,byrow=TRUE)
malthusian <- community(10,3,param,init)
```

mat.model *matrix population model*

Description

Produces the Matrix Population Model matrix for a continuous time structured population model, to be applied in a linear ODE. If there is more than one population, returns a list of matrices, or one block-diagonal matrix created by the combination.

Usage

```
mat.model(data, ns, combine.matrices = FALSE)
```

Arguments

`data`	Either the result of a simulation, to extract the parameters from, or a data.frame containing the parameters.
`ns`	an array of numbers of stages. Use when `data` is a data.frame and the is more than one population.
`combine.matrices`	Logical. Combine the matrices into a single, multi-population matrix?

Examples

```
# example 1
mat.model(create.parameters(n=4))

# example 2
data(malthusian)
mat.model(malthusian)

# example 3
data(twospecies)
mat.model(twospecies,combine.matrices=TRUE)
```

mat.model.base *matrix population model*

Description

Produces the Matrix Population Model matrix for a continuous time structured population model, to be applied in a linear ODE. Unlike `mat.model.base`, only works with a single population. If only the number of stages is provided, returns a ramdom population matrix.

Usage

```
mat.model.base(n = 3, Ds = runif(n, 0, 5), Gs = runif(n - 1, 0, 5),
  Rs = runif(n, 0, 5))
```

Arguments

`n`	The number of life stages. Default is 3.
`Ds`	An n-array with death rates for each stage.
`Gs`	An (n-1)-array with growth rates for each stage but the last.
`Rs`	Either a single reproduction rate for the oldest stage, or an n-array of reproduction rates for each stage.

Examples

```
mat <- mat.model.base(5)
mat2 <- mat.model.base(3,c(1,2,3),c(10,10),100)
```

`proceed` *proceed*

Description

Proceed with a stopped simulation.

Usage

```
proceed(data, time)
```

Arguments

`data`	result of a simulation, created by `community`
`time`	a number: for how long to extend the simulation

`restart` *restart*

Description

Turn back time and restart a simulation from time t

Usage

```
restart(data, time, start = 0)
```

Arguments

`data`	result of a simulation, created by `community`
`time`	a number: for how long to extend the simulation
`start`	a number: an instant in time to begin from

`solution.matrix` *solution.matrix*

Description

The `solution.matrix` function returns the solution to a linear ODE of the form P' = MP, which is merely P(t) = exp(Mt)p0 where p0 is the initial condition

Usage

```
solution.matrix(p0, M, times = c(1:10))
```

Arguments

`p0`	initial condition, as an array
`M`	a square matrix with as many rows as P0
`times`	an array containing the times in which to calculate the solution

Examples

```
mat <- mat.model.base(5)
solution.matrix(c(1,0,0,0,0),mat)
```

`spatialanimation` *Function for ploting simulation as a gif*

Description

The spatialanimation function plots the individuals of the selected stages over time. Use plotsnapshot for plotting a single instant.

Usage

```
spatialanimation(data, times = seq(0, data$maxtime, length.out = 50),
  interval = 0.1, draw = data$num.total:1,
  radius = data$param$radius[draw], color = colorRampPalette(c("darkred",
  "lightgreen"))(length(draw)), movie.name = "facilitationmovie.gif",
  xlim = c(0, data$w), ylim = c(0, data$h))

plotsnapshot(data, t, ...)
```

Arguments

`data`	result of a simulation, created by `community`
`times`	array of times at which to plot
`interval`	a time length to wait between frames
`draw`	an array of stages id, to be drawn bottom to top. Absent stages will not be drawn.
`radius`	Optional. Array representing the sizes in which the individuals will be drawn. Defaults to interaction radius.
`color`	Optional. A color vector
`movie.name`	The filename of the gif that will be saved.
`xlim`	Optional. Limits to the x-axis
`ylim`	Optional. Limits to the y-axis
`t`	a single time at which to plot
`...`	additional parameters to be passed to spatialanimation

Author(s)

Alexandre Adalardo de Oliveira - 16/03/2016

M. Salles

Examples

```
data(twospecies)
spatialanimation(twospecies,draw=c(5,3),times=seq(0,10,1),movie.name="ts.gif")
data(twospecies)
plotsnapshot(twospecies,t=10)
```

`stackplot` *Plots*

Description

Plotting functions.

Usage

```
stackplot(mat, col, legend, log.y = FALSE, perc = F, qt = 100, ...)
```

Arguments

`mat`	A population matrix, as produced by `abundance.matrix` or something that can be coerced to matrix
`col`	Optional. A color vector
`legend`	Optional. An array of names
`log.y`	Logical. Should the y-axis be plotted in a logarithmic scale?
`perc`	Logical. If set to true, will output the y-axis as a percentage instead of the absolute numbers
`qt`	Optional. For distributions, show only up to quantile qt (percentage)
`...`	Further parameters to be passed to the lower level plot function

Details

The `stackplot` function produces a stacked plot of the population over time. Notice that the population should have at least two stages for this function to work.

Examples

```
data(twospecies)
ab <- abundance.matrix(twospecies,seq(0,twospecies$maxtime,by=1))
# species 1
stackplot(ab[,1:3])
# species 2
stackplot(ab[,4:5])
```

twospecies — *A simulation result with two very different species*

Description

A simulation result with two very different species

Usage

```
twospecies
```

Format

A list of data and parameters, generated by community

Examples

```
### Two species competition with growth limitation
nstages <- c(3,2)
init <- list(c(100,0,10),c(100,30))
param1 <- create.parameters(D=c(1,2,.5),G=c(1,1),R=10,dispersal=30,radius=c(0,.5,3))
param2 <- create.parameters(D=c(1,.5),G=c(1),R=4,dispersal=5,radius=c(1,5))
param  <- rbind(param1,param2)
interD <- matrix(c(0,0,0,0,0, 0,-1,0,0,0, 0,0,-2,0,-1, 0,0,-1,0,0, 0,0,-1,0,-2),ncol=5)
interG <- matrix(c(0,0,0,0,-.5, 0,0,0,0,-.5, 0,0,0,0,0, 0,0,0,0,-.8, 0,0,0,0,0),ncol=5)
twospecies <- community(10,nstages,param,init,interactionsD=interD,interactionsG=interG)
```

Index